VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANA SANGAMA, BELAGAVI – 590 018

An Internship Project Report
on

Recipe App

Submitted in partial fulfillment of the requirements for the VIII Semester of
degree of **Bachelor of Engineering in Information Science and Engineering** of
Visvesvaraya Technological University, Belagavi

by

Mohammed Misran
1RN18IS067

Under the Guidance of

Dr. R Rajkumar
Associate Professor
Department of ISE

Department of Information Science and Engineering

RNS Institute of Technology

**Dr. Vishnuvaradhan Road, Rajarajeshwari Nagar post,
Channasandra, Bengaluru-560098**

2021-2022

RNS INSTITUTE OF TECHNOLOGY

Dr. Vishnuvaradhan Road, Rajarajeshwari Nagar post, Channasandra, Bengaluru - 560098

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Internship work entitled *Recipe App* has been successfully completed by **Mohammed Misran (1RN18IS067)** a bonafide student of **RNS Institute of Technology, Bengaluru** in partial fulfillment of the requirements of 8th semester for the award of degree in **Bachelor of Engineering in Information Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during academic year **2021-2022**. The internship report has been approved as it satisfies the academic requirements in respect of internship work for the said degree.

_____   _____   _____

Dr. R Rajkumar   **Dr. Suresh L**   **Dr. M K Venkatesha**
Internship Guide   Professor and HoD   Principal
Associate Professor   Department of ISE   RNSIT
Department of ISE   RNSIT

External Viva

Name of the Examiners   **Signature with Date**

1. _____   **1.** _____

2. _____   **2.** _____

DECLARATION

I, **MOHAMMED MISRAN [USN: 1RN18IS067]** student of VIII Semester BE, in Information Science and Engineering, RNS Institute of Technology hereby declare that the Internship work entitled ***Recipe App*** has been carried out by me and submitted in partial fulfillment of the requirements for the *VIII Semester degree of* ***Bachelor of Engineering in Information Science and Engineering*** *of Visvesvaraya Technological University, Belagavi* during academic year 2021-2022.

Place: Bengaluru
Date: 10-01-2022

MOHAMMED MISRAN

(1RN18IS067)

ABSTRACT

The use of mobile devices such as, smartphone or tablets have increased significantly in the past decade. All these devices use applications that are created for them. These applications can provide many different services including, social media, music streaming, video streaming, ride sharing, online shopping, and video games. Some of these apps need to be constantly connected to the internet to function properly, while others can work offline.

Who said our smartphones don't belong in the kitchen? These days, you can use your phone and tablet to pull up a recipe for any dish: Mexican, Italian, Lebanese, or any other cuisine you can dream of. Whether you run iOS or Android, there's no shortage of freemium and reasonably priced recipe apps.

This report discusses a recipe mobile application that allows users to search for and explore various meal recipes based on several categories. Users can browse a collection of recipes, as well as their prep time and type (vegetarian, non-vegetarian, etc.). Each recipe is accompanied by a well-known YouTube video. The app aims to be efficient while maintaining an intuitive and simple design that offers the user with all of the key features.

ACKNOWLEDGEMENT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ACID: Atomicity, Consistency, Isolation, Durability

API: Application Programming Interface

BSON: Binary JavaScript Object Notation

CPU: Central Processing Unit

CRUD: Create, Read, Update, Delete

DB: Database

DX: Developer Experience

IoT: Internet of Things

JSON: JavaScript Object Notation

MEAN: MongoDB, Express, Angular, Node.js

MERN: MongoDB, Express, React, Node.js

MEVN: MongoDB, Express, Vue.js, Node.js

MIT: Massachusetts Institute of Technology

NoSQL: Not Only SQL

OS: Operating System

RAM: Random Access Memory

SQL: Structured Query Language

SSPL: Server Side Public License

UI: User Interface

UX: User Experience

XML: Extended Markup Language

Chapter 1

INTRODUCTION

1.1 About the Company

ENMAZ Engineering Services Pvt. Ltd.

Figure 1.1 ENMAZ Logo

ENMAZ provides end-to-end solutions and development services for Industrial Internet of Things (IoT) requirements. ENMAZ specializes in:

- IoT
- Industrial IoT
- IoT Analytics
- Embedded Hardware Development
- Embedded Firmware Development
- Cloud Support
- Big Data Analysis and Reporting
- Dashboard UX/UI Design and Development
- Model Analysis

ENMAZ has a team of highly passionate industrial designers & professional engineers with strong educational background from the most reputed institutes of India with cumulative industry experience of 50+ years, from various domains.

1.2 Overview of the Project

Recipe Software is a smartphone app that allows users to browse a wide range of cuisine dishes. The user interface of the application is simple but engaging. The application begins with a splash screen, followed by a welcome screen. Users must first log in before continuing to use the application. After successfully checking in, users are presented with a variety of options. Each recipe has a well-known YouTube video associated with it. The users can then choose their favourite dishes and test out the recipe.

Flutter, an open-source framework for creating beautiful, natively compiled, multi-platform apps from a single codebase, was used to create the user interface. The software may be used on both Android and iOS because to its cross-platform approach.

The API was built using Express, a minimal and flexible Node.js web application framework, in conjunction with MongoDB, an open-source cross-platform document-oriented database.

The code was written in Visual Studio Code, a code editor for building and debugging modern apps, and the API was tested in Postman, a platform for building and using APIs.

The development approach may appear simple because cross-platform programming and the API-First paradigm work hand in hand. However, this is only true if the API is already operational on a production server. Tunneling, a mechanism for sending data from one network to another, was used in the development process as a result.

Finally, Git, a free and open-source distributed version control system, was used to manage code changes and coordinate work among team members.

1.3 About Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, Web platform, and the web from a single codebase.

The major components of Flutter include:

1) *Dart platform*
   Flutter apps are written in the Dart language and make use of many of the language's more advanced features.
2) *Flutter engine*
   Flutter's engine, written primarily in C++, provides low-level rendering support using Google's Skia graphics library. Additionally, it interfaces with platform-specific SDKs such as those provided by Android and iOS.
3) *Foundation library*
   The Foundation library, written in Dart, provides basic classes and functions that are used to construct applications using Flutter, such as APIs to communicate with the engine.
4) *Design-specific widgets*
   The Flutter framework contains two sets of widgets that conform to specific design languages: Material Design widgets implement Google's design language of the same name, and Cupertino widgets implement Apple's iOS Human interface guidelines.

Flutter uses a variety of widgets to deliver a fully functioning application. These widgets are Flutter's framework architecture. Flutter's Widget Catalog provides a full explanation and API on the framework.

1.4 About Express

Express.js, or simply Express, is a backend web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

The original author, TJ Holowaychuk, described it as a Sinatra-inspired server, meaning that it is relatively minimal with many features available as plugins. Express is the back-end component of popular development stacks like the MEAN, MERN or MEVN stack, together with the MongoDB database software and a JavaScript front-end framework or library.

1.5 About MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

Main features:

- Ad-hoc queries
- Indexing
- Replication
- Load balancing
- File storage
- Aggregation
- Server-side JavaScript execution
- Capped collections
- Transactions

MongoDB has official drivers for major programming languages and development environments. There are also a large number of unofficial or community-supported drivers for other programming languages and frameworks.

Chapter 2

LITERATURE REVIEW

2.1 Cross-Platform App Development

In today's highly disruptive and Darwinian mobile app development world, businesses wouldn't risk missing their presence on either platform: Google Play Store or the Apple App Store. Budgeting, however, is usually an issue if businesses go for native apps. This is why cross-platform app development has emerged as the unrivalled choice of businesses that aim for a presence on both Android and iOS.

Cross-platform application development is about building a single application that can run on various operating systems, instead of developing different app versions for each platform. The driving force for cross-platform application development is to produce software that works well in more than one specific digital environment, with the main purpose of selling it to a wider customer base.

