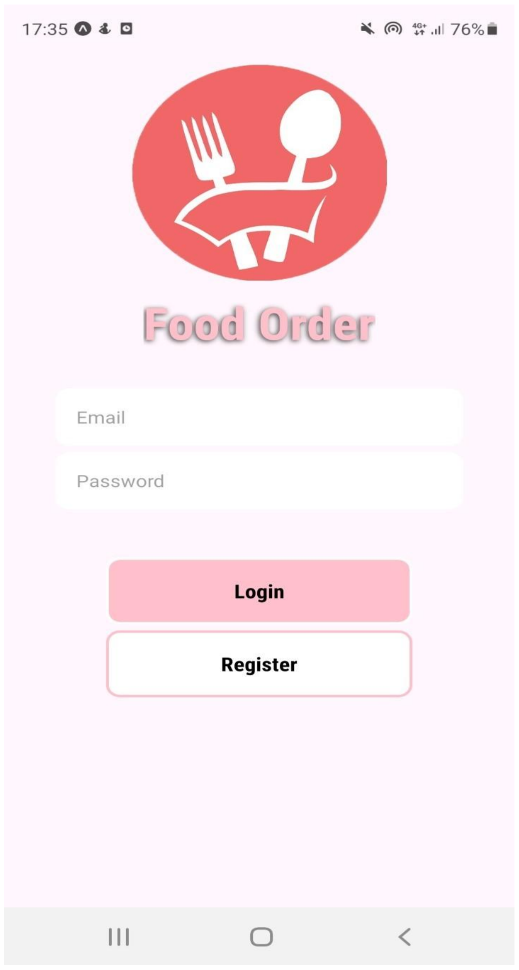
Documentation

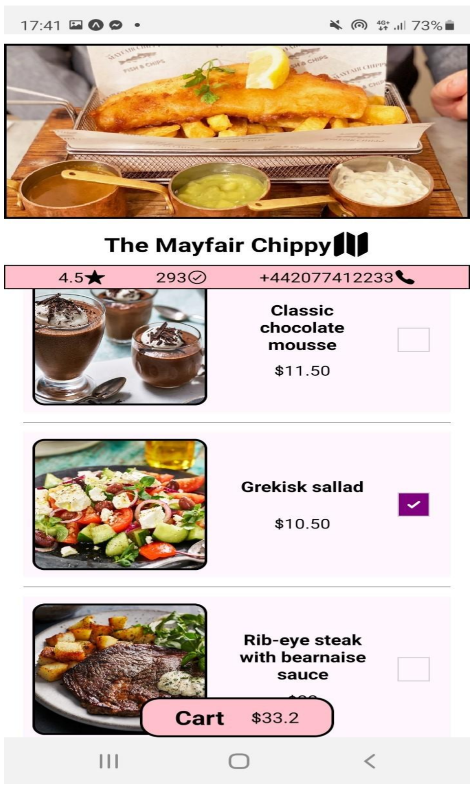


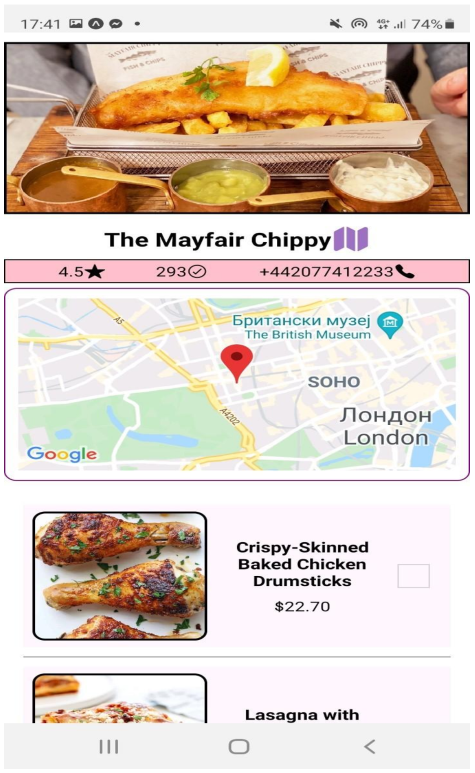
Food Order

Food Order is a mobile application made with React Native as front-end technology and Firebase as Backend-as-a-Service (BaaS).

