Bilkent University
Department of Computer Engineering

Senior Design Project

Final Report

Project Name: Nomad

Project Source Code: *https://github.com/gulnihalm/nomad-design*
Project Website: *https://gulnihalm.github.io/nomad-design.github.io/*

Group Members: Gülnihal Muslu

Berk Ataç

Can Ozan Kaş

Ali Kemal Özkan

Tarık Emin Kaplan

Supervisor: Uğur Güdükbay

Jury Members: H. Altay Güvenir, Özcan Öztürk

Innovation Expert: Veysi İşler

Final Design Report

May 27, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1. Introduction

Nowadays, travelling is an easy and usual activity for people[1]. Everybody wants to explore places that they have never been. However, there are a lot of buildings and places or activities to do in a country/city. Therefore, people have to choose the best of them according to their interests because they usually have a limited time to see a city. Deciding which place is worth seeing or which food is worth eating can be tricky in most times due to people usually preferring to explore cities that they have never seen before. There are some applications and websites for that, but they have never been enough for people that are such travel lovers.

To solve that problem, we thought of a project -which name is Nomad- where people can give trip advice to other people. Nomad means a member of a group of people who move from one place to another instead of living in the same place all the time. Our project idea comes from there. In this project, a person can create a route in a city or country and s/he can give advice on that trip. Users will have a chance to select a route from various list of routes that are created by other users and they can follow that path step by step. Since routes can be seen before and they can be scored, interpreted; people can choose their route easily and more close to their dream trip. With this project travels will become easier and more fun.

This report contains the culmination of the project Nomad. The final architecture and design of our system as well as the final status of the project is presented in this report.

2. Requirements Details

2.1. Functional Requirements

- Users can be authenticated with their social media accounts (facebook, twitter, google). After first login with a social media account, users can login with their email and password.

- Every user will have a profile. Name-surname, finished trips synchronized accounts can be seen from their profile.

- Users can search a route by word. This makes finding a trip from a whole trip list, easier.

- Users can create their own routes. Users can place checkpoints wherever they want.

- Users can follow existing routes created by other people and can vote the routes after finishing a route. In each route, there will be checkpoints. In the checkpoint there will be items that are placed by creators. With collecting those items, users will gain tokens that can be used later to create their own routes.

- Routes may have "themes" if users specify them on creation. For example a route with a "relaxation" theme will have stops at parks, coffee shops etc. whereas a route with a "historical" theme will have stops at historical sightseeing points.

- To gamify the app with AR, while creating a route, the users drop tokens on the checkpoints as they are created, and the users travelling these routes will be notified of the existence of such tokens while they are near.

- Information about routes such as distance, checkpoints, ratings, which people followed that route is provided.

- Users can comment on routes after completing them (for example, to ask for an update at a specific location in the route).

2.2. Non-Functional Requirements

Extensibility:

● Nomad should allow adding new functionalities, features, components and so forth.

● The system will be kept up to date.

Availability:

● Nomad is available 24/7 for the users.

● It will be open for every android user who uses android 8 or higher.

System:

● Nomad is an Android application.

● Application requires an Internet Connection and gps to work. And for some cases a camera is also required.

● Minimalistic user interface is designed.

Performance:

● Nomad does not take more than 5 seconds to log in.

● Application does not take more than 10 seconds to get ready after the app icon is clicked.

Scalability:

● Nomad database is available to further expansion.

Security:

● Nomad secures user information from any possible threats.

Reliability:

● Nomad is stable and crash free.

Usability:

● Application is easy to use for all user types because anyone who can use a mobile phone is a possible user of this application. Thus, we provide a user friendly and simple user interface.

● Application clearly explains what it aims to do and why users should use this app.

3. Final Architecture and Design Details

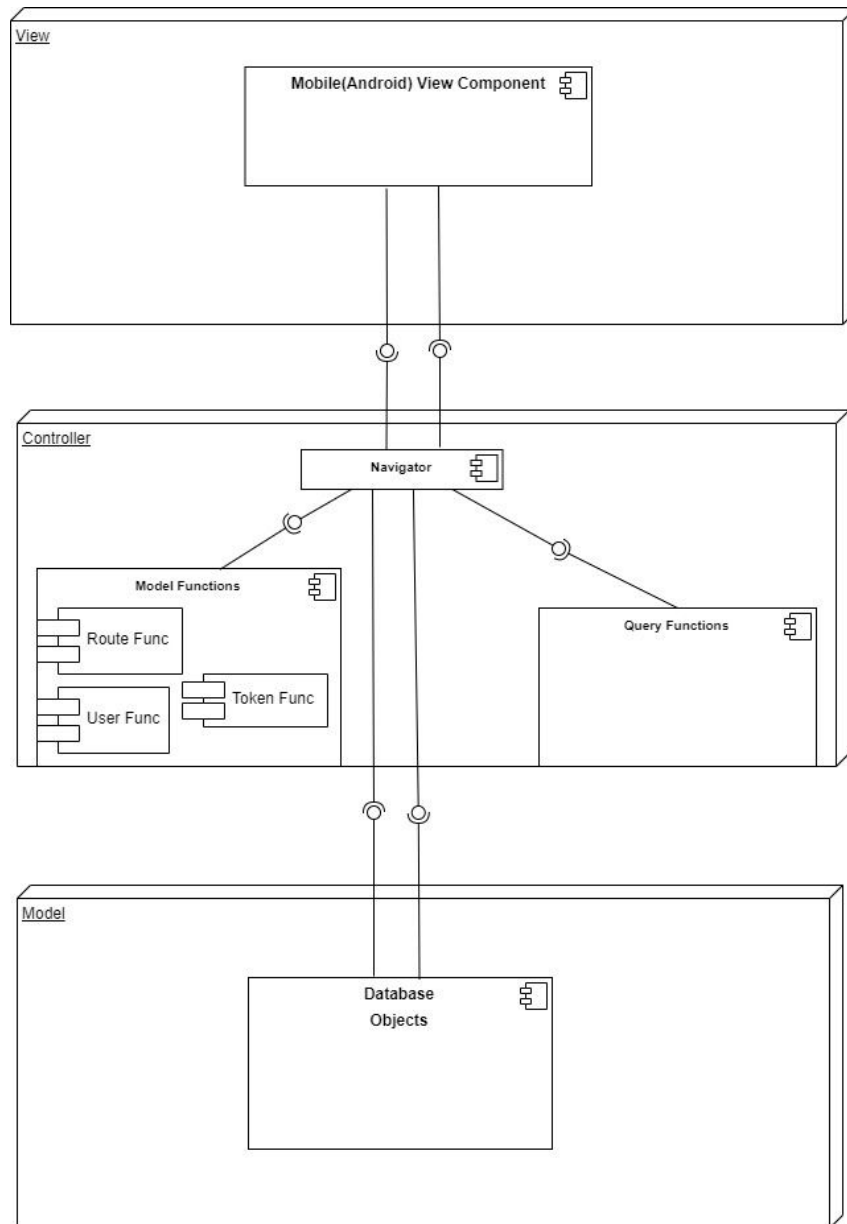## 3.1. Subsystem Decomposition

Figure 1: Subsystems

Our project follows the Model-View-Controller (MVC) architectural pattern. This pattern separates and classifies the software to three main components that are UI elements to be visualized, data to behold and requested, and control elements that will pull/push requests to other components in order to ensure the data flow. The MVC pattern provides

sufficient and genuine features and it is in a harmony with the design decisions made by our team.

In Controller, Model functions serve as a beholder of the application's database entries. It allows queries run by the controller package to access the related user data. Navigator serves as a controller for UI components. Query Functions are database functions, insert, delete etc. Our application Nomad will run on Android devices, in View, we have a mobile view component for our client(android). In Model, Database objects are information(data) we hold in our database, which is provided by our Controller and viewed by our View component.

3.2. Hardware/Software Mapping
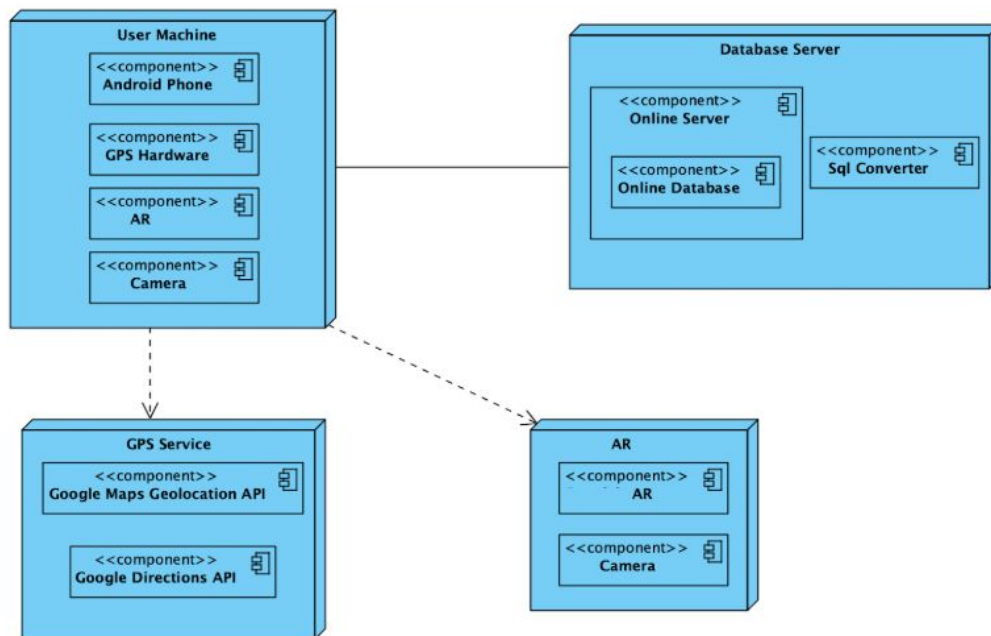
Figure 2: Hardware/Software Diagram

Nomad initially runs on android based platforms. It uses the camera and GPS of the smart-phone. Account information of the users will be stored on an online server with SQL database management system. Thus, it requires internet connection. Client-server architecture is used for this purpose. Client-side communicates with the end-users via Nomad UI. It also handles location detection with GPS. Data management operations are handled in the

server-side hosting the application. Client-side needs to communicate with the database server to get/set the data. This communication will be provided by HTTP.

3.3. Persistent Data Management

Nomad will store various user information such as the achievements, routes that have been used and liked, profile and account information. For this purpose, there will be a sufficiently large database. Since the database needs to store the routes as coordinates, there will be a huge amount of data. In order a user to follow a route s/he needs to get the coordinates from the database. For the beginning we use a free online database, which is easy for us to use at the beginning.

In order to save the routes created by users and not published yet, the device needs to have enough storage as these routes will be kept in the device until it is published. After the route has finished, the route is saved to the database.

3.4. Access Control and Security

Nomad requires social media sign-up at first place. Users can login with their Facebook, Twitter and Google accounts. This is a very secure way to create an account. Also, with that, we prevent spam/bot accounts.

After login with social media accounts users can set a password with some security precautions. Password has to be at least six characters long and must include letter and number. This will prevent from easy passwords that users will define such as "123456" or "abcde". Since social media accounts already have high level protections itself, users will have a chance to secure themselves when using our app.

4. Development/Implementation Details

Our project simply consists of several coordinate records and several functionalities for the user. These functions and coordinates can be used by each user at the viewing stage when the time comes. We needed a

database to keep our records and a server to meet the functionality. React Native cannot access the database by default because the database information must be kept on the device for access, and this creates a security issue. There is a function called "fetch" to provide this feature. This is how we can send a request to a remote server and get a response. Our remote server communicates with our database. In this way, our information is safe. Additionally, we can access our database via phpmyadmin.

React Native uses Javascript and Mobile-native language for its UI elements, so we naturally used those. In addition to that, throughout our application, we do lots of API calls to various services (such as social media logins like facebook, google, twitter and location services like google maps), and also we interact with our database often. React-native interacts with all external services like that through fetch requests. Following is an example of such a fetch request.

```
fetch( hostURL + 'social_login.php', {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            //variables we send to the php file goes here
        })
    })
    .then((res) => res.json())
    .then((res) => {
        //handling of response
    })
    .catch((err) => {
        alert("SC-LG " + err);
        // handling of error
    })
```

In this fetch request, we send a login request related to a social media account to our host (detailed code parts are commented out for this report). In the host, we have our php scripts that we use to connect to our database. As you can see we send and receive our data in JSON format. For our hosting server we used a free online service provider called 000webhost.com and as our database we used a free online database provider remotemysql.com. Our php scripts also have some sql segments in them.
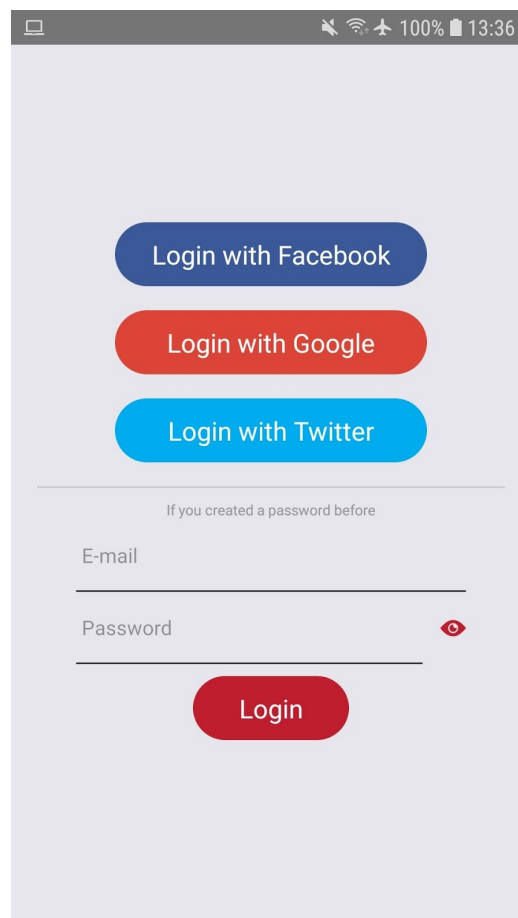
Login Page

Figure 3: Login Page

This is the first page any user sees when they launch our application. Our login is highly functional.

As you can see there are 4 different login options, facebook, google, and twitter logins (throughout the report, we will refer to them as "social

logins" for short), and one for a login with email and password (which we will refer to as email login). A first-time user which just installed our app will have to first use one of the social login options for their first login, and can't login with the email login. Once a social login is done, their account is created and later the user may choose to create a password for their account anytime they want. Later, they can use this password and the email of the social media account (the social login option they chose when they logged in to our app) to use email login. They can also keep logging in with the social login option that they previously chose.

Synchronization

The main functionality that makes our login highly functional is synchronization of different social media accounts in one account. Synchronization brings a lot of additional features along with it, but in this section only the ones related to login will be explained.

The user may choose to synchronize another social media account to their Nomad account in addition to the social login option they chose when they were first logging in to the application. For example, a user that chose google login when first creating their account can later synchronize their facebook account, and after successfully doing so, they can login to our application with both facebook and google logins.

