Study on the Highly Reliable and Secure Data Management System under Weak ICT Environment by Blockchain Technology

by

Name: Ragouguelaba Agoda-Koussema
Student ID: 1416192101

Doctoral Dissertation

Submitted to
Graduate School of Science and Engineering,
Doshisha University

in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Supervisor: Prof. Hirohide Haga

ABSTRACT

This study investigates the design and implementation of data management system with high level reliability and security by blockchain technology. The data access environment provided by blockchain is highly secure and trustworthy. In blockchain system, some data fragments are grouped into one piece called as blocks, and all blocks are connected to create a chain of blocks in database. When blocks are connected, hash value is used to connect blocks properly.

Blockchain technology enables highly secure and reliable data management system under relatively poor ICT environment. For example, developing countries such as African countries do not have sufficient ICT environment. Therefore adopting blockchain technology is suitable for such countries. Based on this consideration, we started to build data management system on the blockchain system. Our primary target is the management of residents' data. Managing residents' data is one of the most essential activities of any governments such as local government and nation-wide government. But under poor ICT environment, enough support of computerization is difficult, and therefore residents cannot receive the benefit of computerization. Therefore, we set our primary goal to the citizens' data management system.

In our prototype, each event of resident such as birth, moving, employment and so on, is assigned to data fragment and certain amount of data fragment, says 20 fragments are packed into block. And each block will be connected to existing blockchain or will create new blockchain under the specific algorithm. In order the implement blockchain system, we have two methods. One is using certain framework which supports the construction of blockchain system. The other method is to implement blockchain system from scratch; without using any framework. We have developed with both methods.

In order to develop blockchain system with frameworks, we have to select appropriate framework. There are some frameworks which support the development of blockchain system. Among them, we selected the MultiChain as a blockchain platform firstly. However, as MultiChain platform is mainly for private blockchain system, it is not suitable for any government level data management system. Therefore, we tried to use another blockchain framework. We selected Hyperledger Fabric which was developed by Linux Foundation. It enables to implement all styles of blockchain system. In order to compare above-mentioned two methods, we developed a blockchain system from scratch.

Our conclusion is that, the implementation with no framework is more flexible than using any framework, and therefore and suitable for specific purpose of blockchain system. This study describes the design and implementation of data management by using Multichain, Hyperledger Fabric and no framework. Furthermore, to provide the best user experience, we also built the web application interface with Java web application framework named PrimeFace. The web application interface will permit to avoid installing any applications in users' PC or smartphone. The implementation of a prototype revealed that no framework blockchain technology is more suitable than MultiChain and Hyperledger Fabric.

Contents

Chapter 1

Introduction

1.1 Primary goal of this study

The primary goal of this study is to investigate, design, and implement a highly secure and reliable data management system in an ICT(Information and Communication Technology) environment with limited resources. To achieve this final goal, we have decided to use the blockchain technology and web application system. Firstly, used as decentralized and trust-less ledger for digital currencies, blockchain is adopted in many fields [1][2][3]. Many sectors, for data management, have focusing on the using the conceivable outcomes given by blockchain technology to realize decentralized, trust-less applications that do not depend on a single trusted party. Some governments are exploring the properties and benefits, as well as the deficiencies of this innovation for their particular utilize cases. Blockchain is not a specific technology for cryptocurrency but general technology for high secure and reliable data management system. We paid attention to this feature of blockchain technology to implement our final goal. Blockchain is often seen as the basis of cryptocurrency. But our is that blockchain is an implementation method of highly secure and reliable database. Blockchain is a general-purpose technology. The usage of blockchain is not limited to cryptocurrency. It has more wide application fields. This is our fundamental viewpoint.

Managing trusted data about people, organizations, properties, and activities is an essential government task. Local, state, and national authorities are responsible for keeping track of details such as birth and death, marital status of citizens, business licenses, properties transactions, and illegal activities. Even for advanced governments, managing and utilizing these data can be difficult. Some documents are only available on paper, and residents must often visit appropriate official sections to update his/her data. How do we keep the data especially the most valuable assets (contracts, properties titles, account statements, and so on) from being altered or even removed by people who gain to our systems legally or illegally?

One of the most important statistics in any country are, in particular, live event records for citizens. Many organizations in developing nations, such as governments, face the challenge of data management. In order to implement useful and efficient data management system, using advanced ICT technology is essential. But developing nations usually do not have sufficient ICT environment. In other words, they don't have adequate communication facilities like high-speed data communication lines. With inadequate communications

infrastructure, the data exchange system cannot operate effectively. There are sophisticated technologies for implementing a highly secure and dependable data management system. But the implementation of such an advanced data management system requires a new approach for implementation under a weak ICT environment.

1.2 Method to archive the primary goal

In order to overcome these difficulties, we begin our research about highly secure and reliable data management system under poor ICT environment. The technologies adopted are web-based application and blockchain interfaces. We evaluate the need for blockchain and the applicability of different types of consensus algorithms and permissions. A certain number of existing implementation frameworks, such as Ethereum [4] were initially taken into consideration. However, these frameworks seemed to be over-specification. There are many unused functions in existing frameworks. This is because many frameworks aim to implement various kind of applications and therefore many functions are embedded in the framework. Therefore, a blockchain system was implemented from scratch firstly. Then we implemented two system using blockchain frameworks. We have selected MultiChain and Hyperledger Fabric, two permissioned blockchains to implement the nation-level data management system. Furthermore, this solution does not involve only blockchain technology, but also requires user interaction with the web application developed with Java programming language and the processing of privacy-sensitive data. People can access website by using wireless communication infrastructure such as a mobile phone network.

The blockchain technology is ingenious invention without any doubt. It is an authentic database ledger which records the history of the activity along the network among the connected devices or users. Through self-administering and self-executing scripts, information is updated and recorded with established permissions on. Distributed virtual machine.

Under-developing countries such as some African countries do not have enough communication infrastructure such as high-speed data communication lines. Poor communication infrastructure disturbs high-level data exchange system. Furthermore, data management systems must be highly secure, reliable and easy to use. There are already some technologies to implement such highly secure and reliable system. However, many of these technologies require rich IT resources and infrastructure to implement them. Some novel technologies to implement effective data management system under poor communication infrastructure requires new idea for realization.

To meet such requirements, the use of blockchain [5] and web application [6] can be one possible solution. Behind blockchain there are several technologies such as cryptography technique and hashing which are used to protect the information. They enable us to implement our final research goal by combining other technology. Blockchain is one of the most promising technologies for our purpose. Web application technology is also essential for under-developing countries. Such countries usually do not have sufficient wired communication networks such as telephone line or high-speed data communication line. This is partly because such infrastructures require huge amount of budget. However

recent advancement of wireless communication network dramatically improves the communication infrastructure of such under-developing countries. By using such wireless communication infrastructure such as mobile phone network, people can access websites and get information.

Based on these considerations, we started implementing the highly secure and reliable data management system by combining the blockchain and web application technologies. Many people still misunderstand what kind of technology the blockchain is and how does it work. It enables to implement the secure and reliable control and protection of data on the network systems. If there is intrusions to data, the hacker can access all data and do whatever he wants with it. It is important to have and save secure and reliable data. By coupling these two technologies: blockchain technology and java web framework, we can get a reliable and secure data for a resident data management system. Our system will help to avoid the loss of traceability and falsification of any data. This technology can record the history of all the transaction at any moment by using the chains of data.

1.3 Structure of this thesis

This thesis consists as follow:

- Chapter 2 describes the related works and blockchain, the essential technology of this study,
- Chapter 3 describes the structure of the system design,
- Chapter 4 describes the Implementation detail without using any framework of blockchain,
- Chapter 5 is about the implementation detail by using Multichain framework,
- Chapter 6 describes the implementation detail by using Hyperledger Fabric framework,
- Chapter 7 is about Web service implementation with PrimeFaces framework,
- Chapter 8 describes the comparison of the three implementations,
- Chapter 9 is a conclusion and future works of this study.

Chapter 2

Related work and essential technology

This chapter describes the related works and essential technology of this study.

2.1 Related work about blockchain technology

At its most basic, blockchain technology is a data store distributed across a network among participants who can reach consensus on the validity of transactions without the need of a central authority to mediate or authenticate. The technology's initial applications were focused on cryptocurrencies in public permissionless networks. It was primarily used in financial applications, the technology can be used to track changes to any physical or digital asset, including data that is valuable to individuals or organizations.

As we look at blockchain technology more closely, we will see four main features: immutability, cryptographic digests, cryptographic signatures, and distributed networks. Each component protects against a specific aspect of unauthorized data changes made with valid user credentials or by hackers. Integrating these blockchain innovations into database allows mainstream systems to profit from the essential security advantages of blockchain with minimal or no adjustments.

Digital ledger has been used to protect trusted records by financial institutions. It has been used too in the healthcare system to share and manage data. Healthcare data are highly sensitive, there is a need to protect it from an illegal access.

Estonia, for example, implementing a technology called Keyless Signature Infrastructure (KSI) to protect all government data. KSI generates hash values, which are numeric values that uniquely reflect large quantities of data. The hash values are spread through a private network of government computers and stored in a blockchain [7].

The authors of the work [1] proposed blockchain technology to be an appropriate infrastructure for healthcare data sharing. They have also introduced a new consensus of algorithm, called "Proof of interoperability" which was based on conformance to the Fast Healthcare Interoperability Resources (FHIR) protocol.

In the work [8] the authors implement a decentralized record management system to handle electronic health records using blockchain technology. Some authors [9] recommended the use of smart contracts to handle clinical trial authorization information on a permissioned Ethereum blockchain and a private

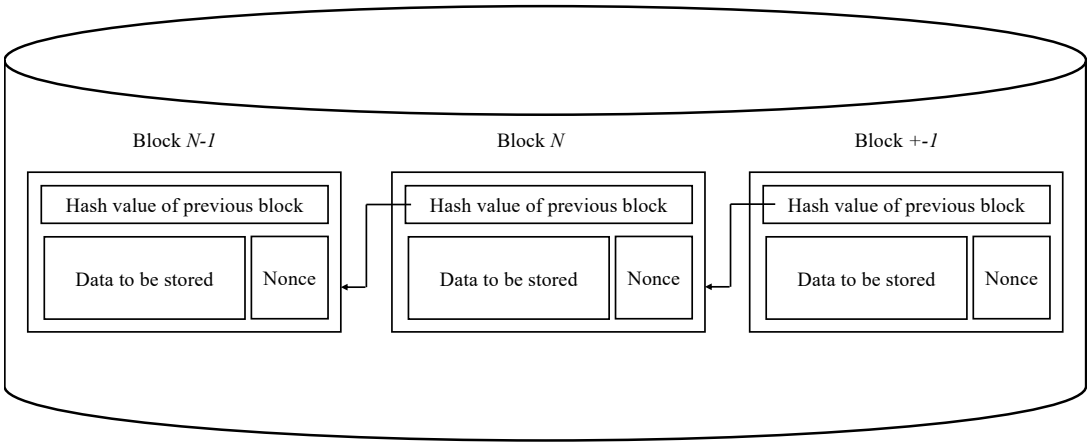

Figure 2.1: Conceptual illustration of blockchain database

IPFS network to store the data structure.

2.2 Definition of blockchain technology

Blockchain is a decentralized or distributed ledger that keeps track of all network transactions. With simple secure private keys such as hash mechanism, each node stores all information on the network. Each block includes information such as the previous block's hash value, data to be stored in the current block, and a nonce value. To create each block, the user must choose an acceptable nonce value for each block and changing the nonce requires a significant amount of computing power. This is the primary explanation for blockchain's ability to provide a highly stable and dependable data management system. Therefore, virtually falsification is impossible. This is the main reason why blockchain provides highly secure and reliable data management system. And blockchain has an attractive function which are very different from the traditional databases. Every node in the network store all blockchain. Therefore, even some accidents such as power failure or cracker's attack destroy the database of one node in the network, all data will be recovered by using the data stored in the other nodes. This is why blockchain has high robust nature. Figure 2.1 is a conceptual illustration of blockchain database.

2.3 How blockchain works

When a user wants to add data in the blockchain, user firstly packs several data fragments into one data package. For example, in case of Bitcoin, each transaction such as transferring money from one user to another is a data fragment. Certain amount of basic data fragments, say 20 fragments, will be packed into one package. After the finish of constructing one package, block construction will start. Each block contains several data such as data package, nonce and hash value of previous block. Each block must be connected to existing blockchain. To connect new block to exiting blockchain, hash value of each block must satisfy the specific condition. For example, if hash value is represented by 256 bits (32 bytes), hash value of first 64 bits (8 bytes) must be 0. To satisfy the condition,

user must compute specific nonce value. To get such hash value, users will set appropriate nonce value. However, as hash function generates virtually random value, user must repeat computing hash value by assigning specific value to nonce for huge amount of times. It requires a millions of millions computing time and power. Once a specific value of nonce which satisfies the condition of hash value is found, newly created block will be connected to existing blockchain. The reason why blockchain has highly secure and reliability is the amount of computing time. As mentioned above, to connect one block to blockchain, user must compute appropriate nonce value. For example, let n be the length of blockchain (the number of blocks in the blockchain), and the attacker modified the data in k-th block. When data in k-th block is falsified by someone, hash value of k-th block will change drastically. Therefore, the connection of blocks will be broken. If malicious attacker tries to connect modified block to blockchain, he/she has to find new nonce value which satisfies the condition of hash value. The change of hash value of k-th block affects following all $(n-k+1)$ blocks; attacker must compute $(n-k+1)$ nonce value again to connect all blocks in the blockchain and this re-computation require quite huge amount of computation. Therefore, falsification of blockchain is virtually impossible. This is why the blockchain system provides highly secure and reliable data.

2.4 Types of blockchain

There are essentially three different kinds of blockchain. They are, public blockchains, private blockchains, and consortium blockchains are the three major types of blockchains. Followings are the summary of these kinds of blockchains.

- **Public Blockchain:** This form of blockchain is open to the general public, and anyone can join the network as a node. Users may or may not be compensated for taking part. They are not held by anyone and are available to the public for participation. Bitcoin is the best example of a distributed blockchain. Any time a user makes a transaction, it is mirrored in the block's copy for all. A decentralized blockchain is used by Bitcoin.

- **Private Blockchain:** As the name suggests, a private blockchain is one that is only accessible by a consortium or group of individuals or organizations that have agreed to share the ledger. Only the owner has the authority to make improvements to it. Federated blockchains are more scalable (high scalability) and provide greater transaction privacy.

- **Consortium Blockchain:** A consortium blockchain is a cross between public and private blockchains. There are federated blockchains, which are managed by a community of people. There are two or more administrative nodes in a group. In contrast to public blockchains, no one can participate in the transaction verification process without the permission of administrative nodes. This type of blockchain is commonly used in the banking industry, for example.

2.5 Available frameworks to implement blockchain system

A variety of available implementations were considered for implementing private and permissioned blockchain. The sections that follow describe the relevant implementations and our reason for not selecting them.

- **Bitcoin** [1] and its client Bitcoin Core5 are intended for financial transactions. Because of its limited space, low throughput and high delay, the Bitcoin network is unsuitable for storing any significant amount of meta data [10]. The consensus algorithm of Bitcoin is only concerned with double-spending of bitcoins, not with data. Although it is technically possible to modify this cryptographic algorithm, the design of bitcoin does not lend itself well to our use case. Since Bitcoin and its most popular implementation are infeasible, we ruled out any attempts at modifying the protocol. In most cases, modifying the existing source code would take more time and resources than building a blockchain from scratch.

- **MultiChain** [11] is a blockchain implementation that is both private and permissioned. As with Bitcoin, it is primarily used for financial transactions, which means that the consensus algorithm would need to be significantly modified. While this is likely easier than it is with Bitcoin, the platforms currently supported by MultiChain are a significant limitation for our use case. MultiChain is currently only supported on 64-bit systems and requires several additional software dependencies. As a result, we cannot use MultiChain as the foundation for our implementation.

- **Ethereum** [4] enables the creation of smart contracts in a variety of programming languages, including Solidity6. These smart contracts are capable of representing transactions that are practically arbitrarily complex. These are not limited to financial transactions, as is the case with Bitcoin, but also allow for the representation of states and state changes, as our use case requires. Despite Ethereum's flexibility, previous research has discovered that even simple smart contracts are prohibitively expensive [12]. Additionally, costs are unpredictable due to frequent changes in the cost structure of executed operations and fluctuations in the exchange rate to Euros7 [13][14]. While the cost issue could be resolved by utilizing a private blockchain, Ethereum's complexity far exceeds the requirements for our use case. When a value needs to be stored in a smart contract, it is updated using a modified Merkle Patricia Trie, which is relatively time consuming, as demonstrated in [15]. Additionally, it has been demonstrated that this results in a slow execution speed as the volume of data increases, which is detrimental to our use case, particularly from the perspective of the users.

- **OpenChain** [16] is a private blockchain that is optimized for energy efficiency, network communication, and block rate. As a result, it is based on a client-server model rather than a peer-to-peer network, and its consensus algorithm is proof of authority rather than proof of work. Because the goal of this work is to create a decentralized and trustless model, a

centralized approach such as the one used by OpenChain's proof of authority consensus algorithm is not applicable. Nominating a designated authority responsible for transaction validation would violate the desired security and trust properties. The same is true for all blockchains that achieve consensus via Proof of Authority (PoA), such as Corda [17].

- **HAWK** [18] is a blockchain that prioritizes privacy. Additionally, several blockchain implementations, such as Tendermint [19], Stellar [20], EOS[21], and OmniLedger [22], use BFT as a consensus algorithm. In comparison, the NEO blockchain [23], which is based on BFT, can be considered stable, but its complexity, which enables the execution of smart contracts similar to those found on Ethereum, exhibits the same issues as the latter.

Due to the unique requirements of our use case and to gain insight into the entire blockchain development and implementation process, we chose to build our own custom blockchain tailored to the specific legal requirements and requirements.

Chapter 3

System Design

3.1 Design goal of our prototype system

Our prototype system will have the following design:

- **Web application platform:** Web application platform is equiped with a database which keeps the citizen data. To share the data with the upper layers of the system, the data will be exported in an appropriate format. the upper layers of the system will be used.The blockchain will be used as service, so it is not a need to be blockchain nodes.
- **Stakeholders (City halls and others outside entities, etc⋯):** The stakeholders who want access to the citizen data stored on the Web platform for research purposes. By default they are not trusted. We need an off-chain to verify their identity before to be accepted as nodes of the blockchain network.
- **Validators:** We assume that the validators are a subset of the blockchain network nodes which assemble new blocks of valid transactions. All entities which were verified and participating as nodes of the blockchain network can be validators.
- **Blockchain model assumed:** The blockchain model assumed can be a consortium blockchain in which the Stakeholders participating as nodes of the network are assumed to be verified off-chain. The stakeholders are trusted once they are verified and allowed to be network nodes.

3.2 Overall structure

In this section we will describe the basic structure of the protype system. The prototype system uses a blockchain technology, Java web application framework PrimeFace [24] and relational database management system MySQL[25]. MySQL is used to store all data of blockchain. Figure 3.1 shows the conceptual structure of our prototype. As shown in Figure 3.1, there are several layers. Bottom-most layer is hard disk or other external storage device. This device stores all data. User will access the stored data by using database management system (DBMS). In our prototype, we use relational DBMS (RDBMS). Blockchain functions are implemented by using blockchain framework. Our prototype system uses MultiChain, which is classified as consortium blockchain.

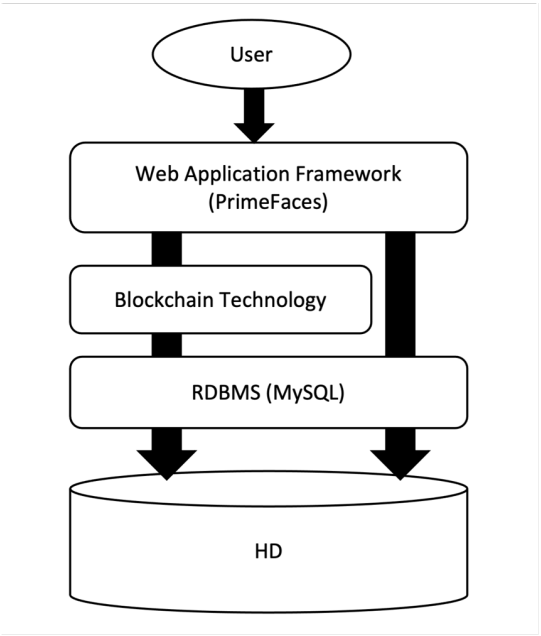

Figure 3.1: Overall structure of the prototype system

Top-most layer is a framework for web application named PrimeFaces. User interface will be implemented by PrimeFaces framework as a web application. Therefore, no specific application is installed on user's PC/smartphone.

3.3 Mapping residents record into blockchain

Primary target of our data management system is to manage citizens life event management. Birth, marriage, and death are the fundamental data. National government or local government will use these data for their activity. And for every citizen, their attributes such as income, qualification, and so on are also very important. These data must be managed with highly secure and reliable form. Falsification must be blocked. The use of blockchain technology enables these requirements.

The prototype deals with each resident record as a data fragment of blockchain. Normally one resident record is generated when a resident was born. Then several update of record will be applied to existing record because of some events such as address change, marriage and so on. And when a resident dead, the dead flag will add to resident record. This is because in blockchain no record can be removed. Instead, dead flag will be added to resident record. As it is virtually impossible to modify the data in the blockchain, the correctness of each data in the blockchain will be guaranteed.

3.4 Design of Database scheme

The following figure 3.2 represents the relation of tables of the database for our system. We have association between all the classes. Some classes have one association, others multiples associations. Each class represents a table in

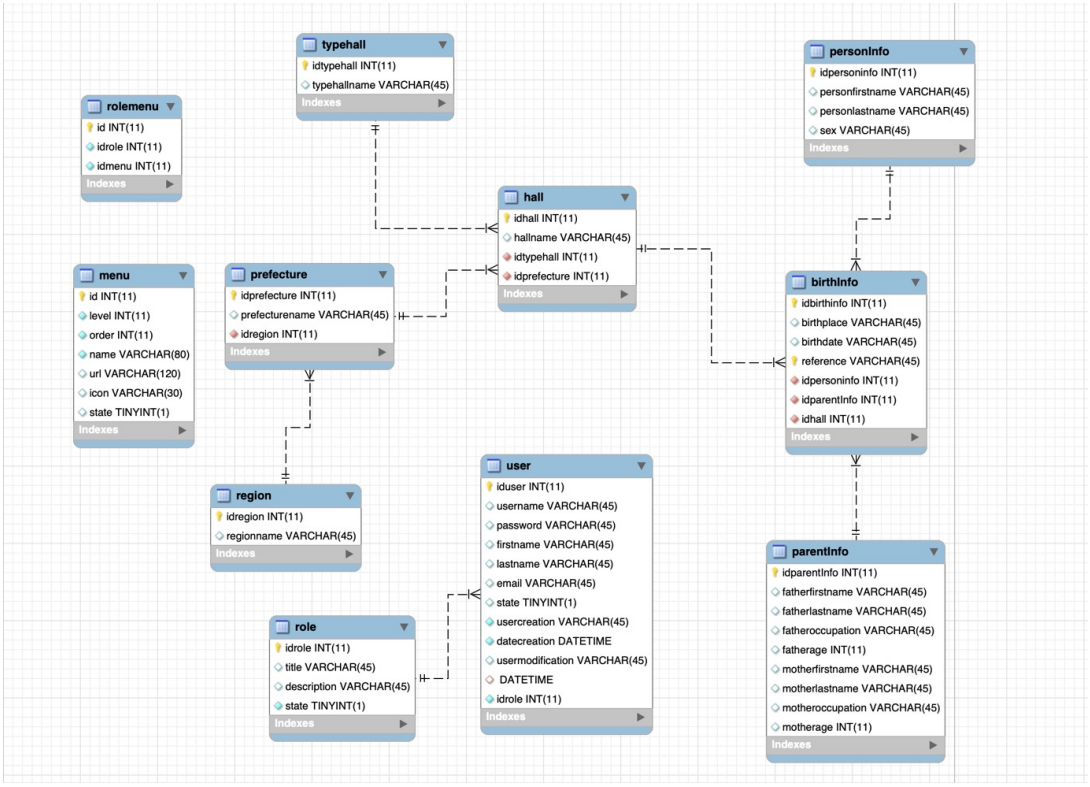

Figure 3.2: Database scheme diagram of our system

Table 3.1: Structure of table "`Region`"

`int`(11) **NOT NULL**	`varchar`(50) **NOT NULL**
`IDRegion`	`RegionName`

MySQL database. Some examples of tables are shown from Table 3.1 to 3.2. Finally, we prepared following tables.

- `Region` table: contains the name of region in the country.
- `Prefecture` table: contains the name of prefecture in the country.
- `TypeHall` table: contains the type of hall (city or town)
- `Hall` table: contains the name of hall and its location
- `BirthInfo` table: contains birth information of citizens.

