AN ABSTRACT OF THE MASTER'S PROJECT REPORT OF

Siri Chandana Gangam for the degree of Master of Science in Computer Science presented on December 7, 2022.

Title: Reshape, an Android Solution

Abstract approved: _____

Dr. Will Braynen

Reshape is a platform that enables you to rearrange the screens of an Android mobile application without having to make any changes to the underlying source code and so without having to re-upload the build to an app marketplace. This can help reduce development costs and speed up deployment. This is similar to A/B-testing solutions available today, but with a focus on reshaping screen flows on the fly, which is a feature currently absent from those solutions. While Reshape is a prototype built for Google's Android platform and so focuses on obviating the need to re-upload to Google Play, Google's app marketplace, it is a proof of concept that is not specific to Google Play and Android. The same idea can be applied to Apple's App Store and iOS, or in principle to any other distribution platform. Reshape includes a Software Development Kit (SDK), a web portal, and a backend that connects the two. The web portal shows a screen flowchart for the mobile application, which reflects the actual order in which the screens

appear in the mobile application. Using the web portal, the Product Owner can change the screen order without having to lean on the help of a software developer. The Android SDK has the logic to rearrange the screens of the Android mobile application based on the order received from the web portal via Reshape's backend.

Reshape, an Android Solution

by

Siri Chandana Gangam

A MASTER'S PROJECT REPORT

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 7, 2022
Commencement June 2023

Master of Science master's project report of Siri Chandana Gangam presented on December 7, 2022.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my master's project report will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my master's project report to any reader upon request.

Siri Chandana Gangam, Author

ACKNOWLEDGEMENTS

It was my parents' sacrifices that enabled me to remain committed to my education, and I wish to thank them for that. I would like to express my gratitude to my mentor, professor Dr. Will Braynen, for his guidance and challenging tasks along this journey which enabled me to comprehend the concepts. I would like to thank my committee members, Dr. Mike Bailey and Dr. Bella Bose, for their flexibility regarding scheduling the exam. The contributions of Bhavana Bandam Reddy and Yashoda in reviewing my code and helping me in maintaining the coding standards are highly appreciated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

Chapter 1: Introduction

1.1 Background

Mobile applications have gained enormous popularity since the invention of smartphones (Islam & Mazumder, 2010). The mobile application development process involves different stages, including strategy, analysis, and planning, as well as user interface design and user experience design, app development, testing, deployment, and support, as shown the Figure 1.1. Identifying the project's goals is the first step in developing an application. The next step is to plan the schedule, calculate the cost, refine the budget, and select the right team to work on the project. The third step is to design the app's appearance and determine how it can be developed to achieve the best user experience. The fourth step is to develop the app in accordance with the app requirements. The most critical tasks must be prioritized. After the code has been developed and tested, a unit test is performed to ensure that the components are working fine individually (Umar, 2019). After the team has developed a few features and is ready to upload the application to the app marketplace, it conducts system and acceptance testing; deployment is initiated if the requirements are met. To upload the application, the team must create a developer account for the app marketplace to which they want to upload. Compiled binaries should be generated first. Then an application should be created on the

app marketplace console, and details like title, description, high-resolution icon, product categorization, tags, and contact details will be requested in the console.

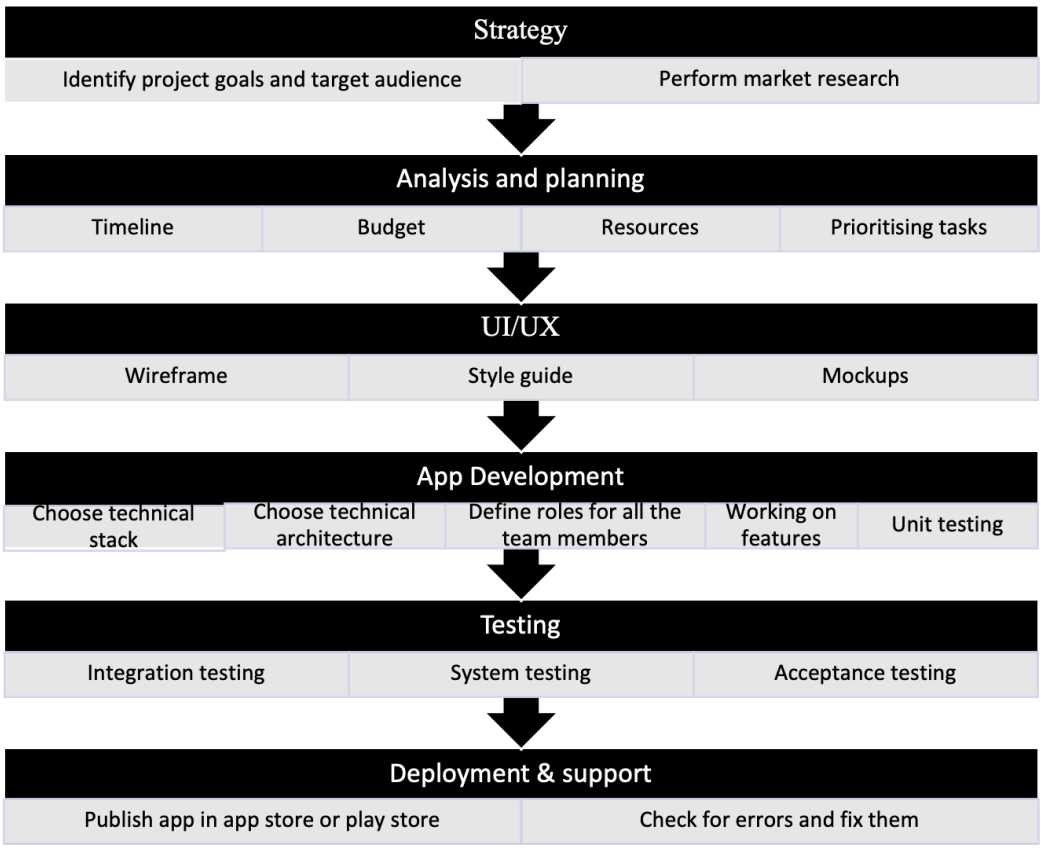


Figure 1.1: Mobile Application Development process

Then the generated compiled binaries must be uploaded. Details regarding the application's access rights and pricing, such as whether it is a free or paid application, must be set up before the stable version is published. Once the app is live, a team of developers will continually address the issues that real customers

(people who downloaded the app from the app marketplace) face and testing will be performed in production. Once the application has been successfully tested, the compiled binaries must be re-uploaded to the app marketplace. By repeating the process of testing and deploying a functionally working build, developers find the application bugs. This is the maintenance stage of mobile application development. Every time a build needs to be re-uploaded, these steps (writing code, testing code, and re-uploading the compiled binaries to the app marketplace) are repeated.

1.2 Problem statement

There are two ways to change the order of screens of the mobile application. One way is to write the code, test the code, generate the compiled binaries and re-upload the compiled binaries. The following code example illustrates how to navigate from one screen to another in a mobile application written in Kotlin:

```
val i = Intent (this, NextActivity::class.java)
startActivity(intent)
```

Activities provide a user interface through which users may interact with an application. In this case, NextActivity refers to the activity of the next screen that contains code for how the screen should look (Sehgal, 2020). With this, the Android operating system starts activity, which is passed in `startActivity()`. If we want to land on a different screen, the activity name must be updated in the code. The rest of the mobile application development process follows, as outlined in section 1.1 (Test and add Android app bundles / APKs to production to make

the application available to everyone on the Google Play Store (C.G.,Thomas and Devi,A.Jayanthila, 2021)). It would be helpful if we could avoid redeploying the compiled binaries in those cases where we only need to reorder screens since the deployment of the executable involves interaction and coordination between various teams within the organization. This matters because there should be an efficient recovery process in the event of an error; otherwise, the application, which is the Play Store, will be impaired. Additionally, deploying compiled binaries is costly. New versions of the app in the Android Play Store take a considerable amount of time to reflect the new screen order if we re-upload the compiled binaries, as the application has to undergo a review process before updating it.

Another way to reorder screens is using A/B testing frameworks such as Optimisely and Leanplum. In this case, you maintain a feature flag on the website of an A/B testing framework. This feature flag is used in the mobile application's code in order to determine what should be the screen flow. This way of reordering screens is better than the first way of reordering screens, as we do not have to redeploy the compiled binaries every time we want to change the screen order. However, it is not an optimum way to reorder screens as we have to define all permutations you want in the mobile application code.

1.3 Proposed Solution

The proposed solution allows us to rearrange the screens without shipping the newly compiled binaries to the Android Play Store (or simply 'Play Store'). An

Android SDK and a web portal are developed to solve this problem. The web portal has a list of screens available in the Android mobile application. The Product Owner can reorder those screens according to the requirements documented in the requirements gathering phase. The developers can use Android SDK and change the order of screens of the Android mobile application to align with the Product Owner's changes to the screen order in the web portal. The proposed solution reduces the developers' effort to change the code and saves the developers' time, so that developers can invest their time in developing something more important than just modifying the order in which screens are presented. Apart from this, the required screen order will be reflected in the Play Store very quickly, as we do not have to redeploy the compiled binaries. We must update the screen order in the web portal. The proposed solution is more effective when used in conjunction with A/B testing. A/B testing is a technique to gather data that can help Product Owners choose between different versions of the application in an empirical and data-driven way. The A fork of the A/B test could, for example, be one screen order while the B fork could be a different screen order.

1.4 Use cases of the proposed solution

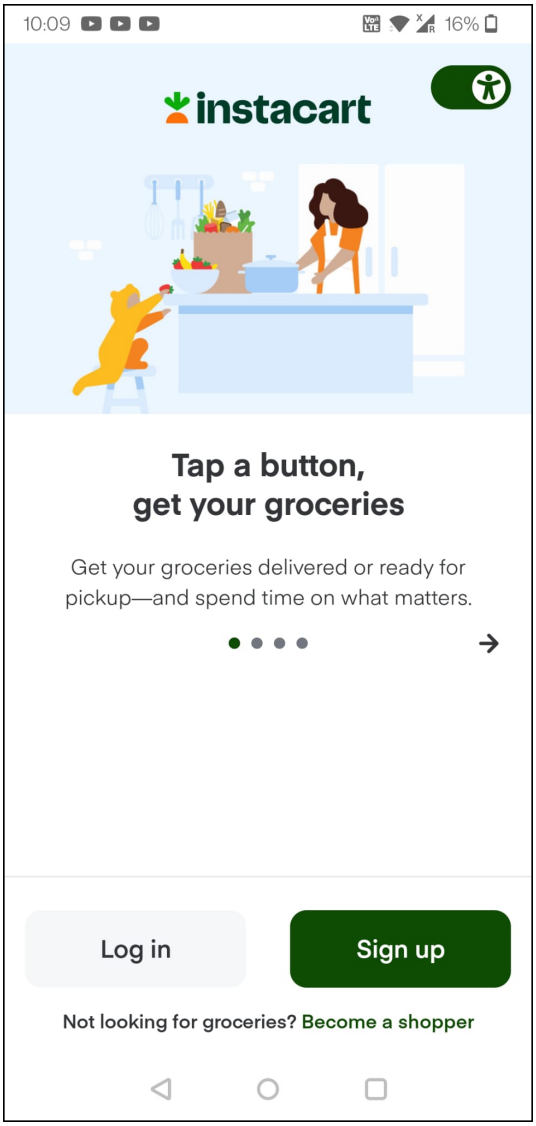
In this section, I gave two sample use cases for Reshape: (1) a retail app and (2) an app to book flight tickets.