2.1.1 Benefits of Cross-Platform App Development

1) *Launching the software simultaneously on various platforms*
   When you work on cross-platform app development, you can launch your software quickly on various platforms. The source code is written once for all platforms. This means you don't need to hire a separate software development team for each platform, as it's possible to launch and update the software by using a variety of cross-platform development tools.

2) *Faster development time*
   It requires the deployment of a single script instead of writing separate scripts for each platform. This significantly speeds up development time and cuts time to market, which benefits everyone, from the dev team all the way to marketing.

3) *Reaching a wider audience*
   Cross-platform application development offers you an opportunity to reach a larger audience. If your app is compatible with multiple platforms and operating systems like the web, iOS, and Android, it can be used by a larger number of users. It's a great way to maximize your exposure – with less effort and time.

4) *Faster and easier updates*
   Since cross-platform applications are Internet-based, updates are nice and easy. Users don't have to download separate updates, which would require the maintenance and support of multiple app versions. The app is updated automatically for all customers to ensure they always have the most current version of the app, which positively impacts its performance.

5) *Cost savings and shorter time to profitability*

All of the above-mentioned advantages entail significant cost savings. With the development speed cross-platform development brings, your time-to-market for each platform is shorter than if you had to create each app from scratch. This means you can get your software to generate revenue much sooner. You don't need to maintain a software development team for each platform you launch your app to, which will bring significant savings over time.

2.1.2 Challenges in Cross-Platform App Development

- Performance hiccups because of inconsistent communication between the native and non-native components of gadgets.
- Cross-platform app developers have difficulty maintaining cross-compliance of apps across devices and operating systems.
- Performance-related glitches can lead to poor user experience.
- If a business app manages part of a corporation and stores the user's data, then going for cross-platform apps is not always a good idea due to security concerns.

However, these challenges are minimal when compared to the benefits of cross-platform app development.

The frameworks for cross-platform app development are Xamarin, Flutter, React Native, Ionic, and many more.

2.2 API-First Approach

Web APIs have been around for nearly 20 years, but it is only in the past few years that the concept of "API first" has gained traction with software teams. The number of developers taking an API-first approach to building products is rising.

An Application Programming Interface, or API for short, is a set of functions that allows an application to interact with external applications, operating systems, microservices, or data. In other words, APIs allow applications to "talk to" and interact with one another.

API-first development puts APIs at the foundation, instead of pre-built or opinionated software solutions or experiences. API-first development ensures that all of the functionality inside the platform is accessible to you through the API.

2.2.1 Benefits of API-First Approach

1) *Development teams can work in parallel*
   API first involves establishing a contract. Creating a contract between services that is followed by teams across an organization allows those teams to work on multiple APIs at the same time. Developers do not have to wait for updates to an API to be released before moving on to the next API. Teams can mock APIs and test API dependencies based on the established API definition.

2) *Reduces the cost of developing apps*
   APIs and code can be reused on many different projects. When a development team wants to build a new app, they don't have to start from scratch which is time-consuming and costly. API-first design also allows most problems to be solved before any code is even written which helps prevent problems when it is time to integrate APIs with applications.

3) *Increases the speed to market*
   Much of the process of building APIs can be automated using tools that allow import of API definition files. API first also makes it possible to add new services and technologies to applications without having to re-architect the entire system.

4) *Ensures good developer experiences*
   Consumers of APIs are most often developers, and developer experience (DX) can make or break the success of an API. API first ensures that developers have positive experiences using your APIs. Well-designed, well-documented, consistent APIs provide positive developer experiences because it's easier to reuse code and onboard developers, and it reduces the learning curve.

5) *Reduces the risk of failure*
   For most companies, APIs are used in nearly every business process – from marketing and sales to communication and consumer-facing applications, which means that APIs can impact every part of your business positively or negatively. API first reduces the risk of failure by ensuring that APIs are reliable, consistent, and easy for developers to use.

2.3 NoSQL

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.

2.3.1 Benefits of NoSQL Database

1) *Flexible schemas*
   NoSQL databases typically have very flexible schemas. A flexible schema allows you to easily make changes to your database as requirements change. You can iterate quickly and continuously integrate new application features to provide value to your users faster.

2) *Horizontal scaling*

Most SQL databases require you to scale-up vertically (migrate to a larger, more expensive server) when you exceed the capacity requirements of your current server. Conversely, most NoSQL databases allow you to scale-out horizontally, meaning you can add cheaper, commodity servers whenever you need to.

3) *Fast queries due to the data model*

Queries in NoSQL databases can be faster than SQL databases. Data in SQL databases is typically normalized, so queries for a single object or entity require you to join data from multiple tables. As your tables grow in size, the joins can become expensive. However, data in NoSQL databases is typically stored in a way that is optimized for queries. Queries typically do not require joins, so the queries are very fast.

4) *Ease of use for developers*

Some NoSQL databases like MongoDB map their data structures to those of popular programming languages. This mapping allows developers to store their data in the same way that they use it in their application code. While it may seem like a trivial advantage, this mapping can allow developers to write less code, leading to faster development time and fewer bugs.

2.3.2 Drawbacks of NoSQL Database

One of the most frequently cited drawbacks of NoSQL databases is that they don't support ACID (atomicity, consistency, isolation, durability) transactions across multiple documents. With appropriate schema design, single record atomicity is acceptable for lots of applications. However, there are still many applications that require ACID across multiple records.

Since data models in NoSQL databases are typically optimized for queries and not for reducing data duplication, NoSQL databases can be larger than SQL databases. Storage is currently so cheap that most consider this a minor drawback, and some NoSQL databases also support compression to reduce the storage footprint.

Depending on the NoSQL database type you select, you may not be able to achieve all of your use cases in a single database. When selecting a NoSQL database, consider what your use cases will be and if a general-purpose database like MongoDB would be a better option.

Chapter 3

ANALYSIS

3.1 Analysis of User Interface

Among various types of mobile apps, the ones devoted to food present a very popular category. Recipes and cooking tips, restaurant and food delivery apps, calorie trackers and food diaries – more and more applications now help people to keep on with all the faces of eating, which means UX designers work on a variety of interfaces of that kind.

We wanted to step aside from the traditional recipe app where users just save the directory of the favourite meals, taken from the app database, or add their own recipes. We had a goal to create a bit more universal food app for users who love cooking. It includes the recipe database which is constantly updated.

The app design should include comprehensive and diverse functionality which is to be presented to users in a simple and clear way. We had to analyze and prioritize all the points, as there was a high risk of overloading the screen. By research and testing, the user scenarios were created to determine which information about the meal in the recipe is found the most important.

As the recipe app is aimed at daily basic operations and quite a diverse target audience, the user interface has to be super easy and accessible for users with different levels of tech-literacy and all types of mobile devices. The application layout must be structured around intuitive navigation, high readability, light background, and eye-catching visuals.

Photography is a good way to impress users with realistic and clear visuals as well as set the needed associations. With rapidly developing photo stock websites, designers have more and more opportunities to find good images; still, for many projects, especially e-commerce ones, the creative teams shoot the original content totally corresponding to the goals of the product. It is especially noticeable in the spheres close to everyday life: fashion, toys, food, drinks, etc.

3.2 Analysis of Data

In building a recipe app, information on how to turn separate ingredients into something tasty should mean the world to you. Without them, your application doesn't make sense. There are three ways to fill your mobile app with recipes:

1) ***Integrate already existing databases of recipes and/or ingredients by using free/paid APIs.***
   Pros:
   - Pre-made databases include thousands of recipes and ingredients, so you'll have a pretty good base to start with from the very beginning.
   - This is the fastest way – you just integrate a ready solution into your application.