About the application

Its purpose is to allow the end users to search for restaurants around the world and order food in the restaurant they want. The user registers or logs into the app. If the user enters invalid email or password, an appropriate message is shown.



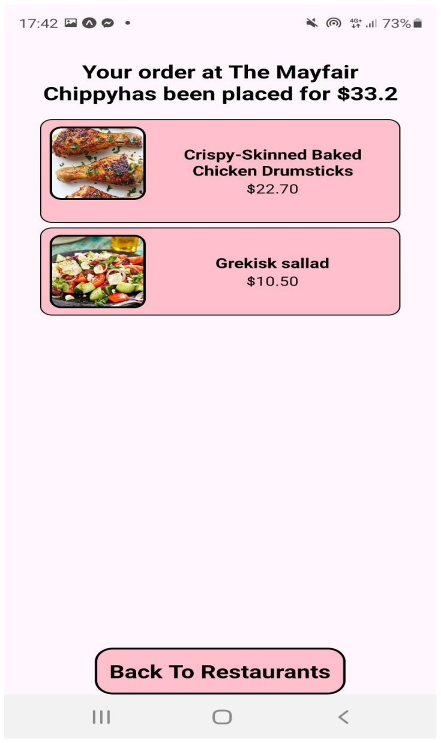


The interface allows the user to search for the city he wants. After that the system shows a list of restaurants in that city. On the home screen, for each restaurant is shown a picture of it, the name of the restaurant, rating and review count. The user can also sort the list of restaurants by reviews count or rating value.

After selecting a restaurant, the user is taken to another screen. In this screen for the restaurant is shown an image, the information from the previous screen and phone number. Also, we have access to the location of the restaurant. Below the restaurant information, the application shows food that the user can order.

The food is hard coded into the app since Yelp API does not offer food for each restaurant. The user can select the food that he wants to order. The total price is shown on a button at the bottom after selecting food.

The user can proceed with the checkout after selecting all the food he wants to order. After checkout the user can return to the restaurant's screen. Also in the top right corner, the user can see all his orders.

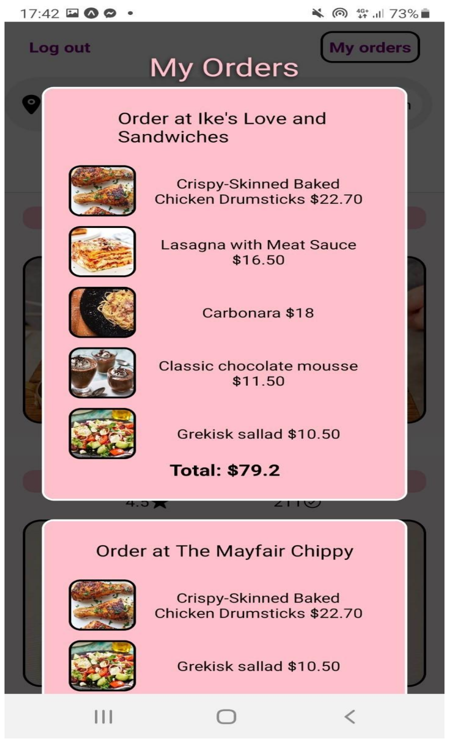


17:42

Your order at The Mayfair Chippyhas been placed for $33.2

Crispy-Skinned Baked Chicken Drumsticks
$22.70

Grekisk sallad
$10.50



17:42

Order at Ike's Love and Sandwiches

Crispy-Skinned Baked Chicken Drumsticks $22.70

Lasagna with Meat Sauce $16.50

Carbonara $18

Classic chocolate mousse $11.50

Grekisk sallad $10.50

Total: $79.2

Order at The Mayfair Chippy

Crispy-Skinned Baked Chicken Drumsticks $22.70

Grekisk sallad $10.50

Why React Native?

