Software Requirements Specification

for

ReverseHand

Conducted By:

Cache Money

Contents

1 Introduction

The vision for this project, is to assist in reduce any power imbalance that consumers may face when seeking trade services.
To achieve this, consumers will be given the opportunity to use a mobile application to make their need for services known to tradesmen. This application will accommodate both consumers, that must be able to post job adverts, and tradesmen, that must be able to register a profile and bid for these job adverts. Communication must also be facilitated between the two parties.

Here follows the relevant acronyms, abbreviations and definitions to accompany this document.

- consumer - a user who requires a handyman and creates job adverts for a handyman.
- tradesman - a user who bids on adverts. This is the aforementioned handyman.
- admin - a user who can monitor other users and make modifications e.g. delete accounts, fix unjust rating, etc.
- advert - the jobs that are created by consumers, that tradesmen can bid on.
- bid - a submission from a tradesman seeking to be hired for a job. The bid will always include a price, or a price range.

The rest of this document will contain a general description of the product including user characteristics, system requirements, which includes both functional and quality requirements, class diagrams, decomposition of the system, a traceability matrix based on this decomposition and finally, acceptance criteria.

2 General Description of Product

2.1 Context of Product

For the Consumer:
If a consumer would like to seek out a handyman for a job and cannot, or does not want to, go to a larger corporation, the consumer can find and independent tradesman instead.

For the Tradesman:
A tradesman can use this application to ease the process of finding business. This allows a tradesman to accept jobs as they have time, and gives them a variety of jobs to inspect and choose from.

For the Admin:
An admin would like to see useful insights and metrics that have use to either

developers or owners of the system, as well as monitoring disputes that arise between users of the system.

2.2 Product Function

This product will need to be able to connect consumers and tradesman to one another through adverts from consumers and bids from tradesman. The consumer creates an advert and the tradesman places a bid on the advert. Then, the consumer can accept a prospective bid and the two different kinds of users can go into business with one another.

2.3 User Characteristics

User 1: Consumer The primary use of the application for the Consumer will be to find well-suited Tradesmen for any jobs they need done, at a price that they feel comfortable paying. The Consumer will be connected with a variety of Tradesmen. These Tradesmen have different skills, ratings, and price bids- all of which the user will be able to evaluate to make a choice that they feel is right for them. Consumers will be able to shortlist a bid if they are interested in it, and later accept a bid from the pool of shortlisted bids when they feel satisfied to do so.

User 2: Tradesman The primary use of the application for a Tradesman will be to find jobs that they are equipped for in a convenient manner. This application will give Tradesmen an alternative way to gain new clients. Tradesmen will also be given an opportunity to build a good reputation with positive customer ratings. Tradesman will be able to specify a general price range on any advert bid, and give a more detailed quote if their bid has been shortlisted.

User 3: Admin Administrative users will interact with the administration system.

2.4 Constraints

- The system will only be available in English.
- There will not be an accompanying web app.
- One account cannot be a consumer and a tradesman account, two separate accounts are required.
- The registered email for an account cannot be edited.

2.5 Assumptions and Dependencies

- The application is dependent on there being a large amount and a large variety of tradesman as well as consumers for the tradesman to interact with.

- The application is dependent on consumers and tradesman being able to agree on payment and such matters.
- The assumption that only tradesmen, and people seeking tradesmen, will use the app.

3 Class Diagrams

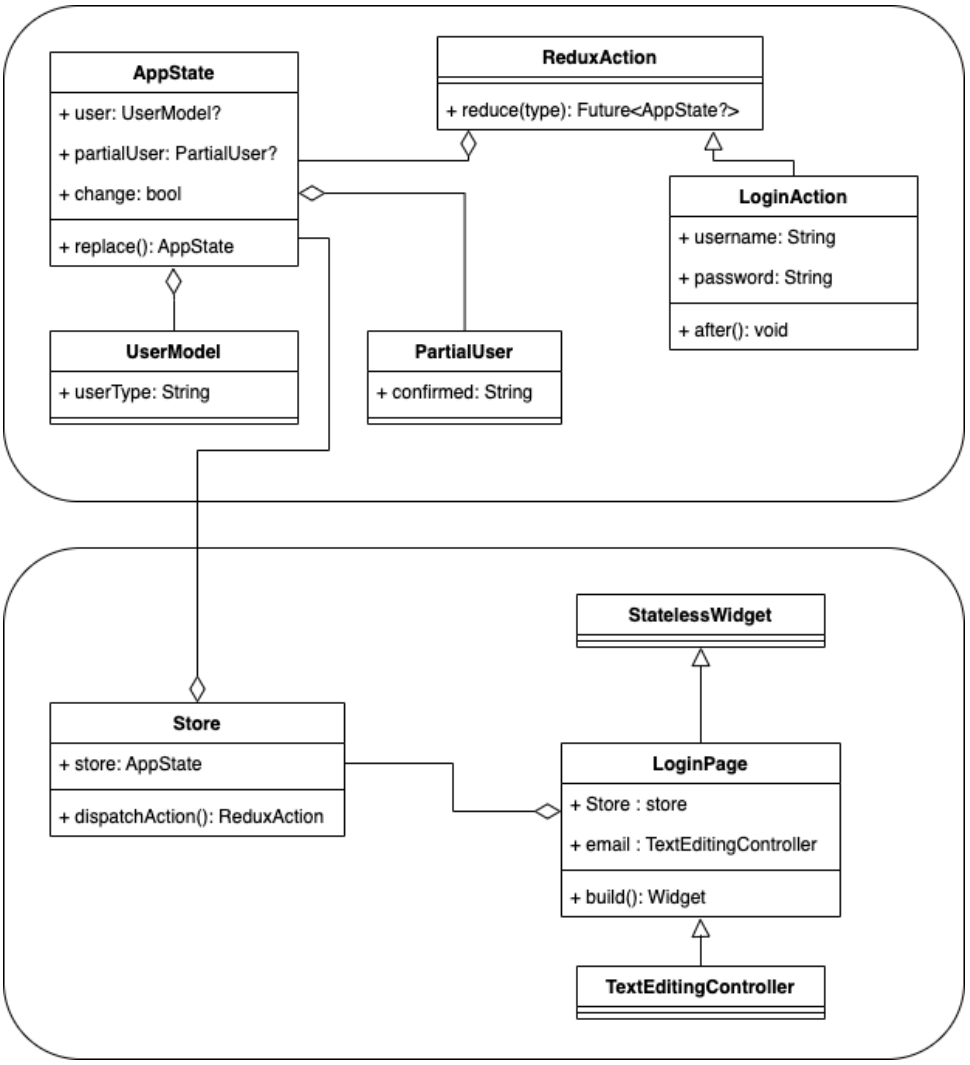

Figure 1: Class Diagram showing interaction with back-end for Login use case.
This excludes the ViewModel as it constructs the store and is discussed later.
Note: This is a simple diagram and each aspect is shown more in depth in subsequent diagrams

