Web Application and Services: Online Payment Services- A1

Candidate number: 245308

Name: Joseph Steven Semgalawe

Submission date: 6th May 2022

Contents

System Architecture and requirements

Each section explains what has been done for each of the requirements based on the framework layers.

Web layer

The web layer consists of the view and controller components of the application.

Both the users and admins required a set of views and controller classes.

The following views and backing beans were created for both the users and admins routes.

/Web Pages

For Users

/users

- dashboard.xhtml – initial view for navigating the provided services
- edit.xhtml – view for editing their user information
- pals.xhtml – view for viewing their list of friends or 'pals'
- transactions.xhtml – view for viewing list of transactions.

/transaction

- request.xhtml – view for handling request transactions
- transfer.xhtml –view for handling transfer transactions

For Admins

/admins

- create.xhtml – view for create new users (admins or users)
- dashboard.xhtml – initial view for navigating to the available functionality as admin
- edit.xhtml – view for updating existing user information
- list.xhtml – view for viewing a paginated list of all users. Also provides ability to edit and destroy users
- transaction.xhtml – view for listing all user transactions
- view.xhtml – view for reading specific user information

Public

Available for both users and admins

/

- login.xhtml – view for logging in user both admins and users
- registration.xhtml – view for creating new users (cannot create admins)
- template.xhtml – view component for creating reusable views

Each .xhtml has been linked explicitly with navigation outcomes for the form POST requests and responses as well GET requests. Routes that have the POST-Redirect-GET have been handled appropriately using the <Redirect /> statement.

Below is a snapshot of the pageFlow created by faces.config.xml - proof and brevity:

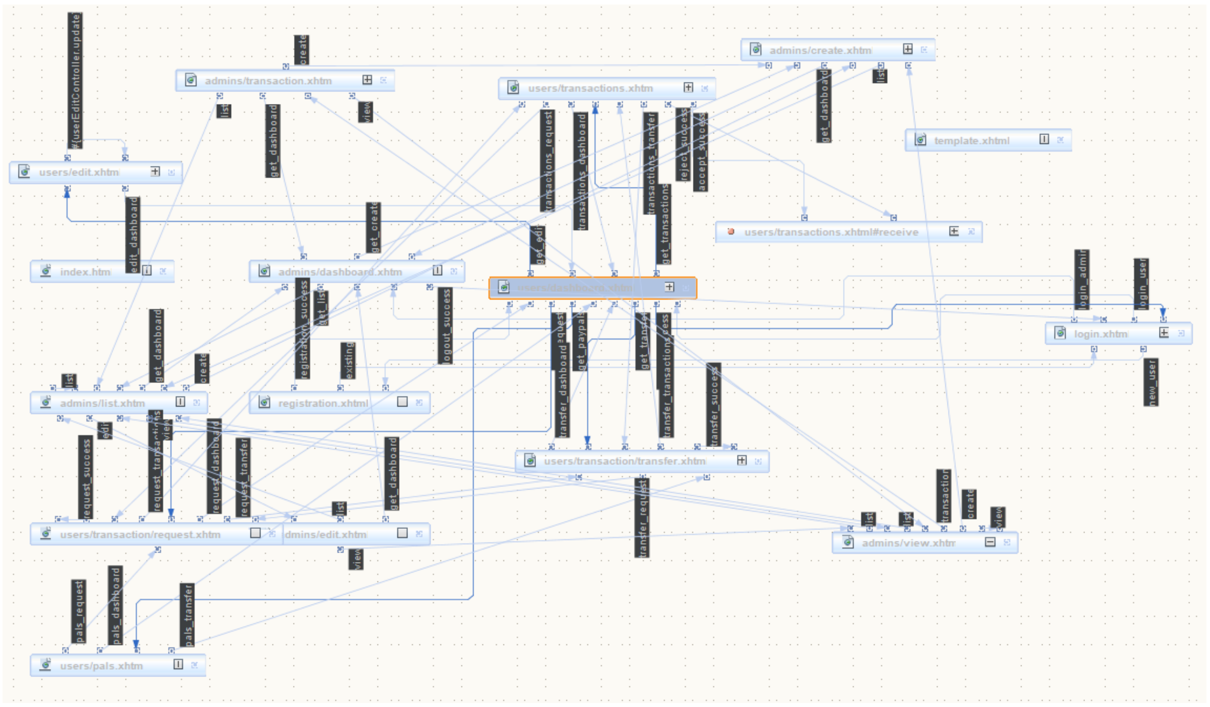


Figure 1: PageFlow 'faces.config.xml'

Backed beans – CDI

Each view was backed with jsf controller class for handling server POST and GET requests.

Below is the list of the backed beans implemented and the views they support.

Classes can be found under the webapps2022.jsf

- LoginController.java
  - This class deals user inputs for the login.xhtml view
  - It contains appropriate EJB classes to communicate with the database for login and validation handling.
- PayPalController.java
  - This class deals with all the admin views.
  - It links with various EJB classes to modify, read, and remove elements from the database.
- RegistrationController.java
  - This class deals with user input for the registration.xhtml
  - It also contains appropriate functionality for user registration and validation responses.

- UserDashboardController.java
    - This class deals with user dashboard navigation to and from other views and their functionalities
    - As well as managing user logout
- UserEditController.java
    - This class handles all every input involved in the edit.xhtml
    - It involves validating currency conversions for users updating or transferring their currency
    - Appropriate EJB functionality used to update user details
- UserPalsController.java
    - This class is handles list functionality for users' friends or pals
    - Most of the pagination for the pals is implemented in here
- UserTransactionController.java
    - This class handles creating of both request and transfer transactions
    - Appropriate EJB is used to POST and update database when user enters or issues transaction.
    - ConversionServices for the currencies are also handled in here with the help of the RestAPI implemented.
- UserTransactionsController.java
    - This class is used to view user transactions both received and sent.
    - Each transaction is paginated and handled in this class
    - Appropriate EJB classes are injected to handle incoming requests i.e., approval or rejection

Validations constraints are applied and handled using the <h:messages /> tag. Personalized errors are added to the life-cycle process using a custom JsfUtil class.

Business Layer

The business layer consists of entity beans used to communicate business logic with JPA persistence managed container.

Each bean was created for each entity in the 'webapps2022.entity' package.

An abstract interface was created to link common persistence methods i.e., em.persist(Object o) can be ejb.create(T o) and can be overridden and implemented to fit each of the entities requirements.

The business layer consists of class functionality from different EJB's to support the following requirements

For the users:

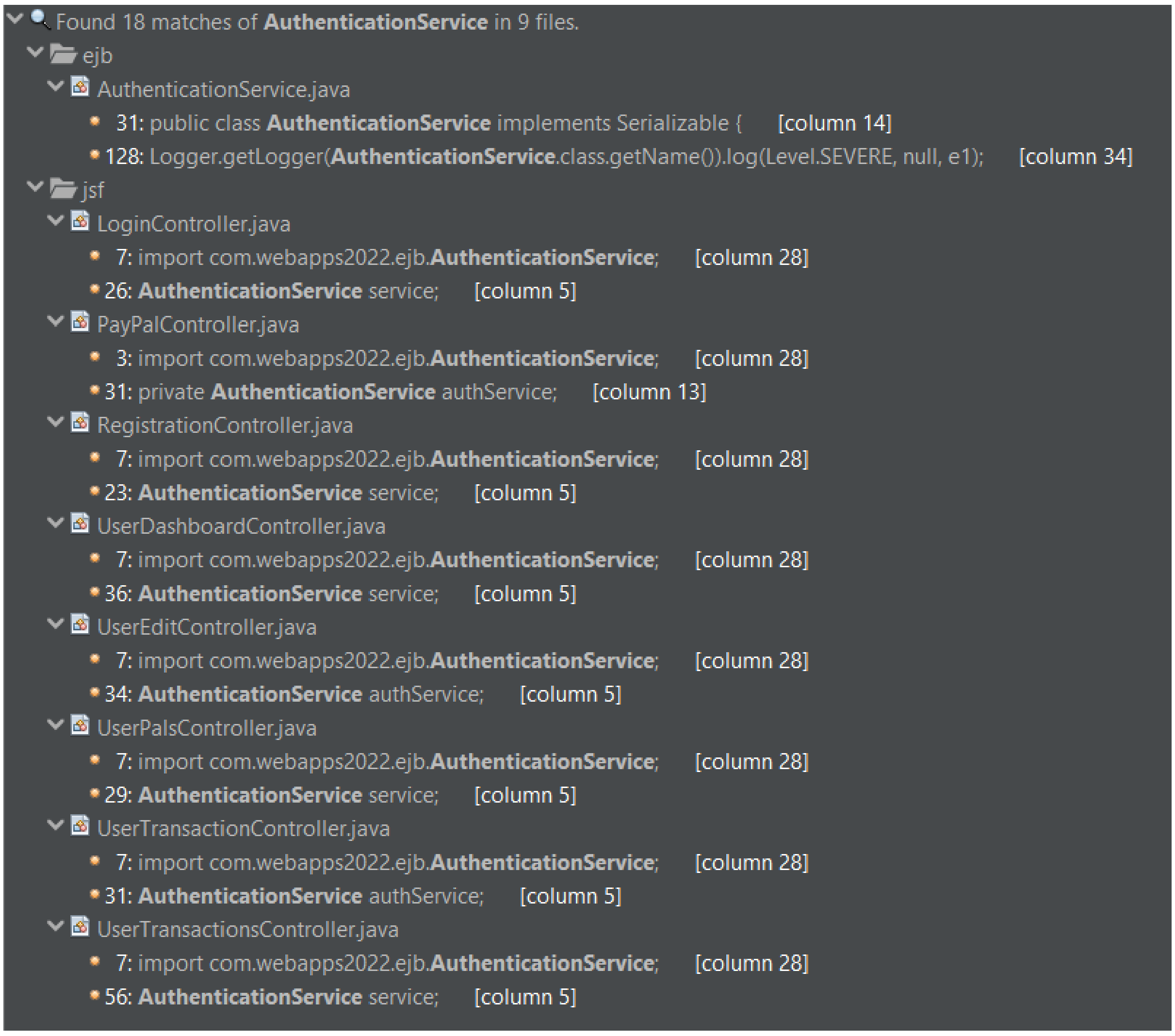
- Viewing all transactions
- Making transactions direct and debit request
- User persistence i.e., CRUD

