# General Objectives

To design and implement a web-based, multi-user payment service using Java Enterprise Edition (Java EE) technologies

# Requirements

Each user has a single online account whose currency is selected upon registration. An initial pretend amount of money of 1,000 GBP is deposited to the user account on startup. A user can select to have their account in GB Pounds, US dollars or Euros. In that case, the system should make the appropriate conversion to assign the right initial amount of money .

A user can instruct the system to make a direct payment to another user. If this request is accepted (i.e. the recipient of the payment exists and there are enough funds), money is transferred (within a single Java EE transaction) to the recipient immediately. A user should be able to check for notifications regarding payments in their account.

A user can instruct the system to request payment from some other user. A user should be able to check about such notifications for requests for payment. They can reject the request, or, in response to it, make a payment to the requesting user.

Users can access all their transactions, that is, sent and received payments and requests for payments as well as their current account balance.

An administrator can see all user accounts and all transactions.

Currency conversion must be implemented by a separate RESTful web service. The actual exchange rates will be statically assigned (hard-coded) in the RESTful service source code.

# Significance

The assignment shall boost the learning and demonstrate the ability to design and implelent user interfaces using Java Server Faces (JSF), design and implement businsess logic using Enterprise Java Beans (EJBs), and design and implelement a secure mult-user system.

# Theory

Java Enterprise Edition (Java EE) , now called Jakarta EE is a platform that provides developers with enterprise features such as distributed computing and web services [1]. Java EE was been accredited to simplifying the need of programming by creating standardized, reusable modular components [2]. Java EE applications can be hosted on application servers, and can be accessed by IoT devices, smartphones and microservices running in Docker containers.

Java Server Face (JSF), Enterprise Java Bean (EJB) and Java Persistence API (JPA) are important aspects of Java EE that can be put together to fit a Model-View-Controller (MVC) model [2]. This can help facilitate the creation of powerful, robust and flexible applications. The MVC model using the Java EE components can be represented as shown:

**Model**

JSF Backing Beans

**Controller**

Primeface (xhtml) Components on browser

**View**

Database

Java and the database are architecturally different. Java is based on objects and classes, while the database is based on queries. As such, an inconvenience exists in handling the database directly from the JDBC. JPA exists to bridge the differences between Java and the database. With this, a developer can access the database by objects using the JPA. This in turn reduces boiler-plate code, and increases efficiency.

EJB is responsible for performing life cycle management, security, transaction management and object pooling [3]. The component is used to enable development of distributed applications with the EJB being the remote component, creating scalable applications by facilitating load balance, clustering and fail-over, and encapsulating business logic to separate it from the presentation layer. Three types of beans exist: session, message-driven and entity.

JSF is a server-side component-based user interface framework [4]. JSF provide components such as input boxes as well as APIs that can be use to manage the states of the components. Using JSF, presentation and business logic can be separated. JSF also provides a rich set of components and methods for accessing and managing them.

Primefaces are the UI components rendered to the user’s browser. These components are HTML-like, written in XHTML.

# System Specifications

The project was divided into six layers: the web layer, the business layer, the data persistence layer, the security layer, the web service layer, the RPC with Apache Thrift layer. This chapter discusses the specifications of the project according to each layer.

## The Web Layer

This contains .xhtml (facelets) pages through which users and administrators interact with the system. The user should be able to: view all their transactions, make direct payments to other registered users and request payments from other registered users. The administrator should be able to see all user accounts, view payment transactions and register new administrators.

## The Business Layer

This layer is to be comprised of Enterprise Java Beans (EJBs) that implement business logic for the system. Container-managed transactions are to be used. Data access to the persistence layer is to only take place through the EJBs.

## The Data Persistence Layer

This is comprised of a JavaDB (Derby) database and JPAs. The database is to be deployed from the JPA definitions, and access to the database is to only take place via the JPAs rather than directly using JDBC.

## The Security Layer

The security layer is to accomplish authorization of users into the system. The roles administrator and user are to be defined. Form-based authentication is to be implemented using jdbcRealm. Communication should take place using HTTPs too.

## The Web Service

This is to implement a RESTful web service that allows conversion between the different currencies supported by the platform. The following path is to be used to access the web service:

baseURL/conversion/{currency1}/{currency2}/{amount\_of\_currency1}

JAX-RS client is to access the web service to make use of the conversion facility.

## RPC with Apache Thrift

This is to provide a date and time for which a transaction is taking place. The service should run on a separate thread on a specified available port.

# Implementation and Results

This section documents the specifications that were achieved or partially achieved in this project.

## The Web Layer

The following were achieved:

Registration of a new user was achieved using the JSF ‘register.xhtml’ alongside its backing bean ‘RegisterView.java’. The new users were able to enter their username, password, and desired currency. Through a post validate event, the system was able to validate that: a user with similar username didn’t exist, the entered password and confirmation password matched. On selecting his/her desired currency, the starting amount of 1,000 GBP was converted and displayed automatically to the new value using AJAX listener.

Sending of money for a user was achieved using the JSF ‘ send\_money.xhtml’ and its backing bean ‘SendView.java’. Though post event listener, the user was alerted if: the amount entered was greater than his balance, or when the user sending money to didn’t exist in the system.

Viewing of transactions was achieved using the JSF ‘transactions.xhtml’ and its backing bean ‘TransactionsView.java’. The user was presented with two tables showing the ingress and egress transactions of the user.

Requesting of money was achieved using the JSF ‘request\_money.xhtml’ and its backing bean ‘RequestMoney.java’. Through post validate listener, the user was alerted if the user from whom he/she was requesting money didn’t exist in the system.