Cons:

- Really good databases aren't free and will cost you a lot.
- You depend on someone else – they may stop supporting the database or change the pricing policy and you'll have to adjust accordingly.
- If you start using someone else's DB, you need to carefully think about what added value you will give to your users nevertheless.

2) *Create your own database.*
   Pros:

- You don't depend on someone else.
- You keep your core business values in-house.
- You get only what you need.
- Public databases include a wide range of recipes that may not necessarily make much sense for you if you're building a niche app.
- You are free to create content that your competitors probably don't have.

Cons:

- It will take quite a lot of time and resources to set up and maintain such a database from scratch.

3) *Let your users upload their recipes and fill up the application in such a way.*
   Pros:

- You don't have to compose the database – your users will do it for you!

Cons:

- You have to put a lot of effort into the moderation of users' recipes. Otherwise, your app will turn into a dump pretty soon (repeated recipes, low-quality content, etc.).
- In the early stages, you have to encourage users somehow – a few will want to use the empty app and be among the first to fill it out.
- You need to find early adopters and motivate them to constantly post relevant content.

Chapter 4

SYSTEM DESIGN

4.1 User Interface

The user interface (UI), in the industrial design field of human–computer interaction, is the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operators' decision-making process.

4.1.1 Initial User Interface

Figure 4.1 UI Layout for Splash Screen
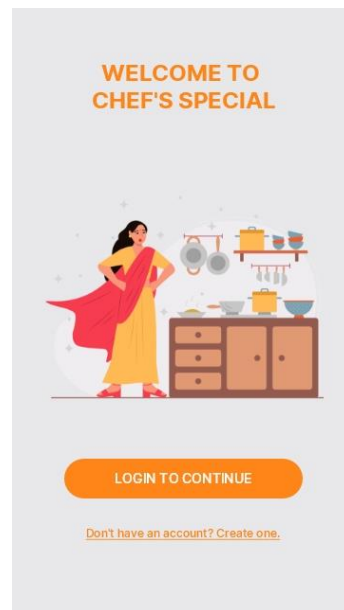
Figure 4.2 UI Layout for Welcome Screen
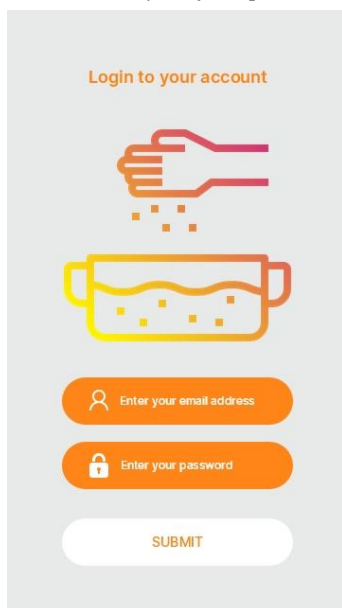
Figure 4.3 UI Layout for Login Screen

Figure 4.4 UI Layout for Home Screen

4.2 API Design

API design refers to the process of developing application programming interfaces (APIs) that expose data and application functionality for use by developers and users. Simplicity of API design depends on the context. A particular design may be simple for one use case but very complex for another, so the granularity of API methods must be balanced. It can be useful to think about simplicity on several levels, including:

1) **Data format:** Support of XML, JSON, proprietary formats, or a combination.
2) **Method structure:** Methods can be very generic, returning a broad set of data, or very specific to allow for targeted requests. Methods are also usually called in a certain sequence to achieve certain use cases.
3) **Data model:** The underlying data model can be very similar or very different to what is actually exposed via the API. This has an impact on usability, as well as maintainability.
4) **Authentication:** Different authentication mechanisms have different strengths and weaknesses. The most suitable one depends on the context.
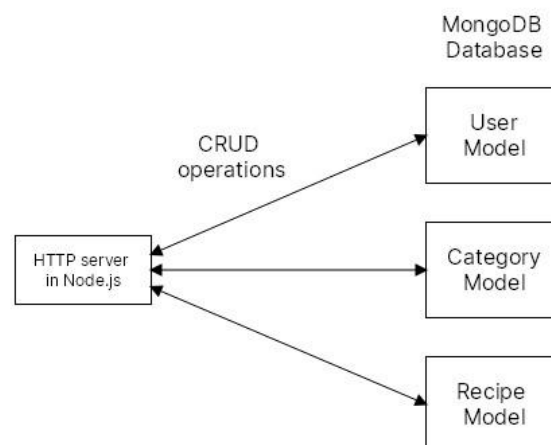
4.2.1 API Design for Recipe App

Figure 4.5 API design for Recipe App

- **Data Format:** Data is stored in BSON format
- **Method Structure:** CRUD operations create, read, update and delete documents
- **Data Model:** MongoDB data modeling
- **Authentication:** User credentials are stored in the User model

Chapter 5

DETAILED DESIGN

5.1 Widget Trees

The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description in order to determine the minimal changes needed in the underlying render tree to transition from one state to the next. The widget tree is how developers create their user interface; developers position widgets within each other to build simple and complex layouts.

Widgets are generally defined in three basic types: Stateful widgets, Stateless widgets, and Inherited widgets. Being the central class hierarchy in the Flutter framework the three basic types of widgets are used in the construction of every Flutter application. Although all the instances of a widget are immutable, the Stateful widget allows the interaction between user and application. By giving access to the method setState, the state can be maintained in separate state objects. Alternatively, the Stateless widget acts as a constant, and before anything displayed can be changed, the widget has to be recreated. The Inherited widget works by allowing another widget to subscribe to the Inherited widget's state allowing the state to be passed down to its children.
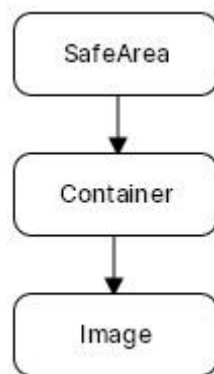
5.1.1 Splash Screen

