Online Diary Java EE Web App Report

Edward McKnight (UP608985)

TODO - 1800 words max (~4 pages), revisit requirements and interface design + submit with this

# Introduction

The aim of the project was to develop a web application which allows members of an organisation to share a common list of contacts and enables them to create appointments with other users of the system. The application should be easy to use and robust, created using the Java EE (Enterprise Edition) 7, JSF (Java Server Faces) and JPA (Java Persistence API) technologies.

The high-level objectives were as follows:

* Implement user management functionality, allowing users to be created, viewed and modified
* Implement appointment management functionality, allowing users to create, view and amend appointments
* Configure container-managed security, ensuring that only authorised users can perform relevant operations

This report explains the design of the application, experiences during the implementation phase (including how problems were overcome), how testing was performed and finally a summary of the project, weighing up what was good and not so good, as well as possible alternative approaches if the project was to be undertaken again.

# Design

TODO – key components, at least 1 diagram showing the flow of information between components, highlight and justify design choices

## MVC structure

The MVC (Model-View-Controller) pattern was used to develop the application, being the standard and most widely used and supported design pattern for Java EE web applications. This pattern separates the data, user interface and logic code from each other rather than muddling it together, making the source code more maintainable and reusable (Javatpoint, accessed on 24/03/2017).

To demonstrate how the key classes and webpages of the application were split into their MVC subparts a structure diagram (figure 1) has been created.

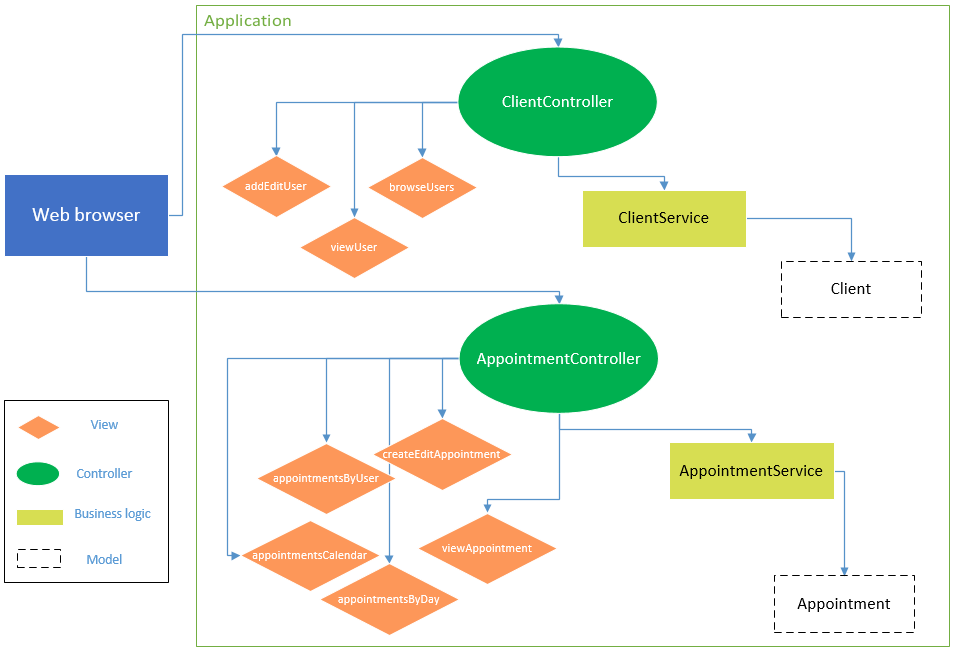


Figure 1 MVC structure of the web application

This diagram graphically shows the point concerning separating out the different aspects of the source code. The controllers act as the “middle man” between the views on one side and the business logic and models on the other side.

## Database

The database server used was “Java DB (Derby)”, which is packaged with the Java EE GlassFish server. This choice was made for simplicity, as no additional NetBeans plugins would be required and it’s a safe assumption to make that it would be compatible with the other technologies being used.

In terms of the database structure, this was automatically generated by JPA once the entity classes had been created. For this project, only two entities were identified: one to represent a user of the system and another to represent an appointment. JPA generated two tables to represent these entities plus an additional table to manage the relationship of multiple users being attendees of an appointment.

## Exceptions

To communicate problems which occurred in the business logic to other subparts (such as controllers and views) exceptions were utilised extensively. This was useful as it allowed business errors to be handled appropriately and relevant error messages to be displayed to the user so that they could take the correct course of action. An example of where this is used is when a new user is being added to the system and the username is already in use, this is detected (when processing the request to add a new user) in the business logic, at which point a “UserAlreadyExistsException” is thrown, which is then caught in the relevant controller and an informative error message is displayed to the user.

## User interface

The user interface has been designed with simplicity and accessibility in mind. Regarding simplicity this meant having a clear navigation flow through the application whereby the user is given a simple button to go back to the previous page or choose from options to go forward, to the next page.

