Mobilná aplikácia Eshop

Semestrálna práca

Vývoj aplikácií pre mobilné zariadenia

Dokumentácia

Autor : Ján Dunaj

Obsah

1 Popis práce

Témou semestrálnej práce je multiplatformový **eshop** so základnou funkcionalitou. V eshope je možné prihlásiť sa alebo registrovať sa, prehľadávať vytvorené produkty , pridávať produkty do košíka, vytvárať objednávky a aj upravovať samotné produkty, t.j. vytvoriť, upraviť a vymazať. Navyše aplikácia disponuje systémom **auto-login** a **auto-logout**.

Za vzor semestrálnej práce som nevolil konkrétnu aplikáciu, nakoľko vytvoriť plnohodnotný eshop je časovo a technicky náročné. Avšak primárnym dôvodom výberu témy bol osobný záujem do tejto oblasti, nakoľko som si chcel rozšíriť poznatky a porovnať tak vývoj mobilnej verzie eshopu oproti vývoju eshopu pomocou webových technológií.

Aplikácia je **user-friendly**, jednoduchá na používanie, pre platformu Android a iOS. Obsahuje iba **základnú funkcionalitu** eshopu.

2 Návrh riešenia

2.1 Popis riešenia

Mobilná aplikácia je vytvorená v **React Native** pomocou knižnice Expo. Využíva **Firebase** databázu na ukladanie používateľov, objednávok a produktov. Aplikácia je robená, tak aby bola spustiteľná aj na Androide (API 23+) aj na iOS.

Aplikácia využíva **centrálne úložisko** telefónu, ktoré využíva na uloženie tokenu a času expirácie prihláseného užívateľa pre auto-login a auto-logout systém.

Aplikácia je riešená tak, že odchytáva výnimky ak nastane v aplikácii chyba , napr. chyba pri komunikácii s DB, chyba pri zápise do úložiska, apod.

2.2 Hlavné užívateľské časti aplikácie

Aplikácia obsahuje hlavné (Drawer) menu z 3 navigátorov, ktoré prepájajú užívateľa medzi hlavnými obrazovkami aplikácie.

1. **Navigátor produktov** – prístup všetkým
    a. Obrazovka všetkých dostupných produktov
    b. Obrazovka detailov vybraného produktu produktu
    c. Obrazovka košíka
2. **Navigátor objednávok** – prihlásený
    a. Obrazovka vytvorených objednávok
3. **Navigátor užívateľov** - prihlásený
    a. Obrazovka produktov vytvorených prihláseným užívateľom
    b. Obrazovka editovania/vytvárania produktu užívateľa
4. **Navigátor prihlasovania** – iba ak neprihlásený
    a. Obrazovka na prihlásenie/registrovanie

2.3 Úložisko aplikácie

Aplikácia využíva **Redux store**, v ktorom uchováva produkty, košík, objednávky a prihláseného užívateľa a databázové úložisko Firebase. Redux vykonáva celú komunikáciu s databázou.

Štruktúra úložiska:

- Produkty
    - Všetky dostupné produkty
    - Produkty vytvorené používateľom
- Košík – využíva iba Redux store
    - Produkty v košíku – produkt, množstvo, cena

    - Celková cena
- Objednávky
    - Objednávky vytvorené používateľom
- Prihlásený užívateľ – využíva iba Redux store
    - Token
    - Id používateľa

Pri ukladaní užívateľa využíva aplikácia aj centrálne úložisko telefónu.

2.3.1 Authentifikácia

Má na starosti menežovanie užívateľov.

Action:

- Obsahuje metódy na prihlásenie/registrovanie/odhlásenie užívateľov a komunikáciu s DB. Menežuje auto-login a auto-logout systém.

Reducer :

- **AUTHENTICATE –** uloženie prihláseného užívateľa
- **LOGOUT –** odhlásenie

2.3.2 Produkty

Má na starosti menežovanie produktov.

Action:

- Metódy na mazanie, editovanie, vytváranie, získanie produktov z DB.

Reducer:

- **CREATE_PRODUCT –** vytvorí nový produkt do úložiska ako kópiu z DB
- **DELETE_PRODUCT –** vymaže produkt z úložiska
- **SET_PRODUCTS –** pridá získané produkty z DB do úložiska
- **UPDATE_PRODUCT –** upraví produkt

2.3.3 Objednávky

Menežuje objadnávky.

Action:

- Metódy na získanie objednávok , vytvorenie novej objednávky pre prihláseného užívateľa. Komunikuje s DB.

Reducer:

- **ADD_ORDER –** vytvorí objednávku do úložiska ako kópiu z DB
- **SET_ORDERS –** uloží načítané objednávky z DB do úložiska

2.3.4 Košík

Menežovanie košíka

Aciton:

- Metódy na pridanie/vymazanie produktu z košíka aj zresetovanie košíka ak prebieha tvorenie objednávky. Nekomunikuje s DB nakoľko košík DB neukladá.

