CSC8019 Software Engineering and Team Project: Group Report

Team 2

The primary functionality of our application allows a user to:

- Create or log into an account
- Read information on the four castles (Alnwick, Auckland, Barnard, and Bamburgh)
- Look at route options for the castles from central Newcastle
- Allow the user to create and save their own plan
- Book travel and entrance tickets for the castles
- Return any purchased tickets
- Review the application

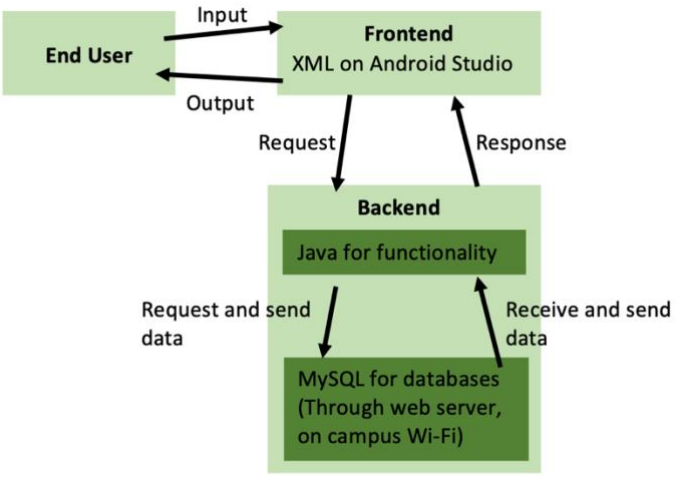

Figure 1: Product Architecture Diagram

Within the front-end architecture of our app, it follows a similar structure to that of the apps we referred to during our analysis process (requirement specification, section 1) and the initial designs of our app (requirement specification, section 7). After logging in, we have a bottom navigation bar which leads the end-user to viewing the specific castles, booking a plan, or viewing their existing plans. Within the castles screens the end-user can look at the general information on the castles, see any nearby amenities, and a gallery of photos of the castles. Within the booking screen, the user can enter their requirements for travel and view details of suitable plans. Within the existing plans screen, the end-user can view, pay for, or return any existing plans. The side drawer navigation bar leads the end user to app settings, their account details, logging out, and reviewing the application. We have also added an about this app screen which details the team and our specific roles.

For the back-end architecture, it is reflected in our UML diagrams (requirements specification, section 7). The domains are implemented as classes which are *castle*, *comment*, *departureTime*, *journey*, *nearbyPOI*, *route*, *ticket*, *time*, *transport*, and *user* for storing the information. Our developers tried to ensure throughout the project that the code they were writing was not overly complex unless it needed to be for functionality – meaning if the project were to gain more developers, they would be able to understand what is going on.

Our databases are also reflective of the E-R diagram (requirements specification, section 7). We have ten entities for departures, routes, forms of transport, the castles, tickets, nearby amenities, journeys, users, reviews, and user avatars. They then have the appropriate attributes to achieve desired functionality.

We fulfilled most of our functional requirements and even added some throughout the duration of the project. We satisfied the core desired functionality of the app asked by the brief. That being an application that allows students to purchase travel and entrance tickets for a day trip from Newcastle city centre to Alnwick, Auckland, Barnard, or Bamburgh castle. On top of that, we allowed for users to create personal accounts, review the application, and change the language settings of the application. The following lower priority requirements were not fulfilled:

- NFR4 Reviews will be moderated for foul language
- FR31 Layout of the app will match up with phone orientation

If there was more time, we would have addressed these however, they were not a high priority.

To test our application, we performed three forms of testing: black box testing, white box testing, and general front-end analysis. For the black box testing, we produced a testing table (attached with source code) that listed all the possible end-user inputs in order to confirm the functions were behaving as we expected. We had two major phases of testing, the first phase phase was following the completion of the first version of the app and the second phase followed the completion of the final version of the application. Within those phases, multiple team members tested the app to ensure it was thoroughly checked. These testing processes were crucial in the development of our application as we discovered many minor issues. Below is a sample of the testing table from the first testing phase:

Location		Problem and/or Suggestion		Priority
		Problem	Suggestion	
		correct thing and says no tickets are available		
		Analogue clock is confusing to use for time selection	Use a digital clock	M
		Cannot choose a return time	The date should stay the same, but the customer should be able to choose what *time* they return or if they want an open (but same day) return	H
			If booking for more than one person, could also show the total price	L
	Find Plans		Since there is only ever one plan, we could just go straight to the details of the plan	M
		Takes a while for plan to load up	Have a loading wheel or bar	L
		Cannot scroll back up on the "View Details" window, will just pull that window down	Make it a new screen and then add a back button	M
	View Details		After saving details, I should be led to the "My Plans" section so I can view and pay for my plan	M
My Plans			Put date in dd-mm-yyyy format	L
Side Menu	**My Account**		What is the nickname option for?	L
		When trying to change email to an invalid one, a message comes up saying its invalid but then I'm almost immediately taken back to the home page, so I don't get a chance to fix my error		M
			When changing password, either get rid of the "old password" input bar *or* add option of	M

Figure 2: page four of the first phase testing table

Due to time, we were not able to amend every single error we found, however we satisfied those of high priority.

For the white box testing on the application, we discovered a few bugs. One example was that the application would keep occupying a large amount of network bandwidth and memory, making the device get heated and run slowly. This issue came from the main thread of *MainActivity.java*. It would continuously download the user's avatar and update the local cache. This bug was fixed by removing the calling function of getting the avatar from the database in *MainActivity.java.* We then added a new logic in *Avatar.java* to get the avatar from the local cache first and then get the avatar from the database.

In the front-end testing, our programmer tested the layouts of this app on different android devices. One component proved by the android library named *TabLayout* worked well on a small device whilst displaying badly on a large resolution device. Checking the android API turned out that this component is neglected in larger devices. Therefore, our front-end programmer created a new component that extended from the android class and resolved this issue. Further, we had a general walkthrough test by which we were checking the front-end was up to a good standard.

One restriction with our testing process was that our application only worked on the University network due to the database only being accessible within the network, so when we tried to cast external tests, the application failed.

Looking at scalability, the app can be used on any android at or past 5.0 and will function on any newer versions in the future. The databases have been set up to allow for additional routes and castles, or even expansion to other cities, which is a realistic potential for the future of this an app.