React Native is an open-source JavaScript framework, designed for building apps on multiple platforms like iOS, Android, and web applications, utilizing the very same code base. It is based on React, and it brings all its glory to mobile app development.

Both frameworks: ReactJS (web) and React Native were brought to life by Facebook. React Native was a Hackathon project aimed at solving the company's biggest pain point - maintaining two code bases for their app. The problem with maintaining two code bases for such a big app. Work duplication and, at times, solving the same problem in two different ways. React Native is a straightforward answer to these problems.

React Native uses JavaScript to compile the app's user interface but using native-OS views. For more complex features, it allows code implementation in OS-native languages (Swift and Objective-C for iOS, and Java and Kotlin for Android).

Why firebase instead of database?

The difference between Firebase and MySQL comes down to how a company handles and manages its data.
The predefined schemas of MySQL determine the structure of data, and this can be a bit restrictive. But the platform also makes manipulating and defining complex data a simple process.
Compared to Firebase, MySQL is better for multi-row transactions. On the other hand, Firebase is a satisfactory choice when it comes to managing huge data sets because NoSQL horizontally scales data, and it is much faster than MySQL.
Also, Firebase Realtime database can easily come into game if we need user sync in the future. Firebase is suitable for real time applications.

How does it work?

The application has four main screens (Login, Home, RestaurantDetails, Order). Initially the login screen is shown to the user. For registration the following function is called.

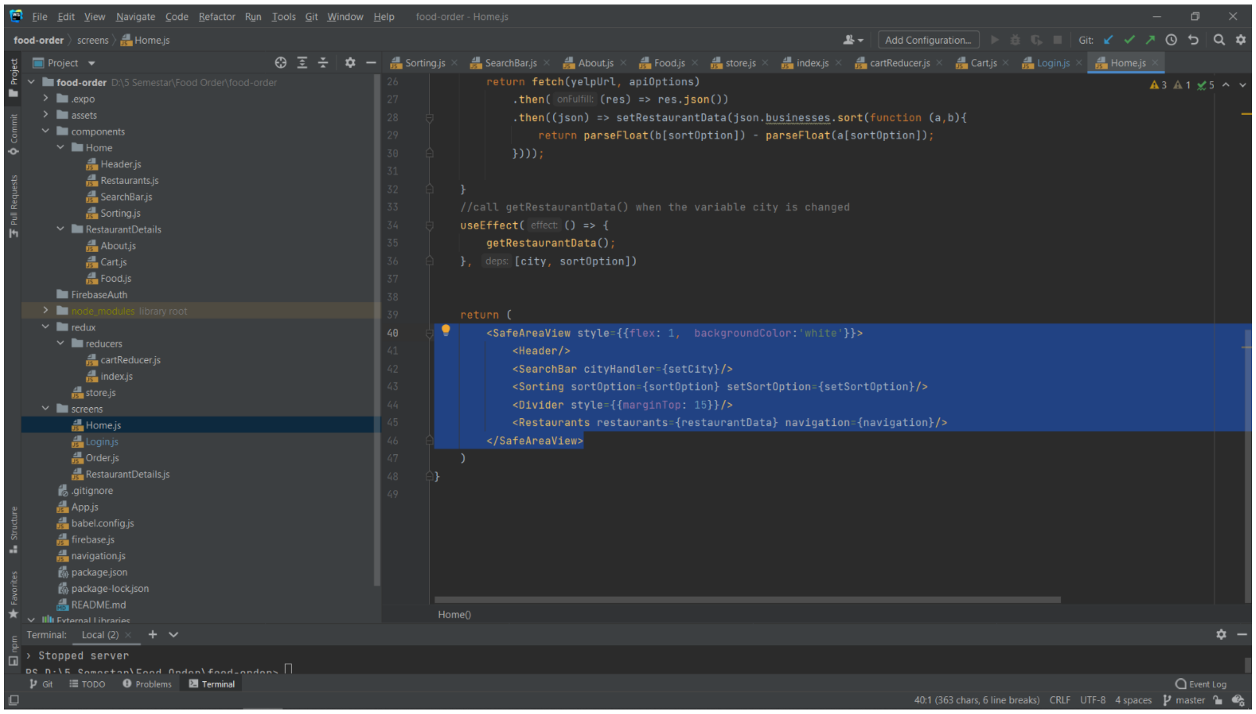
```
const handleSignUp = () => {
   auth
       .createUserWithEmailAndPassword(email, password)
       .then(userCredentials => {
           const user = userCredentials.user;
           console.log('Registered with:', user.email);
       })
       .catch(error => alert(error.message))
}
```