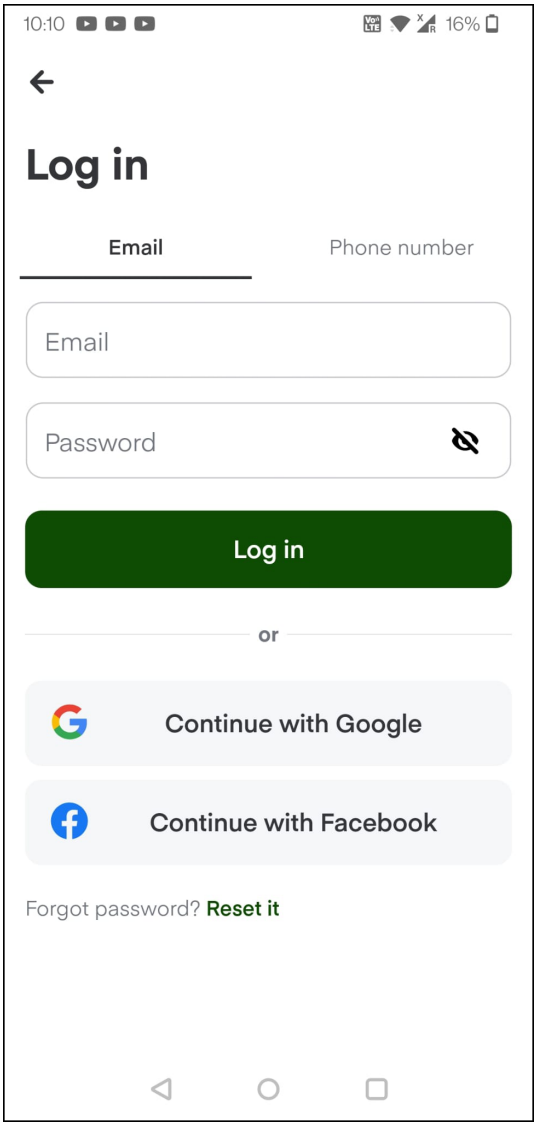
1.4.1 A retail app

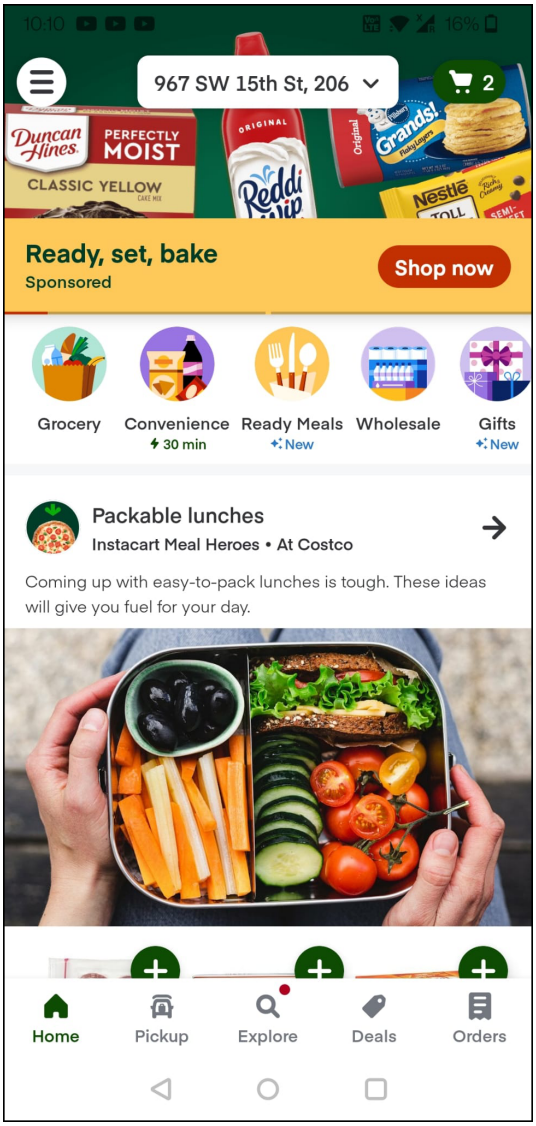
For example,the userflow of Instacart Android mobile application is shown in the figure 1.2:

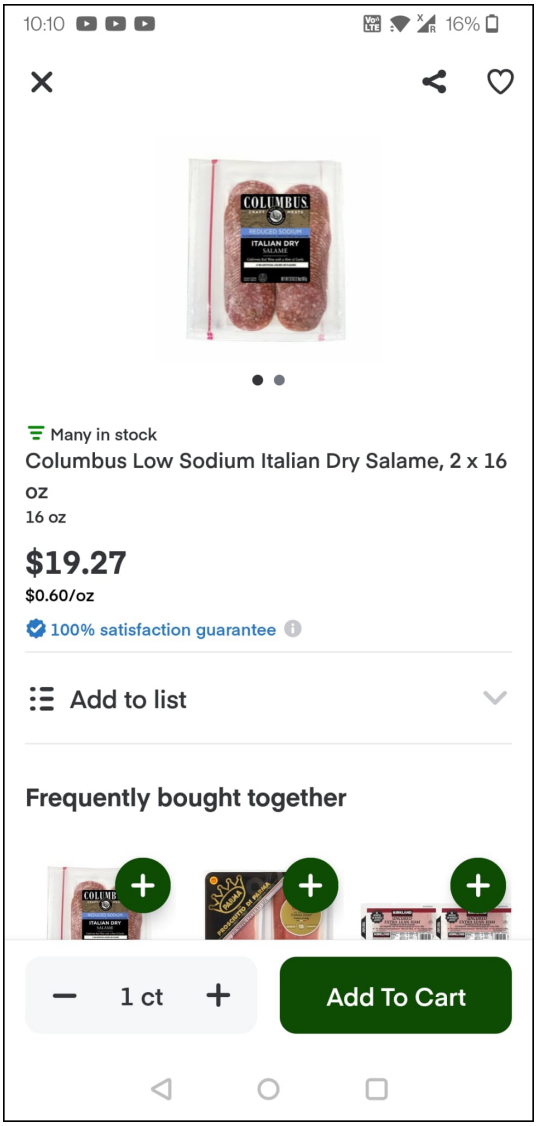
1. When the app is installed, the user will be prompted to log in or register.
2. If the user taps on "Login", it will take them to the login flow.
3. After login, the user will land on the dashboard.
4. On tapping on a specific product, the "product description page" is presented.
5. Users can add the product they like to their shopping cart.
6. Once the user taps on "Add to cart", the contents of the shopping cart are presented.
7. Once the user taps on "Go to checkout", the "Order Confirmed" page is presented.

If this screen flow is analyzed and the results indicate that many users are landing on the login screen but not completing the journey and the business team believes that this may be because users are being asked to log in before they can see the product description, the business team might decide that users should be asked to log in after clicking "Place Order" on the cart page. To make these changes, in the traditional process, the Product Owner would communicate about


10:09
16%
instacart
Tap a button,
get your groceries
Get your groceries delivered or ready for pickup—and spend time on what matters.
Log in
Sign up
Not looking for groceries? Become a shopper


10:10
16%
Log in
Email
Phone number
Email
Password
Log in
or
Continue with Google
Continue with Facebook
Forgot password? Reset it


10:10
16%
967 SW 15th St, 206
2
Duncan Hines
PERFECTLY MOIST
CLASSIC YELLOW
Reddi Wip
Grands!
Nestlé
Ready, set, bake
Sponsored
Shop now
Grocery
Convenience
30 min
Ready Meals
New
Wholesale
Gifts
New
Packable lunches
Instacart Meal Heroes • At Costco
Coming up with easy-to-pack lunches is tough. These ideas will give you fuel for your day.
Home
Pickup
Explore
Deals
Orders


10:10
16%
COLUMBUS
ITALIAN DRY
Many in stock
Columbus Low Sodium Italian Dry Salame, 2 x 16 oz
16 oz
$19.27
$0.60/oz
100% satisfaction guarantee
Add to list
Frequently bought together
1 ct
Add To Cart

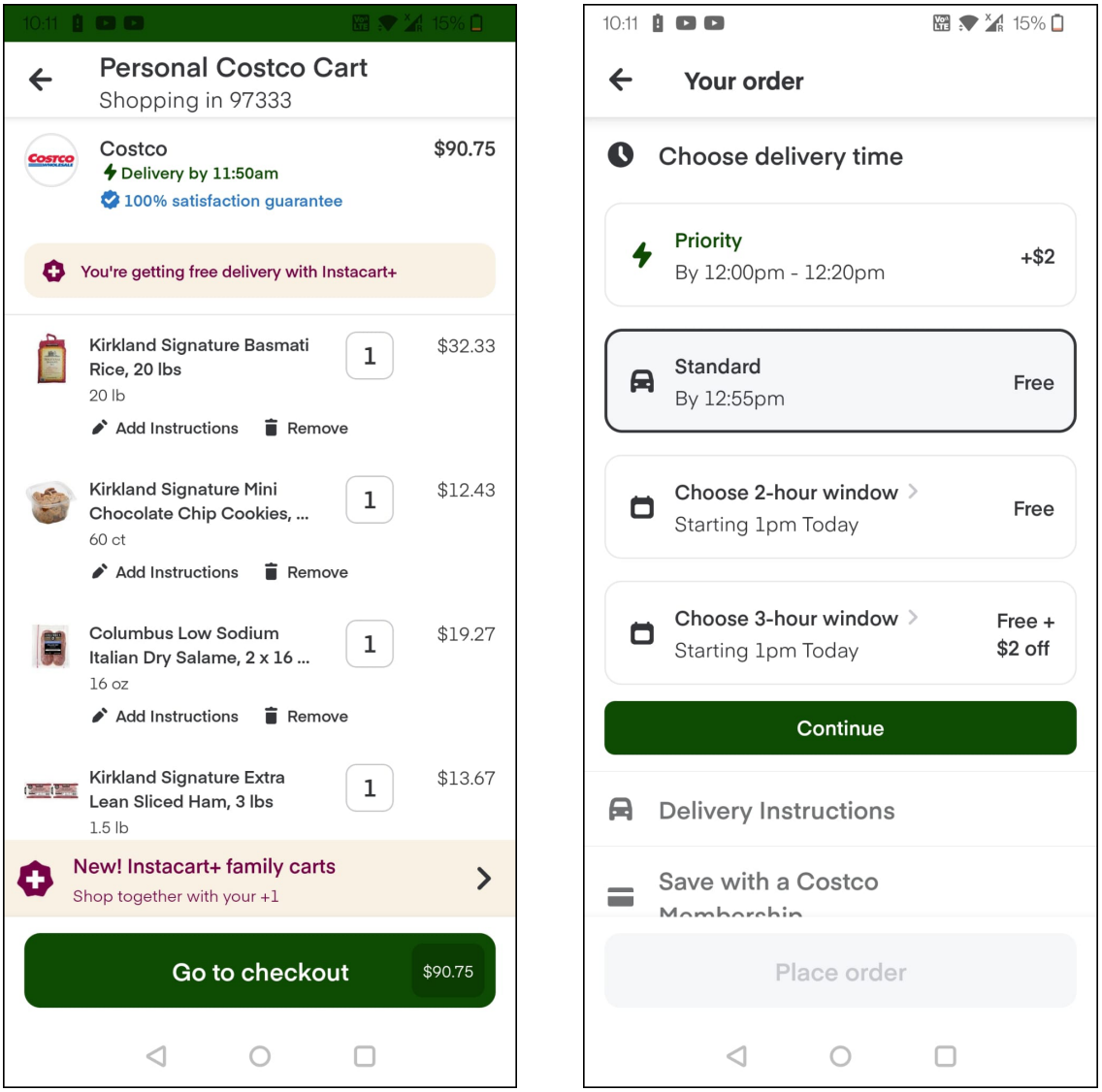


Figure 1.2: Order of screens in Instacart Android app

the changes in the screen order with the developers. Then the developers would change the code, test, and deploy. However, the Reshape platform allows Product Owners to rearrange screens by changing the order in the web portal without the help of developers.