Reducer:

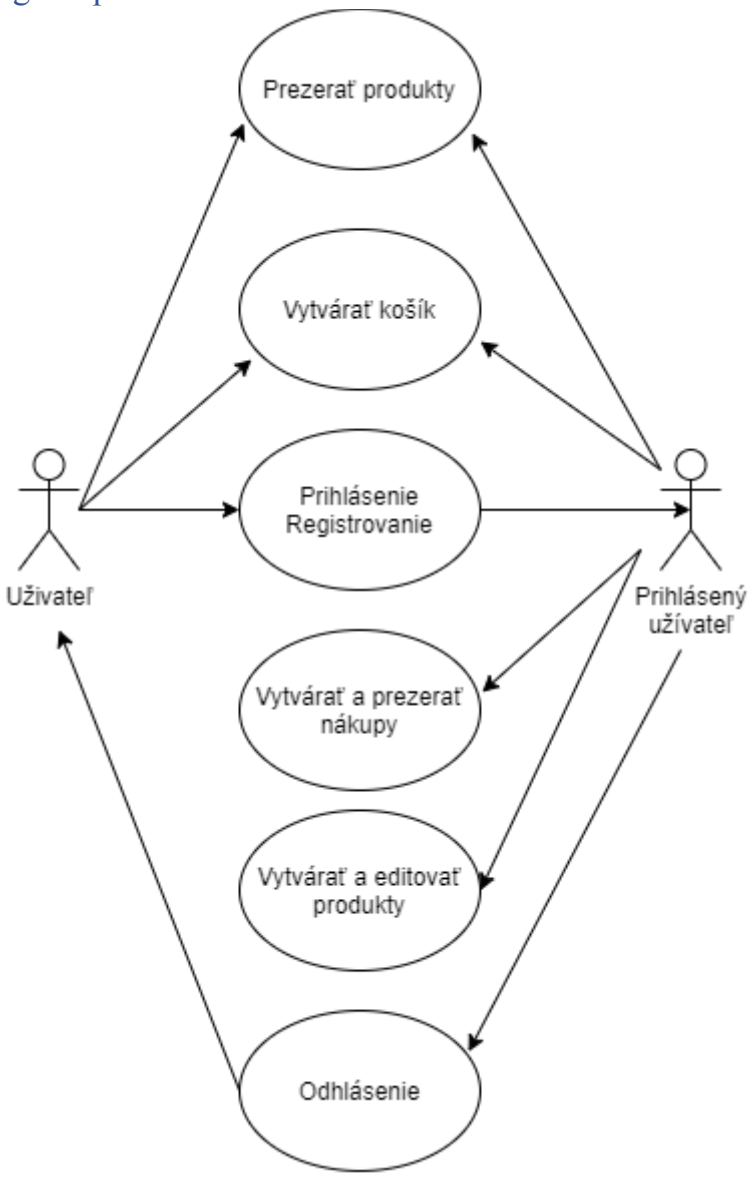
- **ADD_TO_CART –** pridanie produktu do košíka
- **REMOVE_FROM_CART –** vymazanie produktu z košíka
- **ADD_ORDER –** zresetovanie celého košíka

2.4 Dátové modely

Aplikácia využíva 3 dátové modely. Modely využíva Redux store pri ukladaní do úložiska.

- Product – id, ownerId, title, imageUrl, price, description
- Order – id, items, total, date
- CartItem – quantity, productPrice, productTitle, sum
    - Využíva ho košík

2.5 Stavový diagram použitia

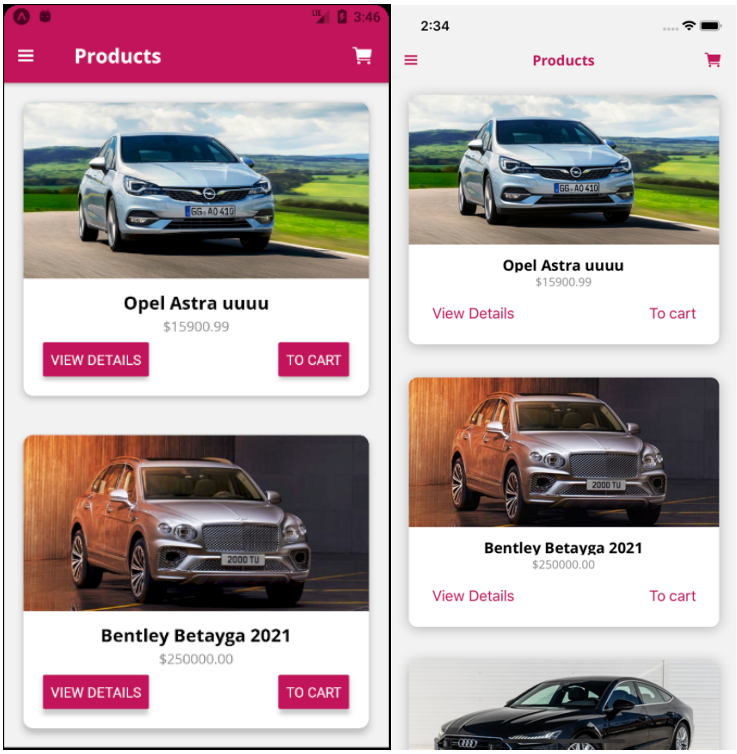


2.6 Obrazovky

Grafická reprezentácia obrazoviek aplikácie sú zobrazené na Android a iOS emulátore.