For the admins

- View all user accounts and balances
- View all payment transactions
- Register more administrators

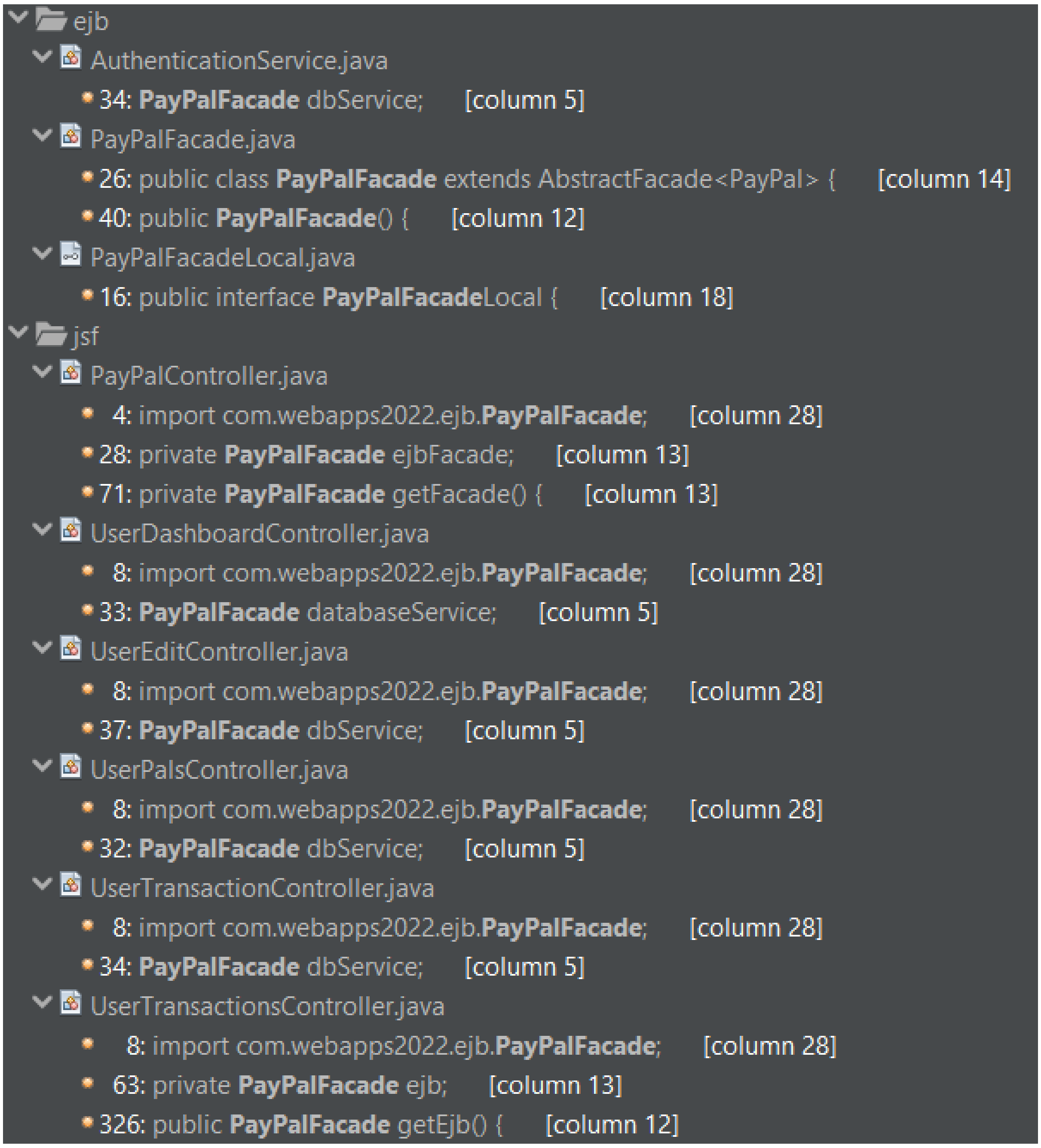
Below is information about each of the entity beans class implemented:

- AuthenticationService
    - This enterprise bean is used for registration and login of user.
    - It contains other functionality such as log out.
    - It handles both admin creation and user



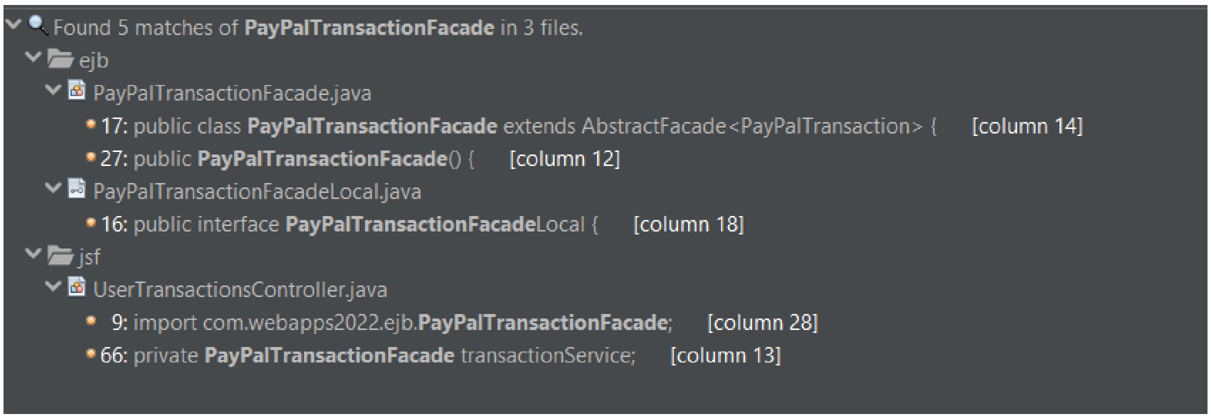
```
Found 18 matches of AuthenticationService in 9 files.
  ejb
    AuthenticationService.java
      31: public class AuthenticationService implements Serializable {      [column 14]
      128: Logger.getLogger(AuthenticationService.class.getName()).log(Level.SEVERE, null, e1);      [column 34]
  jsf
    LoginController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      26: AuthenticationService service;      [column 5]
    PayPalController.java
      3: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      31: private AuthenticationService authService;      [column 13]
    RegistrationController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      23: AuthenticationService service;      [column 5]
    UserDashboardController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      36: AuthenticationService service;      [column 5]
    UserEditController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      34: AuthenticationService authService;      [column 5]
    UserPalsController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      29: AuthenticationService service;      [column 5]
    UserTransactionController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      31: AuthenticationService authService;      [column 5]
    UserTransactionsController.java
      7: import com.webapps2022.ejb.AuthenticationService;      [column 28]
      56: AuthenticationService service;      [column 5]
```