If the user email is not taken and the password satisfies the complexity, a call to the firebase is made and the user is stored. **The password is hashed.**

For the login part we have the following code.

```
const handleLogin = () => {
    auth
        .signInWithEmailAndPassword(email, password)
        .then(userCredentials => {
            const user = userCredentials.user;
            console.log('Logged in with:', user.email);
        })
        .catch(error => alert(error.message))
}
```

The main screens are made of components or parts. For example the Home screen (Home.js) is made of Header, Restaurants, SearchBar and Sorting parts.



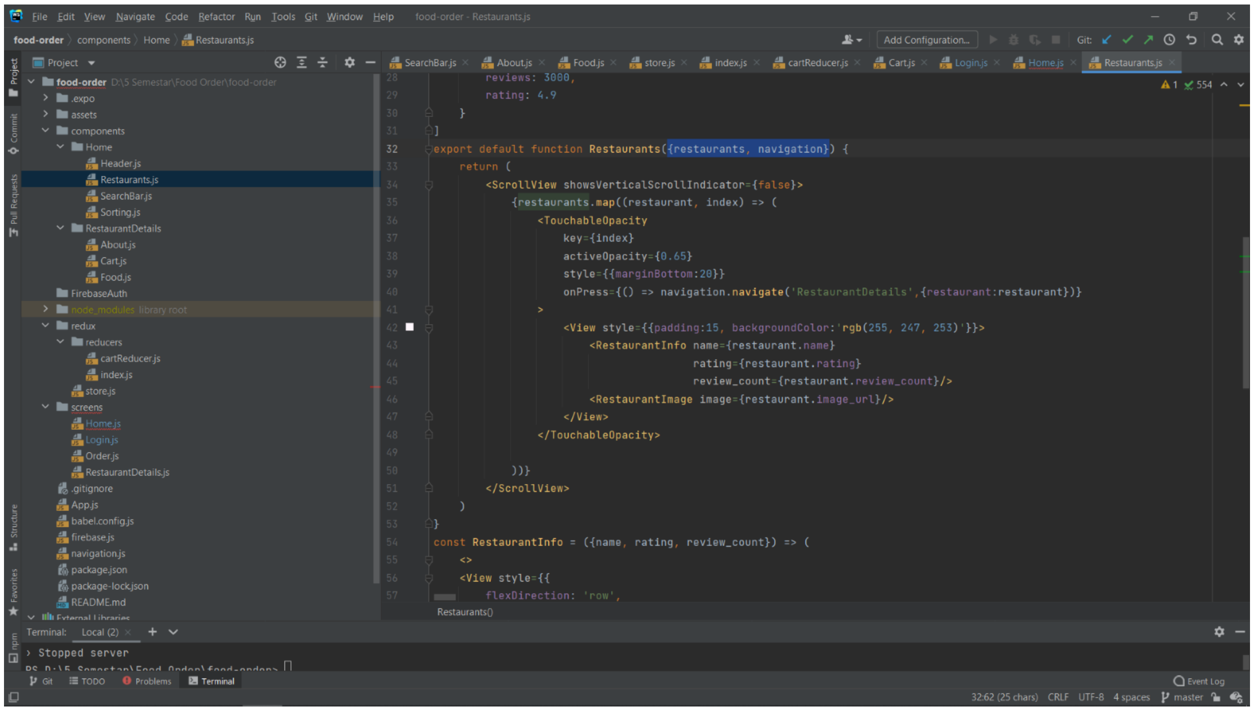
```
        return fetch(yelpUrl, apiOptions)
            .then( onFulfill: (res) => res.json())
            .then((json) => setRestaurantData(json.businesses.sort(function (a,b){
                return parseFloat(b[sortOption]) - parseFloat(a[sortOption]);
            })));

    }
    //call getRestaurantData() when the variable city is changed
    useEffect( effect: () => {
        getRestaurantData();
    },  deps: [city, sortOption])


    return (
        <SafeAreaView style={{flex: 1,  backgroundColor:'white'}}>
            <Header/>
            <SearchBar cityHandler={setCity}/>
            <Sorting sortOption={sortOption} setSortOption={setSortOption}/>
            <Divider style={{marginTop: 15}}/>
            <Restaurants restaurants={restaurantData} navigation={navigation}/>
        </SafeAreaView>
    )
}
```

```
> Stopped server
PS D:\5 Semestar\Food Order\food-order>
```