Table 3.2: Structure of table "`Prefecture`"

`int`(11) **NOT NULL**	`varchar`(50) **NOT NULL**	`int`(11) **NOT NULL**
`IDPrefecture`	`NamePrefecture`	`IDRegion`

Table 3.3: Structure of table "typeHall"

int(11) **NOT NULL**	varchar(50) **NOT NULL**
IDTypeHall	TypeHallName

Table 3.4: Structure of table "Role"

int(11) **NOT NULL**	varchar(45) **NOT NULL**	varchar(11) **NOT NULL**	int(11) **NOT NULL**
IDRole	Title	Description	State

Figure 3.2 (DBMS Scheme Class diagram) contains other tables to support the implementation of systems. But essential tables are shown above.

Sample data to be stored in the tables are shown in Table 3.6 and Table 3.7.

3.5 Development of user interface

Any administrative service must be provided to all citizens equally. The service must not be influenced by the location and social status of citizens. There are still several areas whose public transportation service is not enough to support everyday life. In such an area, people feel difficulties to access public service. For example, in order to get the service, people must go to the government office if there are no ICT supports. And wired ICT infrastructure is still poor in under developing courtiers. But current advancement of wireless communication technology enables to use wireless devices such as cell phone anywhere. Comparing to wired line, wireless communication infrastructure requires less expense. Therefore nowadays, may citizens use smartphones. And people can access internet by smartphone. Therefore, web service is the most promising method to provide administrative service to all citizens.

We developed a web application using Java Primefaces framework. It is an open source framework for Java Server Faces (JSF) (mentioned in section 3.1)[26] featuring over 100 components. By using this framework, we will easily develop web application. As it is an open source framework, we can modify to fit other components such as Multichain blockchain framework, Hyperledger framework or implemented blockchain with no framework. As we have implemented the prototype system as a web application, no user needs to implement specific application in their own smartphone or PC. The detail of web application will be described in chapter 7.

Table 3.5: Structure of table "Hall"

int(11) **NOT NULL**	varchar(45) **NOT NULL**	int(11) **NOT NULL**	int(11) **NOT NULL**
IDHall	HallName	IDTypeHall	IDPrefecture

Table 3.6: Data inserted to table "Region"

IDRegion	RegionName
1	Maritime
2	Savanes
3	Savanes
4	Centrale
5	Kara

Table 3.7: Structure of table "Prefecture"

IDPrefecture	NamePrefecture	IDRegion
12	Des Lacs	1
14	Vogan	1
15	Moyen-Mono	1
16	Bas-Mono	1

Chapter 4

Implementation without using any framework of blockchain

Our work was to implement a resident records management system by encrypting with SH-256 cryptography and implementation of blockchain from scratch. In this chapter, we describe the implementation of our prototype system from scratch.

Blockchain is meant to secure storage of all data included each block. Hashes, which function as a data security mechanism, are difficult to manipulate and thus save all important information. It is decentralized since each node has full access to and modifications to the data. This includes any modifications that are made to every hash, so that the information cannot be leaked or corrupted. Every classified data in the network may thus be securely kept in blocks, accessible and recorded by each node.

4.1 Overview

Major components of prototype system are shown in Figure 3.1. To send and receive data, participants have a node installed in their premise that runs the blockchain node software and web application. Virtual private networks (VPNs) connect all nodes to the internet so that they can communicate with one another without using public IP addresses. To facilitate the clearing process, servers are installed on the premises of each utility.

4.2 Nodes

The nodes are implemented in Java 11. Figure 4.2 illustrates the fundamental architecture of our node implementation. The participants will need to provide private key and public key, as well as user-name to communicate with each other. These are required. The nodes must communicate with each other so that everyone has the same state of the blockchain. The pee-to-peer approach has established itself to ensure that this works with a number of participants. With this approach, all nodes in the network have the same status and communicate with each other without a central authority. As soon as a node receives new information, such a new transaction or a new block, it then sends the information to all other nodes. Each node is both a server and a client, used to synchronize the data of each node.

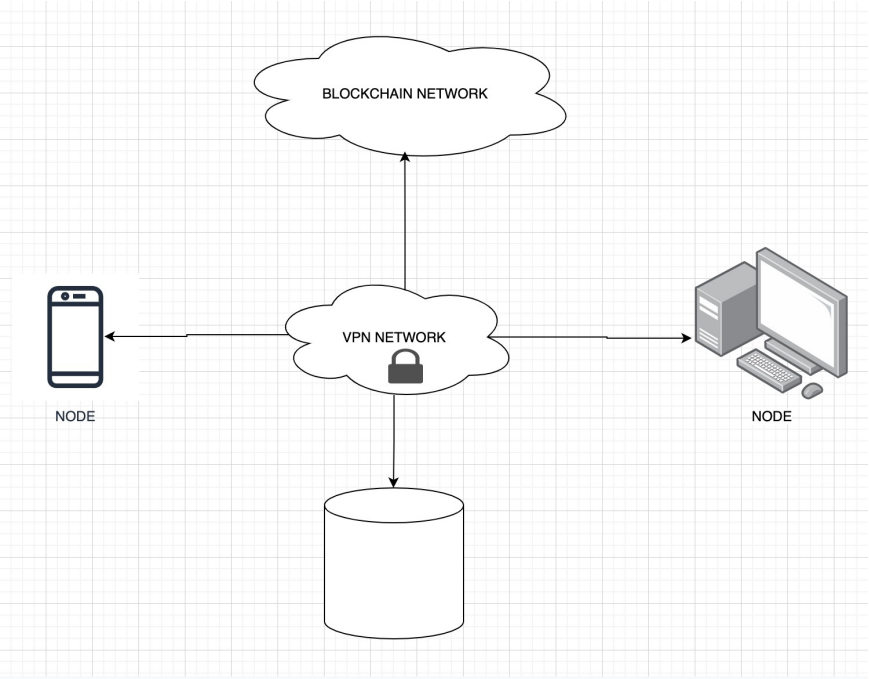

Figure 4.1: Components

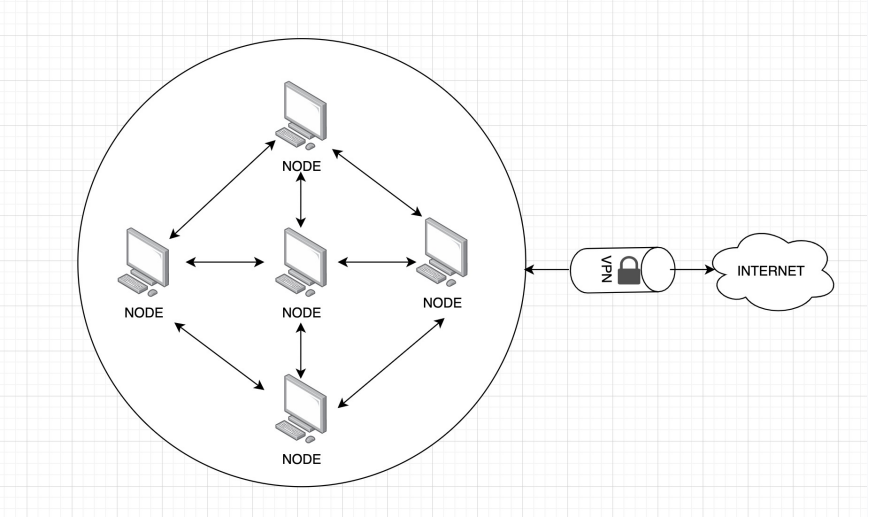

Figure 4.2: Fundamental architecture of nodes

4.3 Data management

Blockchain technology cannot store a large size of data. Therefore, we divide the data in two: essential and non-essential data. In a resident data management system, an essential data can be “name”, “reference number”, “social security number”; and a non-essential data can be “date of birth”, “place of birth”. The administrator can select some fields as essential, these fields will be packaged into a block and stored in the blockchain, while unchecked non-essential perform the SHA256 procedure and store their hash in the block with the essential fields. All of these information is stored in RDMS. The data in the blockchain makes

the essential data tamper-proof and traceable. The non-essential is stored the RDMS to reduce redundancy, and the hash result of the non-essential data is stored in the block. The SHA2546 procedure is done, and the result is compared to that previously saved in the blockchain. If they are same, then the data has not been altered [27].

It is possible to have data of any length hashed using the SHA256 algorithm [28]. This unique numerical representation of data is known as a "hash value". Hash value is generated by specific hash function. Hash function accepts one data and generate hash value. Well-designed hash function generates completely different value from two different input data. Very small change of input data generates completely different hash value.Therefore, if data has been legally or illegally updated, no matter how little of a change, the final hash value will be entirely different. The hash value of data may be used to check its integrity. List 4.1 is the SHA-256 encryption code with Java programming language.

Listing 4.1: SHA-256 encryption Java code

```
1  public String calculateBlockHash() throw UnsupportedEncodingException {
2     String hashData = prevBlockHash+Long.toString(timeStamp)+Integer.
          toString(nonce)+transactions;
3
4     MessageDigest digest = null;
5     byte[] bytes = null;
6
7     try {
8        digest = MessageDigest.getInstance("SHA-256");
9        bytes = digest.digest(hashData.getBytes("UTF-8"));
10    } catch (NoSuchAlgorithmException | UnsupportedEncodingException ex)
          {
11       logger.log(Level.SEVERE, ex.getMessage());
12    }
13
14    StringBuffer buffer = new StringBuffer();
15    for (byt b : bytes) {
16       buffer.append(String.format("%002x",b));
17    }
18    return buffer.toString();
19 }
```

List 4.2 shows how the blockchain integrity is checked. The `isChainValid()` method will loop through the chain and compare the hashes. This method must ensure that the hash variable is equal to the calculated hash and that the hash of the previous block is equal to the `previousHash` variable. This method will return false if any changes are made to the blockchain's blocks.

Listing 4.2: Java code for Validity of blockchain

```
1  public static Boolean isChainValid() throws UnsupportedEncodingException
       {
2     Block currentBlock;
3     Block previousBlock;
4     String hashTarget = new String(new char[difficulty]).replace('\0',
          '0');
5
6     //loop through blockchain to check hashes:|
7     for (int i = 1; i < blockchain.size(); i++) {
8        currentBlock = blockchain.get (i);
9        previousBlock = blockchain.get(i - 1);
10       //compare registered hash and calculated hash:
11       if (!currentBlock.hash.equals(currentBlock.calculateBlockHash()))
             {
12          System.out.printInC'Current Hashes not equal");
13          return false;
```

```
14        }
15        //compare previous hash and registered previous hash
16        if (!previousBlock.hash.equals(currentBlock.previousHash)) {
17           System.out.printInC'Previous Hashes not equal");
18           return false;
19        }
20        //check if hash is solved
21        if (!currentBlock.hash.substring(0, difficulty].equals(hashTarget)
              ) {
22           System.out.printInC'This block hasn't been mined");
23           return false;
24        }
25     }
26  return true;
27  }
```

Because of the hash proof of work system, it takes a long time and a lot of computational power to generate new blocks. As a result, the attacker would require more computational power than all of the peers combined. List 4.3 is the mining the method to perform the proof-of-work.

It will necessitate miners performing proof-of-work by attempting various variable values in the block until its hash begins with a certain number of 0's.

Listing 4.3: Mining function Java code

```
1  public String mineBlock(int prefix) throws UnsupportedEncodingException
2     {
3        String prefixString =
4           new String(new char[prefix]).replace('\0','0');
5        while (!blockHash.substring(0,prefix).equals(prefixString)) {
6           nonce++;
7        blockHash = calculateBlockHash();
8     }
9     return blockHash;
10 }
```

It took some time to mine each block. If someone tampered with the data in your blockchain system, the blockchain would be rendered invalid, and he or she would be unable to create a longer blockchain. On the longest chain, honest blockchains in your network will have a time advantage. A tampered blockchain will be unable to catch up to a longer and more valid chain.

4.4 Web application

A web application is available for user interaction. User-friendliness and abstraction between the blockchain's underlying complexity and the high-level user interaction are the primary goals of this application. There are nodes and database connected to the application. It is used to send new transactions to the network and to receive information about incoming portions. A list of blocks is requested, and transactions related to a particular user are then filtered out. As a result, users can see how many portions the have at any given time. The detail of web application development will be explained in Chapter 7.

4.5 Network setup

Nodes communicate with each other over, the web app over TCP/IP protocols. We use the WebSocket protocol which is essentially a TCP-based protocol. It

```
run:
Trying to Mine block 1...
Block Mined!!! : 00000bb2315c7f25e4f329a73afff6192582c739bab71cbb17908ef92ac388f2
Trying to Mine block 2...
Block Mined!!! : 00000d02bdc6784c2cc62ce17aecec5cece562516ed7a7e47967b0beef97accf
Trying to Mine block 3...
Block Mined!!! : 000004fc1d63148ec56049218421f982d5052ac11e96693a6294ba6359b93e9e

Blockchain is Valid: true

The block chain:
[
  {
    "hash": "00000bb2315c7f25e4f329a73afff6192582c739bab71cbb17908ef92ac388f2",
    "previousHash": "0",
    "data": "This the first block",
    "timeStamp": 1634712335563,
    "nonce": 217045
  },
  {
    "hash": "00000d02bdc6784c2cc62ce17aecec5cece562516ed7a7e47967b0beef97accf",
    "previousHash": "00000bb2315c7f25e4f329a73afff6192582c739bab71cbb17908ef92ac388f2",
    "data": "This the second block",
    "timeStamp": 1634712336147,
    "nonce": 2089627
  },
  {
    "hash": "000004fc1d63148ec56049218421f982d5052ac11e96693a6294ba6359b93e9e",
    "previousHash": "00000d02bdc6784c2cc62ce17aecec5cece562516ed7a7e47967b0beef97accf",
    "data": "This the third block",
    "timeStamp": 1634712340481,
    "nonce": 455013
  }
]
BUILD SUCCESSFUL (total time: 6 seconds)
```

Figure 4.3: Result of the customized blockchain

first initiates a special request through the HTTP/HTTPS protocol for handshake and then creates a TCP connection for exchanging data, and then the server and the client communicate in real time through the TCP connection.

The node's API defines three main types of messages:(i)Block, (ii) Transaction and (iii) status. It is possible for a block message to propagate a newly mined block or to respond to a block request from another node. The web app sends transaction messages, which cause portions to be shifted. As well as being used for debugging, status messages are used to inform users of the node ' s current status.

In our permissioned blockchain, the public keys of all nodes are known to one another and messages with invalid signatures are discarded.

In order to establish a secure connection between the server and the web app, HTTPS is used. Over private virtual network (VPN), all components of our system communicate with one another. The VPN simplifies communication on the network layer because the nodes are located on the different premises, where unchanging IP addresses are not guaranteed and network address translation (NAT) is common.

4.6 Implementation result

After finishing the implementation, we executed our code. Figure 4.3 is a screenshot of running. As figure 4.3 shows, our handmade blockchain system was successfully constructed. Some data, which is shown as "`This is the first`

`block`" etc. could be stored in the block and the value of "`nonce`" was also correctly computed. After finishing the construction of blockchain system, we can develop our target system, residents life event management system, on this handmade blockchain system.

Chapter 5

Implementation by using MultiChain framework

This chapter describe the implementation of target system by using MultiChain framework. MultiChain is an open source fork of Bitcoin that has been extended. It is simple to configure and can be used to launch custom blockchains, both private and public. It provides a carefully curated set of features and enhancements aimed at enterprise and business users.

5.1 Introduction to MultiChain Framework

Installing Multichain is not difficult. Implementor only needs to follow several instructions for installation. To install Multichain it is required some specific hardware and software components. They are as follows:

- **OS**: Linux 64-bit (Ubuntu 12.04+, CentOS 6.2, Debian 7+, Fedora 15+, RHEL 6.2+) or Window 64 bit (Windows 7 or later)
- **RAM**: Minimum 512 MB
- **Disk space**: minimum 1 GB

Our prototype system uses on CentOS 7.3 with 2 GB RAM. Firstly, user must download the package of Multichain platform from website[29]. At the date and time of this thesis, user can download version 2.0.7. After downloading the archive, user only need to expand archive file to specific directory in the system. In our prototype system, Multichain is installed on Linux by executing following commands:

```
$ su       # for changing normal user mode into super-user mode
$ cd /tmp
$ <downloading Multichain archive from website>
$ tar -xvzf multichain-2.0.7.tar.gz
$ cd multichain-2.0.7
$ mv multichaind multiclain-cli multichain-util /usr/local/bin
$ exit     # leave super-user mode
```

5.2 Creating and connecting blockchain with Multichain

After installing Multichain platform, user has to create his/her own blockchain into Multichain platform. To create a new blockchain, user must run the following command:

```
$ multichain-util create <name-of-blockchain>
```

If necessary, user can create a clone of existing blockchain:

```
$ multichain-util clone <old-name-of-blockchain> <new-name-of-blockchain>
```

User, of course, can also set any parameter on the command line using the same name, for example:

```
$ multichain-util create <name-of-blockchain>\
   -maximum-block-size = 20895656
```

After these settings are finalized, user can start running the blockchain in the background (daemon) with the following command:

```
$ multichain <name-of-blockchain> -daemon
```

The parameters file params.dat will be locked, initialize the blockchain and create in the first block (genesis). For the first to connect time to an existing blockchain, first user needs to obtain the node address. The node address is shown whenever multichaind starts up or can be get from the getinfo API call. User can connect using the node address as follows:

```
$ multichain <name-of-blockchain>@<IP_ADDRESS>:<PORT> -daemon
```

In case of private blockchain, user is able to connect immediately because user have not yet been approved permission by an administrator. After permission was approved, user can immediately reconnect to name-of-blockchain using the short form:

```
$ multichain <name-of-blockchain> -daemon
```

Multichain provides full control over permissions at the network level. There are many global permissions that can granted to addresses connect, send, receive, issue, create, mine, activate, admin. All permissions are assigned on a per-address basis, where addresses can either be public key hashes or script hashes. Permissions can be made temporally by limiting them to a specific range of block numbers. All permissions are granted and revoked using network transactions containing special metadata, which are easy to send using grant and revoke commands for `multichain-cli` or the JSON-RPC API. The creator of chain's first genesis block has the all basic permissions. The purpose of blockchain is to create decentralized databases, which are not controlled by any third party. It is necessary to extend the philosophy to the administration of permissions.

5.3 Connecting MultiChain framework and PrimeFaces framework

In order to interact with the blockchain and get responses, we have to connect to the chain using RPC (Remote Procedure Call). The format of the RPC call is the following:

```
{"method":<method_name>,"params":<params_as_array>,"id":<unique_id>,
 "chain_name":<chain_name>}
```

So, in order to interact with the chain, we must create an RPC call with the following parameters:

- `Method_name`: this parameter is the name of the API command we want to call. So if we want to get information about the blockchain we can call the "`getinfo`" command and get all the information. Or if we want to create a new stream in the blockchain we call the "`create`" command.
- `Parameters_if_any_else_pass_null_value`: parameters are not always necessary. There are some API commands that takes no parameters. Examples of those commands are 'getinfo", "`help`" or "`stop`". There are also commands that take one or more parameters. Examples of those commands are "`liststreams`", "`listaddresses`" and many more.
- `Unique_id_number`: this is the number we have to pass to the RPC. It is the unique id of the call and you can generate it easily using java's UUID class.
- `Chain_name`: this is the name of the blockchain. We gave it to the chain when we first generated it. In our example, it is `chain1`". But this parameter is not actually needed, but if it is there it is used as a check.

5.3.1 RPC Client

The first step is to create the client, set credentials, and HTTP connection parameters and use it. Our client needs to know the IP, Port, Username, and Password in order to connect to the chain. We are using Apache `httpclient`. Our configuration file with name `HttpClientConfig` the methods we use for setting the parameters to the HTTP client. In order to set up HPPT client, we have the following functions :

- Function to get the provider credentials: Following is a source code of this function.

```
/**
 * Get new BasicCredentialsProvider
 */
public CredentialsProvider getBasicCredentialProvider() {
   return new BasicCredentialsProvider();
}
```

- Function to configure our parameters: source code of this function is as follow:

```
/**
 * Configure Credentials Provider
 *
 * @param credentialsProvider the credentials provider
 */
public CredentialProvider configure(
       CredentialsProvider credentialsProvider) {
   credentialProvider = getBasicCredentialProvider();
   credentialProvider. setCredentials(
      new AuthScope (
                     chain.getIp(),
                     chain.getPort(),
      new UsernamePasswordCredentials(
                     chain.getUsername(),
                     chain.getPassword())
      );
      return credentialsProvider;
}
```

- Function to set the credential provider on the HTTP client: Following is a source code of this function.

```
/**
 * Get a CloseableHttpClient
 */
public CloseableHttpClient getHttpClient() {
   return HttpCLientBuilder
         .create()
         .setConnectionManagerShared(true)
         .setDefaultcredentialProvider(
               configure(getBasicCredentialProvider()))
         .build();
}
```

5.3.2 The blockchain class

To interact with the blockchain , the client must know the IP, Port, Username, and Password. If there are two or more chains, this information will be passed as an object. This data is stored in the Chain class. As a result, when we configure our application, we add this information to the Chain object. We are ejecting the Chain object into the Configuration class, and we have access to all of the information about the IP, Port, and so on.

5.3.3 Call and responses

After completing all of the preceding steps, our client is now ready to make RPC calls and receive responses. The responses we receive are determined by

```
Oct 26, 2021 12:01:20 PM chain.MultichainService getStringResponse
INFO: Multichain API Successfully Responded
Response: null
Oct 26, 2021 12:01:20 PM chain.MultichainService getStringResponse
INFO: Multichain API Successfully Responded
```

Figure 5.1: Successful connection of multichain framework

the API command we used to initiate the call in the first place. The responses are JSON objects with a variety of properties. As a result, making it a POJO class is not so simple. To make it POJO classes for the responses, and each class could have the JSON properties we expect as the response as properties. Each of us could make a generic object. This is a more complicated solution, but it is a good one. Another option is to create a single POJO class that contains all of the properties. We'll make our own annotations and tag the appropriate properties in our POJO with them so we know which properties correspond to which API command. As a result, we'll add a @Getinfo annotation to every property that we expect to have a value after the "`getinfo`" call. As a result, the `@Getinfo` annotation will be applied to 27 properties. Figure 5.1 shows the successful connection to the multichain blockchain framework.

5.4 Implementation result

Multichain provides full control over permissions at the network level. There are many global permissions that can granted to addresses connect, send, receive, issue, create, mine, activate, admin. All permissions are assigned on a per-address basis, where addresses can either be public key hashes or script hashes. Permissions can be made temporally by limiting them to a specific range of block numbers. All permissions are granted and revoked using network transactions containing special metadata, which are easy to send using grant and revoke commands for `multichain-cli` or the JSON-RPC API. The creator of chain's first genesis block has the all basic permissions. The purpose of blockchain is to create decentralized databases, which are not controlled by any third party. It is necessary to extend the philosophy to the administration of permissions.

We created two nodes for our testing. Each are virtual machines with Ubuntu OS. They will grant each other a permission to get connected. Figure 5.3 shows the initialization of node 1 (chain1) by the node 2 (chain2).

Figure 5.4 demonstrates the permissions granted by node 1 to node 2.

Figure 5.5 is a process of the initialization of node 2 (chain2) by the node 1 (chain1).

And figure 5.6 demonstrates the permissions granted by node 2 to node 1.

As a result, we could confirm the success of implementing blockchain system with Multichain framework.

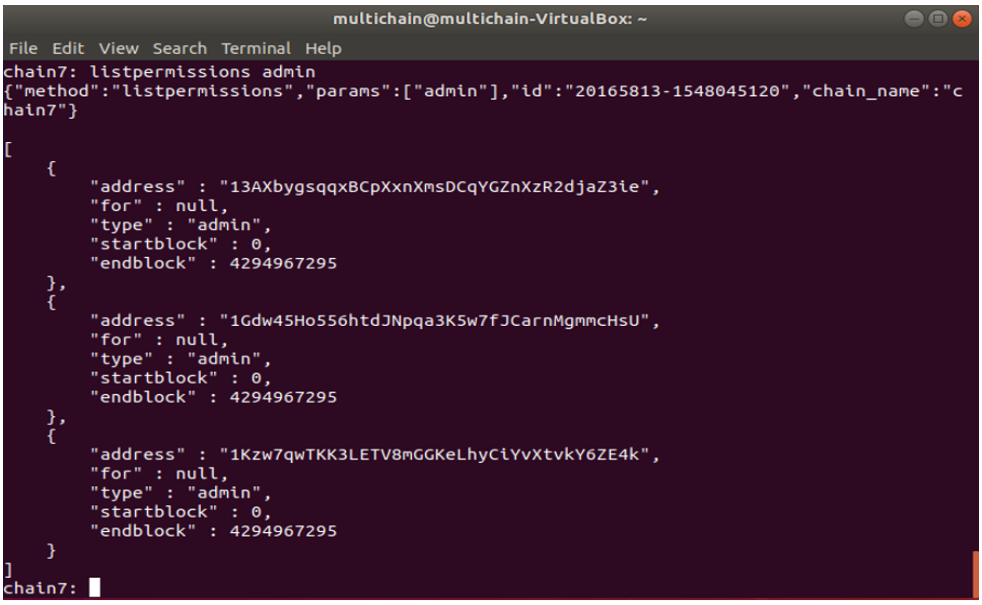

Figure 5.2: Permissions consensus

Figure 5.3: Initialization of two nodes

Figure 5.4: Permissions granted by node 1 to node 2

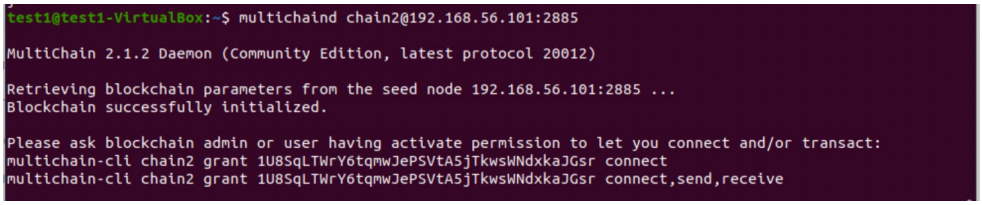

Figure 5.5: Initialization of node 2 (chain2) by the node 1 (chain1)

Figure 5.6: permissions granted by node 2 to node 1

Chapter 6

Implementation by using Hyperledger Fabric framework

6.1 Overall of Hyperledger Fabric

Hyperledger Fabric is an open source enterprise-grade blockchain platform. By using Hyperledger Fabric, users can implement distributed ledger technology (DLT). Hyperledger fabric can be used to develop from small private-level system to large enterprise-level system. It is developed by Linux Foundation. Now Linux Foundation manages the development and release of it. Hyperledge Fabric support mainly Consortium and Private blockchain. It can be used for general-purpose system development.

