

Universal Booking App

A Project Report

*Submitted in the partial fulfilment of the requirement
for the award of the degree of*

Bachelor of Technology

in

COMPUTER ENGINEERING

by

Gaurav Kolhatkar (10303320191124510064)

Anurag Nandaghale (10303320181124510034)

Under the guidance of

Dr. A. W. Kiwelekar



DEPARTMENT OF COMPUTER ENGINEERING
DR. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY
Lonere - 402103, Tal. - Mangaon, Dist. - Raigad (M.S.) INDIA.

Certificate

The report entitled Universal Booking App submitted by Gaurav Kolhatkar (10303320191124510064), Anurag Nandaghale (10303320181124510034) for the partial fulfilment of the requirement for the award of the degree of Bachelor of Technology in Computer Engineering.

Dr. A.W. Kiwelekar
Guide
Department of Computer Engineering

Dr. L. D. Netak
Head
Department Of Computer Engineering

Examiners:

1. _____ (Name: _____)

2. _____ (Name: _____)

Place: Dr. Babasaheb Ambedkar Technological University, Lonere

Date:

Acknowledgements

This work is just not an individual contribution till its completion. I take this opportunity to thank all for bringing it close to the conclusion. A special thanks goes to my Guide Dr. A. W. Kiwelekar, for leading me to many new insights, encouraging and teaching me how to get down to the root of the problem.

I wish to express my sincere thanks to the Head of the department of Computer Engineering, Dr. L. D. Netak. I am also grateful to the department faculty and staff members for their support.

I would also like to thank all my friends and well-wishers for support during the demanding period of this work. I would also like to thank my wonderful colleagues and friends for listening my ideas, asking questions and providing feedback and suggestions for improving my ideas.

Mr. Gaurav Kolhatkar

Mr. Anurag Nandaghale

Abstract

In day-to-day life we have to face wastage of time in many places. Most of the time we waste in waiting for our slot/number/chance in queues at certain places such as hospitals, restaurants, saloon shops, etc.

Many businesses or institutions when require solution to this problem they have to custom made their own application or have to use old or outdated ones.

Our app solves this problem and saves time for customers/public and help businesses or institutions to manage booking system and time efficiently without any hurdles with the help of technology without any need of communication for booking.

Contents

1 Introduction	1
2 Problem Statement	2
3 System Architecture	3
3.1 Dataflow Diagram	3
3.2 Use Case Diagram	4
3.3 UML (Unified Modelling Language).....	5
4 Understanding the application	6
4.1 Authentication	6
4.2 Dashboard.....	8
4.2.1 Business Dashboard	9
4.2.1 Customer Dashboard.....	8
4.3 Menu	10
4.3.1 Settings.....	11
4.4 Search	12
4.5 Booking Terminal.....	13
5 Tech-Stack.....	14
5.1 Frontend.....	14
5.1.1 React.....	14

5.1.2 CSS.....	16
5.2 API.....	17
5.2.1 Node.js and Express.js	17
5.3 Backend	18
5.3.1 Firebase	18
5.3.2 Firebase-Auth.....	19
5.3.3 Cloud Firestore.....	20
6 Implementation.....	21
6.1 Code Snippets	29
7 Applications.....	30
8 Future Scope	31
9 Bibliography.....	32

List Of Figures

Fig 3.1 DataFlow Diagram	3
Fig 3.2 UseCase Diagram	4
Fig 3.3 UML Diagram	5
Fig 4.1 Login Page	6
Fig 4.2 Signup Page	7
Fig 4.3 Customer Dashboard	8
Fig 4.4 Business Dashboard.....	9
Fig 4.5 Dropdown Menu.....	10
Fig 4.6 Settings Dialog box	11
Fig 4.7 Search Page.....	12
Fig 4.8 Slot booking page	13

Chapter 1

Introduction

Our App is for slot bookings and get tokens anywhere and anytime. This software can be used by any businesses or intuitions, whoever wants to implement a booking system, whether they are small or well established, there customers can easily do bookings without visiting the place.

Application contains feature for businesses or intuitions to register themselves. They have to update necessary details about the business and people can watch available time slots and book them.

People can save their time in searching for shops, restaurants, hospitals, etc and also save time by booking available slots online from anywhere.

It also helps in emergency situations like checking available vacant hospitals and book them rather than visiting there.

When the time closes to booked slot both the users get notified that their slot has arrived, and they can reach the place at right time. Due to this user don't have to wait in queues and can save their time, also the businesses or institutions which have limited infrastructures does not need worry about the space which would be consumed by the long queues.

Chapter 2

Problem Statement

In day-to-day life we face problem of waiting in queues in many places, and we face many problems due to this such as:

- **Waste of time:** Waiting in queue for getting our turn waste time for many people, we want to solve the problem so that everyone can use their 24 hrs. a day efficiently. E.g.: Saloon shop, Hospitals, Restaurant, etc.
- **Spread of diseases:** Waiting many people in a queue lead to formation of a crowd. In crowd diseases gets spread very fast. Especially in current days corona is a big threat and we can avoid people waiting in queue to reduce formation of crowd.
- **Infrastructure:** We need huge space, room or shade for if we have to make many people to wait in queue.
- **Environment:** We may face problem of weather conditions like storm, lightning, raining which can cause people harm who are standing in queue.
- **Health Risks:** By standing in a queue for a long time causes number of potentially serious health outcomes, such as lower back and leg pain, cardiovascular problems, fatigue, discomfort, and pregnancy related health outcomes.

So, our application that will reduce all these problems to great extent. Our application helps us by letting business or service provider creating online slots and allowing clients or customers to book vacant slots.