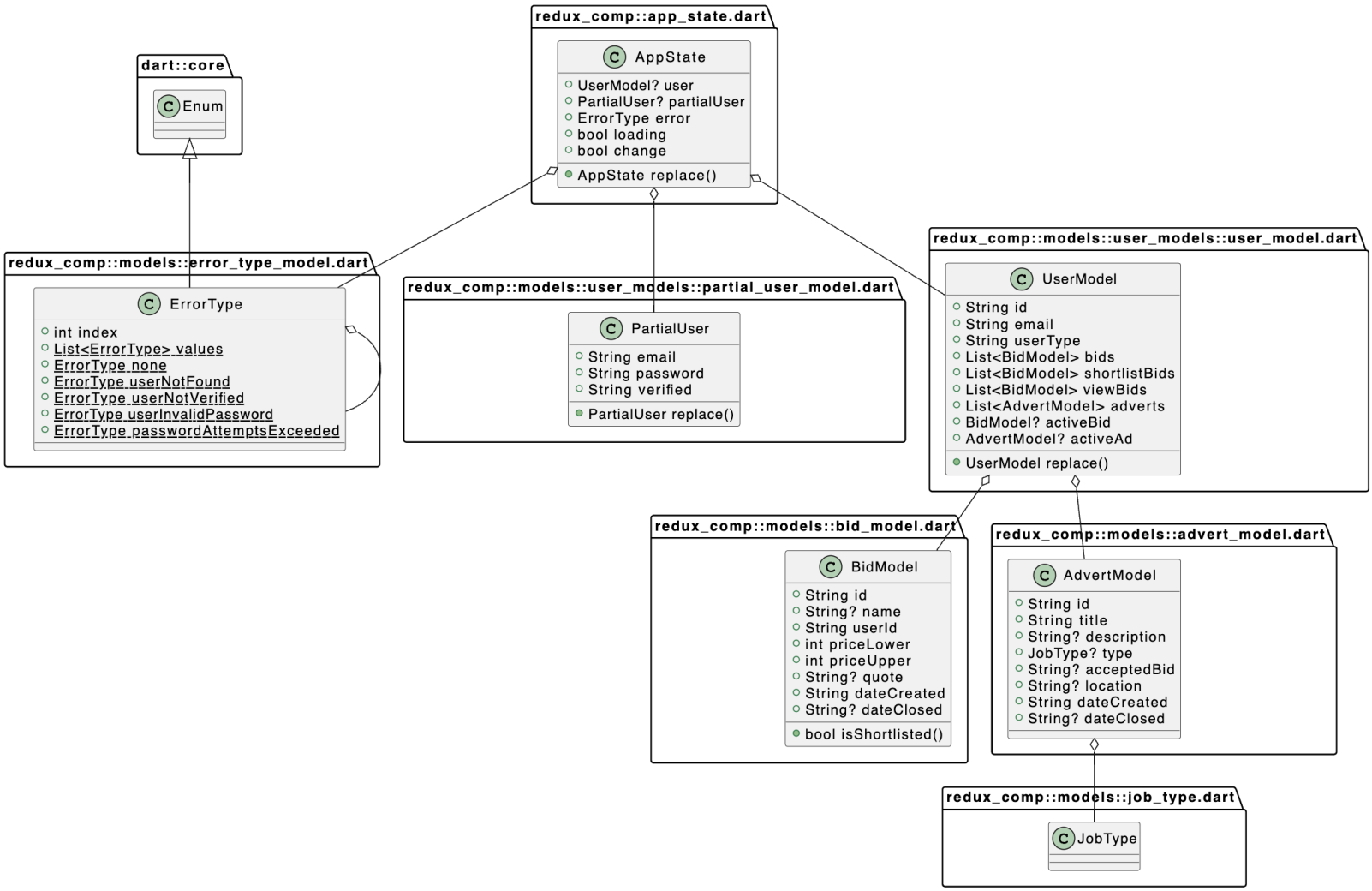

Figure 2: Class Diagram of the App State

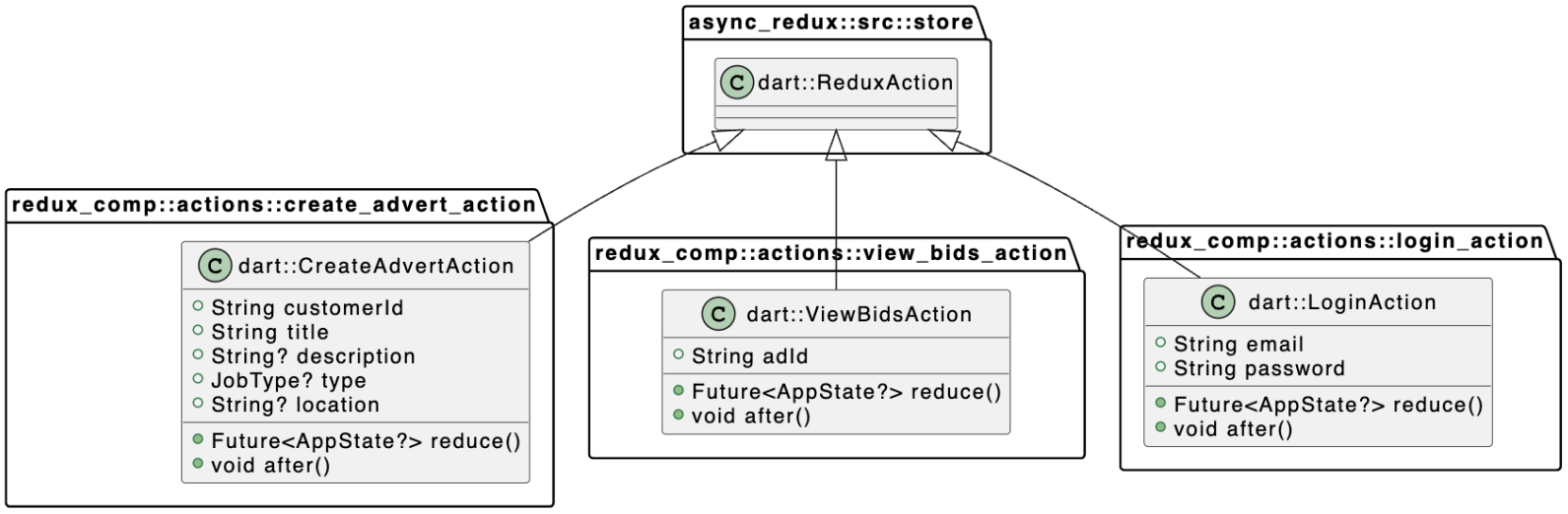

Figure 3:
Class Diagram of some of the Redux Actions
The logic is executed in the reduce function
The after() function is executed after the function

3.1 ViewModel

The ViewModel(Vm) is how the frontend interacts with the state.
It has a VmFactory that constructs the view model with the relevant information needed by the UI.
You can access this ViewModel by making use of the Store which is a variable in every page.
Not all pages are here as they have much the same structure.
Only the ViewModel changes.