Also, if a user tries to login with a social media account that is associated with an email that has been used to login to our application with another social media option we detect that and prompt the user to synchronize. For example, assume that a user has already logged in to our application using twitter and the email associated with their twitter account is "emin.kaplan@ug.bilkent.edu.tr", then assume that they try to login with their facebook account, which is also associated with the same email. This is detected and as mentioned before, the user is prompted to synchronize their accounts. This case does <u>not</u> imply that synchronization requires both social media accounts that are chosen to be synchronized to be associated with the same email.
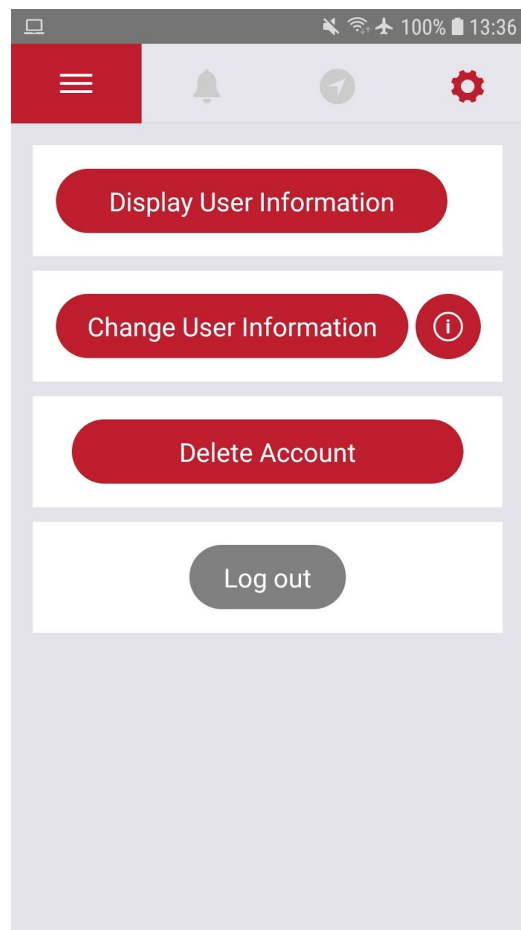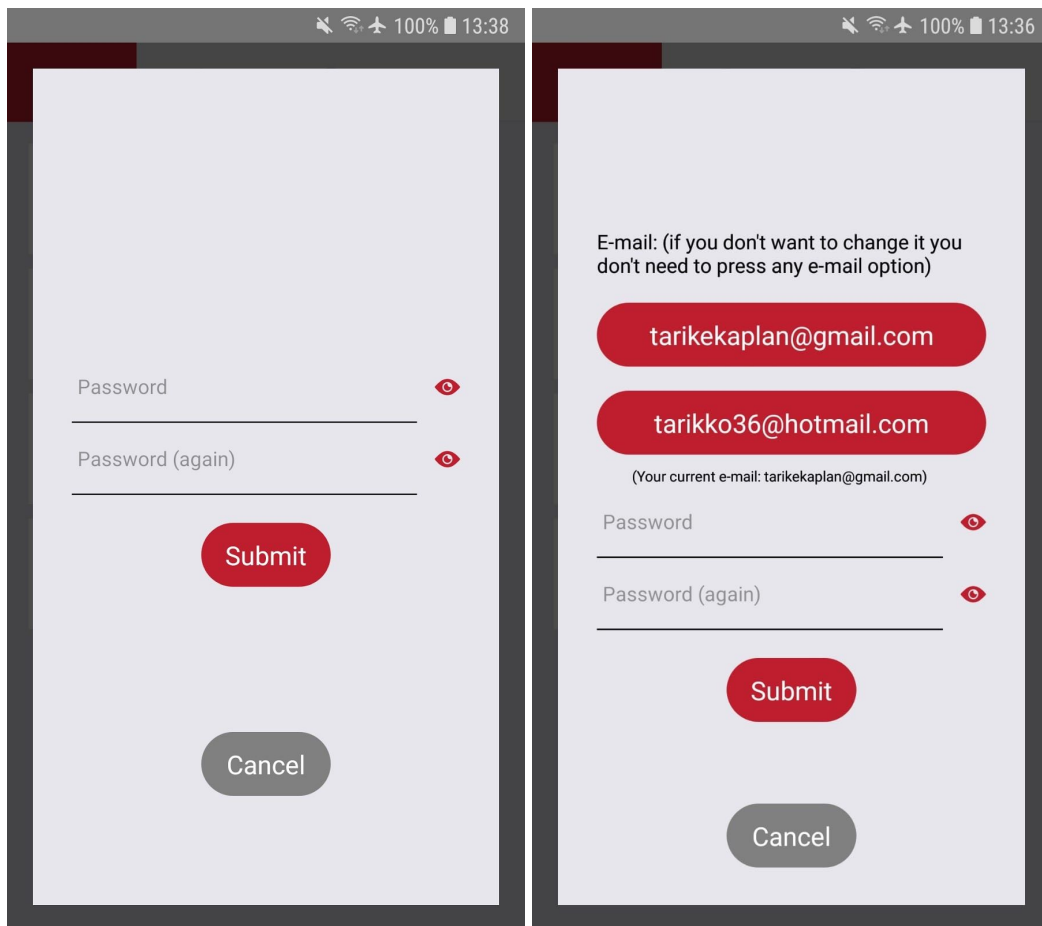
Settings

Figure 4: Settings Page

These are the options available in the settings page. The first one simply just shows all information related to this user's account. The last one simply logs the user out. The second and third one brings pop-up screens called RegisterModal and DeleteModal. Following are the screenshots for them.

RegisterModal

Figure 5: Register Screens

In this pop-up screen the user is prompted to create/change their password. If they have synchronized any social media account that is associated with a different email than the email that is associated with the social media account that they chose to login with, the user is also given a choice to select between those different emails. The chosen email is registered in our database as the main email address that we interact with, if the user has a password related to their account, they need to use that main email address when using email login.
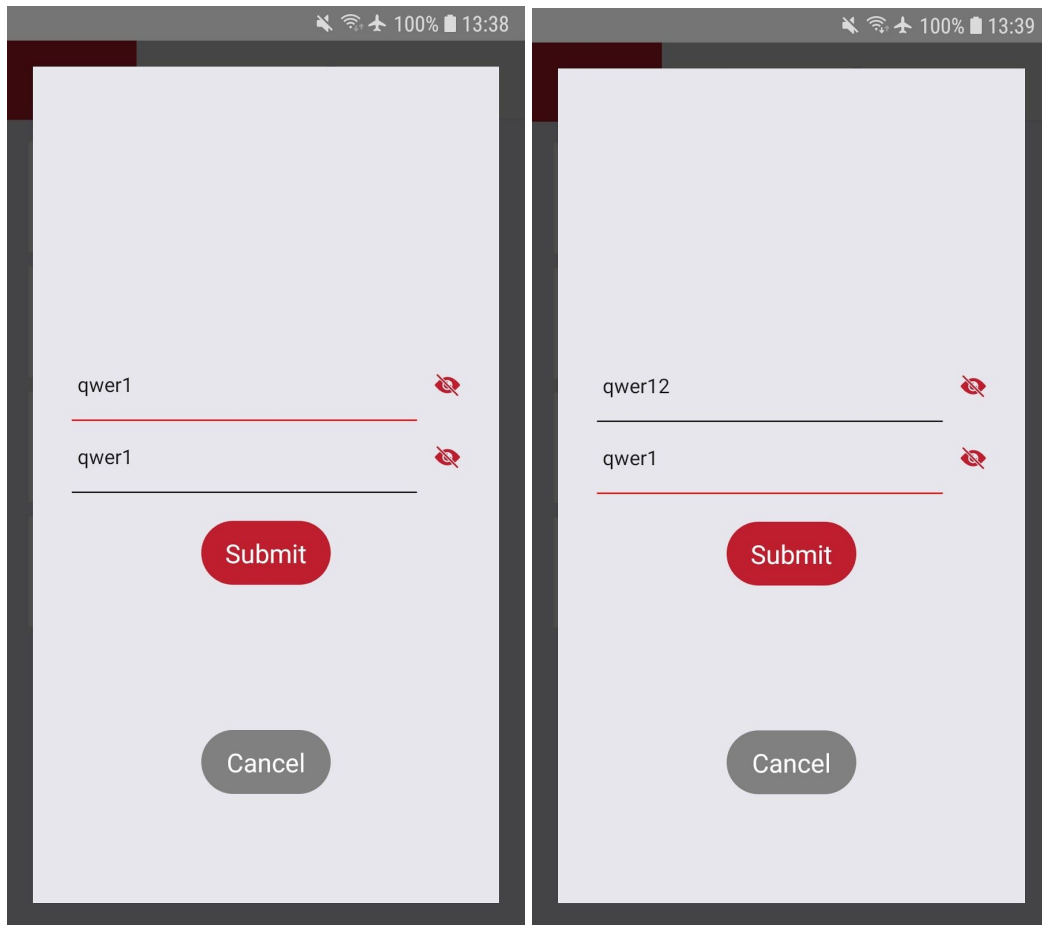
Figure 6: Password Initialization

We have some password requirements and we check the validity of the entered passwords by executing a function after every letter input supplied to the either text inputs in RegisterModal. Visually, a text input's bottom border will turn from black to red to notify the user about the input's invalidity, and it will revert back to black when the input is either empty or valid. The password is valid if the password contains at least 6 characters, at least one of them being a letter and at least one of them being a number. A RegEx (regular expression) is there to check if these conditions are met. The confirmation password is valid if its input is the same as the password's current input.

PasswordTextInput

For password inputs there is no built-in text field component in React Native that hides the input on press of a button, and we didn't find one on the internet that suits our needs exactly so we have made one for ourselves and used throughout the project. In email login and RegisterModal you can see the component, being a different text field, having an eye icon on the side, to optionally show its content.
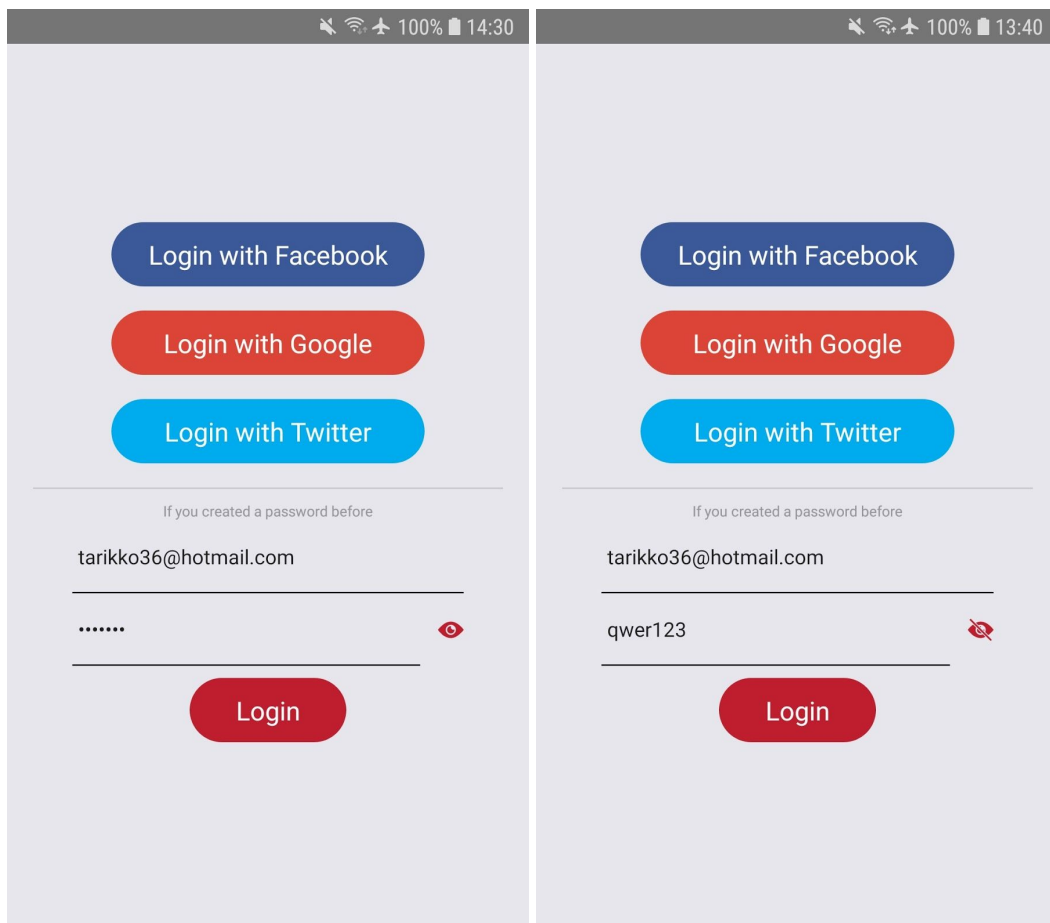
Figure 7: Login with E-mail&Password

DeleteModal
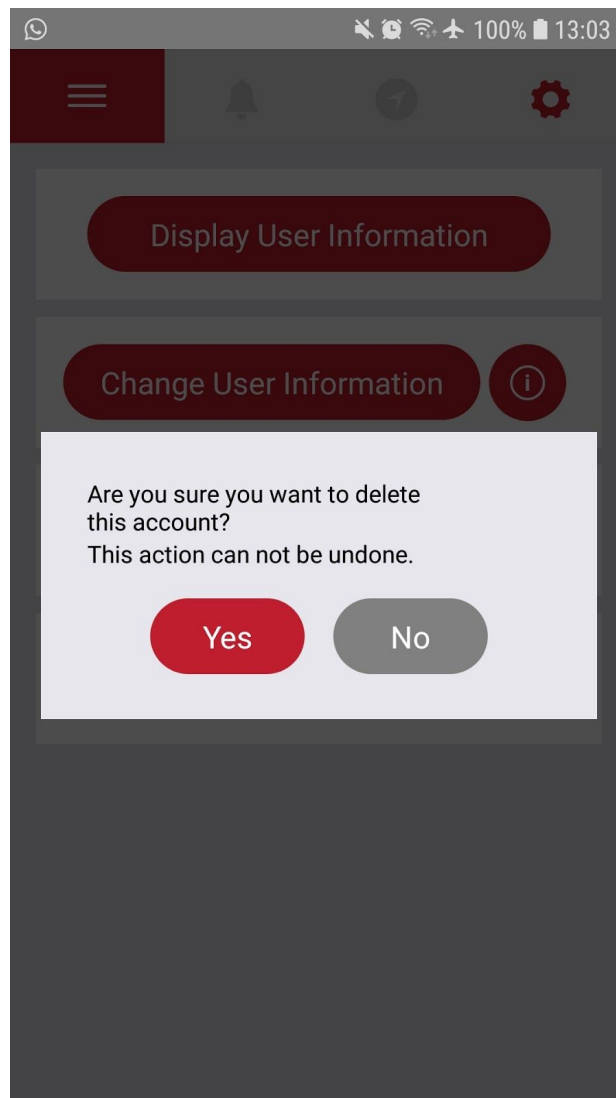
Figure 8: Delete Account

This pop-up is used to simply delete the user's account from our database.