There are some features of the Hyperledger Fabric that distinguish it from other blockchain frameworks. This is what we need to know to keep things simple. Hyperledger Fabric is an approved blockchain. It is not intended to be accessible to the world, but as many individuals as you like can be added. It promotes smart contracts, much as Ethereum does. In Hyperledger Fabric, smart contract is called chaincode (the specific term for smart contract code from Hyperledger Fabric). The smart contracts of Hyperledger Fabric are written in Go language, Java and JavaScript. It is an approved blockchain. It is not intended to be accessible to the world, but as many individuals as you like can be added. It promotes smart contracts, much as Ethereum does.

There is a concept of "channels" in Hyperledger Fabric, where parties that are part of a blockchain may privately establish separate transactions and then send the final state to the main blockchain to be registered. In other blockchains, this is not unlike state networks, but there is an extra layer of privacy. All respondents have recognized identities, preserved by what Hyperledger Fabric calls "Membership Service Providers" (MSP). If you authorize a group of 10 hospitals to participate in your blockchain, the network will be aware of each of the 10 hospitals. This is Hyperledger Fabric's main feature that makes it work well with enterprise solutions.

To reach consensus, Hyperledger Fabric does not use traditional Proof of Work (PoW) or Proof of Stake (PoS) processes. Instead, it uses a sequence of checked transactions, since it is highly approved. Chaincode can only be used to approve a transaction if it is also checked by two parties who execute a transaction, each signing it and then peers who take the transaction. In short, what you need to think about for now is that for them to get included in the ledger, multiple checked participants need to sign transactions. This is a perfect

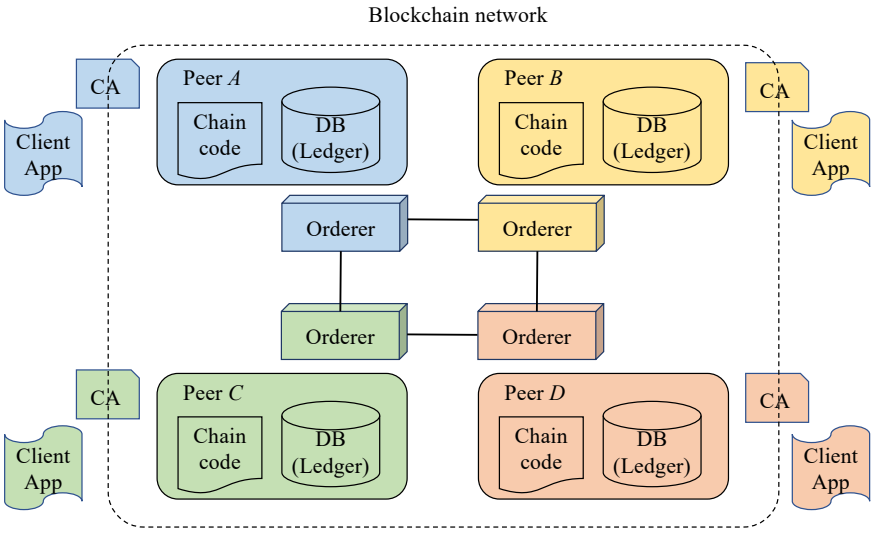

Figure 6.1: Conceptual structure of Hyperledger Fabric

match for blockchains focused on companies that don't have many participants. There is a natural barrier to malicious activity since the participants are familiar to each other. Here, read more. This is a basic flow diagram for transactions. For now, you do not need to comprehend all the specifics. Figure 6.1 shows the conceptual structure of Hyperledger Fabric blockchain network.

In this figure, the same color components correspond to each organiza- tion. There are several basic components in Hyperledger Fabric:

Peer: Each peer contains chaincode and DB. Chaincode describes the smart-contract program.

Chaincode: Chaincode is a kind of application program which implements the smart contract.

Orderer: Orderer controls the order of transaction on the network.

Note that the difference of "chaincode" and "client app" is that chaincode describes the program to implement the smart contract which maintains the transaction logic. By using chaincode, each transaction must be certificate by the consensus of all peers. No certificated transaction will be added to DB. On the other hand, "client app" is an end-user application which uses the data in the blockchain database. This app is almost similar to the usual database application program. More details should be referred to [30] or other resources.

6.2 Hyperledger Fabric workflow

Hyperledger Fabric works as following order:

1. **Create transaction proposal:** The client application in the peer creates a proposal of transaction. This proposal is submitted to blockchain network.

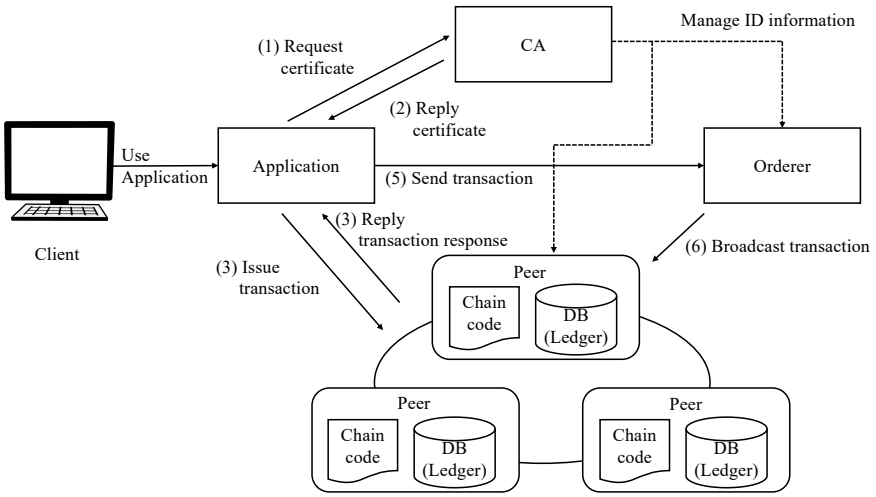

Figure 6.2: Illustration of Hyperledger Fabric workflow

2. **Consensus of transaction:** After submitting the proposal of trans- action to blockchain network, each peer checks if this transaction is valid or not by using chaincode. The results of all peers' chaincode are returned to the client.

3. **Submission to orderer:** After being approved by all peers, the proposal of transaction is sent to the orderer which orders the transaction into a block within the database.

4. **Commitment of transaction:** After finishing the ordering by orderer, the transaction will be sent to all peers to add the ledger.

Figure 6.2 is an illustrated explanation of Hyperledger Fabric workflow. As Hyperledger Fabric clearly separates the consensus algorithm and end-user application, it enable flexible system configuration.

Hyperledger Fabric can be installed on many kinds of machines. Support OS includes Linux, Windows, and macOS. Instruction steps are described many support sites such as [**8**]. After following few instructions, the implementor has nothing further to do. Some hardware and software requirements must be fulfilled to successfully install Hyperledger Fabric. Requirements are show as follow:

- OS: Ubuntu 14.04/16.04LTS (both 64-bit), or macOS 10.12 or greater version
- Software tools:
  - `cURL`: the latest version
  - `git`: the latest version
  - `Docker` engine: version 17.06.2-ce or greater
  - `Docker-compose`: version 1.14 or greater
  - Go language: version 1.13.x

- Node: version 8.9 or later. Note that version 9 is not supported, and version 10 is supported from 10.15.30.
- `npm`: version 5.x
- Python: 2.7.x (3.x.x is not supported).

Install steps of Hyperledger Fabric on the Linux machine is as follow:

1. Install `curl` and `golang`:

   - `sudo apt-get install curl`
   - `sudo apt-get install golang`

   The package golang installs the Go distribution to `/usr/local/go`. The package should put the `/usr/local/go/bin` directory in your `PATH` environment variable. You may need to restart any open terminal sessions for the change to take effect.

2. `export GOPATH=$HOME/go`

3. `export PATH=$PATH:$GOPATH/bin`

   - Install `node.js`, `npm` and Python:
     - `sudo apt get install nodejs`
     - `sudo apt get install npm`
     - `sudo apt get install python`
   - Install Samples, Binary and docker images:
     - `curl -sSL https://bit.ly/2ysbOFE | bash -s`
     - `curl sSL https://bit.ly/2ysbOFE | bash -s -fabric version fabric-ca version thirdparty version`
     - `curl sSL https://bit.ly/2ysbOFE | bash -s - 2.0.1 1.4.6 0.4.18`
   - Export binary path:
     - Go to `FabricSamples` directory
     - Go to `bin` directory
     - Add this path to `$PATH` variable in `~/.bash_profile` .
   - Export `PATH:$PATH:/Users/<your user name>/fabric-sample_2.0/fabric-samples/bin`.

6.3 Invoking and connecting network with Hyperledger Fabric

In Hyperledger Fabric, blockchain is implemented as a virtual network of several nodes. A blockchain network offers a variety of infrastructure services to various applications, such as accounting and contracts. Primarily, smart contracts (chaincode) are used to record transactions on every peer node's ledger. End users may utilize either client applications or the blockchain network itself

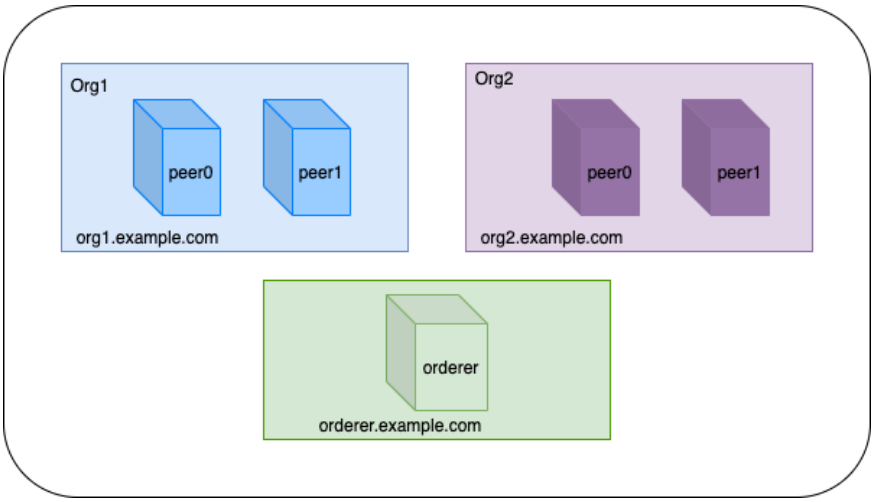

Figure 6.3: First network structure

```
KE212s-MBP:channel ke212$ ./create-artifacts.sh
mychannel
2021-06-22 11:21:22.482 JST [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-06-22 11:21:22.489 JST [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: etcdraft
2021-06-22 11:21:22.489 JST [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaft.Options unset, setting to tick_interval
:"500ms" election_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2021-06-22 11:21:22.489 JST [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: configtx.yaml
2021-06-22 11:21:22.492 JST [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
2021-06-22 11:21:22.492 JST [common.tools.configtxgen] doOutputBlock -> INFO 006 Creating system channel genesis block
2021-06-22 11:21:22.492 JST [common.tools.configtxgen] doOutputBlock -> INFO 007 Writing genesis block
2021-06-22 11:21:22.524 JST [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-06-22 11:21:22.531 JST [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: configtx.yaml
2021-06-22 11:21:22.531 JST [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2021-06-22 11:21:22.534 JST [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
KE212s-MBP:channel ke212$
```

Figure 6.4: Creating channel

as clients. After installation, we start by implementing the first network which will have the structure shown in Figure 6.3:

There is just one node in Orderer at the level of nodes. In Org1 and Org2, each organization has two peers. Peers `Peer0` and `Peer1` are each known as peer. There are thus just five nodes in the First Network. When installed locally, certain components inside First Network are implemented as containers. Docker images are produced from Hyperledger Fabric's Docker images. In the localhost, we initially have five containers running. Typically, a Docker Compose-based container configuration is established using a suitable Docker Compose YAML file. When we start working on the chaincode, there are more containers running.

In order to use blockchain network, the user must create the "channel". The channel is created such that each new channel starts with block 0. The Orderer creation has finished, and block 0 has been formed. Block 0 is created as a file called mychannel.block after the execution of `create-artifacts.sh` command shown in Figure 6.4.

With the mychannel.block, the peers join the channel. The command peer channel join foin for each peer joining the channel shown in Figure 6.5. Next step is anchoring peers. Anchoring means the allocation of "anchor" to each organization. Anchor is an access points between different organizations in blockchain network. Hyperledger Fabric provides shell script to allocate anchor. Figure 6.6 is an execution result of shell-script.

The final step is an activation of network. After restarting machine, Hy-

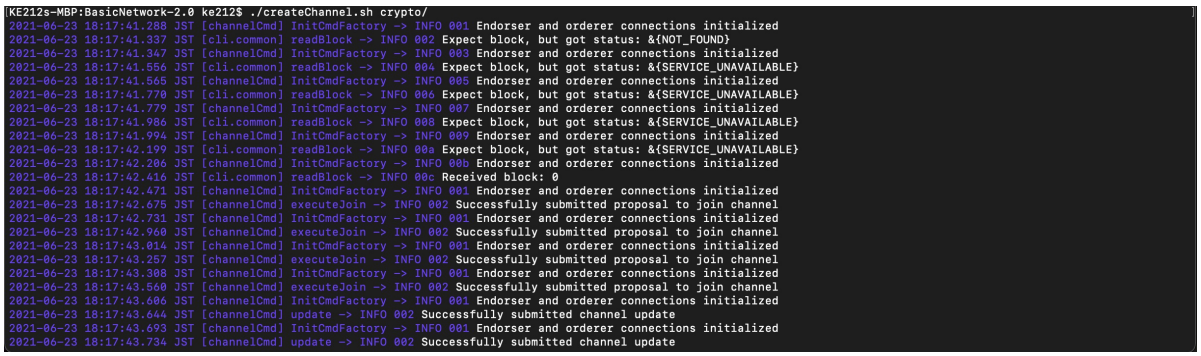

Figure 6.5: Peer joining the channel

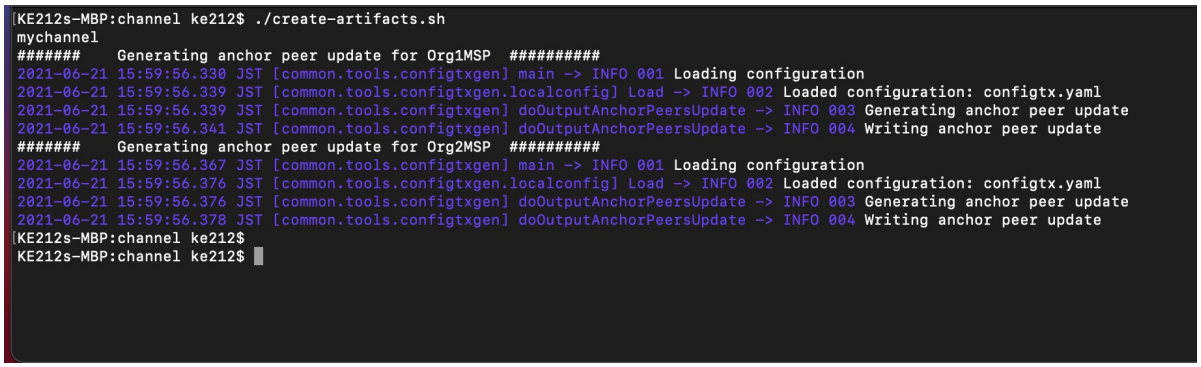

Figure 6.6: Anchoring peers

perledger Fabric automatically awakes the network shown in Figure 8. After installing Hyperledger Fabric platform, it is necessary to activate network and connect the external Java web application. To make this connection, we will use the Hyperledger Fabric Gateway SDK[31] which allows applications to interact with a Hyperledger Fabric blockchain network. Listing 6.1 is the code sample from the official website. Listing 6.1 is a part of sample code written in Java. By executing this sample code, you can verify whether you installation is succeeded or not.

Listing 6.1: Sample code of connecting Hyperledger Fabric to application

```
1  import java.io.IOException;
2  import java.nio.charset.StandardCharsets; import java.nio.file.Path;
3  import java.nio.file.Paths;
4  import java.util.concurrent.TimeoutException;
5  import org.hyperledger.fabric.gateway.Contract;
6  import org.hyperledger.fabric.gateway.ContractException;
7  import org.hyperledger.fabric.gateway.Gateway;
8  import org.hyperledger.fabric.gateway.Network;
9  import org.hyperledger.fabric.gateway.Wallet;
10 import org.hyperledger.fabric.gateway.Wallets;
11 class Sample {
12    public static void main(String[] args) throws IOException {
13    // Load an existing wallet holding identities used to
14    // access the network.
15    Path walletDirectory = Paths.get("wallet");
16    Wallet wallet = Wallets.newFileSystem Wallet(walletDirectory);
17    // Path to a common connection profile describing the network.
18    Path networkConfigFile = Paths.get("connection.json");
19    // Configure the gateway connection used to access the network.
```

```
20     Gateway.Builder builder = Gateway.createBuilder()
21                           .identity(wallet,"user1")
22                           .networkConfig(networkConfigFile);
23     // Create a gateway connection
24     try (Gateway gateway = builder.connect()) {
25        // Obtain a smart contract deployed on the network.
26        Network network = gateway.getNetwork("mychannel");
27        Contract contract =
28           network.getContract("fabcar");
29        // Submit transactions that store state to the ledger.
30        byte[] createCarResult = contract.createTransaction("create Car")
31           .submit("CAR10", "VW", "Polo", "Grey", "Mary");
32        System.out.println(new String(createCarResult ,
33               Standard Charsets.UTF_8));
34        // Evaluate transactions that query state from the ledger.
35        byte[] queryAllCarsResult =
36              contract.evaluateTransaction("queryAllCars");
37        System.out.println(new String(queryAllCarsResult,
38                              Standard Charsets.UTF_F8));
39     } catch (ContractException TimeoutException InterruptedException e)
40        {e.printStackTrace(); }
41     }
42  }
```

6.4 Design and implementation of revised RLEMS with PrimeFaces and MySQL

6.4.1 Overview

To launch our prototype system as a web application, we had to try to build it. We used Hyperledger Fabric blockchain, MySQL database system, and Java web framework PrimeFaces to develop our RLEMS as a web application.The structure of revised system is almost same as previous version.

Overall structure of this system is almost same as previous system. In a previous system, we used MultiChain as a framework of blockchain. In this new system, we used Hyperledger Fabric as a framework of blockchain. MySQL database system, and Java web framework PrimeFaces[24] to develop our RLEMS as a web application. In a previous system, we also used the PrimeFaces framework to implement web service interface. Current system also use the same framework to enable the user interface implementation without changing it. Furthermore, thanks to the adoption of same DBMS and web framework, the number of new codes decreased. We only need to implement blockchain platform related software and some interface between web framework and Hyperledger Fabric.

By using MySQL and blockchain technology, the data will transmit and receive from and to the user interface. We have our application, database, and blockchain platform all together in one system. The Java web framework is often used to do such tasks as inserting, updating, and deleting data on the blockchain. Data that must be sent to the blockchain technology and MySQL database may be sent using the Hyperledger Fabric framework's module. The data is encrypted and hashed to protect it from outside access. No need to fear since this data can be transferred over the internet. The information inside the data may only be given to the other entity that has an authorization from the blockchain network. The core part of this system is the web application built using the Java web platform. First, we try it out. A user must be logged in. After successfully logging in, the user is allowed to add, edit, or remove information.

This new blockchain module is meant to be incorporated into Multichain, which will govern the information flows.

The tests below are examples of communication between the web ap- plication and MySQL database. Authentication must be established at the beginning of the web application. When we have the consideration that in mind, our intention is to get a registration for births in the national database. You'll need to provide information about the kind of hall, the hall's name, a prefecture, an area, and information about the infant to get a birth certificate at the end. A static of birth by area, by prefecture, and by hall will be created using the online application.

6.4.2 Chaincode of our application

As Hyperledger Fabric provides the flexibility of application development by using the "chaincode", which is a kind of application program description. By developing the appropriate chaincode for each application, users can implement their required application flexible on Hyperledger Frabric. List 6.2 describes the part of our chaincode written in Java. The total line of our chaincode is approximately 2,000 lines.

Listing 6.2: Part of our chaincode written in Java

```
1  import org.hyperledger.fabric.contract.Context;
2  import org.hyperledger.fabric.contract.ContractInterface ;
3  import org.hyperledger.fabric.contract.annotation.Contract ;
4  import org.hyperledger.fabric.contract.annotation.Default ;
5  import org.hyperledger.fabric.contract.annotation.Info ;
6  import org.hyperledger.fabric.contract.annotation.Transaction ;
7  import static java.nio.charset.StandardCharsets.UTF_8 ;
8
9  @Contract(
10     name = "MyDataContract",
11     info = @Info(
12         title = "Data contract",
13         description = "A very basic contract",version="0.0.1-SNAPSHOT")
14 )
15 @Default
16 public class MyDataContract implements ContractInterface {
17    public MyDataContract() {
18    }
19    @Transaction()
20    public boolean myDataExists(Context ctx, String myAssetId) {
21       byte[] buffer = ctx.getStub().getState(myDataId);
22       return (buffer != null && buffer.length > 0);
23    }
24
25    @Transaction()
26    public void createMyData(Context ctx, String myDataId, String value)
27    {
28       boolean exists = myAssetExists(ctx, myDataId);
29       if (exists) {
30          throw new RuntimeException(
31               "The data " + myDataId + " already exists"
32               );
33       }
34       MyData data = new MyData();
35       data.setValue(value);
36       ctx.getStub().putState(myDataId,
37                              asset.toJSONString().getBytes(UTF_8));
38    }
39    @Transaction()
40    public MyData readMyData(Context ctx, String myDataId) {
41       boolean exists = myDataExists(ctx, myDataId);
42       if (!exists) {
43          throw new RuntimeException(
```

```
44                      "The data "+myDataId+" does not exist"
45                );
46          }
47          MyData newData = MyData.fromJSONString(new String(ctx.getStub().
                getState(myDataId), UTF_8));
48          return newData;
49       }
50       @Transaction()
51       public void updateMyData(Context ctx, String myDataId, String
             newValue) {
52          boolean exists = myDataExists(ctx, myDataId);
53          if (!exists) {
54             throw new RuntimeException("The data " + myDataId + " does not
                   exist");
55          }
56          MyData data = new MyData();
57          data.setValue(newValue);
58          ctx.getStub().putState(myDataId, data.toJSONString().getBytes(
                UTF_8));
59       }
60       @Transaction()
61       public void deleteMyData(Context ctx, String myDataId) {
62          boolean exists = myDataExists(ctx, myDataId);
63          if (!exists) {
64             throw new RuntimeException("The data " + myDataId + " does not
                   exist");
65          }
66          ctx.getStub().delState(myDataId);
67       }
68    }
```

By using Hyperledger Fabric, we could install the blockchain system snd our application described above.

Chapter 7

Web service implementation with PrimeFaces framework

We tried to implement our prototype system as a web application. To implement our resident management system as a web application, we combine blockchain technology, MySQL database system and Java web framework PrimeFaces.

The data will send and receive from/to user interface through MySQL and blockchain technology. Our system is composed by web application, MySQL database and blockchain technology. The Java web framework is used to insert, update and delete data in the blockchain. There is a module which is used to send data to the blockchain technology and MySQL database. To protect data from illegal access from outside of the network, the data will be hashed and encrypted. These data can be sent through internet without any worry. Only the other entity who has a approval in the blockchain network can get the exact information inside the data.

The fundamental module of this system is the web application developed with the java web framework. We test it firstly. A user needs to be authenticated. After the authentication with his role, the user can insert, update or delete an information. The blockchain module is to be embedded and control the flux of information.