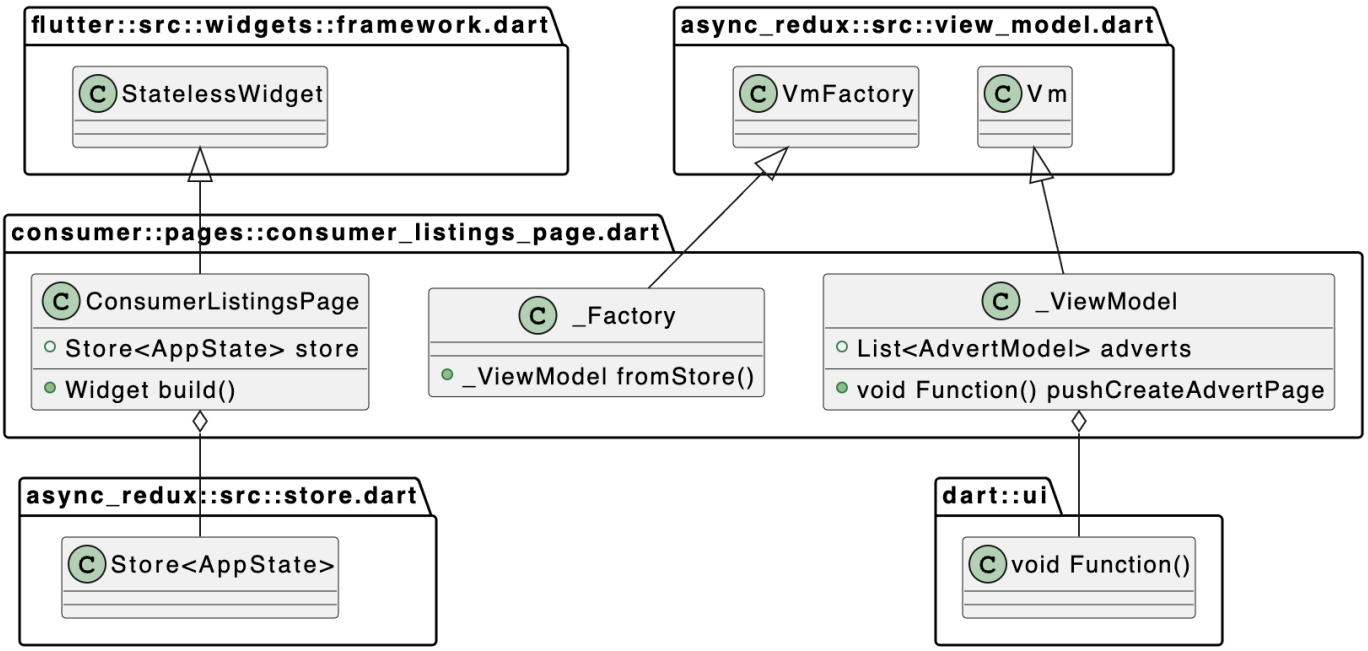

Figure 4: Class Diagram of the Consumer Listings Page

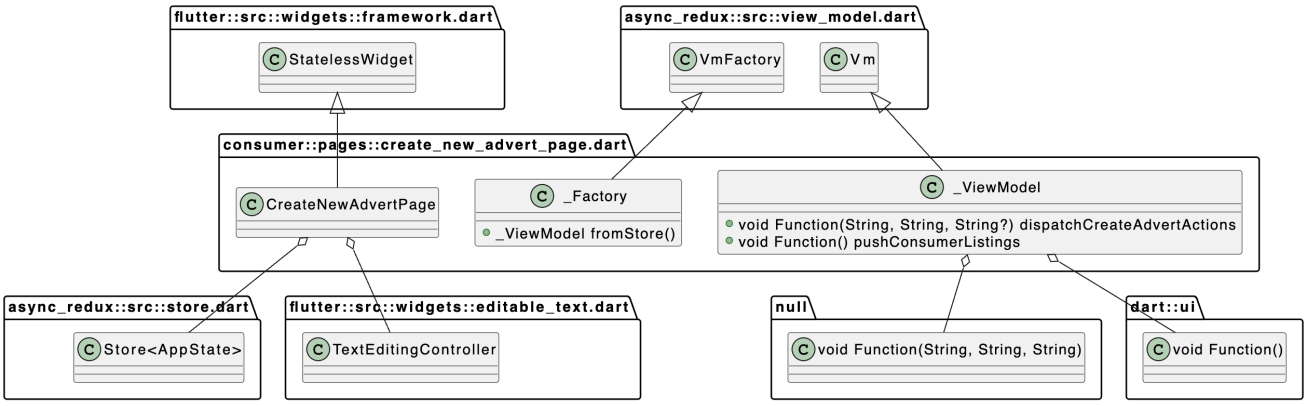

Figure 5: Class Diagram of the Consumer Create Advert Page

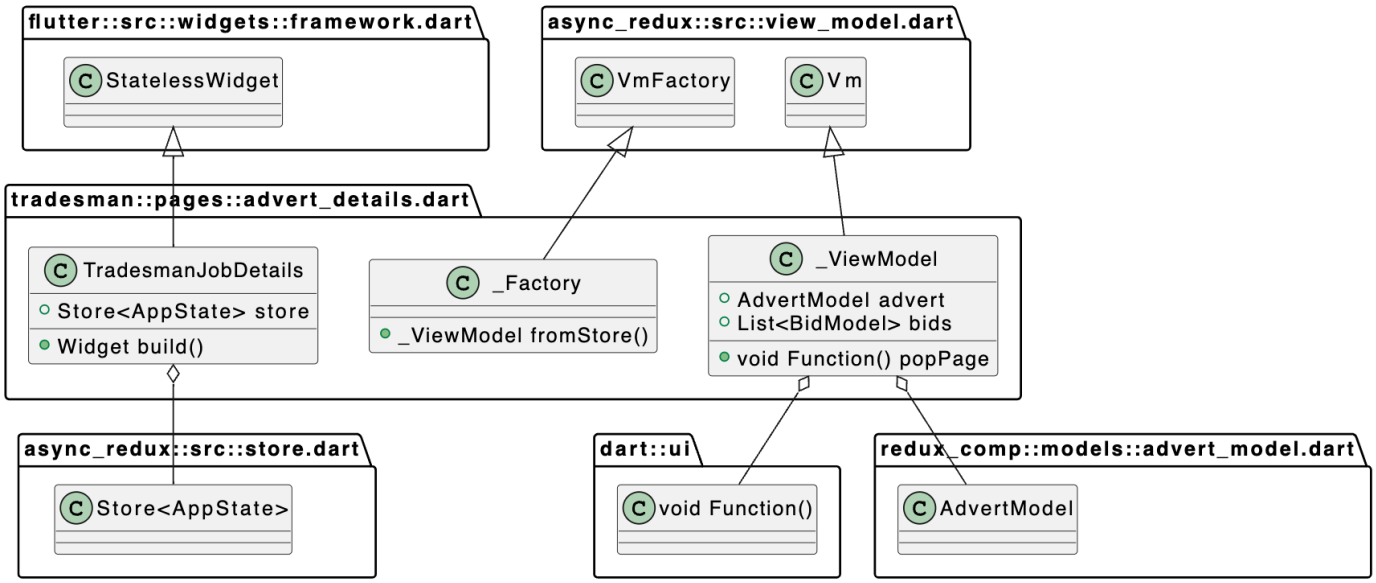

Figure 6: Class Diagram of the Tradesman Advert Details Page

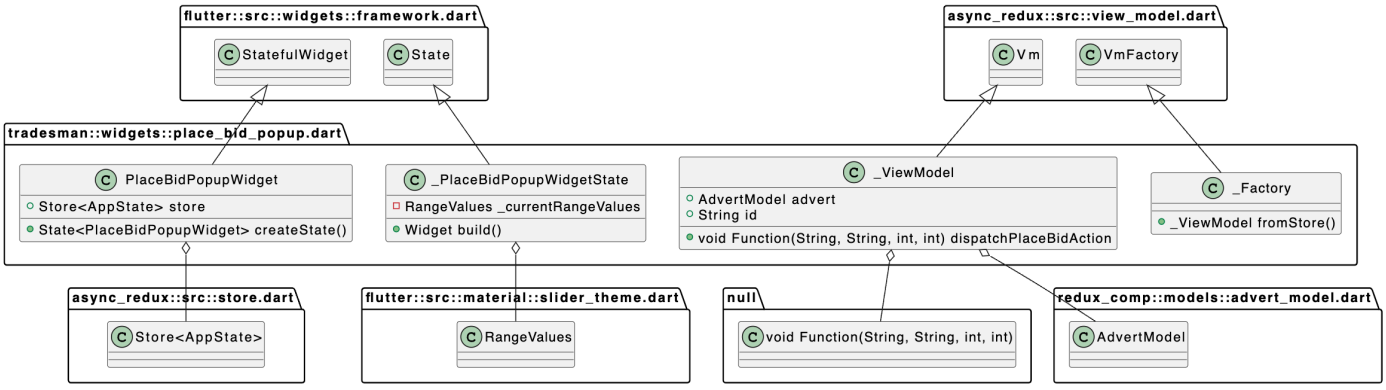

Figure 7: Class Diagram of the Tradesman Place Bid Pop Up

4 System Requirements

4.1 Use cases

- UC1: View Profile
- UC2: Post Job Advert
- UC3: Leave Review
- UC4: Register
- UC5: Find Jobs
- UC6: Place Bids
- UC7: Notifications
- UC8: Choose Bid
- UC9: View Bids
- UC10: Shortlist Bid
- UC11: Close Job
- UC12: Delete and Edit Adverts
- UC13: Delete and Edit Bids
- UC14: Filter Adverts and Bids
- UC15: Geolocation
- UC16: Upload Quote
- UC17: Chat App