Sidebar

In addition to our main page we have a sidebar which allows us to access some other pages with various functionalities.
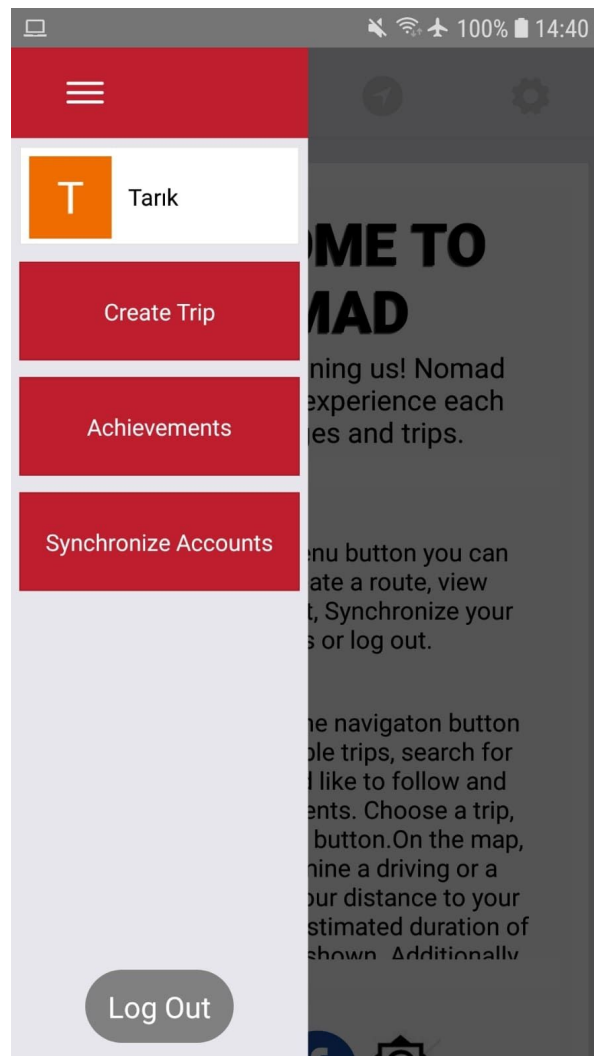
Figure 9: Sidebar

Profile Page
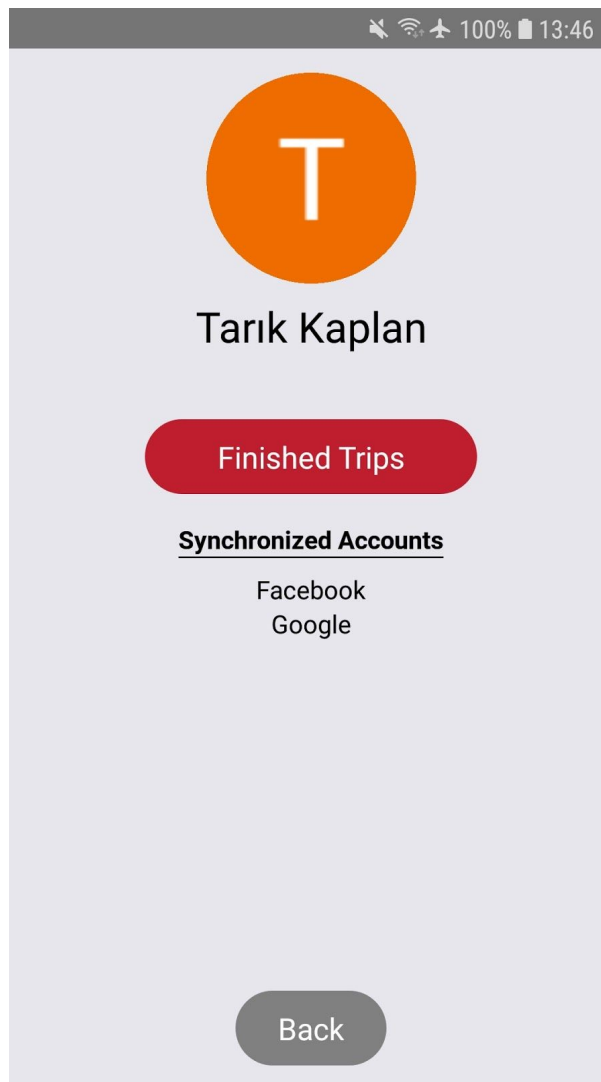
Figure 10: Profile Page

In the profile page you can view all your synchronized social media accounts and access finished trips page.

Finished Trips Page
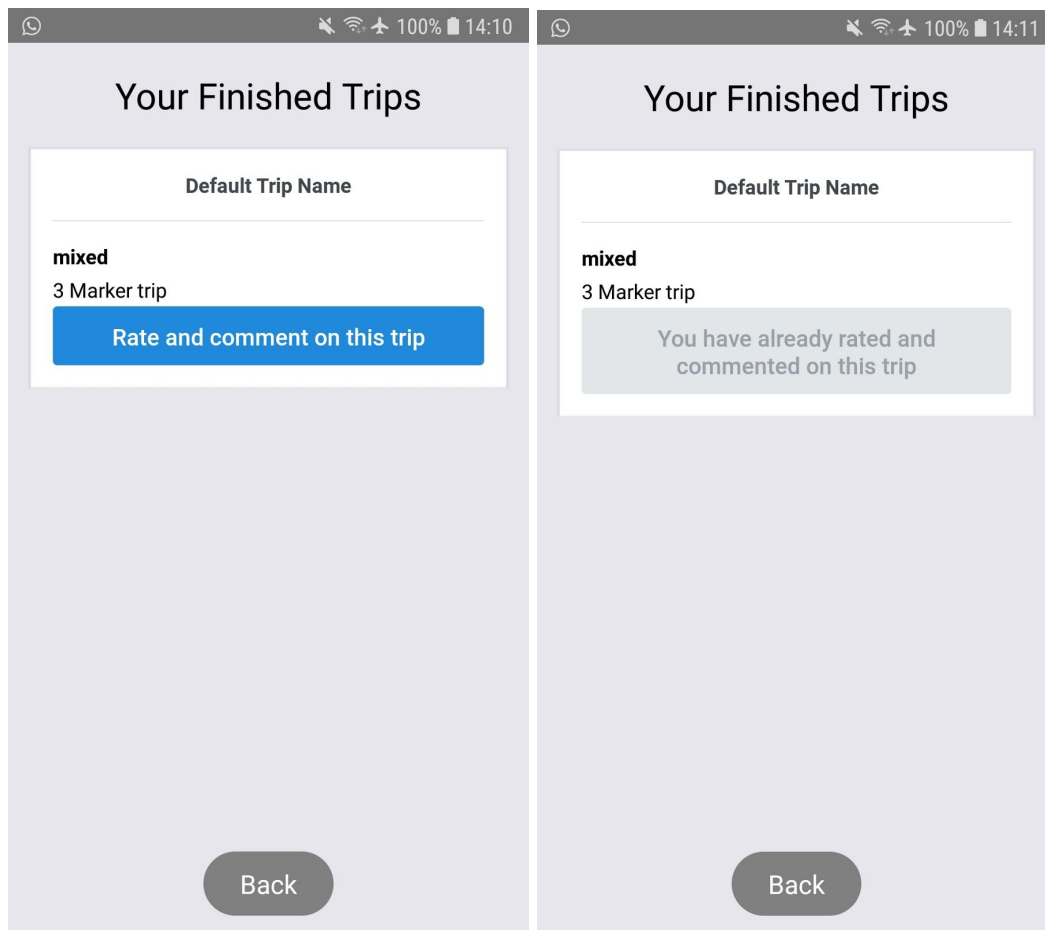
Figure 11: Finished Trips Page

In the finished trips page you can view all the trips you have successfully completed. Additionally, if you want to, you can rate and comment on a trip you have finished. You can't add additional comments once you have submitted your comment and rating on a trip. Unfortunately this also means you cannot change your submitted comment and rating.

Commenting and Rating a Trip

Figure 12: Comment and Rate Page

In this page you can add a comment and a rating out of 5 on a trip you have completed. All the submitted comments and ratings about a particular trip are shown optionally in the main page, when the user taps on the "view comments" button on a trip in the search trips page.

Synchronization Page
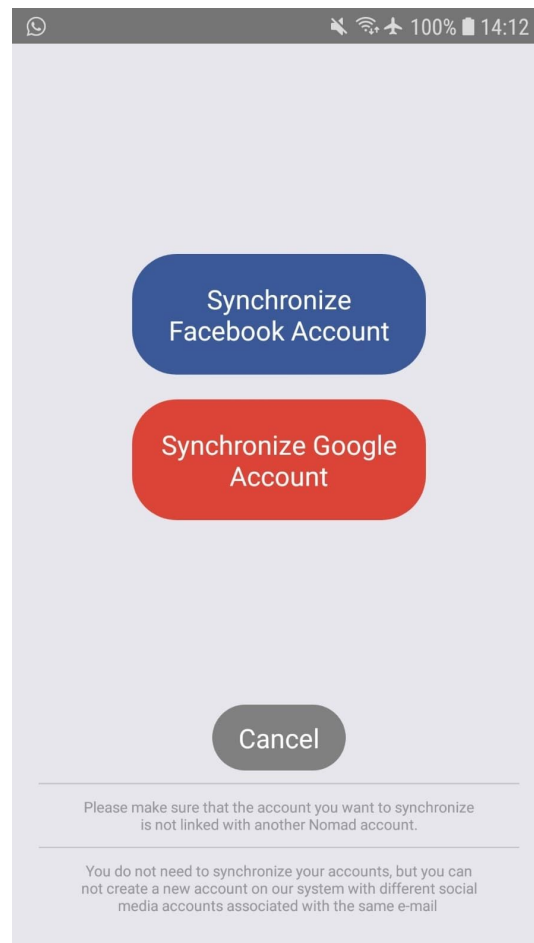
Figure 13: Synchronization Page

In this page you can synchronize an additional social media account to the account you are currently using, to be able to use the already explained synchronization capabilities. The synchronization options presented here are of course the social media accounts that the user has not yet synchronized. In the above figure you can see that google and facebook synchronization options are presented to this user meaning that the social media login this user is currently using is twitter.

The synchronization process is similar to the social login process, in the way that a synchronization request asks for user credentials. However, even if the entered credentials are correct synchronization may fail in the case that the account that wants to be synchronized has been already used to create a different Nomad account.

Create Trip

In Nomad, users will create trips by tapping the "Create Trip" button on the left sidebar. It will direct the user to the screen where s/he will put the checkpoints as markers on the map. The map and the location of the user is displayed via Google's API. We used Maps SDK for Android for the map and Geolocation API for the current location. Thus, the user sees his/her location just like in the Google Map. To start putting the checkpoints, the user will tap the start button which enables marker placing. If the user wants to stop placing markers and check around on the map, s/he can tap to the stop button so that s/he will not place a marker accidentally. Yet, if the user misplaces a marker anyway, s/he can remove it by retapping to the marker. When the user finishes placing markers, s/he can click next to customize it. When the user clicks the next button, the markers will be sent to the next part of creation where the user fills the information fields about the trip and selects a type for the trip. The code below is to locate the position of the user.

```
locateCurrentPosition = () => {
  setInterval(() => {
    Geolocation.getCurrentPosition(position => {
      //console.log(JSON.stringify(position));
      let region = {
        latitude: position.coords.latitude,
        longitude: position.coords.longitude,
        latitudeDelta: 0.005,
        longitudeDelta: 0.005,
      }
      this.setState({pos: region});

    }, error => console.log(error.message), {
      enableHighAccuracy: false,
      timeout: 10000,
      //maximumAge: 3600000,

    })
  }, 1000);
};
```

The code below shows how the markers are placed and tapped on the map.

```
<MapView style={styles.map} provider={PROVIDER_GOOGLE} showsUserLocation={true} showsBuildings={true}
ref={(ref) => this.mapView=ref} initialRegion={this.state.pos}
        onPress={(e) => {
          console.log(e.nativeEvent.coordinate);
          if(this.state.markerPlaceEnabled)
            this.setState({ markers: [...this.state.markers, { latlng: e.nativeEvent.coordinate }] })}}>

  {this.state.markers.map((marker, i) => (<Marker onPress= {(e) => { if(this.state.markerPlaceEnabled) this.onPressMarker(e)}}
  coordinate={marker.latlng} key = {i}/>))}

</MapView>
```

Edit Trip

After placing the markers for checkpoints, the user can give the trip a
name, description and a label. If the user also wants to place notes on the
markers s/he can tap the marker and a text box for that marker will appear.
Once the user gives information about the trip s/he can tap the "Create"
button so that a POST http request will be sent to the server. The php kod in
the server will run and the information about the trip the user fills in the fields
will be sent to the database and recorded with the related tripID and the
userID as well as the checkpoint markers. The markers are recorded with
different markerId's but with the same tripID in the database. The code below
shows how the trip is sent to the database.

```
async submitTrip(){
    console.log("Trip submitted");
    console.log(this.state.markers);
    console.log(this.state.titles);
    console.log(this.state.tripDescription);
    console.log('----------------');
    let obj = JSON.parse(this.props.user)
    await fetch('http://nomad-server2.000webhostapp.com/submitTripInfo.php',{
      method: 'POST',
      headers:{
        Accept: 'application/json',
        'Content-Type':'application/json'
      },
      body: JSON.stringify({
          userID: obj.userID,
          name: this.state.tripName,
          label: this.state.label,
          description: this.state.tripDescription
      })

    })
    .then(response => {response.text()})
    .then(response => {
      console.log('SUBMIT TRIP response is', response);
    }).catch((error) => {
      console.log('error is ', error.message);
    });
    console.log("Trip")

    this.submitMarkers();

}
```

Search Trips

The users can search trips by clicking the navigate icon on the top from the main page. It will direct the user to the search screen where s/he can search trips by words in the name, in the description or the label. At first all trips are displayed on the screen when the user taps the navigate icon. After s/he searches a word, appropriate results are shown in the screen. The trips are filtered with the php code in the server where we send requests, then returned as JSON objects in code to be parsed and displayed to the user. If the user wants, s/he can tap to the "See Comments" button in the Card, specific to the trip which also sends an http request to the server to get comments made on the trip. The user also can follow the trip by tapping the "Follow This Trip" button. Comments are shown with the user who wrote the comments. The code below shows how the trips are displayed.

```
<ScrollView style={styles.scrollView}>
<View>
    <SearchBar
     lightTheme
     placeholder="Type Here..."
     onChangeText = {(searchText) => this.updateSearch(searchText)}
     value = {searchText}
    />
    <Button title='Search' onPress={() => this.searchTrip()}> </Button>

    {tripsForChange.map((trip, index) =>{

        var req = this.getRandom(trip[3],trip[5]);

        return(
        <Card title={trip[2]}
            image={req}>

          <Text style={{fontWeight:"bold"}}>{trip[3]}</Text>
          <Text>{trip[4]}</Text>
          <Button title="See Comments" style = {styles.button} onPress={() => {this.seeComments([trip[0]])}}></Button>
          <Button title='Follow This Trip' onPress={() => this.onPressFollow(trip[0])}> </Button>

        </Card>)
        }

    )}

</View>
</ScrollView>
```

The code below shows how the trips are retrieved after searching. The searched words are sent to the server with a POST request. The php code in the server runs and finds the trips that include searched words in their description, name or label. The filtered trips are returned as response from the

server, parsed and finally kept in an array. The "trips" state, then, is set to the
filled array and the screen is updated.

```
searchTrip(){
    var trips = [];
    var tripsForChange = []
    var {tripsForStandStill} = this.state
    fetch('http://nomad-server2.000webhostapp.com/searchTrip.php',
    {
        method: 'POST',
        headers:{
          Accept: 'application/json',
          'Content-Type':'application/json'
        },
        body: JSON.stringify({
            searchText:this.state.searchText
        })

    })
    .then((response)=> response.json())
    .then((response) => {
        console.log('response from get: ',response);
        let str = JSON.stringify(response);
        str = str.replace(/\\/g, "");
        str = str.substr(1,str.length - 2);

        console.log('str:',str);
        let obj = JSON.parse(str);
        let array=Object.keys(obj).map(function(k){
            return obj[k];
        })

        array[0].forEach(element => {

            var trip = [];


            for(let row in tripsForStandStill){


                if(element.tripID === tripsForStandStill[row][0]){
                    tripsForChange.push(tripsForStandStill[row])
                }
            }
            trip.push(element.tripID);
            trip.push(element.userID);
            trip.push(element.name);
            trip.push(element.label);
            trip.push(element.description);
```

```
                trips.push(trip);

            });

            this.setState({tripsForChange:tripsForChange})
            this.setState({trips});
        }).catch((error) => {
            Alert.alert('The error is',JSON.stringify(error.message));
        });

    }
```

Comments

When the user wants to see the comments about a trip and tapped for that
button, all the comments recorded for that trip in the database are displayed
with the user who wrote the comment. After reviewing the comments, the user
can tap the "Back" button and s/he will be directed to the search trip screen
again. The code below shows how the comments are displayed.

```
render(){
    const comments = this.props.comments;
    const user = this.props.user;
    const obj = JSON.parse(user)
    if(this.state.enabled === false){
        return (

            <View >
                <Text style = {styles.forTitle}>Comments</Text>
                <ScrollView style={styles.scrollView}>
                {
                    comments.map(comment =>

                    <Card style= {styles.container}>
                        <Text>{comment[0] + " "}{comment[1]}</Text>
                        <View style={{borderBottomColor: 'black',borderBottomWidth: 1,}}
                        />
                        <Text>{comment[3]}</Text>
                    </Card>)

                }
                <Text> {"\n"}  </Text>
                </ScrollView>
                <Text> {"\n"}  </Text>
                <TouchableOpacity style={styles.altButton}
                    onPress={()=>this.setState({enabled:true})}>
                    <Text style={styles.buttonText}>Back</Text>
                </TouchableOpacity>
                <Text> {"\n"}  </Text>


            </View>
        );
    }else{
        return (<App guid = {obj.userID} userEmail = {obj.email} userName = {obj.username} userPassword = {obj.password} page = {9}/>);
    }
}
```

Follow Route

In order to construct routes we use map markers which are created in Create Route Component/Edit Route Component. These markers are then sent to our database. Follow Route component gets markers from the database to an array and then this array is mapped to a google map. react-native-maps module lets us place the markers obtained to their coordinates.