1.4.2 Adding a deals page of flights at the correct point of user flow

. In order to increase the likelihood of users noticing the deals page, it should be placed in the right location. The company may experience lower profits if those are not added to the right place. A/B testing can be conducted by moving the deals page to a location that you believe will attract customers. In order to ensure that they are noticed, deals are usually posted on the "Home page." Reshape automates this process of rearranging the screens.

Chapter 2: Architecture

2.1 Tech stack

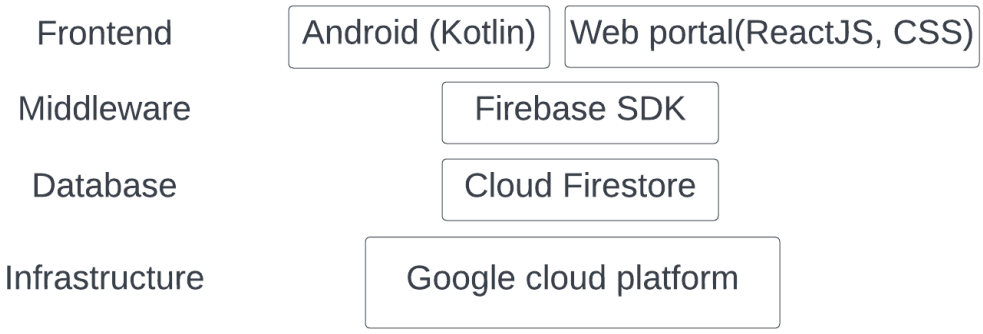


Figure 2.1: Tech stack

2.1.1 Kotlin

The programming language used in this project to develop the mobile application and SDK was Kotlin, which was created by JetBrains. I picked Kotlin for this project for three reasons. The first reason is that application deployment with Kotlin is faster and lighter when compared to Java and prevents applications from growing in size, which results in fewer bugs. The second reason is that Kotlin

distinguishes between references that can hold null and those that cannot hold null. Third reason is that it throws a compilation error if we try to hold null for references that cannot hold null. Several features in Kotlin, including smart casting and type inference, make coding easy.

2.1.2 ReactJS

ReactJS was used in this project to develop the web portal that is required to display the screens present in the mobile application. I picked ReactJS for this project for three reasons. The first reason is that, it is a JavaScript library that is used to build interactive UI components with frequently changing data. In addition, it contributes to the development of rich user interfaces. Second reason is that it offers fast rendering (Dani, Tomar, Srivastava, & Bindal, 2021). Third reason is that components are reusable, and also custom components can be written. Because of the modular structure, it is easier to maintain.

2.1.3 Firebase

Firebase was used in this project to develop the backend. It was designed to assist in building, hosting, and promoting apps by monitoring them and boosting user engagement. It supports Android, iOS, and web applications. Cloud Firestore is used in this project to store the data of screens of the Android mobile application. *Cloud Firestore* is also a database in which you can store data easily, and the

application data will stay synchronized with the cloud but is more structured. It stores all the data in objects known as documents. Hence it is known as a document-model database in which each document has a pair of keys and values. Any kind of data can be stored in this database. It costs relatively less than a Realtime database as it depends on how often the data is stored or retrieved.

2.2 System Architecture

In this system, data is stored in Firebase Firestore. The web portal is developed using ReactJS.

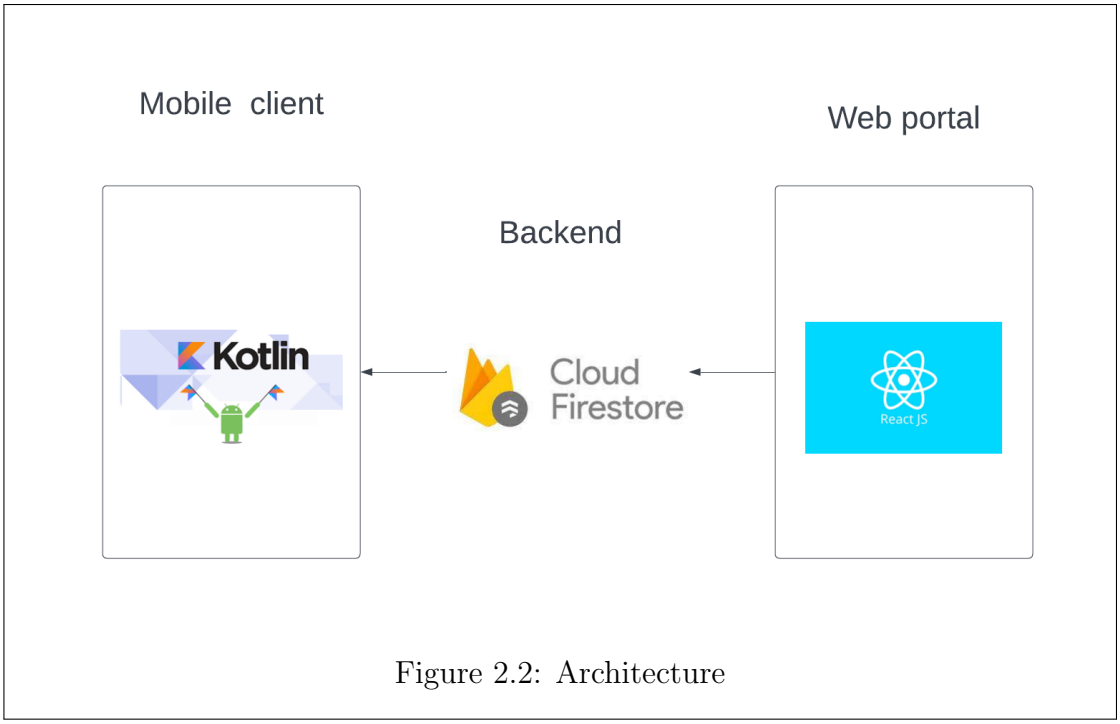


Figure 2.2: Architecture

ReactJS has a component architecture. The user interface is simply a view that

consists of several React components beneath it that are simple structural units like labels, buttons, and text input labels. A React component may hold state, which is the information that the application needs to track in order to function. In addition, the state of an application changes constantly as a result of user actions that determine what displays on the UI.

2.3 Use case diagram

In this project, the web portal, which was developed in ReactJS, has information related to screens that are present in the mobile application. That information was stored in the Cloud Firestore data. Since Firebase Firestore was used in this project, I didn't need to handle server-side code. That data that is stored in the Cloud Firestore is retrieved by the Android mobile application and when this screen order is sent to the SDK (Software Development Kit) which can be integrated with any mobile application, and SDK will change the order of the mobile application according to the order set in the web portal.

