Технически университет- София
Факултет по Компютърни Системи и Технологии

Курсов проект
Програмиране за мобилни устройства

Тема: „Тестова Система“

Изготвил: Марио Веселинов Стоилов
Факултетен Номер: 121211122

Github репозитори - https://github.com/energy-uktc/QuizApp

Съдържание:

Изисквания към мобилното приложение

Да се проектира и реализира програма позволяваща провеждането на тестове за проверяване на знания в дадена област. Да се предостави възможност за добавянето на нови въпроси. Да се включат и въпроси с отворен отговор (например посочване на отговор конкретно число). Като опции на програмата да се включат режими на тестване с време и без. Възможност за печат на тест, печат или запис на справка с
резултати от положени тестове.

Архитектура на приложението. Използвани технологии

Мобилното приложението е базирано на Expo софтуерна рамка и React Native компоненти за изграждането на кросплатформен потребителски интерфейс и Firebase Realtime Database централизирана NoSQL база данни за съхранение на информацията.

React Native е софтуерна рамка с отворен код базирана на JavaScript. Използва се за изграждане на кросплатформени мобилни приложения за iOS и Android. Съществуват свободно поддържани от потребителите разклонения на React Native, които позволяват и разработване на приложения за Windows, MacOS, Apple TV, Android TV. React Native компонентите представляват абстракция върху базовите за платформата компоненти, като по този начин позволява изграждането на потребителски интерфейс, който изглежда естетически консистентно с останалите елементи на мобилната платформа. Например View компонента в React Native се транслира до ViewGroup в Android и UIView в iOS. В таблицата по-долу, взета от официалния сайт на React Native, може да се види връзката между основните компоненти на фреймуърка и съответстващите им в Android и iOS. React Native използва JSX (JavaScript XML) за описване на елементите и вграждането им в JavaScript код. Всеки React Native елемент притежава набор от свойства (props) , които определят как се изобразява един компонент, и състояние (state) ,което съдържа данните с които работи компонента – всяка промяна в състоянието води до повторно изобразяване на елемента. За централно управление и хранилище на състоянието, което се споделя между компонентите се използват пакетите redux, react-redux, redux-thunk.

За навигация между отделните екрани се използва React Navigation пакетите @react-navigation/native, @react-navigation/bottom-tabs, @react-navigation/material-bottom-tabs, @react-navigation/stack.

https://reactnative.dev/docs/intro-react-native-components

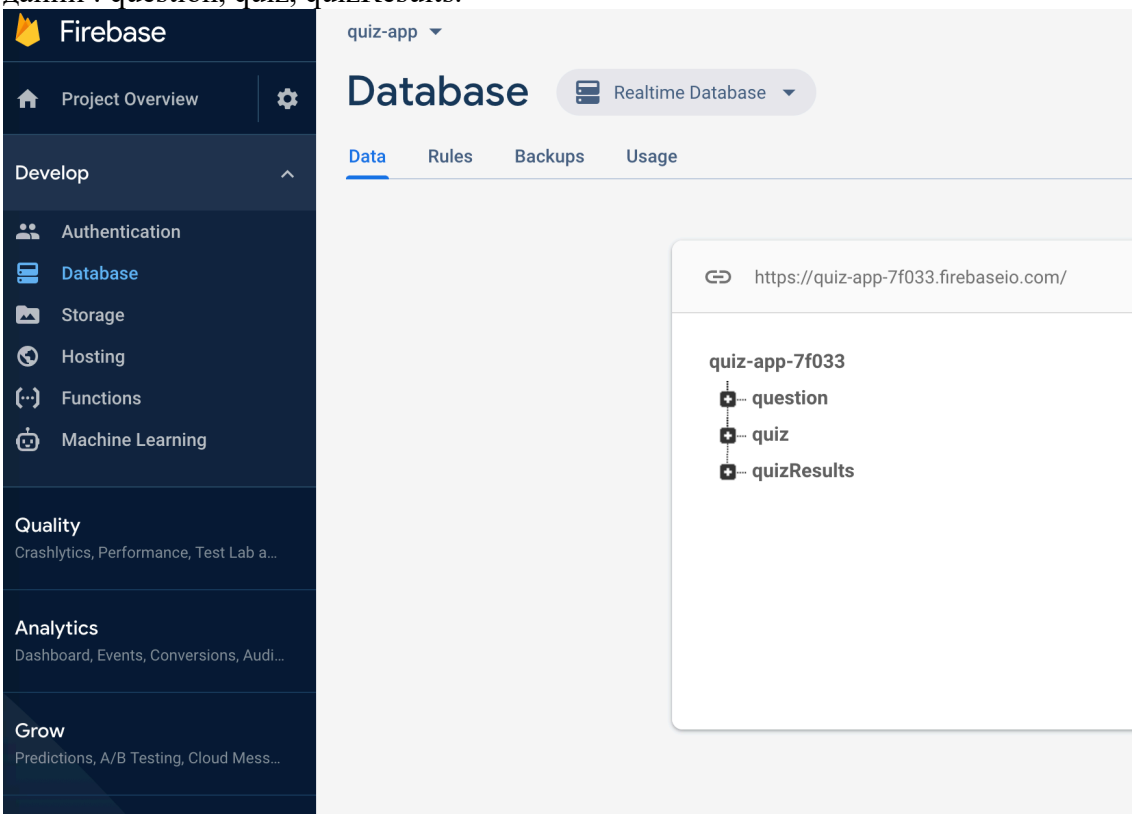
Expo е софтуерна рамка и платформа за React приложения. Представлява набор от услуги