Below is the function used to obtain markers and construct a boolean array filled with "false" values.Purpose of boolean array will be explained.

```javascript
getMarkers(){//get markers from DB and also form bool array, filled with false.
    //let {markersChecked, markersClaimed} = this.state;
    myMarkers = [];
    tripID = this.props.trip;
    fetch( hostURL + 'getMarkers.php')
    .then((response)=> response.json())
    .then((response) => {
        // console.log('response from get: ',response);
        let str = JSON.stringify(response);
        str = str.replace(/\\/g, "");
        str = str.substr(1,str.length - 2);

        let obj = JSON.parse(str);
        let array=Object.keys(obj).map(function(k){
            return obj[k];
        })

        array[0].forEach(element => {

            ///console.log("->",element);
            if ( element.tripID === tripID ){

                m = {
                    text: element.text,
                    latlng:{
                        latitude: parseFloat(element.latitude),
                        longitude: parseFloat(element.longitude)
                    },
                    tripID: element.tripID,
                    markerID: parseInt(element.markerID)
                }

                myMarkers.push(m);
                this.markersChecked.push(false);//put exactly one false for one
                this.m=this.m+1;//increase the size by one
                //markersClaimed.push(false);
                ///console.log("marker found:",element);
            }
        });

    }).catch((error) => {
        alert('The error is',JSON.stringify(error.message));
    });
    //this.setState({markersChecked, markersClaimed});

    return myMarkers;
}
```

Above, markers are pushed to an array and that array is returned. When Follow Route Component mounts, the first thing done is getting markers and putting them in a variable. This is made sure by a componentDidMount() function in react-native. This function is executed once a component mounts.

```
componentDidMount(){
    let {markers} = this.state;
    let {coordinates} = this.state;
    markers = this.getMarkers();//get markers from DB
        coordinates = this.markRoute();
    this.setState({markers});
}
```

Next step is finding our distance to every marker. We have a defined distance limit. Which is 50 meters. We check our distance to each marker for both when we are closer than 50 meters to a marker(s) and when we are further than 50 meters to a marker(s).

Reason for checking distance in two different conditions is, when we are close to a marker under 50 meters, we get a token from that marker (token system will be explained later on). If by any way all markers are close to each other, with less than 50 meters apart from each other, we need to find out which one we are closer to in order to get that one's token only. And when that is done, the marker in question should be excluded from any remaining calculation in order to not get infinite tokens and for finding the second closest marker to get a token from.

Second case to distance checking is done for any marker which is closest to us no matter the distance. This calculation is required for plotting a route. Like the other case after a token is obtained from a marker, it is excluded from this calculation too, in order to get to the next marker with a new route.

This is for markers closer than 50 meters.

```
checkPosition(){//this function is for returning the ID of the closest marker to user within DISTANCE_LIMIT
    const {pos} = this.props;
    const {markers} = this.state;
    //Let {markersChecked} = this.state;
    let count = 0;
    var distance = DISTANCE_LIMIT;
    let flag=0;
    markers.forEach( marker => {
        var diff =  this.measure( pos.latitude, pos.longitude, marker.latlng.latitude, marker.latlng.longitude )
        ///console.log("-------------DISTANCE____IF DISINDAKI:", diff, marker.markerID);


        if ( diff < distance && !this.markersChecked[count]){// if same index for marker in question is true on b
            ///console.log("YOU ARE CLOSE TO MARKER IF ICINDE:",marker);
            ///console.log("DISTANCE IF ICINDE:",this.measure( pos.latitude, pos.longitude, marker.latlng.latitud
            distance = diff
            flag = marker.markerID;
        }
        count++;

    });
    return flag;//ID
}
```

This is for markers further than 50 meters.

```
markRoute(){// This funtion is for getting the coordinates of closest marker to user no matter where the user is for
    const {pos} = this.props;
    const {markers} = this.state;
     let {coordinates} = this.state;
    var count =0;
    var temp=1000000; //this funtion will for markers closer than 1000km
    var flag2 =0;
     let finished2= this.state;
     var m;
    for(let i = 0; i<this.m; i++){//Loop for determining if route is finished
        if (this.markersChecked[i]){// if all bool array true route finished
          this.finished= 1;//finished
        }
        else {
          this.finished =0;//not finished
        }
    }
    markers.forEach(marker => {
        var distance =  this.measure( pos.latitude, pos.longitude, marker.latlng.latitude, marker.latlng.longitude )

        if (distance <= temp && !this.markersChecked[count]){// if same index for marker in question is true on bool
          temp=distance;
          flag2=count;//index of closest marker to user in the marker array
          m=marker;
          this.coordinates= marker.latlng;//coordinates of closest marker under 1000km
          console.log(this.markersChecked,"GİRDİİİİİİİİİİİİİİ")



        }
        count++;
    });
```

From the checkposition() function, obtain the ID of the marker which is closest(<50), we use that ID to obtain the markers index by accessing it from the marker array we constructed. In order to track which marker we need and we are done with. A boolean array is created with the same size as the marker array. We each time we push a marker to array, we also push a "false" to our boolean array. When we get a token from a marker, the same index as that marker from the boolean array is turned true. That an index is true, a marker with the same index is not included in any calculation any more.

So basically, user's progress can be tracked by a simple boolean array, and each algorithm is working by evaluating this array.

markClose() function works similar to checkPosition() function, evaluates markers(>50) which do not have "true" values on the boolean array. react-native-maps-directions module is used to draw a route from our position to the marker in question. Google's Directions API and Geolocation API are used in this case. API key acquired from Google Cloud Platform contains authorization to use both APIs.

Furthermore, conditional rendering is used for displaying a route to each marker, as well as driving routes and walking routes. When we get close to a marker for more than 50 meters, which is a condition, rendered messages and buttons are different.

```jsx
{closeEnough>0 &&//if distance to closest marker under DISTANCE_LIMIT show AR button
  <View style={{flex:1,flexDirection: 'column',height: hp('20%'), width: wp('96%')}}>
   <View style={{backgroundColor:"#BF1E2E", padding:10}} >
    <Text style={styles.textStyle2}>Close to a checkpoint! Please STAY STILL and look for a Token with your camera!</Text>
    <Text style={styles.textStyle3}> Distance :{this.mapdata.distance} km, Duration : {this.mapdata.duration} min</Text>
   </View>
   <View style={{flexDirection: 'row'}}>
    <TouchableOpacity style={styles.buttonStyle} onPress= { () => {this.buttonPress(closeEnough)}}>
     <Text style={styles.textStyle}>Get Token</Text>
    </TouchableOpacity>
    <TouchableOpacity style={styles.buttonStyle} onPress= { () => {this.mapView.animateToRegion(pos, 2000)}}>
     <Text style={styles.textStyle}>Find Me</Text>
    </TouchableOpacity>
    <TouchableOpacity
     style={styles.buttonStyle}
     onPress= { () => {this.mapView.fitToCoordinates([{latitude: this.coordinates.latitude, longitude: this.coordinates.longitude}], {
      edgePadding: {
       right: (width / 20),
       bottom: (height / 20),
       left: (width / 20),
       top: (height / 20),
      }
     });
   }}>
```

View tag above is rendered if the value "closeEnough" is bigger and 0. This is the ID of the marker closest to the user.

After all routes are finished and every token is collected, which means every element of the boolean array is "true", a global value 0 is turned 1, and the finish button is rendered. We press finish and the trip is posted to the database.

```
finishTrip=()=>{
    console.log(this.props.user);
    console.log(this.props.trip);

    fetch(hostURL + 'finishTrip.php', {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            user: this.props.user,
            trip: this.props.trip
        })
    })
    .then((response)=> response.json())
    .then((response) => {
        console.log('finishTrip response: ', response);
        if(response.result == 1){
            this.props.onBack(9);
        }
        else if(response.error != ""){
            alert("Finished trip could not be saved");
        }
    }).catch((error) => {
        alert('Finish trip error: ', error);
    });
}
```

Below code is where the route is drawn.

```
<MapViewDirections
  origin={{latitude: this.props.pos.latitude , longitude: this.props.pos.longitude}}//from
  destination={this.coordinates}//closest maker
  apikey={GOOGLE_MAPS_APIKEY}
  resetOnChange = {false}
  timePrecision = "now"
  presicion = "high"
  mode={this.mode}
  strokeWidth= {5}//kalınlık
  strokeColor = "#BF1E2E"//renk
  onStart={(params) => {
   console.log(`Started routing between "${params.origin}" and "${params.destination}"`);
  }}
  onReady={result => {
    console.log(`Distance: ${result.distance} km`)
    console.log(`Duration: ${result.duration} min.`)
    this.mapdata.distance = result.distance;
    this.mapdata.duration = parseInt(result.duration);

    this.mapView.fitToCoordinates(result.coordinates, {
      edgePadding: {
        right: (width / 20),
        bottom: (height / 20),
        left: (width / 20),
        top: (height / 20),
      }
    });
  }}
/>}
```

AR Component

The AR module used for android systems is ViroReact. But unfortunately this module was broken for android while its iOS counterpart ViroCore was operational. We found out that ViroReact was at an experimental phase so we needed to find another way to implement AR.

We used sensors, camera and images for this. react-native-sensors module is used to access gyroscope data from the device. And react-native-camera module to access the camera of the device.

A phone screen consists of pixels which are then form height and width of the screen. We treated the dimensions (pixels) as a coordinate system. An android phone gyroscope returns angular velocity rad/s along x, y and z axes.

Using this it is possible to determine the angular rotation of the device, which means, which way the device is turned.

```
this.token= {x: 0, y:0}//token place (pixel)
this.state = {
  x: Math.random() < 0.5 ? (Math.random()*(-15)) : (Math.random()*(15)),//
  y: Math.random() < 0.5 ? (Math.random()*(-20)) : (Math.random()*(20)),//
  //t: true,
};
}
```

this.token is the place of the AR object on screen. this.state.x and this.state.y is the initial place of the AR object.

```
componentDidMount() {
  BackHandler.addEventListener('hardwareBackPress', this.handleBackButton);//telefonun geri tuşu çalışmasın
  const subscription = gyroscope.subscribe(({x,y }) => {//gyroscope açık
    x= parseInt((100*x))/100// degree to int
    y= parseInt((100*y))/100
    this.setState(state => ({
      x: x + state.x , y: y + state.y// degree turn
    }));
    //console.log(this.state.x,"XX....YYYY", this.state.y);
    if(this.token.y >2150 || this.token.y<-2150){//360 degree turn
      this.setState({x:0});

    }
    if(this.token.x >2150 || this.token.x <-2150){//360 degree turn
      this.setState({y:0});

    }
    this.token.x = parseFloat((this.state.y - 0.03) * 30)//degree to pixel translation
    this.token.y = parseFloat((this.state.x + 0.05) * 30)//degree to pixel translation

  });
  setUpdateIntervalForType(SensorTypes.gyroscope, 35);//gyroscope interval, smaller for more presicion,
  this.setState({ subscription });                    //but if too small phone heats up! 45 is fine
}
handleBackButton() {//telefonun geri tuşu çalışmasın
  return true;//telefonun geri tuşu çalışmasın
}
componentWillUnmount() {
  BackHandler.removeEventListener('hardwareBackPress', this.handleBackButton);//telefonun geri tuşu çalışmasın
  this.state.subscription.unsubscribe();//gyroscope kapalı
}
```

Every time the device is rotated, measured angular velocity is added every 35 milliseconds. This velocity can be negative or positive. Result of summation along x and y axes are then added to the initial position of the AR

object on screen. This data gives us the degree/seconds which device is rotated along x and y axes. We need to translate this data to pixels. So a degree/seconds to pixel translation is required.

```
this.token.x = parseFloat((this.state.y - 0.03) * 30)//degree to pixel translation
this.token.y = parseFloat((this.state.x + 0.05) * 30)//degree to pixel translation
```

This formula is constructed by trial and error, where we basically aimed to "move the object 50 pixels to the opposite direction of which axis the device is rotated". This formula is calibrated to work on every screen as smooth as possible. Since each device has different height and width. Translated data is put into this.token.x and this.token.y. Then our AR object is rendered at that location and it is re-rendered each time the device is rotated along x or y axes.

This way we implemented our AR but this system caused a problem to us. Coordinate system we constructed had 2 axes x and y. Length of the axes are infinite. So basically we moved the object along a 2d infinite coordinate system. When the device was rotated 360 degrees object was not on screen, it was moving along one of four directions. In order to see the object again, the device needed to be rotated along the opposite rotation, 360 degrees. Screen of the device acted as a small window to an infinite coordinate system where the object was moving along the direction of which axis the device is rotated.

```
if(this.token.y >2150 || this.token.y<-2150){//
    this.setState({x:0});

}
if(this.token.x >2150 || this.token.x <-2150){//
    this.setState({y:0});

}
```

With these constraints, when the device rotated 360 degrees along x or y axes the object is re-rendered on its initial position. Object in question is a

token. When the user taps on a token, the boolean array created on FollowRoute Component, gets a true value.

Achievements

The achievements can be obtained by collecting tokens can be seen by tapping the "Achievements" button, which returns a list of available achievements from the database, on the sidebar. It is available for every user to see. It is an indicator to show how much the user travels for other users. The code below shows the achievements.

```javascript
getAchievements(){
    var achievements = [];


    fetch('http://nomad-server2.000webhostapp.com/getAchievements.php')
    .then((response)=> response.json())
    .then((response) => {
        console.log('response from get: ',response);
        let str = JSON.stringify(response);
        str = str.replace(/\\/g, "");
        str = str.substr(1,str.length - 2);


        let obj = JSON.parse(str);
        let array=Object.keys(obj).map(function(k){
            return obj[k];
        })

        array[0].forEach(element => {

            var achievement = [];
            achievement.push(element.ach_id);
            achievement.push(element.name)
            achievement.push(element.ach_desc);
            achievement.push(element.token);
            achievements.push(achievement);

        });

        this.setState({achievements:achievements})
    }).catch((error) => {
        Alert.alert('The error is',JSON.stringify(error.message));
    });

}
```

5. Testing Details

During the implementation process, we have done testing process frequently because adding more features reveals new problems/bugs. Actually each member of the group became a tester for the app. After every version we used, tested the app for different scenarios. This made solving issues/bugs easy because, sometimes, everybody faced different problems. With the feedback that we share with each other, the app becomes smooth and working fine.

With the tests we conduct, we changed, updated and increased efficiency of our app day by day. Today maybe there are some bugs we have not faced yet. However, when users will start using it, we will get more feedback and according to those feedbacks will make the app better and bug free.

Testings consisted of trial and error. COVID-19 incident left us in a bad situation. We tried to build a navigation app without stepping out from our homes. So in a lot of test cases we have used, we simulated movement instead of actually going out/walking. We did this by manipulating coordinates and distance limits.

For tests we used devices such as Samsung Note 8, Samsung A50.

6. Maintenance Plan and Details

Since we use an online database for the app, if the number of users expands, we may need to change our database to satisfy user requests. That database will not be enough for a very large number of users right now. Other than that we use Google API's (maps, direction etc.). We have limited requests for free accounts right now, however, in the future we may need more capacity than we have now. That is why we will expand Google API request capacity to satisfy user demands.

Other than that this app is only available for android users now. For the future we may add iOS option and web interface according to user

demands. We cannot ignore the number of iOS users that want to use our app.

We will keep the database updated and increase efficiency as the number of users increases.

7. Other Project Elements

7.1. Consideration of Various Factors

7.1.1. Adaptation to massive data

Our application can reach a very large scale. Since the created routes can be published, the users can see all of the routes in the world. This large scale creates the problem of storage. It affects both maintainability and sustainability.