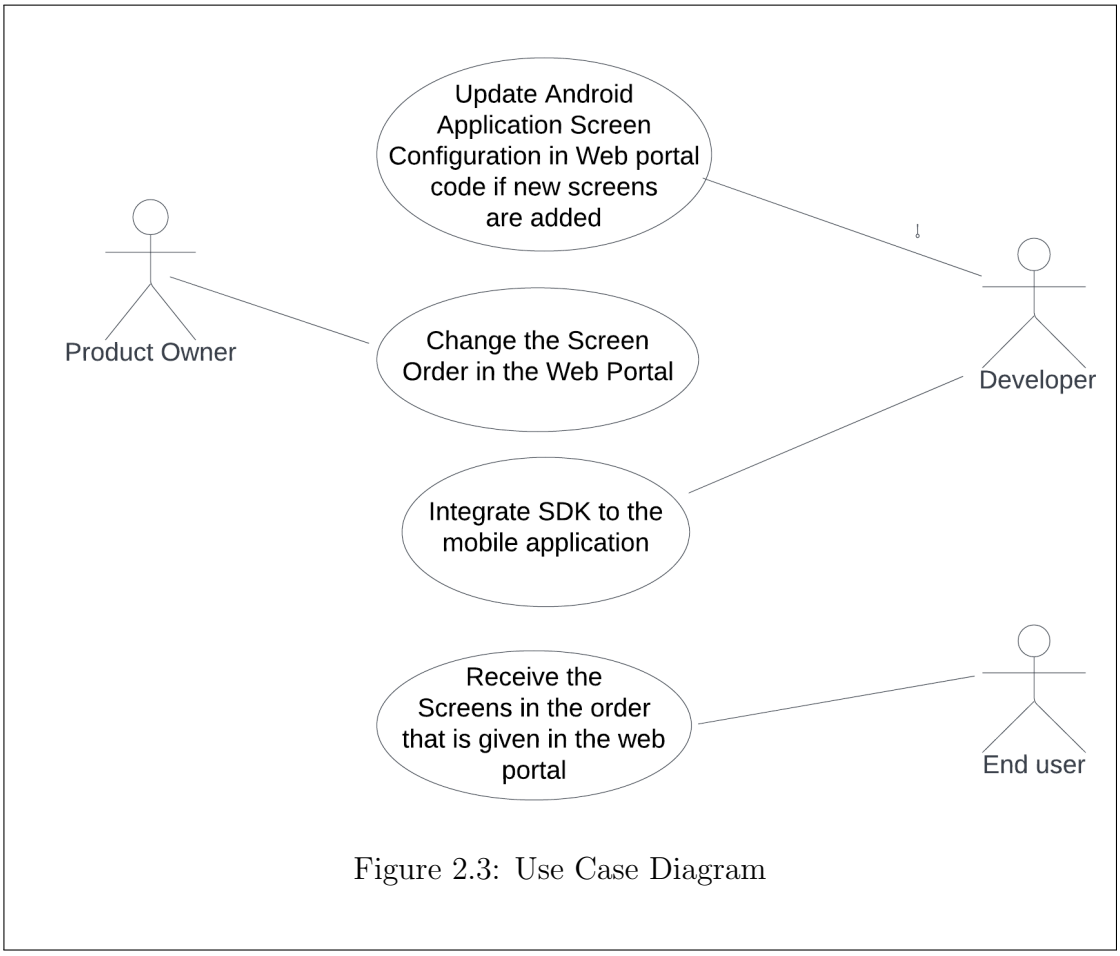


Figure 2.3: Use Case Diagram

2.4 Flowchart

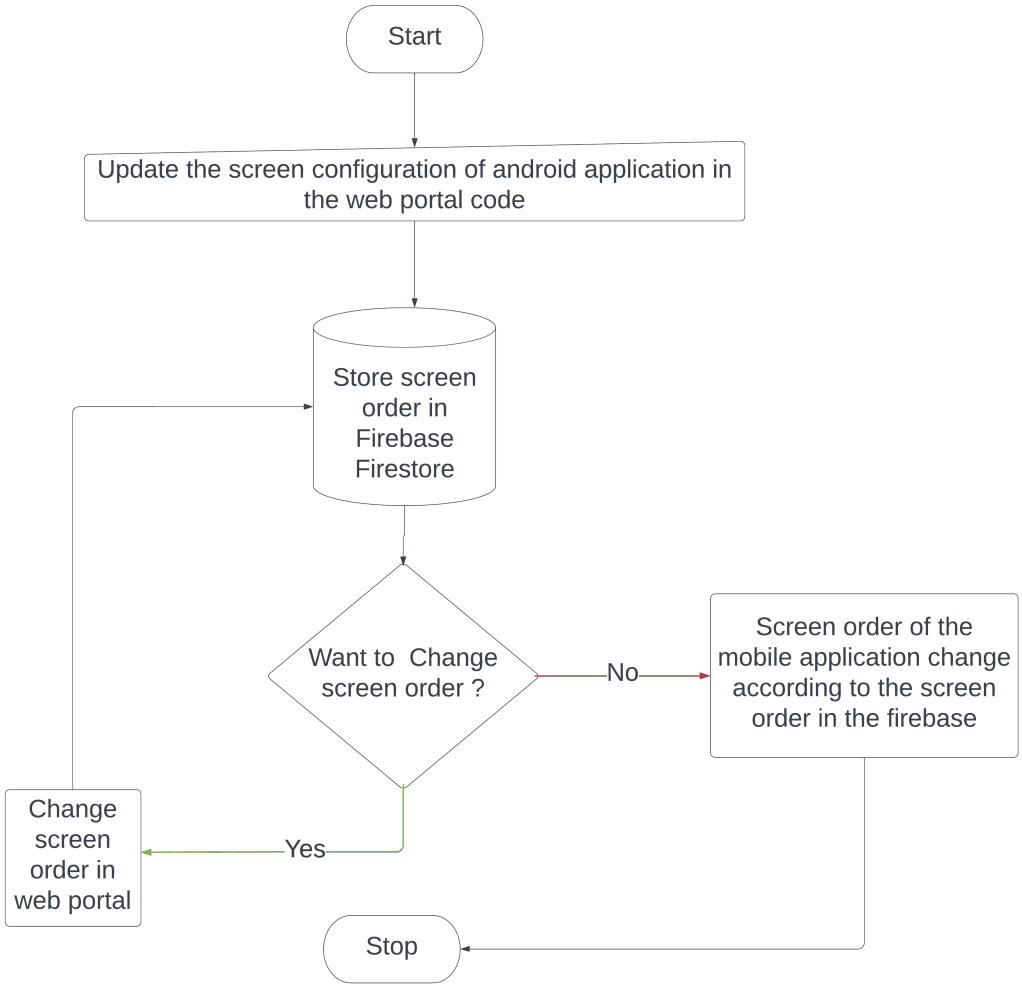


Figure 2.4: Flowchart

Chapter 3: Implementation

Reshape is a platform that enables the Product Owner to rearrange screens of the Android application without actually redeploying the new code in the app marketplace. The following steps were performed to make this happen

3.1 Basic setup

3.1.1 Firebase setup

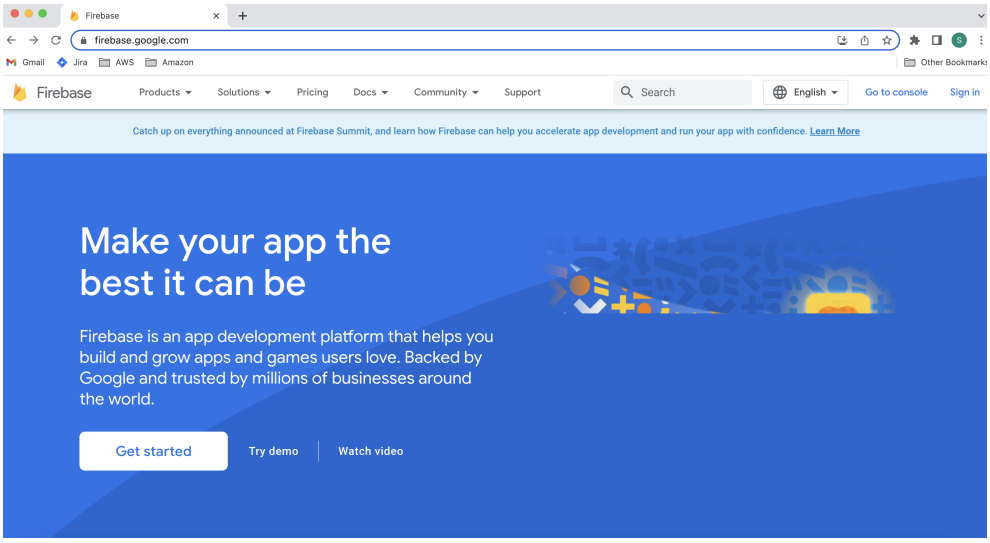


Figure 3.1: Firebase website

I had to set up Firebase as I did not use any server for handling the backend. I used Firestore for storing data.

3.1.1.1 Steps to create a database:

- Go to the official website of Firebase and click on "Go to console" as shown in the figure 3.1.
- To add a new project, click on "Add project." If the project is already added, we can see the list of projects, as shown in the figure 3.2.

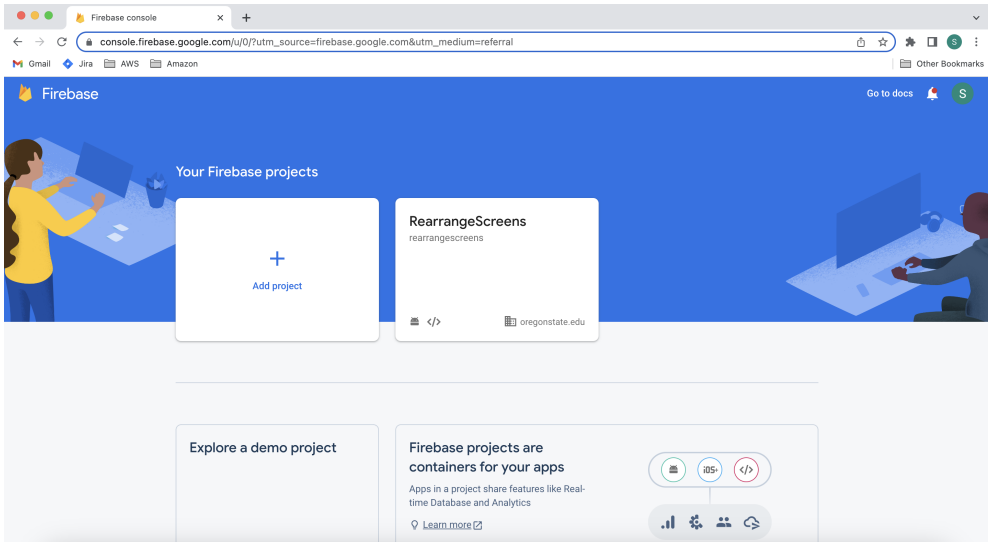


Figure 3.2: Choose Project

- Creating a new project:

On click of “Add project,” you will be taken to a create a project screen where you will be asked to enter the name for the project, check “I confirm,” and continue as shown in the figure 3.3.

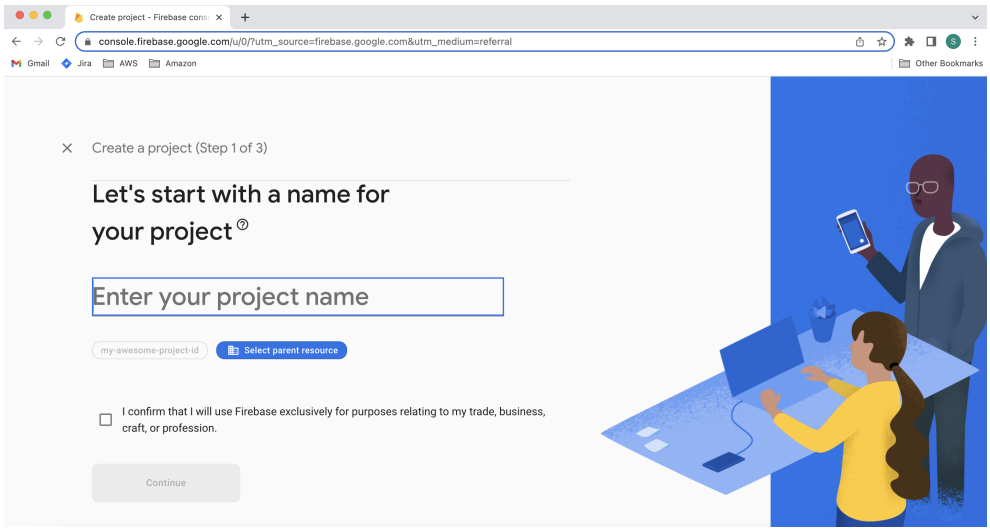


Figure 3.3: Creating a new project

In the second step of setting up the firebase project, you will be asked if you need Google Analytics for your project. You can either enable or disable Google Analytics and click ”Continue” as shown in the figure 3.4

In the third step, you will be asked to configure Google Analytics. Then you should accept the terms and conditions and click on “Create project” as shown in the figure 3.5

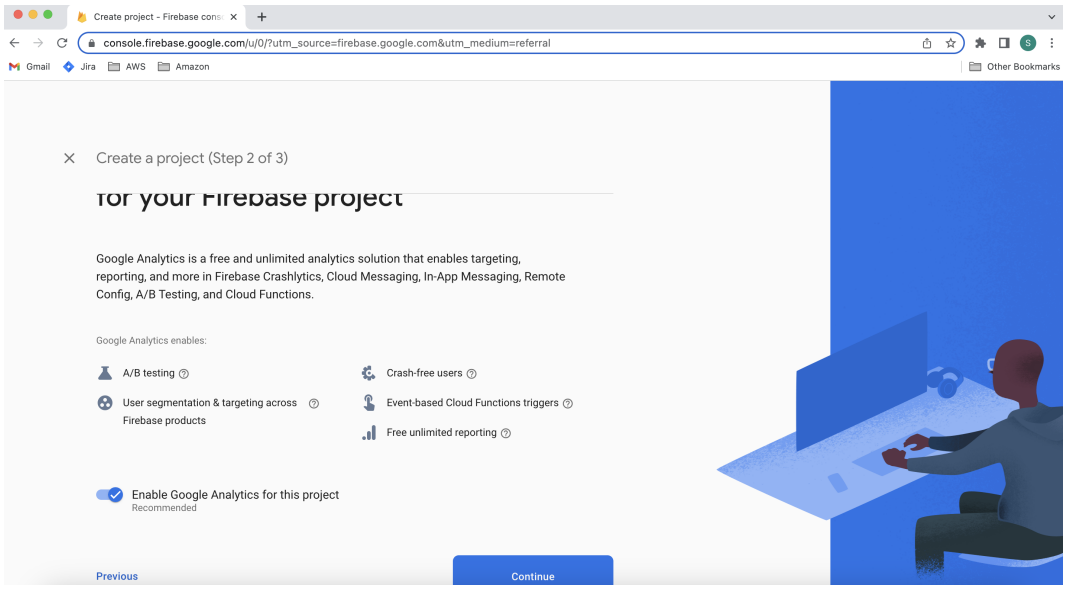


Figure 3.4: Enable or disable Google Analytics

3.1.1.2 Steps to setup project in Firebase Cloud Firestore

Once the project is created, you will be taken to the project console as shown in the figure 3.6:

Once the project is created, Firebase should be added to the application. An application can be iOS, Android, or Web. I used the same database for both the Web portal and the Android application. So, I had to add Firebase to both Android and Web applications.