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrollling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

и инструменти изградени около React Native, които улесняват разработката, компилирането, публикуването и тестването на приложенията. Също така включва и богат набор от пакети за React Native. Expo значително ускорява процеса на разработка на приложението, предоставяйки синхронизация на версии между отделни React Native библиотеки и Live reload на мобилното приложение в процеса на разработка, възможност за тестване директно върху iOS или Android устройство посредством Expo container app.

Firebase Realtime Database облачна услуга, която ни предоставя връзка към NoSQL база данни, където информацията се съхранява в JSON структури. Връзката към нея се осъществява чрез REST API. Мобилното приложение използва три структури в базата

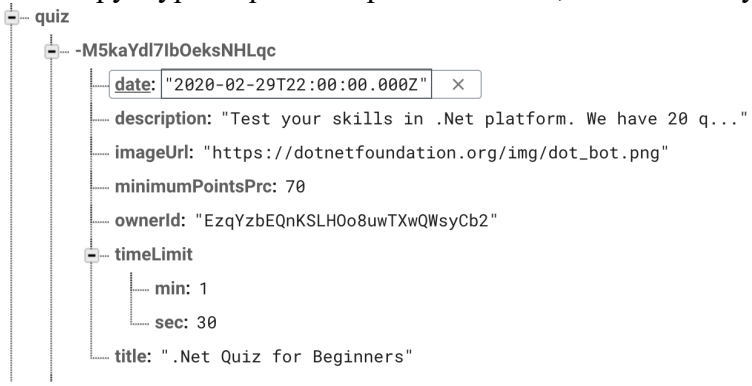
данни : question, quiz, quizResults.



Всеки път, когато нов JSON обект се добави към структурата от данни, за него се генерира уникален ключ. С цел оптимизация на четенето от базата данни въпросите (question) са в отделна структура от тестовете (quiz). Връзката между тях е посредством quizId полето в question.json върху което има дефиниран индекс.

```
{
  "rules": {
          …
          …
    "question": {
       ".indexOn": ["quizId"]
     }
  }
}
```

JSON структурата quiz съдържа тестовете, които се визуализират в приложението.