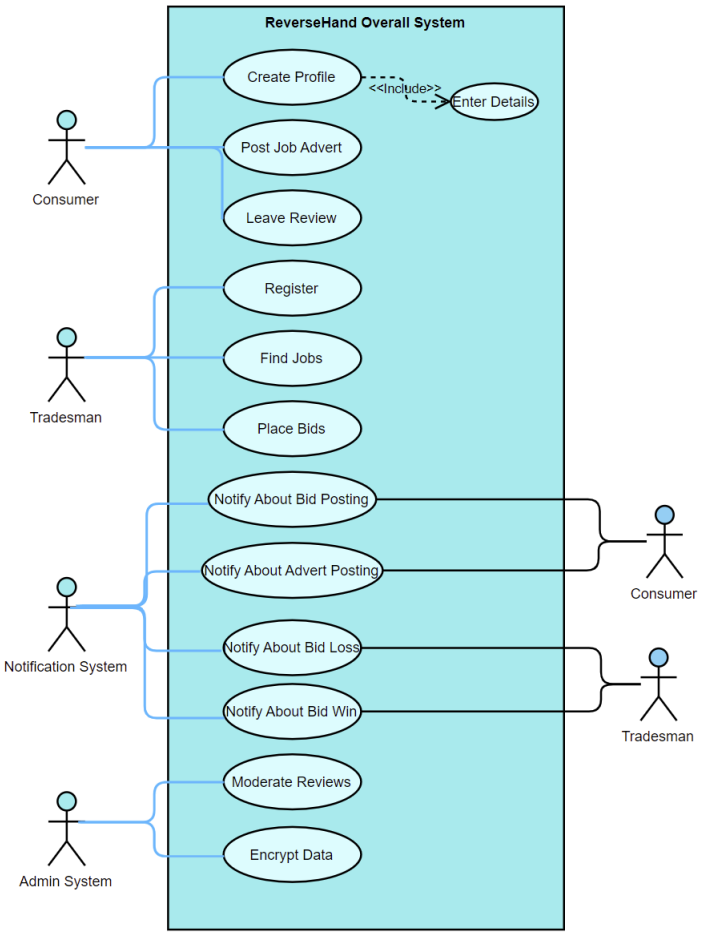

Figure 8: Overall System Use Case Diagram

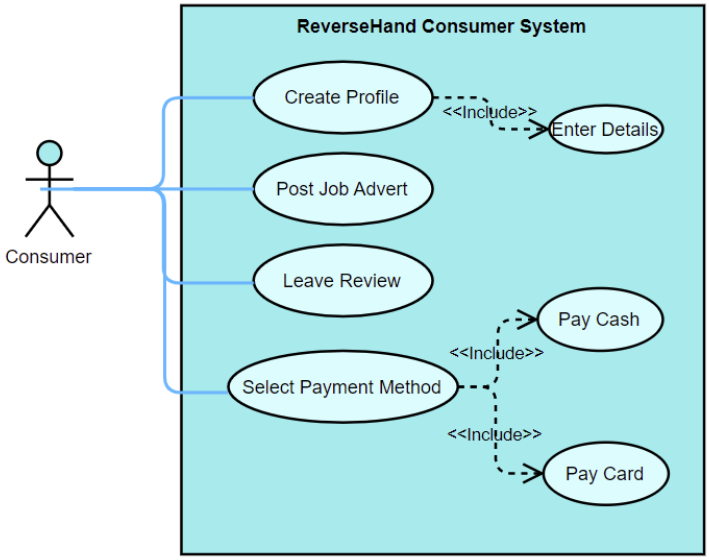

Figure 9: Consumer System Use Case Diagram

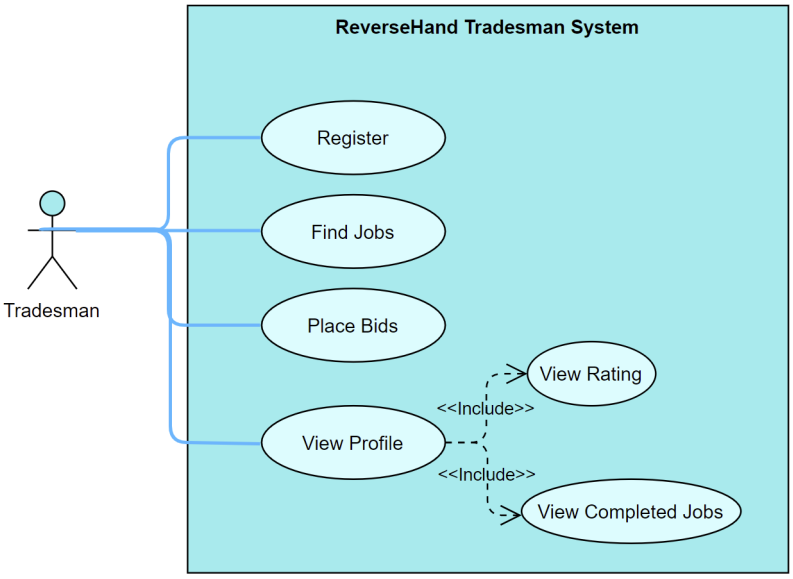

Figure 10: Tradesman System Use Case Diagram

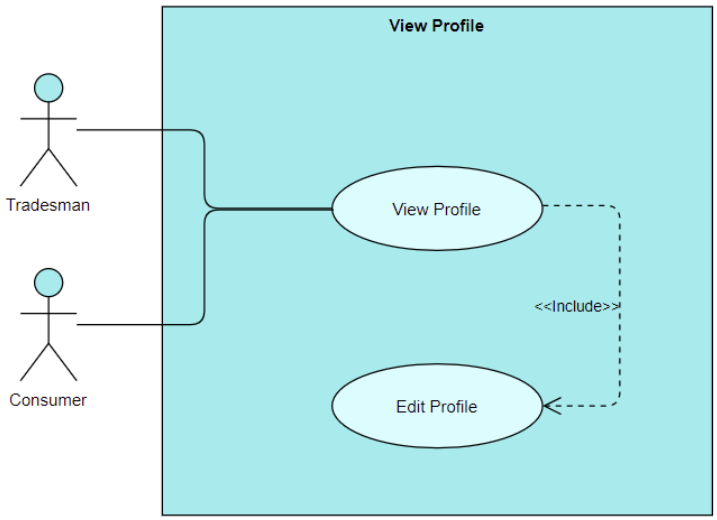

Figure 11: View Profile Use Case Diagram

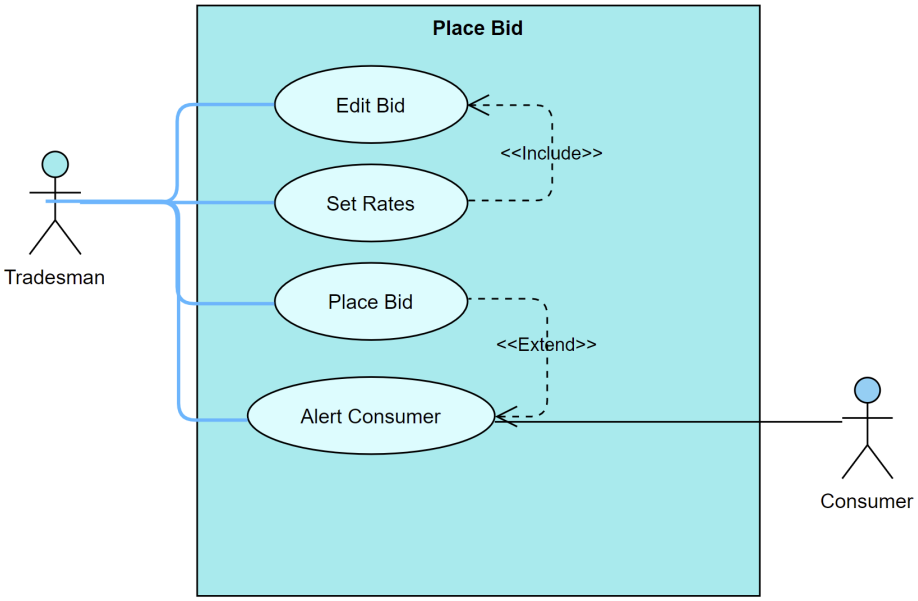

Figure 12: Place Bid Use Case Diagram

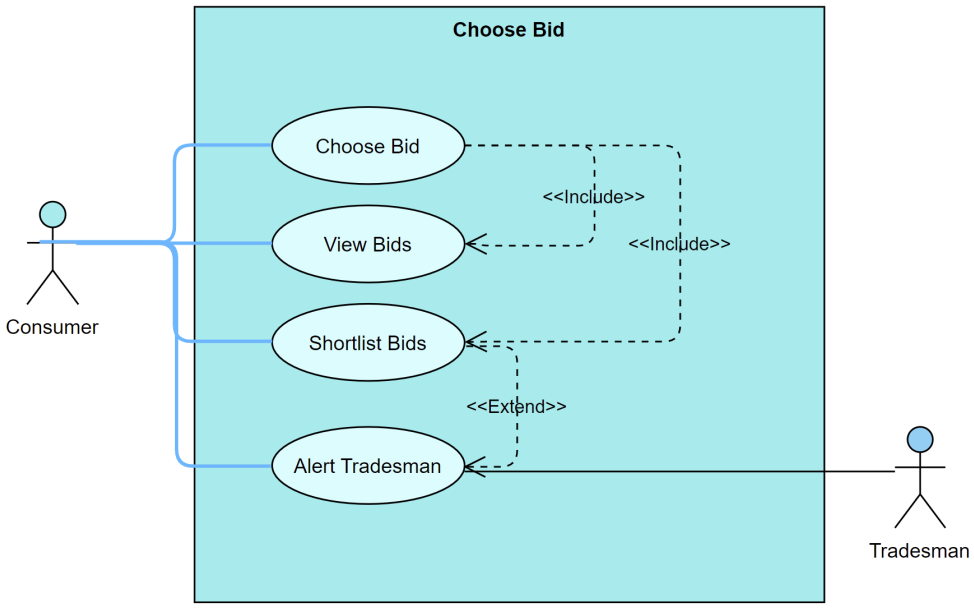

Figure 13: Choose Bid Use Case Diagram

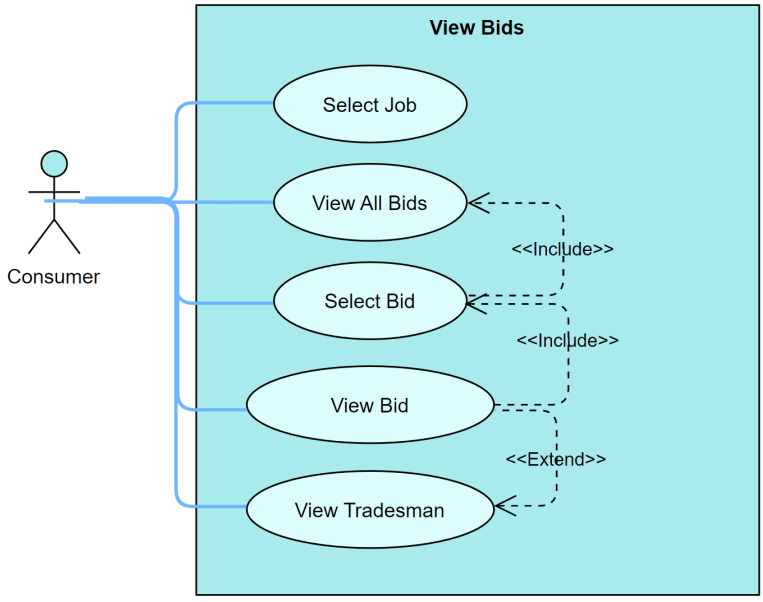

Figure 14: View Bids Use Case Diagram

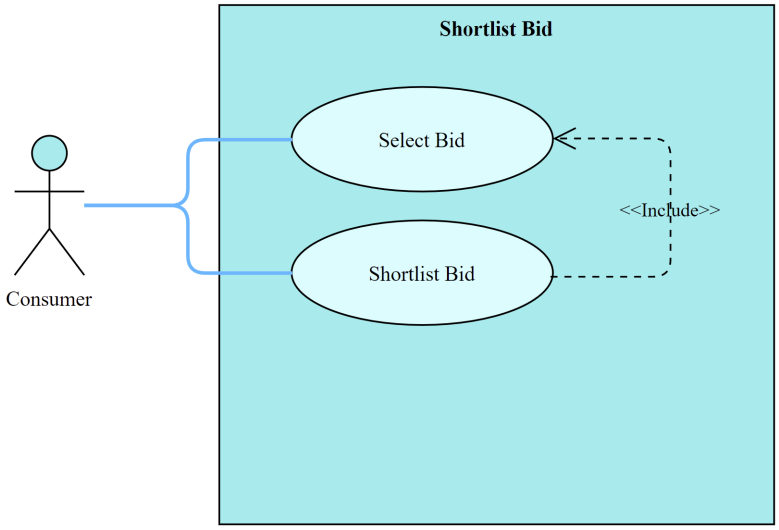

Figure 15: Shortlist Bid Use Case Diagram

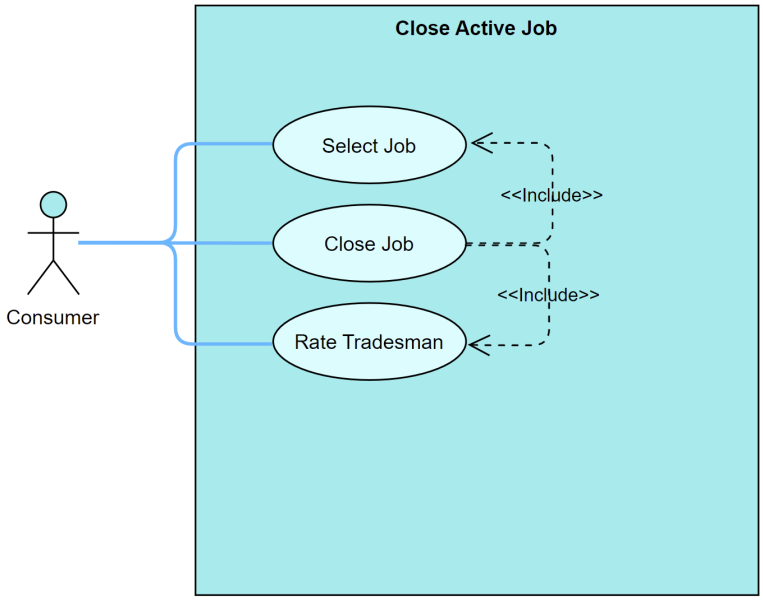

Figure 16: Close Active Jobs Use Case Diagram

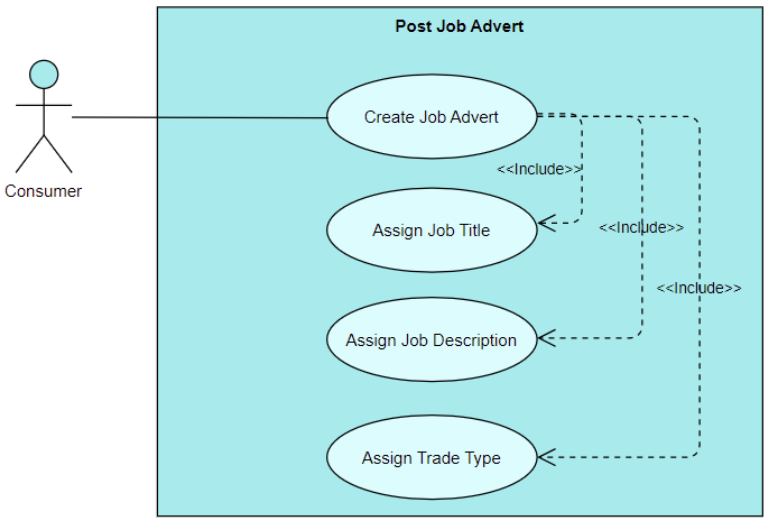

Figure 17: Post Job Advert Use Case Diagram

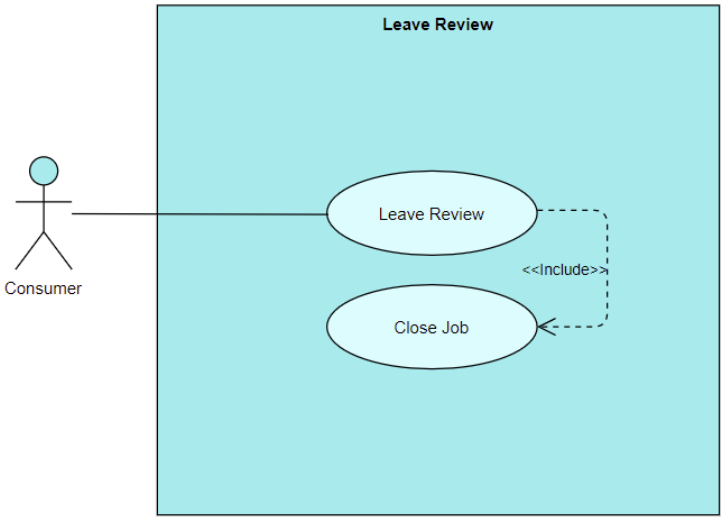

Figure 18: Leave Review Use Case Diagram

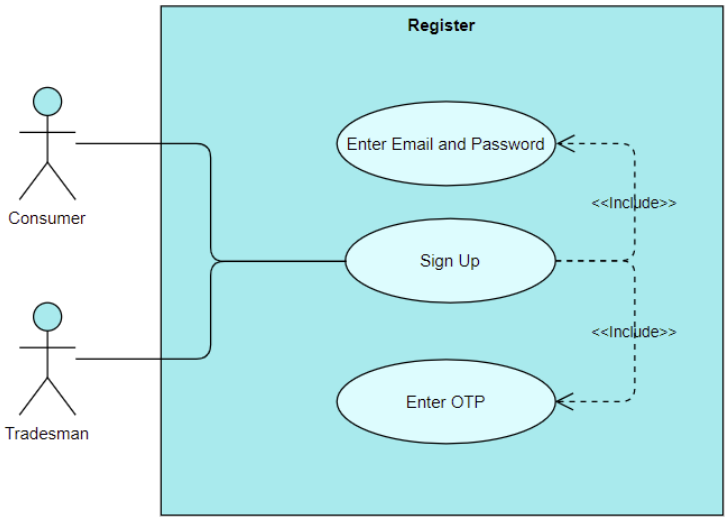

Figure 19: Register Use Case Diagram

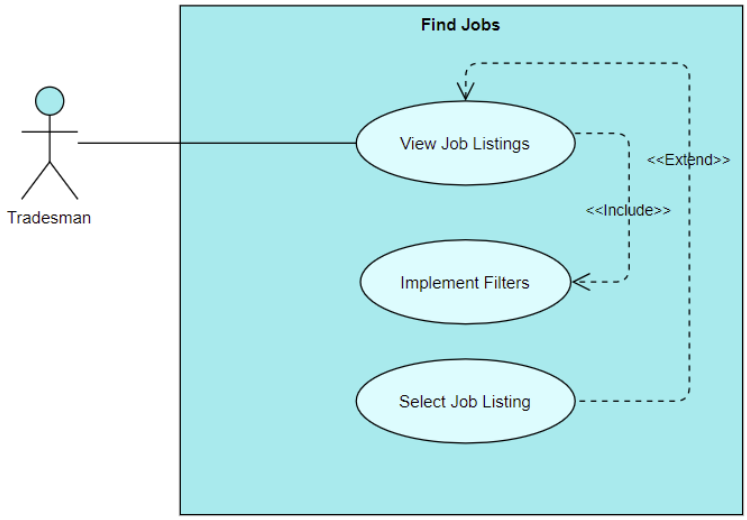

Figure 20: Find Jobs Use Case Diagram

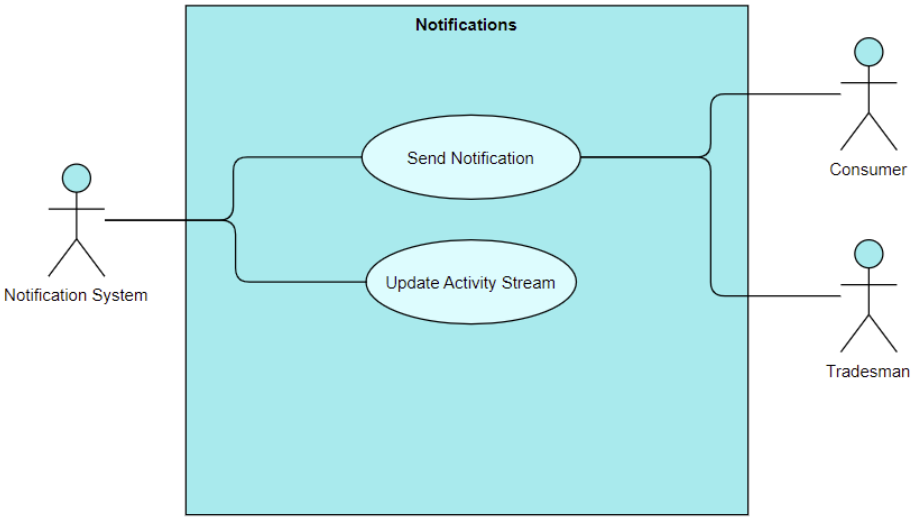

Figure 21: Notifications Use Case Diagram

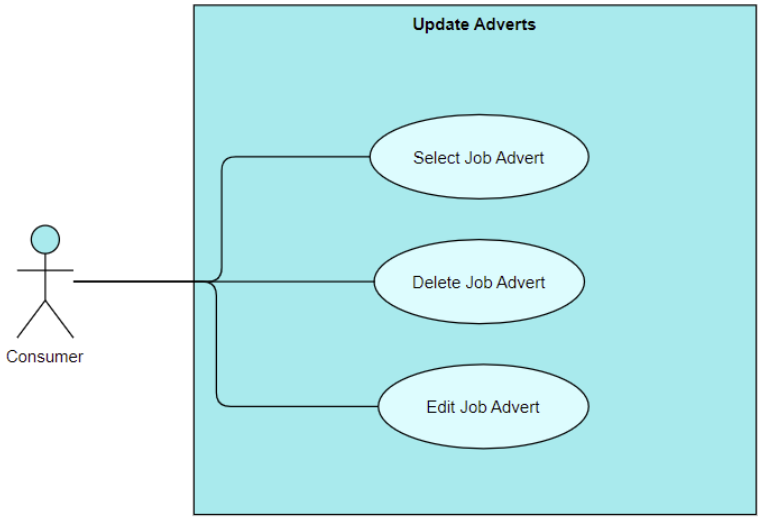

Figure 22: Update Adverts Use Case Diagram

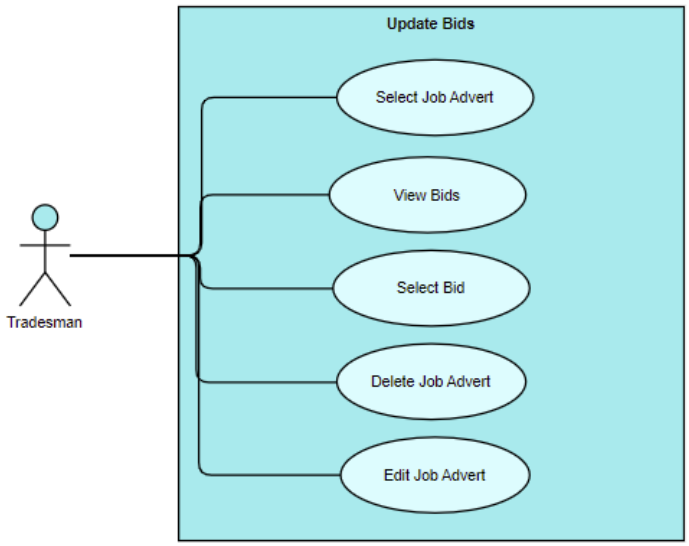

Figure 23: Update Bids Use Case Diagram

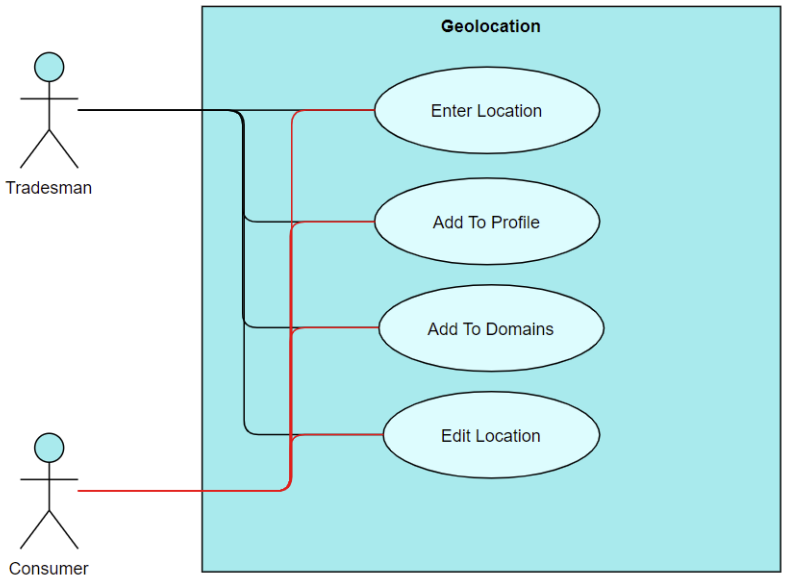

Figure 24: Geolocation Use Case Diagram

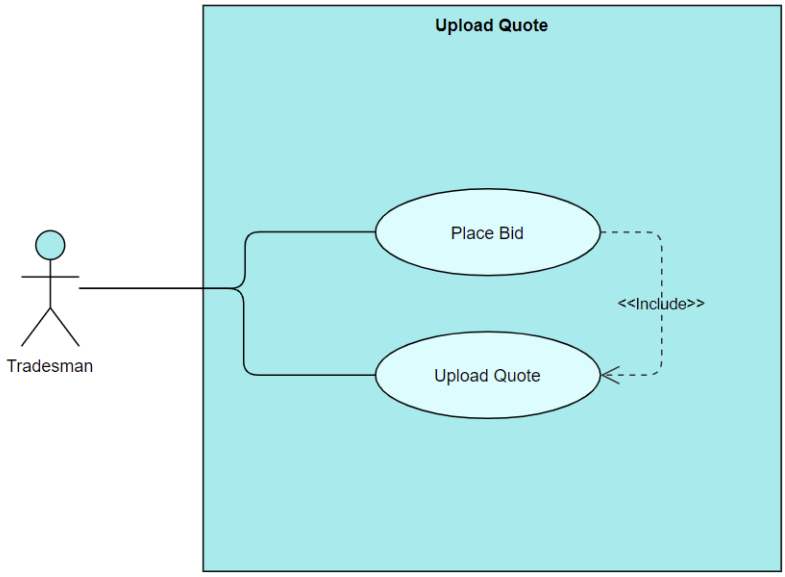

Figure 25: Upload Quote Use Case Diagram

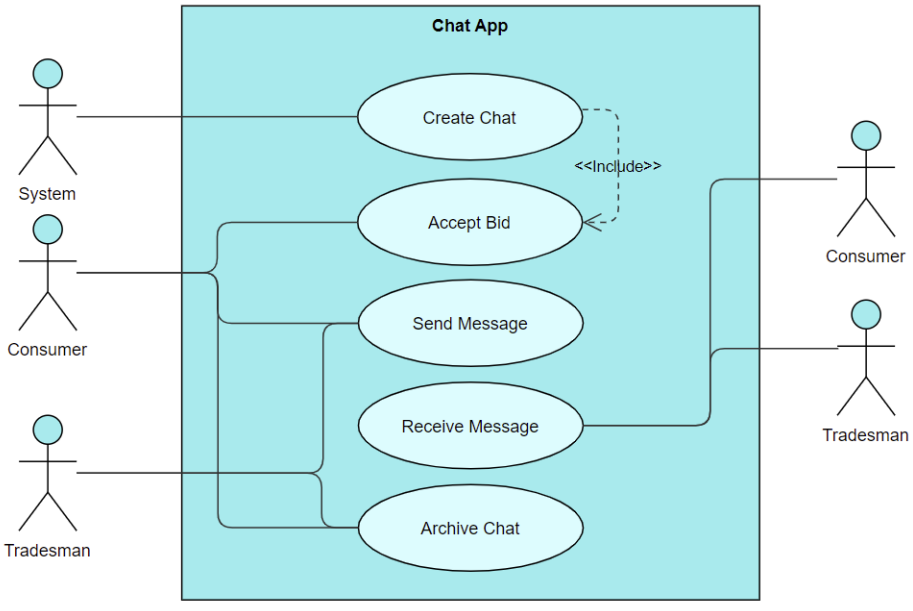

Figure 26: Chat App Use Case Diagram

4.2 User Stories

Consumer:

- As a consumer, I want to create and post a job advert so that I can find a tradesman that is well-equipped for the job, at a price that I am comfortable with.
- As a consumer, I want to see all the active job adverts that I have posted so that I can monitor them.
- As a consumer, I want to open a detailed view of a job advert to see what bids have been placed.