To add the Firebase to the Android application, click on the Android

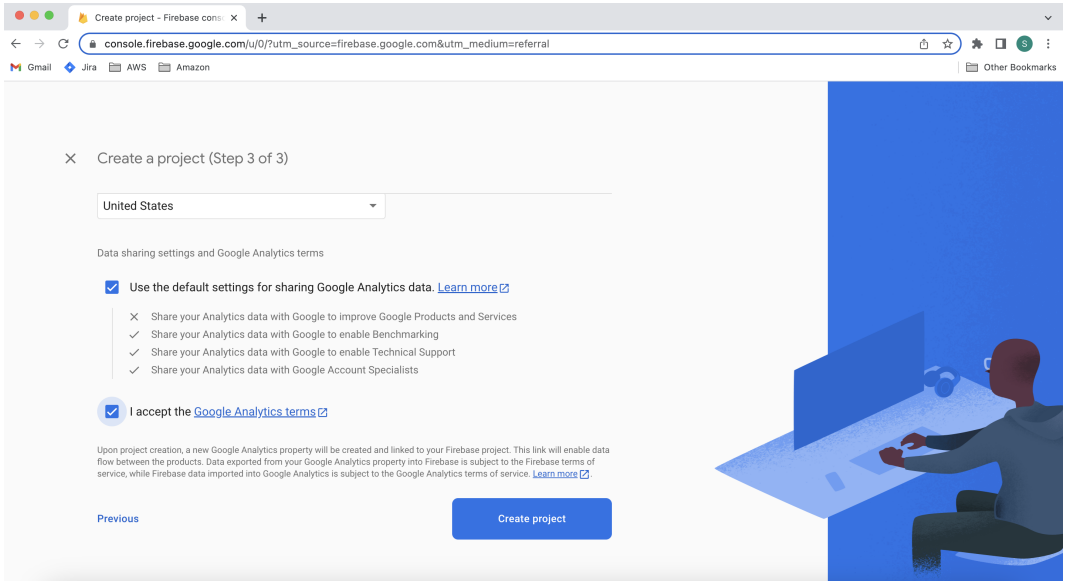


Figure 3.5: Configure Google Analytics

icon. There will be four steps to add an Android application.

In the first step, you will have to register the application, It will ask for a package name to be used in the Android app, an optional application nickname, and a debug signing certificate that is also optional. Then you will need to click "Register app" as shown in the figure 3.7

The second step is to download and add google-services.json and add it to the app-level root directory as shown in the figure 3.8.

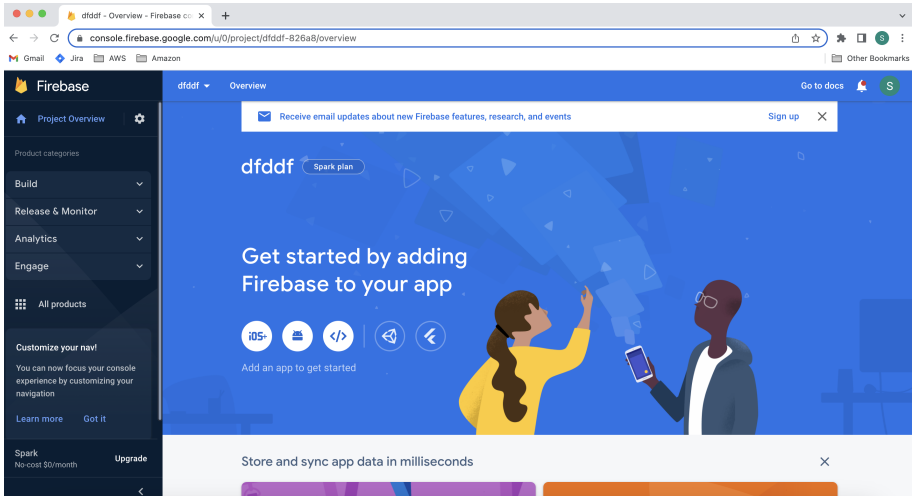


Figure 3.6: Project console

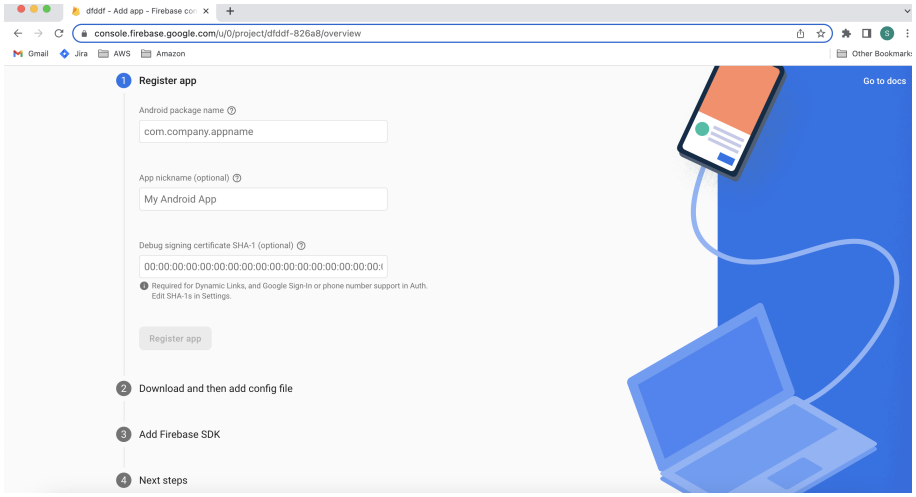


Figure 3.7: Register application

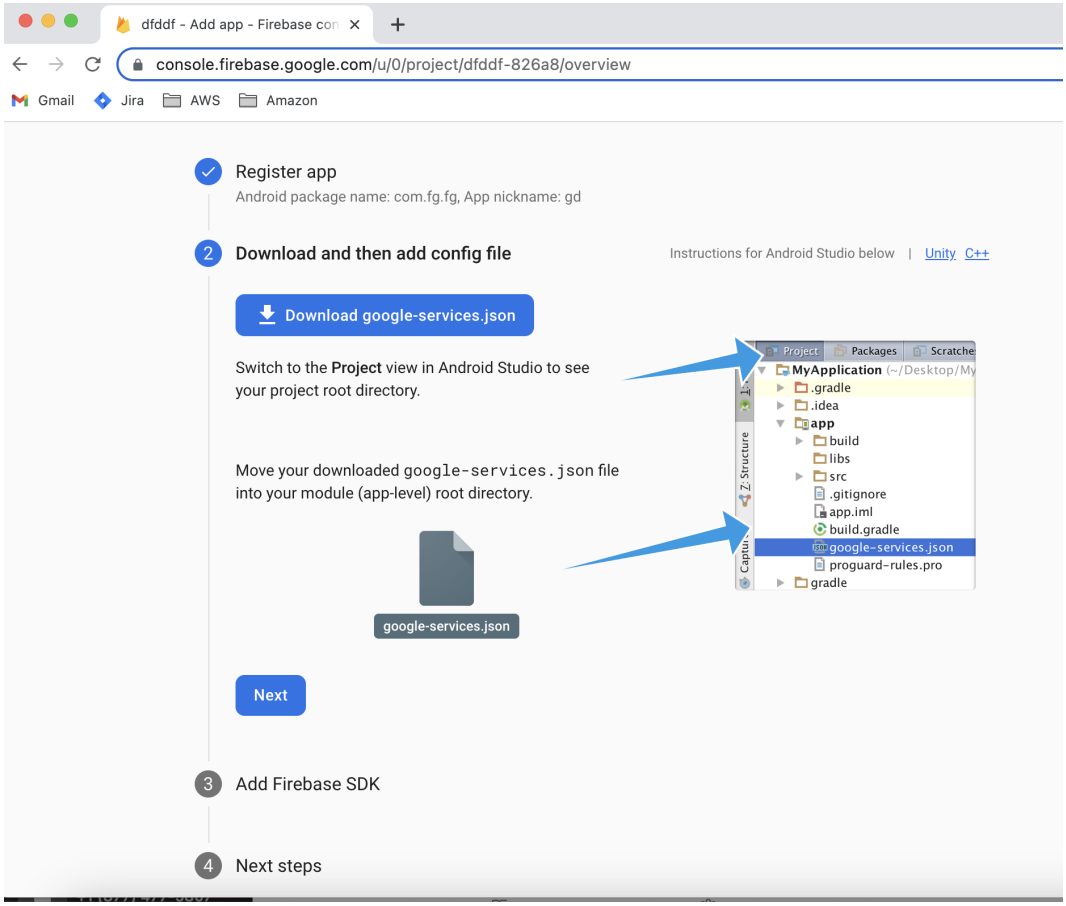


Figure 3.8: Add google-services.json

The third step is to add the Firebase SDK In mobile application code, build.gradle file which is in the root level should be updated as follows (plugin should be added as a build script dependency).

```
buildscript {
```

```
    ext.kotlin_version = '1.6.21'
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.google.gms:google-services:4.3.10'
    }
}

allprojects {
    repositories {
        google()
        jcenter()

    }
}
```

At the module level, build.gradle file, I added google-services.json plugin and the SDKs that are required.

```
apply plugin: 'com.google.gms.google-services'
dependencies {
    implementation 'com.google.firebase:firebase-analytics'
```

```
    implementation 'com.google.firebase:firebase-auth-ktx'
    implementation 'com.google.firebase:firebase-firestore-ktx'
    implementation 'com.google.firebase:firebase-database-ktx:19.7.0'
    implementation 'com.google.firebase:firebase-firestore:21.4.0'
}

repositories {
    mavenCentral()
}
```

To add Firebase to a web application, click on the web icon, and you will land on the screen 3.9.

There are two steps that needs to be followed to add Firebase to a web application:

Initially, I provided the name for the web application. I did not want to host the web application, so I did not check the Firebase Hosting option.