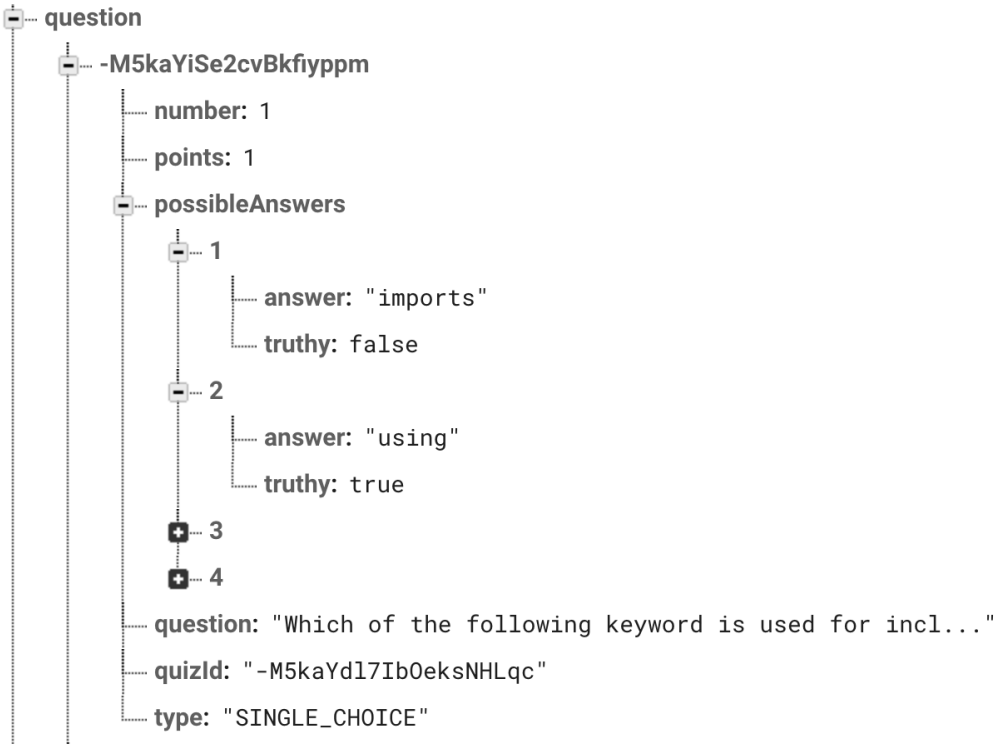


Всеки тест се състои от :

- Уникално ID.
- Дата на публикуване.
- Описание.
- Линк към изображение, което се използва за лого на теста.
- Минимален процент точки, над който теста се смята за успешно преминат.
- OwnerId – ID на потребителя, който е създал теста.
- timeLimit – Разделя се на минути и секунди. Полето е опционално
- Заглавие на теста.

Question документите съдържат набора от въпроси, всеки от които е свързан към определен тест посредством индексираното поле quizId. Поддържат се два типа въпроси:

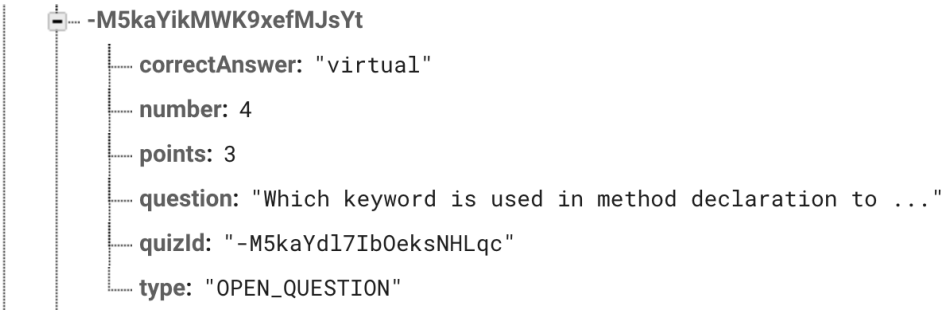
- SINGLE_CHOICE - Въпрос с два или повече възможни отговора, като само един един от тях е верен.



Елементи на структурата question:

- Уникално ID
- number – пореден номер на въпроса в теста.
- points – брой точки, които носи верният отговор.
- possibleAnswers – Списък с възможни отговори. Всеки възможен отговор се състои от текст и флаг, който указва дали е верен.
- question – Текстово поле съдържащо въпроса.
- quizId – ID на теста, към когото принадлежи съответния въпрос.
- type – Типа на въпроса (SINGLE_CHOICE или OPEN_QUESTION)

- OPEN_QUESTION – Въпрос с отворен отговор. Различава се от типа SINGLE_CHOICE по това, че вместо колекция от възможни отговори, съдържа текстово поле answer, в което е верния отговор на въпроса.