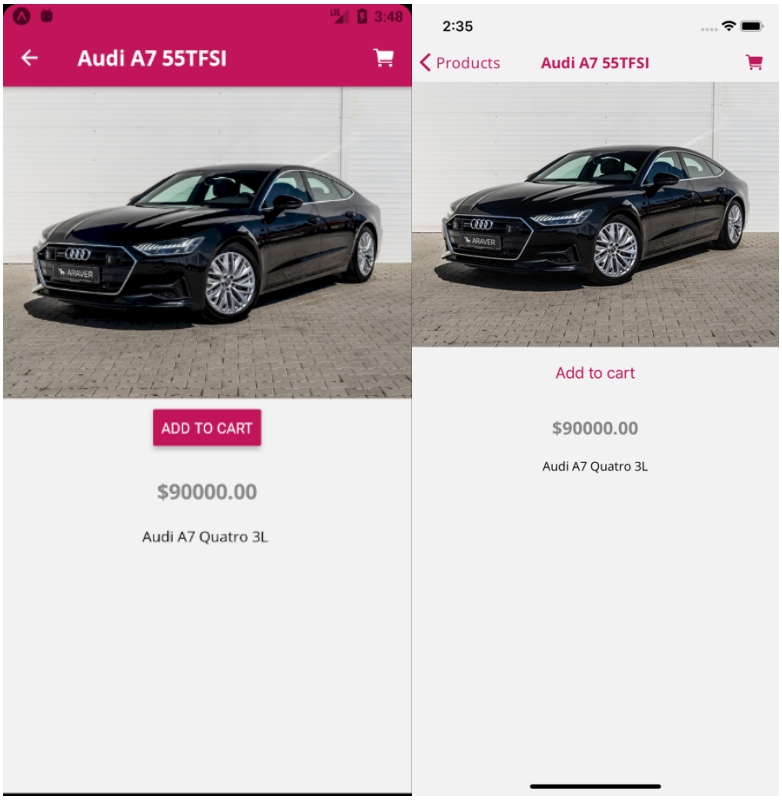
2.6.1 Obrazovka všetkých dostupných produktov

Je to základna obrazovka. Obsahuje všetky vytvorené produkty. Je prístupná všetkým užívateľom.



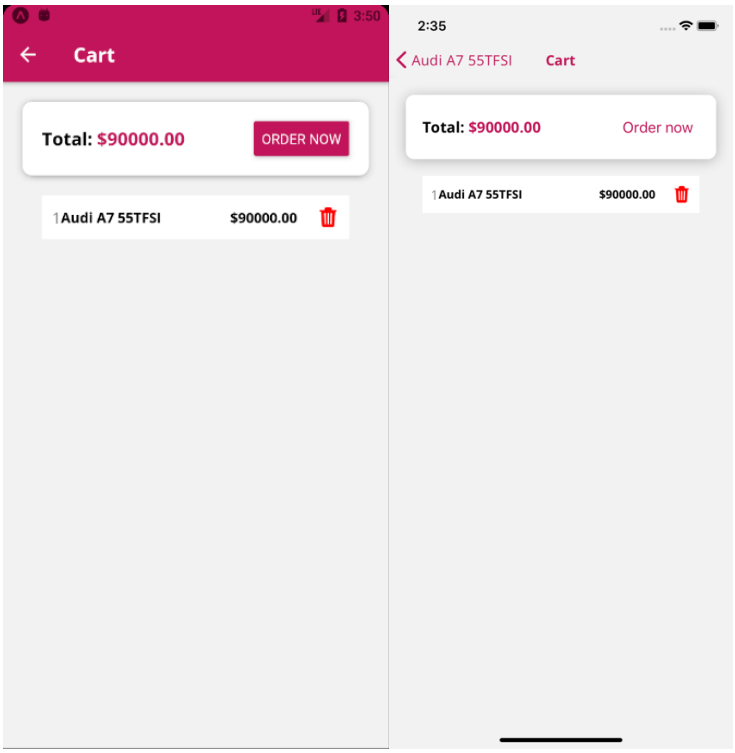
2.6.2 Obrazovka detailov vybraného produktu produktu

Obrazovka vykresľuje detaily produktu, na ktorý užívateľ klikol. Je dostupná pre všetkým.



