Social Library

Aditya Aghi
Mahmoud Adada
Albert Yuen
Bhargava Guntoori

1.0 Application Architecture

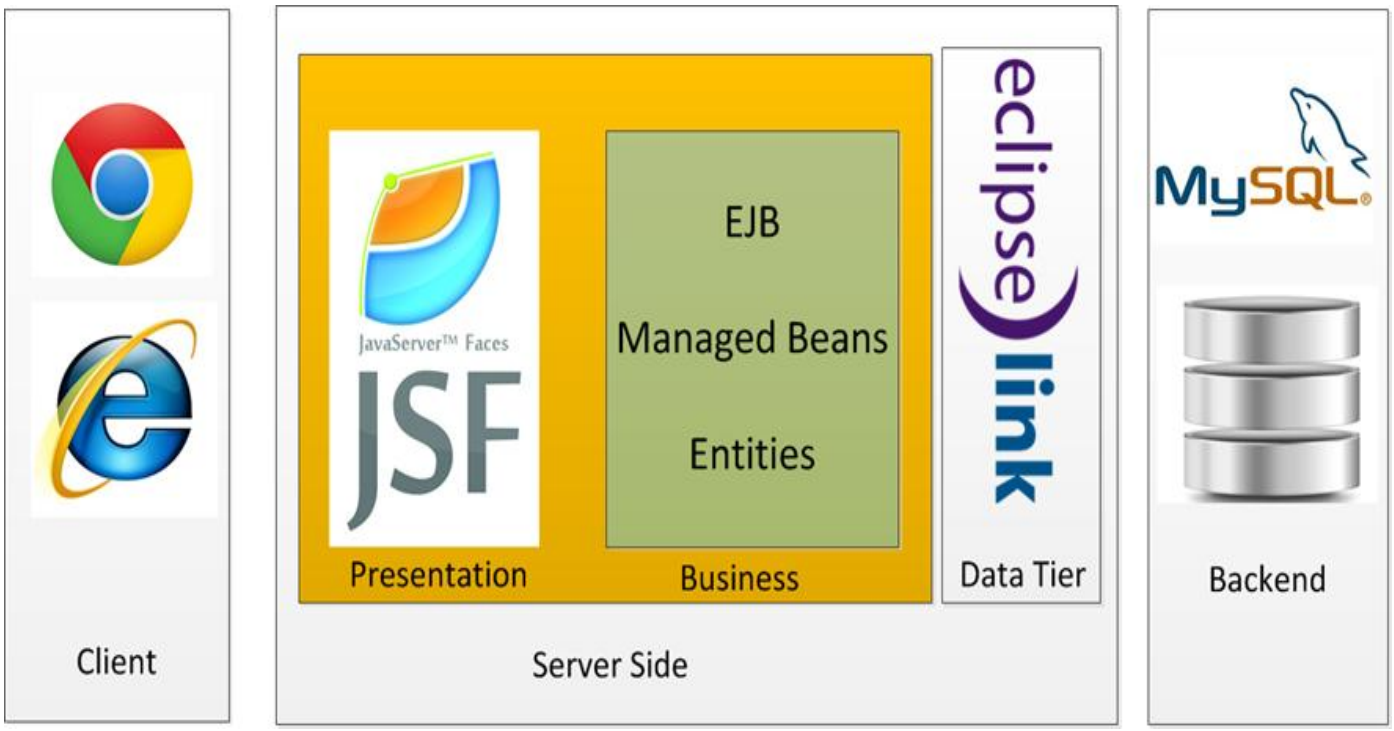


Figure 1: 4-tier architecture

Our application uses 4-tier architecture as demonstrated in Figure 1. The relevant tiers are discussed below:

The Client Side: Our application utilizes web 2.0 technologies. This means we do processing on the client side using JavaScript to save client server communication time and reduce the computational load on the server. One example of this approach is the sorting functionality on our item search page. Sorting on the various columns is handled using JavaScript code.