- PayPalFacade
  - This bean handles all persistence logic relating to the user
  - It also contains methods for handling or querying different user information such as transactions, list of pals etc.
  - Class can be used to modify, and user entity provided



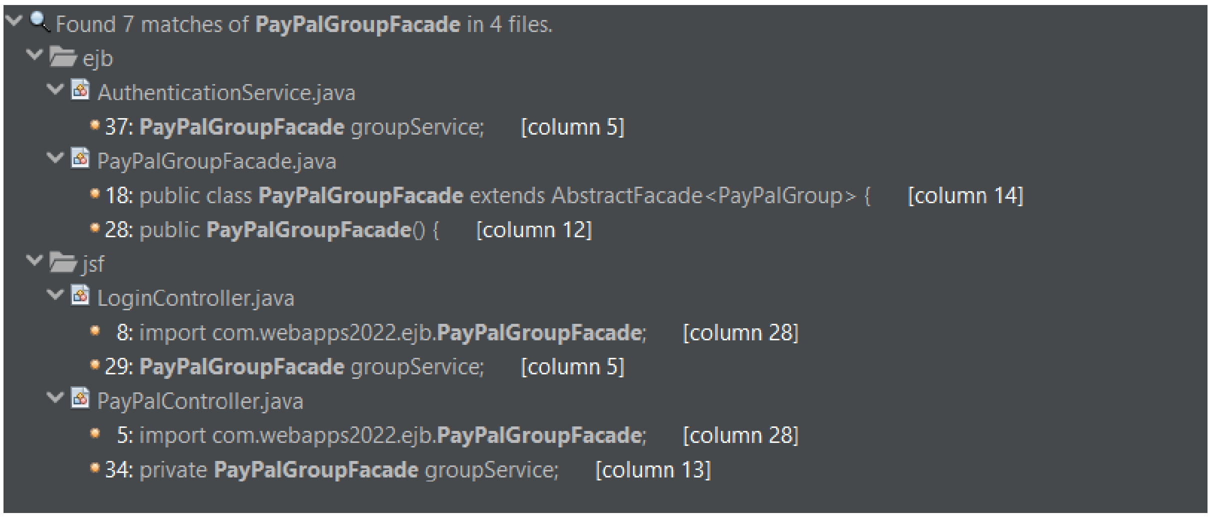
```
ejb
  AuthenticationService.java
     34: PayPalFacade dbService;    [column 5]
  PayPalFacade.java
     26: public class PayPalFacade extends AbstractFacade<PayPal> {    [column 14]
     40: public PayPalFacade() {    [column 12]
  PayPalFacadeLocal.java
     16: public interface PayPalFacadeLocal {    [column 18]
jsf
  PayPalController.java
      4: import com.webapps2022.ejb.PayPalFacade;    [column 28]
     28: private PayPalFacade ejbFacade;    [column 13]
     71: private PayPalFacade getFacade() {    [column 13]
  UserDashboardController.java
      8: import com.webapps2022.ejb.PayPalFacade;    [column 28]
     33: PayPalFacade databaseService;    [column 5]
  UserEditController.java
      8: import com.webapps2022.ejb.PayPalFacade;    [column 28]
     37: PayPalFacade dbService;    [column 5]
  UserPalsController.java
      8: import com.webapps2022.ejb.PayPalFacade;    [column 28]
     32: PayPalFacade dbService;    [column 5]
  UserTransactionController.java
      8: import com.webapps2022.ejb.PayPalFacade;    [column 28]
     34: PayPalFacade dbService;    [column 5]
  UserTransactionsController.java
      8: import com.webapps2022.ejb.PayPalFacade;    [column 28]
     63: private PayPalFacade ejb;    [column 13]
    326: public PayPalFacade getEjb() {    [column 12]
```

- PayPalTransactionFacade
    - Like the above only difference is that instead of being used for PayPal entity it used for the transaction entity
    - They all extend from AbstractFacade therefore have close similarities apart from type



```
Found 5 matches of PayPalTransactionFacade in 3 files.
  ejb
    PayPalTransactionFacade.java
      17: public class PayPalTransactionFacade extends AbstractFacade<PayPalTransaction> {     [column 14]
      27: public PayPalTransactionFacade() {     [column 12]
    PayPalTransactionFacadeLocal.java
      16: public interface PayPalTransactionFacadeLocal {     [column 18]
  jsf
    UserTransactionsController.java
       9: import com.webapps2022.ejb.PayPalTransactionFacade;     [column 28]
      66: private PayPalTransactionFacade transactionService;     [column 13]
```

- PayPalGroupFacade
    - Also, similar only difference is the entity type and query operations.
    - Used to fetch the user role for admin reference.
    - Persist user role for jdbc realm group table



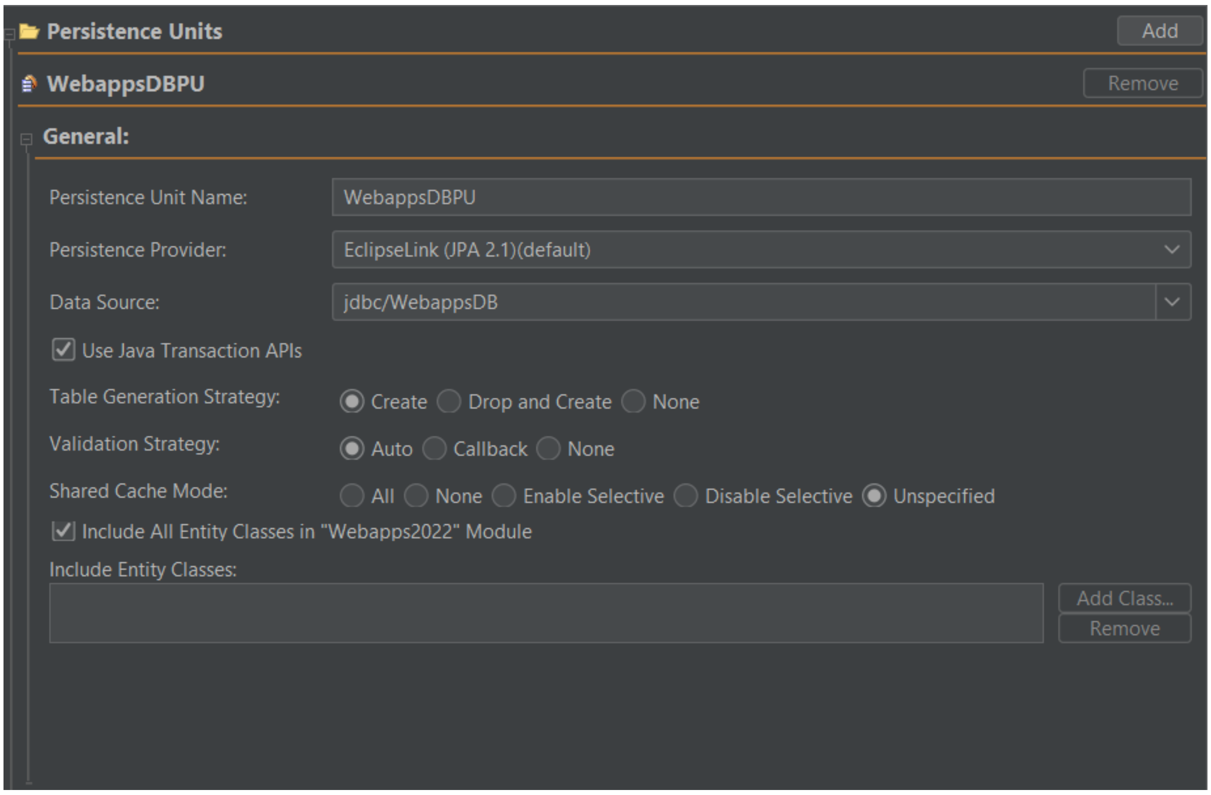
```
Found 7 matches of PayPalGroupFacade in 4 files.
  ejb
    AuthenticationService.java
      37: PayPalGroupFacade groupService;     [column 5]
    PayPalGroupFacade.java
      18: public class PayPalGroupFacade extends AbstractFacade<PayPalGroup> {     [column 14]
      28: public PayPalGroupFacade() {     [column 12]
  jsf
    LoginController.java
       8: import com.webapps2022.ejb.PayPalGroupFacade;     [column 28]
      29: PayPalGroupFacade groupService;     [column 5]
    PayPalController.java
       5: import com.webapps2022.ejb.PayPalGroupFacade;     [column 28]
      34: private PayPalGroupFacade groupService;     [column 13]
```

Data Persistence Layer

This layer refers to the entity-jdbc relationship configuration with the help of JPA.