- As a consumer, I want to open detailed bids made by tradesman so that I can see any detailed information of a bid.
- As a consumer, I want to shortlist bids that I am interested in to receive a more detailed quote breakdown from a tradesman.
- As a consumer, I want to accept a bid of a tradesman that I am happiest with.
- As a consumer, I want to a job to be closed when I have accepted an advert so that I do not receive any additional unnecessary bids.
- As a consumer, I want to create a profile with my relevant information so that I can change my personal details after sign-up without losing my job adverts.
- As a consumer, I want to login with my details to access my account with a Consumer view.
- As a consumer, I want to filter the bids made on my jobs to easily find bids that fit my criteria.
- As a consumer, I want to be put in contact with a tradesman once I have accepted their bid.

Tradesman:

- As a tradesman, I want to browse job adverts so that I can determine if I am interested in bidding for one.
- As a tradesman, I want to open a detailed view of a job advert to see all the information relating to the job.
- As a tradesman, I want to place a bid on a job to indicate my interest to the consumer at a price I am comfortable with. This bid should be a general price range.
- As a tradesman, I want to upload a more detailed quote breakdown if a consumer has decided to shortlist my bid.

- As a tradesman, I want to be made aware when I have been shortlisted for a job so that I can upload the relevant quote in time.
- As a tradesman, I want to be made aware when I have won a bid for a job so that I do not miss the opportunity.
- As a tradesman, I want to create a profile with my relevant information so that I can change my personal details after sign-up without losing my bids and reviews.
- As a tradesman, I want to login with my details to access my account with a Tradesman view.
- As a tradesman, I only want to see jobs that take place in relevant locations to me.