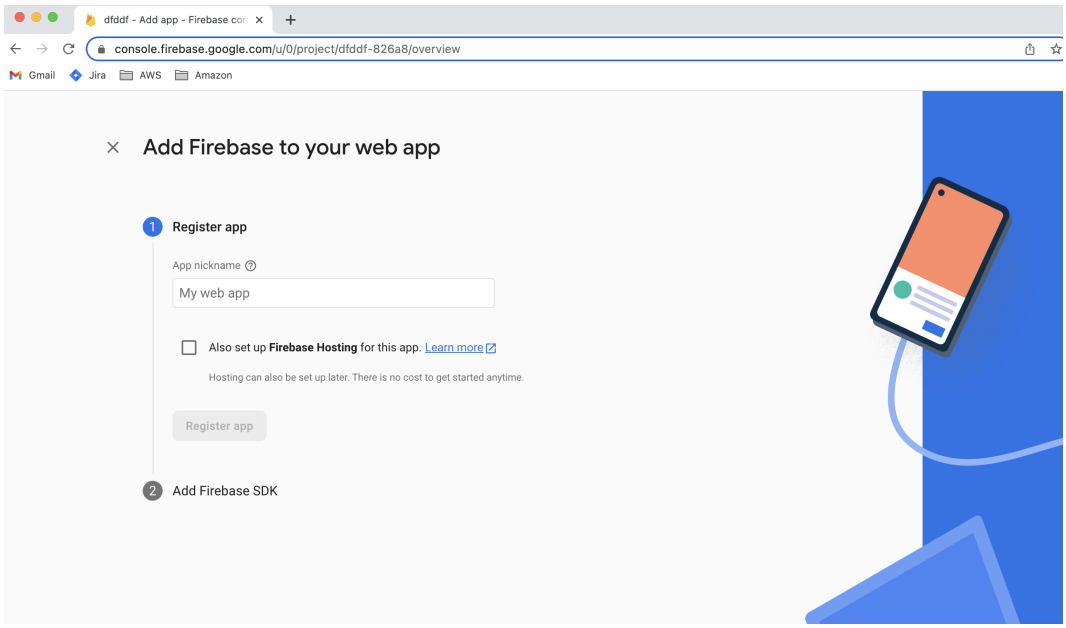


Figure 3.9: Fill details of web app

The second step is to add Firebase SDK to the web app code, I have added the following code.

```
import { initializeApp } from "firebase/app";
const firebaseConfig = {
    apiKey: "AIzaSyBRGZT2zUANDb510Cdn7RlGwORLRSc3VWI",
    authDomain: "rearrangescreens.firebaseapp.com",
    databaseURL:"https://rearrangescreens-default-rtdb.firebaseio.com",
    projectId: "rearrangescreens",
```

```
    storageBucket: "rearrangescreens.appspot.com",
    messagingSenderId: "616944017165",
    appId: "1:616944017165:web:6cc5e70a69b282bc3f9a44",
    measurementId: "G-BS1HJ1C09Q"
};
export const app = initializeApp(firebaseConfig);
```

3.1.2 ReactJS

The following steps were followed in order to set up a ReactJS project as shown in the figure 3.10.

```
siri@10-248-8-74 documentationtest % npx create-react-app rearrangescreens

Creating a new React app in /Users/siri/Desktop/documentationtest/rearrangescreens.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1395 packages in 24s

212 packages are looking for funding
  run `npm fund` for details
```

Figure 3.10: Create ReactJS project

```
Success! Created rearrangescreens at /Users/siri/Desktop/documentationtest/rearrangescreens
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd rearrangescreens
  npm start

Happy hacking!
```

Figure 3.11: ReactJS project created

With this, the basic setup for the ReactJS project is done. The following files are generated as shown in the figure 3.12.

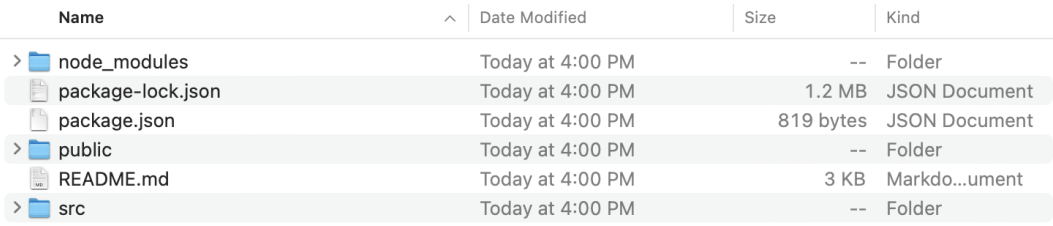


Figure 3.12: Files generated

To run the project, following steps are followed:

```
cd rearrangescreens
npm start
```

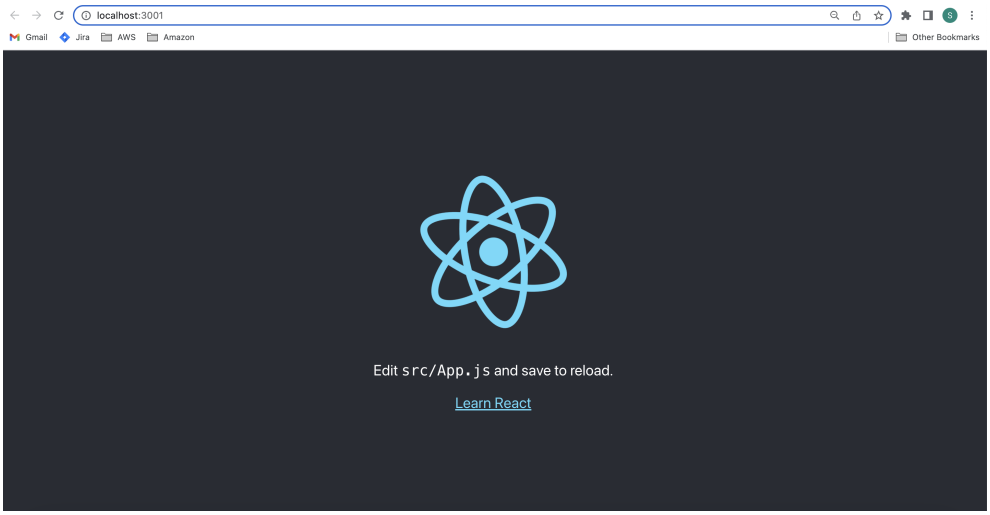


Figure 3.13: Website

3.1.3 Kotlin setup

Android Studio should be installed prior to the steps which are listed below. After opening Android Studio, click on “New project.” You will get to choose the target device and the template as shown in the figure 3.14:

After clicking on “Next,” you will see a screen that prompts you to choose a name for the project, choose a package name for the project, change the project’s save location, change the language in which you want to build the project (Java or Kotlin), choose a minimum SDK, and click ”Finish.” Once you have done these things, setup is complete.

Dependencies are added.

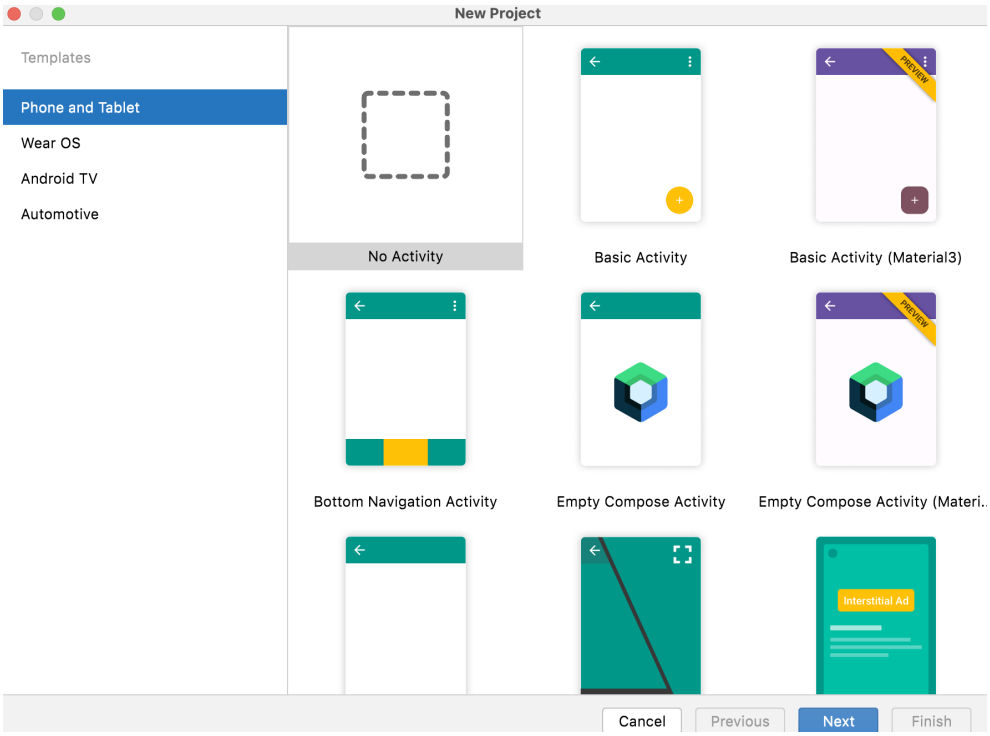


Figure 3.14: Android studio - create project

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-
    jdk7:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
}
```

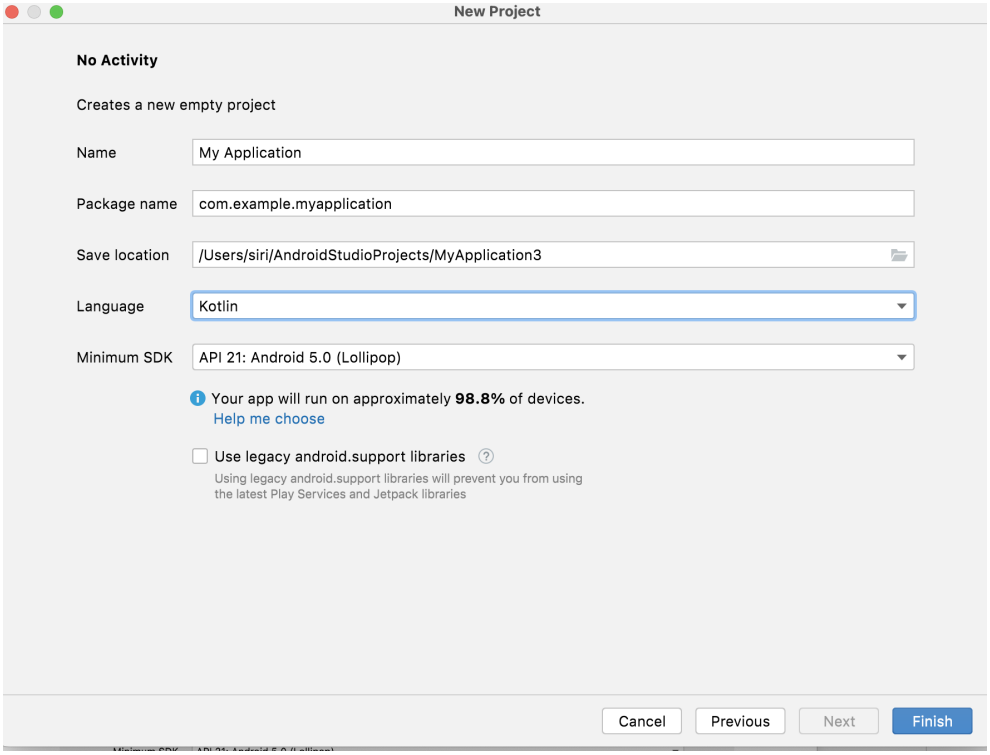


Figure 3.15: Fill details - Android Studio

3.2 Methodology

The screen configuration of the mobile application was written in the web portal code.