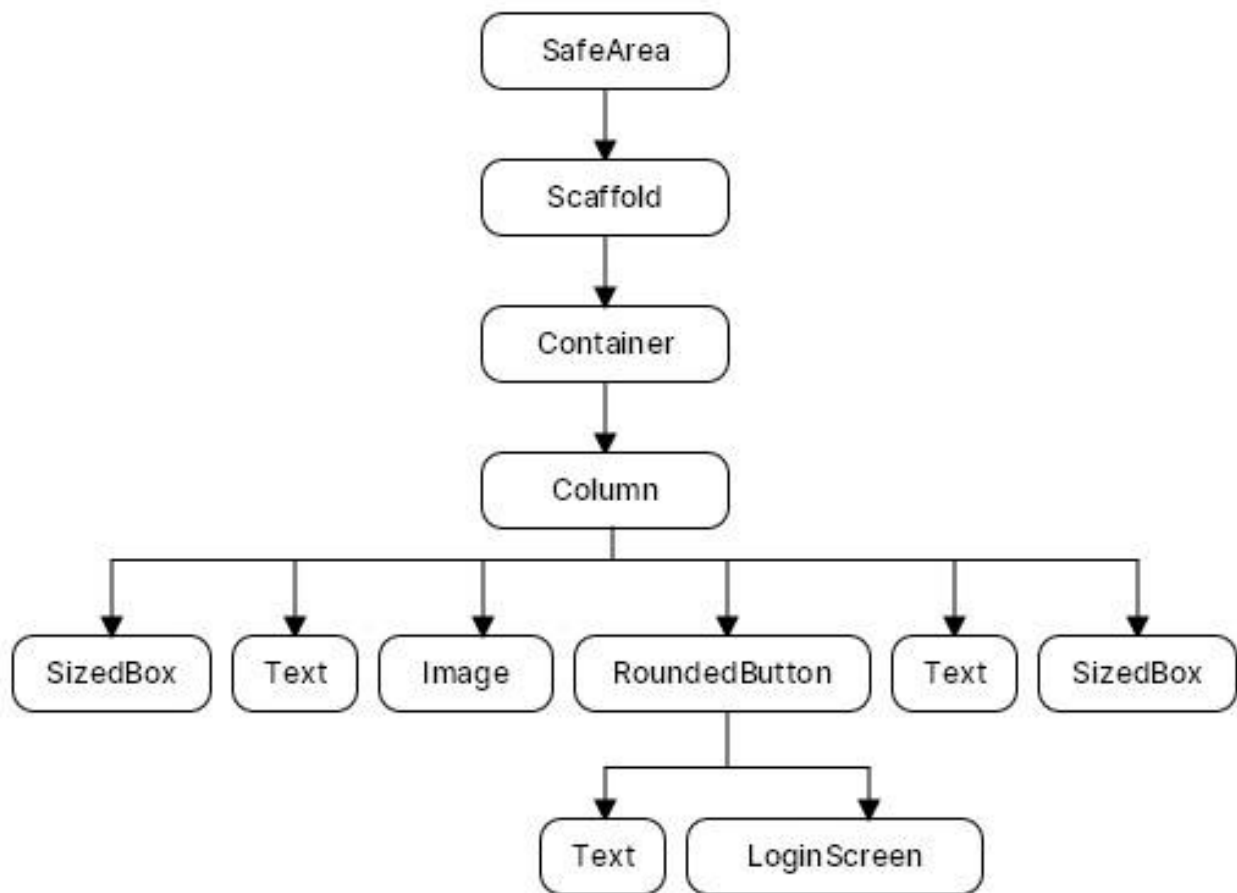
Figure 5.1 Splash Screen Widget Tree

5.1.2 Welcome Screen

Figure 5.2 Welcome Screen Widget Tree
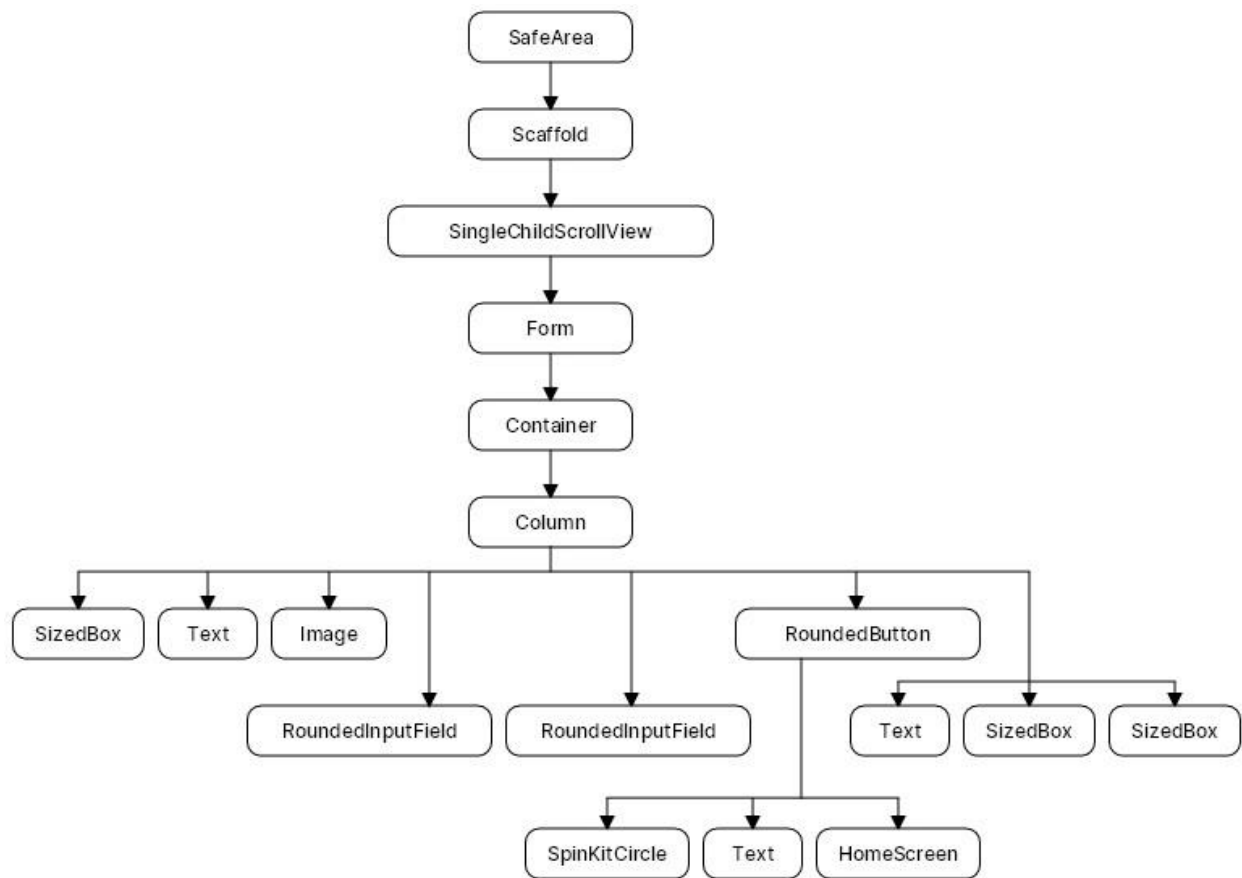
5.1.3 Login Screen