The following requirements are expected and fulfilled:

- Data model should be written as a set of persistence entities (JPA)
    - All entity implementations can be found under the package "webapps2022.entities"
    - Each entity maps to a table that is defined and configured using annotations
    - The creation the tables is dependent on the persistence.xml configuration which was set to create on deployment.
- Access to the database must always take place through manipulating JPA entities
    - Access to database is carried using managed Persistence container.
    - Each transaction is therefore managed by JPA internally and query to database goes through EntityManager and established persistence context
- Do not access the database directly using JDBC
    - PersistanceContext created and linked with payara server
    - JDBC pool is created and linked to JDBC Resource
    - Configuration can be seen below
- A correctly configured persistence.xml file
    - Below is the persistence.xml configuration for this project



Persistence Units

WebappsDBPU

General:

Persistence Unit Name: WebappsDBPU

Persistence Provider: EclipseLink (JPA 2.1)(default)

Data Source: jdbc/WebappsDB

[✓] Use Java Transaction APIs

Table Generation Strategy: [✓ Create] Drop and Create None

Validation Strategy: [✓ Auto] Callback None

Shared Cache Mode: All None Enable Selective Disable Selective [✓ Unspecified]

[✓] Include All Entity Classes in "Webapps2022" Module

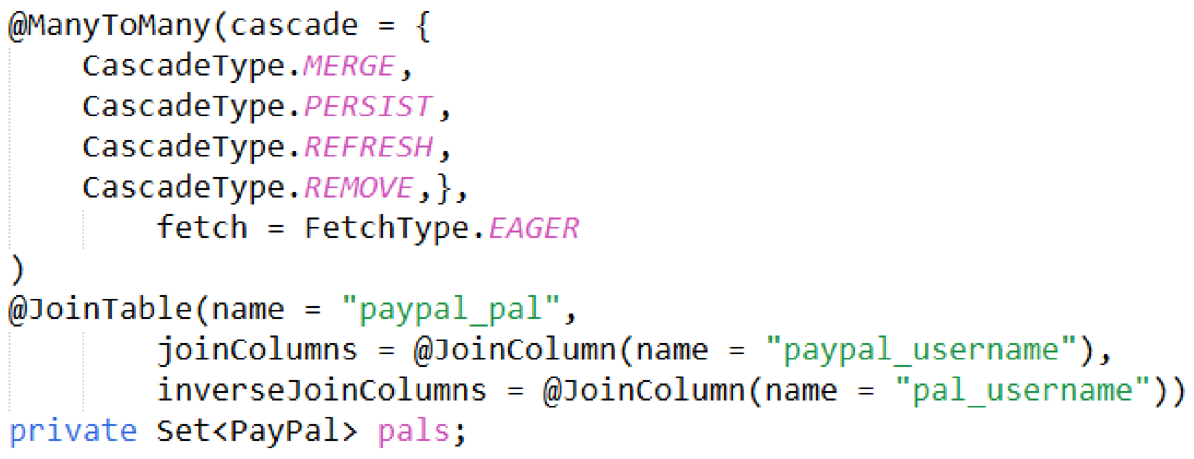
Include Entity Classes:

- Annotations are required to define associations among different entities

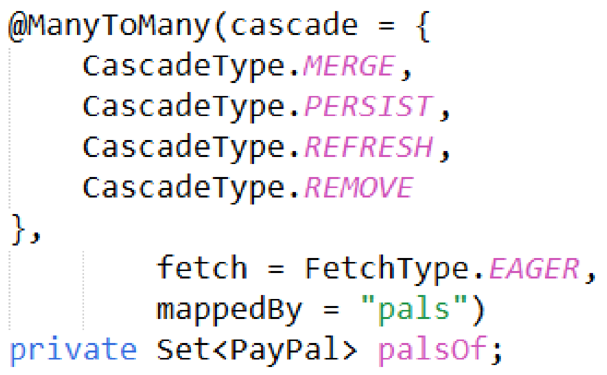
Below are the different relationship snippets implemented for the defined entities

The outline will be done for each entity class for each defined relationship

1. PayPal to PayPal @ManyToMany Relationship
    a. This relationship is created to link users to other users as payment friends
    b. A user can have many friends and friends who are users can also have the user and other users as friends.
    c. When a user is deleted or updated, we cascade the result to all the join table below defining the relationship



```
@ManyToMany(cascade = {
    CascadeType.MERGE,
    CascadeType.PERSIST,
    CascadeType.REFRESH,
    CascadeType.REMOVE,},
        fetch = FetchType.EAGER
)
@JoinTable(name = "paypal_pal",
        joinColumns = @JoinColumn(name = "paypal_username"),
        inverseJoinColumns = @JoinColumn(name = "pal_username"))
private Set<PayPal> pals;
```



```
@ManyToMany(cascade = {
    CascadeType.MERGE,
    CascadeType.PERSIST,
    CascadeType.REFRESH,
    CascadeType.REMOVE
},
        fetch = FetchType.EAGER,
        mappedBy = "pals")
private Set<PayPal> palsOf;
```

2. PayPal to PayPalTransaction @OneToMany – @ManyToOne
    a. A user can have as many transactions associated to it.
    b. However, transactions cannot map to many different users but one
    c. Cascading is also defined in the relationship
    d. Ownership is managed by the user
    e. JoinColumn is used instead of JoinTable like above, so no other table is created

```
@OneToMany(fetch = FetchType.EAGER,
        mappedBy = "user",
        cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REMOVE
@JoinColumn(name = "transaction_fk")
private List<PayPalTransaction> transactions;
// update transactions only when we remove from database and manged state.
// when we add to managed state [to sychornise any changes]
// when we update the object by refreshing to synchonise changes.
```

```
@ManyToOne
@JoinColumn(name = "user_fk", nullable = false)
PayPal user;
```

Apart from the relationship annotation additional validation decorators for JSF and JPA are included. JPA uses @Column or @JoinColumn using the attributes defined in the qualifier definition

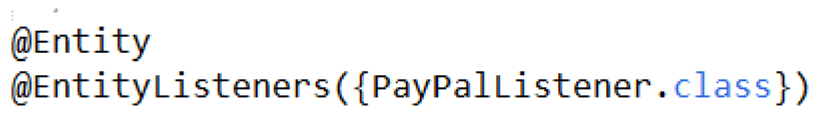
Additional Queries used by implemented EJB's are defined using the @NamedQuries(list{}) and @NamedQuery(name, query)

```
@NamedQueries({
    @NamedQuery(name = PayPalTransaction.GET_STATUS_AS_USER,
            query = PayPalTransaction.GET_STATUS_AS_USER_QUERY),
    @NamedQuery(name = PayPalTransaction.GET_STATUS_AS_RECEPIENT,
            query = PayPalTransaction.GET_STATUS_AS_RECEPIENT_QUERY),
    @NamedQuery(name = PayPalTransaction.COUNT_STATUS_AS_USER,
            query = PayPalTransaction.COUNT_STATUS_AS_USER_QUERY),
    @NamedQuery(name = PayPalTransaction.COUNT_STATUS_AS_RECEPIENT,
            query = PayPalTransaction.COUNT_STATUS_AS_RECEPIENT_QUERY)
})
```

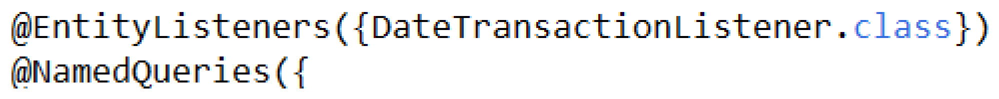
```
@NamedQueries({
    @NamedQuery(name = PayPal.GET_TRANSACTIONS,
            query = PayPal.GET_TRANSACTIONS_QUERY),
    @NamedQuery(name = PayPal.COUNT_TRANSACTIONS,
            query = PayPal.COUNT_TRANSACTIONS_QUERY),
    @NamedQuery(name = PayPal.GET_TRANSACTION,
            query = PayPal.GET_TRANSACTION_QUERY),
    @NamedQuery(name = PayPal.GET_USER,
            query = PayPal.GET_USER_QUERY), // might no be needed with em.find();
    @NamedQuery(name = PayPal.GET_PALS,
            query = PayPal.GET_PALS_QUERY),
    @NamedQuery(name = PayPal.COUNT_PALS,
            query = PayPal.COUNT_PALS_QUERY),
    @NamedQuery(name = PayPal.GET_PAL,
            query = PayPal.GET_PAL_QUERY)
})
```

Entity listeners are also implemented to log and add timestamp functionalities for each of the entities respectively.