Chapter 3

System Architecture

3.1 Dataflow Diagram

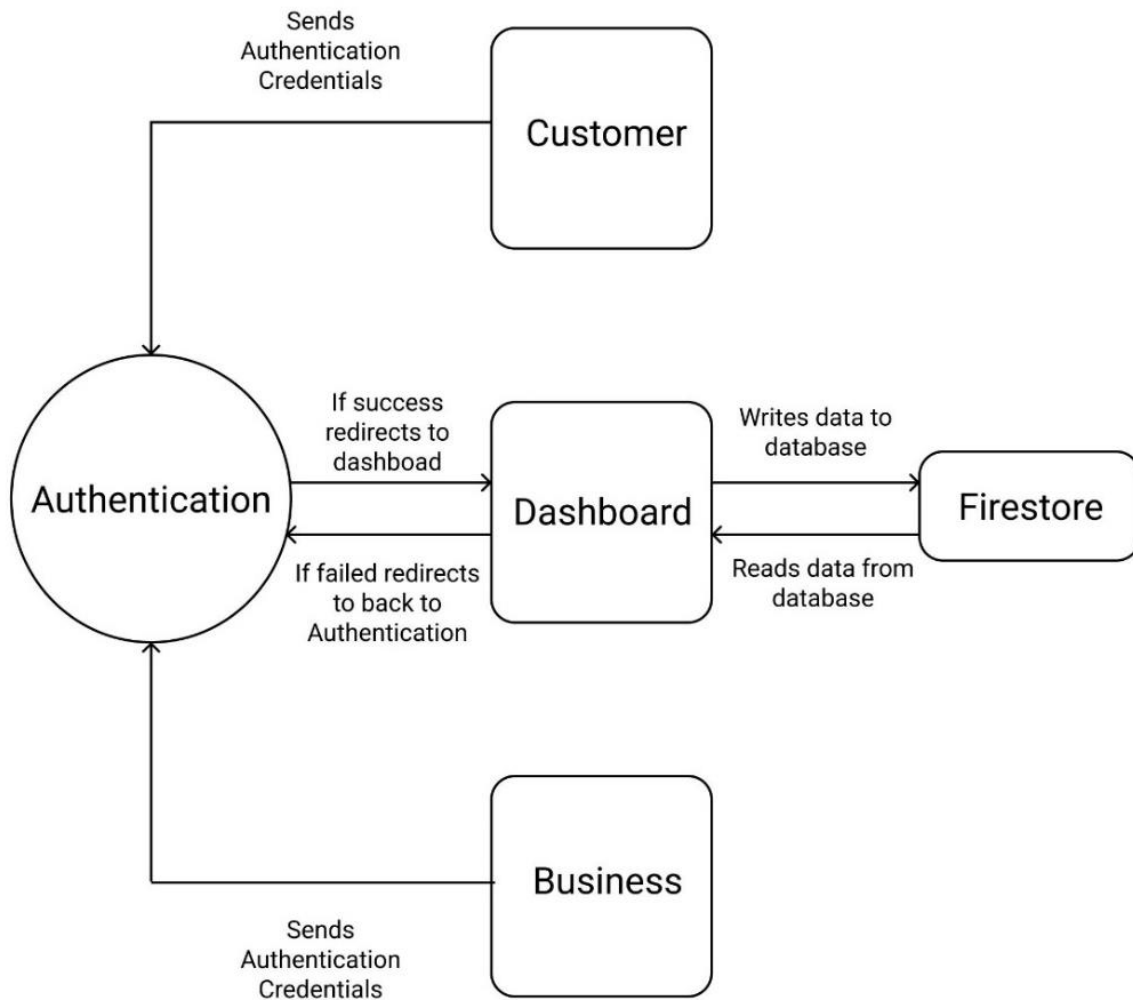


Figure 3.1: Dataflow diagram

3.2 Use Case Diagram

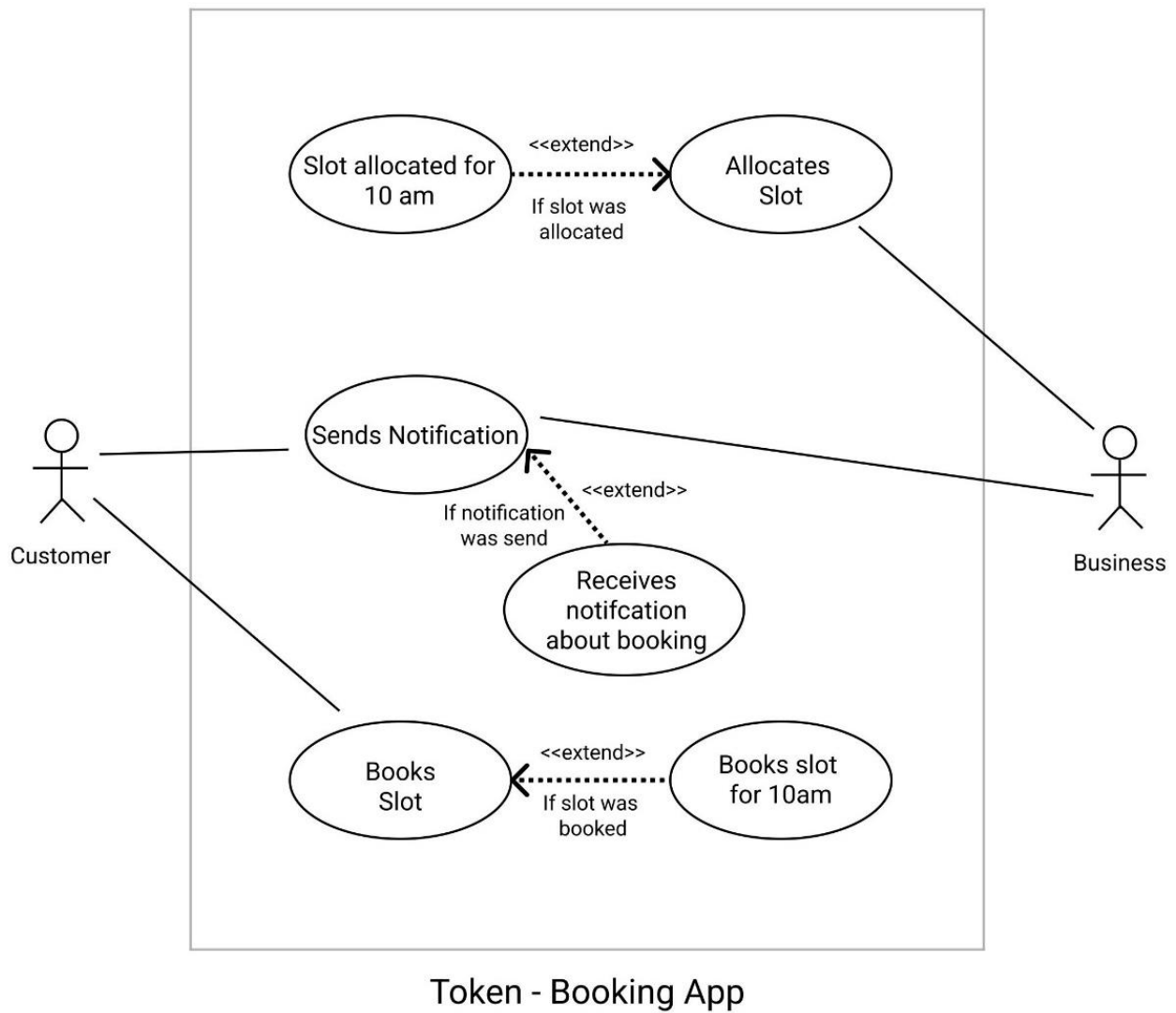


Figure 3.2: Use case diagram

3.3 UML (Unified Modelling Language)

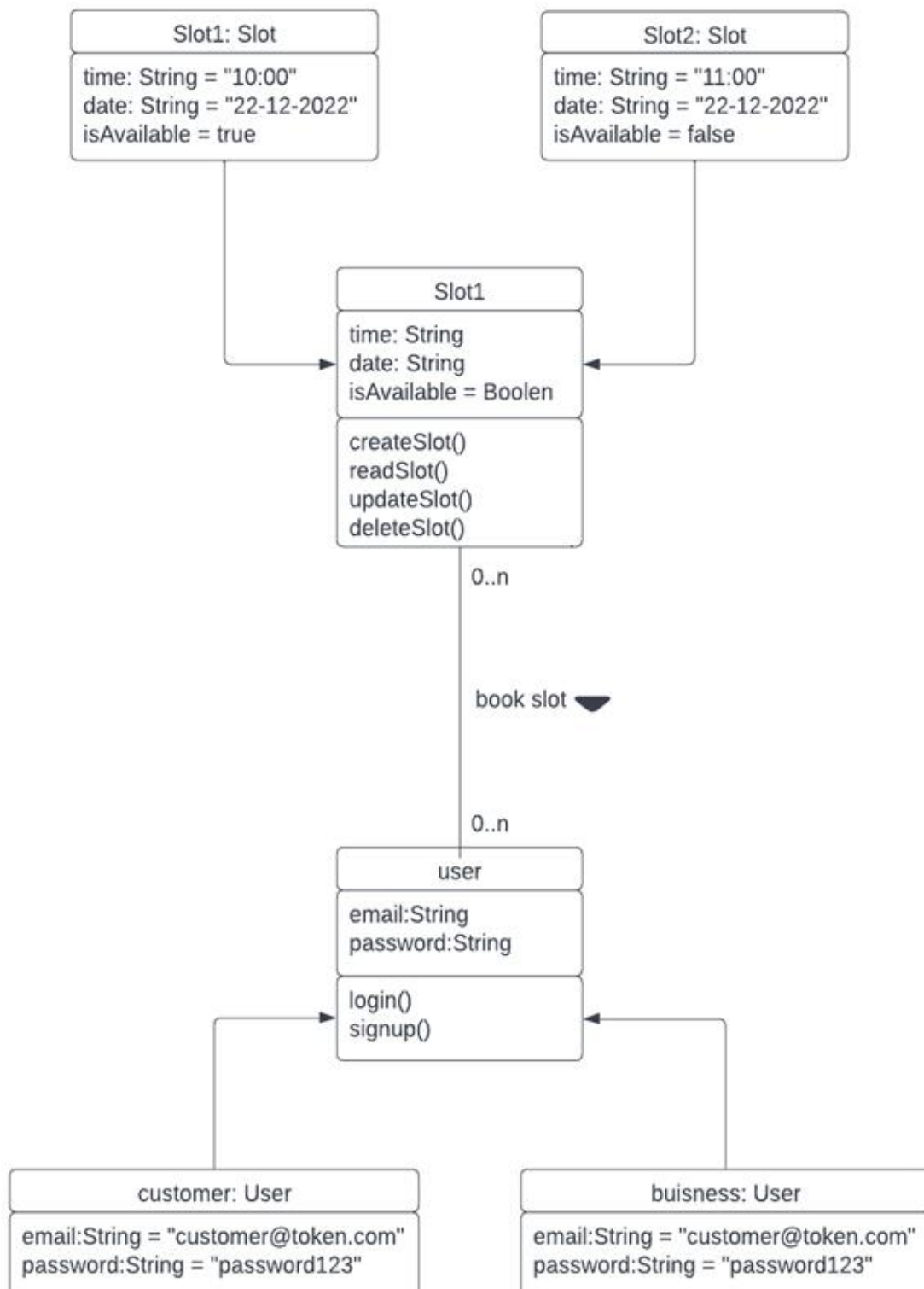


Figure 3.3: UML diagram

Chapter 4

Understanding the application

4.1 Authentication

Users of our application will be business, start-ups, and their customers. We need authentication so that we can restrict any third party to change the bookings, slot availability, settings and everything that is set by user. It will also help us to differentiate between each user and maintain their respective profile, and dashboard, and can apply different settings on the profile.

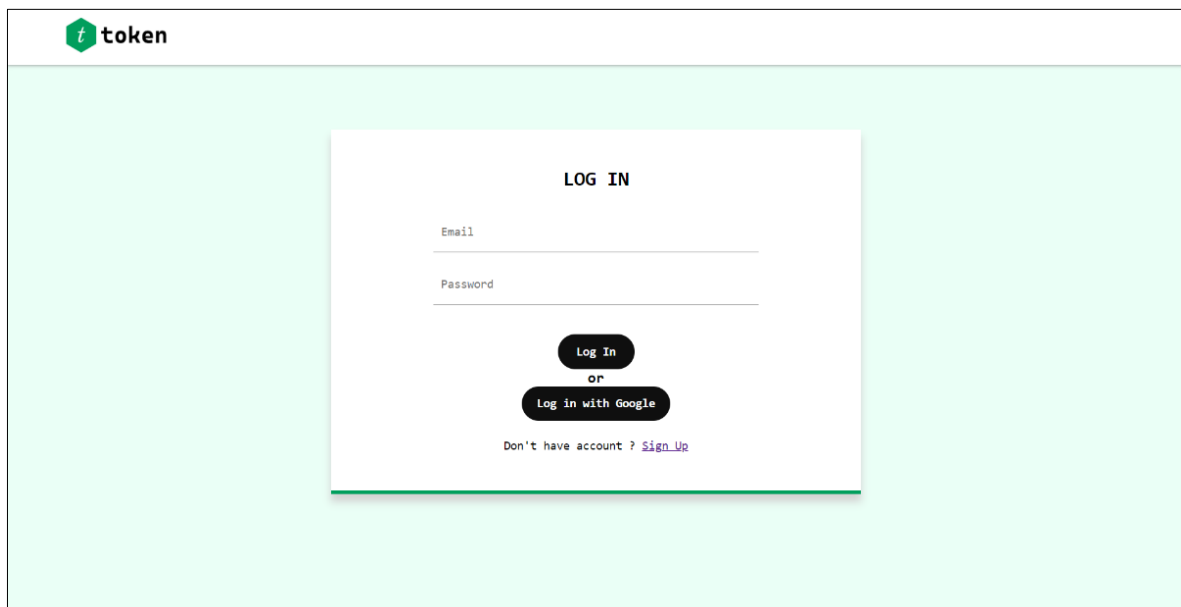


Figure 4.1: Login Page

This is the root route for the application's frontend if user is not authenticated, if user is authenticated then this route is replaced by dashboard route.