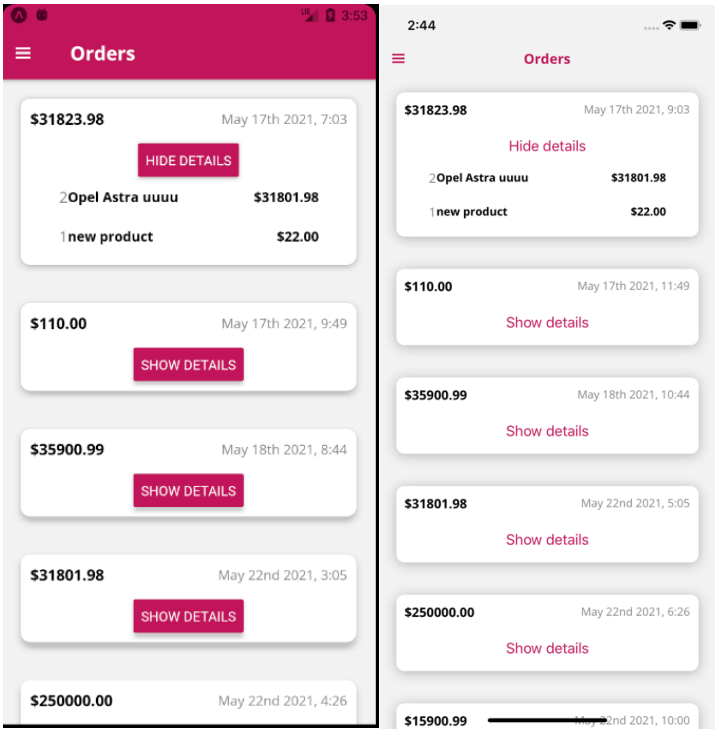
2.6.3 Obrazovka košíka

Obsahuje informácie o košíku a produktoch čo sú v ňom. Košík je dostupný všetkým, ale vytvoriť objednávku môže len prihlásený užívateľ. Neprihláseného nasmeruje na prihlasovaciu obrazovku.



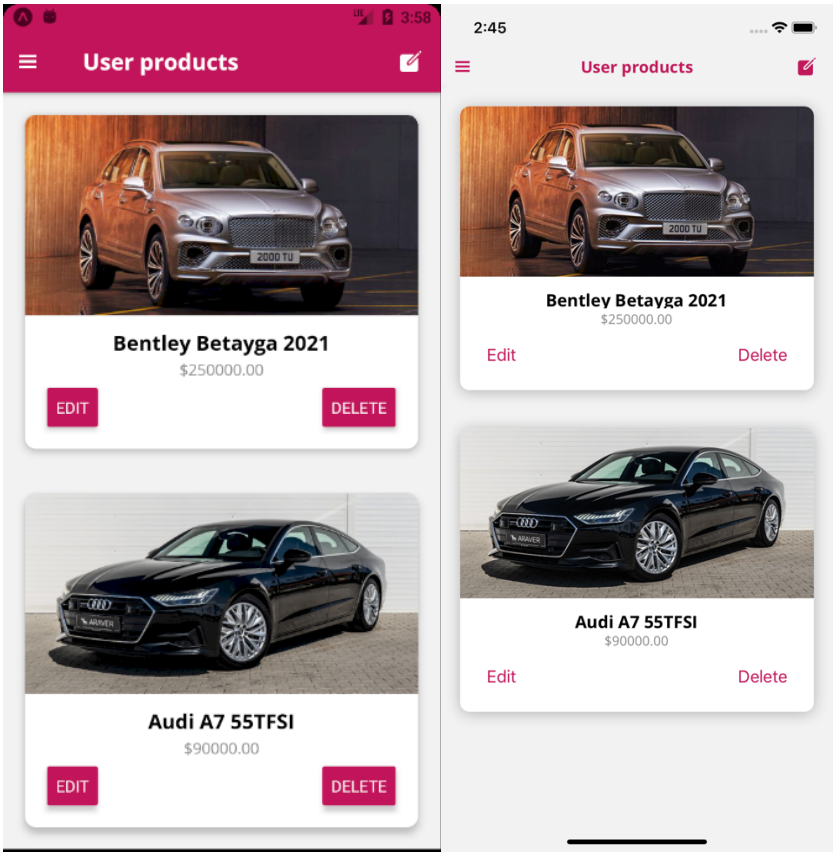
2.6.4 Obrazovka vytvorených objednávok

Obsahuje informácie o objednávkach prihláseného užívateľa. Je dostupná iba pre prihláseného užívateľa.



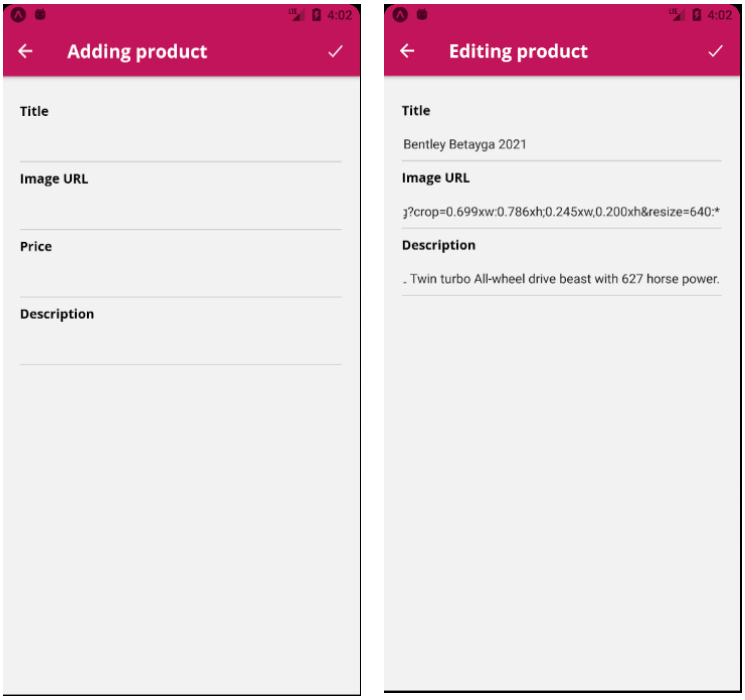
2.6.5 Obrazovka produktov vytvorených prihláseným užívateľom