The following test is the communication between the web application and MySQL database on the one hand, in the other hand the blockchain technology. At the starting of the web application, we need to authenticate. We must have in mind that; our purpose is to get a registration for births in the national record. To have a birth certificate at the end, the user needs to have information about Type of hall, Hall name, prefecture, region, information about the child. With the web application, we will have a static of birth by region, by prefecture and by hall.

The test of creating block, creating a consensual governance model and using multiple keys for extra security in blockchain technology is an important test work.

7.1 General description of Java web application

Web server is a software that can process the client request and send the response back to the client. Apache is one of the most widely used example web server. Web server is installed on some physical machine and listens to client

request on specific port. Web client is a software that helps in communicating with the server. Our browsers can be considered as web client: Firefox, Google Chrome, Safari, etc. when something is requested from server (through URL), web client takes care of creating a request and sending it to server and then parsing the server response and present it to the user. Web server and client uses a common communication protocol, HTTP (Hyper Text Transfer Protocol) which is the communication protocol between server and client. HTTP runs on top of TCP/IP protocol.

Java web application is used to create a dynamic website. Basically, any websites are created with static HTML pages. It means that the user always get the same webpage. However, the information displayed on the screen will be dynamic when web application is used. A web application provides a dynamic ex- tension of web or application server. Web applications are two types: presentation-oriented and service-oriented.

- Presentation-oriented generates an interactive web pages containing various types of markup language (HTML, XML, and $\cdots$) and dynamic content in response to requests.
- Service-oriented implements the endpoint of a web service. Presentation-oriented applications are often clients of service-oriented web applications.

Java servlet technology is the foundation of all web application technologies. There are many java web technologies. We can name: Java Server Pages, Java Server Faces, Java Server Pages Standard Tag Library. Java web applications are packaged as web archive (WAR) and it has a defined structure we can export above dynamic web project as WAR file and unzip it to check the hierarchy.

7.2 Sample User interface

Firstly, when the user accesses the web application, the authentication is required, which is shown in Figure 7.1. After being authenticated, the user can use the application according to his access rights. In order to access functions, user will use the main page which contains function menu. You can find the selection window in Figure 7.2. For example, if the user is an administrative user, he/she can create, update or delete data in the database. Figure 7.3 and Figure 7.4 are sample interface for updating data in the database. After the finish of data registration, end users can get their official record by entering their information through the web. Users enter the information through the interface shown in Figure 7.4. User can get, for example, following birth certificate for official use. Figure 7.5 is a sample official birth certificate document generated by the system.

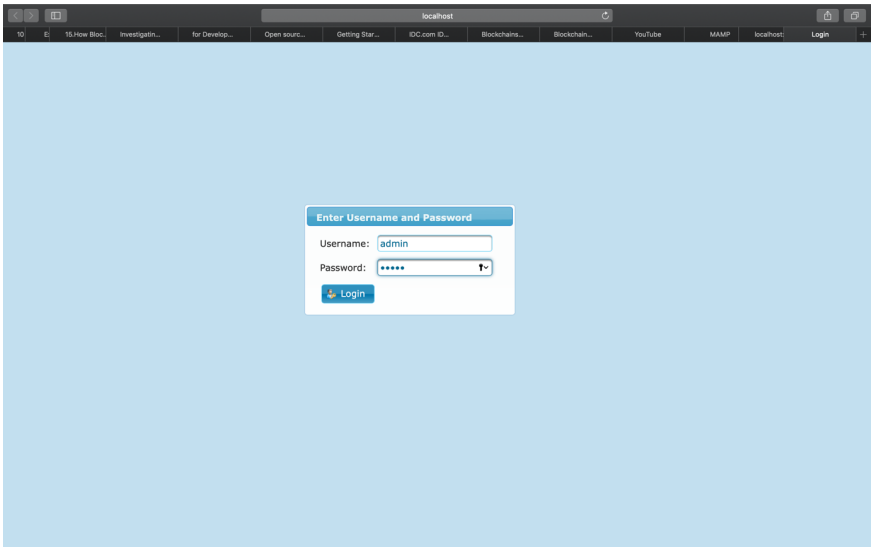

Figure 7.1: Authentication window

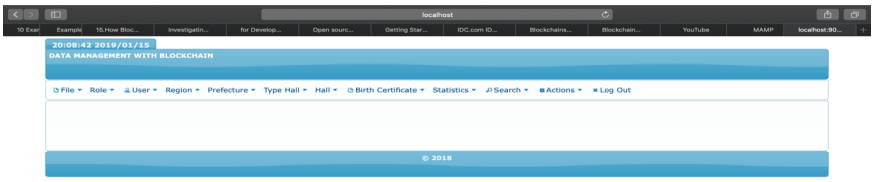

Figure 7.2: Main menu window

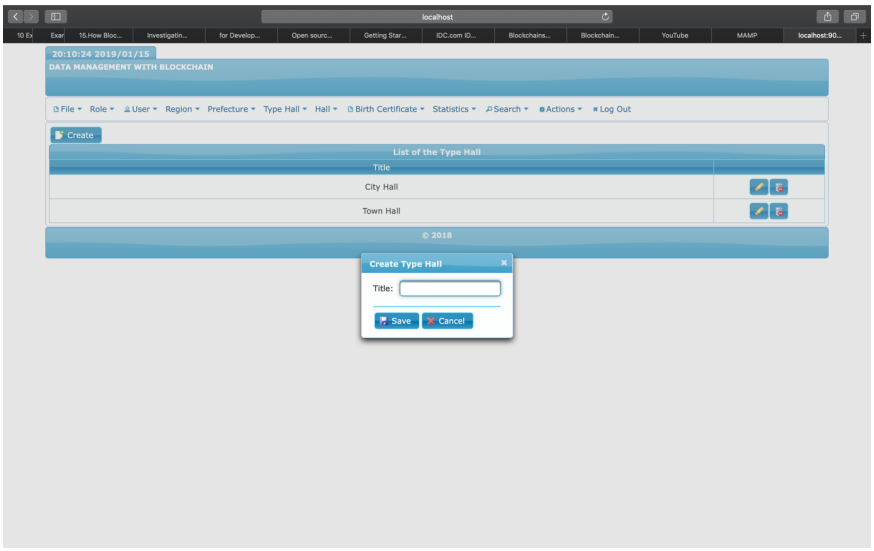

Figure 7.3: Window for updating data in the database.

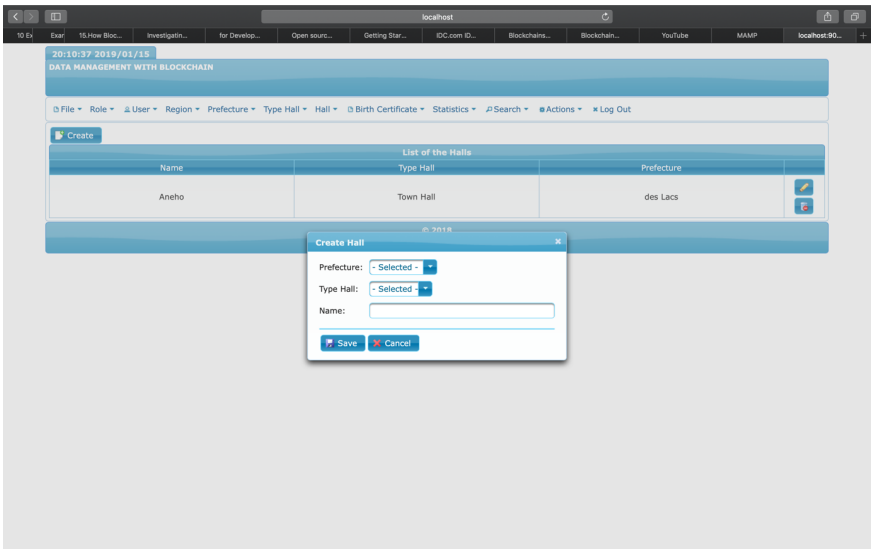

Figure 7.4: Entering user information to get user's official record.

BIRTH CERTIFICATE

Registration Date: 3/31/18 12:00 AM

Refrence number: TG9995406E-3C35-4FE2-8663-5E4BA7C69C6BTG

Child Name: AGBODJAN Kafui Pelagie

Father Name: AGBODJAN Kodjovi

Gender: Female

Father Age: 36

Date of birth: 3/20/18 12:00 AM

Father Occupation: Farmer

Place of birth: Hilacondji

Mother Name: MESSAN Akpedje

City hall/Town hall: Hilacondji

Mother Age: 24

Prefecture: des Lacs

Mother Occupation: House wife

Figure 7.5: Generated official birth certificate.

Chapter 8

Comparison of the three implementations

8.1 Comparison of no-framework implementation and with-framework implementation

In comparison to the previously used frameworks MultiChain and Hyperledger Fabric, using no framework has the following advantages:

- Compare to the two blockchain frameworks MultiChain and Hyperledger Fabric, our customize built blockchain is flexible and not huge. It has few functions. We can add functions as and when we need.
- Our customize blockchain does not require high-level computing environment.
- It is developed for a specific purpose and it id suitable for our application.
- Flexibility of implementation language: we can use any programming language to develop our blockchain. We have chosen to use Java programming language.

8.2 Comparison of MultiChain framework and Hyperledger Fabric framework

Comparing to MultiChain, which was used in previous system, Hyperledger Fabric has following advantages:

- Flexibility of organization structure:Basically MultiCahin is for the development of private blockchain system. On the other hand, Hyperledger Fabric is not restricted to private blockchain system. It supports consortium blockchain, which is suitable for our application. Furthermore, Hyperledger Fabric clearly separate the blockchian system and application development. Especially "smart contract", which is the program triggered by the transaction in the blockchain or any information imported from outside of the system. Hyperledger Fabric support this "smart contract" by "chaincode", and MultiChain introduced the "smart filters" from version 2.0. The structure of "smart contract" of Hyperledger Fabric is clearer than that of MultiChain. Therefore, Hyperledger Fabric supports more flexible and easy to develop functions.

- Flexibility of implementation language: Both of these two platforms support Java and JavaScript. Hyperledger Fabric supports Go language. But there is not so serious difference. Developing application with both of these platform has no serious difference.

- Richness of support information: Hyperledger Fabric is supported by IBM, Intel, NEC and other major venders. Therefore finding support information such as implementation, developing application, operations, and learning courses is easy. Comparing Hyperledger Fabric, MultiChain has less support information. Therefore, learning, implementing and operation is easier in Hyperledger Fabric.

8.3 Conclusion of comparison

Three methods were to implement our system. We used no framework for the first implementation and we used two frameworks Multichain and Hyperledger Fabric for two others implementations. Based on these three implementation, we decided to use no framework to implement our target system. Of course, using frameworks has some advantages. However, in order to implement frameworks, a lot of efforts, time, and supports are necessary. It takes time to learn, to implement all the features of these two frameworks, MultiChain and Hyperledger Fabric. They are huge and we cannot use all their functions. In other side, it is more flexible to not use any framework and features are added as things progress. Furthermore, as almost all frameworks are developed for general-purpose, some unnecessary functions are implemented. And for example, even if we do not use function f_1, we have to install f_1 to use the framework. And if the implementation of function f_1 requires specific environment such as support library software, we have to install this library software. But when we install more library software, we sometimes meet some problems about installation such as compatibility of software version. When we tried to install Hyperledger Fabric, we met this problem and a lot of effort were used to resolve this trouble. If we do not use any framework, we can install only functions we need. We recommend the implementation without any framework to have more flexibility and more control. Therefore, our final conclusion is

> Use of no framework is better than the use of any framework for our purpose

Chapter 9

Conclusion

We implemented a high-reliability and secure data management system using blockchain in this article. With the help of blockchain technology, a dependable and secure data management system may be easily implemented. Our prototype attempted to find out how dependable and safe the blockchain technology is when it comes to data management. Furthermore, we used the user interface for the online application. No need to make any application installations on consumers' personal computers or cellphones when using a web application strategy. By using this strategy, the security of data management is maintained. Future work includes speeding up processing. Currently, the computational time required to link one block to an outgoing blockchain increases due to the necessity to search for an adequate nonce value. The system must handle more requests quicker than it does presently to keep up. GPU parallel processing will be possible with this. GPU computing allows massively parallel processing. Another future work option is to use Python to construct a blockchain from scratch; or we use another blockchain framework.

Further quantitative evaluation is necessary for more precise evaluation. For example, the number of code liens, execution time and other quantitative measurements should be done for further evaluation.

References

[1] S. Nakamoto, Bitcoin: *A Peer-to-Peer Electronic Cash System*. Technical report, `https://bitcoin.org/bitcoin.pdf`,(2008)

[2] F. Tschorsch, B. Scheuermann, "Bitcoin and beyond: a technical survey on decentralized digital currencies". *IEEE Comm. Surv. Tutor.*, **18**(3), pp.2084-2123 (2016)

[3] K. Christidis, M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things". *IEEE Access.* **4**, pp.2292-2303, (2016)

[4] G. Wood, Ethereum: "A Secure Decentralized Generalized Transaction Ledger." Technical report, Ethereum. `https://ethereum.github.io/yellowpaper/paper.pdf` (2017)

[5] Bashir, I. (2018) *Mastering Blockchain* (2nd revised)., Packt Publishing., Birmingham, UK. (2018)

[6] Purewal, S. (2014) *Learning Web App Development*, O'Reilly Media Inc., North Sebastopol.

[7] `https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/using-blockchain-to-improve-data-management-in-the-public-sector`

[8] André Henrique Mayer and al., Electronic health records in a Blockchain: A systematic review, *Health Informatics Journal*, **26**(2)pp.1273–1288, SAGE (2019).

[9] Mathis Steichen *et al.*, Blockchain-based, Decentralized Access control for IPFS. 2018 *IEEE International Conference on Blockchain.* pp.1499–1506, (2018)

[10] F. Knirsch, A. Unterweger, G. Eibl, D. Engel, in *Sustainable Cloud and Energy Services: Principe and Practices. Chap. 4*, ed. By W. Rivera. Privacy-Preserving Smart Grid Tariff Decisions with Blockchain-based Smart Contracts (Springer, Cham), pp. 85-116, (2017)

[11] G. Greenspan, "MultiChain Private Blockchain – White Paper". *Technical report*, Coin Sciences Ltd (2015). `http://www.multichain.com/download/Multichain-White-Paper.pdf`

[12] A. Unterweger, F. Knirsch, C. Leixnering, D. Engel, "Lessons learned from Implementing a Privacy-Preserving Smart Contract in Ethereum "in *2018*

9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). pp.1–5 (2018)

[13] J. Bucko, D. Palova, M. Vejacka, in Central European Conference In Finance And Economics. Security and Trust in Cryptocurrencies (Technical University of Kosice, Herlany,), pp. 14-24 (2015)

[14] A. Unterweger, F. Knirsch, C. Leixnering, D. Engel, Update: Lessons Learned from Implementing a Privacy-Preserving Smart Contract in Ethereum. Technical report, Center for Secure Energy Informatics, Salzburg University of Applied Sciences, Austria. `http://www.en-trust.at/papers/Unterweger18a-t.pdf` (2017)

[15] F. Knirsch, A. Unterweger, K. Karlsson, D. Engel, S. B. Wicker, Update: Evaluation of a Blockchain-based Proof-of-Possession Implementation. Technical report, Center for Secure Energy Informatics, Salzburg University of Applied Sciences, Austria. `http://www.en-trust.at/papers/Knirsch18a-t.pdf` (2018)

[16] `https://www.openchainproject.org/`

[17] M. Hearn, Corda: "A distributed ledger, Version 0.5". *Technical report. Corda.* `https://www.corda.net/content/corda-technical-whitepaper.pdf`

[18] A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts, in *2016 IEEE Symposium on Security and Privacy (SP)*. pp.839-858 (2016)

[19] `https://tendermint.com/`

[20] D. Maziéres, The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus. *Technical report*, Stellar Development Foundation. `https://www.stellar.org/papers/stellar-consensus-protocol.pdf` (2016)

[21] `https://eos.io/`

[22] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, B. Ford, OmniLedger: "A Secure, Scale-Out, Decentralized Ledger via Sharding". *IEEE Symp. Secur. Priv.*, pp.583-598 (2018)

[23] `https://neo.org/`

[24] PrimeTek., *PrimeFaces framework*, `https://www.primefaces.org/gettingstarted`

[25] Seyed, M.M.. and Tahaghoghi, W. H., *Learning MySql*, O'Relilly Media Inc., North Sebastopol. (2006)

[26] Geary, C. S. and Horstmann, D., *Core Java Server Faces*, Prentice Hall, Boston, MA. (2010)

[27] M. Di Pierro, What is the blockchain?, *Comput. Sci. Eng.* **19** (5) pp.92-95. (2017)

[28] D. Rachmawati, J. Tarigan, A. Ginting, A. Comparative study of message digest 5 (md5) and sha256 algorithm, *J. Phys.: Conf. Ser.* 978 012116 (2018)

[29] `https://www.multichain.com/download-install`

[30] Baset, S., Desrosiers, L., et al, *Hands-On Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer*, 2018, Packt Publishing.

[31] `https://github.com/hyperledger/fabric-gateway-java`

Appendix A

Implementation Source Programs

This appendix presents whole source code of our prototype system. The software structure (module relations) are shown in Figure A.1. From next section each files will be shown and short explanation will be give. Our project is divided in five modules:

- Model/Entity module: the properties that must be saved in the database are defined by model module.
- DAO module: will hold the interfaces required for accessing the persistence layer
- Manager Beans module: A contains the controller interfaces
- Blockchain Client module: it contains different code to get connected to the blockchain interface
- Web pages: all about the pages that users will use on a browser

A.1 Database connection between DMBS and Java

To be able to store objects in the database JPA in Java, we need to define entities classes. They are grouped in the module Model/Entity. In our case we have the

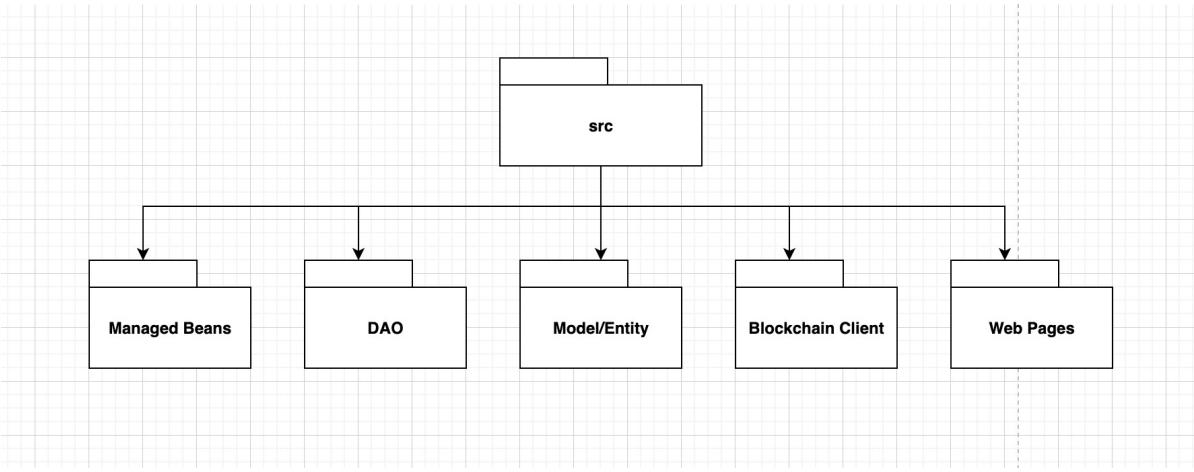

Figure A.1: Software structure

following entities codes are part of the Model/Entity module :

A.1.1 `TypeHall` entity class to store data in `TypaHall` table in database

This module creates table "`TypeHall`" in the database. It manages the operations of insert, update and delete.

```
1  import java.util.Collection;
2  import java.util.HashSet;
3  import java.util.Set;
4  import javax.persistence.CascadeType;
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.FetchType;
8  import javax.persistence.GeneratedValue;
9  import static javax.persistence.GenerationType.IDENTITY;
10 import javax.persistence.Id;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.OneToMany;
13 import javax.persistence.Table;
14 import javax.xml.bind.annotation.XmlTransient;
15 import org.hibernate.annotations.ResultCheckStyle;
16 import org.hibernate.annotations.SQLDelete;
17 import org.hibernate.annotations.Where;
18
19 @Entity
20 @Table(name="typehall"
21     ,catalog="birthDB"
22 )
23 @SQLDelete(sql = "UPDATE typehall SET state = 'DELETED' WHERE idtypehall
       = ?", check = ResultCheckStyle.COUNT)
24 @Where(clause = "state <> 'DELETED'")
25 @NamedQuery(name ="Typehall.findByName", query = "SELECT t FROM Typehall
       t WHERE like :typehallname")
26 public class Typehall implements java.io.Serializable {
27
28     private Collection<Hall> hallCollection;
29     private Integer idtypehall;
30     private String typehallname;
31     private Set halls = new HashSet(0);
32
33     public Typehall() {
34         this.idtypehall = 0;
35     }
36
37     public Typehall(String typehallname) {
38         this.typehallname = typehallname;
39     }
40
41     public Typehall(String typehallname, Set halls) {
42        this.typehallname = typehallname;
43        this.halls = halls;
44     }
45
46      @Id @GeneratedValue(strategy=IDENTITY)
47     @Column(name="idtypehall", unique=true, nullable=false)
48     public Integer getIdtypehall() {
49         return this.idtypehall;
50     }
51
52     public void setIdtypehall(Integer idtypehall) {
53         this.idtypehall = idtypehall;
54     }
55     @Column(name="typehallname", length=45)
56     public String getTypehallname() {
57         return this.typehallname;
58     }
59
60     public void setTypehallname(String typehallname) {
61         this.typehallname = typehallname;
```

```
62      }
63
64  @OneToMany(fetch=FetchType.LAZY, mappedBy="typehall")
65      public Set getHalls() {
66          return this.halls;
67      }
68
69      public void setHalls(Set halls) {
70          this.halls = halls;
71      }
72
73      @OneToMany(cascade = CascadeType.ALL, mappedBy = "idtypehall")
74      @XmlTransient
75      public Collection<Hall> getHallCollection() {
76          return hallCollection;
77      }
78
79      public void setHallCollection(Collection<Hall> hallCollection) {
80          this.hallCollection = hallCollection;
81      }
82  }
```

A.1.2 `Role` entity class to store data in `Role` table in database

This module creates table '`Role`" in the database. It manages the operations of insert, update and delete.

```
1   import java.util.Collection;
2   import java.util.HashSet;
3   import java.util.Set;
4   import javax.persistence.CascadeType;
5   import javax.persistence.Column;
6   import javax.persistence.Entity;
7   import javax.persistence.FetchType;
8   import javax.persistence.GeneratedValue;
9   import static javax.persistence.GenerationType.IDENTITY;
10  import javax.persistence.Id;
11  import javax.persistence.NamedQuery;
12  import javax.persistence.OneToMany;
13  import javax.persistence.Table;
14  import javax.xml.bind.annotation.XmlTransient;
15  import org.hibernate.annotations.DynamicInsert;
16  import org.hibernate.annotations.DynamicUpdate;
17  import org.hibernate.annotations.ResultCheckStyle;
18  import org.hibernate.annotations.SQLDelete;
19  import org.hibernate.annotations.Where;
20
21  @Entity
22  @Table(name="role"
23      ,catalog="birthDB"
24  )
25  @DynamicInsert @DynamicUpdate
26  @SQLDelete(sql = "UPDATE role SET state = 'DELETED' WHERE idrole = ?",
        check = ResultCheckStyle.COUNT)
27  @Where(clause = "state <> 'DELETED'")
28  @NamedQuery(name ="Role.findByName", query = "SELECT r FROM Role r WHERE
         like :title")
29  public class Role implements java.io.Serializable {
30
31      private Integer idrole;
32      private String title;
33      private String description;
34      private Boolean estate;
35      private Collection<User> userCollection;
36      private Set rolemenus = new HashSet(0);
37      private Set users = new HashSet(0);
38
39      public Role() {
40          this.idrole = 0;
```

```
41      }
42
43      public Role(String title) {
44          this.title = title;
45      }
46
47      public Role(String title, String description, Boolean estate) {
48         this.title = title;
49         this.description = description;
50         this.estate = estate;
51      }
52
53      public Role(String title, String description, Boolean estate, Set
            rolemenus, Set users) {
54         this.title = title;
55         this.description = description;
56         this.estate = estate;
57         this.rolemenus = rolemenus;
58         this.users = users;
59      }
60
61      @Id @GeneratedValue(strategy=IDENTITY)
62      @Column(name="idrole", unique=true, nullable=false)
63      public Integer getIdrole() {
64          return this.idrole;
65      }
66
67      public void setIdrole(Integer idrole) {
68          this.idrole = idrole;
69      }
70
71      @Column(name="title", length=45)
72      public String getTitle() {
73          return this.title;
74      }
75
76      public void setTitle(String title) {
77          this.title = title;
78      }
79
80      @Column(name="description", length=45)
81      public String getDescription() {
82          return this.description;
83      }
84
85      public void setDescription(String description) {
86          this.description = description;
87      }
88
89      @OneToMany(cascade=CascadeType.ALL, fetch=FetchType.LAZY, mappedBy="
            role")
90      public Set getRolemenus() {
91          return this.rolemenus;
92      }
93
94      public void setRolemenus(Set rolemenus) {
95          this.rolemenus = rolemenus;
96      }
97
98      @OneToMany(cascade=CascadeType.ALL, fetch=FetchType.LAZY, mappedBy="
            role")
99      public Set getUsers() {
100         return this.users;
101     }
102
103     public void setUsers(Set users) {
104         this.users = users;
105     }
106
107     @Column(name="estate")
108     public Boolean getEstate() {
```

```
109         return estate;
110     }
111 
112     public void setEstate(Boolean estate) {
113         this.estate = estate;
114     }
115 
116     @OneToMany(cascade = CascadeType.ALL, mappedBy = "idrole")
117     @XmlTransient
118     public Collection<User> getUserCollection() {
119         return userCollection;
120     }
121 
122     public void setUserCollection(Collection<User> userCollection) {
123         this.userCollection = userCollection;
124     }
125 }
```