7.1.1.1. Safety

This constraint is very important. Users share information about themselves through our app, so safety is one of the top important things according to our criteria. When they are creating routes, their GPS will always open during that time and people may feel suspicious about that since they can be known by someone else in that time. Users' current location while following a trip or creating a trip only visible to him/herself. Thus, other users or third party apps cannot see a user's current location. It can be really hard to get user's trust. In order to give that feeling, we took further security for our application. Also a user is required to create a password with at least six characters with letters and numbers. So they cannot set an easy password such as "123456".

7.1.1.2. Global

Since our search criteria can include other nations too, it can be used globally. If a tourist wants a good experience, s/he may want popular routes of that country and our application will give these routes based on popularity. After searching among those options, the tourist can decide on the best

routes for her/him. We made our app to be used globally. So, everybody can use it wherever they want. For future work we may add different languages.

7.1.1.3. Economic

Adaptation to data will require money for storage. It becomes harder to control something that is growing because the parts cannot be treated equally. Any new additional feature will be needed for maintenance but its sustainability can be affected if the effect of maintenance will be less than sustainability. If we try to restrict sustainability, in that case, our changes may not be that satisfying for users. Today we have few users and data. However, if our users grow exponentially we will make investment for data storage and security of the data.

7.1.2. Efficient usage in terms of time while searching and downloading the path

The data to store a route is relatively high for other objects in our system. If the amount of data increases, the cost of searching increases too. Creating a route is relatively simple. In the user side of view, it is just a path and some details about routes such as notes, checkpoints, and location information. However, we store the path by sampling the path with GPS. With that information, we recreate the route on the user's device.

7.1.2.1. Economic

The amount of data storage units will be highly dependent on the sampling rate of the paths. If we sample them in half of the previous frequency, the data is doubled so the potential storage units will be doubled. In that sense, the economy will be a powerful constraint.

7.1.2.2. Global

If the application serves everyone from just one server, the efficiency will dramatically fall. Downloading the content is problematic already and if the people from all around the world try to use our application, this problem is getting bigger and bigger. Since our app is not global right now, it is not causing a problem, but when it's getting bigger we will increase efficiency with updates.

7.1.3. Unfairly taking advantage of collecting tokens

Our application should be bug free and take precautions to eliminate the unfair advantage of collecting tokens. It is important because it can affect the trust of users. In that situation, the decrease in the number of users would probably be high. To prevent that the user really should follow to collect tokens and create new trip. Creating a new trip also required a minimum distance. According to the distance users can place a certain amount of tokens to the route.

7.1.3.1. Social

If the trust of society is lost just one time, it would be very hard to regain it. However, sometimes no matter how hard you try to eliminate it, they can happen.

7.1.3.2. Environmental

If a bug happens and there are some routes that are more advantageous during that bug, these roads can be used excessive amounts. Unfortunately, if there are people, it means that there is also pollution. These people may pollute those routes by throwing garbage in the simplest sense. There is no such problem right now, but, with some solid precautions we will prevent them immediately when it happens.

7.1.3.3. User's tolerance

One possible solution that comes to mind is to block the entrance to the application or make the tokens valueless. But this solution can change the perception of the users to our application before the bug and after the solution. They can think that the application is too amateur and decide not to use it. So, users will have a chance to regain tokens and create apps. They need to follow some routes to gain a few tokens.

Judgement Description: **Adapting to massive data can be problematic.**		
	Impact Level	**Impact Description**
Impact in Safety	8	Safety in bigger systems becomes harder to establish.
Impact in Global	5	Such a great scale will make sustainability harder.
Impact in Economic	7	Purchasing new data storage units can be costly.

Judgement Description: **Time efficiency for searching and downloading**		
	Impact Level	**Impact Description**
Impact in Global	5	Number of servers is important for that problem.
Impact in Economic	8	The cost of new servers within a country can be a problem.

Judgement Description: **Bugs about collecting tokens may occur**		
	Impact Level	**Impact Description**
Social	9	People may lose their trust on our application.
Environmental	8	If they use a beneficial bugged route, people may pollute there.
User's tolerance	9	Users may lose their tolerance because of this bug.

7.2. Ethics and Professional Responsibilities

The object that will be placed by the users may cause accidents. There are places that people should be careful when they visit, like the Grand Canyon, as the slightest distraction can cause people to get hurt or fall. When we consider it in a more general sense, this is also valid for any street or building depending on the location of the object. If an object placed by a user can be collected only by standing on a street or top of a building, searching for this object poses a risk for human life. Thus, we took this situation into consideration while implementing the project.

Creating or completing routes may require visiting places like museums. If those museums are also checkpoints, meaning that the traveler needs to collect or place an object there, the traveler should use the camera on his/her phone, which is generally illegal. If we allow placing objects in such places, we would be forcing people to break the laws.

We also keep account information of the users secure, meaning that it cannot be shared and it is protected against third party software. When there

is a need for statistical data, the account information that we keep can be used. Yet, it cannot contain the personal information of the users. We also get the current location of a user when creating a trip or following a trip. Since the current location of the user is very confidential, we do not share this info with third party apps.

7.3. Judgements and Impacts to Various Contexts

Nomad is a social app. People need to sign up to the app, share their routes for maintenance of the app. Since sharing personal stuff is so popular these days, we made a judgement that people will want to share their routes through other people. Because, our app was constructed on this main idea. We have to think that people will want to share or people will want to follow a route using Nomad. To make it happen we tried to make a user-friendly and safe app.

Judgement Description:	People want to share their trips, routes to other people.	
	Impact Level	Impact Description
Impact in Global Context	High	Since our app will be open for global, every user should want to create a route or follow a route.
Impact in Economic Context	Low	Users will not spend money for using our app. However, we should make investment in our app if user numbers are increased.
Impact in Environmental Context	Medium	Our app should be considered as a safe and usable app in the environment.

Impact in Societal Context	High	Since, Nomad is a social app, it should make a good impact on the society. because people should suggest our app to others, to help us grow.

7.4. Teamwork and Peer Contribution

Gulnihal Muslu:

- Research about the framework, libraries, API's and AR
- All the reports
- Google Maps usage
- Back-end and Front-end of creating, searching following a trip, comments and achievements.
- Database

Can Ozan Kas:

- Search for remote database and server
- All the reports
- Back-end of creating, searching, following a trip, comments, achievements and profile
- Front-end of creating, searching, following a trip, comments and achievements
- Database

Tarık Emin Kaplan:

- Research about the framework, libraries, social media API's
- Front-end and Back-end of all login options, synchronization, settings page (modals included), profile, finished trips, comment and rate pages.
- Front-end of main page and sidebar
- Integration of map-using components to the application

- Setting up necessary keys and acquiring necessary permissions for all social logins and Google maps
- Database
- All the reports

Berk Atac:

- Research about the framework, libraries, API's and AR
- All the reports
- Google Maps usage, Google Directions usage
- Back-end and Front-end of creating, following a trip (map/route backend and front-end).
- Back-end and Front-end AR Component. Integration of AR.
- Front-end main-page

Ali Kemal Ozkan:

- Research about React-Native Framework, JavaScript, Google API's and AR Component
- All the reports (Most of the final report)

7.5. Project Plan Observed and Objectives Met

We basically followed the project plan that we offered in the analysis report. We divided the work in phases. A phase is made of various tasks and work packages (WP). All WPs have a leader assigned to it. Through the whole project all members will get to be the leader of multiple work packages of various types. Tasks, which are part of WPs (and sometimes directly a part of phases), are assigned to any number of group members, depending on the difficulty, workload and collaboration need of each task (a task that is not assigned to a specific member is assigned to all members). For example, a specific modelling task is usually assigned to only one member, however long lasting tasks which require more effort like implementation, or complex tasks which require all members to be on the same page such as class designs, or

meetings which require all members to be present are assigned to all members.

We tried to be strict with days and deadlines but it is hard to obey the plans in real life. Sometimes, things get harder, we spend more time on a phase, research or development. In that time, we helped each other and we spent more time than we planned. Therefore, we had to do other parts faster. Especially with pandemic situation, our plan halts a little bit, but we recover in a short time. Since we had more time to develop than we expected, delay of the deadline helped us in the development process of our app. During this period, we did online meetings often, and kept in touch with other group members every time.

As we said in the analysis report, we followed a development process with agile characteristics. We started the development process in our high-level design phase, where we created our design and implemented our project simultaneously. In each phase, where a development process is present, we planned the development process (iteration) for that phase, then we carried out the necessary development steps and we ended the iteration with a reflectionary meeting. These development processes are present in the two design phases and the final phase. The development process in the final phase is longer than the ones in design phases, and we carried out most of our implementation work there.

We have met most of our objectives during this development process. However, there are some features that we did not implement. We left those for future work and maintenance plans. For instance, for now we do not have a "add to favorites" feature as we decided that it is not essential for the purpose of Nomad and the users can see the trips they have followed anyway, so it can be implemented later. Also we do not have a feature for "seeing other users' profile" as it is also not essential for Nomad as the users can already see the evaluations of other users for a certain trip.

7.6. New Knowledge Acquired and Learning Strategies

We used React Native framework, which is a really useful framework while working on an android app. React Native is an open source mobile application framework and it is widely used since it supports cross platform development and has an active community which creates many libraries for various uses. Therefore, there are a lot of resources and tutorials for new starters to learn. It also requires a high level of JavaScript knowledge. React Native uses JavaScript so we started working on it. Tutorials and youtube videos really helped us in that process. When we got stuck during implementation or got an error that we don't know why, we asked stackoverflow. Stackoverflow is the number one website that helped during the development part.

To get user's locations or track a user location we used Google directions API and maps API. We used these for the first time but, Google's own tutorials are really useful and they really helped us.

Since the app can be used from everywhere we had to use an online database. For our hosting server we used a free online service provider called 000webhost.com and as our database we used a free online database provider "remotemysql.com".

Each member on the team learned how to develop a mobile application using react-native. As a side note, we learned that react-native modules created for iOS systems are generally better than android ones. For example android devices do not get location information while on the app in the background, iOS devices do.

8. Conclusion and Future Work

To sum up, we constructed an app which users make their trips more enjoyable. Users can create routes to wherever they want and checkpoints to share with other users. Thus everyone will have a chance to follow a

route or create a route. We think that this app will make trips more fun. Everyone will find their dream route with this app even if they visit that city/country for the first time.

We used mainly React-Native, Javascript and Google API's to make this app release. We add AR objects to the app which works in checkpoints, to make it more fun, so users can collect tokens using their cameras.

The app still needs some improvements and upgrades. We will add some safety precautions and will increase efficiency to satisfy user requests with expansion of the number of users. We will improve our search algorithms and add some suggestion options to offer trips to users according to their interests or previous trips. We may add a messaging system with other users to make this app more social.

Application definitely has potential but more work and additional features are needed to make it more appealing to users. Our system is built in a way which we can add new features easily.

9. User's Manual and Installation Instructions

9.1. Installation

To install the app without an .apk file, users need a computer with react-native setup and android device or emulator. After connecting an android device or emulator, the project can be downloaded from https://github.com/gulnihalm/nomad-design this address. After download, users need to open console/terminal and do followings:

1. cd "project-location" (change directory to project folder)
2. Run "npm install" in root project folder
3. "cd android" then run "gradlew clean" until it works
4. "cd -project-folder- " then "npx react-native run-android"

Or, users simply need to run .apk of our app on their android device.

9.2. User's Manual

9.2.1. Login & Sign Up
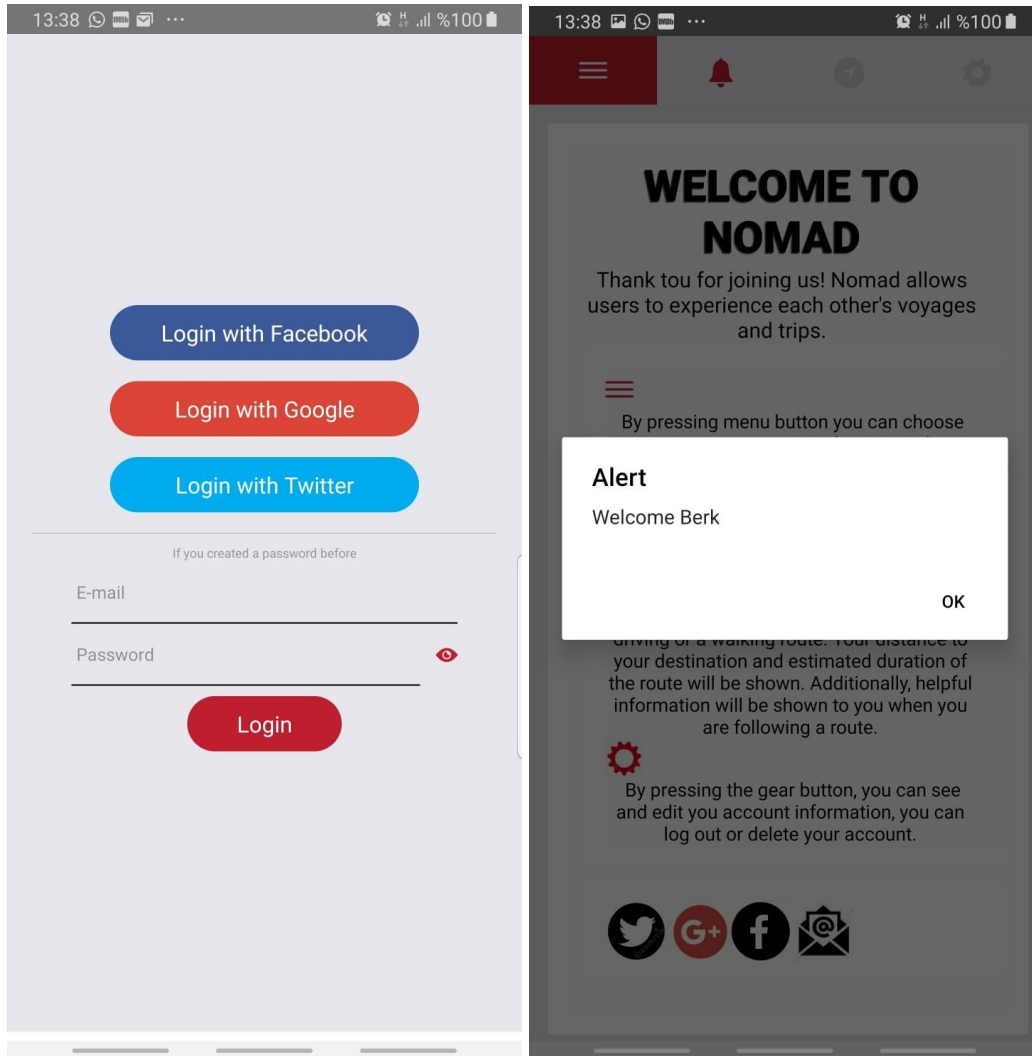
Figure 14: Login

Users can sign in with their social media account in the first place. After login, a welcome message will show up with special to their user name.