- As a tradesman, I want to see other bids that have been made on an advert so that I can provide a competitive price.
- A a tradesman, I want to be put in contact with a consumer once they have chosen my bid.

4.3 Functional Requirements

- **FR1:** A User must be able to Sign Up and Login to the relevant partitioned app.
- **FR2:** A User must be able to edit their Profile.
- **FR3:** A User must receive notifications when relevant actions occur.
- **FR4:** Relevant Consumers and Tradesman must be able to communicate with each other without exposing private information.
- **FR5:** A Consumer must be able to post and edit an advert.
- **FR6:** A Consumer must be able to shortlist and accept a bid.
- **FR7:** A Consumer must be able to record their home address.
- **FR8:** A Consumer must be able to rate a Tradesman after a job has been completed.
- **FR9:** A Tradesman must be able to place bids, and see bids, on adverts they are interested in.
- **FR10:** A Tradesman must be able to see adverts in the locations they are willing to travel to.
- **FR11:** An Admin System must be implemented.
- **FR12:** The system must encrypt any data end to end to ensure data security.
- **FR13:** The system must allow for automatic code testing and validation when changes are made.
- **FR14:** Consumers must be able to see the adverts they have posted.

4.4 Quality Requirements

- **QR1:** The system must not expose any personal details of any of its users unless necessary.
- **QR2:** The system must be easy to navigate and pleasing to the eye.
- **QR3:** The system must not take longer than 10 seconds to list jobs available for bidding.
- **QR4:** The system must be able to scale depending on the amount of current users.
- **QR5:** The system must be able run on any modern smartphone.
- **QR6:** The system must support only mobile interfaces.
- **QR7:** The UI must be fast and responsive.
- **QR8:** The system must run on the latest iOS and Android OS.

