**Advanced Software Engineering**

**Project Reflective Essay**

Group 1

Department of Engineering & Informatics

University of Sussex, Sussex House, Brighton, BN1 9RH, United Kingdom

[**1.0 Introduction: Project Context**](#_ogopoaqc1jdr) **3**

[**2.0 Platforms & programming languages**](#_ytyvxzzphj9) **3**

[2.1 Front-end: React-Native](#_ctpne51scbrk) 3

[2.1.1 React Native Components](#_d56hgint1vxw) 4

[2.1.2 Deploying both Android and IOS](#_7oqrmulzh1ep) 4

[2.2 Back-end: NodeJS](#_ai05yxgduj5b) 4

[2.3 Database: MongoDB vs. MySQL](#_df4g18cdt5uj) 4

[2.4 Hosting: Ngrok vs. Heroku](#_uc4e509bfwie) 5

[**3.0 Software Development practices**](#_5bv24x2ukosf) **6**

[3.1 Tool integration: issue/bug tracking](#_v5p02jwdrnm8) 6

[3.2 Version control:](#_wupnr0z0bth5) 6

[3.3 Peer & remote programming](#_r3dil0pbsnz6) 7

[3.5 Knowledge Transfer & Mentoring](#_4hcple2zlaat) 7

[3.6 Code Reviews](#_h3enctf849lt) 7

[3.7 Testing](#_7wtntdg6dhiv) 7

[**4.0 Conclusion**](#_gmuyrcazb8d4) **8**

# **1.0 Introduction: Project Context**

Essentially, the project is about building a cross-platform app that works on Android and iPhone. The group was allocated to form a software development house. Team members are multi-disciplined, with spanning skills, knowledge and experience in software development, computer sciences and information technology in general.

The problem presented to the team was to develop software for a customer in iteratively revealed requirements; The required software deliverables were revealed step-by-step. which were vague at the start and in some cases require the team to conduct business requirements and software specification sessions with the customer to close the gap between what’s going to be delivered and what was the customer's expectation. emphasis was on delivering a product that is well specified, has high-quality, and can be easily maintained.

The challenge of meeting these objectives is to find the appropriate tools and approaches for a controlled software development methodology that heavily leans towards agile software development. Due to the nature of the project, there was a high emphasis on quality control automation and we will demonstrate how we achieved quality control. ? Something like that… of software development. Key focus areas to a successful product in this environment were to plan, design, build and test using modern approaches to configuration management, version control, testing and build automation.

Aside from technical requirements, we were also challenged with working effectively as a team. We would need to demonstrate team synergy by communication and problem solving techniques to ultimately deliver a high-quality product. This type is collaboration would allow us to follow a controlled and effective software development cycle, similar to that of industry.

In this essay, we will look into the different software decisions tackled in our development lifecycle, some of which were planned initially as per our project plan, and others that were dealt with on an ad-hoc basis. We will also reflect on the emerging behaviors of teamwork and communication that aimed to achieve team synergy and any challenges we faced.

# **2.0 Platforms & programming languages**

# **2.1 Front-end: React-Native**

We decided in our project plan to use React Native (RN) as a cross-platform environment since React is using native code similar to JavaScript to compile code and JavaScript is a common language between all of our team members. We found there were many online, open-source, resources available and that the stack-overflow community is very active with React Native guidance. We understand there are competitor market products such as Angular framework. Ultimately, we made the decision to stick with React because of the extensive support network and accessible tutorials available online.

### **2.1.1 React Native Components**

As we dived into RN, we learned there were a few useful components that could be leveraged for our project. For the map component of our project, we utilised react-native-maps Mapview, Slider, and Marker. Each of these components use built in call-out functions and props that were useful for our customization. To create different screens for our app, we utilised RN app and stack navigation within our App.js file which included code and imports for each .js file used in navigation. The main driver behind using components built in-house to RN was to avoid potential bugs while deploying on Android and IOS.

Overall, using React-Native was a good decision because by the end of delivering the 5th deliverable we had picked up and learned a lot about the platform so as we were able to develop features and implement ideas very quickly and swiftly. The modularity of React-Native also made it very flexible for us to make adjustments and changes as we deemed necessary without impacting other functionalities.

### **2.1.2 Deploying both Android and IOS**

We utilised a build tool called Expo (which is a partner to RN) to launch and share our app across users. In order to display our app on each device, each member was responsible for downloading the Expo Client application and running a start command from terminal. Each start has a unique QR code generated which is located on both terminal and from a browser local host. Users can use the Expo application from their device to scan the QR code provided and render the application. Since choosing RN as our framework, we successfully launched an application both on Android and IOS. This was tested by multiple group members throughout the project life-cycle. Using Expo was an excellent choice as it was very convenient as a build can be performed really quickly and with only one command, and applying and code changes immediately reflect on the app within the same build; rather than having to compile and build every time. in some cases we would even send a screenshot of our QR code to each other to test each other’s builds between Android and IOS.