```
const data = [
    {
        "current": 1,
        prev: null,
```

```
    "one": 2,
    "two": 9,
    "path": "images/screen1.png",
    height: 484,
    width: 195,
    position: {
        x: 0, y: 180
    }
}, {
    "current": 2,
    prev: { current: 1, button: "one" },
    "one": 3,
    "two": 4,
    "path": "images/screen2.png",
    height: 296,
    width: 161,
    position: {
        x: 242, y: 70
    }
}, {
    "current": 3,
    prev: { current: 2, button: "one" },
    "one": 5,
```

```
    "two": -1,
    "path": "images/screen3.png",
    height: 193,
    width: 166,
    position: {
        x: 483, y: 15
    }
}, {
    "current": 4,
    prev: { current: 2, button: "two" },
    "one": 6,
    "two": -1,
    "path": "images/screen4.png",
    height: 169,
    width: 172,
    position: {
        x: 483, y: 281
    }
}, {
    "current": 5,
    prev: { current: 3, button: "one" },
    "one": 7,
    "two": -1,
```

```
    "path": "images/screen5.png",
    height: 241,
    width: 175,
    position: {
        x: 750, y: 15
    }
}, {
    "current": 6,
    prev: { current: 4, button: "one" },
    "one": 8,
    "two": -1,
    "path": "images/screen6.png",
    height: 205,
    width: 170,
    position: {
        x: 750, y: 281
    }
}, {
    "current": 7,
    prev: { current: 5, button: "one" },
    "one": 0,
    "two": -1,
    "path": "images/screen7.png",
```

```
    height: 184,
    width: 158,
    position: {
        x: 1033, y: 15
    }
}, {
    "current": 8,
    prev: { current: 6, button: "one" },
    "one": 0,
    "two": -1,
    "path": "images/screen8.png",
    height: 165,
    width: 163,
    position: {
        x: 1033, y: 281
    }
}, {
    "current": 9,
    prev: { current: 1, button: "two" },
    "one": 10,
    "two": -1,
    "path": "images/screen9.png",
    height: 233,
```

```
        width: 157,
        position: {
            x: 242, y: 522
        }
    }, {
        "current": 10,
        prev: { current: 9, button: "one" },
        "one": 1,
        "two": -1,
        "path": "images/screen10.png",
        height: 234,
        width: 152,
        position: {
            x: 483, y: 516
        }
    }
]
```

Here, `current` is the current screen number, and `prev` identifies whether there is a screen before the current one. A null value is set for `prev` when there is no screen; otherwise, the following information should be added:

```
prev:
    {
        current : 9,
```

```
    button : "one"
}
```

`current` refers to the previous screen number, and `button` refers to the button on which it reached this screen. `one` refers to the screen number on which it will land upon clicking the first button. `two` refers to the screen number on which it will land upon clicking the second button of the screen. `path` has the path where a particular image to that screen is located. Here, `height`, `width` are the dimensions of the image which represents the screen. `position` says where exactly that image should be located in the screen.

This data is stored in the Cloud Firestore database as shown in the figure 3.16.

This data is retrieved from the mobile application and rearranging happens according to the screen order received from the database. RearrangeScreensModule SDK has a function which will be used to decide what the next screen should be.

```
pageChanger (activity: Activity, screenNum: Int = -5,
state: Int = 0)
```

Name of the activity, screen number, and state will be passed to the function pageChanger, where state will be 0 if the first button in the screen is clicked and 1 if the second button is clicked. Traditionally, we move from one screen to another screen using

```
val i = Intent (this, NextActivity::class.java)
startActivity(i)
```

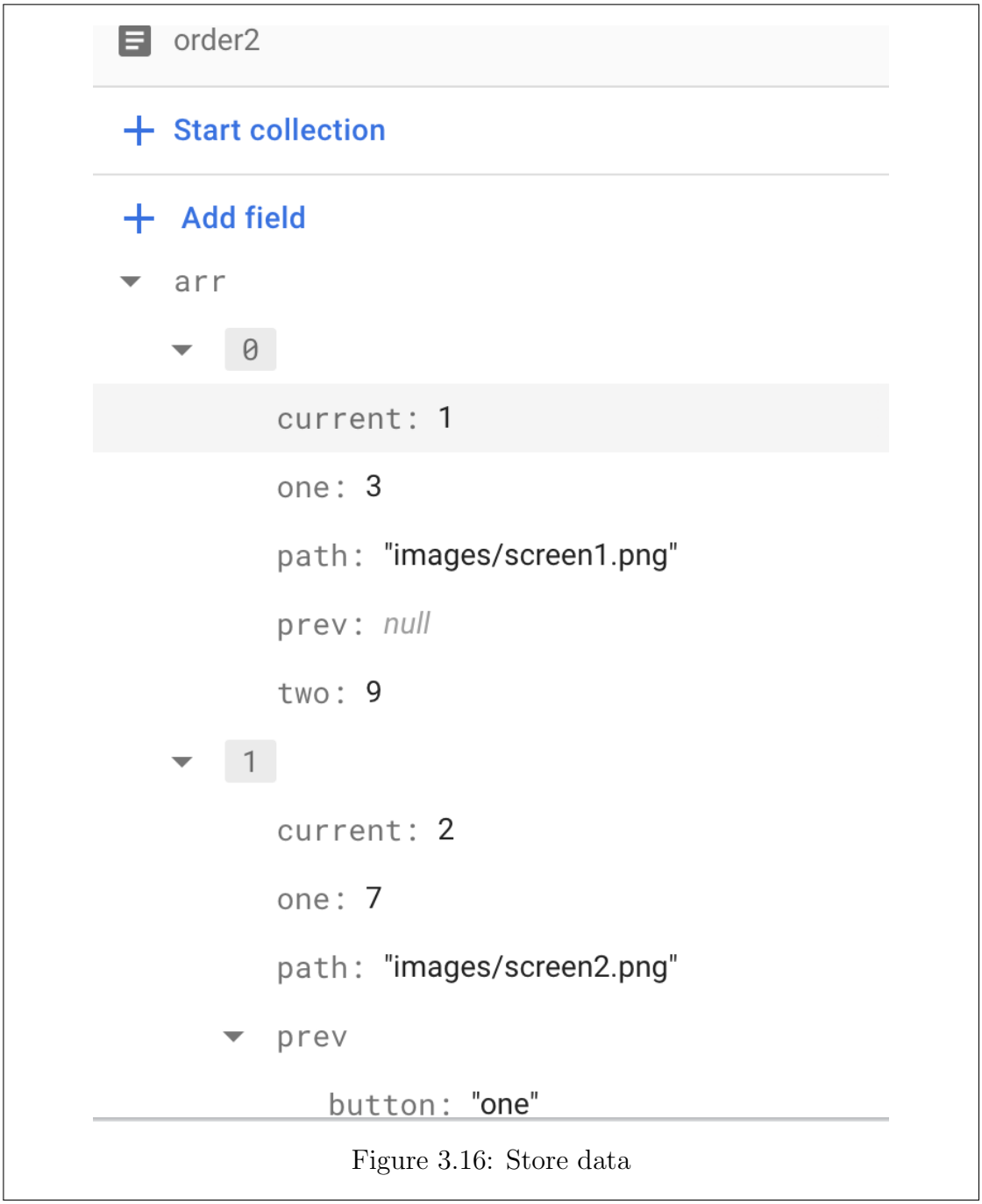


Figure 3.16: Store data

We replace the above code with