5 System Decomposition

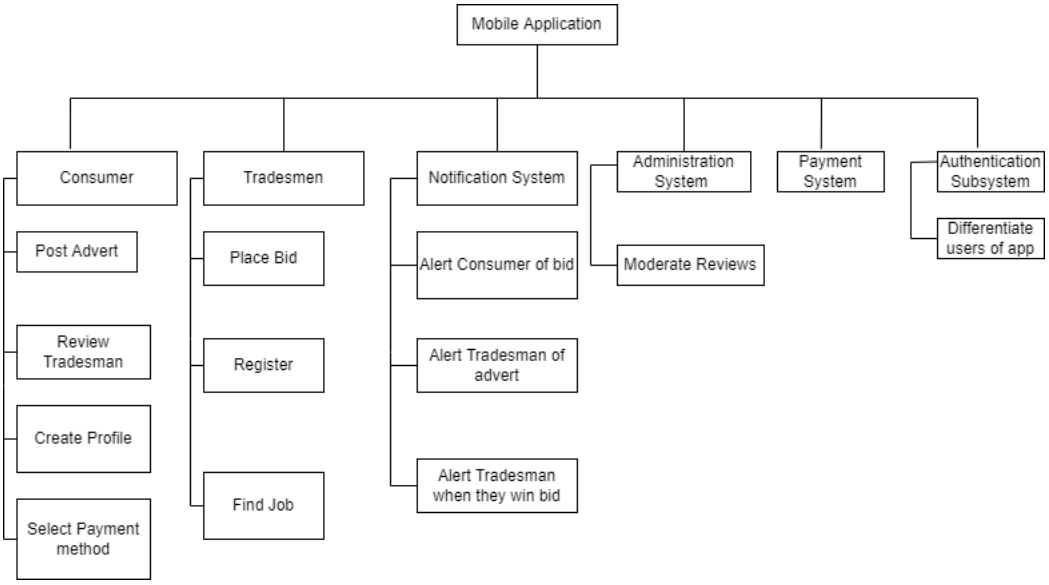

Figure 27: Systems Functional Decomposition Diagram

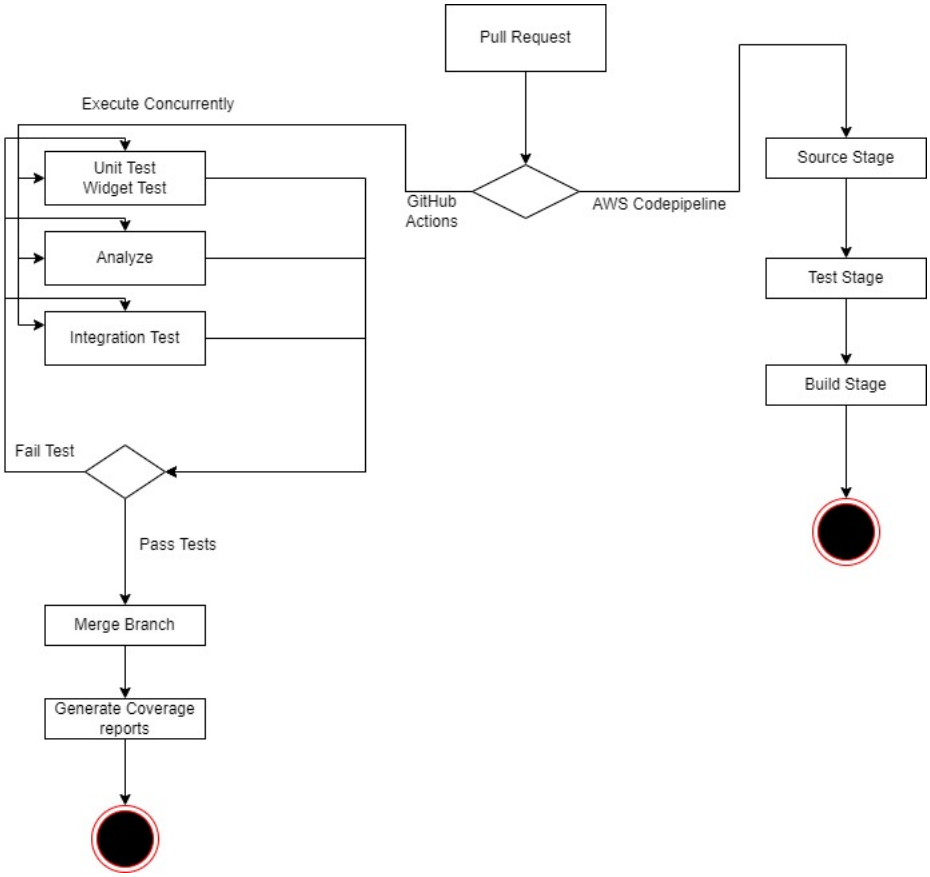

Figure 28: Pipeline Decomposition

6 Traceability Matrices

Subsystems					
	Consumer	Tradesmen	Notification	Administration	CI/CD
FR1	x				
FR2		x			
FR3		x			
FR4			x		
FR5				x	
FR6				x	
FR7	x				
FR8	x				
FR9			x		
FR10	x				
FR11	x				
FR12					x
FR13				x	

Use Cases												
	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12
FR1		x										
FR2				x	x	x						
FR3	x											
FR4							x	x				
FR5												x
FR6												
FR7			x									
FR8	x											
FR9										x		
FR10												
FR11												
FR12												
FR13												

7 Acceptance Criteria

All functional requirements must be met as well as hosting of the API and any other required back end services (e.g. database).
This includes:

1. Allowing a consumer to post a job and a respective tradesman to place bids.
2. Allow both tradesman and consumers to register an account.
3. Allow a tradesman to be rated and the rating to be displayed.

8 Architectural Requirements

8.1 Architectural Design Strategy

The decomposition strategy is being used. Our application can already very easily be divided into two main parts- the Tradesman subsystem and the Consumer

subsystem. Each of these subsystems consist of smaller subsystems (which sometimes overlap).
Our strategy is to identify subsystems through the use of user stories and what their requirements will be.

8.2 Architectural Styles

We are using a modified redux pattern. Our form of the redux pattern does not contain any middleware. Instead, our reducers are async reducers. Our actions and reducers are also combined together underneath one class where the class represents the action and the reduce function represents the reducer.
All our navigation will be done defined in the app.dart file as this is the entry point to our program. Navigation is done through the redux pattern and can be accomplished in one of two ways:

1. A dispatch action moves us to a new page.
2. An action moves the page after it has altered the state.

The system is component-based and consists of a smaller number of loosely coupled components, specifically:
Resolvers (which interact with the DB) are loosely coupled to reducers
Reducers/Actions all inherit from a shared class and act as a bridge between the UI and fetching data stored in the state
The UI is made up on a set of reusable components - some packaged by flutter, others are custom components designed by us

8.3 Architectural Design and Pattern

- Cloud Computing - The system is making use of cloud computing. This is because it allows for all cloud resources to easily and dynamically scale, thereby allowing for unlimited growth. The system makes use of the following kinds of cloud computing: IaaS, SaaS, MBaaS and FaaS.
- Redux Pattern - The redux pattern is being used to manage the state.
- Pipeline - The output of the Database is the input of resolvers. The output of resolvers is the input of reducers. The output of reducers is the input of the UI. The entire process flows in the opposite direction as well.
- Event-Driven - The redux pattern by design is an event-driven architecture.

8.4 Architectural Constraints

All cloud services must be AWS services as our client is AWS. Preferably, using AWS EC2 should be avoided.
The system must follow best practices where possible, e.g. using a single table in a NoSQL database instead of a multiple tables with relationships.