This page has email and password login as well as the g-mail login. If user is not signed up then login fails. Firebase authentication is used for both g-mail and password authentication and g-mail authentication. Firebase authentication makes it easy to implement authentication by just calling login and signup functions.

If user is not registered in the app, then it has to signup first before performing login. Sign up is the creation of an online account using an e-mail address and password or g-mail account.

It simply creates a new account. When you wish to access application for the very first time, you need to sign up.

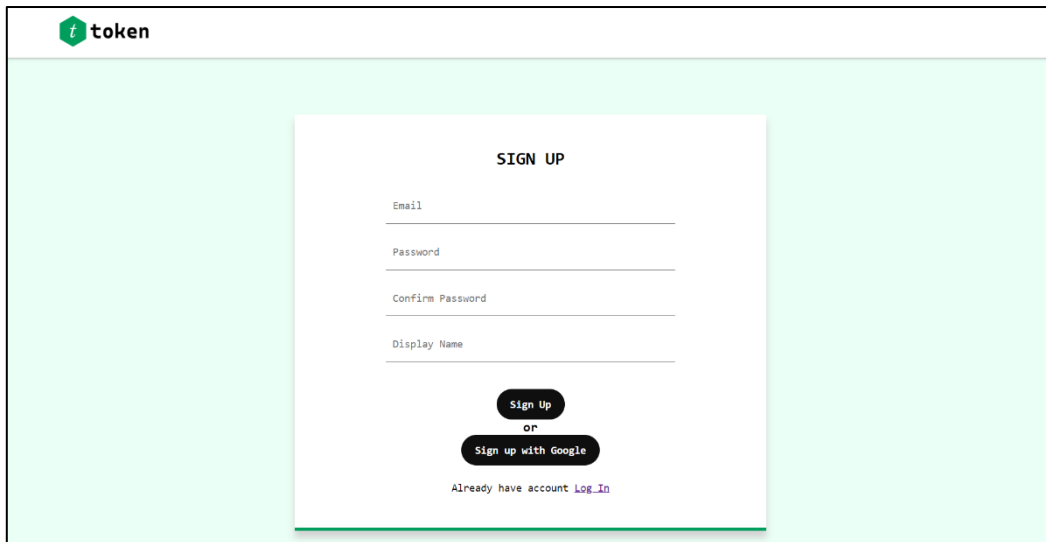


Figure 4.2: Signup Page

You only need to enter your essential information like username, email ID or password to authenticate yourself as a registered user. When user registers first time it is registered with role of customer by default, but if user wants to change the role from customer to business, then it can be done from the drop-down menu after signing in.