A.1.3 `Hall` entity class to store data in `Hall` table in database

This creates `Hall` table in the database. It manages the operations of insert, update and delete.

```
1  import java.util.Collection;
2  import java.util.HashSet;
3  import java.util.Set;
4  import javax.persistence.CascadeType;
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.FetchType;
8  import javax.persistence.GeneratedValue;
9  import static javax.persistence.GenerationType.IDENTITY;
10 import javax.persistence.Id;
11 import javax.persistence.JoinColumn;
12 import javax.persistence.ManyToOne;
13 import javax.persistence.NamedQuery;
14 import javax.persistence.OneToMany;
15 import javax.persistence.Table;
16 import javax.xml.bind.annotation.XmlTransient;
17 import org.hibernate.annotations.ResultCheckStyle;
18 import org.hibernate.annotations.SQLDelete;
19 import org.hibernate.annotations.Where;
20 
21 @Entity
22 @Table(name="hall"
23     ,catalog="birthDB"
24 )
25 @SQLDelete(sql = "UPDATE hall SET state = 'DELETED' WHERE idhall = ?",
       check = ResultCheckStyle.COUNT)
26 @Where(clause = "state <> 'DELETED'")
27 @NamedQuery(name ="Hall.findByName", query = "SELECT h FROM Hall h WHERE
        like :hallname")
28 public class Hall implements java.io.Serializable {
29     private Collection<BirthInfo> birthInfoCollection;
30      private Integer idhall;
31      private Prefecture prefecture;
32      private Typehall typehall;
33      private String hallname;
34      private Set birthInfos = new HashSet(0);
35 
36     public Hall() {
37         this.idhall = 0;
38         this.typehall = new Typehall();
39         this.prefecture = new Prefecture();
40     }
41 
42 
43     public Hall(Prefecture prefecture, Typehall typehall) {
44         this.prefecture = prefecture;
```

```
45          this.typehall = typehall;
46      }
47
48      public Hall(Prefecture prefecture, Typehall typehall, String
            hallname, Set birthInfos) {
49         this.prefecture = prefecture;
50         this.typehall = typehall;
51         this.hallname = hallname;
52         this.birthInfos = birthInfos;
53      }
54
55      @Id @GeneratedValue(strategy=IDENTITY)
56      @Column(name="idhall", unique=true, nullable=false)
57      public Integer getIdhall() {
58          return this.idhall;
59      }
60
61      public void setIdhall(Integer idhall) {
62          this.idhall = idhall;
63      }
64
65  @ManyToOne(fetch=FetchType.LAZY)
66      @JoinColumn(name="idprefecture", nullable=false)
67      public Prefecture getPrefecture() {
68          return this.prefecture;
69      }
70
71      public void setPrefecture(Prefecture prefecture) {
72          this.prefecture = prefecture;
73      }
74
75  @ManyToOne(fetch=FetchType.LAZY)
76      @JoinColumn(name="idtypehall", nullable=false)
77      public Typehall getTypehall() {
78          return this.typehall;
79      }
80
81      public void setTypehall(Typehall typehall) {
82          this.typehall = typehall;
83      }
84      @Column(name="hallname", length=45)
85      public String getHallname() {
86          return this.hallname;
87      }
88
89      public void setHallname(String hallname) {
90          this.hallname = hallname;
91      }
92
93  @OneToMany(fetch=FetchType.LAZY, mappedBy="hall")
94      public Set getBirthInfos() {
95          return this.birthInfos;
96      }
97
98      public void setBirthInfos(Set birthInfos) {
99          this.birthInfos = birthInfos;
100     }
101
102 @OneToMany(cascade = CascadeType.ALL, mappedBy = "idhall")
103     @XmlTransient
104     public Collection<BirthInfo> getBirthInfoCollection() {
105         return birthInfoCollection;
106     }
107
108     public void setBirthInfoCollection(Collection<BirthInfo>
            birthInfoCollection) {
109         this.birthInfoCollection = birthInfoCollection;
110     }
111 }
```

A.1.4 `Prefecture` entity class to store data in `Prefecture` table in database

This module creates the table "`Prefecture`" in the database. It manages the operations of insert, update and delete.

```
1  import java.io.Serializable;
2  import java.util.Collection;
3  import java.util.HashSet;
4  import java.util.Set;
5  import javax.persistence.CascadeType;
6  import javax.persistence.Column;
7  import javax.persistence.Entity;
8  import javax.persistence.FetchType;
9  import javax.persistence.GeneratedValue;
10 import static javax.persistence.GenerationType.IDENTITY;
11 import javax.persistence.Id;
12 import javax.persistence.JoinColumn;
13 import javax.persistence.ManyToOne;
14 import javax.persistence.NamedQuery;
15 import javax.persistence.OneToMany;
16 import javax.persistence.Table;
17 import javax.xml.bind.annotation.XmlTransient;
18 import org.hibernate.annotations.ResultCheckStyle;
19 import org.hibernate.annotations.SQLDelete;
20 import org.hibernate.annotations.Where;
21
22 @Entity
23 @Table(name="prefecture"
24     ,catalog="birthDB"
25 )
26 @SQLDelete(sql = "UPDATE prefecture SET state = 'DELETED' WHERE
       idprefecture = ?", check = ResultCheckStyle.COUNT)
27 @Where(clause = "state <> 'DELETED'")
28 @NamedQuery(name ="Prefecture.findByName", query = "SELECT p FROM
       Prefecture p WHERE like :prefecturename")
29
30 public class Prefecture implements Serializable {
31
32     private Collection<Hall> hallCollection;
33     private Integer idprefecture;
34     private Region region;
35     private String prefecturename;
36     private Set halls = new HashSet(0);
37
38     public Prefecture() {
39         this.idprefecture = 0;
40         this.region = new Region();
41     }
42
43
44     public Prefecture(Region region) {
45         this.region = region;
46     }
47     public Prefecture(Region region, String prefecturename, Set halls) {
48        this.region = region;
49        this.prefecturename = prefecturename;
50        this.halls = halls;
51     }
52
53     @Id
54     @GeneratedValue(strategy=IDENTITY)
55     @Column(name="idprefecture", unique=true, nullable=false)
56     public Integer getIdprefecture() {
57         return this.idprefecture;
58     }
59
60     public void setIdprefecture(Integer idprefecture) {
61         this.idprefecture = idprefecture;
62     }
```

```
63
64      @ManyToOne(fetch=FetchType.LAZY)
65       @JoinColumn(name="idregion", nullable=false)
66      public Region getRegion() {
67          return this.region;
68      }
69
70      public void setRegion(Region region) {
71          this.region = region;
72      }
73
74      @Column(name="prefecturename", length=45)
75      public String getPrefecturename() {
76          return this.prefecturename;
77      }
78
79      public void setPrefecturename(String prefecturename) {
80          this.prefecturename = prefecturename;
81      }
82
83      @OneToMany(fetch=FetchType.LAZY, mappedBy="prefecture")
84      public Set getHalls() {
85          return this.halls;
86      }
87
88      public void setHalls(Set halls) {
89          this.halls = halls;
90      }
91
92      @OneToMany(cascade = CascadeType.ALL, mappedBy = "idprefecture")
93      @XmlTransient
94      public Collection<Hall> getHallCollection() {
95          return hallCollection;
96      }
97
98      public void setHallCollection(Collection<Hall> hallCollection) {
99          this.hallCollection = hallCollection;
100     }
101
102 }
```

A.1.5 `Region` entity class to store data in `Region` table in database

This module creates the table "`Region`" in the database. It manages the operations of insert, update and delete.

```
1  import java.util.Collection;
2  import java.util.HashSet;
3  import java.util.Set;
4  import javax.persistence.CascadeType;
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.FetchType;
8  import javax.persistence.GeneratedValue;
9  import static javax.persistence.GenerationType.IDENTITY;
10 import javax.persistence.Id;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.OneToMany;
13 import javax.persistence.Table;
14 import javax.xml.bind.annotation.XmlTransient;
15 import org.hibernate.annotations.ResultCheckStyle;
16 import org.hibernate.annotations.SQLDelete;
17 import org.hibernate.annotations.Where;
18
19 @Entity
20 @Table(name="region"
21     ,catalog="birthDB"
22 )
```

```
23 @SQLDelete(sql = "UPDATE region SET state = 'DELETED' WHERE idregion =
       ?", check = ResultCheckStyle.COUNT)
24 @Where(clause = "state <> 'DELETED'")
25 @NamedQuery(name ="Region.findByName", query = "SELECT r FROM Region r
       WHERE like :regionname")
26 public class Region implements java.io.Serializable {
27
28     private Collection<Prefecture> prefectureCollection;
29     private Integer idregion;
30     private String regionname;
31     private Set prefectures = new HashSet(0);
32
33     public Region() {
34         this.idregion = 0;
35     }
36
37     public Region(String regionname) {
38         this.regionname = regionname;
39     }
40
41     public Region(String regionname, Set prefectures) {
42        this.regionname = regionname;
43        this.prefectures = prefectures;
44     }
45
46     @Id
47     @GeneratedValue(strategy=IDENTITY)
48     @Column(name="idregion", unique=true, nullable=false)
49     public Integer getIdregion() {
50         return this.idregion;
51     }
52
53     public void setIdregion(Integer idregion) {
54         this.idregion = idregion;
55     }
56
57     @Column(name="regionname", length=45)
58     public String getRegionname() {
59         return this.regionname;
60     }
61
62     public void setRegionname(String regionname) {
63         this.regionname = regionname;
64     }
65
66     @OneToMany(fetch=FetchType.LAZY, mappedBy="region")
67     public Set getPrefectures() {
68         return this.prefectures;
69     }
70
71     public void setPrefectures(Set prefectures) {
72         this.prefectures = prefectures;
73     }
74
75     @OneToMany(cascade = CascadeType.ALL, mappedBy = "idregion")
76     @XmlTransient
77     public Collection<Prefecture> getPrefectureCollection() {
78         return prefectureCollection;
79     }
80
81     public void setPrefectureCollection(Collection<Prefecture>
           prefectureCollection) {
82         this.prefectureCollection = prefectureCollection;
83     }
84
85 }
```

A.1.6 `BirthInfo` entity class to store data in `BirthInfo` table in database

This module creates the table "`BirthInfo`" in the database. It manages the operations of insert, update and delete.

```
1  import java.util.Date;
2  import java.util.UUID;
3  import javax.persistence.Column;
4  import javax.persistence.Entity;
5  import javax.persistence.FetchType;
6  import javax.persistence.GeneratedValue;
7  import static javax.persistence.GenerationType.IDENTITY;
8  import javax.persistence.Id;
9  import javax.persistence.JoinColumn;
10 import javax.persistence.ManyToOne;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.Table;
13 import javax.persistence.Temporal;
14 import javax.persistence.TemporalType;
15 import org.hibernate.annotations.ResultCheckStyle;
16 import org.hibernate.annotations.SQLDelete;
17 import org.hibernate.annotations.Where;
18
19 @Entity
20 @Table(name="birthInfo"
21     ,catalog="birthDB"
22 )
23 @SQLDelete(sql = "UPDATE birthInfo SET state = 'DELETED' WHERE
       birthinfoid = ?", check = ResultCheckStyle.COUNT)
24 @Where(clause = "state <> 'DELETED'")
25 @NamedQuery(name ="BirthInfo.findByName", query = "SELECT b FROM
       BirthInfo b WHERE like :birthplace")
26 public class BirthInfo implements java.io.Serializable {
27
28     private int idbirthinfo;
29     private String reference;
30     private Hall hall;
31     private String childfirstlastname;
32     private String gender;
33     private Date birthdate;
34     private String birthplace;
35     private String fatherfirstlastname;
36     private String fatheroccupation;
37     private Integer fatherage;
38     private String motherfirstlastname;
39     private String motheroccupation;
40     private Integer motherage;
41     private Date registrationdate;
42
43     public BirthInfo() {
44         this.idbirthinfo = 0;
45         this.hall = new Hall();
46     }
47
48     public BirthInfo( Hall hall) {
49         this.hall = hall;
50     }
51
52     public BirthInfo( String childfirstlastname, String gender, Date
           birthdate, String birthplace, String fatherfisrtlastname, String
            fatheroccupation, Integer fatherage, String motherfisrtlastname
           , String motheroccupation, Integer motherage, Hall hall, Date
           registrationdate, String reference) {
53         this.childfirstlastname = childfirstlastname;
54         this.gender = gender;
55         this.birthdate = birthdate;
56         this.birthplace = birthplace;
57         this.fatherfirstlastname = fatherfisrtlastname;
58         this.fatheroccupation = fatheroccupation;
59         this.fatherage = fatherage;
60         this.motherfirstlastname = motherfisrtlastname;
```

```
61          this.motheroccupation = motheroccupation;
62          this.motherage = motherage;
63          this.hall = hall;
64          this.registrationdate = registrationdate;
65          this.reference = reference;
66      }
67
68
69      @Id
70      @GeneratedValue(strategy=IDENTITY)
71      @Column(name="idbirthinfo", nullable=false)
72      public int getIdbirthinfo() {
73          return this.idbirthinfo;
74      }
75
76      public void setIdbirthinfo(int idbirthinfo) {
77          this.idbirthinfo = idbirthinfo;
78      }
79
80      @Column(name="reference", nullable=false, length=45)
81      public String getReference() {
82          return this.reference;
83      }
84
85      public void setReference(String reference) {
86          this.reference = reference;
87      }
88
89  @ManyToOne(fetch=FetchType.LAZY)
90      @JoinColumn(name="idhall", nullable=false)
91      public Hall getHall() {
92          return this.hall;
93      }
94
95      public void setHall(Hall hall) {
96          this.hall = hall;
97      }
98
99
100     @Column(name="childfirstlastname", nullable=false)
101     public String getChildfirstlastname() {
102         return childfirstlastname;
103     }
104
105     public void setChildfirstlastname(String childfirstlastname) {
106         this.childfirstlastname = childfirstlastname;
107     }
108
109     @Column(name="gender", nullable=false)
110     public String getGender() {
111         return gender;
112     }
113
114     public void setGender(String gender) {
115         this.gender = gender;
116     }
117
118
119     @Column(name="birthdate", length=10)
120     @Temporal(TemporalType.DATE)
121     public Date getBirthdate() {
122         return this.birthdate;
123     }
124
125     public void setBirthdate(Date birthdate) {
126         this.birthdate = birthdate;
127     }
128
129     @Column(name="birthplace", length=45)
130     public String getBirthplace() {
131         return this.birthplace;
```

```
132     }
133
134     public void setBirthplace(String birthplace) {
135         this.birthplace = birthplace;
136     }
137
138
139     @Column(name="fatherfirstlastname", nullable=false)
140     public String getFatherfirstlastname() {
141         return fatherfirstlastname;
142     }
143
144     public void setFatherfirstlastname(String fatherfirstlastname) {
145         this.fatherfirstlastname = fatherfirstlastname;
146     }
147
148
149     @Column(name="fatherage", nullable=false)
150     public Integer getFatherage() {
151         return fatherage;
152     }
153
154     public void setFatherage(Integer fatherage) {
155         this.fatherage = fatherage;
156     }
157
158
159     @Column(name="fatheroccupation", nullable=false)
160     public String getFatheroccupation() {
161         return fatheroccupation;
162     }
163
164     public void setFatheroccupation(String fatheroccupation) {
165         this.fatheroccupation = fatheroccupation;
166     }
167
168
169     @Column(name="motherfirstlastname", nullable=false)
170     public String getMotherfirstlastname() {
171         return motherfirstlastname;
172     }
173
174     public void setMotherfirstlastname(String motherfirstlastname) {
175         this.motherfirstlastname = motherfirstlastname;
176     }
177
178
179     @Column(name="motherage", nullable=false)
180     public Integer getMotherage() {
181         return motherage;
182     }
183
184     public void setMotherage(Integer motherage) {
185         this.motherage = motherage;
186     }
187
188     @Column(name="motheroccupation", nullable=false)
189     public String getMotheroccupation() {
190         return motheroccupation;
191     }
192
193     public void setMotheroccupation(String motheroccupation) {
194         this.motheroccupation = motheroccupation;
195     }
196
197
198     @Column(name="registrationdate", length=10)
199     @Temporal(TemporalType.TIMESTAMP)
200     public Date getRegistrationdate() {
201         return this.registrationdate;
202     }
```

```
203
204     public void setRegistrationdate(Date registrationdate) {
205         this.registrationdate = registrationdate;
206     }
207
208
209     public String getRefNumber(){
210         String rand = UUID.randomUUID().toString();
211         return reference = "TG"+rand.toUpperCase()+"TG";
212     }
213
214     @Override
215     public String toString()
216     {
217         return "Birth" +
218             "\n\t Reference: " + this.reference +
219             "\n\t Child Name: " + this.childfirstlastname +
220             "\n\t Gender: " + this.gender +
221             "\n\t Place of Birth: " + this.birthplace +
222             "\n\t Date of Birth: " + this.birthdate +
223             "\n\t Date of registration: " + this.registrationdate;
224     }
225
226 }
```

A.1.7 `User` entity class to store data in `User` table in database

This module creates table "`User`" in the database. It manages the operations of insert, update and delete.

```
 1 import java.io.Serializable;
 2 import java.util.Date;
 3 import javax.persistence.Column;
 4 import javax.persistence.Entity;
 5 import javax.persistence.FetchType;
 6 import javax.persistence.GeneratedValue;
 7 import static javax.persistence.GenerationType.IDENTITY;
 8 import javax.persistence.Id;
 9 import javax.persistence.JoinColumn;
10 import javax.persistence.ManyToOne;
11 import javax.persistence.NamedQuery;
12 import javax.persistence.Table;
13 import javax.persistence.Temporal;
14 import javax.persistence.TemporalType;
15 import org.hibernate.annotations.DynamicInsert;
16 import org.hibernate.annotations.DynamicUpdate;
17 import org.hibernate.annotations.ResultCheckStyle;
18 import org.hibernate.annotations.SQLDelete;
19 import org.hibernate.annotations.Where;
20
21 @Entity
22 @Table(name="user"
23     ,catalog="birthDB"
24 )
25 @DynamicInsert @DynamicUpdate
26 @SQLDelete(sql = "UPDATE typehall SET state = 'DELETED' WHERE user = ?",
       check = ResultCheckStyle.COUNT)
27 @Where(clause = "state <> 'DELETED'")
28 @NamedQuery(name ="User.findByName", query = "SELECT u FROM User u WHERE
       like :username")
29
30 public class User implements Serializable {
31
32     private Integer iduser;
33     private Role role;
34     private String username;
35     private String password;
36     private String firstname;
37     private String lastname;
```

```
38      private String email;
39      private Boolean estate;
40      private Date datecreation;
41
42      public User() {
43          this.iduser = 0;
44          this.role = new Role();
45      }
46
47      public User(Role role){
48          this.role = role;
49      }
50
51      public User(Role role, String username, String password, String
            firstname, String lastname, String email, Boolean estate, Date
            datecreation) {
52         this.role = role;
53         this.username = username;
54         this.password = password;
55         this.firstname = firstname;
56         this.lastname = lastname;
57         this.email = email;
58         this.estate = estate;
59         this.datecreation = datecreation;
60      }
61
62      @Id @GeneratedValue(strategy=IDENTITY)
63      @Column(name="iduser", unique=true, nullable=false)
64      public Integer getIduser() {
65          return this.iduser;
66      }
67
68      public void setIduser(Integer iduser) {
69          this.iduser = iduser;
70      }
71
72  @ManyToOne(fetch=FetchType.LAZY)
73      @JoinColumn(name="idrole", nullable=false)
74      public Role getRole() {
75          return this.role;
76      }
77
78      public void setRole(Role role) {
79          this.role = role;
80      }
81
82
83      @Column(name="username", length=45)
84      public String getUsername() {
85          return this.username;
86      }
87
88      public void setUsername(String username) {
89          this.username = username;
90      }
91
92
93      @Column(name="password", length=45)
94      public String getPassword() {
95          return this.password;
96      }
97
98      public void setPassword(String password) {
99          this.password = password;
100     }
101
102
103     @Column(name="firstname", length=45)
104     public String getFirstname() {
105         return this.firstname;
106     }
```

```
107
108     public void setFirstname(String firstname) {
109         this.firstname = firstname;
110     }
111
112
113     @Column(name="lastname", length=45)
114     public String getLastname() {
115         return this.lastname;
116     }
117
118     public void setLastname(String lastname) {
119         this.lastname = lastname;
120     }
121
122
123     @Column(name="email", length=45)
124     public String getEmail() {
125         return this.email;
126     }
127
128     public void setEmail(String email) {
129         this.email = email;
130     }
131
132
133     @Column(name="estate")
134     public Boolean getEstate() {
135         return this.estate;
136     }
137
138     public void setEstate(Boolean estate) {
139         this.estate = estate;
140     }
141
142     @Temporal(TemporalType.DATE)
143     @Column(name="datecreation", nullable=false, length=10)
144     public Date getDatecreation() {
145         return this.datecreation;
146     }
147
148     public void setDatecreation(Date datecreation) {
149         this.datecreation = datecreation;
150     }
151 }
```

A.2 Hibernate connection

A.2.1 Persistence file to connect to Hibernate

This module permit the application to be connected to a relational database management system MySQL and to persist and retrieve information. It can be found in DAO (Data Access Object) module.

```
1  import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
2  import org.hibernate.cfg.Configuration;
3  import org.hibernate.service.ServiceRegistry;
4  public class HibernateUtil {
5      private static SessionFactory sessionFactory;
6      public static SessionFactory getSessionFactory(){
7          if(sessionFactory == null){
8              Configuration configuration = new Configuration().configure()
                   ;
9              ServiceRegistry serviceRegistry = new
                   StandardServiceRegistryBuilder().applySettings(
                   configuration.getProperties()).build();
10   sessionFactory= configuration.buildSessionFactory(serviceRegistry);
```

```
11          }
12          return sessionFactory;
13      }
14  }
```

A.2.2 Hibernate configuration code

Following is a Hibernate configuration code.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate␣
       Configuration␣DTD␣3.0//EN" "http://hibernate.sourceforge.net/
       hibernate-configuration-3.0.dtd">
3
4  <hibernate-configuration>
5    <session-factory>
6      <property name="hibernate.dialect">org.hibernate.dialect.
           MySQLDialect</property>
7      <property name="hibernate.connection.driver_class">com.mysql.jdbc.
           Driver</property>
8      <property name="hibernate.connection.url">jdbc:mysql://
           localhost:8889/birthDB</property>
9      <property name="hibernate.connection.username">root</property>
10     <property name="hibernate.connection.password">root</property>
11     <property name="hibernate.show_sql">true</property>
12     <property name="hibernate.current_session_context_class">thread</
           property>
13     <mapping resource="model/Hall.hbm.xml"/>
14     <mapping resource="model/Menu.hbm.xml"/>
15     <mapping resource="model/Prefecture.hbm.xml"/>
16     <mapping resource="model/User.hbm.xml"/>
17     <mapping resource="model/Region.hbm.xml"/>
18     <mapping resource="model/Rolemenu.hbm.xml"/>
19     <mapping resource="model/Role.hbm.xml"/>
20     <mapping resource="model/BirthInfo.hbm.xml"/>
21     <mapping resource="model/Typehall.hbm.xml"/>
22   </session-factory>
23 </hibernate-configuration>
```

It is permit to use data access pattern (DAO) in Java to separate low level data accessing. We define data access interface where we define the standard operations to be `performe3d` on a model objet and a concrete data access class which implements above interface. We have the following codes represent the data access codes of our application.

A.3 Data Access Objects

It is permit to use data access pattern (DAO) in Java to separate low level data accessing. We define data access interface where we define the standard operations to be performed on a model objet and a concrete data access class which implements above interface. We have the following codes represent the data access codes of our application and they can be found in the DAO module.

A.3.1 BirthInfo DAO

This module is to access data from the table "`BirthInfo`" in the relational database management system MySQL.