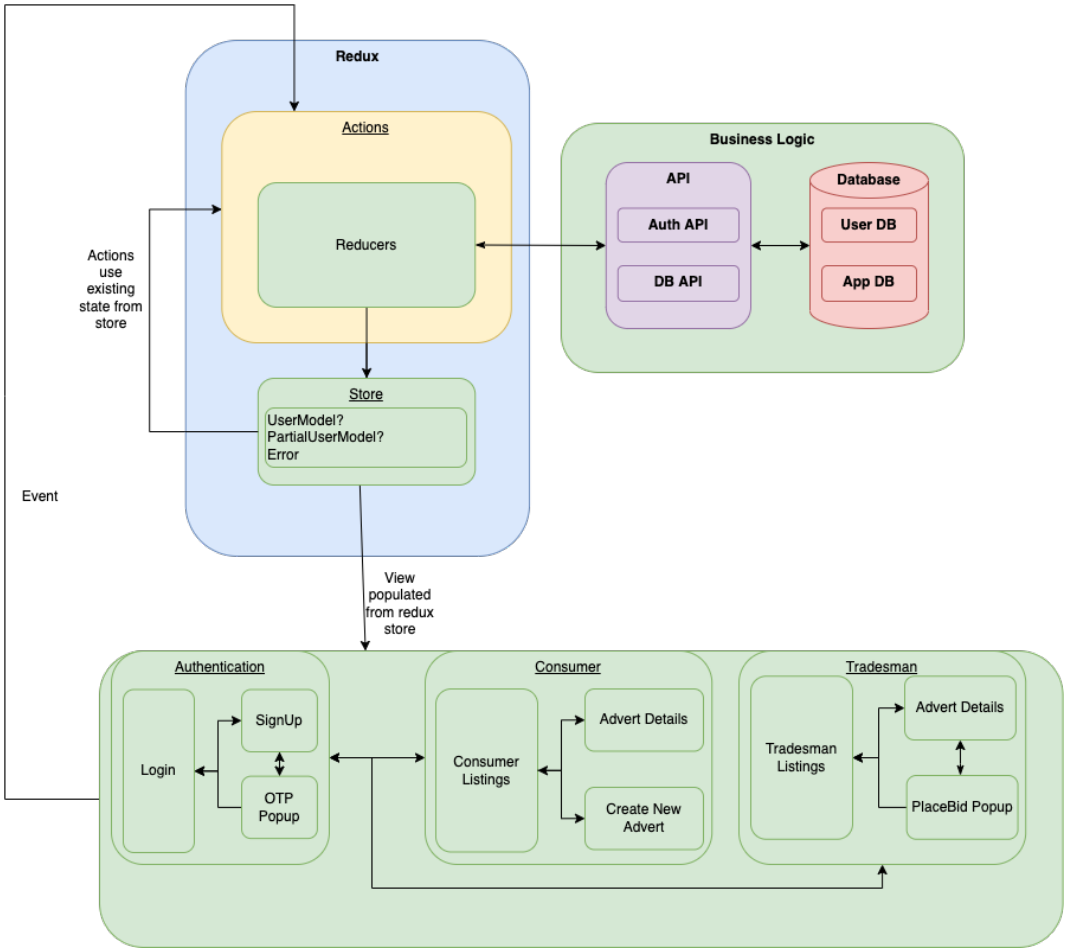

Figure 29: Overview Diagram of the ReverseHand System

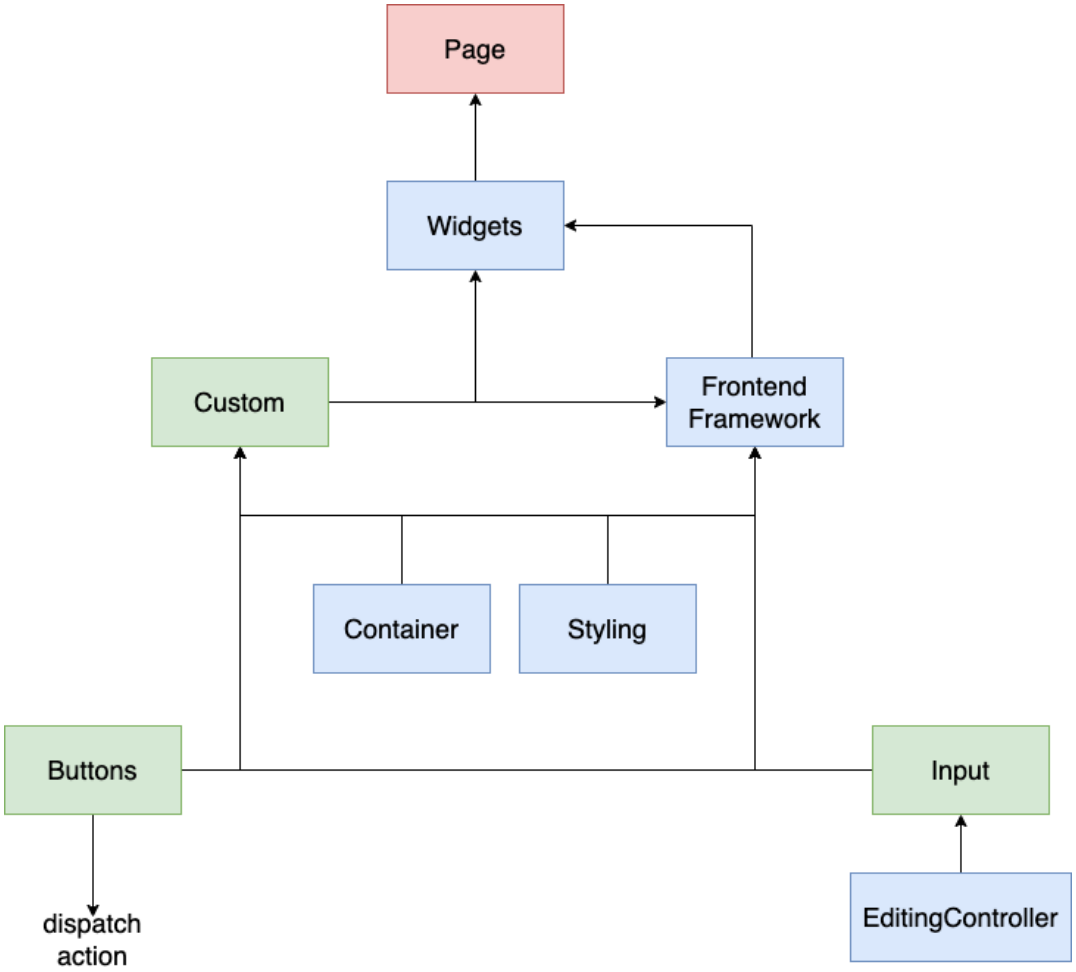

Figure 30: Diagram of a component subsystem.

Figure 31:
Diagram of a single table mock database.
The goal is to be able to fetch all the relevant information related to your access patterns in *at most* one query.
An access pattern is similar to a use case for the DB
eg: Get all adverts a customer has posted - Query"Partkey" : "C001"

8.5 Architectural Quality Requirements

1. Administrability - The system must be able to administer user reviews.
2. Availability - The system must be available 99% of the time with essentially no downtime.
3. Confidentiality - The system must not share any user data without the user's permission.
4. Durability - The system must not lose data if there is a crash.
5. Usability - The system must allow users to achieve the task effectively.
6. Accuracy and Precision - Any geolocation data used must hold accurate and precise information as to not provide bad information. The accuracy of the users location must not be off by greater than 1km.
7. Affordability - The system must be affordable for the client. The system must make other bids aware to a tradesman to allow them to post a more competitive price.
8. Debuggability - The system must be comprised of separate components that are easy to debug, to give future developers the ability to isolate sections of code where there are issues.
9. Dependability - The system must be dependable and not falter in any area. The system should not experience any crashes under any condition.
10. Deployability - The system must be easily and automatically deployed.
11. Discoverability - The system must allow for easy finding of data and metadata in an easy manner. The user must be able to view data on theme selves and select data of others.
12. Failure Transparency - The systems must not make the users aware of failures.
13. Fault Tolerance - The system must be able to handle errors gracefully and all errors must be recoverable.
14. Installability - The system should be easy to install.
15. Learnability - The system must be easy to learn and master.
16. Maintainability - The system must be easy to maintain to allow future changes to be quick and painless.
17. Reliability - The system must notify the user if any process or operation does not complete.
18. Responsiveness - The system should not take longer than 5 seconds to connect, assuming good a connection.

19. Robustness - The system should handle bad input gracefully and allow the user to reenter their input.
20. Scalability - The system should dynamically scale to the amount of resources required to provide optimal functionality.
21. Simplicity - The system should be simple, but still powerful, such that a user can easily learn how to use the app.
22. Testability - The system must be easy and intuitive to test.

8.6 Technology Choices

Frontend
The app's frontend is making use of the Dart programming language along with the Flutter framework. The reason this was chosen over other popular frontend mobile frameworks is because it works the best on the M1 MacBooks being used in the team. The manner in which the UI is built was also very appealing to the team, as well as only needing to use one language instead of three (HTML/CSS/TypeScript).
Using flutter instead of something such as React also has the benefit of being easy to use with debug tools since only one language is used.
Flutter also has an excellent redux library called async_redux which makes managing state very easy.

Backend
A heavy reliance on AWS services will be present, mostly on AWS Amplify.
Are system should be able to be used around the world and AWS services are globally available meaning that it is a perfect fit for our system.
A large benefit of Amplify (specifically the services which Amplify wraps) is that everything can dynamically scale as needed, or shrink with peak traffic times, or as the user size changes. This will allow the client to save money by ensuring they do not pay any more than is necessary.

AWS Amplify is a wrapper for other AWS services. These include:

- AWS DynamoDB
- AWS Cognito
- AWS AppSync
- AWS Lambda

Why we chose these technologies

DynamoDB is a key-value NoSQL database. The reason we went with this database is twofold. Firstly, it is Amplify's default and the recommended database to use with Amplify. Secondly, due to the scalability and performance, we can allow for a large amount of bids as well as allowing us to perform data analytics on a large amounts of data.

DynamoDB can also scale to manage an increase in query operations so there is Scaliablilty is achieved through DynamoDB structure and single table design. DynamoDB requires a primary key in every table, this primary key is then hashed and stored in the database. This implies that the amount of data stored in the database does not affect the speed of a queries execution. Performance is achieved through the AWS best practice of Single Table design. This means that all data is stored in a single table, and a this eliminates the DynamoDB equivalent of the JOIN operation. We are making use of a composite primary key, which is comprised of a unique partition and sort key. The enables us to retrieve relevant data in at most one query. This reduces the time taken to perform the query and the number of read operations required, and therefore reduces the cost to use the database.

Furthermore the benefits of single table design cascade down the system pipeline, as the Lamda resolvers accessing the DB will execute fairly quick, driving down their cost as well.

Cognito is a user authentication management tool which allows us to integrate many features into our sign-up, log-in and log-out. In order to make a truly secure and production ready user authentication system is a project on it's own that requires a lot of expertise. Therefore, it makes sense to use Cognito to deliver a secure authentication process and take full advantage of Cognito features.
These features inlclude:

- On user signup an OTP can be sent to verify a user through email, or cellNo, or both.
- Signup with third party services, e.g. Google.
- Streamlined password management
  - Password security specification
  - User verification on password reset
  - Password security
  - Multi-Factor Authorisation
- An admin API for administrators to manage users
- You can specify user groups to neatly partition your users into groups with different access to your apps feature.
- Will cache user credentials on device so the user doesn't have to re-login every time.

This allows us to focus on the usability of the authentication process over the security.

AppSync manages the GrapQL schema and the API. This includes the endpoint. The benifit of AppSync is that it combines the API's of different AWS services into one coherent API to be used by the application. It was made to specifically be used with amplify and provides an easy way for us to integrate our own custom resovlers in the form of lambda functions as well as a easy to use graphql playground.

AWS Lambda is serverless functions which contain the code for our GraphQL resolvers. AppSync calls these resolvers which in turn interface with the database. The functions are provisioned with AWS Cloud Formation.
We use Lamda as our app needs to be as cheap as possible to host and to be able to dynamically provision more services to the backend as needed. Therefore Lambda being serverless accomplishes both these things being a serverless function.

Additionally, AWS Codepipeline, AWS Codebuilld and AWS Code Deploy are used and more information is given below.

Testing
Flutter has built in testing capabilities for unit, integration and E2E testing.

Monorepo
Nx, along with the nxrocks plugin, is being used. Nx is a monorepo management tool and nxrocks adds Flutter functionality to Nx.
Custom scripts to help automate tasks are also created and used.

Design
The state of our application is being managed using the redux design pattern. Specifically, a modified variant where instead of making use of middleware, calls to the backend are taken care of inside the reducers.

Continuous Integration, Continuous Delivery
GitHub Actions as well as AWS Codepipeline are used in combination. GitHub Actions provides convenience to developers in that it is integrated with GitHub, so it is easy to see the results of the tests. An issue is that GitHub Actions provides no security when it comes to deploymen,t as anyone that can view the ReverseHand repository, will be able to download the APK produced by the workflow. Additionally, it is difficult to setup an integration testing environment for mobile applications on GitHub Actions. For these two reasons, AWS Codepipeline is also used mainly because of the security aspect.

AWS CodeBuild is used by AWS Codepipeline to build the APK and it is then stored in an S3 bucket on AWS which only the development team has access to. Additionally, AWS Codepipeline makes use of AWS CodeDeploy which can easily provide an environment to run the integration tests. It can be seen how the two pipelines complement each other.

9 Deployment Model

Our entire back end services are serverless so therefore they are all broken up into different nodes instead of under one root computer.

All configuration and serverless functions code are stored in a file storage system.