Roomer Technical Plan

Nonfunctional Requirements

To ensure the performance and user experience usually expected from a commercial application, the Roomer website will adhere to the following non-functional requirements:

- **Posting a listing** will give a user confirmation that their listing has been received near-instantaneously, and the listing itself will be added to the system within a few seconds.
- Buyers should be able to be **notified via email** that a seller is interested and how to contact them within minutes of a seller expressing interest in a listing.
- **The listing feed** will continuously scroll until the bottom of all relevant results has been reached. The buffer when loading results will take under 5 seconds to return a new page of results.
- **The search feature** will allow detailed searches with many filters, based on specific criteria such as square footage, number of bedrooms and bathrooms, pet friendliness, price, location, ect.
- The product will support **all modern browsers** and be built with the intention of eventually supporting mobile application versions for iOS and Android.
- The frontend design will implement **responsive design** to support various screen sizes and devices, including mobile browsers.

System Architecture

The Roomer application will consist of two separate parts; the frontend written web application and backend services. The front end will be developed using React Native. The back end services will be hosted in AWS using Javascript to write code where necessary. The following diagram shows how we propose the basic architecture of the Roomer application.

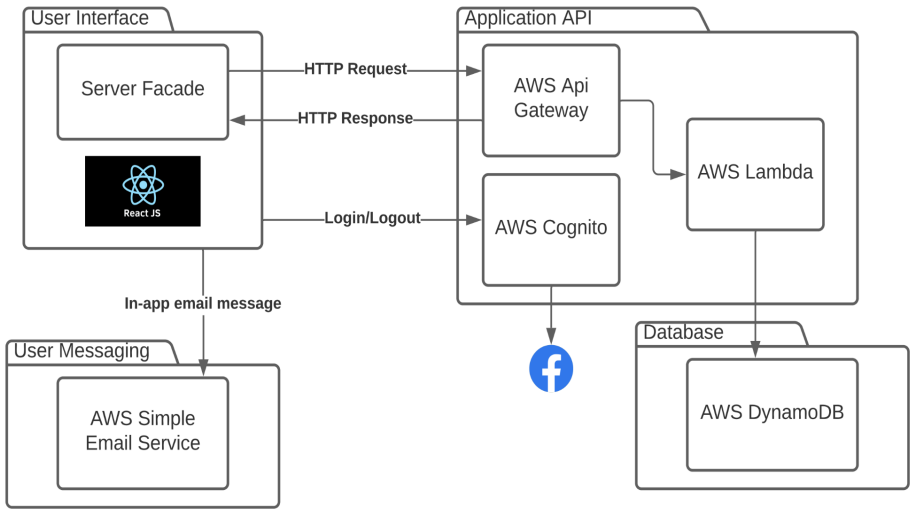
Front End

The front end of the Roomer web application will be written in Javascript using the React Native Framework. By using the React Native framework, the Roomer team will build an application that fulfills the requirements of being browser agnostic and mobile friendly. Additionally, React Native allows the possibility of creating a native IOS and Android application in the future if the Roomer team would want to port the web application.

Back End

The back end of the Roomer web application will use AWS Cloud functions and services. The main backend will be handled by creating an AWS Api Gateway to handle restful HTTP requests and trigger lambda functions to handle said requests. An additional selection that we have made is to use AWS DynamoDB as the database for the Roomer application. We believe that DynamoDB is the correct choice because of its ability to scale well as the application grows and the features that DynamoDB brings. Another service that the Roomer team has decided to use is the AWS Cognito feature. This will allow users to select registering and log in via Facebook, Apple, Google, etc. Lastly we have selected to use the AWS Simple Email Service. By using AWS Simple Email Service, the Roomer application will allow sellers to contact buyers without making the buyers disclose their information.

By using React Native and AWS technologies, the roomer team will build a MVP for the sponsors that both meets the needs as set by the project sponsors and is cost effective. Because the Roomer team will be using AWS technologies, there is little to no cost for server infrastructure. Additionally using the aforementioned technologies allows for further development and expansion of the Roomer application.



Development Tools

Version Control

For version control, we will use a git repository hosted privately on Github. This way team members can work asynchronously while managing code conflicts. Additional information, such as tickets, future PBIs, etc. will be kept on Jira. Any documentation and spec files will be kept in the company's Microsoft Teams folder.

Continuous Integration and Deployment Framework

We will use the Expo framework in order to deploy code seamlessly and quickly, as well as keep doors open to iOS and Android development further down the line. Additionally, we will use Jest for unit testing and Selenium for functional testing in the browser. Additional research is required to see how best to automate testing of the AWS backend (but all Lambda functions will be written in Node.js for easy testing across the front and backend..

IDE

Our team will use VSCode, because of how lightweight and flexible the code editor is. Additionally, we will use the Expo Tools extension to make Expo integration easy and seamless. The React Native Tools extension will be used for debugging, as well as Prettier and ESLint for formatting and linting respectively.

Next Steps

There are few things we need to take into consideration before moving on to the development process. First and foremost we want to present our technical plan to our stakeholders and to get their inputs about the plan and make the necessary changes required to fit the vision that they have for the application. This is a crucial step because we haven't made a set decision on our technical plans and we want to get as much information as possible before we proceed to make the final decisions.

One of the ambiguities that we still have about the application is if it's suitable and it's something our liaison would agree to if the application is serverless. There are many advantages

to why we should go serverless, but the main reason is so that we wouldn't have to worry about configuring the server as it's been taken care of by the service providers. On the other hand, there are also limitations to what we can do if we decide to go serverless such as response latency, debugging, etc. Thus, it's important to see if our liaisons have any preferences, and even if they don't it is our responsibility to make sure that they're well informed.

We also have a hard time choosing CICD tools because of the lack of experience and knowledge when it comes to CICD. We know that most development processes integrate a CICD pipeline because it allows faster code building and checking. Because of the automation that comes with a CICD pipeline, it helps the developers to save time and effort and also a quick identification of problems. This is an appealing tool to integrate but we first need to do research about which is the best CICD tool to use, especially in web applications. It's also good to discuss if we even should have a CICD pipeline.

Lastly, we also need to decide what libraries we can use to write unit tests. Since we'll be using React Native for Roomer, it is important for us to explore the different libraries that can help us with testing. A big part of it is knowing what libraries can also write unit tests for our lambda functions.

We're confident that if we follow these next steps it will tremendously help us with what we're trying to accomplish which is a simple system maintenance and also a simple development workflow.