```
1  import java.util.List;
2  import model.BirthInfo;
3
```

```
4  public interface birthinfoDAO {
5
6      public List<BirthInfo> findAll();
7      public List<BirthInfo> SearchByReferenceNumber(String reference);
8      public boolean create(BirthInfo birthinfo);
9      public boolean update(BirthInfo birthinfo);
10     public boolean delete(Integer id);
11
12 }
13
14 import java.util.List;
15 import model.BirthInfo;
16 import org.hibernate.Query;
17 import org.hibernate.Session;
18 import org.hibernate.SessionFactory;
19 import util.HibernateUtil;
20
21 public class birthinfoDaoImpl implements birthinfoDAO{
22
23     public List<BirthInfo> findAll() {
24         List<BirthInfo> listbirthinfo = null;
25         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
26         Session session = sessionFactory.openSession();
27         String sql = "FROM BirthInfo";
28         try {
29             session.getTransaction().begin();
30             listbirthinfo = session.createQuery(sql).list();
31             session.getTransaction().commit();
32         } catch (Exception e) {
33             session.beginTransaction().rollback();
34             session.close();
35         }
36         return listbirthinfo;
37     }
38
39     public List<BirthInfo> SearchByReferenceNumber(String reference) {
40         List<BirthInfo> researchList = null;
41         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
42         Session session = sessionFactory.openSession();
43         String sql = "FROM BirthInfo B where B.reference =:reference";
44         try {
45             session.getTransaction().begin();
46             Query qu = session.createQuery(sql);
47             qu.setParameter("Reference", reference);
48             researchList = qu.list();
49             int count = researchList.size();
50             session.getTransaction().commit();
51         } catch (Exception e) {
52             session.beginTransaction().rollback();
53             session.close();
54         }
55         return researchList;
56     }
57
58     @Override
59     public boolean create(BirthInfo birthinfo) {
60         boolean test;
61         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
62         Session session = sessionFactory.openSession();
63         try {
64             session.getTransaction().begin();
65             session.save(birthinfo);
66             session.flush();
67             session.getTransaction().commit();
68             test = true;
69         } catch (Exception e) {
70             test = false;
71             session.getTransaction().rollback();
```

```
72              session.close();
73          }
74          return test;
75      }
76
77      @Override
78      public boolean update(BirthInfo birthinfo) {
79          boolean test;
80          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
81          Session session = sessionFactory.openSession();
82          try {
83              session.getTransaction().begin();
84              session.merge(birthinfo);
85              session.flush();
86              session.getTransaction().commit();
87              test = true;
88          } catch (Exception e) {
89              test = false;
90              session.beginTransaction().rollback();
91              session.close();
92          }
93          return test;
94      }
95
96      @Override
97      public boolean delete(Integer id) {
98          boolean test;
99          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
100         Session session = sessionFactory.openSession();
101         try {
102             session.getTransaction().begin();
103             BirthInfo birthinfo = (BirthInfo) session.load(BirthInfo.
                    class, id);
104             session.delete(birthinfo);
105             session.getTransaction().commit();
106             test = true;
107         } catch (Exception e) {
108             test = false;
109             session.beginTransaction().rollback();
110             session.close();
111         }
112         return test;
113     }
114
115 }
```

A.3.2 TypeHall DAO

This module is to access data from the table "`TypeHall`" in the relational database management system MySQL.

```
1  import java.util.List;
2  import model.Typehall;
3
4  public interface typehallDAO {
5
6      public List<Typehall> findAll();
7      public List<Typehall> selectItems();
8      public boolean create(Typehall typehall);
9      public boolean update(Typehall typehall);
10     public boolean delete(Integer id);
11 }
12
13 import java.util.List;
14 import model.Region;
15 import model.Typehall;
16 import org.hibernate.Session;
```

```
17 import org.hibernate.SessionFactory;
18 import util.HibernateUtil;
19
20 public class typehallDaoImpl implements typehallDAO{
21
22     @Override
23     public List<Typehall> findAll() {
24         List<Typehall> listTypehall = null;
25         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
26         Session session = sessionFactory.openSession();
27         String sql = "FROM Typehall";
28         try {
29             session.getTransaction().begin();
30             listTypehall = session.createQuery(sql).list();
31             session.getTransaction().commit();
32         } catch (Exception e) {
33             session.getTransaction().rollback();
34             session.close();
35         }
36         return listTypehall;
37     }
38
39     @Override
40     public List<Typehall> selectItems() {
41         List<Typehall> listtypehall = null;
42         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
43         Session session = sessionFactory.openSession();
44         String sql = "FROM Typehall";
45         try {
46             session.getTransaction().begin();
47             listtypehall = session.createQuery(sql).list();
48             session.getTransaction().commit();
49         } catch (RuntimeException e) {
50             session.getTransaction().rollback();
51             session.close();
52             throw e;
53         }
54         return listtypehall;
55     }
56
57     @Override
58     public boolean create(Typehall typehall) {
59         boolean test;
60         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
61         Session session = sessionFactory.openSession();
62         try {
63             session.getTransaction().begin();
64             session.save(typehall);
65             session.flush();
66             session.clear();
67             session.getTransaction().commit();
68             test = true;
69         } catch (Exception e) {
70             test = false;
71             session.getTransaction().rollback();
72             session.close();
73         }
74         return test;
75     }
76
77     @Override
78     public boolean update(Typehall typehall) {
79         boolean test;
80         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
81         Session session = sessionFactory.openSession();
82         try {
83             session.getTransaction().begin();
```

```
84              session.merge(typehall);
85              session.flush();
86              session.clear();
87              session.getTransaction().commit();
88              test = true;
89          } catch (Exception e) {
90              test = false;
91              session.getTransaction().rollback();
92              session.close();
93          }
94          return test;
95      }
96
97      @Override
98      public boolean delete(Integer id) {
99          boolean test;
100         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
101         Session session = sessionFactory.openSession();
102         try {
103             session.getTransaction().begin();
104             Region region = (Region) session.load(Region.class, id);
105             session.delete(region);
106             session.flush();
107             session.clear();
108             session.getTransaction().commit();
109             test = true;
110         } catch (Exception e) {
111             test = false;
112             session.getTransaction().rollback();
113             session.close();
114         }
115         return test;
116     }
117 }
```

A.3.3 Hall DAO

This module is to access data from the table "`Hall`" in the relational database management system MySQL.

```
1  import java.util.List;
2  import model.Hall;
3  public interface hallDAO {
4
5      public List<Hall> findAll();
6      public List<Hall> selectItems();
7      public boolean create(Hall hall);
8      public boolean update(Hall hall);
9      public boolean delete(Integer id);
10 }
11
12 import java.util.List;
13 import model.Hall;
14 import org.hibernate.Session;
15 import org.hibernate.SessionFactory;
16 import util.HibernateUtil;
17
18 public class hallDaoImpl implements hallDAO{
19
20     @Override
21     public List<Hall> findAll() {
22         List<Hall> listhall = null;
23         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
24         Session session = sessionFactory.openSession();
25         String sql = "FROM Hall";
26         try {
27             session.getTransaction().begin();
```

```
28              listhall = session.createQuery(sql).list();
29              session.getTransaction().commit();
30          } catch (Exception e) {
31              session.getTransaction().rollback();
32              session.close();
33          }
34          return listhall;
35      }
36
37      @Override
38      public List<Hall> selectItems() {
39          List<Hall> listhall = null;
40          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
41          Session session = sessionFactory.openSession();
42          String sql = "FROM Hall";
43          try {
44              session.getTransaction().begin();
45              listhall = session.createQuery(sql).list();
46              session.getTransaction().commit();
47          } catch (RuntimeException e) {
48              session.getTransaction().rollback();
49              session.close();
50              throw e;
51          }
52          return listhall;
53      }
54
55      @Override
56      public boolean create(Hall hall) {
57          boolean test;
58          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
59          Session session = sessionFactory.openSession();
60          try {
61              session.getTransaction().begin();
62              session.save(hall);
63              session.flush();
64              session.clear();
65              session.getTransaction().commit();
66              test = true;
67          } catch (Exception e) {
68              test = false;
69              session.getTransaction().rollback();
70              session.close();
71          }
72          return test;
73      }
74
75      @Override
76      public boolean update(Hall hall) {
77          boolean test;
78          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
79          Session session = sessionFactory.openSession();
80          try {
81              session.getTransaction().begin();
82              session.merge(hall);
83              session.flush();
84              session.clear();
85              session.getTransaction().commit();
86              test = true;
87          } catch (Exception e) {
88              test = false;
89              session.getTransaction().rollback();
90              session.close();
91          }
92          return test;
93      }
94
95      @Override
```

```
96      public boolean delete(Integer id) {
97          boolean test;
98          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
99          Session session = sessionFactory.openSession();
100         try {
101             session.getTransaction().begin();
102             Hall hall = (Hall) session.load(Hall.class, id);
103             session.delete(hall);
104             session.flush();
105             session.clear();
106             session.getTransaction().commit();
107             test = true;
108         } catch (Exception e) {
109             test = false;
110             session.getTransaction().rollback();
111             session.close();
112         }
113         return test;
114     }
115 }
```

A.3.4 Prefecture DAO

This module is to access data from the table "`Prefecture`" in the relational database management system MySQL.

```
1  import java.util.List;
2  import model.Prefecture;
3
4  public interface prefectureDAO {
5
6      public List<Prefecture> findAll();
7      public List<Prefecture> selectItems();
8      public boolean create(Prefecture prefecture);
9      public boolean update(Prefecture prefecture);
10     public boolean delete(Integer id);
11 }
12
13 import java.util.List;
14 import model.Prefecture;
15 import org.hibernate.Session;
16 import org.hibernate.SessionFactory;
17 import util.HibernateUtil;
18
19 public class prefectureDaoImpl implements prefectureDAO{
20
21     @Override
22     public List<Prefecture> findAll() {
23         List<Prefecture> listprefecture = null;
24         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
25         Session session = sessionFactory.openSession();
26         String sql = "FROM Prefecture";
27         try {
28             session.getTransaction().begin();
29             listprefecture = session.createQuery(sql).list();
30             session.getTransaction().commit();
31         } catch (Exception e) {
32             session.getTransaction().rollback();
33             session.close();
34         }
35         return listprefecture;
36     }
37
38     @Override
39     public List<Prefecture> selectItems() {
40         List<Prefecture> listprefectures = null;
```

```
41          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
42          Session session = sessionFactory.openSession();
43          String sql = "FROM Prefecture";
44          try {
45              session.getTransaction().begin();
46              listprefectures = session.createQuery(sql).list();
47              session.getTransaction().commit();
48          } catch (RuntimeException e) {
49              session.getTransaction().rollback();
50              session.close();
51              throw e;
52          }
53          return listprefectures;
54      }
55
56      @Override
57      public boolean create(Prefecture prefecture) {
58          boolean test;
59          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
60          Session session = sessionFactory.openSession();
61          try {
62              session.getTransaction().begin();
63              session.save(prefecture);
64              session.flush();
65              session.clear();
66              session.getTransaction().commit();
67              test = true;
68          } catch (Exception e) {
69              test = false;
70              session.getTransaction().rollback();
71              session.close();
72              throw e;
73          }
74          return test;
75      }
76
77      @Override
78      public boolean update(Prefecture prefecture) {
79          boolean test;
80          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
81          Session session = sessionFactory.openSession();
82          try {
83              session.getTransaction().begin();
84              session.merge(prefecture);
85              session.flush();
86              session.clear();
87              session.getTransaction().commit();
88              test = true;
89          } catch (Exception e) {
90              test = false;
91              session.getTransaction().rollback();
92              session.close();
93          }
94          return test;
95      }
96
97      @Override
98      public boolean delete(Integer id) {
99          boolean test;
100         Session session = HibernateUtil.getSessionFactory().
                getCurrentSession();
101         try {
102             session.getTransaction().begin();
103             Prefecture prefecture = (Prefecture) session.load(Prefecture.
                    class, id);
104             session.delete(prefecture);
105             session.flush();
106             session.clear();
```

```
107             session.getTransaction().commit();
108             test = true;
109         } catch (Exception e) {
110             test = false;
111             session.getTransaction().rollback();
112             session.close();
113         }
114         return test;
115     }
116 }
```

A.3.5 Region DAO

This module is to access data from the table "`Region`" in the relational database management system MySQL.

```
1  import java.util.List;
2  import model.Region;
3
4  public interface regionDAO {
5
6      public List<Region> findAll();
7      public List<Region> selectItems();
8      public boolean create(Region region);
9      public boolean update(Region region);
10     public boolean delete(Integer id);
11 }
12
13 package dao;
14
15 import java.util.List;
16 import model.Region;
17 import org.hibernate.Session;
18 import org.hibernate.SessionFactory;
19 import util.HibernateUtil;
20
21 public class regionDaoImpl implements regionDAO{
22
23     @Override
24     public List<Region> findAll() {
25         List<Region> listRegion = null;
26         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
27         Session session = sessionFactory.openSession();
28         String sql = "FROM Region";
29         try {
30             session.getTransaction().begin();
31             listRegion = session.createQuery(sql).list();
32             session.getTransaction().commit();
33         } catch (Exception e) {
34             session.getTransaction().rollback();
35             session.close();
36         }
37         return listRegion;
38     }
39
40     @Override
41     public List<Region> selectItems() {
42         List<Region> listregions = null;
43         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
44         Session session = sessionFactory.openSession();
45         String sql = "FROM Region";
46         try {
47             session.getTransaction().begin();
48             listregions = session.createQuery(sql).list();
49             session.getTransaction().commit();
50         } catch (RuntimeException e) {
51             session.getTransaction().rollback();
```

```
52              session.close();
53              throw e;
54          }
55          return listregions;
56      }
57
58      @Override
59      public boolean create(Region region) {
60          boolean test;
61          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
62          Session session = sessionFactory.openSession();
63          try {
64              session.getTransaction().begin();
65              session.save(region);
66              session.flush();
67              session.getTransaction().commit();
68              test = true;
69          } catch (Exception e) {
70              test = false;
71              session.getTransaction().rollback();
72              session.close();
73          }
74          return test;
75      }
76
77      @Override
78      public boolean update(Region region) {
79          boolean test;
80          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
81          Session session = sessionFactory.openSession();
82          try {
83              session.getTransaction().begin();
84              session.merge(region);
85              session.flush();
86              session.getTransaction().commit();
87              test = true;
88          } catch (Exception e) {
89              test = false;
90              session.getTransaction().rollback();
91              session.close();
92          }
93          return test;
94      }
95
96      @Override
97      public boolean delete(Integer id) {
98          boolean test;
99          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
100         Session session = sessionFactory.openSession();
101         try {
102             session.getTransaction().begin();
103             Region region = (Region) session.load(Region.class, id);
104             session.delete(region);
105             session.getTransaction().commit();
106             test = true;
107         } catch (Exception e) {
108             test = false;
109             session.getTransaction().rollback();
110         }
111         return test;
112     }
113 }
```

A.3.6 User DAO

This module is to access data from the table "`User`" in the relational database management system MySQL.

```
1  import java.util.List;
2  import model.User;
3  public interface userDAO {
4      public User findByUser(User user);
5      public User login(User user);
6      public List<User> findAll();
7      public boolean create(User user);
8      public boolean update(User user);
9      public boolean delete(Integer id);
10
11 }
12
13
14 import java.util.List;
15 import model.User;
16
17 import org.hibernate.Session;
18 import org.hibernate.SessionFactory;
19 import util.HibernateUtil;
20
21 /**
22  *
23  * @author islabkc-319
24  */
25 public class userDaoImpl implements userDAO{
26
27     private String username;
28     private String password;
29
30     public String getUsername() {
31         return username;
32     }
33
34     public void setUsername(String username) {
35         this.username = username;
36     }
37
38     public String getPassword() {
39         return password;
40     }
41
42     public void setPassword(String password) {
43         this.password = password;
44     }
45
46     @Override
47     public User findByUser(User user) {
48         User usermod = null;
49         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
               ;
50         Session session = sessionFactory.openSession();
51         String sql = "FROM User WHERE username = '"+user.getUsername()
               +"'";
52         try {
53             session.getTransaction().begin();
54             usermod = (User) session.createQuery(sql).uniqueResult();
55             session.getTransaction().commit();
56         } catch (RuntimeException e) {
57             session.getTransaction().rollback();
58             throw e;
59         }finally {
60             if (session.isOpen()){
61                 session.close();
62             }
63         }
64         return usermod;
```

```
65      }
66
67      @Override
68      public User login(User user) {
69          User user1 = this.findByUser(user);
70          if (user1 != null) {
71              if (!user.getPassword().equals(user1.getPassword())) {
72                  user1 = null;
73              }
74          }
75          return user1;
76      }
77
78      @Override
79      public List<User> findAll() {
80          List<User> listuser = null;
81          SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
82          Session session = sessionFactory.openSession();
83          String sql = "FROM User u left join fetch u.role";
84          try {
85              session.getTransaction().begin();
86              listuser = session.createQuery(sql).list();
87              session.getTransaction().commit();
88          } catch (RuntimeException e) {
89              session.getTransaction().rollback();
90              throw e;
91          }finally {
92              if (session.isOpen()){
93                  session.close();
94              }
95          }
96          return listuser;
97      }
98
99
100     @Override
101     public boolean create(User user) {
102         boolean test;
103         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
104         Session session = sessionFactory.openSession();
105         try {
106             session.getTransaction().begin();
107             session.save(user);
108             session.flush();
109             session.clear();
110             session.getTransaction().commit();
111             test = true;
112         } catch (RuntimeException e) {
113             test = false;
114             session.getTransaction().rollback();
115             throw e;
116         }finally {
117             if (session.isOpen()){
118                 session.close();
119             }
120         }
121         return test;
122     }
123
124     @Override
125     public boolean update(User user) {
126         boolean test;
127         SessionFactory sessionFactory = HibernateUtil.getSessionFactory()
                ;
128         Session session = sessionFactory.openSession();
129         try {
130             session.getTransaction().begin();
131             session.merge(user);
132             session.flush();
```

```
133             session.clear();
134             session.getTransaction().commit();
135             test = true;
136         } catch (RuntimeException e) {
137             test = false;
138             session.getTransaction().rollback();
139             throw e;
140         }finally {
141             if (session.isOpen()){
142                 session.close();
143             }
144         }
145         return test;
146     }
147 
148     @Override
149     public boolean delete(Integer id) {
150         boolean test;
151         Session session = HibernateUtil.getSessionFactory().
                getCurrentSession();
152         try {
153             session.getTransaction().begin();
154             session.delete(id);
155             session.flush();
156             session.clear();
157             session.getTransaction().commit();
158             test = true;
159         } catch (Exception e) {
160             test = false;
161             session.getTransaction().rollback();
162         }finally {
163             if (session.isOpen()){
164                 session.close();
165             }
166         }
167         return test;
168     }
169 
170 }
```

A.4 JavaBeans code to implement user interface

We used the managed beans to link the UI JSF components so that when we fill out a form on web page, the entered values are automatically assigned to the corresponding filed in our java bean. The following codes are packed in the Managed Bean module.

A.4.1 userBean code

It will handle the navigation between "`User`" view and the other views.

```
1  import dao.userDAO;
2  import dao.userDaoImpl;
3  import java.awt.event.ActionEvent;
4  import java.io.File;
5  import java.io.FileInputStream;
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.io.PrintWriter;
9  import javax.inject.Named;
10 import javax.enterprise.context.SessionScoped;
11 import java.io.Serializable;
12 import java.sql.Connection;
13 import java.sql.DriverManager;
14 import java.sql.SQLException;
```

```
15 import java.util.ArrayList;
16 import java.util.HashMap;
17 import java.util.List;
18 import java.util.Map;
19 import javax.faces.application.FacesMessage;
20 import javax.faces.context.FacesContext;
21 import javax.servlet.ServletOutputStream;
22 import javax.servlet.http.HttpServletResponse;
23 import model.User;
24 import net.sf.jasperreports.engine.JRException;
25 import net.sf.jasperreports.engine.JRExporterParameter;
26 import net.sf.jasperreports.engine.JasperCompileManager;
27 import net.sf.jasperreports.engine.JasperExportManager;
28 import net.sf.jasperreports.engine.JasperFillManager;
29 import net.sf.jasperreports.engine.JasperPrint;
30 import net.sf.jasperreports.engine.JasperReport;
31 import net.sf.jasperreports.engine.export.JRHtmlExporter;
32 import net.sf.jasperreports.engine.export.JRHtmlExporterParameter;
33
34 @Named(value = "userBean")
35 @SessionScoped
36 public class UserBean implements Serializable {
37
38     private List<User> users;
39     private User selectedUser;
40     private Integer iduser;
41     private Boolean estate;
42     User newUser = new User();
43
44     public UserBean() {
45         this.selectedUser = new User();
46         this.users = new ArrayList<User>();
47     }
48
49
50     public List<User> getUsers() {
51         userDAO userDao = new userDaoImpl();
52         this.users = userDao.findAll();
53         return users;
54     }
55
56     public User getSelectedUser() {
57         return selectedUser;
58     }
59
60     public void setSelectedUser(User selectedUser) {
61         this.selectedUser = selectedUser;
62     }
63
64     public void btnCreateUser(ActionEvent actionEvent){
65         userDAO userDao = new userDaoImpl();
66         String msg;
67
68         if (userDao.create(this.selectedUser)) {
69             msg = "User created correctly.";
70             FacesMessage message = new FacesMessage(FacesMessage.
                   SEVERITY_INFO, msg, null);
71             FacesContext.getCurrentInstance().addMessage(null, message);
72             newUser = new User();
73         } else {
74             msg = "Error to create user.";
75             FacesMessage message = new FacesMessage(FacesMessage.
                   SEVERITY_ERROR, msg, null);
76             FacesContext.getCurrentInstance().addMessage(null, message);
77         }
78     }
79
80     public void btnUpdateUser(ActionEvent actionEvent)
81     {
82         userDAO userDao = new userDaoImpl();
83         String msg;
```

```
84          if (userDao.update(this.selectedUser)) {
85              msg = "User correctly updated.";
86              estate = this.selectedUser.getEstate();
87              System.out.println(this.selectedUser.getEstate());
88              System.out.println("ID user = "+this.selectedUser.getEstate()
                    );
89              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
90              FacesContext.getCurrentInstance().addMessage(null, message);
91              newUser = new User();
92          } else {
93
94              msg = "Error to update the user.";
95              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
96              FacesContext.getCurrentInstance().addMessage(null, message);
97          }
98      }
99
100     public void btnDeleteUser(ActionEvent actionEvent)
101     {
102         userDAO userDao = new userDaoImpl();
103         String msg;
104         if (userDao.delete(this.selectedUser.getIduser())) {
105             msg = "User correctly deleted.";
106             FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
107             FacesContext.getCurrentInstance().addMessage(null, message);
108         } else {
109             msg = "Error to delete the user.";
110             FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
111             FacesContext.getCurrentInstance().addMessage(null, message);
112         }
113     }
114
115     public void showReport(ActionEvent actionEvent){
116             iduser = this.selectedUser.getIduser();
117             System.out.println(this.selectedUser.getIduser());
118             System.out.println("ID user = "+this.selectedUser.getIduser()
                    );
119
120         try{
121                 String jrxmlFile = FacesContext.getCurrentInstance().
                        getExternalContext().getRealPath("/reports/report2.
                        jrxml");
122                 Connection con = DriverManager.getConnection("jdbc:mysql
                        ://localhost:8889/birthDB", "root", "root");
123                 Map parameter = new HashMap();
124                 parameter.put("iduser", iduser);
125                 InputStream input = new FileInputStream(new File(
                        jrxmlFile));
126                 JasperReport jasperReport = JasperCompileManager.
                        compileReport(input);
127                 JasperPrint jasperPrint = JasperFillManager.fillReport(
                        jasperReport, parameter, con);
128                 HttpServletResponse httpServletResponse = (
                        HttpServletResponse) FacesContext.getCurrentInstance()
                        .getExternalContext().getResponse();
129                 httpServletResponse.addHeader("contextType", "application
                        /pdf");
130                 ServletOutputStream servletOutputStream =
                        httpServletResponse.getOutputStream();
131                 JasperExportManager.exportReportToPdfStream(jasperPrint,
                        servletOutputStream);
132                 FacesContext.getCurrentInstance().responseComplete();
133
134         }catch(Exception e){
135             System.out.println(e.getMessage());
136         }
137     }
```

```
138 }
```

A.4.2 typehallBean code

It will handle the navigation between "TypeHall" view and the other views.