9.2.2. Welcome Page and Sidebar
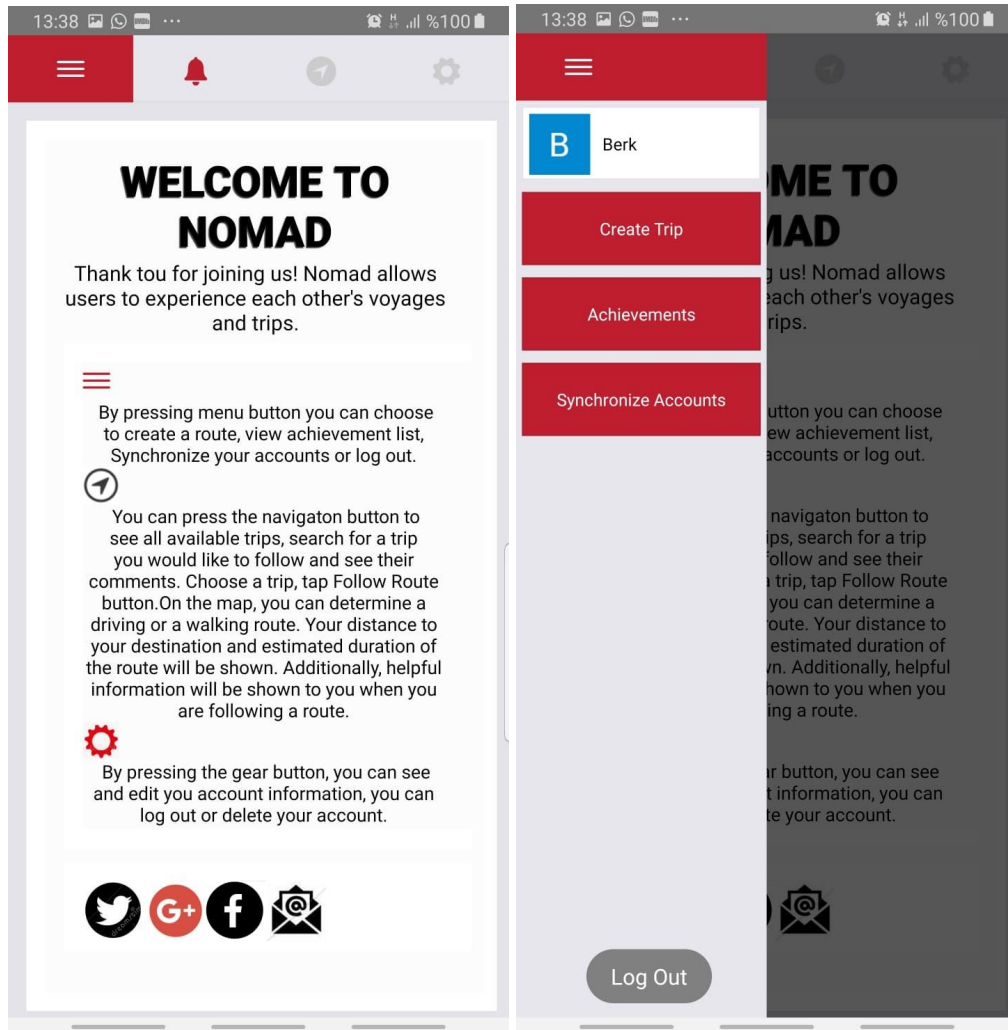
Figure 15: Welcome Page and Sidebar

After a welcome message users will see our main page which shows instructions for our app. With clicking the sidebar button, users can see options such as create trip, achievements and synchronize accounts.

9.2.3. Synchronize Accounts
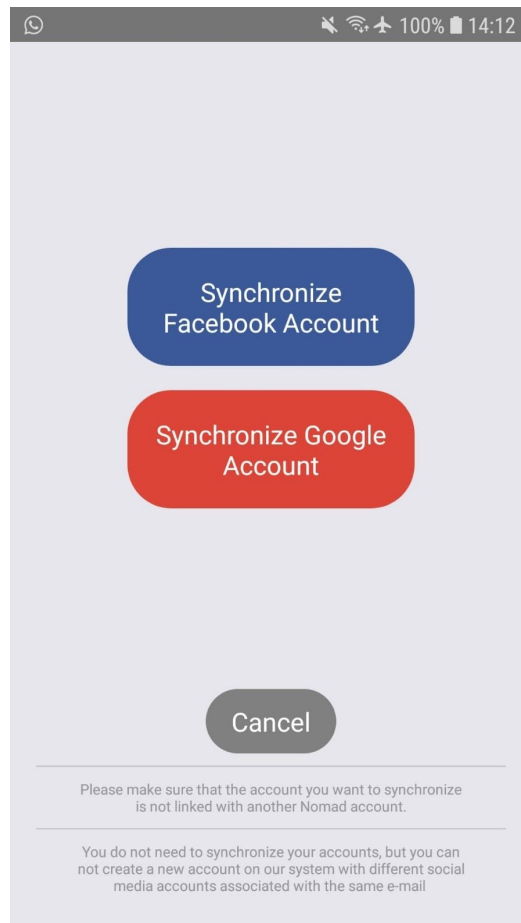
Figure 16: Synchronize Accounts

From the sidebar, users can synchronize their accounts. For example, if a user login with twitter account, s/he can connect facebook and google accounts to the same Nomad account. In that way users will have a chance to login to the same account with three different social media options.

9.2.4. Profile Page
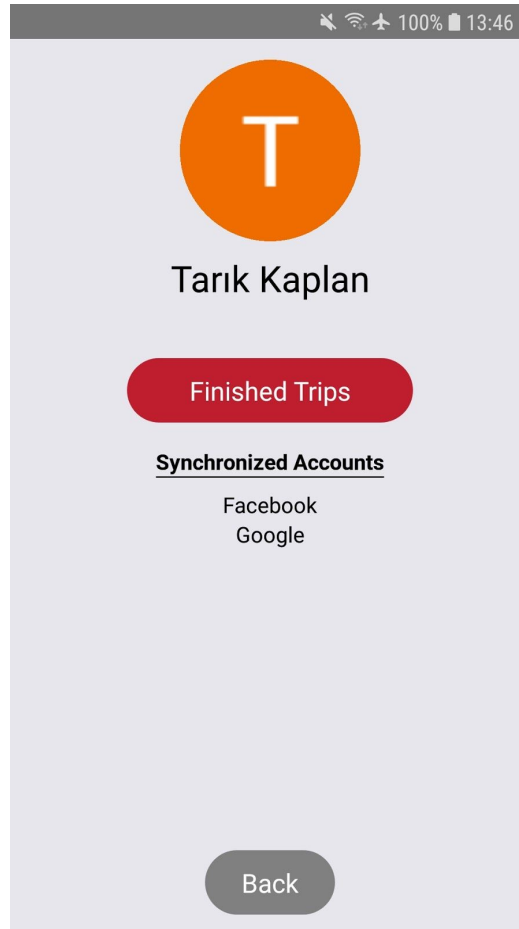
Figure 17: Profile Page

Users can see their profile page by clicking their name on the sidebar. They can see their synchronized accounts and finished trips on this page.

9.2.5. Finished Trips & Make Comment
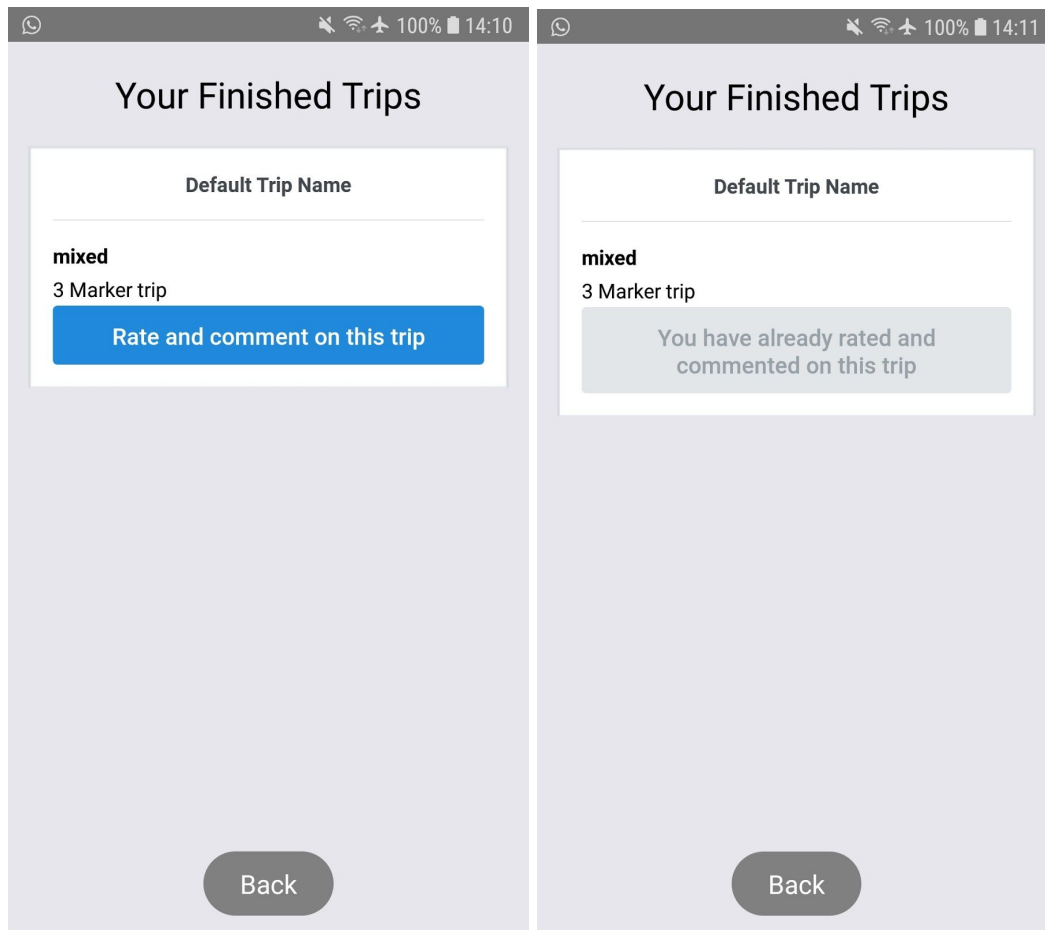
Figure 18: Finished Trips

Users can see their finished trip from their profile. They can make comment and give rates out of five-star. If rate is already given, this option will be unenabled.

Users can make comment to their trips after finishing it. They can write comment and give starts out of five. After submitting the comment, it will be published in that trip's comments section.

9.2.6. List Trips and Search Trip
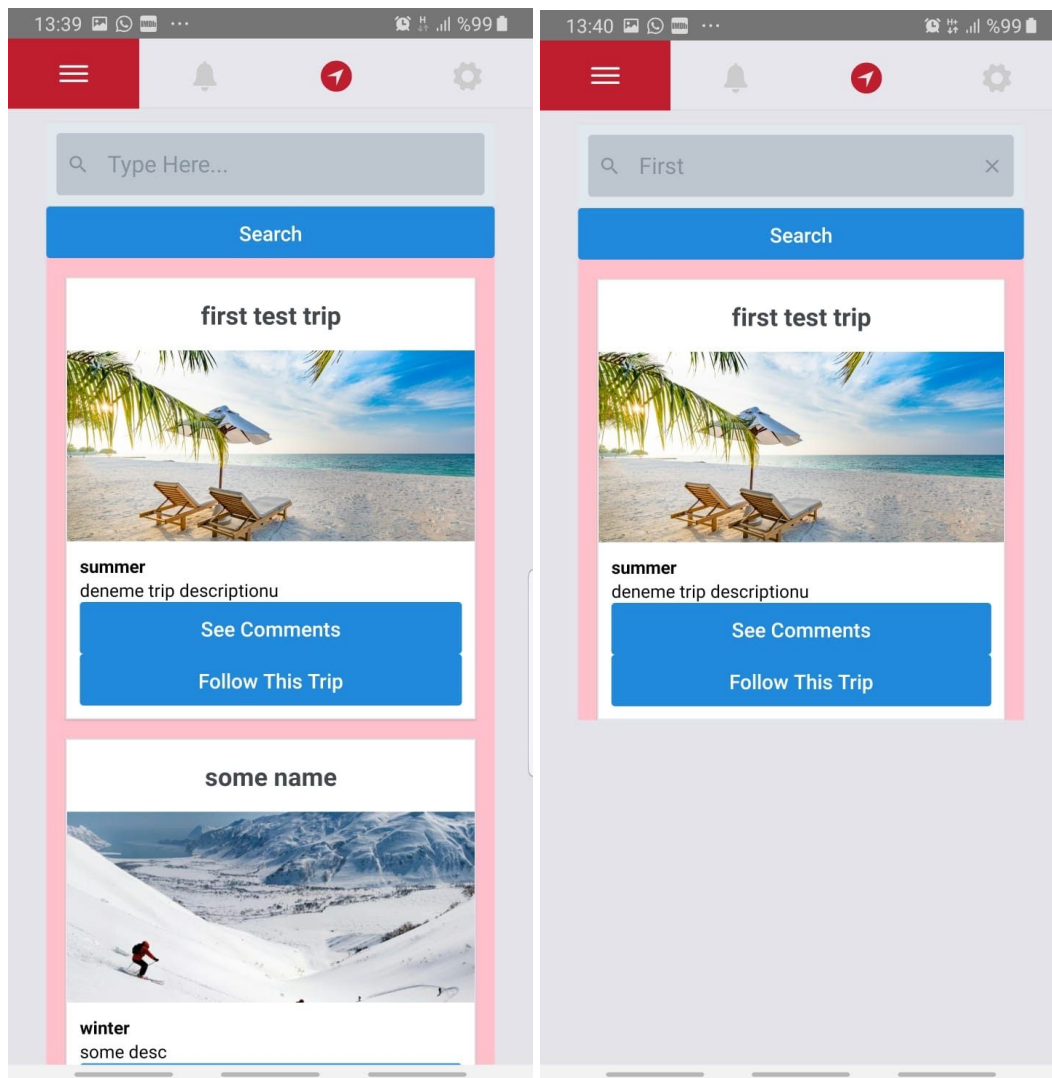
Figure 20: List Trips and Search Trip

On the top middle of the main page there is a navigation button to see all available trips. Users can see all trips from there and make a search by word for a specific trip. In the trip page there are see comments and follow this trip options. Users can follow a trip from this option or can see comments before following.

9.2.7. Comments
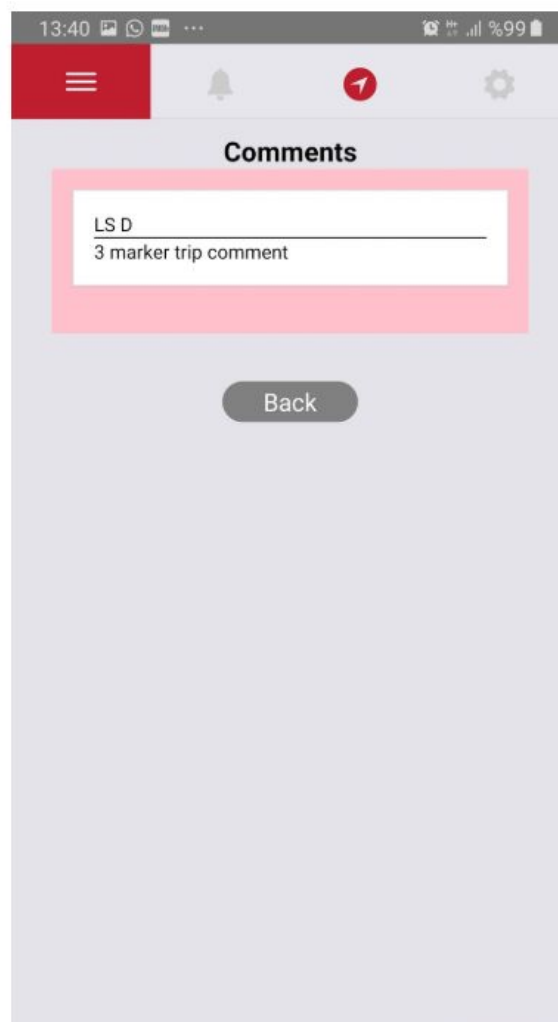
Figure 21: Comments of Specific Route

Users can list the comments of a trip by clicking the "see comments" option on the trip page.   With back option or using sidebar, users can see every page they want.