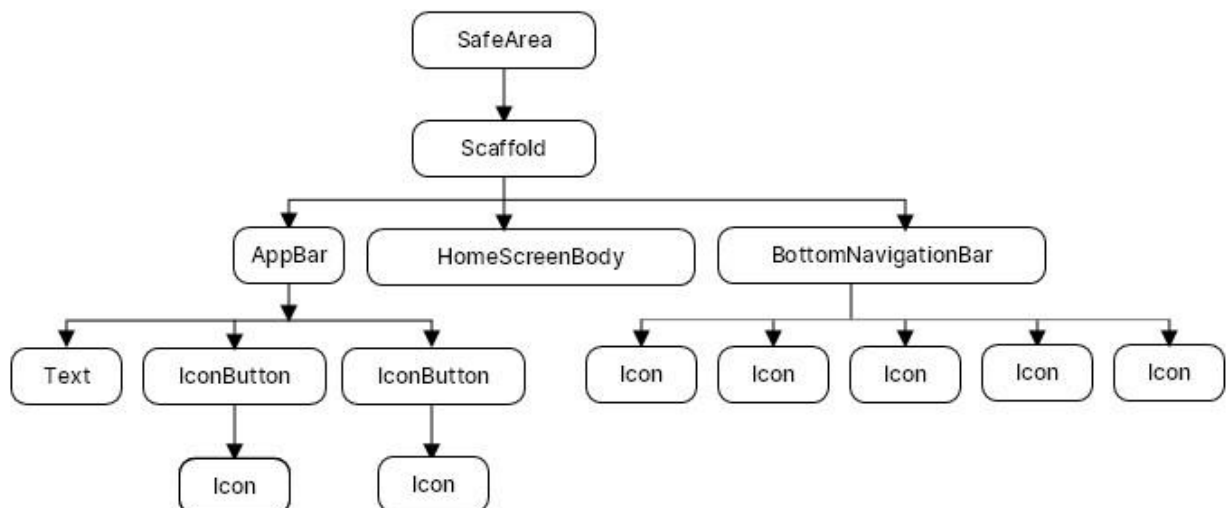*Figure 5.3 Login Screen Widget Tree*

5.1.4 Home Screen

Figure 5.4 Home Screen Widget Tree

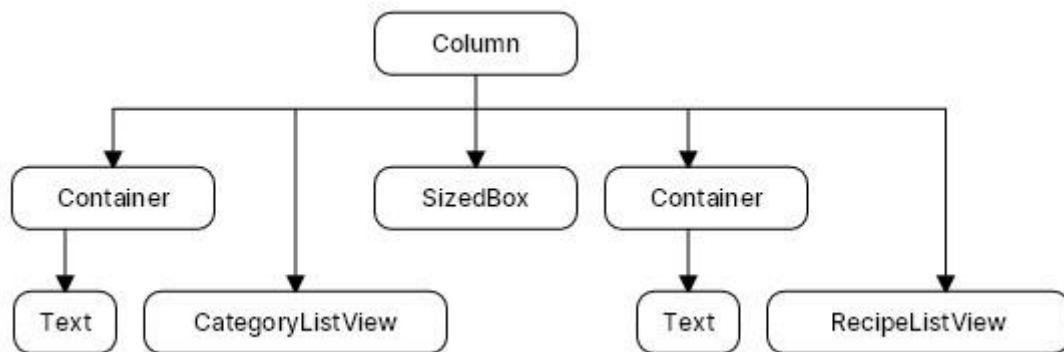## 5.1.5 Home Screen Body

Figure 5.5 Home Screen Body Widget Tree
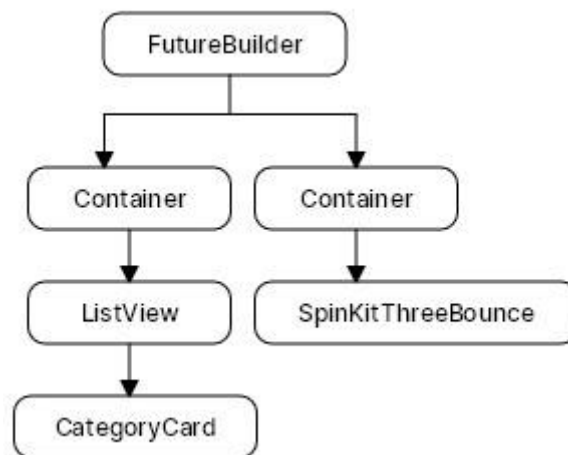
5.1.6 Category List View

Figure 5.6 Category List View Widget Tree
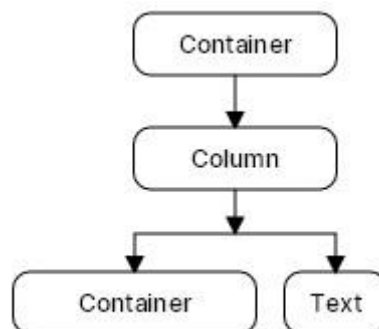
5.1.7 Category Card

Figure 5.7 Category Card Widget Tree
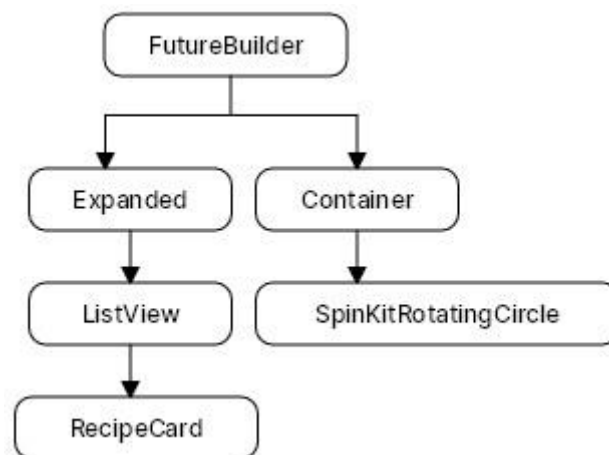
5.1.8 Recipe List View

Figure 5.8 Recipe List View Widget Tree
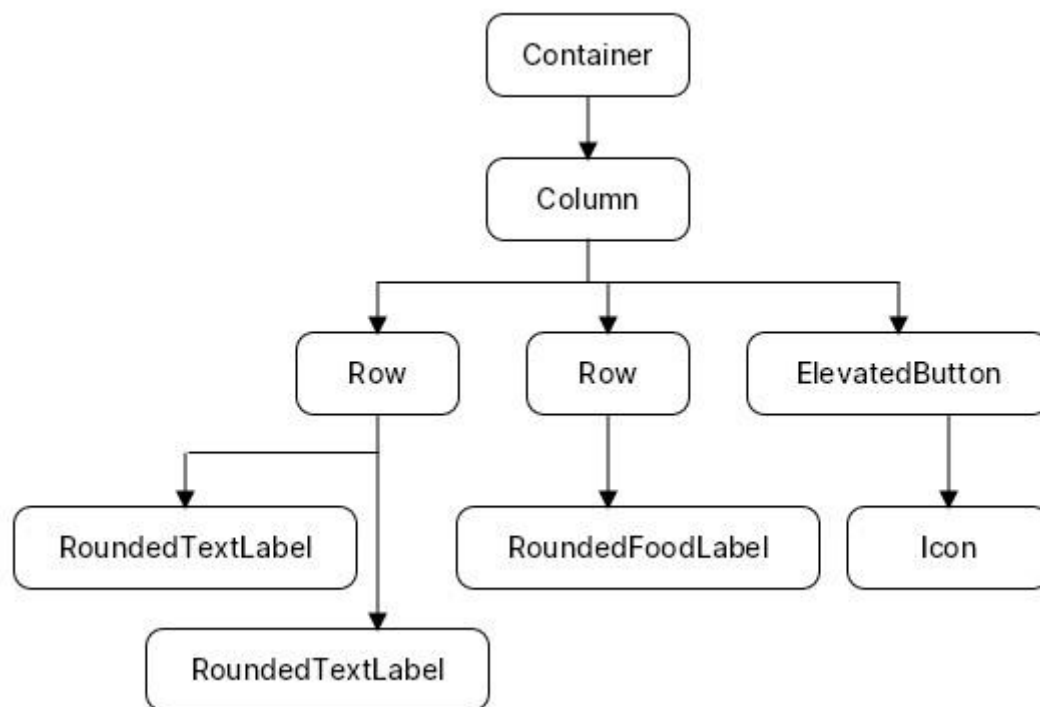
5.1.9 Recipe Card

Figure 5.9 Recipe Card Widget Tree
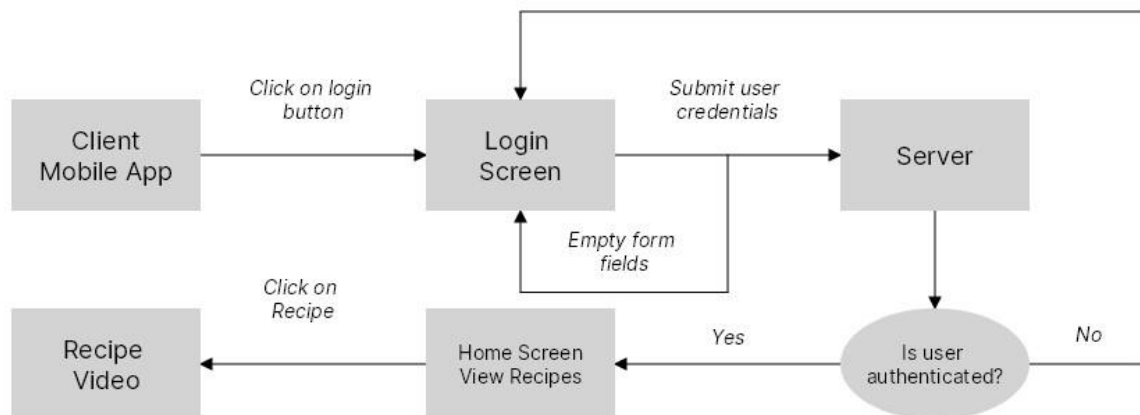
5.2 Data Flow