## **2.2 Back-end: NodeJS**

For the back-end we used nodeJS. We designed our backend using Express API, which is an API based on the Express Library. This choice to use nodeJS was proven to be successful for our team because it was a simple implementation to pick-up and apply for our project. We found there were plenty of online guidance demonstrating how to use NodeJS. It was beneficial to use this language for our backend as our front end uses a similar language -JSX-. Having two related languages on both ends made it easy to understand the flow of communication from our user interface to our cloud database (front end to backend). Ultimately we were able to integrate both ends quickly, troubleshoot, debug, and conduct code review because of the accessibility of JavaScript.

## **2.3 Database: MongoDB vs. MySQL**

After trying chronologically Google Cloud Platform, Firebase, AWS we decided to use MongoDB. We deployed a mongoDB database which is hosted on a Google-Cloud-Platform. There are a few reasons why integration of the database with the backend was very easy and intuitive for our team. For the connection between mongoDB and our express api we used a package called “mongoose”. This package was particularly helpful for filtering the data when we want to filter for price. MongoDB CLI interface made it easy to replace our data and upload additional data whenever there was a specification or requirement that demanded so (e.g., adding more robust housing price data files). This platform choice also proved to eliminate a lot of overhead that may have otherwise arisen while trying to use database management tools by allowing us to manipulate datasets according to the changing requirements. It wasn’t our initial choice to use MongoDB as we first attempted to use Google Cloud’s MySQL storage service. In the end, MySQL proved to be far more difficult to work with the interface, data management, and integration with our application.

## **2.4 Hosting: Ngrok vs. Heroku**

The backend was initially being hosted on the local server and connected to the internet by using Ngrok on our backend developer’s machine. For the server automation we used “nodemon” this was needed because when ever we changed something backend you need to deploy it again but with this nodemon script it was so easy to automate this process. And although back-end code was maintained and available through our git repository, it was very complicated to duplicate the set-up on other machines of remaining developers, especially front-end; This was due to the complicated dependencies, and version compatibility with NodeJs and other components. This had led to a bottleneck in the ability of them to develop, test, verify and push features; instead, developers had to defer code changes and new features to a single person that had a working setup and that person would then have to test and feedback.

This was solved at a later stage by moving the backend altogether to a cloud (Heroku). This was a great choice and improvement to our infrastructure because it meant that front-end developers no longer had to set-up dependencies or defer testing to a single person. This way, front-end developers were able to treat the back-end as a solid working blackbox and in turn raised feature development output. Moreover, cloud hosting was preferable for quality of the application as a whole.

Furthermore, using this specific cloud service, was very simplistic with a managable learning-curve. This choice allowed for automated continuous integration and improvement as the cloud was connected to our git repository, meaning that any change pushed to the repository can be automatically deployed to the cloud.

This cloud solution in particular allowed us to build a development pipeline where there is a staging environment and a production environment. Push requests would be first deployed to the staging environment, tested and monitored. and when verified to be correct and stable then it may be deployed immediately to the production environment. CI/CD is beautiful!

Heroku also allows us to run the code in any GitHub pull request in a complete, disposable, separate container (called Review App in Heroku world!). each container has a unique URL you can share, making them a great way to propose, test, and merge changes to our code base. You can configure those containers to spin up automatically for each pull request, or you can create them manually from your app’s Pipeline page in the Heroku Dashboard. Had this project had to go further on, this solution was obviously going to cut corners in the aspects of communication, deployment, testing and CI/CD; automating the redundant tasks, making more time for creative work.

# **3.0 Software Development practices**

## **3.1 Tool integration: issue/bug tracking**

We used a built-in system within Github to create and track issues. Github has a useful built-in space on the main repo’s site called “issues” which allows users of the application (since this is a public repo, anyone with read access) to write any issue related to a release. We worked hard to document bugs and system glitches and resolve them in a timely manner. We also used Trello as a communication platform to document learnings, research and other helpful information. Each member used these tools to their discretion. Unfortunately, we didn’t spend much time resolving the issues we had noted to our tracking systems, but I feel we did a decent job at using WhatsApp (sometimes a bit overkill at times) as a communication to jot down ad-hoc issues to resolve.

For the future, it would be great if each team member would communicate if they’re experiencing any issues with the application or environment set-up through a public forum such as GitHub, instead of keeping the issues to their local machines. We must remember that the software is open source which means at any point in time anyone could launch the software from their machines and incur the same errors we are experiencing from a development perspective.

## **3.2 Version control**

We used GitHub as our version control system. In practice we had a centralised policy, where everyone forked the main repository, used branches of their fork to test out new features and then those versions would be pushed from the fork back into the base repository, master branch. Each team member was encouraged to implement branching that was both identifiable and simple, meaning the name of a branch would be specific to the feature under development during that branch and simple enough that it is just one functionality instead of groups of functionality chucked into one branch.