These components are imported at the top of the JavaScript file. The components are also wrapped in SafeAreaView, because this doesn't allow the content to be shown in the inaccessible areas.
The application uses navigation to switch from one screen to another. We can also pass data from one screen to another. For example, we can see that in the Restaurant component we are passing restaurants and navigation.

```
import React from 'react';

import {NavigationContainer} from "@react-navigation/native";
import {createStackNavigator} from "@react-navigation/stack";
import Home from "./screens/Home";
import RestaurantDetails from "./screens/RestaurantDetails";
import Order from "./screens/Order";
import {Provider as ReduxProvider} from 'react-redux'
import configureStore from "./redux/store";
import Login from "./screens/Login";

const store = configureStore();

export default function RootNavigation() {
   const Stack = createStackNavigator();

   const screenOptions = {
       headerShown: false,
   };
   return (
       <ReduxProvider store={store}>
           <NavigationContainer>
               <Stack.Navigator initialRouteName="Login"
screenOptions={screenOptions}>
                   <Stack.Screen name="Login" component={Login} />
                   <Stack.Screen name='Home' component={Home}/>
                   <Stack.Screen name='RestaurantDetails'
component={RestaurantDetails}/>
                   <Stack.Screen name='Order' component={Order}/>
               </Stack.Navigator>
           </NavigationContainer>
       </ReduxProvider>
   );
}
```

For the navigation part we can see that each screen is wrapped in <NavigationContainer> and initialRouteName is Login screen. Also everything is wrapped inside ReduxProvider (redux is described later).



```
        reviews: 3000,
        rating: 4.9
    }
]
export default function Restaurants({restaurants, navigation}) {
    return (
        <ScrollView showsVerticalScrollIndicator={false}>
            {restaurants.map((restaurant, index) => (
                <TouchableOpacity
                    key={index}
                    activeOpacity={0.65}
                    style={{marginBottom:20}}
                    onPress={() => navigation.navigate('RestaurantDetails',{restaurant:restaurant})}
                >
                    <View style={{padding:15, backgroundColor:'rgb(255, 247, 253)'}}>
                        <RestaurantInfo name={restaurant.name}
                                        rating={restaurant.rating}
                                        review_count={restaurant.review_count}/>
                        <RestaurantImage image={restaurant.image_url}/>
                    </View>
                </TouchableOpacity>

            ))}
        </ScrollView>
    )
}
const RestaurantInfo = ({name, rating, review_count}) => (
    <>
    <View style={{
        flexDirection: 'row',
```

```
> Stopped server
```

Here we are receiving the data. We are fetching the restaurants using the getRestaurantData function in Home.js . Every time we select a different city in the search bar, this function is getting called and we are getting different restaurants. Also, the restaurants are sorted depending on which sort of option is selected. The sort of option is stored in state.

```
const [sortOption, setSortOption] = useState(sortingOption[0]);
const getRestaurantData = () => {
   const yelpUrl =
`https://api.yelp.com/v3/businesses/search?term=restaurants&location=${city}`;
   const apiOptions = {
       headers: {
           Authorization: `Bearer ${YELP_API_KEY}`
       },
   }
   return fetch(yelpUrl, apiOptions)
       .then((res) => res.json())
       .then((json) => setRestaurantData(json.businesses.sort(function (a,b){
           return parseFloat(b[sortOption]) - parseFloat(a[sortOption]);
       })));

}
//call getRestaurantData() when the variable city is changed
useEffect(() => {
   getRestaurantData();
}, [city, sortOption])
```

With the help of the `GooglePlacesAutocomplete` the autocomplete in the search bar is achieved. The sorting option is stored in state.