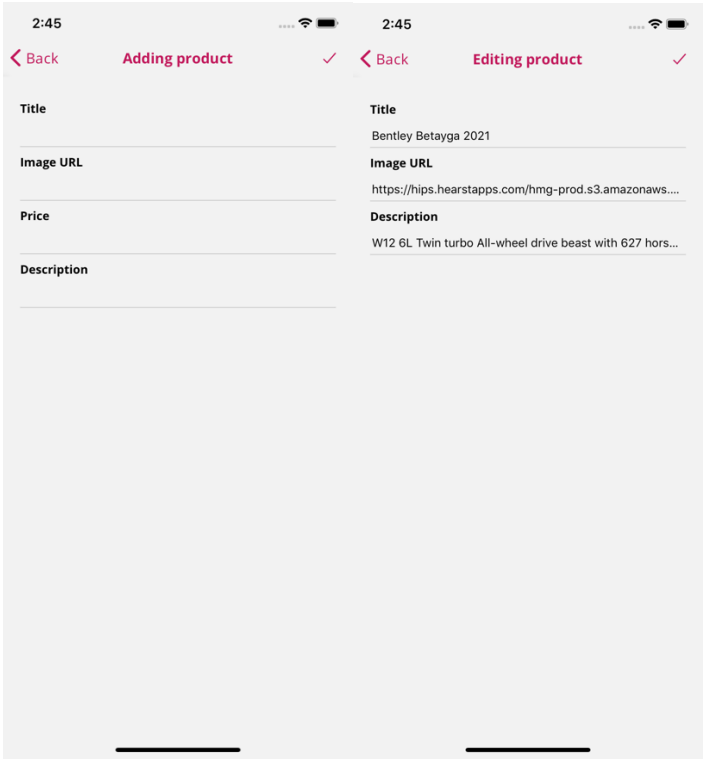
Obsahuje iba produkty, ktoré vytvoril užívateľ. Je podobná s obrazovkou pre všetky produkty.



2.6.6 Obrazovka editovania/vytvárania produktu užívateľa

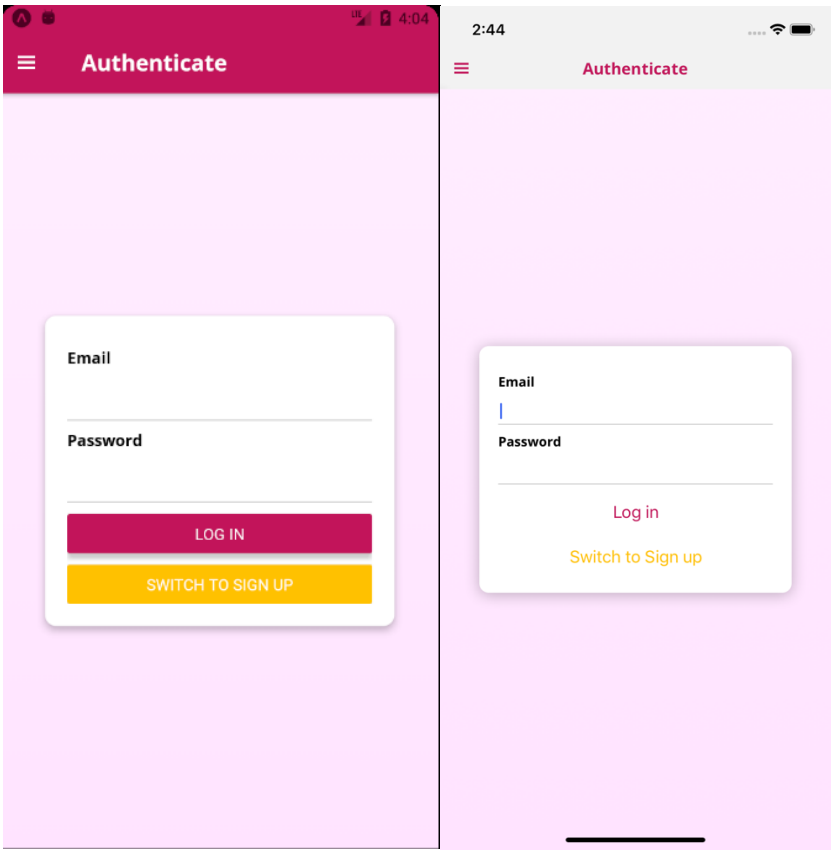
Obsahuje formulár, ktorým sa vytvára nový produkt. Formulár musí byť valídny. Obrazovka sa využíva aj na editovanie aj na vytváranie produktu. Pri editovaní sa predvyplnia polia už dátami existujúceho produtku.