Presentation Tier: The presentation tier is based on Java Server Faces (JSF) technologies. Each page is in xhtml format with calls to some java functions. The page gets rendered at time of request. Rendering involves evaluating the java function calls and replacing them with html.

Business Tier: The xhtml pages access relevant entities through managed beans interface. Managed beans are special java objects that can be created and destroyed by glassfish

server (or other java ee servers). Most of our managed beans are request scoped which means they maintain state through any given request. In other words a new managed bean is created for each request and is destroyed after serving request.

Eclipse Link: All our entities are defined using the JPA(java persistence API) technology. JPA allows us to define database tables as java objects. The relevant database tables get created on deployment of application. The data in the table is made accessible to the managed beans using stateless facades (Inherits from JPA class AbstractFacade<T>). Eclipse link is one of the implementation of JPA which we decided to use in our project.

Backend Database: Our data is stored in a MySQL database. We needed minimal configuration of the database. Once we create a schema for our application and grant relevant permissions eclipse link takes care of the rest.

2.0 Web Page Design

The application contains the following java server faces webpages:

1) addLocation.xhtml: Allows the user to view pre-existing locations in the database and add new ones.
2) editUser.xhml: Contains 2 forms, first one is pre populated with user's basic information like name, address etc. It allows user to edit these fields (except email since that is also login name, if user wishes to change email then he must delete old account and create new one). After editing the user can save the changes by submitting the form using the save button. The second form allows the user to change password.
3) Result.xhtml: displays all the items owned by the user. The page has options for user to change visibility, status (borrowed, available etc.), location and delete any of his item. The changes happen on the fly using AJAX technology, the user does not need to submit any form. The page also has a link to searchMasterItem.xhtml which allows user to add new item.
4) searchMasterItem.xhtml: Lets user search the master list of items, the search is populated as the user types using AJAX technology (like Google's new search as you type feature). The user is able to add an item to his list from the search list. There is also an option to add a new item to the master list.
5) searchItem.xhtml: Allows user to search across all the items available for him to borrow. (public items and friend's items). User can search on one or more of the following fields: item name, user and location. The results are sortable based on columns Item, Category, Location and Owner. Each result has a link that results to the userItem.xhtml page for the particular item.
6) userItem.xhtml: A popup page showing details of an item available for borrowing. If the user has borrowing permissions it allows him/her to send a borrow request with a customized message.
7) userWelcome.xhtml: Homepage of the user has sections for friends list, anti-friends list, incoming friend requests, borrowed items and incoming borrow requests. The friends section has links to delete contact and make contact anti-friend. The anti-friends section has links to delete contact and make contact friend. Incoming borrow and friend requests have options to accept or decline. There is a link to userSearch.xhtml to search for new users.
8) userSearch.xhtml: Allows searching for users by name or email (either value can be entered in the same search field). Similar to searchMasterItem.xhtml the search is populated as you type. You can send a friend request to any of the users returned by the search result.
9) masterPage.xhtml: provides header and footers for all pages in the application. It contains links to various parts of the application, including searching for items, location management , editing user details and user homepage. This provides a standard view of navigation between pages and makes sure there is minimal coupling between the components.

3.0 Database Design

Facades and Database access

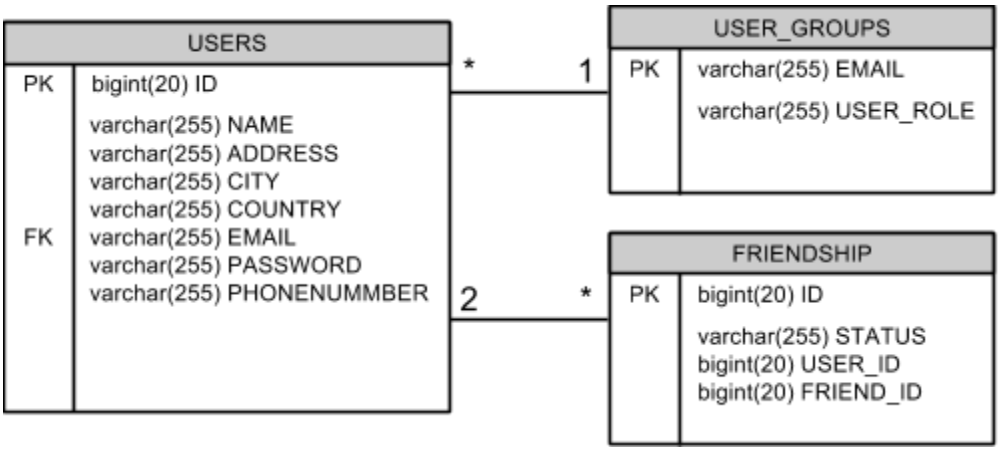
Access to database is done through facades. All facades extend from **AbstractFacade<T>** which contain functions that provide basic query functionality for a given template type:

- ***create****(T entity)* - Creates a new record in the database given an entity instance. This performs an **insert** to the corresponding table representing the entity using values assigned to object *entity*.
- ***edit****(T entity)* - Performs an **update** on the database record corresponding to *entity*.
- ***remove****(T entity)* - Performs a **delete** on the database record corresponding to *entity*.
- ***find****(Object id)* - Performs a **select** looking for a record with field values matching those assigned to object *entity*.
- ***findAll****()* - Performs a **select** to retrieve all records for the table corresponding to the template type.

Other facades in the system either simply act as a concrete facade or provides additional functions to perform more specific queries - these are listed in the sections below.

Tables relating to Users

USERS
This stores the user records in our system. A user record encapsulates all personal information we have on the user.
USER_GROUPS
This stores the two distinct users in our system, admins and end users.
FRIENDSHIP
This stores friend/anti-friend relationships. A record in this table represents an existing or pending request for friendship as well as a one sided marking of an anti-friend.



Access to these tables is provided through three facades; **UsersFacade, User_GroupsFacade** and **FriendshipFacade**. **UsersFacade** provides searching for users via their email or by their name through two functions:

- ***findByLogin****(String login)* - Searches for **USERS** records whose **EMAIL** field contains the substring *login*.
- ***findUser****(String name)* Searches for **USERS** records whose **NAME** field contains the substring *name*.

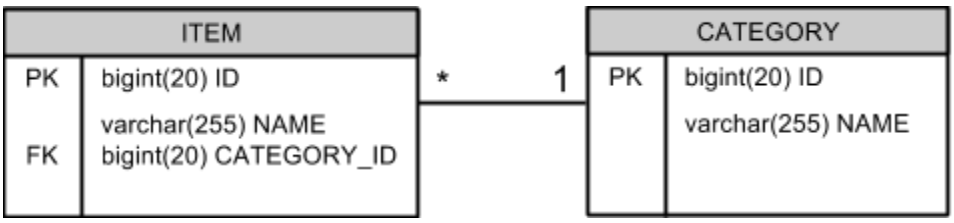
Tables relating to Master Items

ITEM

This stores the “master item” records in our system. This table encapsulates all item types currently in our system. If the user wants to add an item to his/her owned items list, a master item record must first be created if one doesn’t already exist for the item in our records.

CATEGORY

This stores the types of items that can be stored in the system.



Access to these two tables is provided by two facades; **ItemFacade** and **CategoryFacade**. **ItemFacade** allows searching of **ITEM** records by the **NAME** field through the function:

- ***findByName****(String name)* - Searches for **ITEM** records whose **NAME** field contains the substring *name*.

Tables relating to User Items

USERITEM

This stores the “user item” records in our system. The user items are the instances of our master items which a user actually owns. When a user adds an item to his collection, this is represented by a user item.

LOCATION

This stores the “location” records in our system. These are the possible locations of items to be in. After adding a new user item to his/her collection, the user can change the details of the item, which includes the location the item is at. If the location doesn’t exist the user can add it into the system.