9.2.8. Follow Route
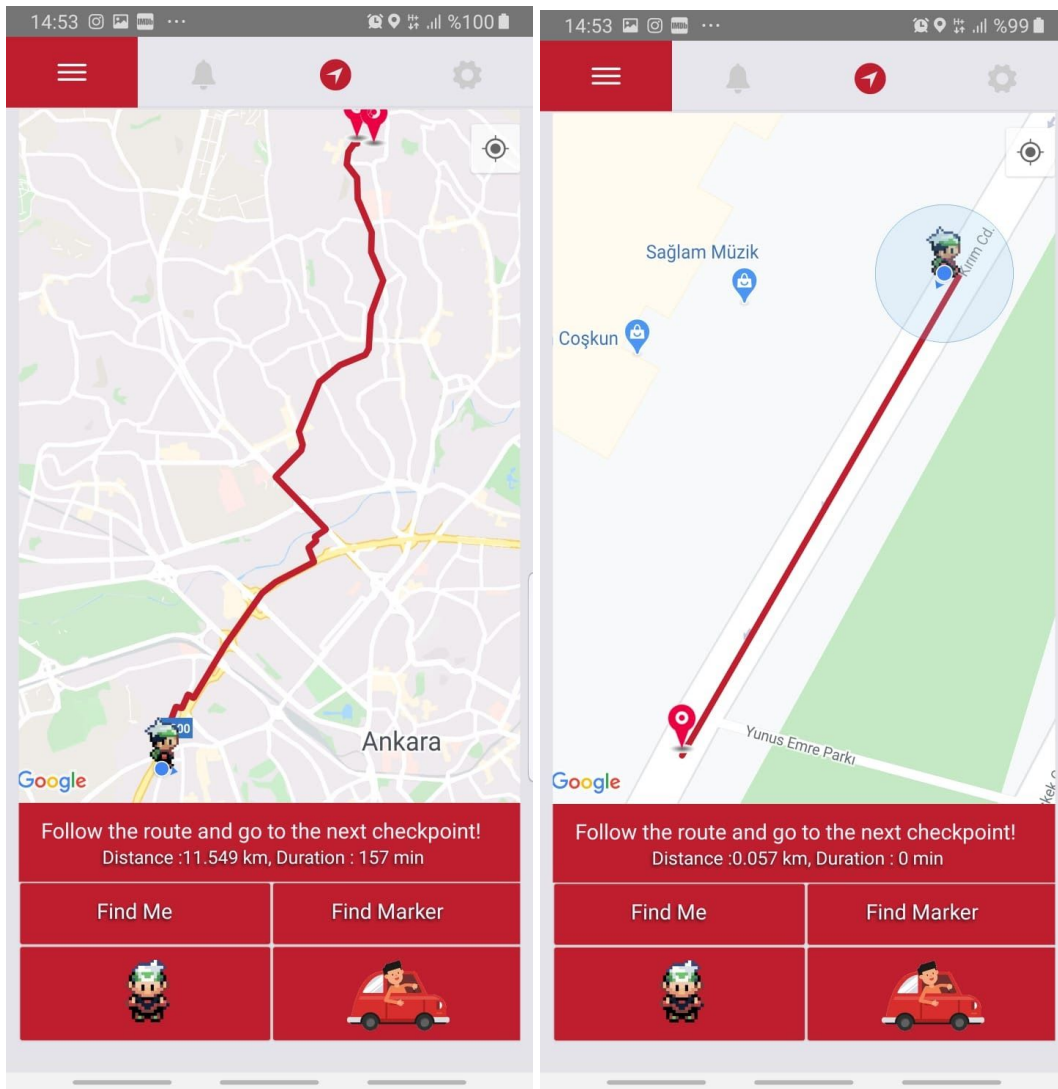
Figure 22: Follow Route

Users can choose a trip from trip list and follow a trip. After clicking follow this trip button, the map will open and it will show the duration and distance. Users can see themselves or markers on the map by clicking "find me" and "find marker" buttons.
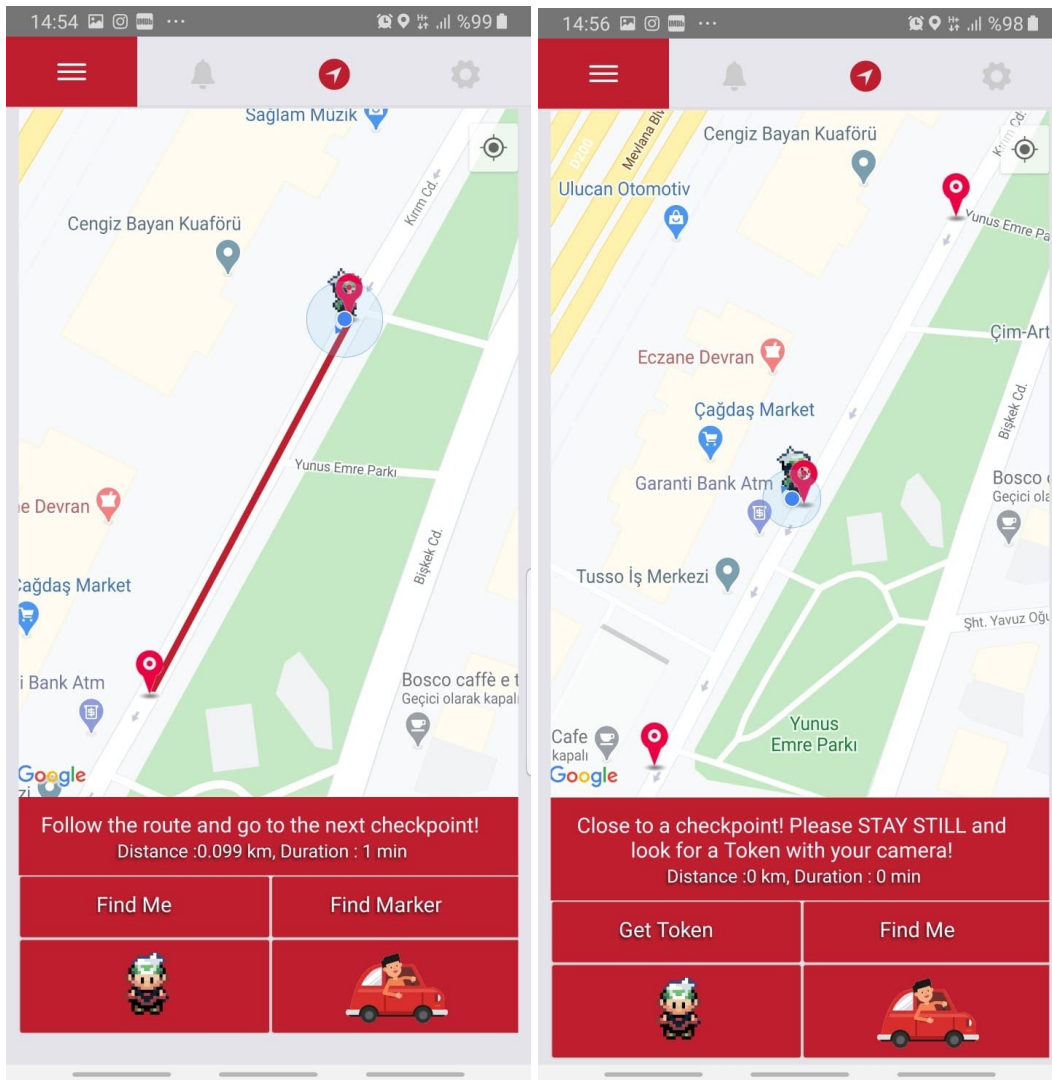
Figure 23: Progress on Route

After getting close to a checkpoint, a message will show up on the screen that says "Close to a checkpoint!.. look for a token with your camera!". If this message shows up, get token button will be active and users can look for a token around.
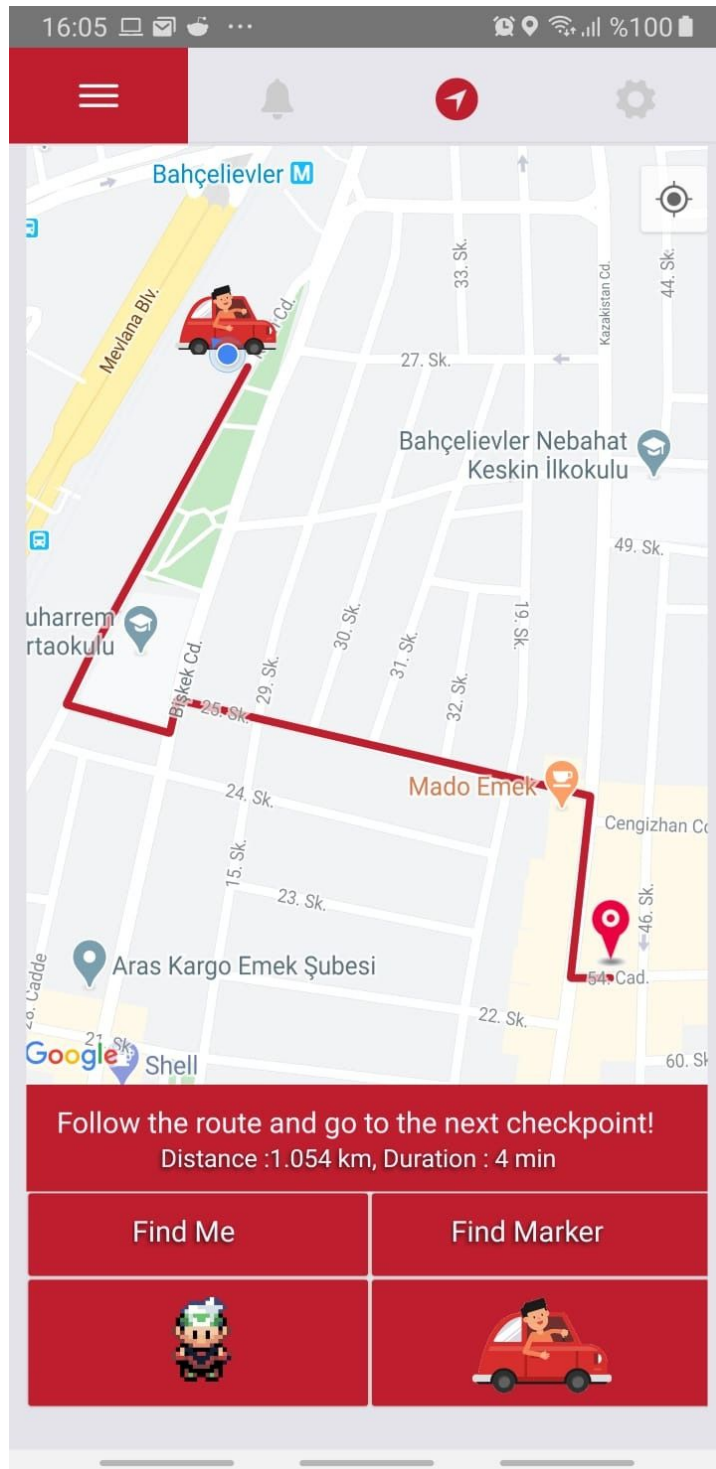
Figure 24: Follow trip with car

Users can follow a trip with a car or by walking. They can choose that from the bottom of followtrip screen. At any time during the trip, they can change this option. After changing it, route and duration will be updated in real time.

Figure 25: Getting token with using camera

This is the most enjoyable part of Nomad. After clicking get token button, the camera of the device will be open. Users have to search for a token around, after finding it, they click on the token and this finishes the collecting token process for that checkpoint.
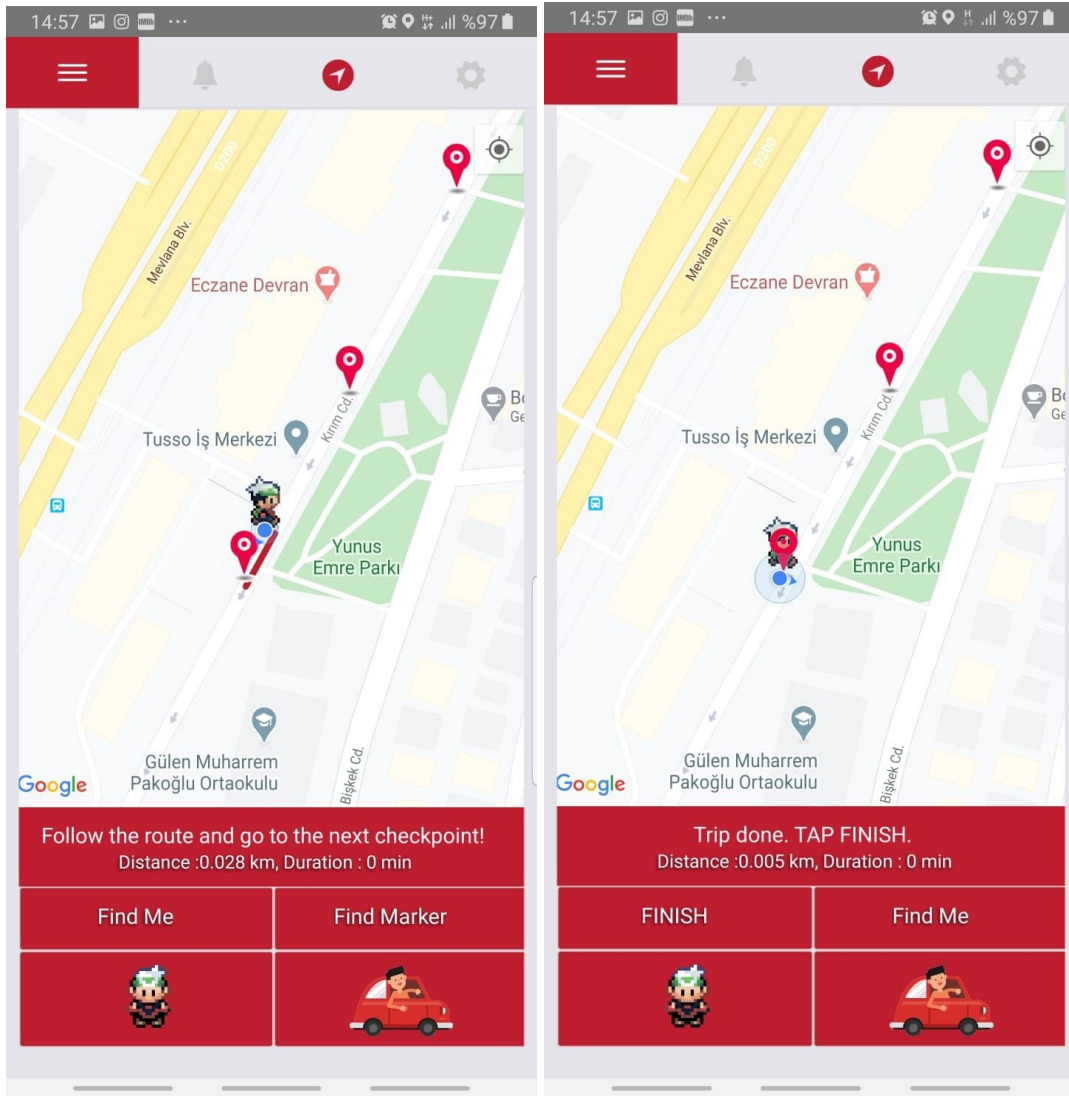
Figure 26: Finish Trip

After following all checkpoints and collecting token users can finish the trip by clicking the finish button.