In terms of accessibility, a minimal number of images have been used and more importantly no information that users may want to be aware of has been embedded into images, this means accessibility software such as screen readers shouldn’t struggle to process the generated HTML. Buttons and text have also been made relatively large to make them more readable to low-vision users.

# Implementation and testing

TODO – Experiences implementing the artefact, choice of development tools, difficult areas of development, how problems were overcome, how components were tested

## Approach

Development was conducted using an iterative and incremental approach, with subsystems and functionality being implemented one by one and tested continuously. An example is the user management subsystem, which was developed separately and prior to the system to manage appointments. Using this method, any bugs or errors encountered could more easily be traced back to what was being worked on during that time and efficiently resolved (TutorialsPoint, accessed on 24/03/2017).

## JSF Framework

TODO – BootsFaces + JSF templating

## Challenges

The project was a challenging programming task as although the author had experience using Java and web languages, they’d not used the Java EE technology before. There was a steep learning curve to understand the concepts behind Java EE and how it differed from other web development technologies such as PHP, which is why the prior unit lectures were vital.

A challenge encountered early on, after the entity classes had been created, was that Java kept generating an error when trying to start the web server and create the database. After reading the error log it appeared that this problem was arising because one of the entities had been named as “User”, which as a reserved word in SQL, was causing problems when JPA tried to create a table named as such. The easiest solution to this problem was to rename the entity to “Client” instead. After making this change the database was successfully created.

After the user management subsystem had been implemented a problem was noticed that the search was case-sensitive, e.g. searching for “Admin” would yield no results when trying to find a user with the username of: “admin”. This was resolved by changing the corresponding JPA query to include the “LOWER()” function so that both the search text and usernames were forced to lower case; which made the entire search case-insensitive, as desired (Strings in JPQL, accessed on 24/03/2017).

Another issue, concerning searching for users, was found after implementing AJAX into the search functionality. This allowed the search results to be updated without refreshing the page, but doing so also broke the corresponding action buttons for each user (i.e. view, edit and delete). This is due to a bug in Mojarra (the implementation of JSF, packaged with GlassFish) in which view states are not given to buttons after they’re updated using AJAX. BalusC provided some JavaScript code to fix this issue (2016), which was successfully implemented and solved the problem. This same fix was later reused for the search appointments pages.

## Testing

Unit testing was attempted using NetBeans’ built in tool for generating JUnit tests, however these tests failed with errors concerning the test file not being able to find an EJB (Enterprise Java Beans) container to load when running the tests. The dependency (which contains this embedded EJB container) was added to the Maven configuration, however the unit tests still gave the same error.

Although this was a setback, as continuous testing had been carried out throughout the application’s development it did not have too severe an impact on the project overall.

Black box testing was performed when the entire application had been finalised, in addition to after each subsystem had been implemented. During development, the specification was repeatedly referenced and revisited, to ensure that it was being adhered to.

# Summary

Overall the application was completed to a professional standard which considered: security, design and functionality considerations, meeting the specification. However, this doesn’t mean it is perfect and improvements can always be made.

One piece of further work to be done, is to perform further research and debugging to get the unit tests and embedded EJB container to work correctly, this would help to identify and resolve bugs if more functionality is added later.

If the project was to be carried out again, Java’s internationalisation library (i18n) would be used from the beginning. This would allow the entire application to be made usable by organisations using different languages if a corresponding translations file is present. Also, if BootsFaces was known about at the start of the project, it would have been used from then, which would have saved some time in having to retrospectively add its functionality. This time could have then been spent to focus on getting unit tests to work correctly, for example.

Finally, concerning the user interface’s design, initially the navigation was made to be simple to use but this choice has resulted in it being slow to navigate to pages deep within the page hierarchy. To rectify this an additional mechanism of navigation could be added (possibly in the form of links at the side of the page) which would allow users to jump to areas of the application which are accessed frequently, such as adding a new user.

# References

BalusC. (2016). *h:commandButton/h:commandLink does not work on first click, works only on second click*. Retrieved from StackOverflow: http://stackoverflow.com/questions/11408130/hcommandbutton-hcommandlink-does-not-work-on-first-click-works-only-on-second

Javatpoint. (2017). *Model 1 and Model 2 (MVC) Architecture*. Retrieved from Javatpoint: http://www.javatpoint.com/model-1-and-model-2-mvc-architecture

*Strings in JPQL*. (2017). Retrieved from ObjectDB: http://www.objectdb.com/java/jpa/query/jpql/string#LOWER\_and\_UPPER\_-\_Changing\_String\_Case\_

TutorialsPoint. (2017). *Iterative Model*. Retrieved from TutorialsPoint: https://www.tutorialspoint.com/sdlc/sdlc\_iterative\_model.htm