Definition can be show below:



```
@Entity
@EntityListeners({PayPalListener.class})
```



```
@EntityListeners({DateTransactionListener.class})
@NamedQueries({
```

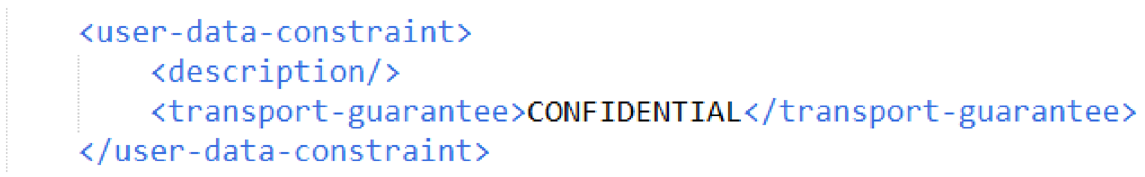
Other definitions have been omitted for brevity.

Security layer

This layer is relating to the security configuration for the implemented web application.

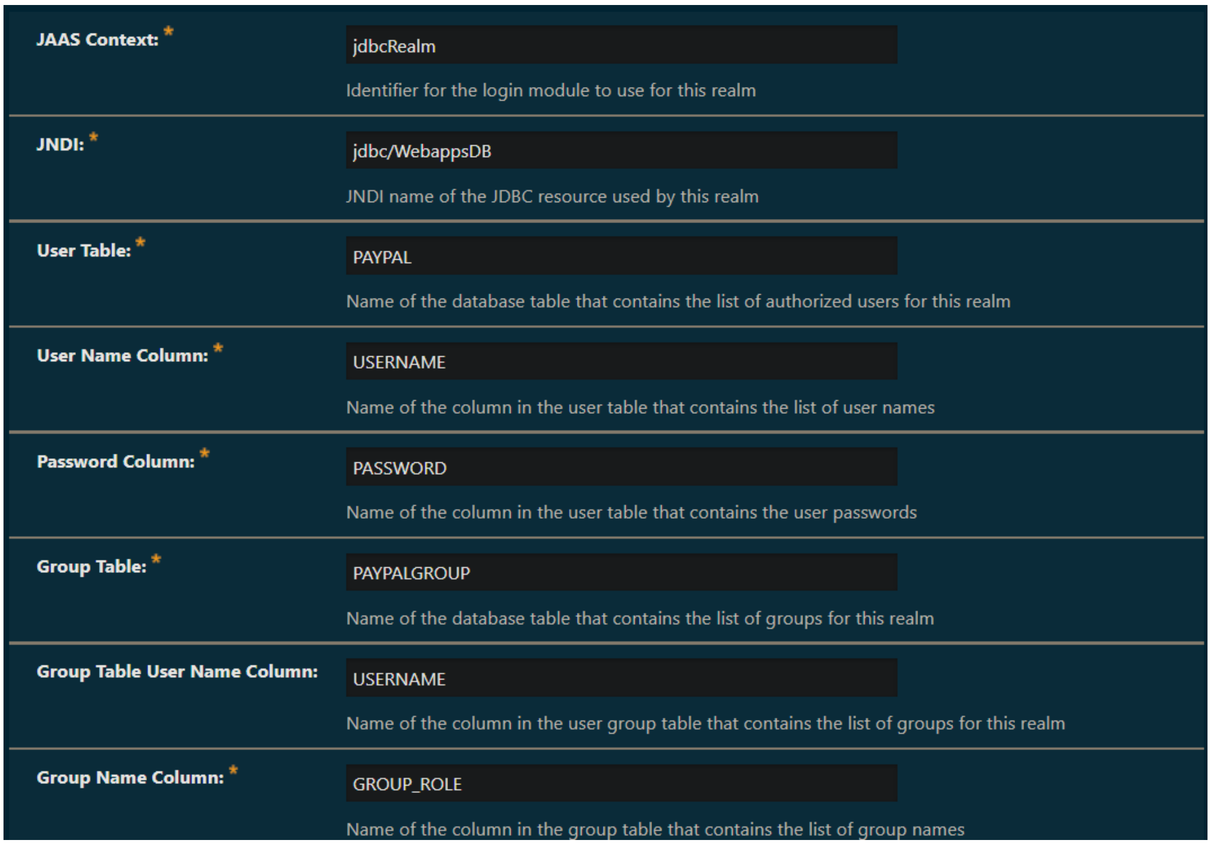
The following requirements were expected:

- Communication over HTTPS
    - This is achieved by defining the user data constraints in web.xml
    - CONFIDENTIAL means that all http requests from port 10000 will be redirected to secure https:8181
    - Chrome advanced settings had to be enabled to allow https without certification



```
<user-data-constraint>
    <description/>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

- Form-based authentication or Basic
    - Form-based authentication was implemented using JDBC realm
    - Realm is defined as WebappsRealm as suggested on the spec

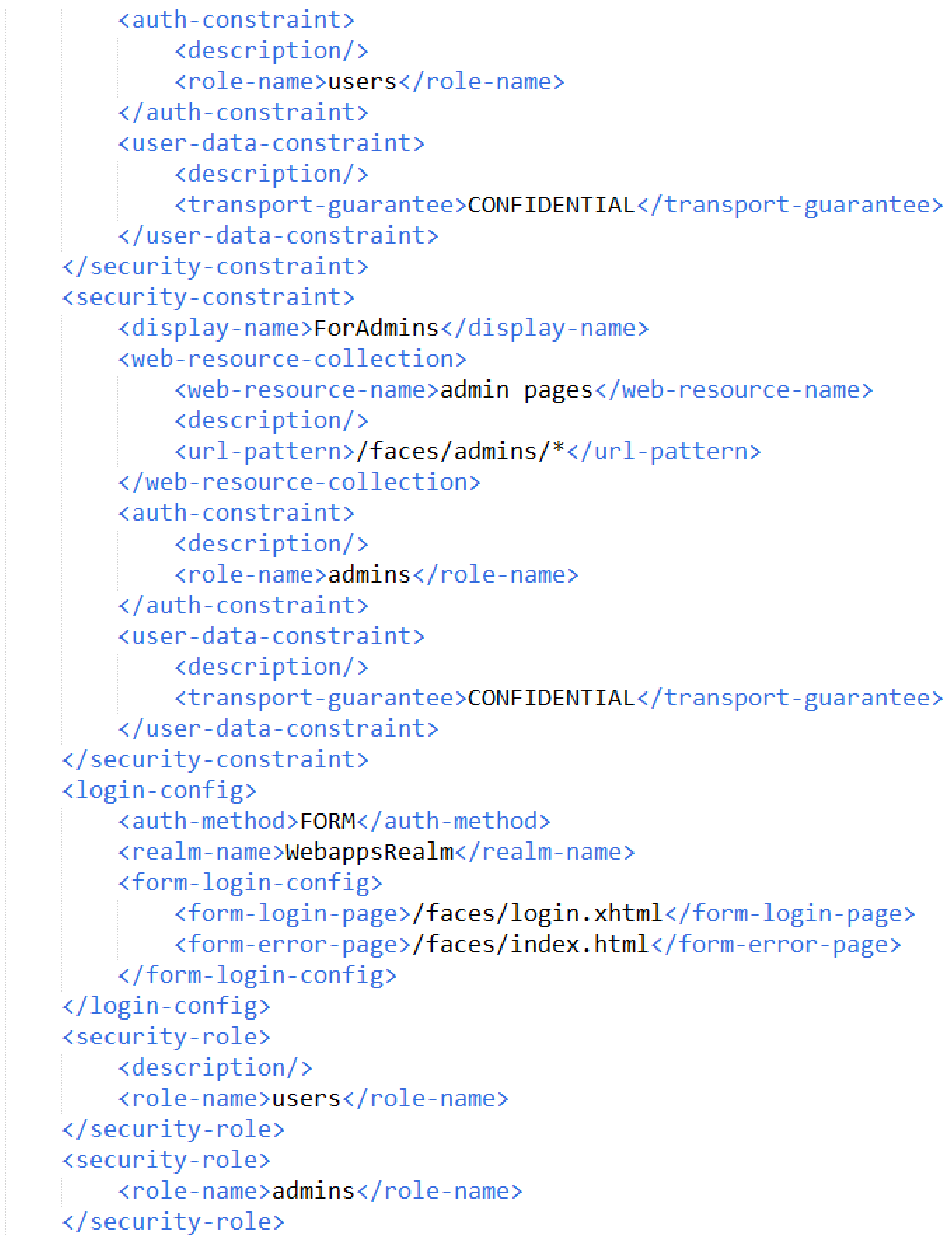


JAAS Context: *	jdbcRealm Identifier for the login module to use for this realm
JNDI: *	jdbc/WebappsDB JNDI name of the JDBC resource used by this realm
User Table: *	PAYPAL Name of the database table that contains the list of authorized users for this realm
User Name Column: *	USERNAME Name of the column in the user table that contains the list of user names
Password Column: *	PASSWORD Name of the column in the user table that contains the user passwords
Group Table: *	PAYPALGROUP Name of the database table that contains the list of groups for this realm
Group Table User Name Column:	USERNAME Name of the column in the user group table that contains the list of groups for this realm
Group Name Column: *	GROUP_ROLE Name of the column in the group table that contains the list of group names