Figure 5.10 Data Flow Diagram for Recipe App

- The user must login to use the app.
- If he/she enters invalid credentials, he/she will be redirected to the login screen again.
- On successful login, the user is sent to the home screen where he/she is presented with a list of different recipes.
- The recipes data is fetched from the backend server before rendering the data on the screen.
- The user can then select a recipe of his choice and he/she will be presented with a video showing the steps to make the dish.
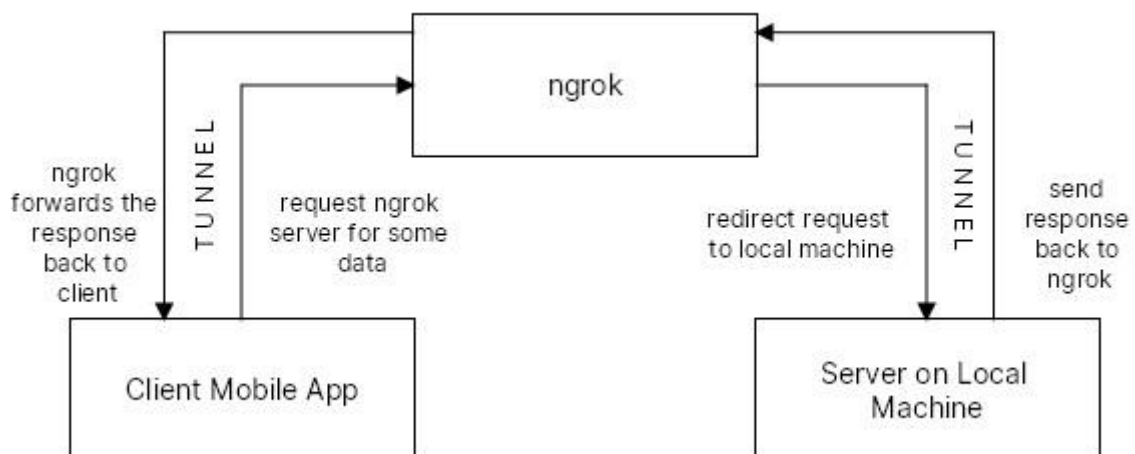

Figure 5.11 Tunneling traffic from client to local server

- By making use of tunnels, we can redirect traffic from the client (i.e., mobile app) to the server running on our local machine.

Chapter 6

IMPLEMENTATION

Implementation is the process of defining how the system should be built, ensuring that it is operational and meets quality standards. It is a systematic and structured approach for effectively integrating software-based service or component into the requirements of end users.

6.1 Project Specification

6.1.1 Software

- **Frontend** – Flutter
- **Flutter Dependencies** – cupertino_icons, http, flutter_spinkit, flutter_dotenv, url_launcher
- **Backend** – JavaScript (Node.js)
- **Node.js Dependencies** – crypto-js, dotenv, express, mongoose
- **Database** – MongoDB Atlas
- **Tools** – Visual Studio Code, Postman
- **Version Control** – Git

6.1.2 Hardware

- **Operating System** – Windows 10 Home 64-bit
- **Processor** – Intel® Core™ i5-10300H CPU @ 2.50GHz
- **RAM** – 8 GB

6.2 Discussion of Code

6.2.1 Entry Point for Recipe App

```
void main() async {
  await dotenv.load(fileName: 'lib/assets/config.env');
  runApp(RecipeApp());
}

class RecipeApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(fontFamily: 'FuzzyBubbles'),
      home: SplashScreen(),
    ); // MaterialApp
  }
}
```

Figure 6.1 Entry Point for Recipe App

- The dotenv.load() method will load configuration at runtime from a .env file which can be used throughout the application.
- The runApp() method will inflate the given widget and attach it to the screen.

6.2.2 Login Functionality

```
Future<bool> validatePassword(User user) async {
  final response = await http.post(
    Uri.parse("${dotenv.env['BACKEND_SERVER']}/api/v1/users/login"),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8'
    },
    body: jsonEncode(<String, String>{
      "email": user.email,
      "password": user.password,
    }),
  );

  return (response.statusCode == 200);
}
```

Figure 6.2 validatePassword() method

- The http.post() method is used to send user credentials data to the backend server.
- If the response status code is 200, then the user is successfully logged in.

6.2.3 Data Binding

```
class Category {
  String id;
  String title;
  String image;

  Category({required this.id, required this.title, required this.image});

  factory Category.fromJson(dynamic json) {
    return Category(
      id: json['_id'],
      title: json['title'],
      image: json['image'],
    );
  }
}
```

Figure 6.3 Category class

- The fromJson() method is used to serialize JSON inside model classes.
- In the above example, the constructor constructs a new Category instance from a map structure.

6.2.4 Fetch Recipes from Server

```
Future<List<Recipe>> fetchRecipes() async {
  final response = await http
      .get(Uri.parse("${dotenv.env['BACKEND_SERVER']}/api/v1/recipes"));

  final responseBody = jsonDecode(response.body);

  if (response.statusCode == 200) {
    final recipes = responseBody['recipes'] as List;
    List<Recipe> results =
        recipes.map((recipe) => Recipe.fromJson(recipe)).toList();
    return results;
  }

  return <Recipe>[];
}
```

Figure 6.4 Fetch recipes from API

- The http.get() method is used to send GET requests to the server.
- The JSON response from the server is decoded and converted to a list of Recipe instances.

6.2.5 Launch Recipe Video

```
class RecipeCard extends StatelessWidget {
  final Recipe recipe;

  const RecipeCard({required this.recipe});

  String convertPrepTime(int prepTime) {···

  void _launchInBrowser(String url) async {
    if (!await launch(url)) {
      throw "Could not launch $url";
    }
  }

  @override
  Widget build(BuildContext context) {···
}
```
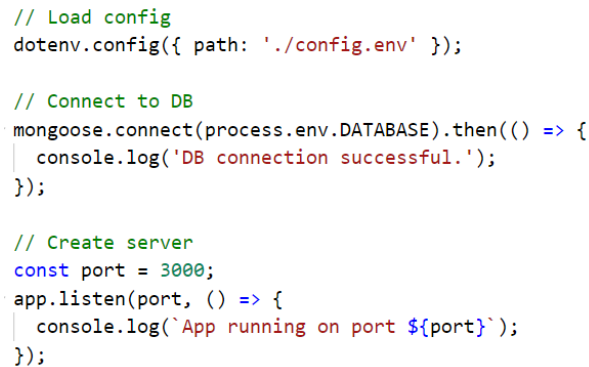
Figure 6.5 RecipeCard widget

- The launch() method parses the specified URL string and delegates handling of it to the underlying platform.
- If successful, the video opens in the client's browser.

6.2.6 Create HTTP Server

```
// Load config
dotenv.config({ path: './config.env' });

// Connect to DB
mongoose.connect(process.env.DATABASE).then(() => {
  console.log('DB connection successful.');
});

// Create server
const port = 3000;
app.listen(port, () => {
  console.log(`App running on port ${port}`);
});
```

Figure 6.6 Creating server

- Load runtime configuration using dotenv.
- Connect to the MongoDB database using mongoose.
- Start the server and listen on port 3000.

6.2.7 Create Express App

```
// Create express app
const app = express();

// Body-parser
app.use(express.json({ limit: '10kb' }));

// Routers
app.use('/api/v1/categories', categoryRouter);
app.use('/api/v1/recipes', recipeRouter);
app.use('/api/v1/users', userRouter);

// Unhandled routes
app.all('*', (req, res) => {
  res.status(404).json({ status: 'fail', message: `${req.url} not found.` });
});
```

Figure 6.7 Create express app

- Create the express app.
- Add middleware to handle request body parsing.
- Add middleware to handle both known and unknown routes.