9.2.9. Create Trip
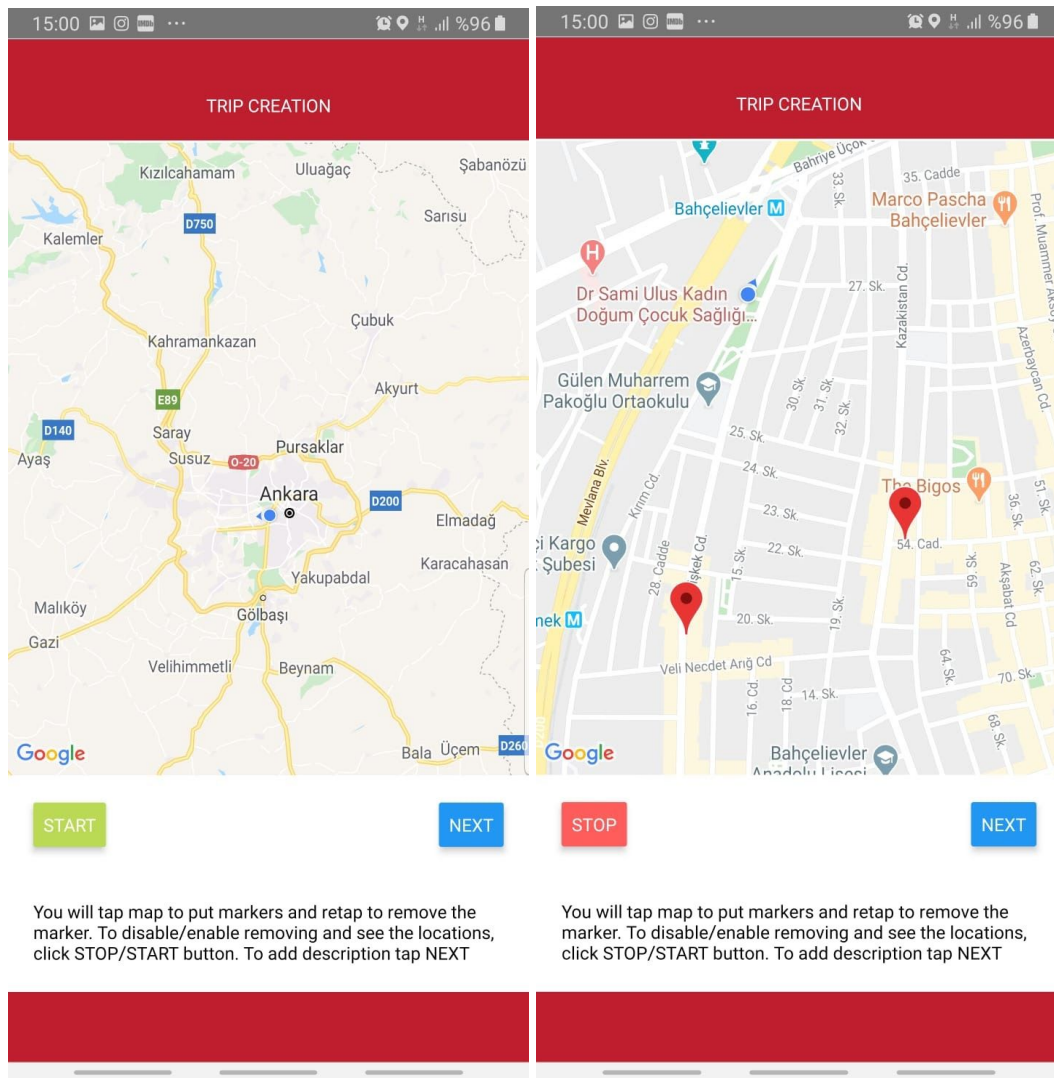
Figure 27: Create Trip

Users can create a trip using sidebar-create trip option. After clicking it, create trip page will open. By clicking start, user will have chance to put markers (which will be checkpoints) on the map. They can press stop to disable putting markers. After putting all markers to the maps, user will click next button and set a trip with those markers.
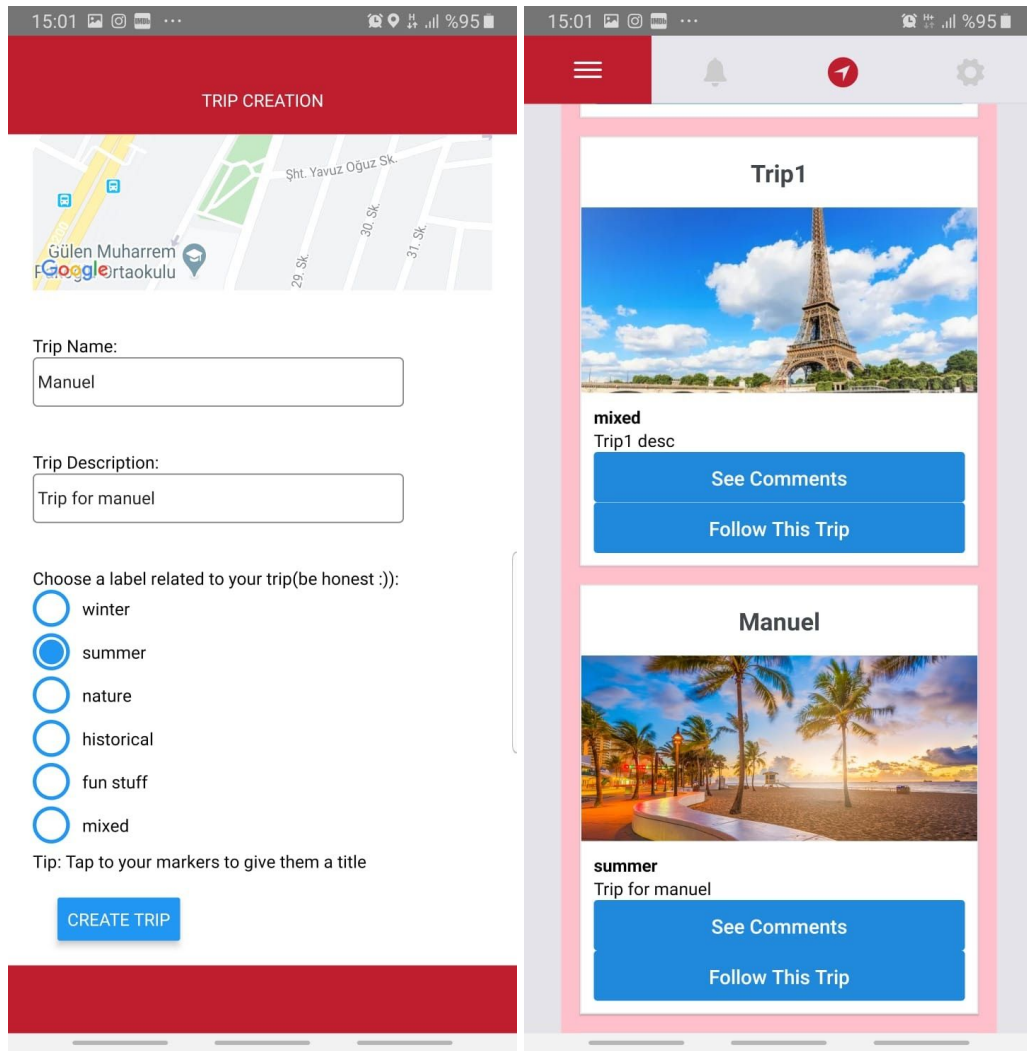
Figure 28: Trip Creation Page

After clicking next button, user will have a chance to name the trip, give the trip description and choose trip theme. In addition, users can give title to all markers by seperate. With create trip button, trip will be published on trips page immediately.

9.2.10. Achievements
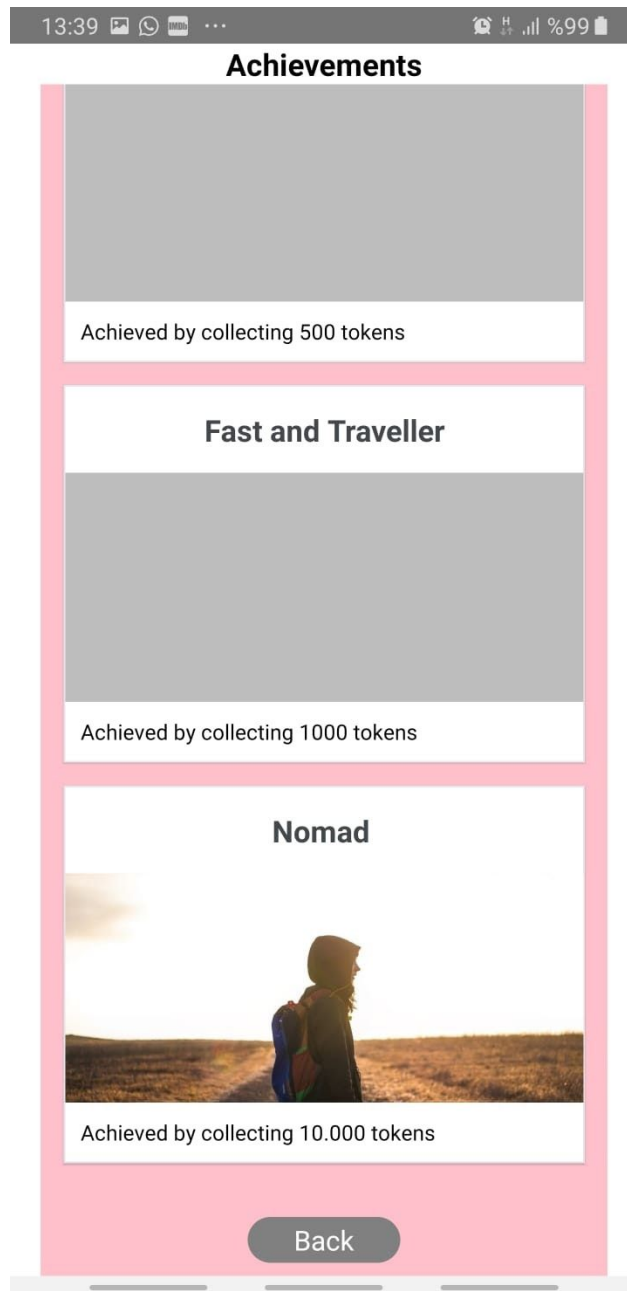
Figure 29: Achievements

Users can see their achievements from the sidebar menu. Achievements can be gained by collecting tokens. For example, to gain the "Nomad" title, a user needs to collect 10.000 tokens.

9.2.11. Settings
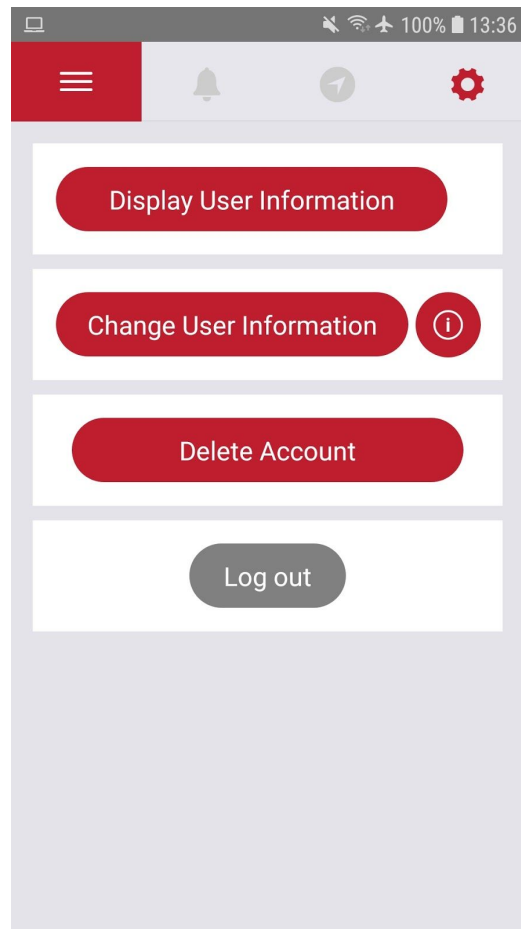
Figure 30: Settings

From settings page users can see their info, change their info or delete their accounts. Also users can log out from the settings panel.

9.2.12. Change User Information

Figure 31: Set Password & Change Mail

Users can see a password for their accounts using change user information. They should set a password with at least six characters long including numbers and letters. They can also change their emails, which are taken from their social media accounts.
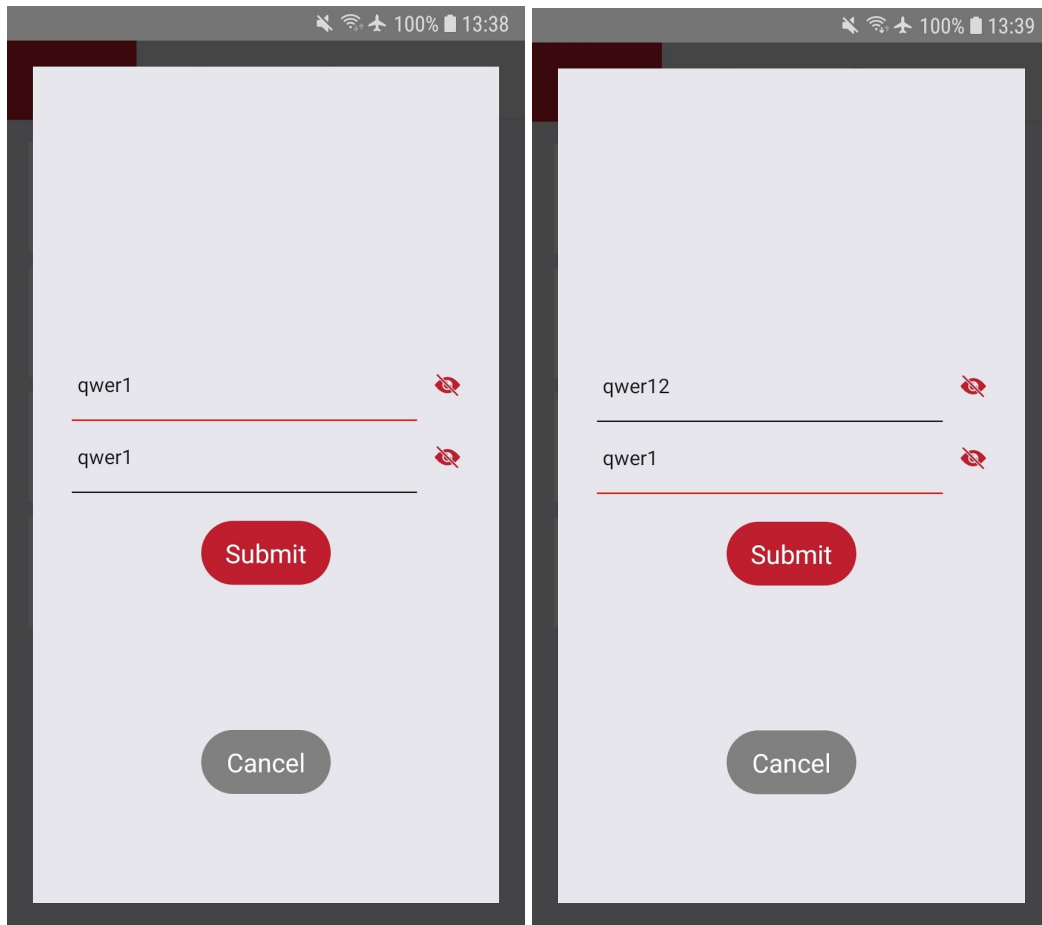
Figure 32: Set Password Screen

Users should set a password by typing the password twice. If they type one password different there will be an error and cannot submit that password. Also, they can see what they write using the eye icon.
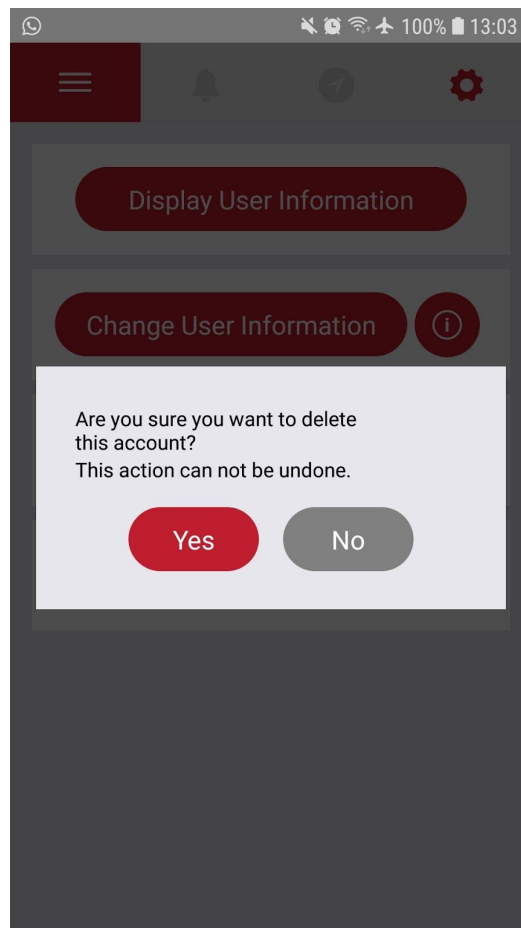
9.2.13. Delete Account

Figure 33: Delete Account

Users can delete their account from their settings page. This action will delete the user from our database and cannot be undone.

10. References

[1] J. Fund, "Travel Is So Much Better Than It Was," National Review, 02-Jan-2017. [Online]. Available: www.nationalreview.com/2017/01/international-travel-today-much-easier-cheaper-it-was/. [Accessed: 12-Oct-2019].