ITEMSTATUS

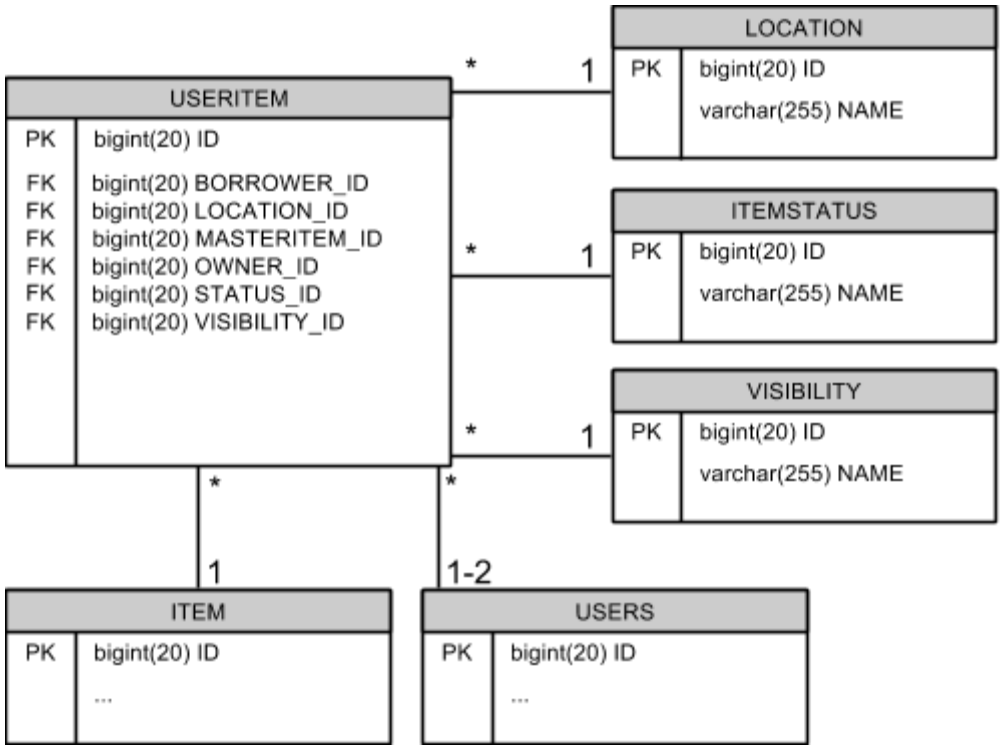
This stores the values of the status that a user item can be in: available, borrowed, unavailable, lost and recalled.

VISIBILITY

This stores the values of the visibility that a user item can be set to: public friends only and private.

ITEMREQUEST

This stores all the borrowing requests made by users on any user item in the system. Any record existing in this table is a item request that has not yet been accepted and needs to be looked at by the owner, or could be cancelled by the requester.



Access to these two tables is provided by four facades; **UserItemFacade**, **LocationFacade**. **ItemStatus** and **VisibilityFacade**. **UserItemFacade** allows searching of **USERITEM** records by

the attributes of the user item and also by owner through the functions:

- ***findItem****(String itemName, String location, String user, String me)* - Searches for **USERITEM** records whose **NAME, LOCATION.NAME, OWNER.NAME** field contains the substrings *itemName, location* and *user*. The query performed in this function joins the **USERITEM**, **ITEM**, **USERS** and **LOCATION** tables together and filters on the aforementioned criterion.
- ***findBorrowedItems****(Long user)* - Searches for **USERITEM** records whose **BORROWER_ID** is equal to *user*.

4.0 Presentation Tier

Our presentation tier is based on the Java Server Faces (JSF) technology. JSF comes inbuilt with a front view controller which is called java faces servlet. The java faces servlet handles URL decoding and fetches the relevant JSF page which eases navigation.
For authentication we use inbuilt user management capability of the glassfish server. We can configure glassfish server by providing it table names where username, password and user role (admin/basic user) are saved. We specify security clearance level for URL paths in our web.xml file. Glassfish blocks or forwards requests based on these rules.
A diagram depicting the front view controller and authentication mechanism is shown in the figure below.