This is the /signup route for the application's frontend if user is not authenticated, if user is authenticated then this route is replaced by dashboard route. It has email and password signup as well as the g-mail signup. If user already exists, then signed fails.

Firebase authentication is used for both g-mail and password authentication and g-mail authentication. Firebase authentication makes it easy to implement authentication by just calling login and signup functions.

4.2 Dashboard

Dashboard is usually the one page that the users see first thing in the web application. It is the page that shows the analysis of the application's data, trends, summaries etc. In many cases it dynamically reports important pieces of data from the web application. From the Dashboard the users can drill down to get more information about a particular piece of data.

4.2.1 Customer Dashboard

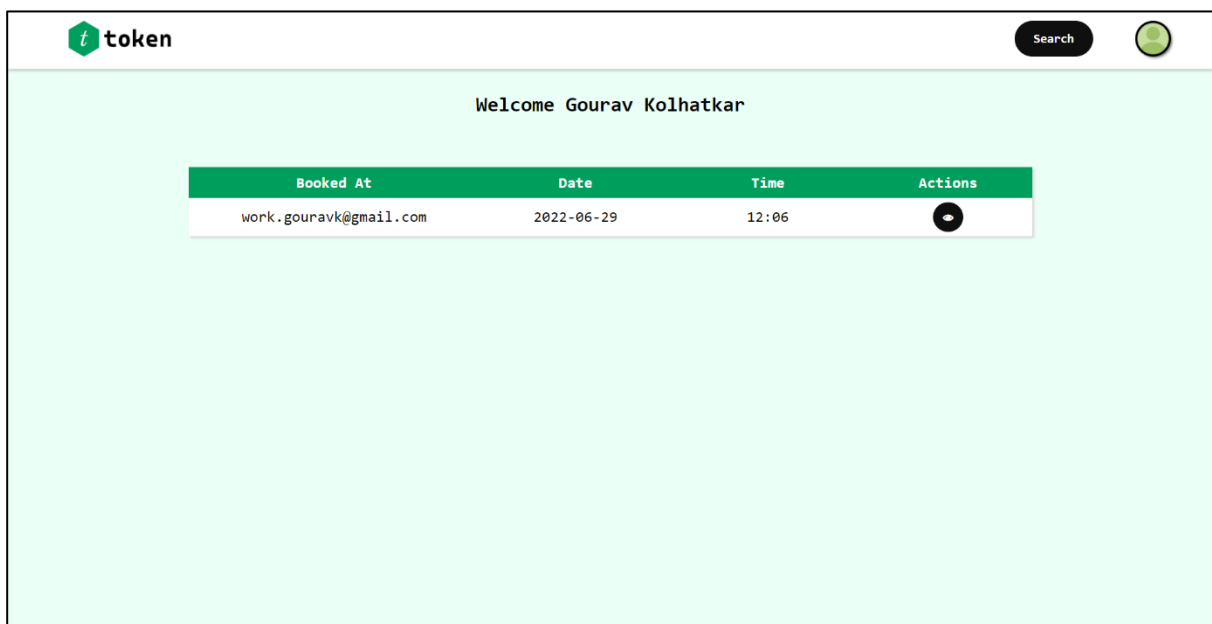


Figure 4.3: Customer Dashboard

It is the page opened once user login in application, here customer can see all its booked slots with other information such as where slot is booked, at what time and on which date.

Customer can also cancel or view the booking details in the form of receipt and can navigate to anywhere in the application from this place.

4.2.1 Business Dashboard

It is the page opened once user login in application if its role is business, here business can see all its booked and un-booked slots and Business can add, delete, edit, and see all its slots created and can navigate to anywhere in the application from this place.

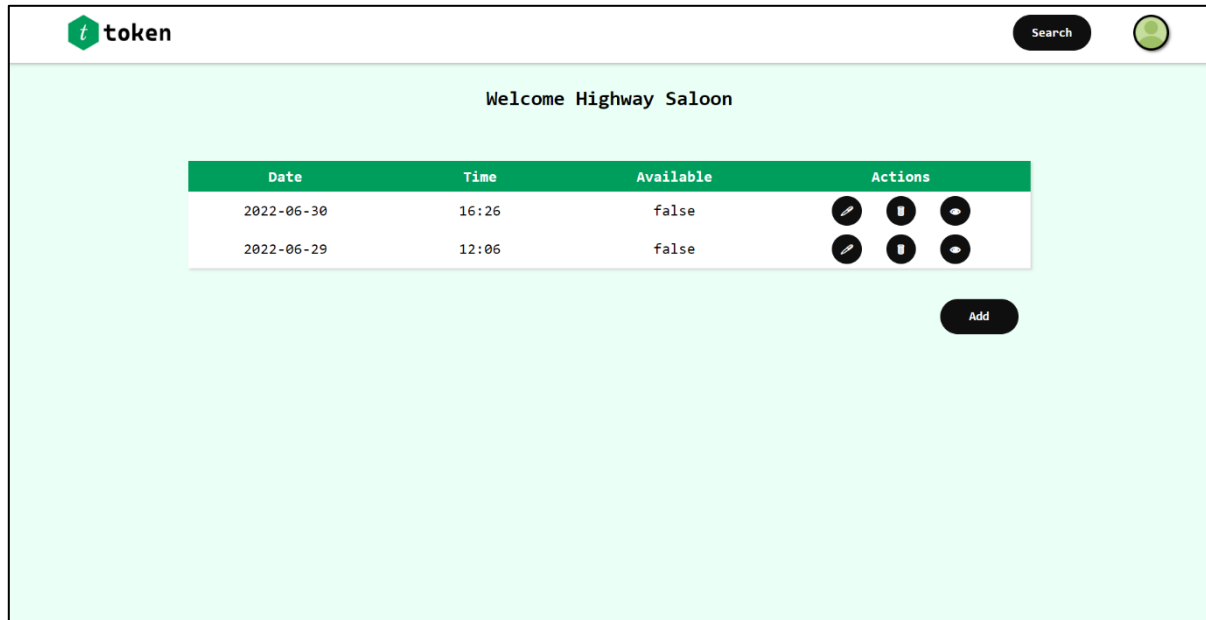


Figure 4.4: Business Dashboard

Business can also view the booking details in the form of receipt and can navigate to anywhere in the application from this place.

Business has addition action buttons as compared to customers for editing and updating the added slots. There is also the Add button on which when clicked businesses can add new slots with date, time and availability.

4.3 Menu

On clicking the top right profile icon, a dropdown menu appears which contains settings and logout button.