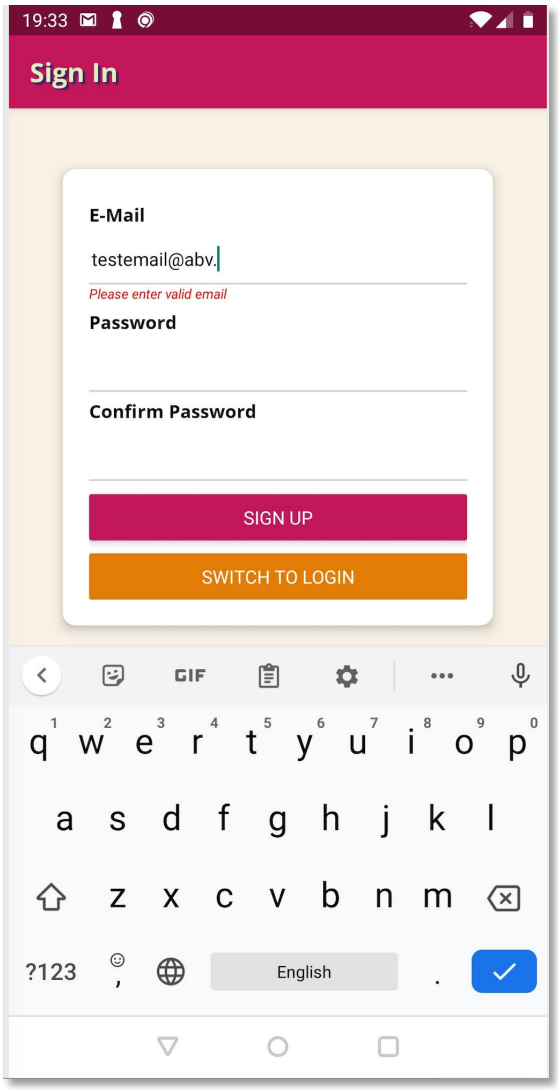
quizRezults съдържа резултатите от положени тестове обединени по ID на потребителя. Под ID-то на потребителя се намира списък от тестовете, които е положил.

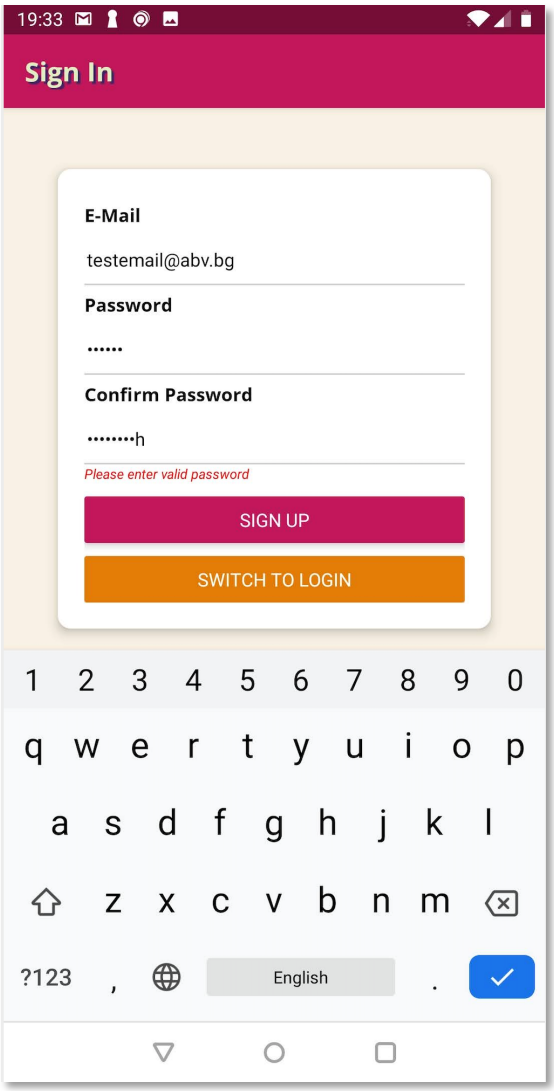
Съдържа се информация дали теста е преминат успешно или не, както и във всеки въпрос се запазва отговорът даден от потребителя.