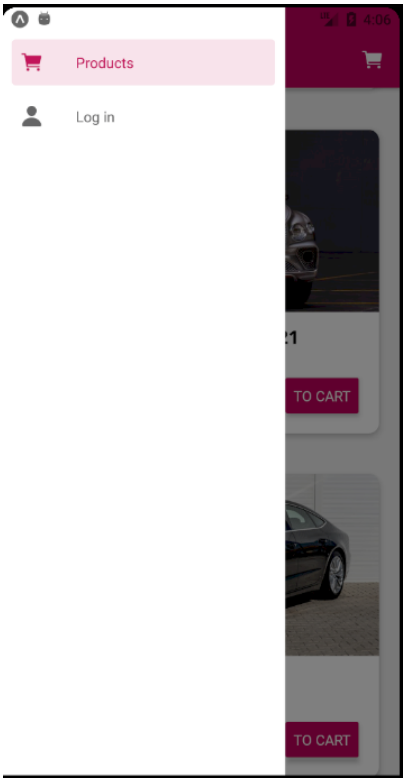
2.6.7 Obrazovka na prihlásenie/registrovanie

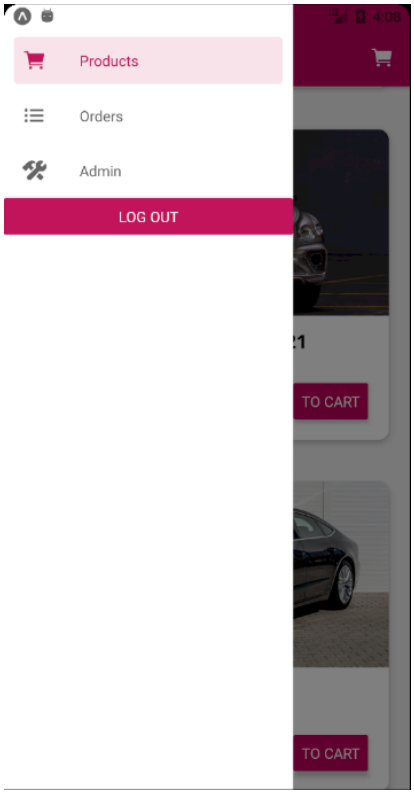
Obsahuje formulár na prihlásenie alebo registráciu používateľa.

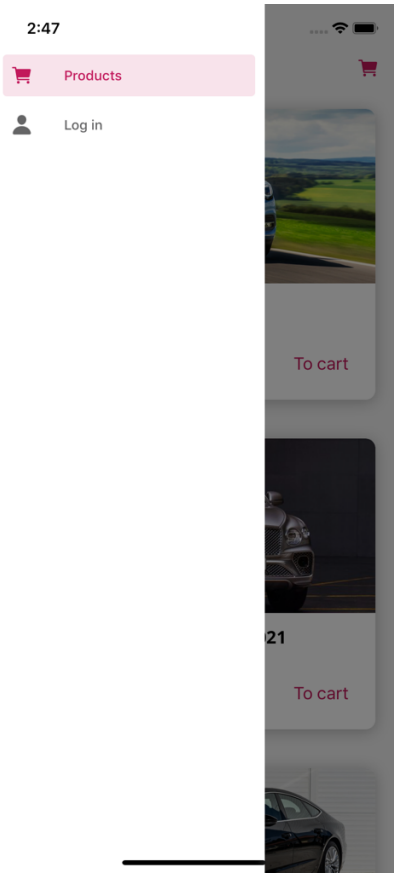


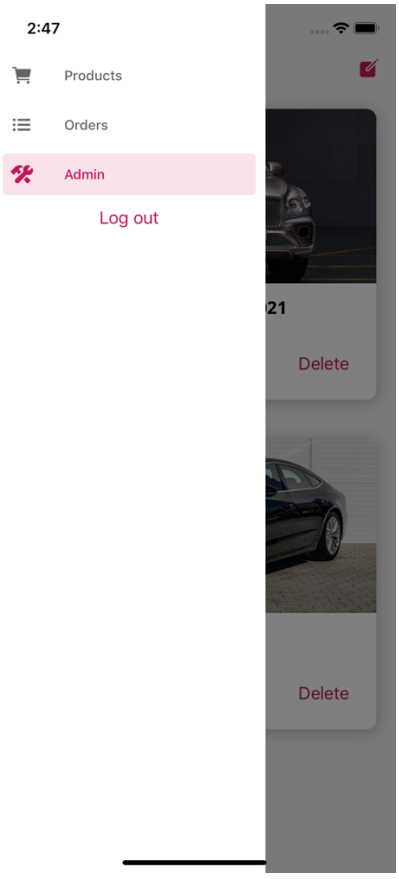
2.7 Navigácia

Takto vyzerá bočná navigácia pred prihlásením a po prihlásení.









3 Implementácia

Aplikácia bola vytvorená v prostredí Visual Studio Code pomocou knižnice Expo na vývoj React Native aplikácií. Pri programovaní som používal Android (API 23 a 30) emulátor, iOS emulátor a reálne zariadenie (Xiaomi Redmi Note 7, Xiaomi Redmi Note 9, iPhone XS max, Samsung Galaxy A20e).