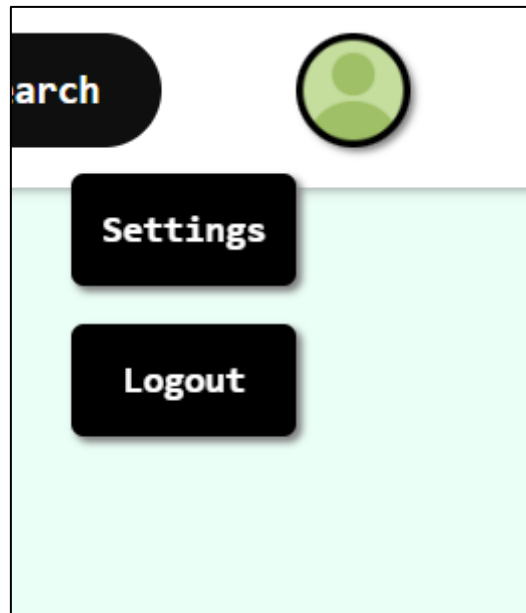


Figure 4.5: Dropdown Menu

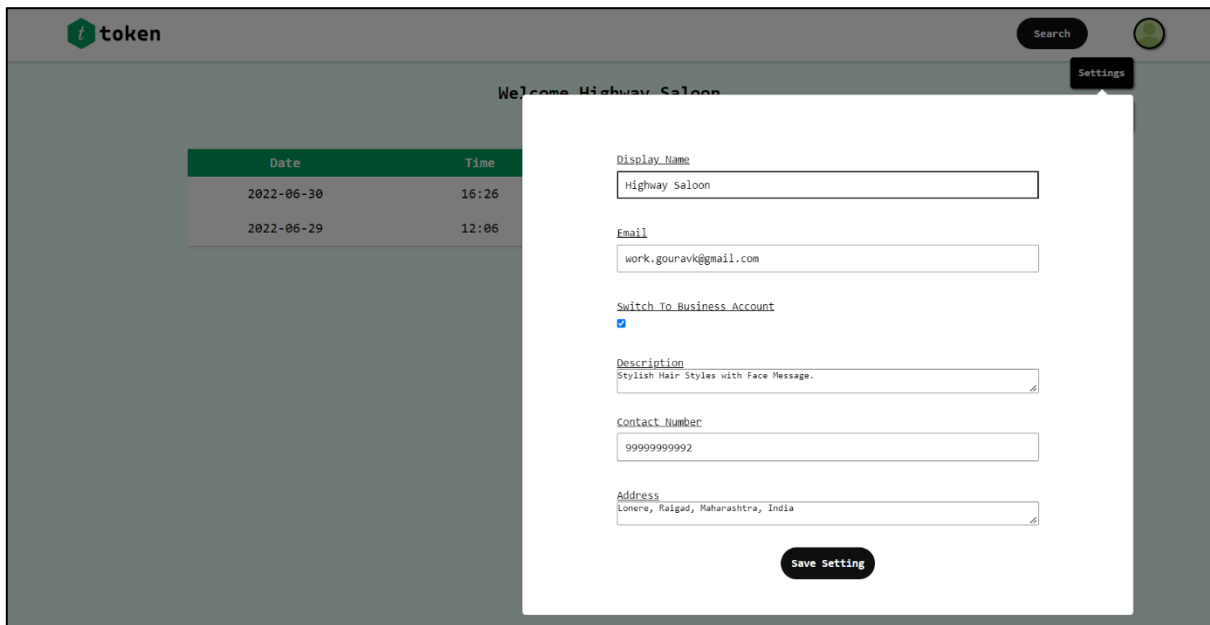
Each button is a react component with black colour and when hovered changes colour to green. After clicking on settings button, a setting dialog box appears which is made using reactjs-popup package.

When the logout button is clicked firebase authentication logout function is called which changes the authentication state from logged in to logged out.

These buttons are specific for authenticated users, they are not visible when user is logged out. They are part of header and have state of visibility dependent on state of authentication of user.

4.3.1 Settings

When settings button is clicked a setting dialog box appears which contains user details and a checkbox to switch user roles from customer to business and vice versa.



The screenshot shows a web application interface with a header bar containing the 'token' logo, a search button, and a settings button. A settings dialog box is open, displaying the following fields:

Date	Time
2022-06-30	16:26
2022-06-29	12:06

Display Name: Highway Saloon

Email: work.gouravk@gmail.com

Switch To Business Account: ☒

Description: Stylish Hair Styles with Face Massage.

Contact Number: 9999999992

Address: Lonere, Raigad, Maharashtra, India

Save Setting

Figure 4.6: Settings Dialog box

Using the settings customer or business can change their profile details such as Display name and email. If user wants to switch roles, then it can do by checking or unchecking the “Switch to Business Account” box.

If user is customer, then it has to check the box to change profile to business and if user is business then it has to uncheck the box to change profile to customer.

If user is business, then three additional field are enabled for it which are description about the business, contact number of the business and address of business where business is located physically. After clicking on the save settings button all the changes done in settings are saved in database and are reflected on profile.

4.4 Search

On the search screen anyone can search different businesses and organizations and book the slots in them.

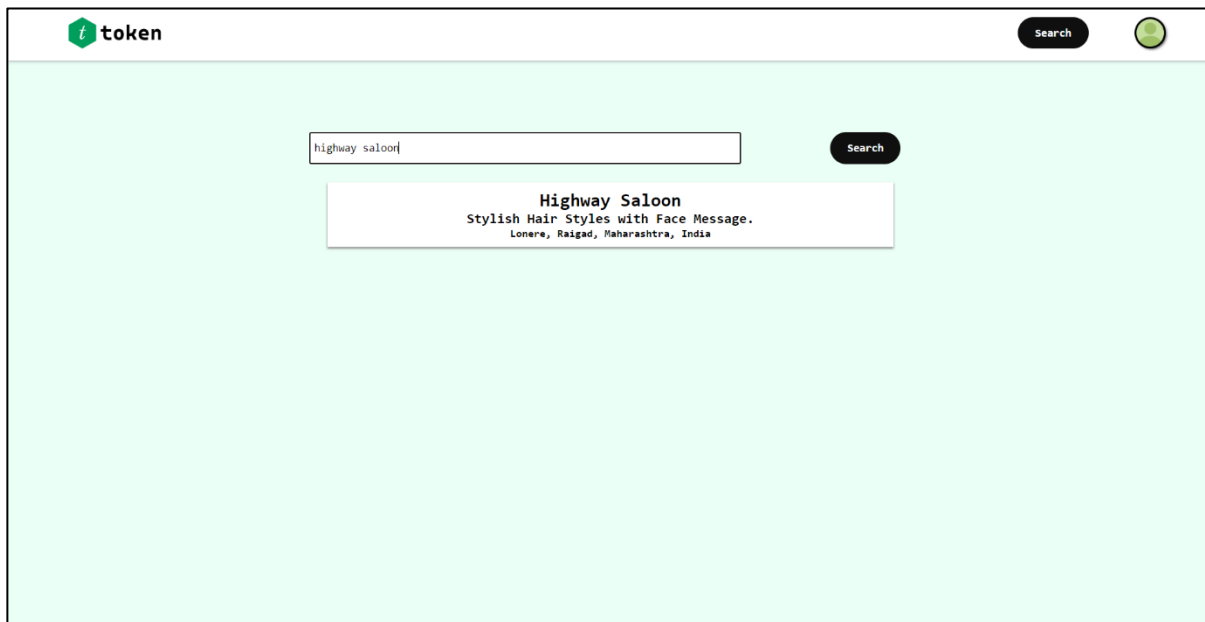


Figure 4.7: Search Page

This page gets rendered when user clicks on the search button on the top of the header. Here all the top businesses are shown on the app.

If user wants to search any businesses on the app by there business name, business description which describes the details about the business and business address which is location of business located physically.

When search term is typed on in the search text field on the search page and clicked on the searched button then all the businesses having the related search terms in its display name, description, and location.