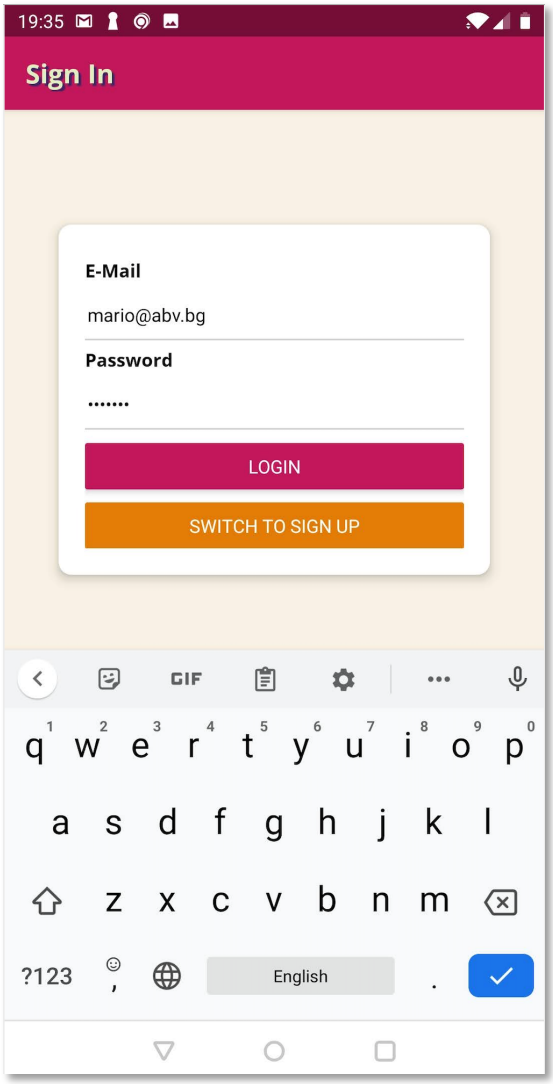
Firebase Authentication предоставя backend услуги за управление на потребителите – добавяне на потребител, удостоверяване на съществуващ потребител. Използва се като допълнение към Firebase Realtime Database , за ограничаване на достъпа до данни, които са собственост на друг ползвател на приложението. Например всеки потребител получава достъп единствено до своите резултати от положени тестове. Firebase Auth предоставя широк набор от услуги и възможности за изграждане на управление за удостоверяване на потребителите. В мобилното приложение е използван подхода с идентификация чрез имейл и парола. Към сървъра се изпраща имейл и парола на потребителя и като резултат се получава localId ,idToken , refreshToken.
localId е уникалният ключ с който е асоцииран потребителя.
idToken-а е символен низ, който се добавя към заявките, които се изпращат към Firebase Realtime Database REST API, за удостоверяване, че изпращача на заявката е регистриран и е потребител на приложението. idToken има определен период на валидност и не може да бъде използван след изтичането му. За издаването на нов idToken се използва refreshToken-а, който се съхранява в AsyncStorage -а на приложението.