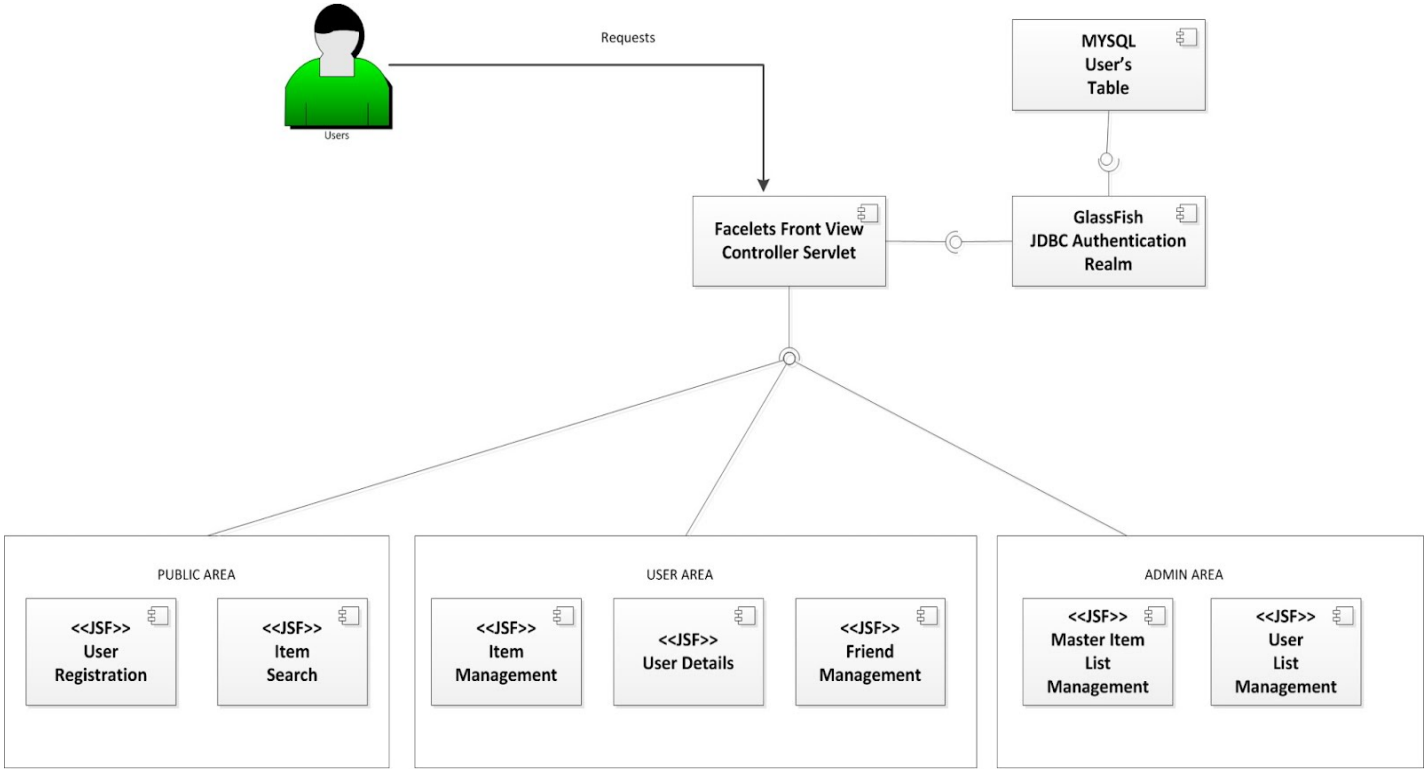


Figure: JSF Front View Controller mechanism

The presentation tier can access data objects using managed beans. Managed beans are special object which are created and destroyed automatically by glassfish server. We use request scoped managed beans which means they maintain state across any given request. The same managed bean may be used by multiple JSF pages. The managed beans access the databases using facades. The relations between the managed beans(business tier) and JSF pages (presentation tier) is depicted in figure below.