6.2.8 Mongoose User Schema

```
const userSchema = mongoose.Schema({
  name: { ...
  },
  email: { ...
  },
  password: { ...
  },
  salt: { type: String, select: false },
});

userSchema.pre('save', function (next) {
  this.salt = CryptoJs.lib.WordArray.random(128 / 8);
  const hash = CryptoJs.PBKDF2(this.password, this.salt, {
    keySize: 128 / 32,
  }).toString();
  this.password = hash;
  next();
});

userSchema.methods.validatePassword = function (candidatePwd, hash, salt) {
  const candidateHash = CryptoJs.PBKDF2(candidatePwd, salt, { keySize: 128 / 32 });
  return candidateHash == hash;
};

const User = mongoose.model('User', userSchema);
```

Figure 6.8 User schema

- Before saving user data to the database, take the raw password and generate a hash of this password. Store this hashed password in the database along with its salt value.
- For validating password, create a hash of the candidate password using the salt value of stored in the database. If the generated hash matches the hash stored in the database, then return true; otherwise, false.

6.2.9 Read All Recipes from the Database

```
exports.getAllRecipes = async (req, res) => {
  const recipes = await Recipe.find();
  res.status(200).json({ status: 'success', recipes });
};
```

Figure 6.9 Get all recipes

- The find() method will return all the documents of the corresponding model from the database.
- Return the list of documents as a JSON response.

Chapter 7

TESTING

Software Testing is the process used to help identify the correctness, completeness, security and quality of the developed computer software. Testing is the process of technical investigation and includes the process of executing a program or application with the intent of finding errors.

Case ID	Test Case	Expected Result	Actual Result	Status
1	Click on "LOGIN TO CONTINUE" button	Display login screen to the user	Displays login screen to the user	Pass
2	Enter valid user credentials in login form and submit the form	Process form data by sending it to the server	Processes form data by sending it to the server	Pass
3	Submit empty login form	Display error message "Invalid Credentials"	Displays error message "Invalid Credentials"	Pass
4	Fetch recipes by sending GET request to the server	Get the list of recipes with response status code	Response status code is 200 Recipe list is returned	Pass
5	Validate user password by sending POST request to the server	Perform validation check to see if password matches with user in the database	If password matches, respond with status code 200 Otherwise respond with status code 401	Pass

Table 7.1 Test Cases

Chapter 8

RESULTS
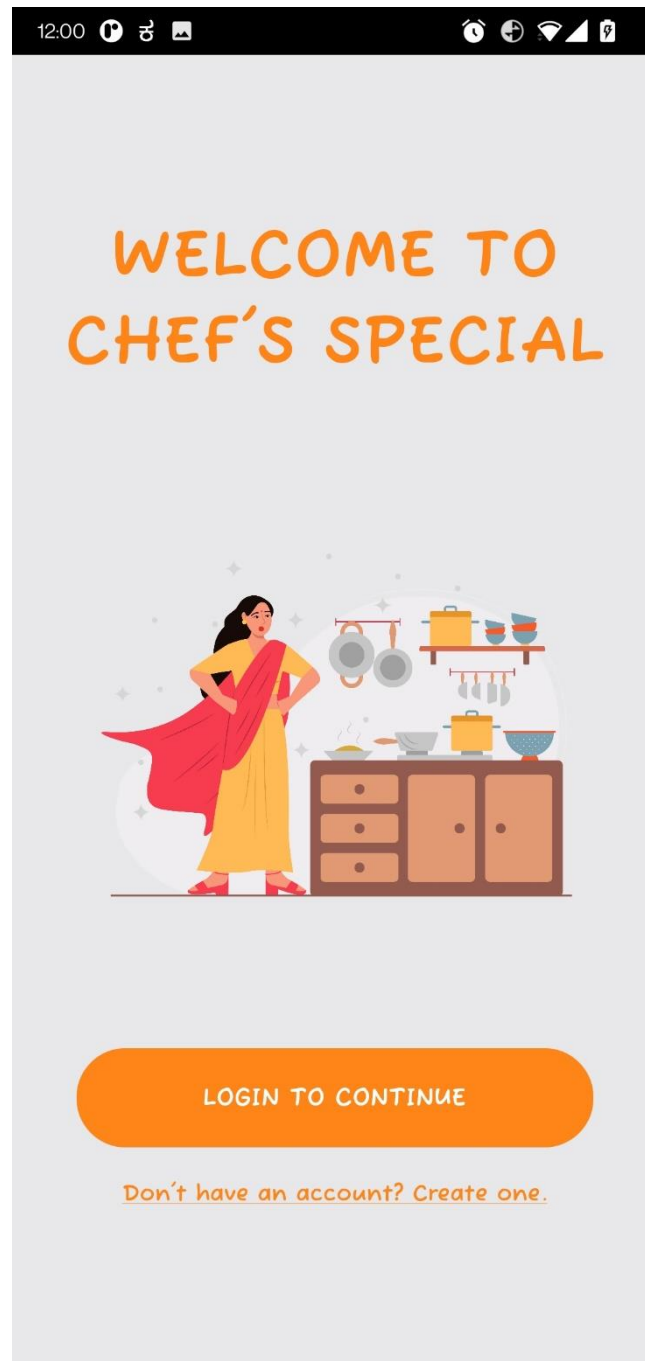

Figure 8.1 Splash Screen Output
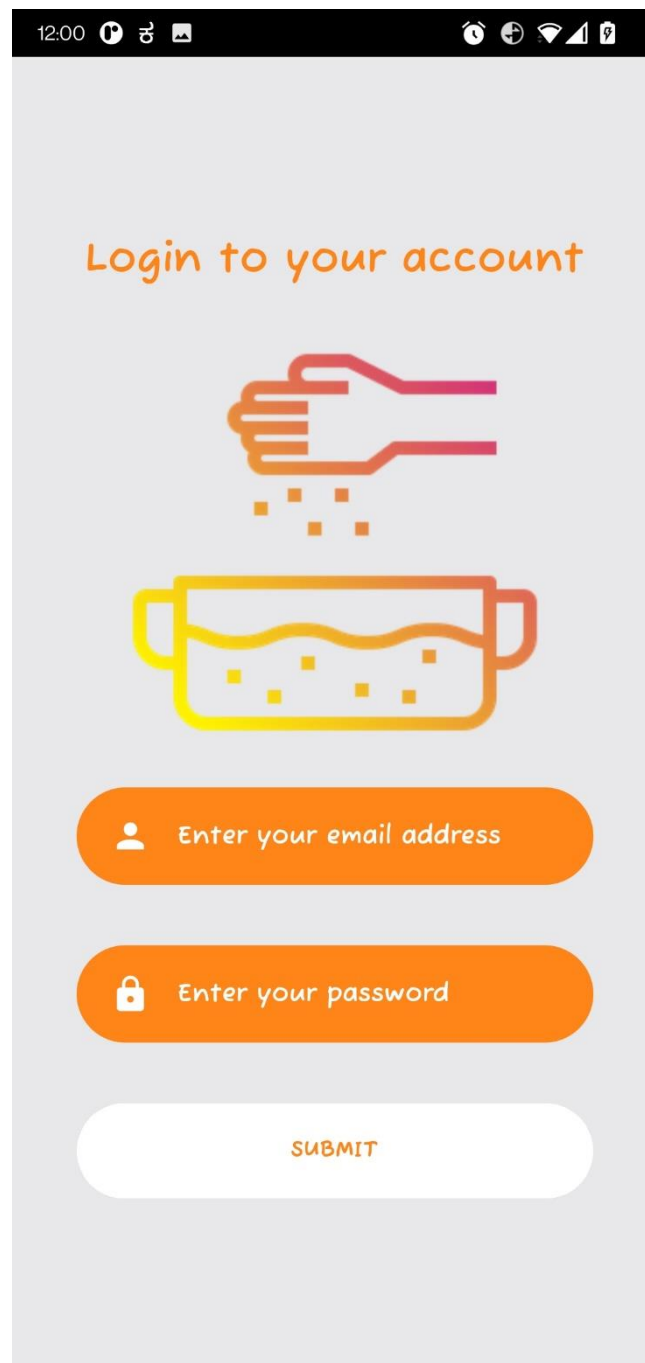
Figure 8.2 Welcome Screen Output