This is the Json that the Yelp API call returns.

```
GET
https://api.yelp.com/v3/autocomplete?text=del&latitude=37.786882&longitude
=-122.399972
```

```
{
  "total": 1316,
  "businesses": [
    {
      "rating": 4.5,
      "price": "$$",
      "phone": "+14154212337",
      "id": "molinari-delicatessen-san-francisco",
      "categories": [
        {
          "alias": "delis",
          "title": "Delis"
        }
      ],
      "review_count": 910,
      "name": "Molinari Delicatessen",
      "url":
"https://www.yelp.com/biz/molinari-delicatessen-san-francisco",
      "coordinates": {
        "latitude": 37.7983818054199,
        "longitude": -122.407821655273
      },
      "image_url":
"http://s3-media4.fl.yelpcdn.com/bphoto/6He-NlZrAv2mDV-yg6jW3g/o.jpg",
      "location": {
        "city": "San Francisco",
        "country": "US",
        "address2": "",
        "address3": "",
        "state": "CA",
        "address1": "373 Columbus Ave",
        "zip_code": "94133"
      }
    },
    // ...
  ]
}
```

After selecting a restaurant, the restaurant data is passed to another screen (RestaurantDetails.js).

```
import React from 'react';

import {View, Text, StatusBar} from 'react-native';
import About from "../components/RestaurantDetails/About";
import Food from "../components/RestaurantDetails/Food";
import Cart from "../components/RestaurantDetails/Cart";

export default function RestaurantDetails({route, navigation}) {
   return (
       <View style={{flex:1, marginTop:StatusBar.currentHeight,
backgroundColor:'white'}}>
           <About restaurant={route.params.restaurant}/>
           <Food restaurantName={route.params.restaurant.name} />
           <Cart navigation={navigation}/>
       </View>
   )
}
```

The restaurant data is placed inside the route that we passed earlier. So for the restaurant we have the About, Food and Cart components. In the About component we have the details for the restaurant, in the Food component we have a list of the food (hard coded) and the Cart component is shown when we click on the Cart button after ordering food.
The Map is placed inside the About section.

```
const Map = ({region}) => (
   <View style={{padding:10,borderColor:'purple', borderWidth:
1,marginTop:5,borderRadius:10}}>
       <MapView style={{width:'100%',height:180}}
        initialRegion={region}
       >
           <Marker coordinate={region} />
       </MapView>
   </View>

)
```

For the ordering part, we are storing the food and the restaurant name inside the redux store.

Redux is a javascript library made to help you manage the state of your application. It does that by providing the developer with a centralized place called the store, where the state is saved and modified through actions and reducers.

Redux helps you deal with shared state management, but like any tool, it has tradeoffs. It's not designed to be the shortest or fastest way to write code. It's intended to help answer the question "When did a certain slice of state change, and where did the data come from?", with predictable behavior. There are more concepts to learn, and more code to write. It also adds some indirection to your code, and asks you to follow certain restrictions. It's a trade-off between short term and long term productivity.

```
let defaultState = {
   selectedItems: {items: [], restaurantName: ''}
}
let cartReducer = (state = defaultState, action) => {
   let newState = {...state};
   switch (action.type) {

       case 'ADD_TO_CART': {
           newState.selectedItems = {
               items: [...newState.selectedItems.items, action.payload],
               restaurantName: action.payload.restaurantName
           };
           return newState;
       }
       case 'REMOVE_FROM_CART': {
           newState.selectedItems = {
               items: [
                   ...newState.selectedItems
                       .items
                       .filter((item) => item.title !== action.payload.title)
               ],
               restaurantName: action.payload.restaurantName
           }
           return newState;
       }
       case 'CLEAR_CART':{
           newState.selectedItems = {
               items: [],
               restaurantName: ''
           };
           return newState;
       }
       default:
           return state;
   }
};
export default cartReducer;
```