```
1  import dao.typehallDAO;
2  import dao.typehallDaoImpl;
3  import java.awt.event.ActionEvent;
4  import javax.inject.Named;
5  import javax.enterprise.context.SessionScoped;
6  import java.io.Serializable;
7  import java.util.ArrayList;
8  import java.util.List;
9  import javax.faces.application.FacesMessage;
10 import javax.faces.context.FacesContext;
11 import javax.faces.model.SelectItem;
12 import model.Typehall;
13 @Named(value = "typehallBean")
14 @SessionScoped
15 public class TypehallBean implements Serializable {
16
17     private List<Typehall> typehalls;
18     private Typehall selectedTypehall;
19     private List<SelectItem> selectOneItems;
20     Typehall newTypehall = new Typehall();
21
22     public TypehallBean() {
23         this.selectedTypehall = new Typehall();
24         this.typehalls = new ArrayList<Typehall>();
25     }
26
27     public List<Typehall> getTypehalls() {
28         typehallDAO typehallDao = new typehallDaoImpl();
29         this.typehalls = typehallDao.findAll();
30         return typehalls;
31     }
32
33     public Typehall getSelectedTypehall() {
34         return selectedTypehall;
35     }
36
37     public void setSelectedTypehall(Typehall selectedTypehall) {
38         this.selectedTypehall = selectedTypehall;
39     }
40
41
42     public List<SelectItem> getSelectOneItemsTypeHall() {
43         this.selectOneItems = new ArrayList<SelectItem>();
44         typehallDAO typehallDao = new typehallDaoImpl();
45         List<Typehall> typehalls = typehallDao.selectItems();
46         for (Typehall typehall : typehalls) {
47             SelectItem selectItem = new SelectItem(typehall.getIdtypehall
                   (), typehall.getTypehallname());
48             this.selectOneItems.add(selectItem);
49         }
50         return selectOneItems;
51     }
52
53     public void btnCreateTypehall(ActionEvent actionEvent){
54         typehallDAO typehallDao = new typehallDaoImpl();
55         String msg;
56         this.selectedTypehall.setHalls(this.selectedTypehall.getHalls());
57         this.selectedTypehall.setTypehallname(this.selectedTypehall.
               getTypehallname());
58
59         if (typehallDao.create(this.selectedTypehall)) {
60             msg = "Type Hall created correctly.";
```

```
61              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
62              FacesContext.getCurrentInstance().addMessage(null, message);
63              newTypehall = new Typehall();
64          } else {
65              msg = "Error to create the Type Hall.";
66              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
67              FacesContext.getCurrentInstance().addMessage(null, message);
68          }
69      }
70
71      public void btnUpdateTypehall(ActionEvent actionEvent)
72      {
73          typehallDAO typehallDao = new typehallDaoImpl();
74          String msg;
75          if (typehallDao.update(this.selectedTypehall)) {
76              msg = "Type Hall correctly updated.";
77              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
78              FacesContext.getCurrentInstance().addMessage(null, message);
79              newTypehall = new Typehall();
80          } else {
81              msg = "Error to update the Type Hall.";
82              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
83              FacesContext.getCurrentInstance().addMessage(null, message);
84          }
85      }
86
87      public void btnDeleteTypehall(ActionEvent actionEvent)
88      {
89          typehallDAO typehallDao = new typehallDaoImpl();
90          String msg;
91          if (typehallDao.delete(this.selectedTypehall.getIdtypehall())) {
92              msg = "User correctly Type Hall.";
93              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
94              FacesContext.getCurrentInstance().addMessage(null, message);
95          } else {
96              msg = "Error to delete the Type Hall.";
97              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
98              FacesContext.getCurrentInstance().addMessage(null, message);
99          }
100     }
101 }
```

A.4.3 hallBean code

It will handle the navigation between "`Hall`" view and the other views.

```
1  import dao.hallDAO;
2  import dao.hallDaoImpl;
3  import java.awt.event.ActionEvent;
4  import javax.inject.Named;
5  import javax.enterprise.context.SessionScoped;
6  import java.io.Serializable;
7  import java.util.ArrayList;
8  import java.util.List;
9  import javax.faces.application.FacesMessage;
10 import javax.faces.context.FacesContext;
11 import javax.faces.model.SelectItem;
12 import model.Hall;
13
14 @Named(value = "hallBean")
15 @SessionScoped
16 public class HallBean implements Serializable {
17
```

```
18      private List<Hall> halls;
19      private Hall selectedHall;
20      private List<SelectItem> selectOneItemsHall;
21      Hall newHall = new Hall();
22
23      public HallBean() {
24          this.selectedHall = new Hall();
25          this.halls = new ArrayList<Hall>();
26      }
27
28      public List<Hall> getHalls() {
29          hallDAO hallDao = new hallDaoImpl();
30          this.halls = hallDao.findAll();
31          return halls;
32      }
33
34      public Hall getSelectedHall() {
35          return selectedHall;
36      }
37
38      public void setSelectedHall(Hall selectedHall) {
39          this.selectedHall = selectedHall;
40      }
41
42      public List<SelectItem> getSelectOneItemsHall() {
43          this.selectOneItemsHall = new ArrayList<SelectItem>();
44          hallDAO hallDao = new hallDaoImpl();
45          List<Hall> halls = hallDao.selectItems();
46          for (Hall hall : halls) {
47              SelectItem selectItem = new SelectItem(hall.getIdhall(), hall
                    .getHallname());
48              this.selectOneItemsHall.add(selectItem);
49          }
50          return selectOneItemsHall;
51      }
52
53      public void btnCreateHall(ActionEvent actionEvent){
54          hallDAO hallDao = new hallDaoImpl();
55          String msg;
56
57          if (hallDao.create(this.selectedHall)) {
58              msg = "Hall created correctly.";
59              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
60              FacesContext.getCurrentInstance().addMessage(null, message);
61              newHall = new Hall();
62          } else {
63              msg = "Error to create Hall.";
64              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
65              FacesContext.getCurrentInstance().addMessage(null, message);
66          }
67      }
68
69      public void btnUpdateHall(ActionEvent actionEvent)
70      {
71          hallDAO hallDao = new hallDaoImpl();
72          String msg;
73          if (hallDao.update(this.selectedHall)) {
74              msg = "Hall correctly updated.";
75              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
76              FacesContext.getCurrentInstance().addMessage(null, message);
77              newHall = new Hall();
78          } else {
79              msg = "Error to update the Hall.";
80              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
81              FacesContext.getCurrentInstance().addMessage(null, message);
82          }
83      }
```

```
84
85      public void btnDeleteHall(ActionEvent actionEvent)
86      {
87          hallDAO hallDao = new hallDaoImpl();
88          String msg;
89          if (hallDao.delete(this.selectedHall.getIdhall())) {
90              msg = "Hall correctly deleted.";
91              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
92              FacesContext.getCurrentInstance().addMessage(null, message);
93          } else {
94              msg = "Error to delete the Hall.";
95              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
96              FacesContext.getCurrentInstance().addMessage(null, message);
97          }
98      }
99  }
```

A.4.4 prefectureBean code

It will handle the navigation between "`Prefecture`" view and the other views.

```
1   import dao.prefectureDAO;
2   import dao.prefectureDaoImpl;
3   import java.awt.event.ActionEvent;
4   import java.io.File;
5   import java.io.FileInputStream;
6   import java.io.InputStream;
7   import javax.inject.Named;
8   import javax.enterprise.context.SessionScoped;
9   import java.io.Serializable;
10  import java.sql.Connection;
11  import java.sql.DriverManager;
12  import java.util.ArrayList;
13  import java.util.HashMap;
14  import java.util.List;
15  import java.util.Map;
16  import javax.faces.application.FacesMessage;
17  import javax.faces.context.FacesContext;
18  import javax.faces.model.SelectItem;
19  import javax.servlet.ServletOutputStream;
20  import javax.servlet.http.HttpServletResponse;
21  import model.Prefecture;
22  import net.sf.jasperreports.engine.JasperCompileManager;
23  import net.sf.jasperreports.engine.JasperExportManager;
24  import net.sf.jasperreports.engine.JasperFillManager;
25  import net.sf.jasperreports.engine.JasperPrint;
26  import net.sf.jasperreports.engine.JasperReport;
27
28  @Named(value = "prefectureBean")
29  @SessionScoped
30  public class PrefectureBean implements Serializable {
31
32      private List<Prefecture> prefectures;
33      private Prefecture selectedPrefecture;
34      private List<SelectItem> selectOneItems;
35      private Integer idprefecture;
36      Prefecture newPrefecture = new Prefecture();
37
38      public PrefectureBean() {
39          this.selectedPrefecture = new Prefecture();
40          this.prefectures = new ArrayList<Prefecture>();
41      }
42
43      public List<Prefecture> getPrefectures() {
44          prefectureDAO prefectureDao = new prefectureDaoImpl();
45          this.prefectures = prefectureDao.findAll();
46          return prefectures;
```

```
47      }
48
49      public Prefecture getSelectedPrefecture() {
50          return selectedPrefecture;
51      }
52
53      public void setSelectedPrefecture(Prefecture selectedPrefecture) {
54          this.selectedPrefecture = selectedPrefecture;
55      }
56
57      public List<SelectItem> getSelectOneItemsPrefecture() {
58          this.selectOneItems = new ArrayList<SelectItem>();
59          prefectureDAO prefDao = new prefectureDaoImpl();
60          List<Prefecture> prefs = prefDao.selectItems();
61          for (Prefecture pref : prefs) {
62              SelectItem selectItem = new SelectItem(pref.getIdprefecture()
                    , pref.getPrefecturename());
63              this.selectOneItems.add(selectItem);
64          }
65          return selectOneItems;
66      }
67
68      public void btnCreatePrefecture(ActionEvent actionEvent){
69          prefectureDAO prefectureDao = new prefectureDaoImpl();
70          String msg;
71
72          if (prefectureDao.create(this.selectedPrefecture)) {
73              msg = "Prefecture created correctly.";
74              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
75              FacesContext.getCurrentInstance().addMessage(null, message);
76              newPrefecture = new Prefecture();
77          } else {
78              msg = "Error to create Prefecture.";
79              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
80              FacesContext.getCurrentInstance().addMessage(null, message);
81          }
82      }
83      public void btnUpdatePrefecture(ActionEvent actionEvent){
84          prefectureDAO prefectureDao = new prefectureDaoImpl();
85          String msg;
86          if (prefectureDao.update(this.selectedPrefecture)) {
87              msg = "Prefecture correctly updated.";
88              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
89              FacesContext.getCurrentInstance().addMessage(null, message);
90              newPrefecture = new Prefecture();
91          } else {
92              msg = "Error to update the Prefecture.";
93              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
94              FacesContext.getCurrentInstance().addMessage(null, message);
95          }
96      }
97
98      public void btnDeletePrefecture(ActionEvent actionEvent){
99          prefectureDAO prefectureDao = new prefectureDaoImpl();
100         String msg;
101
102         if (prefectureDao.delete(this.selectedPrefecture.getIdprefecture
                ())) {
103             msg = "Prefecture correctly deleted.";
104             FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
105             FacesContext.getCurrentInstance().addMessage(null, message);
106         } else {
107             msg = "Error to delete the Prefecture.";
108             FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
109             FacesContext.getCurrentInstance().addMessage(null, message);
```

```
110         }
111     }
112
113     public void showReport(ActionEvent actionEvent){
114             idprefecture = this.selectedPrefecture.getIdprefecture();
115             System.out.println(this.selectedPrefecture.getIdprefecture())
                    ;
116             System.out.println("ID Prefecture = "+this.selectedPrefecture
                    .getIdprefecture());
117         try{
118                 String jrxmlFile = FacesContext.getCurrentInstance().
                        getExternalContext().getRealPath("/reports/report3.
                        jrxml");
119                 Connection con = DriverManager.getConnection("jdbc:mysql
                        ://localhost:8889/birthDB", "root", "root");
120                 Map parameter = new HashMap();
121                 parameter.put("idprefecture", idprefecture);
122                 InputStream input = new FileInputStream(new File(
                        jrxmlFile));
123                 JasperReport jasperReport = JasperCompileManager.
                        compileReport(input);
124                 JasperPrint jasperPrint = JasperFillManager.fillReport(
                        jasperReport, parameter, con);
125                 HttpServletResponse httpServletResponse = (
                        HttpServletResponse) FacesContext.getCurrentInstance()
                        .getExternalContext().getResponse();
126                 httpServletResponse.addHeader("contextType", "application
                        /pdf");
127                 ServletOutputStream servletOutputStream =
                        httpServletResponse.getOutputStream();
128                 JasperExportManager.exportReportToPdfStream(jasperPrint,
                        servletOutputStream);
129                 FacesContext.getCurrentInstance().responseComplete();
130
131         }catch(Exception e){
132             System.out.println(e.getMessage()); }
133     }
134 }
```

A.4.5 regionBean code

It will handle the navigation between "`Region` view and the other views.

```
1  import dao.regionDAO;
2  import dao.regionDaoImpl;
3  import java.awt.event.ActionEvent;
4  import javax.inject.Named;
5  import javax.enterprise.context.SessionScoped;
6  import java.io.Serializable;
7  import java.util.ArrayList;
8  import java.util.List;
9  import javax.faces.application.FacesMessage;
10 import javax.faces.context.FacesContext;
11 import javax.faces.model.SelectItem;
12 import model.Region;
13
14 @Named(value = "regionBean")
15 @SessionScoped
16 public class RegionBean implements Serializable {
17
18     private List<Region> regions;
19     private Region selectedRegion;
20     private List<SelectItem> selectOneItemsRegion;
21     Region newRegion = new Region();
22
23     public RegionBean() {
24         this.selectedRegion = new Region();
25         this.regions = new ArrayList<Region>();
26     }
```

```
27
28      public List<Region> getRegions() {
29          regionDAO regionDao = new regionDaoImpl();
30          this.regions = regionDao.findAll();
31          return regions;
32      }
33
34      public Region getSelectedRegion() {
35          return selectedRegion;
36      }
37
38      public void setSelectedRegion(Region selectedRegion) {
39          this.selectedRegion = selectedRegion;
40      }
41
42      public List<SelectItem> getSelectOneItemsRegion() {
43          this.selectOneItemsRegion = new ArrayList<SelectItem>();
44          regionDAO regionDao = new regionDaoImpl();
45          List<Region> regions = regionDao.selectItems();
46          for (Region region : regions) {
47              SelectItem selectItem = new SelectItem(region.getIdregion(),
                    region.getRegionname());
48              this.selectOneItemsRegion.add(selectItem);
49          }
50          return selectOneItemsRegion;
51      }
52
53      public void btnCreateRegion(ActionEvent actionEvent){
54          regionDAO regionDao = new regionDaoImpl();
55          String msg;
56
57          if (regionDao.create(this.selectedRegion)) {
58              msg = "Region created correctly.";
59              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
60              FacesContext.getCurrentInstance().addMessage(null, message);
61              newRegion = new Region();
62          } else {
63              msg = "Error to create Region.";
64              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
65              FacesContext.getCurrentInstance().addMessage(null, message);
66          }
67      }
68
69      public void btnUpdateRegion(ActionEvent actionEvent)
70      {
71          regionDAO regionDao = new regionDaoImpl();
72          String msg;
73          if (regionDao.update(this.selectedRegion)) {
74              msg = "Region correctly updated.";
75              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
76              FacesContext.getCurrentInstance().addMessage(null, message);
77              newRegion = new Region();
78          } else {
79              msg = "Error to update the Region.";
80              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
81              FacesContext.getCurrentInstance().addMessage(null, message);
82          }
83      }
84
85      public void btnDeleteRegion(ActionEvent actionEvent)
86      {
87          regionDAO regionDao = new regionDaoImpl();
88          String msg;
89          if (regionDao.delete(this.selectedRegion.getIdregion())) {
90              msg = "Region correctly deleted.";
91              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
```

```
92              FacesContext.getCurrentInstance().addMessage(null, message);
93          } else {
94              msg = "Error to delete the Region.";
95              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
96              FacesContext.getCurrentInstance().addMessage(null, message);
97          }
98      }
99
100 }
```

A.4.6 birthBean code

It will handle the navigation between "`BirthInfo`" view and the other views.

```
1   import dao.birthinfoDAO;
2   import dao.birthinfoDaoImpl;
3   import java.awt.event.ActionEvent;
4   import java.io.File;
5   import java.io.FileInputStream;
6   import java.io.InputStream;
7   import javax.inject.Named;
8   import javax.enterprise.context.SessionScoped;
9   import java.io.Serializable;
10  import java.sql.Connection;
11  import java.sql.DriverManager;
12  import java.text.ParseException;
13  import java.util.ArrayList;
14  import java.util.Calendar;
15  import java.util.Date;
16  import java.util.HashMap;
17  import java.util.List;
18  import java.util.Map;
19  import java.util.UUID;
20  import javax.faces.application.FacesMessage;
21  import javax.faces.context.FacesContext;
22  import javax.servlet.ServletOutputStream;
23  import javax.servlet.http.HttpServletResponse;
24  import model.BirthInfo;
25  import net.sf.jasperreports.engine.JasperCompileManager;
26  import net.sf.jasperreports.engine.JasperExportManager;
27  import net.sf.jasperreports.engine.JasperFillManager;
28  import net.sf.jasperreports.engine.JasperPrint;
29  import net.sf.jasperreports.engine.JasperReport;
30
31  @Named(value = "birthBean")
32  @SessionScoped
33  public class BirthBean implements Serializable {
34
35      private List<BirthInfo> births;
36      private BirthInfo selectedBirth;
37      private List<BirthInfo> searchByReferenceNumberList;
38      private Integer idbirthinfo;
39      BirthInfo newBirthinfo = new BirthInfo();
40
41      public BirthBean() {
42          this.selectedBirth = new BirthInfo();
43          this.births = new ArrayList<BirthInfo>();
44      }
45
46      public List<BirthInfo> getBirthinfos() {
47          birthinfoDAO birthDao = new birthinfoDaoImpl();
48          this.births = birthDao.findAll();
49          return births;
50      }
51
52      public BirthInfo getSelectedBirth() {
53          return selectedBirth;
54      }
```

```
55
56      public void setSelectedBirth(BirthInfo selectedBirth) {
57          this.selectedBirth = selectedBirth;
58      }
59
60      public void btnCreateBirth(ActionEvent actionEvent) throws
            ParseException{
61          birthinfoDAO birthDao = new birthinfoDaoImpl();
62          String msg;
63
64          if (birthDao.create(this.selectedBirth)) {
65              msg = "Birth created correctly.";
66              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
67              FacesContext.getCurrentInstance().addMessage(null, message);
68              newBirthinfo = new BirthInfo();
69
70          } else {
71              msg = "Error to create Birth.";
72              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
73              FacesContext.getCurrentInstance().addMessage(null, message);
74          }
75      }
76
77      public void btnUpdateBirth(ActionEvent actionEvent)
78      {
79          birthinfoDAO birthDao = new birthinfoDaoImpl();
80          String msg;
81          if (birthDao.update(this.selectedBirth)) {
82              msg = "Birth correctly updated.";
83              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
84              FacesContext.getCurrentInstance().addMessage(null, message);
85              newBirthinfo = new BirthInfo();
86          } else {
87              msg = "Error to update the Birth.";
88              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
89              FacesContext.getCurrentInstance().addMessage(null, message);
90          }
91      }
92
93      public void btnDeleteBirth(ActionEvent actionEvent)
94      {
95          birthinfoDAO birthDao = new birthinfoDaoImpl();
96          String msg;
97          if (birthDao.delete(this.selectedBirth.getIdbirthinfo())) {
98              msg = "Birth correctly deleted.";
99              FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_INFO, msg, null);
100             FacesContext.getCurrentInstance().addMessage(null, message);
101         } else {
102             msg = "Error to delete the Birth.";
103             FacesMessage message = new FacesMessage(FacesMessage.
                    SEVERITY_ERROR, msg, null);
104             FacesContext.getCurrentInstance().addMessage(null, message);
105         }
106     }
107
108     public void searchbyRefenceNumber(){
109         birthinfoDAO birthDao = new birthinfoDaoImpl();
110         String msg;
111         this.searchByReferenceNumberList = birthDao.
                SearchByReferenceNumber(this.selectedBirth.getReference());
112         int count = searchByReferenceNumberList.size();
113         FacesMessage message = new FacesMessage(FacesMessage.
                SEVERITY_INFO, "Reference Number Selected:", Integer.toString
                (count));
114         FacesContext.getCurrentInstance().addMessage(null, message);
115     }
```

```
116
117     public List<BirthInfo> getSearchByReferenceNumberList() {
118         return searchByReferenceNumberList;
119     }
120
121     public void setSearchByReferenceNumberList(List<BirthInfo>
            searchByReferenceNumberList) {
122         this.searchByReferenceNumberList = searchByReferenceNumberList;
123     }
124
125     public void showReport(ActionEvent actionEvent){
126             idbirthinfo = this.selectedBirth.getIdbirthinfo();
127             System.out.println(this.selectedBirth.getIdbirthinfo());
128             System.out.println("ID Prefecture = "+idbirthinfo);
129         try{
130                 String jrxmlFile = FacesContext.getCurrentInstance().
                        getExternalContext().getRealPath("/reports/report4.
                        jrxml");
131                 Connection con = DriverManager.getConnection("jdbc:mysql
                        ://localhost:8889/birthDB", "root", "root");
132                 Map parameter = new HashMap();
133                 parameter.put("idbirthinfo", idbirthinfo);
134                 parameter.put("Image", FacesContext.getCurrentInstance().
                        getExternalContext().getRealPath("/resources/images/
                        coatarmsTogo.png"));
135                 InputStream input = new FileInputStream(new File(
                        jrxmlFile));
136                 JasperReport jasperReport = JasperCompileManager.
                        compileReport(input);
137                 JasperPrint jasperPrint = JasperFillManager.fillReport(
                        jasperReport, parameter, con);
138                 HttpServletResponse httpServletResponse = (
                        HttpServletResponse) FacesContext.getCurrentInstance()
                        .getExternalContext().getResponse();
139                 httpServletResponse.addHeader("contextType", "application
                        /pdf");
140                 ServletOutputStream servletOutputStream =
                        httpServletResponse.getOutputStream();
141                 JasperExportManager.exportReportToPdfStream(jasperPrint,
                        servletOutputStream);
142                 FacesContext.getCurrentInstance().responseComplete();
143
144         }catch(Exception e){
145             System.out.println(e.getMessage());
146         }
147     }
148
149     public Date getTodayDate(Date date){
150         Calendar calendar = Calendar.getInstance();
151         return calendar.getTime();
152     }
153
154     public String getRefNumber(){
155             String rand = UUID.randomUUID().toString();
156             String ref = "TG"+rand.toUpperCase()+"TG";
157             return ref;
158     }
159
160 }
```

A.5 XHTML code for JSF

The following codes represents the xhtml code using JSF.

A.5.1 Login page

It can be found in the Web pages module. We can access to the application by putting credentials.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD␣XHTML␣1.0␣Transitional//EN" "http://
       www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml"
4        xmlns:h="http://java.sun.com/jsf/html"
5        xmlns:p="http://primefaces.org/ui"
6        xmlns:f="http://java.sun.com/jsf/core">
7      <h:head>
8          <title>Login</title>
9          <link href="#{facesContext.externalContext.requestContextPath}/
               resources/css/login.css" type="text/css" rel="stylesheet"/>
10     </h:head>
11     <h:body>
12         <div class="container">
13             <p:growl id="growl" showDetail="true" life="3000" />
14             <h:form id="formLogin">
15                 <p:panel header="Enter␣Username␣and␣Password">
16                     <h:panelGrid columns="2" cellpadding="5">
17                         <h:outputLabel for="username" value="Username:" />
18                         <p:inputText value="#{loginBean.user.username}"
19                                 id="username" required="true" label="
                                       username" />
20
21                         <h:outputLabel for="password" value="Password:" />
22                         <p:password value="#{loginBean.user.password}"
23                                 id="password" required="true" label="
                                       password"/>
24                         <f:facet name="footer">
25                             <p:commandButton id="loginButton" value="Login"
                                    action="#{loginBean.login(event)}" update=
                                   ":growl" icon="icon-login"
26                                          oncomplete="handleLoginRequest(
                                               xhr,␣status,␣args)"/>
27                         </f:facet>
28                     </h:panelGrid>
29                 </p:panel>
30             </h:form>
31         </div>
32         <script type="text/javascript">
33             function handleLoginRequest(xhr, status, args) {
34                 if(args.validationFailed || !args.loggedIn) {
35                     jQuery('#formLogin').effect("shake", { times:3 }, 100)
                           ;
36                 } else {
37                     location.href = args.route;
38                 }
39             }
40         </script>
41     </h:body>
42 </html>
```

A.5.2 Top page

It can be found in the Web pages module. After granted the access, we have the top page.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
       www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml"
4        xmlns:ui="http://java.sun.com/jsf/facelets"
5        xmlns:h="http://java.sun.com/jsf/html"
6        xmlns:p="http://primefaces.org/ui">
```