Figure 8.3 Login Screen Output
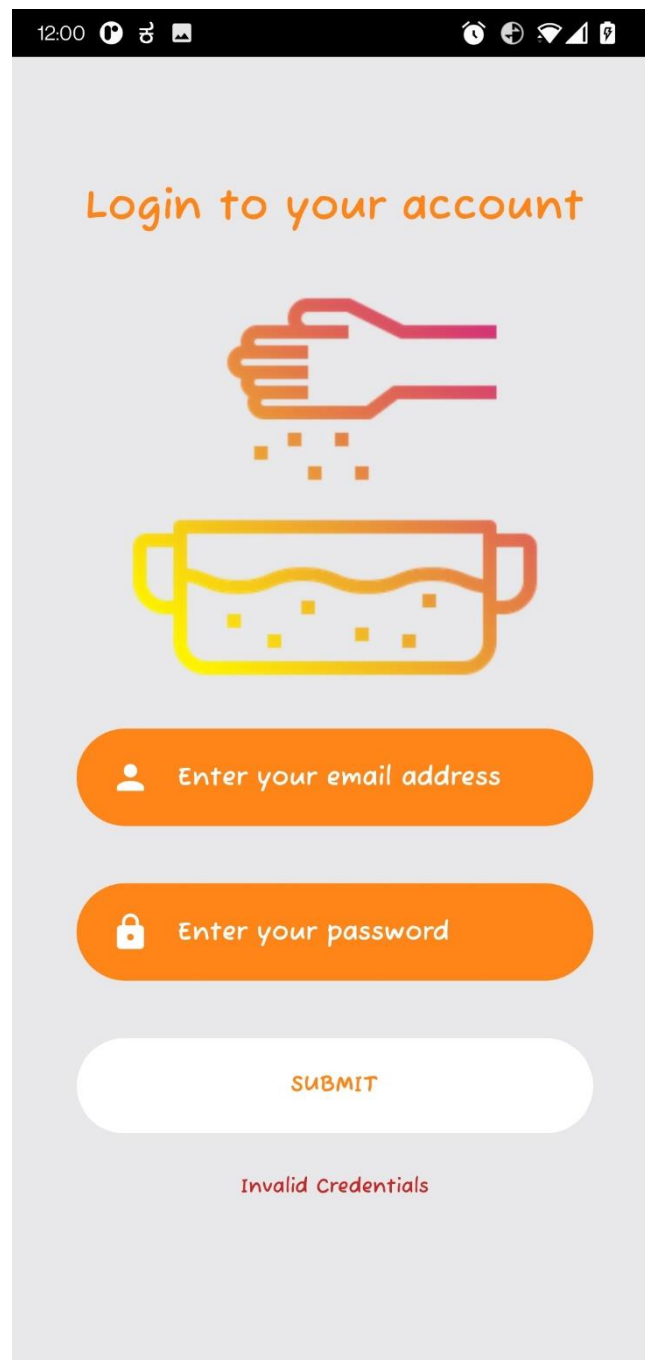
Figure 6.4 Login Screen Processing Output

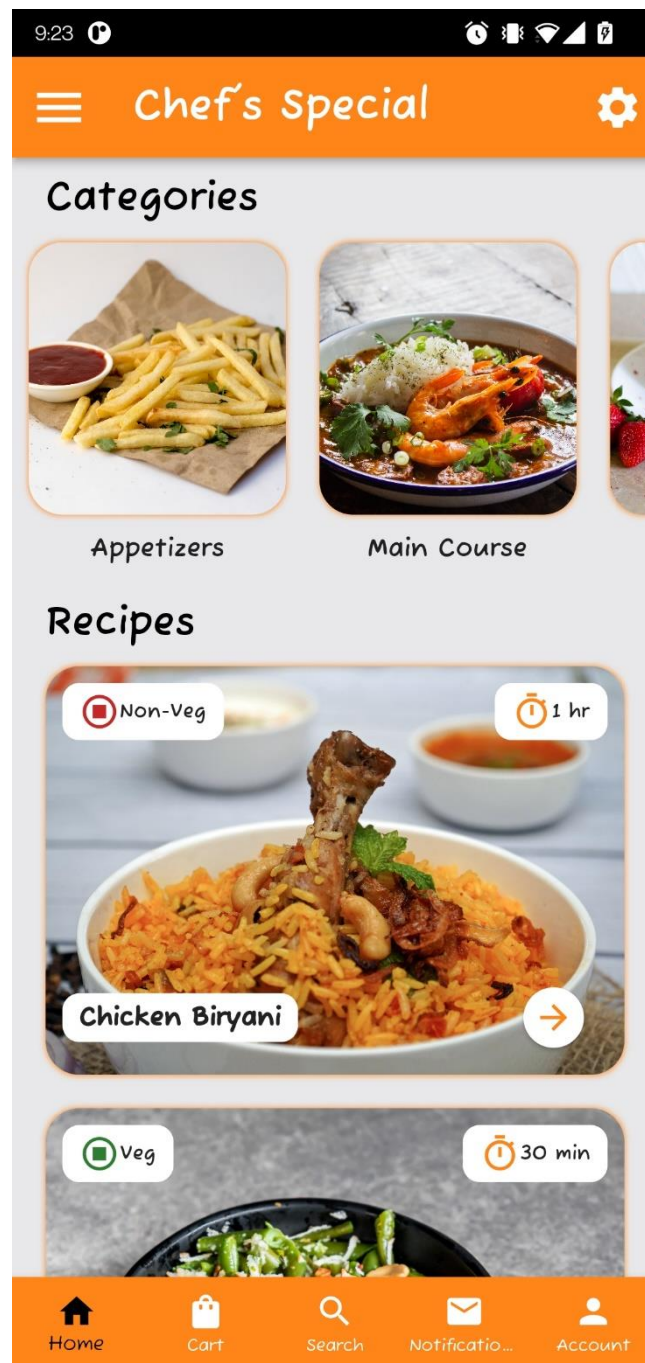*Figure 6.5 Login Screen Error Output*

Figure 6.6 Home Screen Output

Chapter 9

CONCLUSION AND FUTURE ENHANCEMENT

9.1 Conclusion

This application is extremely handy and useful for cooking variety of recipe with minimum search effort from internet. It will help people to save their time and energy in finding recipes for daily routine as well as for special occasions. And since this is a mobile application, users have the luxury to check for recipes wherever they are. Also, since we have developed the application in a cross-platform framework, the application can be used by a broad range of users. However, there are also some features that the application is short of such as searching and filtering, categorizing recipes, saving recipes, etc.

9.2 Future Enhancements

- Additional screen for each recipe, which shows the ingredients and text description of it.
- Enhanced interactivity, allowing users to add their own recipes.
- Organizing and moderating user-added recipes in a proper way.
- Bookmarking recipes.
- Personalization feature to let the users customize the app settings for their own experience.
- Additional security features such as signing up users and verifying them, two-factor authentication, account backup, and API security features.

REFERENCES

[1] https://docs.flutter.dev/

[2] https://nodejs.org/dist/latest-v16.x/docs/api/

[3] https://expressjs.com/en/5x/api.html

[4] https://docs.atlas.mongodb.com/

[5] https://ngrok.com/docs

[6] https://www.redhat.com/en/topics

[7] https://asperbrothers.com/blog/

[8] https://www.netsolutions.com/insights/

[9] https://www.sitecore.com/da/knowledge-center/blog