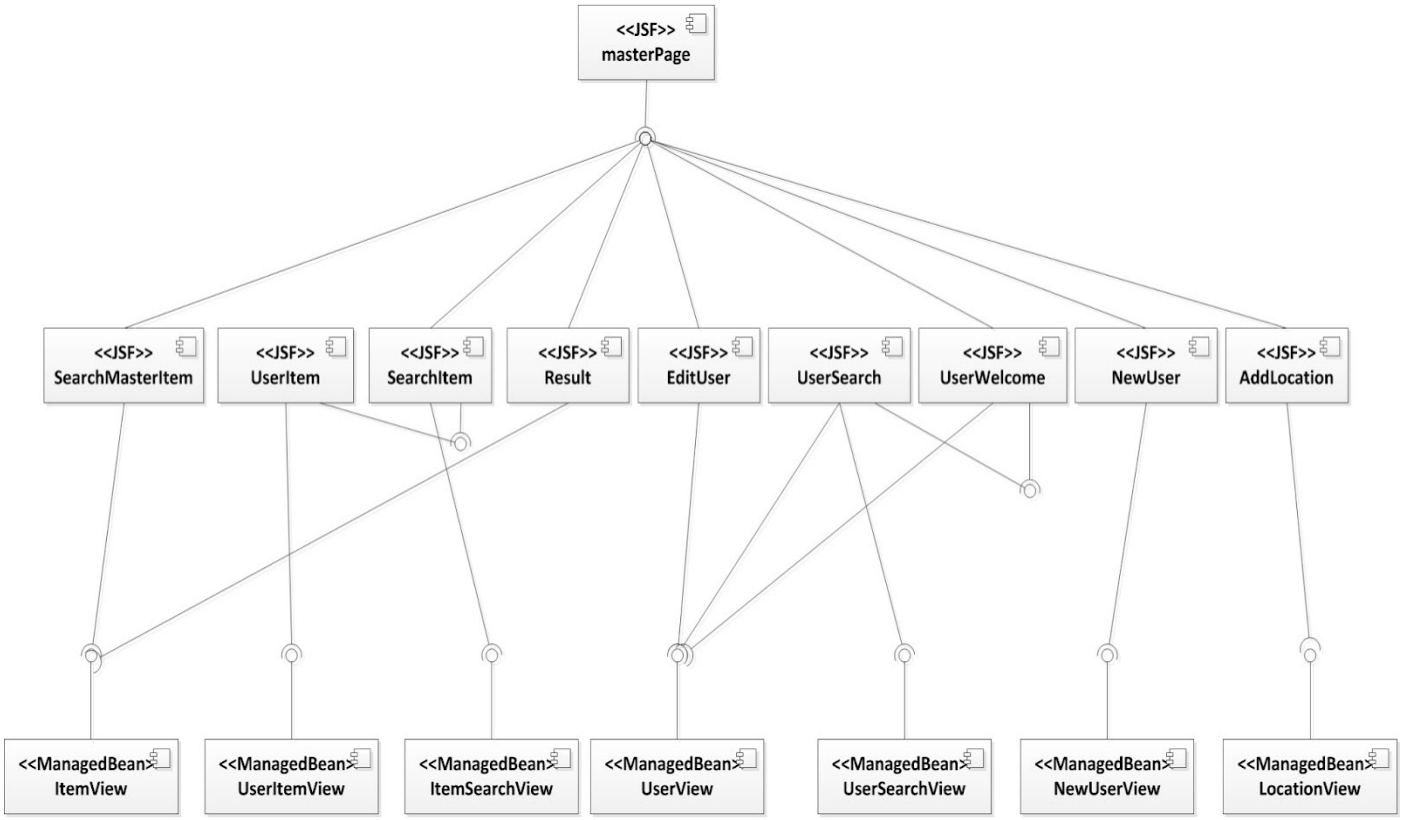