```xml
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>WebappsRealm</realm-name>
    <form-login-config>
        <form-login-page>/faces/login.xhtml</form-login-page>
        <form-error-page>/faces/index.html</form-error-page>
    </form-login-config>
</login-config>
```

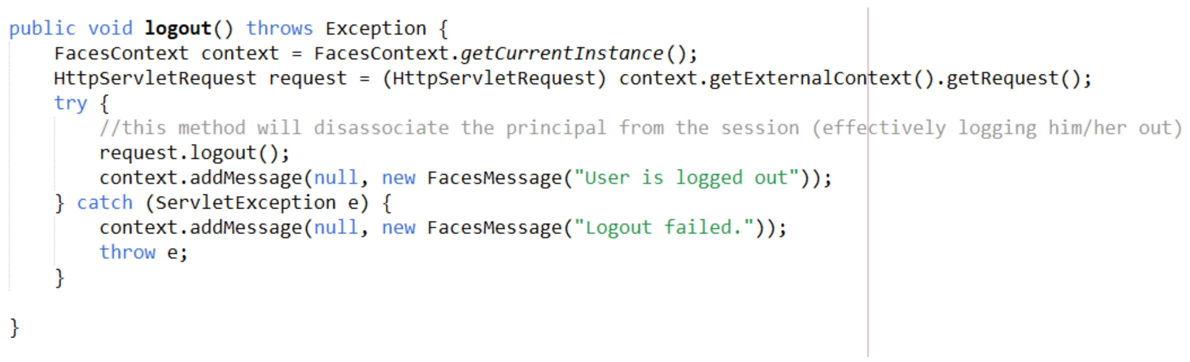
- Declarative web page restriction
    - Security constraints are defined for each user role to protect web page navigation

```xml
<security-constraint>
    <display-name>ForUsers</display-name>
    <web-resource-collection>
        <web-resource-name>user pages</web-resource-name>
        <description/>
        <url-pattern>/faces/users/*</url-pattern>
    </web-resource-collection>
```



```xml
        <auth-constraint>
            <description/>
            <role-name>users</role-name>
        </auth-constraint>
        <user-data-constraint>
            <description/>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <display-name>ForAdmins</display-name>
        <web-resource-collection>
            <web-resource-name>admin pages</web-resource-name>
            <description/>
            <url-pattern>/faces/admins/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <description/>
            <role-name>admins</role-name>
        </auth-constraint>
        <user-data-constraint>
            <description/>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <login-config>
        <auth-method>FORM</auth-method>
        <realm-name>WebappsRealm</realm-name>
        <form-login-config>
            <form-login-page>/faces/login.xhtml</form-login-page>
            <form-error-page>/faces/index.html</form-error-page>
        </form-login-config>
    </login-config>
    <security-role>
        <description/>
        <role-name>users</role-name>
    </security-role>
    <security-role>
        <role-name>admins</role-name>
    </security-role>
```

- Logout
    - Logout is achieved sending a request to the server to logout logged in user.



```java
public void logout() throws Exception {
    FacesContext context = FacesContext.getCurrentInstance();
    HttpServletRequest request = (HttpServletRequest) context.getExternalContext().getRequest();
    try {
        //this method will disassociate the principal from the session (effectively logging him/her out)
        request.logout();
        context.addMessage(null, new FacesMessage("User is logged out"));
    } catch (ServletException e) {
        context.addMessage(null, new FacesMessage("Logout failed."));
        throw e;
    }

}
```

- Declarative security to restrict EJB methods
    - This was unfortunately not implemented.
    - I could not figure out a way to apply @RolesAllowed to the abstract class or the PayPalFacade without preventing both admins and users from creating their accounts.
    - A fix for this with time would have to separate registration for both admin and users to separate methods where one method just does the user registration and the other handles admin account creation rather than a single method taking the user type
- Initial administration registration
    - Manually done by adding one user after persistence creation.
    - I could not figure out how <property name="javax.persistence.schema-generation.create-source" value="script"/> could be linked to a script that adds onto the created tables without creating a script that creates all the tables to include the admin account.
    - Again, with time this could have been solved.

Web Service Layer

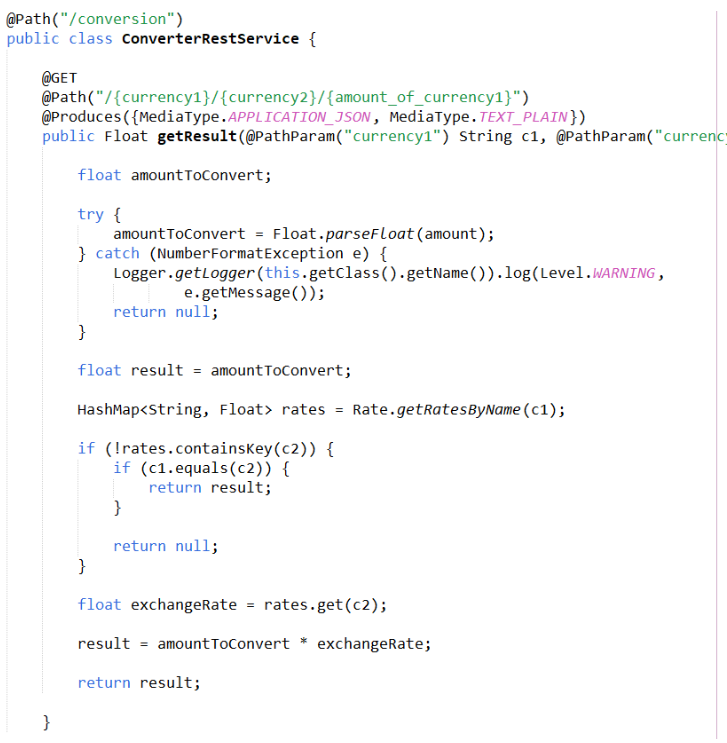
This layer is used to implement the currency conversion api. Deployed on the same server accessed through HTTP.

The resource is exported as /conversion. The base URL is http://localhost:10000/api

The following were the requirements

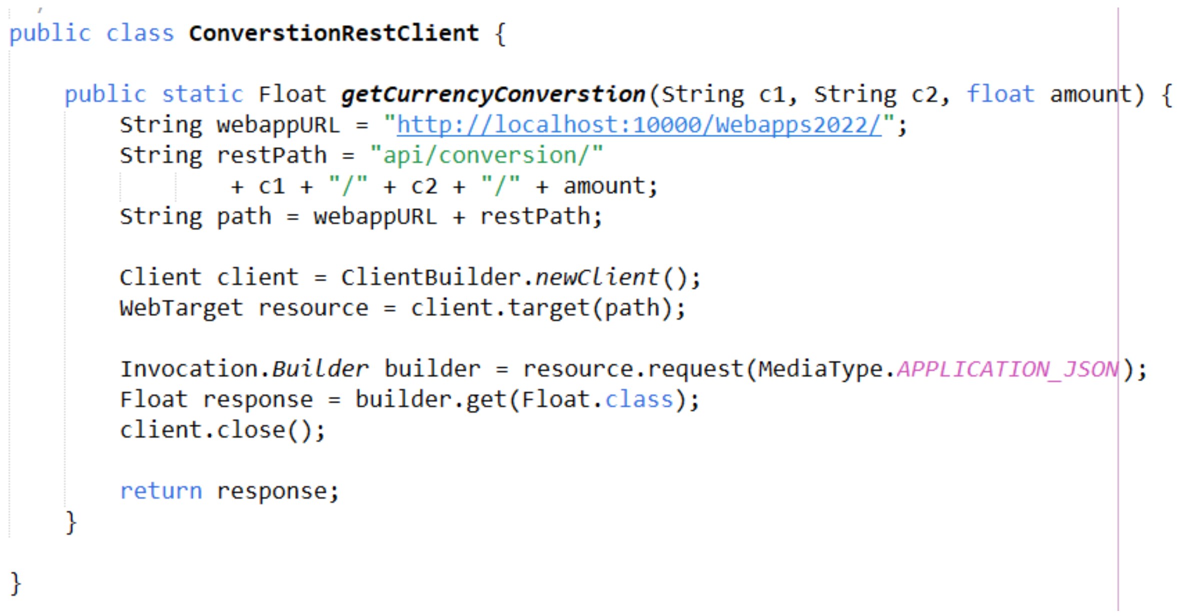
- The resource is exported as /conversion following this URL; *baseURL/conversion/{currency1}/{currency2}/{amount_of_currency1}*

    - The rest server and client were implemented inside the “websapps2022.restservice”



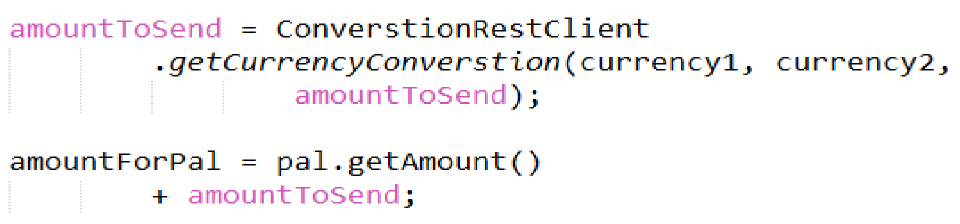
```
@Path("/conversion")
public class ConverterRestService {