4.5 Booking Terminal

In the booking terminal customer can book slots and then customer and respective business gets a confirmation mail about the booking.

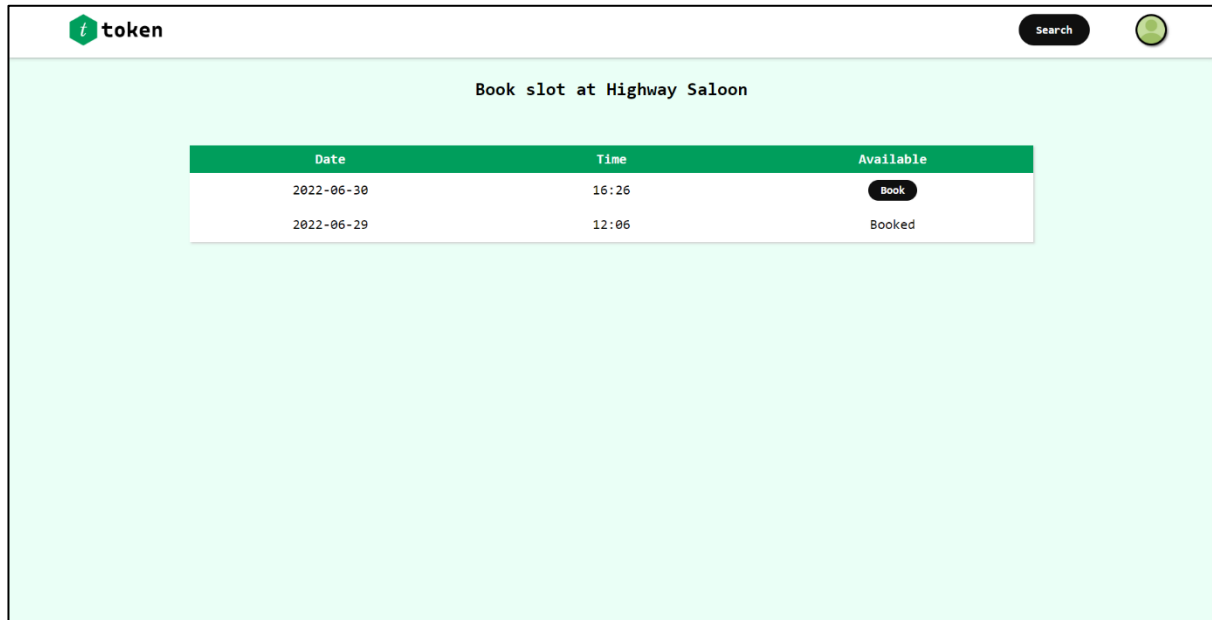


Figure 4.8: Slot Booking Page

Chapter 5

Tech-Stack

5.1 Frontend

Our app frontend is made up of React and CSS. It includes Web pages such as Authentication pages, Dashboard, Settings, Search, Logo, Buttons. We chose client-side authentication using Firebase-client API to authenticate the user using email and password or g-mail account. It represents View from MVC architecture.

Following are the frontend routes -

- login - /
- signup - /signup
- dashboard - /dashboard
- bookslots - /u/[user_id]
- search - /search

Front-end means part of a website that the user interacts with directly is termed the front end. It is also referred to as the ‘client side’ of the application. It includes everything that users experience directly: text colours and styles, images, graphs and tables, buttons, colours, and navigation menu. HTML, CSS, and JavaScript are the languages used for Front End development. The structure, design, behaviour, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers. Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e. it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

5.1.1 React

React is a free and open-source front-end JavaScript library for building user interfaces or UI components.



We used it to because it has Reusability and treats DOM as Components or in other words as object which can be called anywhere and can be used. It has another derived library called React Native which can be used for creating cross platform apps from a single code base which is can help to further update our app for multiple platform.

ReactJS tutorial provides basic and advanced concepts of ReactJS. Currently, ReactJS is one of the most popular JavaScript front-end libraries which has a strong foundation and a large community.

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

Our ReactJS tutorial includes all the topics which help to learn ReactJS. These are ReactJS Introduction, ReactJS Features, ReactJS Installation, Pros and Cons of ReactJS, ReactJS JSX, ReactJS Components, ReactJS State, ReactJS Props, ReactJS Forms, ReactJS Events, ReactJS Animation and many more.

The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. It uses virtual DOM (JavaScript object), which improves the performance of the app. The JavaScript virtual DOM is faster than the regular DOM. We can use ReactJS on the client and server-side as well as with other frameworks. It uses component and data patterns that improve readability and helps to maintain larger apps.

5.1.2 CSS

CSS is used for defining the styles for web pages, to describe the look and formatting of a document which is written in a HTML.



Following are the few frontend details used in app -

- Theme colour - #009E5C
- Background - #EAEFF6
- Buttons and Text - #0F0F0F
- Button Text Colour - #000000
- Dialog Box Background - #000000
- Shadow colour - #BEBEBE
- Font Family – monospace

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

CSS is among the core languages of the open web and is standardized across Web browsers according to W3C specifications. Previously, development of various parts of CSS specification was done synchronously, which allowed versioning of the latest recommendations. You might have heard about CSS1, CSS2.1, CSS3. However, CSS4 has never become an official version.

From CSS3, the scope of the specification increased significantly and the progress on different CSS modules started to differ so much, that it became more effective to develop and release recommendations separately per module. Instead of versioning the CSS specification, W3C now periodically takes a snapshot of the latest stable state of the CSS specification

5.2 API

5.2.1 Node.js and Express.js

Express.js, or simply Express, is a backend web application framework for Node.js, released as free and open-source software for building the applications which are server side, event-driven and made using JavaScript.



We used Express.js to quickly create our API which is a bridge between our Cloud, Backend and Frontend. It represents Models and Controllers from MVC architecture.

Following are the API routes -

- slots
 - create slot (post) - /api/v1/slots
 - read slots (post) - /api/v1/get-slots
 - update slot (put) - /api/v1/put-slot
 - delete slot (delete) - /api/v1/delete-slot
- users
 - create user (post) - /api/v1/users
 - update user role (put) - /api/v1/users
 - read user role (post) - /api/v1/get-role
- businesses
 - read businesses (get) - /get-businesses
 - read businesses by id (post) - /get-businesses-by-id

Node.js is an open source and cross-platform runtime environment for executing JavaScript code outside of a browser. You need to remember that NodeJS is not a framework and it's not a programming language. Most of the people are confused and understand it's a framework or a programming language. We often use Node.js for building back-end services like APIs like Web App or Mobile App. It's used in production by large companies such as Paypal, Uber, Netflix, Walmart and so on.

Express is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middle ware and routing. It adds helpful utilities to Node.js's HTTP objects. It facilitates the rendering of dynamic HTTP objects.

5.3 Backend

Backend is the server-side of the website. It stores and arranges data, and also makes sure everything on the client-side of the website works fine. It is the part of the website that you cannot see and interact with. It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application.

5.3.1 Firebase

Firebase is Platform as A Service (PaaS) which provides cloud services such as Authentication, Database, Hosting, Analytics, etc.



Google Firebase is a Google-backed application development software that enables developers to develop iOS, Android and Web apps. Firebase provides tools for tracking analytics, reporting and fixing app crashes, creating marketing and product experiment.