3.1 Štruktúra programu

V programe som zvolil rozdelenú štruktúru na časti do komponentov. Jednotlivé časti sú uložené do priečinkov, ktoré tvoria logické usporiadanie aplikácie. Týmto spôsobom je aplikácia pripravená na jednoduché budúce rozšírenie.

Aplikácie je robená architektúrov MVC.

Program obsahuje tieto priečinky :

- Screens – obsahuje všetky používané obrazovky aplikácie
- Components – obsahuje komponenty, ktoré využívajú obrazovky
- Constants – obsahuje základné nastavenia pre aplikáciu, napr. základná farba
- Assets - obrázky, ikony a fonty apod.
- Navigation – obsahuje súbory pre vytvorenie navigačného menu v aplikácii
- Store – Obsahuje reducers a actions pre vytvorenie Redux store
- Models – dátové modely pre Redux
- Data – obsahuje dummy-data, ktoré som používal na začiatku ako factory

3.2 App.js

Vstupný bod aplikácie. Tento súbor má na starosti spustenie celej aplikácie.

Pri spustení aplikácie vytvorí inštanciu Redux store, kde načíta všetky dáta potrebné pre aplikáciu.

```
const rootReducer = combineReducers({
  products: productsReducer,
  cart: cartReducer,
  orders: ordersReducer,
  auth: authReducer,
  appLoading: appLoadingReducer
});

// Inicializacia redux store
const store = createStore(rootReducer, applyMiddleware(ReduxThunk));
```

Po vytvorení úložiska, sa aplikácia najskôr snaží načítať fonty a potom načíta aplikáciu aj s navigátorom aplikácie.

```
// Hlavna funkcia aplikacie
export default function App() {
  const [fontLoaded, setFontLoaded] = useState(false);

  // Pri nacitavani fontov je aplikacia v stave nacitavania
  if (!fontLoaded) {
    return (
      <AppLoading
        startAsync={fetchFonts}
```

```
        onFinish={() => {
          setFontLoaded(true);
        }}
        onError={console.warn}
      />
    );
  }
  // Hlavna cast aplikacie - po nacitani fontov
  return (
    <Provider store={store}>
      <MainNavigation />
    </Provider>
  );
}
```

3.3 Navigátor

Na vytvorenie navigácie aplikácia využíva knižniciu **react-navigation v5**. Hlavná navigácia pozostáva s bočného menu (Drawer) – ShopNavigator. **MainNavigationContainer** vstupuje do App.js

```
// Obalova fukcia pre navigaciu aplikacie
const MainNavigationContainer = (props) => {
  const isAuth = useSelector((state) => state.auth.token != null ? true : fals
e);
  const navRef = useRef();
  useEffect(() => {
    if (!isAuth) {
      navRef.current.navigate("Auth");
    }
  }, [isAuth]);

  return (
    <NavigationContainer ref={navRef}>
      <ShopNavigator />
    </NavigationContainer>
  );
};
```

ShopNavigator, resp Drawer, obsahuje stackNavigatory, pre navigáciu v eshope, admin panely, authentifikáciu a objednávky.

```
const ProductsStack = createStackNavigator();
const OrdersStack = createStackNavigator();
const UsersStack = createStackNavigator();
const AuthStack = createStackNavigator();

// Hlavné menu - obsahuje všetky StackNavigatory
const Drawer = createDrawerNavigator();
```

ProductsStack – obrazovky **ProductsOverview, ProductDetails, Cart**
OrdersStack – obrazovka **Orders**
UsersStack – obrazovky **EditProduct, UserProducts**
AuthStack – obrazovka **Auth**

3.4 Komunikácia s databázou

Komunikáciu s databázou rieši Redux store jeho v action komponente (store/actions). Využívam metódu fetch, pomocou ktorej aplikácia komunikuje s DB.

```
const response = await fetch("https://reactnative-vamz-default-
rtdb.firebaseio.com/nazov_dokumentu.json");
```

Priklad pridávania produktu do DB. Je potrebné poslať aj token , ináč DB vyhodí výnimku.

```
const response = await fetch(https://reactnative-vamz-default-
rtdb.firebaseio.com/products.json?auth=${token}`,
{
     method: "POST",
     headers: {
            "Content-Type": "application/json",
     },
     body: JSON.stringify({
                    title,
                    imageUrl,
                    description,
                    price,
                    ownerId: userId,
                }),
     }
);
```

Ak nastane chyba pri komunikácii s DB, tak vyhodí program výnimku, ktorú zachytia iné metódy a vypíšu chybu.

```
if (!response.ok) {
    throw new Error("Something went wrong!");
}
```

Ak komunikácia prebehne úspešne, je možné získať dáta s DB

```
const resData = await response.json();
```

Tieto dáta potom ďalej využíva Redux Store, ktoré si ukladá.