Accepting/Rejecting of money request was achieved using the JSF ‘received\_request.xhtml’ and its backing bean ‘ReceivedRequestView.java’.

Viewing of all users by the administrator was achieved through the JSF ‘see\_users.xhtml’ and its backing bean ‘SeeUserView.java’.

Viewing of all transactions made by a user by an administrator was achieved through the JSF ‘user\_transactions.xhtml’ and its backing bean ‘TransactionsViewByAdmin.java’. From the list of users, the administrator was able to select a particular user to view transactions of. Parameters of the selected user were first passed from the ‘see\_users.xhtml’ page to it’s backing bean ‘SeeUserView.java’, from where the bean opened the ‘user\_ytransaction.xhtml’ JSF, passing the parameters for the selected user via GET parameters.

Registration of a new administrator by an existing administrator was achieved using the JSF ‘add\_admin.xhtml’ and its backing bean ‘AddAdminView.java’. Validation was done to ensure an admin with a similar username didn’t exist.

## The Business Layer

Data access to the persistence layer was done only using EJBs in this project. The following EJBs were defined for this project through the following files:

**‘InitializeAdminEJB.java’**

Facilitated initialization of the first admin acoint ‘admin1’. Was a singleton EJB executed during startup of the web application using the @Singleton, @Startup and @PostConstruct annotations.

**‘UserEJB.java’**

This EJB facilitated registration of a new user, retrieval of a particular user’s information and getting a list of all users as needed by the admin.

**‘SendMoneyEJB.java’**

Sending of money from one account to another was achieved using this.

**‘MoneyTransaferEJB.java’**

Through this EJB, the transactions of a specified user where able: to be retrieved through the relevant JPA, get their account balance, get notifications of new unread received money transfers and mark the transactions as being read.

**‘MoneyRequestEJB.java’**

This was used to request for money, get notifications of money requests made to a user, accept/reject money requests.

## The Data Persistence Layer

JavaDB (Derby) was used in this project. Creation of the database was automatically done on deployment of the web application through JPA definitions. Four JPAs were defined for this project through the following files:

**‘User.java’**

Dealt with user information. The following attributes were defined for the JPA: username, password, and currency. The username was the primary key. Two named queries were defined to: fetch all users and find user with a given username.

**‘Group.java’**

Dealt with group information. The following attributes were defined for the JPA: username and groupname. The username was the primary key. The groupname defined which group the user belonged to: user or admin.

**‘MoneyTransfer.java’**

Dealt with money transfer records. The following attributes were defined for the JPA: id, sender, recipient, amount, checked. Id was the primary key.

Six named queries where defined in the JPA to: get user’s egress transfer, get user’s ingress transfers, get notifications for ingress transfers, mark ingress transfer as being checked and get the amount of money sent and received for a user.

## The Security Layer

Form-based authorization was achieved for logging in. JDBCRealm was used to achieve this. Administrators were able to view their pages which users couldn’t view, and converse for users. Constraints were defined to achieve this. Sessions were used to facilitates persistence of user information on login. Users and administrators were able to logout of the system at their will.

HTTPS was not achieved in the project.

Declarative security to restrict access to EJB methods was also not achieved in the project.

## The Web Service

Currency conversion was achieved using a RESTful web service. A JAX-RS client was coded to access the web service. The specified URI structure was used in the web service.

## RPC with Apache Thrift

RPC with Apache Thrift was not achieved in the project.

# User Manual

A video walkthrough of the assignment was provided. The video shows all the steps of a user from: registering, sending money, requesting for money, viewing transactions, accepting/rejecting money requests and viewing notifications for money received and money requests. The video also show the admin processes: viewing all users, viewing a user’s transactions and registering a new admin. The video ends showing how access is restricted for unauthorized pages according to role.

# Conclusion

This assignment was able to create a payment service web application using Java EE by using JSFs, EJBs and JPAs. A user was able to register and login into the system. GB Pound, US dollar and Euro currencies were supported by the system, with a RESTful web service and a JAX-RS client facilitating conversions between different currencies. An equivalent of 1,000 GB Pound was initially deposited to the user account on registration, from an admin account created via a singleton EJB on application deployment.

Users were able to send money to other registered users, view their transactions, request for money from other users, accept/reject money requests from other users, and view notifications for new ingress money transfers and money requests. Admins were able to register other admins, view all registered users and view transactions for selected users.

Login was form-based, relying on jdbcRealm to perform authentication. Users with user role where unable to access admin files.

The assignment was however unable to accomplish: HTTPS access of the system and RPC with Thrift server. Overall, the assignment provided an invaluable learning chance and a bulk of the specifications were met.

# References

|  |  |
| --- | --- |
| [1] | jatatpoint, "Java EE," Javatpoint, [Online]. Available: https://www.javatpoint.com/java-ee. [Accessed 08 04 2022]. |
| [2] | M. Debnath, "Integrating JSF, EJB, and JPL: A Case Study," developer.com, 24 11 2014. [Online]. Available: https://www.developer.com/java/integrating-jsf-ejb-and-jpl-a-case-study/. |
| [3] | javatpoint, "What is EJB," javatpoint.com, [Online]. Available: https://www.javatpoint.com/what-is-ejb. [Accessed 08 4 2022]. |
| [4] | javatpoint, "JavaServer Faces," javatpoint, [Online]. Available: https://www.javatpoint.com/what-is-jsf. [Accessed 8 4 2022]. |