```
Global2.pageChanger(this, 1)
```

This says that the first button is clicked so the third argument (0 is taken by default) is not passed and ”1” is the screen Number that must match the configuration that is written in the web portal.

Chapter 4: Results

The web portal developed to reorder the screens is shown in Figure 4.1.

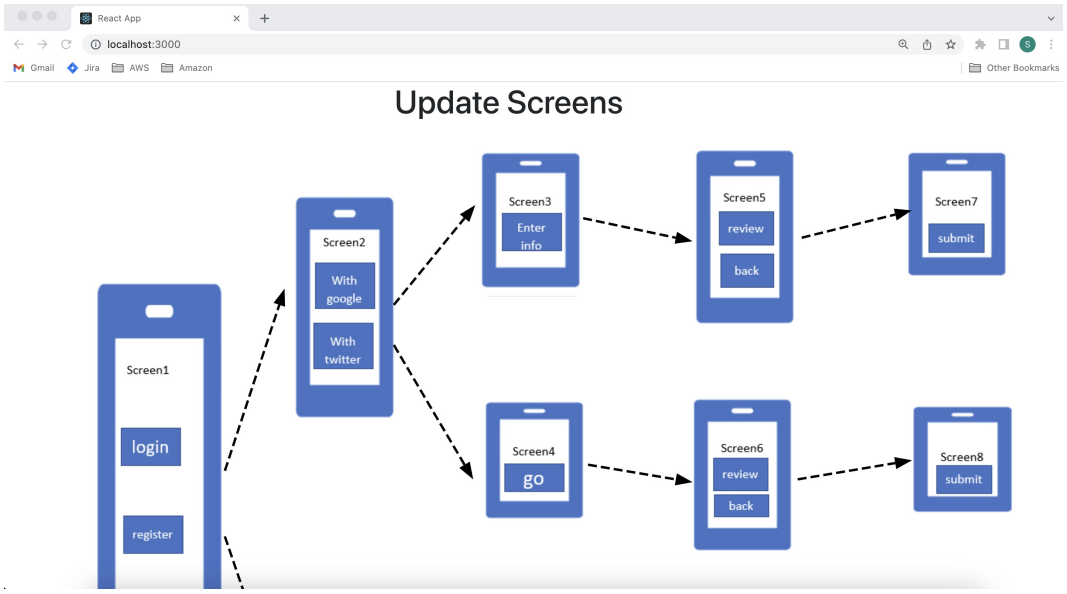


Figure 4.1: Web portal screenshot

Screens are arranged to represent the screen flow. If the Product Owner chooses to rearrange screens 9 and 10, he/she needs to write the number assigned to a particular screen that he/she wants to rearrange. in the text boxes as shown in Figure 4.2.

Once the "Replace" button is pressed, the screen order will be changed. Once the new order is saved in the Firestore, a popup will come as shown in Figure 4.3.

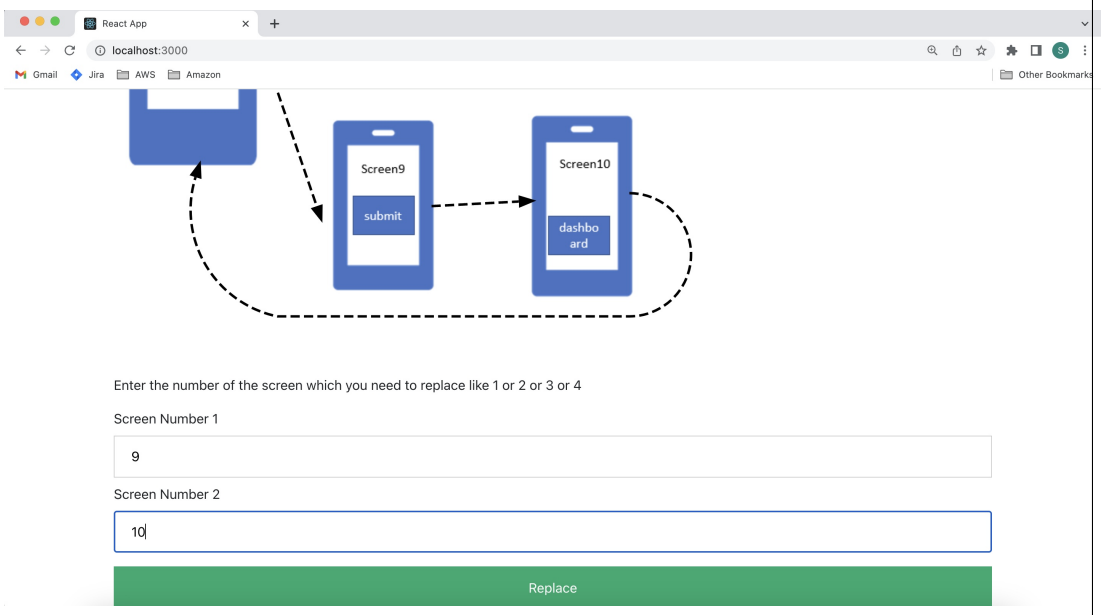


Figure 4.2: Input

Screen flow will be modified based on the screen order stored in the database as shown in Figure 4.4.

Once the screen order is changed in the web portal, the order of screens in the mobile application also changes, as shown in Figure 4.5.

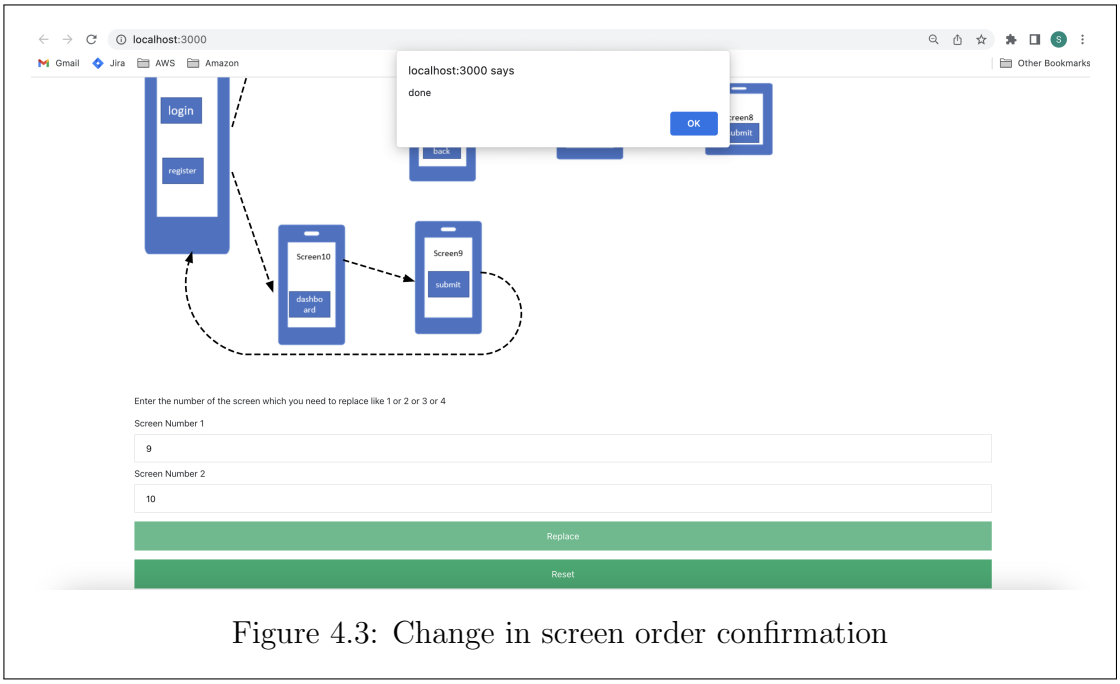


Figure 4.3: Change in screen order confirmation

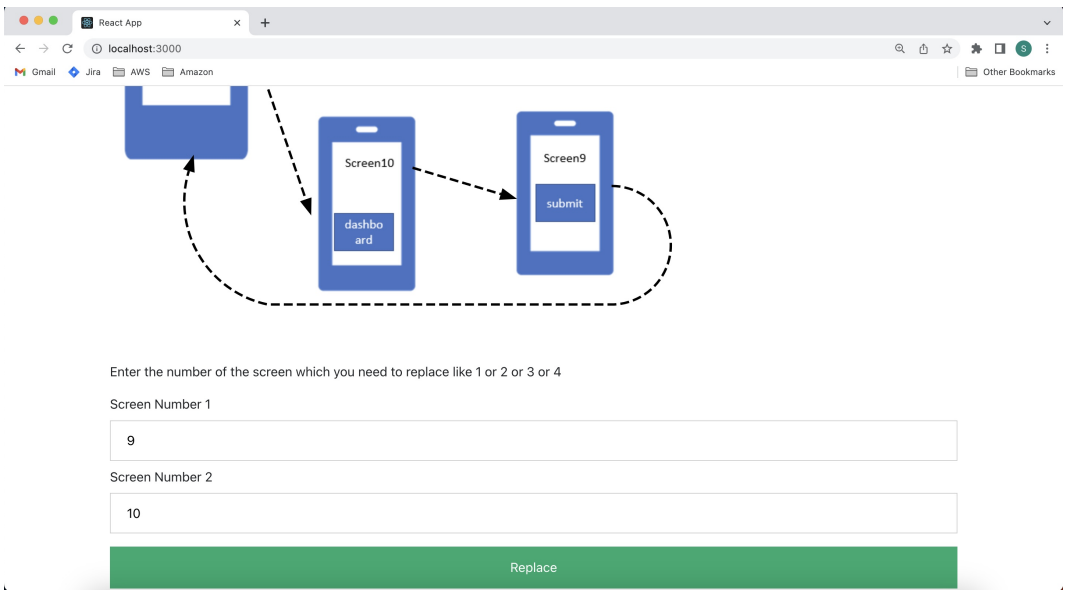


Figure 4.4: Screen flow of a mobile application

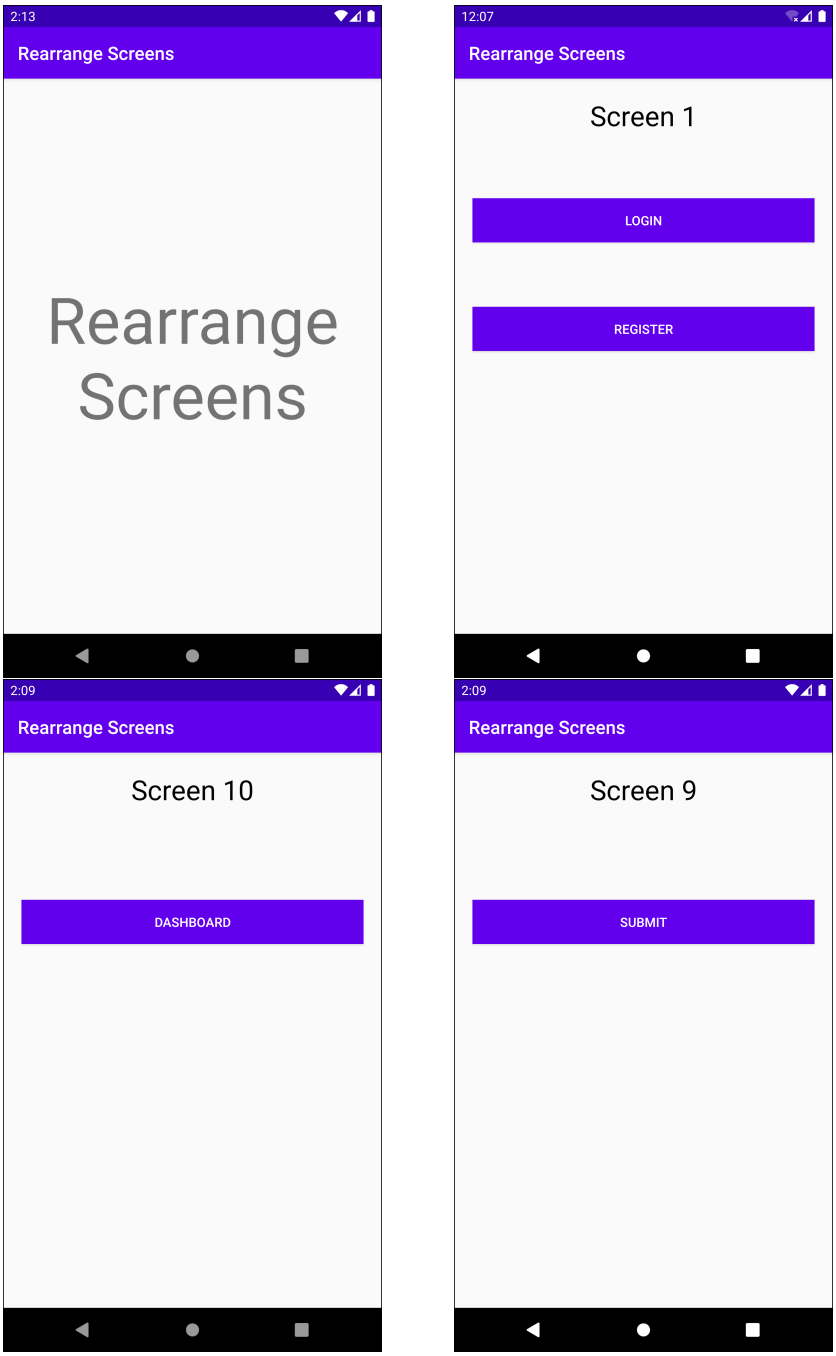


Figure 4.5: Order of screens in Android app

Chapter 5: Conclusion

Reshape is a platform for mobile apps that allows the screens of a mobile application to be changed remotely without having to publish an app update. When an Android app is updated, it must be reviewed by the Google Play Store, which can take 24 to 48 hours or even longer. But with Reshape, the Product Owner can immediately reorder the screens of an application already live in the app marketplace. However, there are a few limitations. The screens that the Product Owner wants to rearrange must be independent of each other. This means that if the Product Owner wants to exchange the current screen and a screen that displays data that depends on the response from the API call that happens on the current screen, they will be unable to use Reshape to do so. Also, Reshape was developed for screens that have a maximum of two buttons but could be extended for use with screens with more buttons.

Chapter 6: Scope of improvement

Currently, Reshape can rearrange the screens of an Android application. It will be possible to extend this functionality to iOS applications as well. It would also be helpful if Reshape could rearrange dependent screens. Screen configuration of the Android mobile application is another area that can be automated. Also, the UI of the web portal of Rearrange Screens can be more interactive.

References

C.G.,Thomas and Devi,A.Jayanthila. (2021). A Study and Overview of the Mobile App Development Industry. *International Journal of Applied Engineering and Management Letters*, *5*(1), 115–130. Retrieved 2022-12-13, from https://srinivaspublication.com/wp-content/uploads/2021/06/10.-A-study-and-overview_Fullpaper.pdf doi: 10.47992/IJAEML.2581.7000.0097

Dani, D., Tomar, V., Srivastava, V., & Bindal, V. (2021). A Research Paper on Football Website Development Using ReactJS/Firebase. *International Journal of All Research Education Scientific Methods*, *9*(7), 9. Retrieved from http://www.ijaresm.com/a-research-paper-on-football-website-development-using-reactjs/firebase

Islam, D. M. R., & Mazumder, T. (2010). Mobile application and its global impact. *International Journal of Engineering & Technology*, *10*(6), 72–78. Retrieved from https://www.researchgate.net/publication/308022297_Mobile_application_and_its_global_impact

Sehgal, T. (2020). *Google Login and Logout in Android With Firebase (Kotlin Implementation).* Retrieved 2022-11-23, from https://medium.com/swlh/google-login-and-logout-in-android-with-firebase-kotlin-implementation-73cf6a5a989e

Umar, M. A. (2019). Comprehensive study of software testing: Categories, levels, techniques, and types. *International Journal of Advance Research, Ideas and Innovations in Technology*, *5*, 32-40. Retrieved from https://www.semanticscholar.org/paper/Comprehensive-study-of-software-testing%3A-levels%2C-Umar/a917c32fef7b8fc2a515b1ce7e7daa5128ee735a