Firebase offers a number of services, including:

- **Analytics** – Google Analytics for Firebase offers free, unlimited reporting on as many as 500 separate events. Analytics presents data about user behaviour in iOS and Android apps, enabling better decision-making about improving performance and app marketing.
- **Authentication** – Firebase Authentication makes it easy for developers to build secure authentication systems and enhances the sign-in and onboarding experience for users. This feature offers a complete identity solution, supporting email and password accounts, phone auth, as well as Google, Facebook, GitHub, Twitter login and more.
- **Cloud messaging** – Firebase Cloud Messaging (FCM) is a cross-platform messaging tool that lets companies reliably receive and deliver messages on iOS, Android and the web at no cost.
- **Realtime database** – the Firebase Realtime Database is a cloud-hosted NoSQL database that enables data to be stored and synced between users in real time. The data is synced across all clients in real time and is still available when an app goes offline.
- **Crashlytics** – Firebase Crashlytics is a real-time crash reporter that helps developers track, prioritize and fix stability issues that reduce the quality of their apps. With crashlytics, developers spend less time organizing and troubleshooting crashes and more time building features for their apps.
- **Performance** – Firebase Performance Monitoring service gives developers insight into the performance characteristics of their iOS and Android apps to help them determine where and when the performance of their apps can be improved.
- **Test lab** – Firebase Test Lab is a cloud-based app-testing infrastructure. With one operation, developers can test their iOS or Android apps across a variety of devices and device configurations. They can see the results, including videos, screenshots and logs, in the Firebase console.

5.3.2 Firebase-Auth

Firebase Authentication aims to make building secure authentication systems easy, while improving the sign-in and onboarding experience for end users. It provides an end-to-end identity solution,

supporting email and password accounts, phone auth, and Google, Twitter, Facebook, and GitHub login, and more.

5.3.3 Cloud Firestore

Cloud Firestore is a NoSQL document database that lets you easily store, sync, and query data for your mobile and web apps - at global scale.

Chapter 6

Implementation

6.1 Code Snippets

```
const { db } = require("../model/db");
const admin = require("../firebase-service")

const createBusiness = async (data) => {
  const userSlotsCollection = await db.collection("users-slots").where("uid", "==", `${data.uid}`).get();
  userSlotsCollection.docs[0].ref.update({ businessData: data.businessData })
}

const readBusinesses = async () => {
  const users = []
  const uids = await db.collection("users-slots").where("isCustomer", "==", false).get();

  for (const doc of uids.docs) {
    const auth = admin.auth();
    const user = await auth.getUser(doc.data().uid);
    const businessData = doc.data().businessData
    users.push({
      uid: user.uid,
      displayName: user.displayName,
      email: user.email,
      businessData: businessData
    })
  }
  return users;
}

const readBusinessesByUid = async (uid) => {
  const users = []
  const auth = admin.auth();
  const user = await auth.getUser(uid);
  const businessDoc = await db.collection("users-slots")
    .where("uid", "==", uid).get();

  if (businessDoc.docs.length !== 0) {
    const businessData = businessDoc.docs[0].data().businessData
    users.push({
      uid: user.uid,
      displayName: user.displayName,
      email: user.email,
      businessData: businessData
    })
  }
  return users;
}

const readAllBookingData = async (data) => {
  const colName = data.isCustomer ? "booked-at" : "booked-by";
  const userSlotsCollection = await db.collection("users-slots").where("uid", "==", `${data.uid}`).get();
  const slots = {}
  if (userSlotsCollection.docs.length !== 0) {
    const slotsDocRef = await userSlotsCollection.docs[0].ref.collection(colName).get()
    slotsDocRef.docs.map(doc => {
      slots[doc.id] = doc.data()
    })
  }
  return slots
}
```

```

const addBookingData = async (uid, collectionName, userData, slotData) => {
  const slots = await db
    .collection("users-slots")
    .where("uid", "==", `${uid}`).get();

  const docRef = await slots.docs[0].ref
    .collection(collectionName)
    .add({ ...userData, data: slotData });

  const get = await docRef.get();
  const res = {
    docId: get.id,
    ...get.data()
  }
  return res
}

const updateBookingData = async (uid, collectionName, userData, slotData) => {
  const slots = await db
    .collection("users-slots")
    .where("uid", "==", `${uid}`).get();

  const dRef = await slots.docs[0].ref
    .collection(collectionName)
    .get()

  await dRef.docs[0].ref.update({ ...userData, data: slotData });
}

```

Above code handles business logic in the firestore database in firebase such as creating the collections and documents for business, reading all the business data at one time, reading a business by its UID, adding the new businesses and updating the data related to businesses.

createBusiness - This is an asynchronous javascript function which creates business data such as display name, business description, business address and phone number in the “user-slots” collection using the user uid.

readBusinessesByUid - This is an asynchronous javascript function which reads business data such as display name, business description, business address and phone number from the “user-slots” collection using the user uid.

readBusinesses - This is an asynchronous javascript function which reads all the businesses present in the firestore database in the firebase.

readAllBookingData - This is an asynchronous javascript function which reads all the booking data of customers and slots booked and available of businesses from the firestore database in the firebase.

addBookingData - This is an asynchronous javascript function which adds the booking data of customers and slots booked and available of businesses in the firestore database in the firebase.

updateBookingData - This is an asynchronous javascript function which updates the booking data of customers and slots booked and available of businesses in the firestore database in the firebase.

```
const { db } = require("../model/db");
const admin = require("firebase-admin");

const createUser = async (uid, isCustomer) => {
  const users = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  if (users.empty) {
    console.log("user created", isCustomer);
    const docRef = await db.collection("users-slots").add({ uid, isCustomer });
    const get = await docRef.get()
    const data = get.data();
    return data;
  } else {
    return {};
  }
}

const updateUserRole = async (uid, isCustomer) => {
  const users = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  users.forEach(doc => {
    doc.ref.update({ "isCustomer": isCustomer })
  })
}
```

```
const readUserRole = async (uid) => {
  const users = await db.collection("users-slots").where("uid", "==", uid).get();
  const dc = {
    data: ""
  }
  users.forEach(doc => {
    dc.data = doc.data();
  })
  return dc.data.isCustomer;
}

const readUserData = async (uid) => {
  const auth = admin.auth();
  const user = await auth.getUser(uid);
  const userSlot = await db.collection("users-slots").where("uid", "==", uid).get();
  const userData = userSlot.docs[0].data()
  const data = { uid: uid, ...userData }
  data["displayName"] = user.displayName;
  data["email"] = user.email;
  return data;
}
```

Above code handles User Data in the firestore database in firebase such as creating the collections and documents for users, reading all the users data by it's UID, reading and updating user role, adding the new users and updating the data related to businesses.

`createUser` - Creates new user in the firestore database in firebase with user data such as default user role, user display name, user email and encrypted password.

`updateUserRole` - updates default user role or current user role from customer to business or business to customer or vice versa. If user first time changes from customer to business then they have to business description, business address and business phone number.

`readUserRole` - Reads user role and data related to user role from firestore database in the firebase. If user is customer then its booking details are shown and if user is business then its slots details are shown with other features to add, modify and delete slots.