The food is wrapped in ScrollView. The ScrollView is used if vertical or horizontal overflow happens so we can scroll and see the rest of the items. The food is hard coded and stored in the menu array.

```
const menu = [
   {
       image_url:
'https://healthyrecipesblogs.com/wp-content/uploads/2020/09/baked-drumsticks-f
eatured-2021.jpg',
       title: 'Crispy-Skinned Baked Chicken Drumsticks',
       price: '22.70'
   },
{
. . .
]
```

In this part of the code we iterate through the menu array and for each food we show the FoodImage, FoodDetails and Checkbox for putting the food on cart.

```
<ScrollView style={{padding: 15}} showsVerticalScrollIndicator={false}>
   {menu.map((food, index) => (
       <View key={index}>
           <View
               style={{flexDirection:
'row',marginTop:10,paddingTop:7,paddingBottom:7,paddingLeft:7,
backgroundColor:'rgb(255, 247, 253)'}}>
               <FoodImage image={food.image_url}/>
               <FoodDetails name={food.title} price={food.price} />
               <BouncyCheckbox iconStyle={{borderColor: 'lightgray',
borderRadius: 0}}
                               onPress={(checkboxValue) => checkboxValue ?
selectItem(food, checkboxValue)
                               : unselectItem(food, checkboxValue)}
                               isChecked={isFoodInCart(food, cartItems)}
                               fillColor='purple'/>
           </View>
           <Divider style={{marginTop:10}} color='black'/>
       </View>
   ))}
</ScrollView>

const FoodDetails = ({name, price}) => (
   <View style={{flexShrink:1,flex:1,padding:20, justifyContent:
'space-evenly', alignItems: 'center'}}>
       <Text style={{fontWeight:'bold',
fontSize:15,textAlign:'center'}}>{name}</Text>
       <Text>${price}</Text>
   </View>
)
```

```
const FoodImage = ({image}) => (
   <View>
       <Image source={{
           uri: image
       }}
              style={{
                  borderColor: 'black',
                  borderRadius:10,
                  borderWidth: 2,
                  height: 135,
                  width: 135
              }}
       />
   </View>
)
```

The BouncyCheckBox has an onPress event that takes the checkboxValue. If the checkbox is already unselected and is clicked then `selectItem(food, checkboxValue)` is called.

```
const dispatch = useDispatch();
const selectItem = (item, checkboxValue) => dispatch({
   type: 'ADD_TO_CART',
   payload: {...item, restaurantName:restaurantName, checkboxValue:
checkboxValue}
});
```

Previously we saw that cartReducer has an action type. After executing the following code, the food is added to the array and newState is returned.

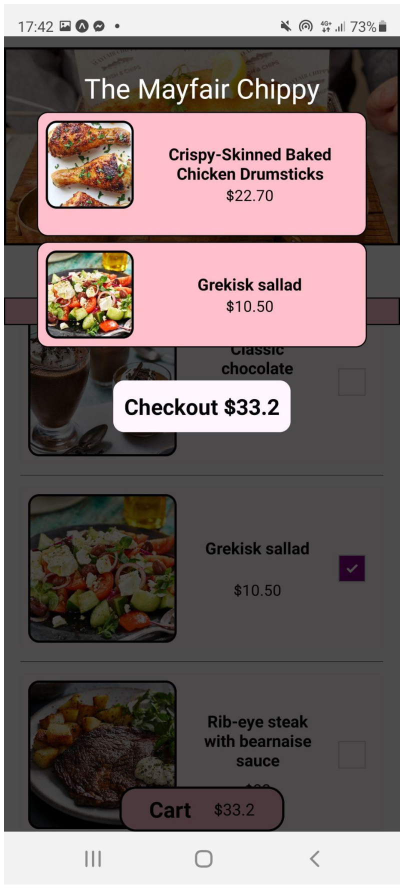
```
case 'ADD_TO_CART': {
           newState.selectedItems = {
               items: [...newState.selectedItems.items, action.payload],
               restaurantName: action.payload.restaurantName
           };
           return newState;
       }
}
```

Previously we saw that cartReducer has an action type. After executing the following code, the food is added to the array and newState is returned.