    @GET
    @Path("/{currency1}/{currency2}/{amount_of_currency1}")
    @Produces({MediaType.APPLICATION_JSON, MediaType.TEXT_PLAIN})
    public Float getResult(@PathParam("currency1") String c1, @PathParam("currency2") String c2, @PathParam("amount_of_currency1") Strin

        float amountToConvert;

        try {
            amountToConvert = Float.parseFloat(amount);
        } catch (NumberFormatException e) {
            Logger.getLogger(this.getClass().getName()).log(Level.WARNING,
                    e.getMessage());
            return null;
        }

        float result = amountToConvert;

        HashMap<String, Float> rates = Rate.getRatesByName(c1);

        if (!rates.containsKey(c2)) {
            if (c1.equals(c2)) {
                return result;
            }

            return null;
        }

        float exchangeRate = rates.get(c2);

        result = amountToConvert * exchangeRate;

        return result;

    }
```

- Implement Rest Client using JAX-RS
    - ConvertionRestClient class handles this requirement



```
public class ConverstionRestClient {

    public static Float getCurrencyConverstion(String c1, String c2, float amount) {
        String webappURL = "http://localhost:10000/Webapps2022/";
        String restPath = "api/conversion/"
                + c1 + "/" + c2 + "/" + amount;
        String path = webappURL + restPath;

        Client client = ClientBuilder.newClient();
        WebTarget resource = client.target(path);

        Invocation.Builder builder = resource.request(MediaType.APPLICATION_JSON);
        Float response = builder.get(Float.class);
        client.close();

        return response;
    }

}
```

- Consumed by Backed-bean using ConvertionRestClient
    - Consumed by UserTransactionController and other classes



```
amountToSend = ConverstionRestClient
        .getCurrencyConverstion(currency1, currency2,
                amountToSend);