Príklad vyvolanie uloženia produktu do Redux store z dát z DB po uložení do DB.

```
dispatch({
   type: CREATE_PRODUCT,
   productData: {
        id: resData.name,
        title,
        imageUrl,
        price,
        description,
        ownerId: userId,
   },
});
```

3.5 Auto-login

Auto-login systém funguje tak, že pri prihlásení užívateľa uloží aj do centrálneho úložiska v telefóne.

```
const saveDataToAsyncStorage = (token, userId, expirationDate) => {
  AsyncStorage.setItem(
    "userData",
    JSON.stringify({
      token: token,
      userId: userId,
      expiryDate: expirationDate.toISOString(),
    })
  );
};
```

Tým pádom, keď užívateľ zruší aplikáciu a znovu spustí, ako prvé sa pozrie program do tohto úložiska, či existuje záznam o prihlásenom užívateľovi , ak áno skontroluje validitu tokenu a ak je valídny, tak užívateľa nechá prihláseného a do redux store uloží dáta o užívateľovi.

```
const tryLogin = async () => {
     const userData = await AsyncStorage.getItem("userData");
     // Ak user nie je ulozeny
     if (!userData) {
         props.navigation.navigate("Products");
         dispatch(appLoadingActions.loaded());
         return;
    }
    const userDataToJson = JSON.parse(userData);
    const { token, userId, expiryDate } = userDataToJson;
    const expirationDate = new Date(expiryDate);
   // Ak user nema validny token
   if (expirationDate <= new Date() || !token || !userId) {
       props.navigation.navigate("Products");
       dispatch(appLoadingActions.loaded());
       return;
   }
   // Ak user ma platny token - je prihlaseny
   const expirationTime = expirationDate.getTime() - new Date().getTime();
   props.navigation.navigate("Products");
   dispatch(authActions.authenticate(userId, token, expirationTime));
   dispatch(appLoadingActions.loaded());
};
export const authenticate = (userId, token, expiryTime) => {
  //console.log("authentication");
  return (dispatch) => {
    // nastavi auto-logout timer
    dispatch(setLogoutTimer(expiryTime));
    // prihlasovanie/registracia
    dispatch({ type: AUTHENTICATE, userId: userId, token: token });
  };
};
```

Nastaví aj timer pre automatické odhlásenie.

```
// Nastavi casovac do kedy je token prihlaseneho uzivatela validny
const setLogoutTimer = (expirationDate) => {
  return (dispatch) => {
    timer = setTimeout(() => {
      dispatch(logout());
    }, expirationDate);
  };
};
```

3.6 Auto-logout

Po nastevení času odpočtu, do kedy je token valídny, program čaká kým odpočet neskončí. Po vypršaní času program automaticky užívateľa odhlási.

```
const setLogoutTimer = (expirationDate) => {
  return (dispatch) => {
    timer = setTimeout(() => {
      dispatch(logout());
    }, expirationDate);
  };
};
export const logout = () => {
  console.log("logout");
  clearTimer();
  AsyncStorage.removeItem("userData");
  return { type: LOGOUT };
};
// Vymaze casovac pre auto-logout
const clearTimer = () => {
  if (timer) {
    clearTimeout(timer);
  }
};
```

3.7 Validácia polí formulára

Validáciu polí rieši komponent Input. Ten obsahuje metódu, ktorá zisťuje aké parametre má kontrolovať a podľa toho zvolí či bude po zadaní vstupu do poľa reťazec valídny.

```
const textChangedHandler = (text) => {
    const emailRegex = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|
(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-
9]+\.)+[a-zA-Z]{2,}))$/;
    let isValid = true;
    // Ak su data required
    if (props.required && text.trim().length === 0) {
      isValid = false;
    }
    // Ak email
```

```
    if (props.email && !emailRegex.test(text.toLowerCase())) {
      isValid = false;
    }
    // Min hodnota
    if (props.min != null && +text < props.min) {
      isValid = false;
    }
    // Max hodnota
    if (props.max != null && +text > props.max) {
      isValid = false;
    }
    // Min dlzka
    if (props.minLength != null && text.length < props.minLength) {
      isValid = false;
    }
    // Po validacii uloz data do Inputu
    dispatch({ type: INPUT_CHANGE, value: text, isValid: isValid });
  };
```

Po kontrole validity dáta pošle do formulára. Ten uchováva dáta a validitu všetkých polí formulára. Takto vyzerá úložisko formulára.

```
formReducer = (state, action) => {
  if (action.type === FORM_INPUT_UPDATE) {
    const updatedValues = {
      ...state.inputValues,
      [action.input]: action.value,
    };
    const updatedValidities = {
      ...state.inputValidies,
      [action.input]: action.isValid,
    };
    let updatedFormIsValid = true;
    for (const key in updatedValidities) {
      if (!updatedValidities[key]) {
        updatedFormIsValid = false;
      }
    }
    return {
      ...state,
      inputValues: updatedValues,
      inputValidies: updatedValidities,
      formIsValid: updatedFormIsValid,
    };
  }
  return state;
};
const [formState, dispatchFormState] = useReducer(formReducer, {
    inputValues: {
```

```
      title: editedProduct ? editedProduct.title : "",
      imageUrl: editedProduct ? editedProduct.imageUrl : "",
      description: editedProduct ? editedProduct.description : "",
      price: "",
    },
    inputValidies: {
      title: editedProduct ? true : false,
      imageUrl: editedProduct ? true : false,
      description: editedProduct ? true : false,
      price: editedProduct ? true : false,
    },
    formIsValid: editedProduct ? true : false,
  });
```