Report

1.1 How My Design fits with the 3-tier architectural model and the model-view-controller software pattern

This web apps project fits within this described model and structure using packages. These packages are:

* webapps2020.jsf (Tier 1/Presentation tier)
* webapps2020.ejb (Tier 2/Business Logic tier)
* webapps2020.entity (Tier 3/Data Access Logic tier)

The Presentation Tier (Tier 1)

The view and controller in the MVC software pattern are represented inside this presentation tier. The presentation tier is made up of two main components: A number of backing beans and a set of Java Server Faces (JSFs) that are linked to these backing beans. The role of the backing bean is to capture user interaction and to respond by calling methods in the business logic tier to provide the JSFs with up to date data to display in the rendered xhtml to the user. Data entry is first captured in the JSF before it is passed into the business logic tier and then persisted into the database via the data access tier. The view in this case is represented in the .xhtml files that are used to render html that is presented to the user. The navigation between these .xhtml pages are defined in a faces-config.xml file. The JSFs in capturing and passing on user gestures and interactions along with passing up to date data to the view leaves them representing the controller in this case. Moreover, the JSFs define application behaviour with calls to business methods via EJBs and maps user interaction to updates in the model. All of this behaviour is that of a controller.

The Business Logic Tier (Tier 2)

The business logic tier is made up of a number of Enterprise Java Beans that interact with the data access tier by creating named queries. Persistent data is passed from the data access tier to be presented to the user in the presentation tier via the business logic tier The tier performs the registration logic, login logic, and gets data regarding transactions, users and requests. This tier acts as an interface between the presentation tier and the data access tier to ensure that the presentation tier never directly accesses the data store. This modularisation allows us to create a presentation tier that is agnostic about the technologies that underpin the datastore, as long as the business logic tier remains a consistent means by which it can interface with the data.

The Data Access Tier (Tier 3)

The data access tier is made up of a number of JPA entities each with a number of defined queries that can be used by the EJBs to access the stored data. These entities are used to define JPA database tables that are created on the Payara server. This data is passed through to the presentation tier via the business logic tear to be presented to the user. All data persisted into this tier is captured in the presentation tier and then passed through to be persisted via the business logic tier.

Both business logic and data access tiers function as the model in the MVC pattern in this case.. They expose the application functionality and encapsulate the application’s state and keep the view constantly updated of this.

1.2 Security

How is the system secured?

* Secure digest function (SHA-256)
* JDBCRealm-based authentication
* Declarative and programmatic J2EE security
* Authorisation constraints
* Web-resource collections
* Secure connections

Advantages

* With user-passwords encrypted using a digest function we ensure that this confidential information is never passed in plain-text which would represent a security breach
* JDBCRealm-based authentication using forms allows the dynamic registration of new users. This user information is persisted and stored in a database. This provided benefits versus the use of a file realm as registration can be done by users via the presentation tier instead of having to be done server side manually
* Web-resource collections utilise URL-patterns to restrict actors from accessing resources that they are not intended to access. This results in actors only interacting with the application in the way that was intended, protecting confidential information

Disadvantages

* Additional application server configuration and a web.xml files are required when using programmatic and declarative J2EE security features
* Web-resource collections and authorisation constraints can become unmanageable as the number of roles and pages scales. This can result in a lot of config that needs managing.
* Digest functions can be vulnerable to man-in-the-middle attacks. Other alternatives such as SSL are less vulnerable to this and could be considered

1.3 Extend design as to not be a single point of failure

To ensure the system does not act as a single point of failure replication and redundancy could be introduced into the design.

Replication

Currently the system is reliant on a single server for all data storage, including confidential login credentials. The failure of this server would be catastrophic for the system and would result in a complete loss of service to all users. The solution to this issue is replication, the act of introducing multiple servers. The benefits for this are twofold. The first of these benefits is that load will be balanced across the multiple servers allowing the system to handle greater loads and perform more efficiently under these loads. The second benefit of this is having the fall back in the case that one of the servers goes down. Traffic can be directed to the healthy server to minimise the loss of access that users experience. In having the data also replicated across the servers we can ensure that users always receive consistent data regardless of the server in use.

Redundancy

We can also introduce redundancy into our system to ensure that any downtime experienced by the users is minimised. This would likely be a back-up in which data is persisted at regular intervals to keep it in sync with the active servers. This also provided a known good environment that can be restored from in the event of data loss or corruption.

1.4 Dealing with concurrent users accessing functionality and data

Currently, the system uses entity managers to manage interactions with the underlying database. The transactions within the system are managed by these container-based entity managers. Utilising entity managers alongside transactions ensures that the sharing of resources is handled in an effective and automated manner. With each interaction between the business logic tier and the data access tier spawning a new transition, we ensure that no data is inconsistent or lost during these interactions. A transaction ensures that data is only committed into the database if the interaction was completed successfully and in its entirety otherwise persisting the data will be aborted. In the event that two interactions occur at the same time, transactions ensure that data is persisted in a consistent, reproducible fashion, to make sure that data isn’t overwritten or lost or inconsistent between these two calls. This is essential for this project to make sure that the balance held by each user remains accurate to ensure that funds are reflected in each transaction between users. This is performed by checking that the data is the same at point as of query and prior to the persistence of the new data. If this isn’t the case and some inconsistencies may be introduced, the transaction can be rolled back.