There are a few lessons we learned with Gtihub: never have one owner of a repo. We should make sure the repo has equal admin ownership across the team. This can be accomplished by deploying an “organisation” in Github and creating a repository within it. Selecting users and changing permissions to administrative control can be accomplished quickly with this context. Otherwise, as in our case for this group project, we had one owner of a single repo and it was a challenge for them to grant everyone administrative access which delayed releases in many cases.

## **3.3 Peer & remote programming**

Most of the programming was done individually & remotely. Although in practice whenever code broke or we weren’t able to successfully implement a change, peer programming proved to be the solution. It seemed like through peer programming we were able to combine knowledge and troubleshoot faster. Peer programming also eliminated shortcomings of remote communication. Going forward all features and solving bugs were done via peer programming and seemed to be very effective & efficient for synergizing team expertise and learning.

## **3.5 Knowledge Transfer & Mentoring**

There was an overall lack of experience with the platforms and programming languages used. This has prevented team members from taking the role of mentoring others. Nevertheless, looking back on this it seems that it would have been beneficial as we picked up the tools and became comfortable with platforms to do knowledge transfer & mentoring sessions. We would have benefited from more in-person group meetings for this purpose. Although time was very constrained.

## **3.6 Communication Tools**

In order to share learning resources and track each member's progress in task related knowledge learning, we chose trello as a communication tool. Trello is a schedule management application. Our work on trello is all around "board" function. Each member of our group can create a to-do list based on our own "board". We also used our own board to paste the useful links of some tutorials ,articles of background research and videos which is relevant to the assignment. When we finish our own work, we can mark the task status as complete, which is quite convenient for us to follow up the whole project.

Apart from the Trello, we also used WhatsApp to convert the information of our daily process on the app. Because of its immediacy, we also informed every member of our team when we held daily emergency meetings. When we checked the running performance of the app on the IOS/ Android mobile device, we often used its photo function to share picture information.

## **3.7 Code Reviews**

Code reviews were done through the peer programming sessions and before any commit push to the master. More regular code reviews could have been beneficial to increase team experience and to share knowledge. In the future, we would like to implement tools such as Github Code Review.

## **3.8 Testing**

We have a great considering our mobile application testing procedure which fully deliver for usability, functional and consistency. As we have a couple of tools for developing our application such as Github for program version control. I used git clone from Github for local testing and check the expectation result of application on iOS and android mobiles as a result of cross-platform application testing.

The initial complicated set-up made it difficult for testers to do their blackbox testing. As team members faced many compatibility and environmental issues. Going forward we have improved this and overcome those obstacles and saved time in testing.

In regards to load testing, we had a huge amount of data which we had to process -over 4 GB-; to overcome this we started with a limited number of data for testing our application and gradually increased the size to optimize performance.

For integration testing, we were using Postman to ensure our back-end’s input parameters and output objects are correctly as specified.

With regards to unit and component testing, we have researched and implemented some test cases using JestJs, which is “a delightful JavaScript Testing Framework with a focus on simplicity”. it requires only a very minimal setup and works for React-Native and NodeJs, which made it perfect for us.

# **4.0 Things that went badly that we would not do again**

This section inspired by the groups that Martin mentioned us in previous years, so we thought it can be great if we also mention our experience. First, Project Plan should’nt be underestimated and each group member should work in detail on it. The part the we most struggled was Conflict Resolution in our Project Plan. Since we didn’t have a complete resolution it was very difficult to solve afterwards, and people were unsatisfied. With this project we also understood the written rules for being a team. Secondly group communication was not good as we wanted. This can be the result of random selection for Project team. For us the random selection for this group was unlucky, and If we had chance to change it, we would. The lesson we took from here, if we have problems, we should seek assistance as soon as possible. Waiting for a response or ignoring the problems is not solving it. Lastly although we tried to divide the work so many times, We couldn’t divide the work evenly and it made us slow in the process.

# **5.0 Conclusion**

The project was challenging on many levels. On one hand, achieving high team participation and maintaining synergy and effective communication was our biggest struggle. And on the other hand, the technical aspects of learning new programming languages, platforms such as React-Native and NodeJs and implementing modern practices of software engineering kept us tirelessly active on our application development. Other aspects such as using automation, cloud services, testing tools and version control were challenges we believe we had achieved success with in this project.

As a group with a variety of skill sets and software development experience, coming out of this group collaboration, we feel as if we have gained a lot of knowledge, insight and understanding of how using software engineering technologies, tools and approaches come together to an astounding flow of art!

We recognize that there is great potential for improving the quality of our software development method. We look to use our learnings from this group project to make better decisions in other software projects or in industry and carry with us a sense of quality in our work and ethics for collaborating with a global cohort.