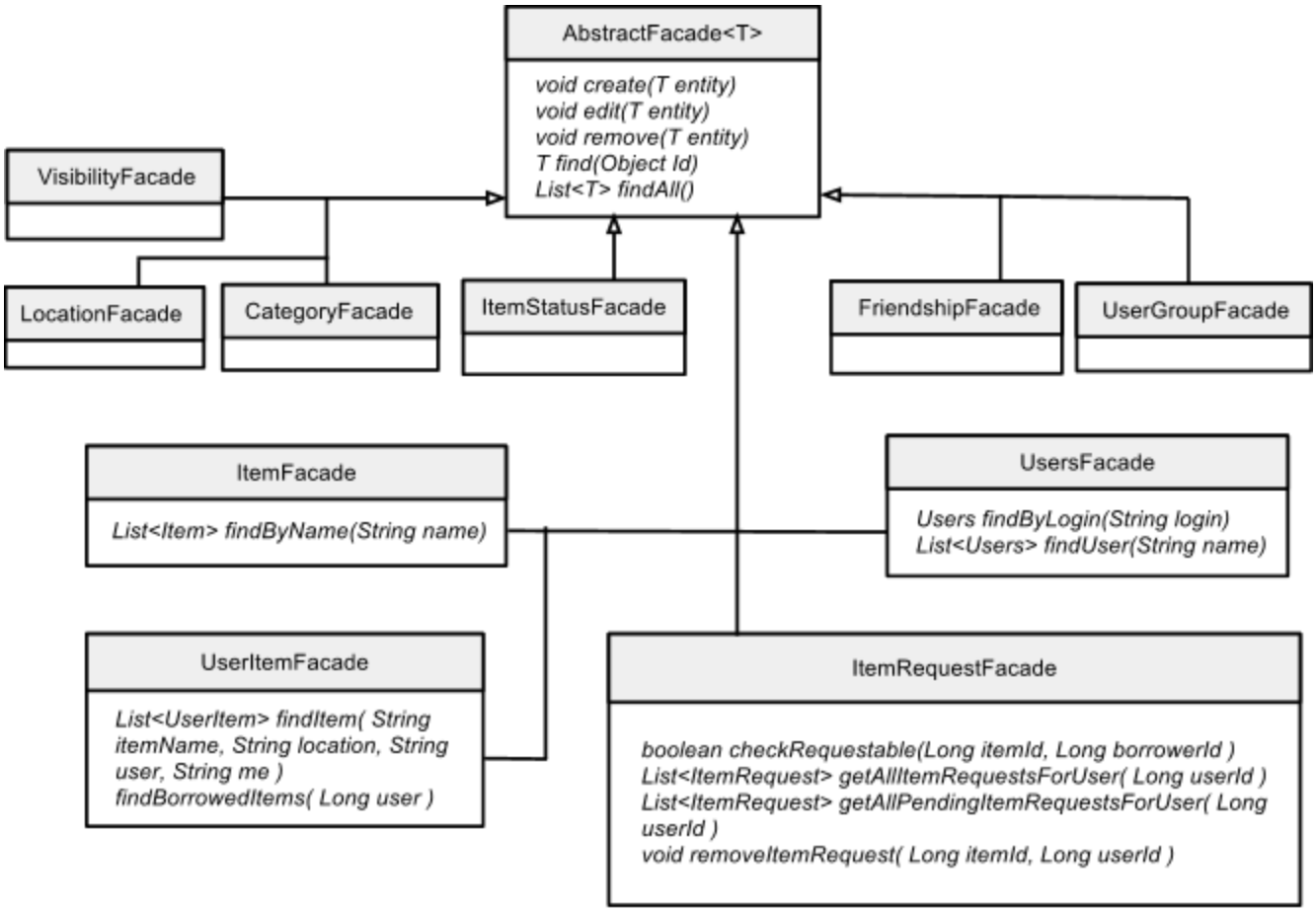