```
 7
 8      <h:head>
 9          <meta http-equiv="Content-Type" content="text/html; charset=UTF
                -8" />
10          <link href="#{facesContext.externalContext.requestContextPath}/
                resources/css/default.css" rel="stylesheet" type="text/css"
                />
11          <link href="#{facesContext.externalContext.requestContextPath}/
                resources/css/cssLayout.css" rel="stylesheet" type="text/css"
                 />
12          <title></title>
13      </h:head>
14
15      <h:body>
16
17          <div id="container">
18              <p:clock pattern="HH:mm:ss yyyy/MM/dd" mode="client"/>
19              <div id="top" class="ui-widget-header ui-corner-all">
20                  <ui:insert name="top">DATA MANAGEMENT WITH BLOCKCHAIN</ui
                        :insert>
21              </div>
22              <div id="menu">
23                  <h:form>
24                      <p:menubar>
25                          <p:submenu label="File" icon="ui-icon-document">
26                              <p:menuitem value="Edit" url="#" icon="ui-icon-
                                    pencil"/>
27                              <p:separator />
28                              <p:menuitem value="Quit" url="#" />
29                          </p:submenu>
30
31                          <p:submenu label="Role">
32                              <p:menuitem value="Edit" url="#{appBean.
                                    basePath}role/role.xhtml" icon="ui-icon-
                                    pencil"/>
33                          </p:submenu>
34
35                          <p:submenu label="User" icon="ui-icon-person">
36                              <p:menuitem value="Edit" url="#{appBean.
                                    basePath}user/user.xhtml" icon="ui-icon-
                                    pencil"/>
37                          </p:submenu>
38
39                          <p:submenu label="Region">
40                              <p:menuitem value="Edit" url="#{appBean.
                                    basePath}region/region.xhtml" icon="ui-icon
                                    -pencil"/>
41                          </p:submenu>
42
43                          <p:submenu label="Prefecture">
44                              <p:menuitem value="Edit" url="#{appBean.
                                    basePath}prefecture/prefecture.xhtml" icon
                                    ="ui-icon-pencil"/>
45                          </p:submenu>
46
47                          <p:submenu label="Type Hall">
48                              <p:menuitem value="Edit" url="#{appBean.
                                    basePath}typehall/typehall.xhtml" icon="ui-
                                    icon-pencil"/>
49                          </p:submenu>
50
51                          <p:submenu label="Hall">
52                              <p:menuitem value="Edit" url="#{appBean.
                                    basePath}hall/hall.xhtml" icon="ui-icon-
                                    pencil"/>
53                          </p:submenu>
54
55                          <p:submenu label="Birth Certificate" icon="ui-icon
                                -document">
56                              <p:menuitem value="Register" url="#{appBean.
                                    basePath}birth/birthinfo.xhtml" icon="ui-
```

```
                            icon-pencil"/>
57                          <p:menuitem value="Print" url="#" icon="ui-icon
                                -print" />
58                      </p:submenu>
59
60                      <p:submenu label="Statistics">
61                          <p:menuitem value="By Region" url="#" icon="ui-
                                icon-arrowreturnthick-1-w" />
62                          <p:menuitem value="By Prefecture" url="#" icon
                                ="ui-icon-arrowreturnthick-1-e" />
63                          <p:menuitem value="By Hall" url="#" icon="ui-
                                icon-arrowreturnthick-1-e" />
64                      </p:submenu>
65
66                      <p:submenu label="Search" icon="ui-icon-search">
67                          <p:menuitem value="By Region" url="#" icon="ui-
                                icon-arrowreturnthick-1-w" />
68                          <p:menuitem value="By Prefecture" url="#" icon
                                ="ui-icon-arrowreturnthick-1-e" />
69                          <p:menuitem value="By Hall" url="#" icon="ui-
                                icon-arrowreturnthick-1-e" />
70                      </p:submenu>
71
72                      <p:submenu label="Actions" icon="ui-icon-gear">
73                          <p:submenu label="Ajax" icon="ui-icon-refresh">
74                              <p:menuitem value="Save" actionListener="#{
                                    menuBean.save}" icon="ui-icon-disk"/>
75                              <p:menuitem value="Update" actionListener
                                    ="#{menuBean.update}" icon="ui-icon-
                                    arrowrefresh-1-w"/>
76                          </p:submenu>
77                          <p:submenu label="Non-Ajax" icon="ui-icon-
                                newwin">
78                              <p:menuitem value="Delete" actionListener
                                    ="#{menuBean.delete}" icon="ui-icon-
                                    close" ajax="false"/>
79                          </p:submenu>
80                      </p:submenu>
81                      <p:menuitem value="Log Out" actionListener="#{
                            loginBean.logout}" icon="ui-icon-close"
                            oncomplete="handleLoginRequest(xhr, status,
                            args)"/>
82                  </p:menubar>
83              </h:form>
84          </div>
85          <div id="content" class="center_content ui-widget-content ui-
                corner-all">
86              <ui:insert name="content"></ui:insert>
87          </div>
88          <div id="bottom" class="ui-widget-header ui-corner-all">
89              <ui:insert name="bottom">&copy; 2018</ui:insert>
90          </div>
91      </div>
92      <script type="text/javascript">
93          function handleLoginRequest(xhr, status, args) {
94              if(args.loggedOut) {
95                  location.href = args.route;
96              }
97          }
98      </script>
99  </h:body>
100
101 </html>
```

A.5.3 Birth registration page

It can be found in the Web pages module. We can insert or update data in the table "`BirthInfo`".

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
       www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml"
4        xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
5        xmlns:h="http://java.sun.com/jsf/html"
6        xmlns:p="http://primefaces.org/ui"
7        xmlns:f="http://java.sun.com/jsf/core">
8
9      <body>
10
11         <ui:composition template="./../home.xhtml">
12
13             <ui:define name="content">
14                 <p:growl id="msgs" showDetail="true" />
15                 <h:form>
16                     <p:commandButton id="btnBirthCreate" update=":
                           formCreate" oncomplete="PF('dialogBirthCreate').
                           show()" icon="icon-new" title="Create" value="
                           Create"/>
17                 </h:form>
18                 <h:form id="formDataTable" style="text-align: center">
19                     <p:dataTable id="births" var="birth" value="#{
                           birthBean.birthinfos}" >
20                     <f:facet name="header">
21                         List of the Birth Registered
22                     </f:facet>
23                     <p:column headerText="Reference" style="width:8%">
24                         <h:outputText value="#{birth.reference}" />
25                     </p:column>
26
27                     <p:column headerText="Child Name" style="width
                           :8%">
28                         <h:outputText value="#{birth.childfirstlastname
                               }" />
29                     </p:column>
30
31                     <p:column headerText="Sex" style="width:8%">
32                         <h:outputText value="#{birth.gender}" />
33                     </p:column>
34
35                     <p:column headerText="Date of Birth" style="width
                           :8%">
36                         <h:outputText value="#{birth.birthdate}">
37                             <f:convertDateTime type="date" pattern="
                                   yyyy-MM-dd" />
38                         </h:outputText>
39                     </p:column>
40
41                     <p:column headerText="Place of birth" style="width
                           :8%">
42                         <h:outputText value="#{birth.birthplace}" />
43                     </p:column>
44
45                     <p:column headerText="Father Name" style="width
                           :8%">
46                         <h:outputText value="#{birth.
                               fatherfirstlastname}" />
47                     </p:column>
48
49                     <p:column headerText="Mother Name" style="width
                           :8%">
50                         <h:outputText value="#{birth.
                               motherfirstlastname}" />
51                     </p:column>
52
53                     <p:column headerText="Date of Registration" style
                           ="width:8%">
54                         <h:outputText value="#{birth.registrationdate
                               }">
```

```
55                          <f:convertDateTime type="date" pattern="
                                yyyy-MM-dd" />
56                      </h:outputText>
57                  </p:column>
58
59                  <p:column style="width:4%">
60                      <p:commandButton id="btnUpdate" update=":
                            formUpdate" oncomplete="PF('
                            dialogBirthUpdate').show()" icon="icon-edit
                            " title="Modify">
61                          <f:setPropertyActionListener value="#{birth
                                }" target="#{birthBean.selectedBirth}"
                                />
62                      </p:commandButton>
63                      <p:commandButton id="btnDelete" update=":
                            formDelete" oncomplete="PF('
                            dialogBirthDelete').show()" icon="icon-
                            delete" title="Delete">
64                          <f:setPropertyActionListener value="#{birth
                                }" target="#{birthBean.selectedBirth}"
                                />
65                      </p:commandButton>
66                      <p:commandButton id="btnPrint" update=":
                            formPrint" oncomplete="PF('dialogBirthPrint
                            ').show()" icon="ui-icon-print" title="
                            Print">
67                          <f:setPropertyActionListener value="#{birth
                                }" target="#{birthBean.selectedBirth}"
                                />
68                      </p:commandButton>
69                  </p:column>
70
71              </p:dataTable>
72          </h:form>
73
74          <h:form id="formCreate">
75              <p:dialog header="Create Birth" widgetVar="
                    dialogBirthCreate" resizable="false" id="
                    dlgBirthCreate"
76                       showEffect="fade" hideEffect="explode" modal
                             ="true">
77                  <h:panelGrid id="display" columns="2" cellpadding
                        ="4" style="margin:0 auto;">
78
79                      <h:outputText value="Child Full Name" />
80                      <p:inputText value="#{birthBean.selectedBirth.
                            childfirstlastname}" placeholder="Child
                            Full Name" size="40"/>
81
82                      <h:outputText value="Gender" />
83                      <p:selectOneMenu value="#{birthBean.
                            selectedBirth.gender}" style="width:125px">
84                          <f:selectItem itemLabel="- Select Sex -"
                                itemValue = "" />
85                          <f:selectItem itemLabel="Male" itemValue =
                                "Male" />
86                          <f:selectItem itemLabel="Female" itemValue
                                = "Female" />
87                      </p:selectOneMenu>
88
89                      <h:outputText value="Date of birth" />
90                      <p:calendar id="button" value="#{birthBean.
                            selectedBirth.birthdate}" pattern="yyyy/MM/
                            dd" label="DatePosted:" placeholder="Date
                            of Birth" showOn="button" required="true"/>
91
92                      <h:outputText value="Place of birth" />
93                      <p:inputText value="#{birthBean.selectedBirth.
                            birthplace}" placeholder="Place of Birth"
                            size="40"/>
94
```

```
95                          <h:outputText value="Hall" />
96                          <p:selectOneMenu value="#{birthBean.
                                selectedBirth.hall.idhall}" style="width
                                :125px">
97                              <f:selectItem itemLabel="- Select Hall -"
                                    itemValue = "" />
98                              <f:selectItems value="#{hallBean.
                                    selectOneItemsHall}" />
99                          </p:selectOneMenu>
100
101                         <h:outputText value="Father Full Name" />
102                         <p:inputText value="#{birthBean.selectedBirth.
                                fatherfirstlastname}" placeholder="Father
                                Full Name" size="40"/>
103
104                         <h:outputText value="Father Age " />
105                         <p:inputText value="#{birthBean.selectedBirth.
                                fatherage}" placeholder="Father Age" size
                                ="10"/>
106
107                         <h:outputText value="Father Occupation" />
108                         <p:inputText value="#{birthBean.selectedBirth.
                                fatheroccupation}" placeholder="Father
                                Occupation" size="40"/>
109
110                         <h:outputText value="Mother Full Name" />
111                         <p:inputText value="#{birthBean.selectedBirth.
                                motherfirstlastname}" placeholder="Mother
                                Full Name" size="40"/>
112
113                         <h:outputText value="Mother Age" />
114                         <p:inputText value="#{birthBean.selectedBirth.
                                motherage}" placeholder="Mother Age" size
                                ="10"/>
115
116                         <h:outputText value="Mother Occupation" />
117                         <p:inputText value="#{birthBean.selectedBirth.
                                motheroccupation}" placeholder="Mother
                                Occupation" size="40"/>
118
119                         <h:outputText value="Date of Registration"/>
120                         <p:calendar value="#{birthBean.selectedBirth.
                                registrationdate}" pattern="yyyy/MM/dd"
                                showOn="button" required="true" size="40"
                                />
121
122                         <h:outputText value="Registration Reference"/>
123                         <p:inputText value="#{birthBean.selectedBirth.
                                refNumber}" disabled="true" size="40"/>
124
125                         <f:facet name="footer">
126                             <p:separator/>
127                             <p:commandButton id="btnCreateAccept"
                                    update=":formDataTable, :msgs"
                                    oncomplete="PF('dialogBirthCreate').
                                    hide()" action="#{birthBean.
                                    btnCreateBirth(actionEvent)}" icon="
                                    icon-save" title="Save" value="Save"/>
128                             <p:commandButton id="btnCreateCancel"
                                    oncomplete="PF('dialogBirthCreate').
                                    hide()" icon="icon-cancel" title="
                                    Cancel" value="Cancel"/>
129                         </f:facet>
130                     </h:panelGrid>
131                 </p:dialog>
132             </h:form>
133
134             <h:form id="formUpdate">
135                 <p:dialog header="Modify Birth" widgetVar="
                        dialogBirthUpdate" resizable="false" id="
                        dlgBirthUpdate"
```

```
136                         showEffect="fade" hideEffect="explode" modal
                               ="true">
137                     <h:inputHidden value="#{birthBean.selectedBirth.
                            idbirthinfo}" />
138                     <h:panelGrid id="display" columns="2" cellpadding
                            ="4" style="margin:0 auto;">
139                         <h:outputText value="Child Full Name" />
140                         <p:inputText value="#{birthBean.selectedBirth.
                                childfirstlastname}" placeholder="Child
                                Full Name" size="40"/>
141
142                         <h:outputText value="Gender" />
143                         <p:selectOneMenu value="#{birthBean.
                                selectedBirth.gender}" style="width:125px">
144                             <f:selectItem itemLabel="- Select Sex -"
                                    itemValue = "" />
145                             <f:selectItem itemLabel="Male" itemValue =
                                    "Male" />
146                             <f:selectItem itemLabel="Female" itemValue
                                    = "Female" />
147                         </p:selectOneMenu>
148
149                         <h:outputText value="Date of birth" />
150                         <p:calendar id="button" value="#{birthBean.
                                selectedBirth.birthdate}" pattern="yyyy/MM/
                                dd" label="DatePosted:" placeholder="Date
                                of Birth" showOn="button" required="true"
                                />
151
152                         <h:outputText value="Place of birth" />
153                         <p:inputText value="#{birthBean.selectedBirth.
                                birthplace}" placeholder="Place of Birth"
                                size="40"/>
154
155                         <h:outputText value="Hall" />
156                         <p:selectOneMenu value="#{birthBean.
                                selectedBirth.hall.idhall}" style="width
                                :125px">
157                             <f:selectItem itemLabel="- Select Hall -"
                                    itemValue = "" />
158                             <f:selectItems value="#{hallBean.
                                    selectOneItemsHall}" />
159                         </p:selectOneMenu>
160
161                         <h:outputText value="Father Full Name" />
162                         <p:inputText value="#{birthBean.selectedBirth.
                                fatherfirstlastname}" placeholder="Father
                                Full Name" size="40"/>
163
164                         <h:outputText value="Father Age " />
165                         <p:inputText value="#{birthBean.selectedBirth.
                                fatherage}" placeholder="Father Age" size
                                ="40"/>
166
167                         <h:outputText value="Father Occupation" />
168                         <p:inputText value="#{birthBean.selectedBirth.
                                fatheroccupation}" placeholder="Father
                                Occupation" size="40"/>
169
170                         <h:outputText value="Mother Full Name" />
171                         <p:inputText value="#{birthBean.selectedBirth.
                                motherfirstlastname}" placeholder="Mother
                                Full Name" size="40"/>
172
173                         <h:outputText value="Mother Age" />
174                         <p:inputText value="#{birthBean.selectedBirth.
                                motherage}" placeholder="Mother Age" size
                                ="40"/>
175
176                         <h:outputText value="Mother Occupation" />
177                         <p:inputText value="#{birthBean.selectedBirth.
```

```
                                motheroccupation}" placeholder="Mother
                                Occupation" size="40"/>
178
179                         <h:outputText value="Date of Registration"/>
180                         <p:calendar value="#{birthBean.selectedBirth.
                                registrationdate}" pattern="yyyy/MM/dd"
                                showOn="button" required="true" size="40"
                                />
181
182                         <h:outputText value="Registration Reference"/>
183                         <p:inputText value="#{birthBean.selectedBirth.
                                refNumber}" disabled="true" size="40"/>
184
185                         <f:facet name="footer">
186                             <p:separator/>
187                             <p:commandButton id="btnUpdateAccept"
                                    update=":formDataTable, :msgs"
                                    oncomplete="PF('dialogBirthUpdate').
                                    hide()" action="#{birthBean.
                                    btnUpdateBirth(actionEvent)}" icon="
                                    icon-save" title="Save" value="Save"/>
188                             <p:commandButton id="btnUpdateCancel"
                                    oncomplete="PF('dialogBirthUpdate').
                                    hide()" icon="icon-cancel" title="
                                    Cancel" value="Cancel"/>
189                         </f:facet>
190                     </h:panelGrid>
191                 </p:dialog>
192             </h:form>
193             <h:form id="formDelete">
194                 <p:confirmDialog id="confirmDialog" message="Are you
                        sure to delete the Birth?" showEffect="fade"
                        hideEffect="explode"
195                             header="Delete Birth" severity="alert"
                                widgetVar="dialogBirthDelete">
196                     <h:inputHidden value="#{birthBean.selectedBirth.
                            idbirthinfo}"/>
197                     <p:commandButton id="confirm" value="Accepted"
                            update=":formDataTable:, :msgs" oncomplete="PF
                            ('dialogBirthDelete').hide()"
198                                 action="#{birthBean.btnDeleteBirth(
                                        actionEvent)}" icon="icon-check
                                        "/>
199                     <p:commandButton id="decline" value="Canceled"
                            onclick="PF('dialogBirthDelete').hide()" type
                            ="button" icon="icon-cancel"/>
200                 </p:confirmDialog>
201             </h:form>
202
203             <h:form id="formPrint">
204                 <p:confirmDialog id="confirmDialog" message="Are you
                        sure to Print the Birth Certificate?" showEffect="
                        fade" hideEffect="explode"
205                             header="Print Birth Certificate" severity="
                                alert" widgetVar="dialogBirthPrint">
206                     <h:inputHidden value="#{birthBean.selectedBirth.
                            idbirthinfo}"/>
207                     <p:commandButton id="confirm" value="Accepted"
                            update=":formDataTable" oncomplete="PF('
                            dialogBirthPrint').hide()"
208                                 action="#{birthBean.showReport(
                                        actionEvent)}" ajax="false"
                                        icon="icon-check" />
209                     <p:commandButton id="decline" value="Canceled"
                            onclick="PF('dialogBirthPrint').hide()" type="
                            button" icon="icon-cancel"/>
210                 </p:confirmDialog>
211             </h:form>
212         </ui:define>
213
214     </ui:composition>
```

```
215
216     </body>
217 </html>
```

A.5.4 Hall registration page

It can be found in the Web pages module. We can insert or update data in the table "`Hall`".

```
 1 <?xml version='1.0' encoding='UTF-8' ?>
 2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
       www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
 3 <html xmlns="http://www.w3.org/1999/xhtml"
 4       xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
 5       xmlns:h="http://java.sun.com/jsf/html"
 6       xmlns:p="http://primefaces.org/ui"
 7       xmlns:f="http://java.sun.com/jsf/core">
 8
 9     <body>
10
11         <ui:composition template="./../home.xhtml">
12
13             <ui:define name="content">
14                 <p:growl id="msgs" showDetail="true" />
15                 <h:form>
16                     <p:commandButton id="btnHallCreate" update=":
                           formCreate" oncomplete="PF('dialogHallCreate').
                           show()" icon="icon-new" title="Create" value="
                           Create"/>
17                 </h:form>
18                 <h:form id="formDataTable" style="text-align: center">
19                     <p:dataTable id="halls" var="hall" value="#{hallBean.
                           halls}" >
20                         <f:facet name="header">
21                             List of the Halls
22                         </f:facet>
23
24                         <p:column headerText="Name" style="width:24%">
25                             <h:outputText value="#{hall.hallname}" />
26                         </p:column>
27
28                         <p:column headerText="Type Hall" style="width
                               :24%">
29                             <h:outputText value="#{hall.typehall.
                                   typehallname}" />
30                         </p:column>
31
32                         <p:column headerText="Prefecture" style="width
                               :24%">
33                             <h:outputText value="#{hall.prefecture.
                                   prefecturename}" />
34                         </p:column>
35
36                         <p:column style="width:4%">
37                             <p:commandButton id="btnUpdate" update=":
                                   formUpdate" oncomplete="PF('
                                   dialogHallUpdate').show()" icon="icon-edit"
                                    title="Modify">
38                                 <f:setPropertyActionListener value="#{hall
                                       }" target="#{hallBean.selectedHall}" />
39                             </p:commandButton>
40                             <p:commandButton id="btnDelete" update=":
                                   formDelete" oncomplete="PF('
                                   dialogHallDelete').show()" icon="icon-
                                   delete" title="Delete">
41                                 <f:setPropertyActionListener value="#{hall
                                       }" target="#{hallBean.selectedHall}" />
42                             </p:commandButton>
43                         </p:column>
```

```
44
45                  </p:dataTable>
46              </h:form>
47              <h:form id="formCreate">
48                  <p:dialog header="Create Hall" widgetVar="
                        dialogHallCreate" resizable="false" id="
                        dlgHallCreate"
49                           showEffect="fade" hideEffect="explode" modal
                                ="true">
50                      <h:panelGrid id="display" columns="2" cellpadding
                            ="4" style="margin:0 auto;">
51
52                          <h:outputText value="Prefecture:" />
53                          <p:selectOneMenu value="#{hallBean.selectedHall
                                .prefecture.idprefecture}">
54                              <f:selectItem itemLabel="- Selected -"
                                    itemValue = "" />
55                              <f:selectItems value="#{prefectureBean.
                                    selectOneItemsPrefecture}" />
56                          </p:selectOneMenu>
57
58                          <h:outputText value="Type Hall:" />
59                          <p:selectOneMenu value="#{hallBean.selectedHall
                                .typehall.idtypehall}">
60                              <f:selectItem itemLabel="- Selected -"
                                    itemValue = "" />
61                              <f:selectItems value="#{typehallBean.
                                    selectOneItemsTypeHall}" />
62                          </p:selectOneMenu>
63
64                          <h:outputText value="Name:" />
65                          <p:inputText value="#{hallBean.selectedHall.
                                hallname}" size="40"/>
66
67                          <f:facet name="footer">
68                              <p:separator/>
69                              <p:commandButton id="btnCreateAccept"
                                    update=":formDataTable, :msgs"
                                    oncomplete="PF('dialogHallCreate').hide
                                    ()" action="#{hallBean.btnCreateHall(
                                    actionEvent)}" icon="icon-save" title="
                                    Save" value="Save"/>
70                              <p:commandButton id="btnCreateCancel"
                                    oncomplete="PF('dialogHallCreate').hide
                                    ()" icon="icon-cancel" title="Cancel"
                                    value="Cancel"/>
71                          </f:facet>
72                      </h:panelGrid>
73                  </p:dialog>
74              </h:form>
75              <h:form id="formUpdate">
76                  <p:dialog header="Modify Hall" widgetVar="
                        dialogHallUpdate" resizable="false" id="
                        dlgUserUpdate"
77                           showEffect="fade" hideEffect="explode" modal
                                ="true">
78                      <h:inputHidden value="#{hallBean.selectedHall.
                            idhall}" />
79                      <h:panelGrid id="display" columns="2" cellpadding
                            ="4" style="margin:0 auto;">
80
81                          <h:outputText value="Prefecture:" />
82                          <p:selectOneMenu value="#{hallBean.selectedHall
                                .prefecture.idprefecture}">
83                              <f:selectItem itemLabel="- Selected -"
                                    itemValue = "" />
84                              <f:selectItems value="#{prefectureBean.
                                    selectOneItemsPrefecture}" />
85                          </p:selectOneMenu>
86
87                          <h:outputText value="Type Hall:" />
```

```
88                          <p:selectOneMenu value="#{hallBean.selectedHall
                                .typehall.idtypehall}">
89                              <f:selectItem itemLabel="- Selected -"
                                    itemValue = "" />
90                              <f:selectItems value="#{typehallBean.
                                    selectOneItemsTypeHall}" />
91                          </p:selectOneMenu>
92
93                          <h:outputText value="Name:" />
94                          <p:inputText value="#{hallBean.selectedHall.
                                hallname}" size="40"/>
95
96                          <f:facet name="footer">
97                              <p:separator/>
98                              <p:commandButton id="btnUpdateAccept"
                                    update=":formDataTable, :msgs"
                                    oncomplete="PF('dialogHallUpdate').hide
                                    ()" action="#{hallBean.btnUpdateHall(
                                    actionEvent)}" icon="icon-save" title="
                                    Save" value="Save"/>
99                              <p:commandButton id="btnUpdateCancel"
                                    oncomplete="PF('dialogHallUpdate').hide
                                    ()" icon="icon-cancel" title="Cancel"
                                    value="Cancel"/>
100                         </f:facet>
101                     </h:panelGrid>
102                 </p:dialog>
103             </h:form>
104             <h:form id="formDelete">
105                 <p:confirmDialog id="confirmDialog" message="Are you
                        sure to delete the Hall?" showEffect="fade"
                        hideEffect="explode"
106                             header="Delete Hall" severity="alert"
                                widgetVar="dialogHallDelete">
107                     <h:inputHidden value="#{hallBean.selectedHall.
                            idhall}"/>
108                     <p:commandButton id="confirm" value="Accepted"
                            update=":formDataTable:, :msgs" oncomplete="PF
                            ('dialogHallDelete').hide()"
109                                 action="#{hallBean.btnDeleteHall(
                                        actionEvent)}" icon="icon-check
                                        "/>
110                     <p:commandButton id="decline" value="Canceled"
                            onclick="PF('dialogHallDelete').hide()" type="
                            button" icon="icon-cancel"/>
111                 </p:confirmDialog>
112             </h:form>
113         </ui:define>
114     </ui:composition>
115
116 </body>
117 </html>
```