In the Cart.js we have Cart button and total price. The Cart button is visible if the items array is not empty (we have selected at least one food). The total price is calculated with the next code

```
const {items, restaurantName} = useSelector((state) =>
state.cartReducer.selectedItems)

const calculateTotalPrice = () => items
    .map((item) => Number(item.price))
    .reduce((prev, cur) => prev + cur, 0);

const totalPriceFormatted = '$' + calculateTotalPrice();
```

After clicking the button, the next modal view is shown to the user. After clicking on Checkout, the order is stored into the firebase and the cart is cleared.



```
const addOrderToFirebase = () => {

    const db = firebase.firestore();

    db.collection('orders').add({
        userEmail: userName,
        items:items,
        restaurantName: restaurantName,
        createdAt:
firebase.firestore.FieldValue.serverTimestamp(),
    })
    setModalVisible(false);
    navigation.navigate('Order', {
        navigation: navigation,
        items: items,
        restaurantName: restaurantName,
        totalPrice: totalPriceFormatted
    })
    clearCart();

}
```

For every order we store the email of the user, items, restaurantName and creation date.

In the home screen there is a “My Orders” button.

```
const [myOrdersVisible, setMyOrdersVisible] = useState(false);
const [ordersArray, setOrdersArray] = useState([])
```

We use state for the modal visibility and the orders. Getting all the orders that the current user made with the following code

```
firebase.firestore()
   .collection('orders')
   .where("userEmail","==",userEmail===undefined ? "" : userEmail)
   .get()
   .then(querySnapshot => {
       let array = [];
       querySnapshot.forEach(documentSnapshot => {
           array.push(documentSnapshot.data());
       });
       setOrdersArray(array);
   });
```

After we fill the ordersArray, we can display the list of the orders with the list of food in each order, food description, price and total price of the order.

For this application the following libraries are installed and used:

npm install react-native-google-places-autocomplete
npm install react-native-vector-icons
npm install react-native-elements
npm install @react-navigation/native
npm install @react-navigation/stack
npm install react-native-gesture-handler
npm install react-native-bouncy-checkbox
npm install tslib
npm install react-redux
npm install @react-navigation/native-stack
npm install react-native-reanimated
npm install reanimated-bottom-sheet
npm install lottie react-native

Conclusion

Create great mobile apps

React Native is great for mobile apps. It provides a slick, smooth and responsive user interface, while significantly reducing load time. It's also much faster and cheaper to build apps in React Native as opposed to building native ones, without the need to compromise on quality and functionality.

Use existing skills

Since React Native is based on JavaScript, your developers will not need long to get to grips with it. Having said this, it's an open-source and community-driven framework, so if they ever need support, it's widely available online.

Add third-party plugins

React Native also allows you to easily incorporate third-party plug-ins and APIs, including maps and payment systems.

Final thoughts

So what should you know about React Native? Here are the key takeaways:

- **If you’re using the Facebook or Airbnb mobile app, you’re using React Native without even knowing it.**
- **React Native apps are easy to write, saving time for developers and cutting costs for project managers.**
- **React Native apps lower your development and maintenance costs, because you don’t have to deal with two separate codebases for iOS and Android.**
- **Since React Native is just a wrapper for native components, there’s nothing stopping you from adding native Java or Swift code where you need it.**
- **At the end of the day, you’re still coding in JavaScript. There’s no need to learn Swift/Java, or to add developers with such skills to your roster.**
- **React Native is growing fast with no signs of stopping.**