Figure : Relation between presentation and business tier.

ic

5.0 Business Tier



The facades as described in the previous section all extend from a single **AbstractFacade<T>**. These facades provide the logic necessary to write and read from the database. The facades are used by the managed view beans to access the entities in our system and also to write entities into the database. The functions of the facades are as follows:

- ***create****(T entity)* - Creates a new record in the database given an entity instance. This performs an **insert** to the corresponding table representing the entity using values

assigned to object *entity*.

- ***edit****(T entity)* - Performs an **update** on the database record corresponding to *entity*.
- ***remove****(T entity)* - Performs a **delete** on the database record corresponding to *entity*.
- ***find****(Object id)* - Performs a **select** looking for a record with field values matching those assigned to object *entity*.
- ***findAll****()* - Performs a **select** to retrieve all records for the table corresponding to the template type.

- ***findByLogin****(String login)* - Searches for **USERS** records whose **EMAIL** field contains the substring *login*.
- ***findUser****(String name)* Searches for **USERS** records whose **NAME** field contains the substring *name*.
- ***findByName****(String name)* - Searches for **ITEM** records whose **NAME** field contains the substring *name*.
- ***findItem****(String itemName, String location, String user, String me)* - Searches for **USERITEM** records whose **NAME, LOCATION.NAME, OWNER.NAME** field contains the substrings *itemName, location* and *user*. The query performed in this function joins the **USERITEM**, **ITEM**, **USERS** and **LOCATION** tables together and filters on the aforementioned criterion.
- ***findBorrowedItems****(Long user)* - Searches for **USERITEM** records whose **BORROWER_ID** is equal to *user*.

6.0 Development Process

Our first step before beginning any development was to get the environment setup, we installed GlassFish as our application server, EclipseLink as our persistence middleware and we used MySQL as the backend. For our development environment we used Netbeans 7.1 and subversion. We deployed a sample application to test that all the layers were working together without a problem. Next we began the development process, we took a bottom up approach, and the first step was designing the Entity Relationship Diagram. By defining all the entities and programing them in java, we were able to get EclipseLink to generate the tables for us. We then built stateless façades for each entity with methods that allowed us to retrieve and save the entities. We then implemented the authentication process by using the GlassFish built in authentication module. At this point we had to define the different views we would have and make sure they would cover all the features specified in the assignment. We then divided the view ups and we each built a set of views. A view consisted of making the JSF page and its corresponding managed bean. We created a master template that the other JSF's would use so we would have similar layouts and not repeat code. After all the views were built, we started improving the look of the JSF pages using JavaScript and CSS, while simultaneously testing all the functionality to make sure everything worked.

7.0 Lessons Learned

Since this was the first time any of us worked with java enterprise edition we learned quite a few lessons in the process. We will highlight some of the key ones below:

- Use preexisting components whenever possible. We found using preexisting components such as the GlassFish authentication module, can really help save time if they can be integrated easily and provide the required functionality.

- JEE has a relatively steep learning curve that may cause development to go slowly at first as you get used to the framework. However once you get used to the framework development time decreases significantly. In our case after we crated our first view, we became familiar enough with the system that creating the other views was a much quicker process.

- Communication is very important especially when working on a new platform in a short time constraint. While developing we were always in close contact with one another as we were all just learning about JEE, so we were able to share our knowledge with each other so that two different group members wouldn't get stuck on the same problem.

- Don't fight the framework. The framework enforces certain design constraints on you such as only template tags in the JSF page as opposed to JSP's where you can wright full java code. It is best not to try and find workarounds to do things like have business logic in the JSF page, as that makes development longer and more complicated.

8.0 Future Work

Due to the time constraints we were not able to fully polish the project and add the extra bells and whistles we would want to. Below is a list of some of the future work we would want to do:

- Add consistency to our UI. Currently our UI is not very consistent, with some parts not receiving as much time as the others.

- Add more AJAX to our UI, currently we only use AJAX sporadically and would like to add it throughout our application.

- Add Google maps support so that users can view the location of their items on a map, and select their location from a map.

- Add driving directions to go pick up or return an item.

- Integrate with third party product APIs so retrieve information on items in our database, eg. Amazon products api, IMDB for DVDs, ect.

- Add comments to items.

- Integrate with Google and Facebook authentication.