Online Diary Java EE Web Application Report

Introduction

The purpose of this project was to create a web-based application for users to be able to create and share appointments as well as other user information with other uses within the system. The application was to be built with technologies found within the Java Enterprise Edition.

This report goes over an attempt at such an application, noting requirements and design as well as discussing the final application and the difficulties with building it.

Design

With the Model View Controller pattern being the standard for design patterns in Java EE web applications, the application also took on this approach. The pattern works by separating all the main components of the application, the data, the code and the user interface. This makes the program much more manageable and ultimately scalable.

The model layer deals with the business logic of the system and its states while being independent from the view/presentation layer. The controller fetches data from this layer and then sends it to the view/presentation layer. Use of session beans and entity classes make up this part of the program.

The controller layer consists of different controllers designed to interface between the view/presentation layer and the model layer. Controllers receive requests from the view/presentation layer and processes them and any validation that goes along with them.

The view layer is the “output” of the application in the form of a user interface and what the user sees. This layer is used to format and display the data fetched by each of the controllers.

The flow of information between these layers can be seen by the diagram below.

Session beans

(model)

Controllers

Client

Entity classes

(model)

Database

Views

This design was chosen as it fits the MVC pattern which comes with numerous benefits, notable the fact that because presentation, control and data persistence and data behavior are all kept separate, it means that the application can be more flexible to change. When modifications are implemented on one of the components it means that little impact occurs on the other components. This makes it much easier to scale the project if it needed to expand. This structure also reduces code complexity and reduces duplicated code.

An example of the interaction between a client on their web browser with a system using this pattern maybe when a “Delete User” button is clicked on any of the views displayed to the user. When this button is clicked an action occurs which will then call the UserBean in order to delete the user.

This looks something like this in the xhtml page:

*<h:commandButton action="#{user.editAction(delete)}" />*

And on the UserBean:

*@ManagedBean(name="user")*

*@SessionScoped*

*public class UserBean{*

*public String editAction(String id) {*

*//id = "delete"*

*}*

*}*

Example code from Mkyong (2010, para. 2).

Database

The database used was Java DB as it is packaged within the Java EE GlassFish server and was used due to its simplicity. The actual structure of the database is determined by JPA when the entity classes get created. The two entities within this project are simply the user and appointment entities, so JPA will generate two tables for each of these as well as another table to maintain the relationship between different users being involved in different appointments.

User Interface

As mentioned in the Implementation section, the main focus on this application was the backend, the business logic and on how everything connects together. Also due to the constraints mentioned in said section, there was not enough time to implement all the views, and as the business logic still has an error in it preventing development for the time being, only a few views have been created and they are incredibly simple, easy to use HTML web pages with simple styling. There is very little clutter on the page and as such they are easy to use.

Implementation

Difficulties with programming in Java EE and NetBeans

Unfortunately, the final application is not fully functional. A lot of the backend, business logic has been designed and implemented and while in theory it should work, many problems have occurred along the way. Not all of these problems were able to be sorted and as such the program cannot really be used at this time. A lot of this comes down to time constraints, but also due to the nature of NetBeans being slow to redeploy between changes but mostly from very hard to solve error messages. Throughout the development process, multiple errors arouse with JNDI, GlassFish (database server) and with various other things. It was a challenging application to build and while it doesn’t function as to how it is intended; a lot has been learnt from this process.

A simple waterfall-like linear approach was taken when approaching the development of this application. Small parts of functionality where added at a time and tested at regular intervals until issues arouse with JNDI. For a long period of time the program was unable to build and run. This bug persisted and despite rigorous research into why this bug was occurring, no solution was found. As a last-ditch effort, the developer decided to focus on continuing the rest of the project by focusing on developing out the back end of the application in hopes that the bug could be solved at a later date and that then it would be easy to join up the front end with the back end. While the bug that kept persisting eventually disappeared after countless attempts to change things ever so slightly and build the software, other bugs soon appeared and overtime as things become more complex and confusing bugs with very little explanations became difficult to manage. This seriously limited the amount of time the developer could spend on implementation and in fact, one of the bugs is still currently unsolved and prevents the program from running fully, despite the amount of work spent on the back-end code.

Looking back, having a thoroughly detailed plan would certainly have helped, but it is difficult to do this before attempting to build an application in Java EE for the first time as a developer who is programming with new tools and concepts will find it difficult to plan with these tools and concepts in mind. However, much was learnt in this iteration and if the project was attempted again, better planning and attention to detail should be high on the list of priorities.

Summary

Altogether the development of this project was an overwhelmingly challenging task for the developer who has had only a little bit of experience with Java before. The developer had never developed with Java EE or any similar web technologies before and as such had to learn a lot in order to progress with this task. Many things could have been done differently but even the process and attempting to get an application in Java EE to work, has taught the developer a lot about Java EE and web technologies.

In hindsight, a more structured approach with plan and goals would have benefitted the developer as it would have allowed them to manage their work load alongside other work. The developer should have practiced and learnt Java EE a bit more in a relaxed setting before setting out on this project as this would have allowed cut wasted time down as it would mean that they would already know the basics and common issues that may occur in the beginning stages.

References

Mkyong. (2010). *4 ways to pass parameter from JSF page to backing bean.* Retrieved from <https://www.mkyong.com/jsf2/4-ways-to-pass-parameter-from-jsf-page-to-backing-bean/>