`readUserData` - Reads user related data from firestore database and firebase authentication in the firebase. If user is customer then its booking details are shown along with display name and email, and if the user is business then its slots details, other features to add, modify and delete slots are shown along with business description, business address and business phone number.

```

const { db } = require("../model/db");

const createSlot = async (date, time, isAvailable, uid) => {
  const usersSlots = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  const slotsDocRef = usersSlots.docs[0].ref.collection("slots").add({ date, time, isAvailable });
  const get = await (await slotsDocRef).get();
  const data = await get.data();
  const response = {};
  response[get.id] = data;
  return response;
}

const readAllSlots = async (uid) => {
  const userSlotsCollection = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  const slots = {};
  if (userSlotsCollection.docs.length !== 0) {
    const slotsDocRef = await userSlotsCollection.docs[0].ref.collection("slots").get();
    slotsDocRef.docs.map(doc => {
      slots[doc.id] = doc.data()
    })
  }
  return slots
}

const updateSlot = async (uid, slotId, data) => {
  const slots = await readAllSlots(uid);
  const userSlotsCollection = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  await userSlotsCollection.docs[0].ref.collection("slots").doc(slotId).update(data);
  return slots;
}

const deleteSlot = async (uid, slotId) => {
  const slots = await readAllSlots(uid);
  const userSlotsCollection = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  await userSlotsCollection.docs[0].ref.collection("slots").doc(slotId).delete();
  return slots;
}

const deleteBookedSlot = async (uid, slotId, colName) => {
  const userSlotsCollection = await db.collection("users-slots").where("uid", "==", `${uid}`).get();
  await userSlotsCollection.docs[0].ref.collection(colName).doc(slotId).delete();
  return { "success": true };
}

```

Above code handles Slots Data in the firestore database in firebase such as creating the collections and documents for Slots, reading all the slots data by UIDs, reading all slots at a time, reading slots by UID, updating slots data, adding the new slots, updating the slots data, deleting slots and deleting the booked slots by customers.

createSlot - Creates new slots in the firestore database in firebase with slot data such as slot date, slot time, availability for the businesses, slot Id and User Id.

readAllSlots - Reads all the slots data of the businesses by its UID such as slot date, slot time, availability for the businesses, slot Id and User Id.

updateSlot - Updates the slots data of the businesses by its UID such as slot date, slot time, availability for the businesses, slot Id and User Id.

deleteSlot - Deletes the slots from the firestore database in firebase of the business by its UID from the slots document.

deleteBookedSlot - Cancels the slots from the firestore database in firebase of the business by its UID from the booked-by document of Businesses and booked-at document of the customers.

```

import React, { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import { Route, Routes, Navigate } from 'react-router-dom';
import { Logo, SearchAndProfile } from './components';
import { Dashboard, Login, Signup, BookSlots } from './screens';
import { useDispatch } from 'react-redux';
import { isLoggedIn } from './actions/isLoggedIn';
import { getAuth } from './api/auth';
import { onAuthStateChanged } from 'firebase/auth';
import LoadingOverlay from 'react-loading-overlay';
import './styles/App.css';
import Search from './screens/Search';

function App() {

  const [isCustomer, setIsCustomer] = useState(true);
  const authUser = useSelector(state => state.userReducer)
  const loading = useSelector(state => state.loadingReducer)
  const dispatch = useDispatch();
  LoadingOverlay.propTypes = undefined

  useEffect(() => {
    onAuthStateChanged(getAuth(), (user) => {
      dispatch(isLoggedIn(user))
    })
  }, [authUser]);

  return (
    <LoadingOverlay
      active={loading}
      spinner={true}
      text="Loading..."
      styles={{ content: "" }}
    >
      <React.Fragment>
        <header>
          <Logo/>
          {
            authUser
              ? <SearchAndProfile isCustomer={isCustomer} setIsCustomer={setIsCustomer} />
              : ""
          }
        </header>
        <Routes>
          <Route path="/u/:uid" element={BookSlots} />
          <Route path="/search" element={Search} />
          {ProtectedRoute(!authUser, "/", <Login />, "/dashboard")}
          {ProtectedRoute(!authUser, "/signup", <Signup />, "/dashboard")}
          {ProtectedRoute(authUser, "/dashboard", <Dashboard isCustomer={isCustomer} setIsCustomer={setIsCustomer} />, "/")}
        </Routes>
      </React.Fragment></LoadingOverlay>
    );
  }

  const ProtectedRoute = (protector, path, component, to) => {
    return (
      <Route path={path} element={
        protector ? component : <Navigate to={to} replace={true} />
      } />
    )
  }

  export default App;

```

Above code is the entry point for the frontend of the app. It defines different routes for the front as give below and which page to render when a particular route is called, it also contains the protected routes which can only be accessed if the user is authenticated such as dashboard or booking terminal.

Following are the frontend routes -

- / (login) -

It renders the login authentication page with email and password or g-mail account after successful login user gets redirected to dashboard for respective user roles such as customer or business.

- /signup (signup) -

It renders the login authentication page with email and password or g-mail account after successful login user gets redirected to dashboard.

- /dashboard (dashboard) -

After successful authentication user gets navigate to dashboard route. It renders the dashboard for respective user roles such as customer or business. Using dashboard, a user can see booked slots or cancel the booked slots. It can also navigate to search and profile settings.

- /search (search) -

From the dashboard or without authentication user can navigate to search route to see and search different businesses on the app along with display name, business description and business address. After clicking on the desired or searched business user gets redirected to booking terminal where customers can book the slots in that business.

- /u/[user_id] (bookslots) -

It renders the booking terminal of business using its UID in the query parameter. In the booking terminal customers can see different slots which are available and as well as unavailable. Major use of booking terminal is to book the slots in a particular business.

This code also implements the loading animation which is executed whenever user of the app has to wait for a process to complete.

It also defines which routes are public that can be accessed without authentication and routes which are protected that can't be accessed without authentication.

Using the reusability of React framework it defines a responsive header which include app logo and different public and protected header elements.

Chapter 7

Applications

- **Increase efficiency in customer- business trade**

Booking Slots Help businesses and customers to allocate and book slots respectively without visiting the place also get notified at correct time which saves time unlike waiting in long queues.

- **Business advertisement**

By business registering on application public can know that such facility exist and public can know about it. It can help new start up to advertise with less cost.

- **Medical Emergencies**

If hospitals register on application, then common public can know which hospitals are vacant, saves time for patient by not visiting many hospitals in search of vacancy.

- **Educational Institutions**

Booking admission slots, parents-teachers meeting slots in schools, fees payment related meetings, future education planning slots, exams slots, lectures slots, booking sports equipment for particular slot time, library slots (also we can customise application to book available books in library for required slot period), and all types of slots that educational institutions provide.

- **Ticket counters**

Avoid long waiting in line for longer time in queue (in some random time period) for booking offline tickets, by splitting time periods into slots and let public book them online.

- **Every place with queues**

To avoid wastage of time and energy in waiting in queue

Chapter 8

Future Scope

- Application has various uses, but it can be updated again in future and various features can be added. Some of them are:
- It's a web application made in React.js so it can be converted into multi device support for mobile and desktop using React Native.
- It can be used in future to grow and advertise the business and institutions.
- It can be used globally across the countries for international bookings.
- By adding location update using different APIs such as google maps API, users can book the slots to their nearest facility.
- By connecting with social media people can know what are the shops that their friends are using and can decide which is better.
- Feedbacks and reviews will help users to book proper slots and allows organizations and businesses to improve.

Chapter 9

Bibliography

- Free Code Camp articles
<https://www.freecodecamp.org/news>
- Express.js documentation
<https://expressjs.com/en/4x/api.html>
- Node.js documentation
<https://nodejs.org/en/docs>
- React.js documentation
<https://reactjs.org/docs/getting-started.html>
- Firebase documentation
<https://firebase.google.com/docs>