amountForPal = pal.getAmount()
        + amountToSend;
```

RTC with Apache Thrift

Did not attempt this due to time.

User manual

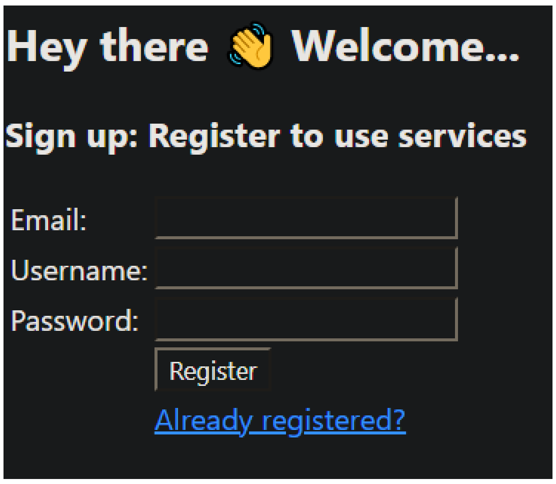
This section provides screenshots and details about what each view does for user reference

Users’ registration

This view handles user registration.

URL: /faces/registration.xhtml

When you initially visit the page without submission you should see the following



Hey there 👋 Welcome...

Sign up: Register to use services

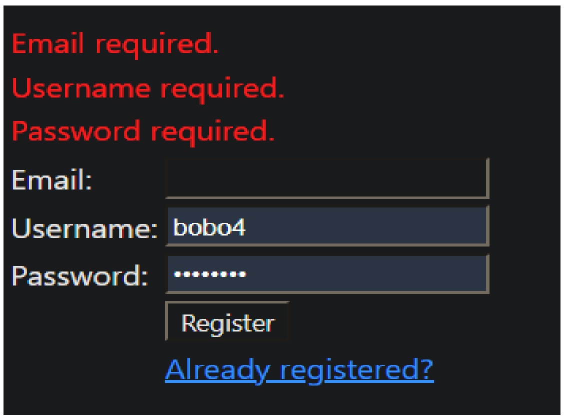
Email:

Username:

Password:

Here you can fill in the required details and register or if you are already register you can go to the login page via the link.

All the inputs are required so if you miss one or all of them the following errors will pop up respectively.



Email required.
Username required.
Password required.

Email:

Username: bobo4

Password: •••••••••

Ignore the filled forms this is because edge or chrome will autofill saved passwords. So, upon refresh this should not show up for you.

When successful user will be login in automatically and the dashboard view should be shown.

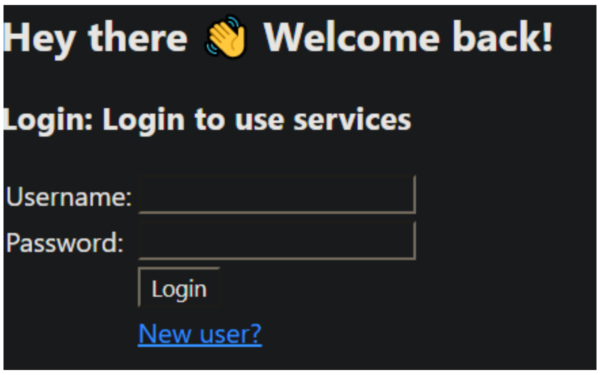
Please refer to user dashboard

Users' login

This view handles user login for both admins and users

URL: /faces/login.xhtml

Initially the view is loaded as shown below



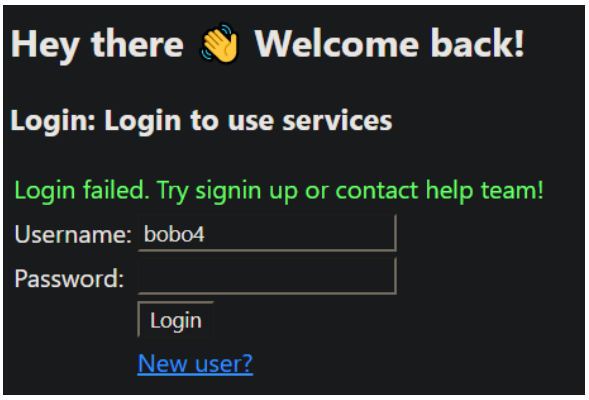
Hey there 👋 Welcome back!

Login: Login to use services

Username:

Password:

On failed login i.e., incorrect password or user does not exist the following error will be returned.



Hey there 👋 Welcome back!

Login: Login to use services

Login failed. Try signin up or contact help team!

Username: bobo4

Password:

bobo4 in this example is an admin user therefore he will be redirected to the admin dashboard on success. Please refer to user dashboard

If account does not exist, then user can navigate to register account using link

Users' dashboard

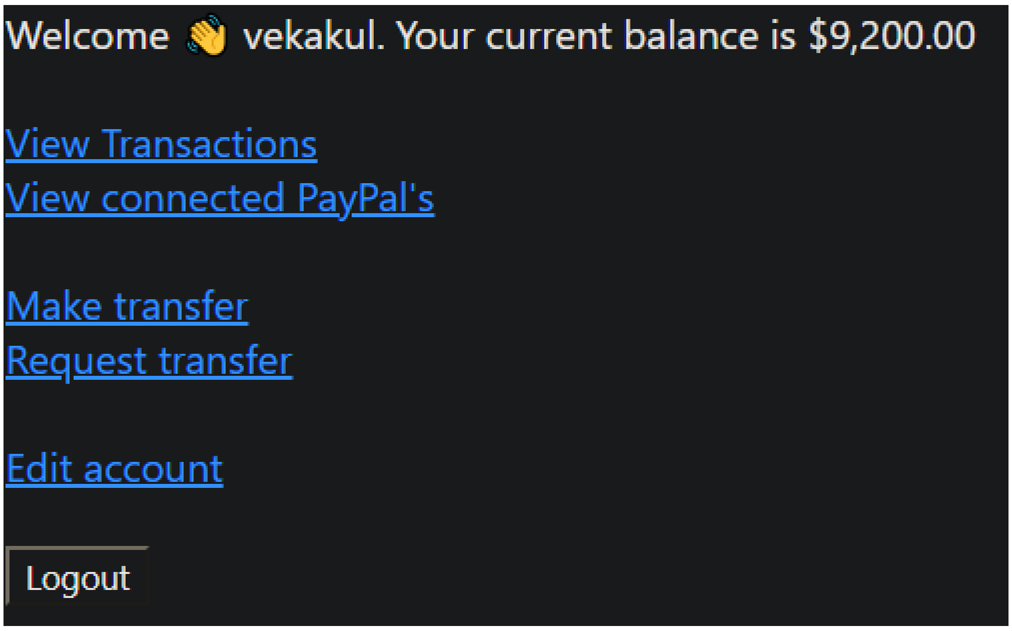
This view handles user's dashboard. Dashboard is different for both admins and users.

As a user you can use dashboard to navigate around.

As normal user you can logout from here as well by pressing the logout button and will be redirected to the login page.

Normal users can make transfers by navigating to /users/transaction/transfer.xhtml using the Make transfer link. The same can be done for request transactions.

Users can also navigate to view all transactions received and sent. Please refer to user transaction


Welcome 👋 vekakul. Your current balance is $9,200.00

View Transactions
View connected PayPal's

Make transfer
Request transfer

Edit account

Logout

Users can also navigate to view their friends list at "View connected PayPal's"

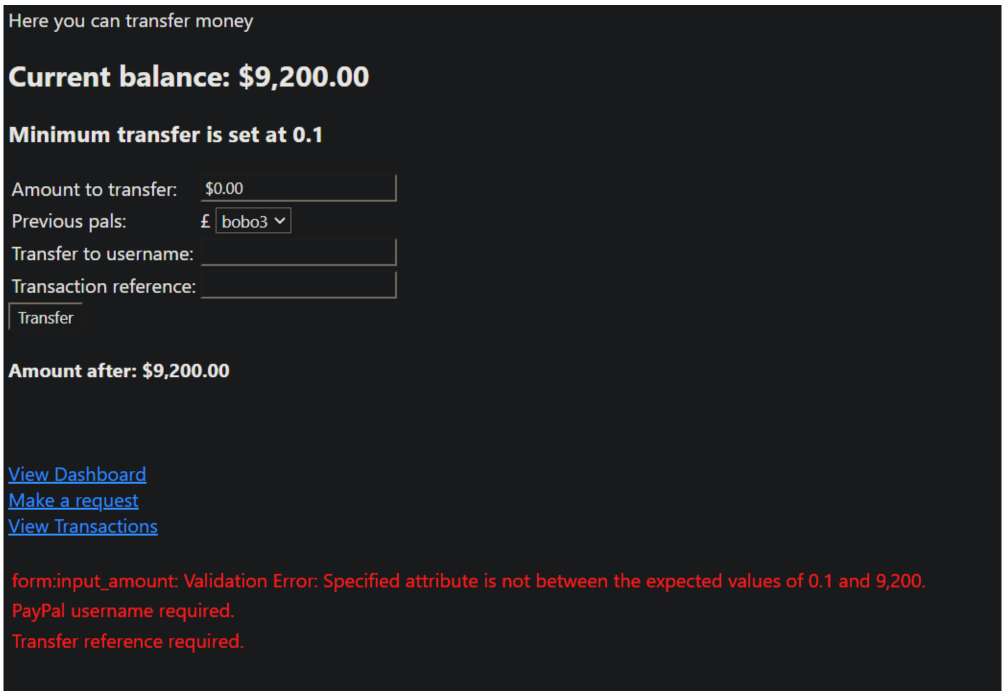
Users' transaction

User transactions are split into two you have request.xhtml and transfer.xhtml transactions

Each of them is separate .xhtml files. Under the /faces/users/transaction/[type].xhtml

Users can send money on transfer and request money.

Each input is required for both views and also the minimum of both is 0.1 and the maximum for transfer is the user's current balance.


Here you can transfer money

Current balance: $9,200.00

Minimum transfer is set at 0.1

Amount to transfer:	$0.00
Previous pals:	£ bobo3
Transfer to username:	
Transaction reference:	

Transfer

Amount after: $9,200.00

View Dashboard
Make a request
View Transactions

form:input_amount: Validation Error: Specified attribute is not between the expected values of 0.1 and 9,200.
PayPal username required.
Transfer reference required.

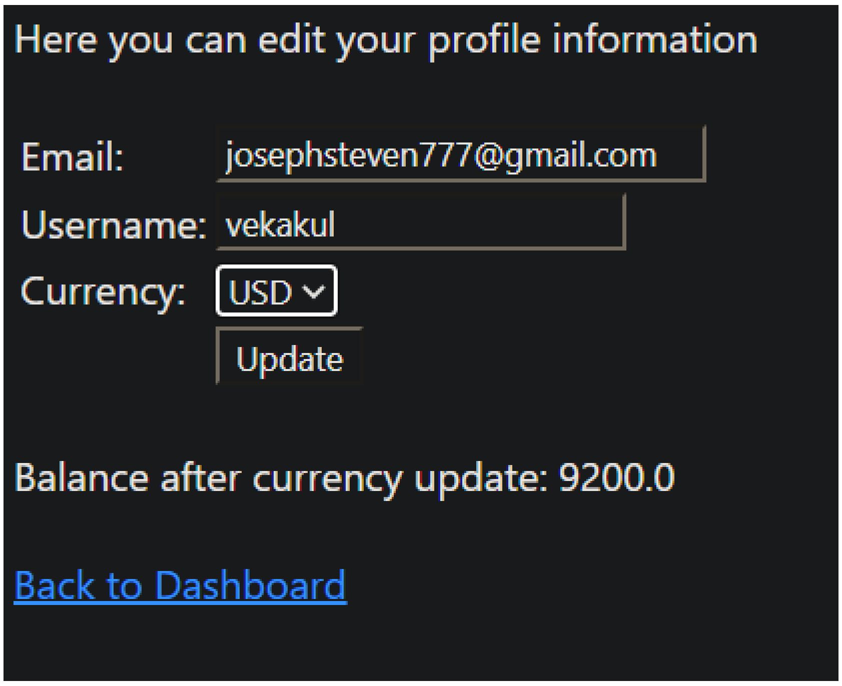
Users can select from a list of friends as well as if there are any. If not, previous pals' input is not rendered. Pal is added if they do not exist after a successful transaction.

Users edit

Users can edit their profile information using this view. It can be navigated from the dashboard or using the url /users/edit.xhtml

If nothing is changed then page redirects with same inputs.

If changing currency, then conversion rate is applied automatically and shown



Here you can edit your profile information

Email: josephsteven777@gmail.com

Username: vekakul

Currency: USD

Balance after currency update: 9200.0

Updated details are returned in the same view on successful update