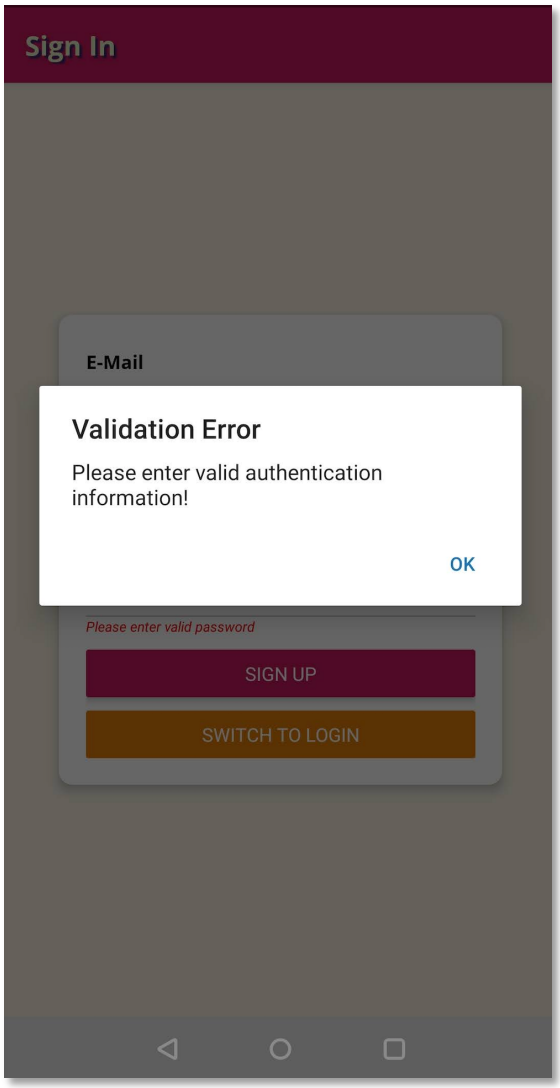
Описание на мобилното приложение. Ръководство за употреба

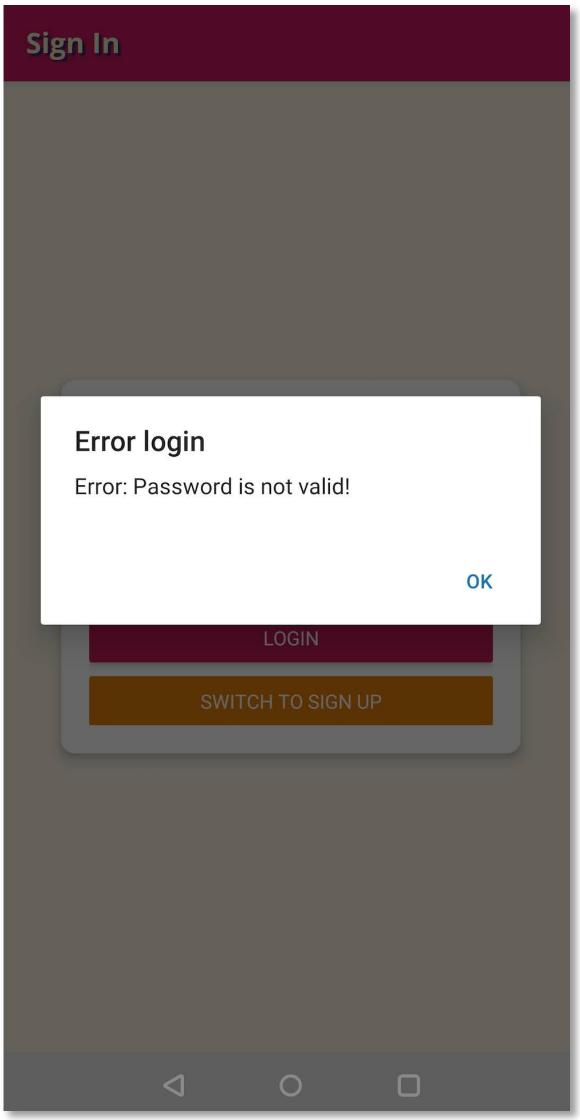
1. Създаване или удостоверяване на профила на потребител.- При отваряне на приложението за първи път, потребителят вижда екрана за аутентикация. Ако няма създаден профил трябва да се регистрира. За навигация между формата за регистрация и формата за удостоверява се използват бутоните SWITCH TO LOGIN и SWITCH TO SIGN UP. За регистрация и идентификация на потребител се използват имейл и парола. Полетата за имейл и пароли са защитени с валидации – прави се проверка за валидността на формата на имейл адреса по време на въвеждане, както и дали двете полета за парола съвпадат в случай на създаване на нов потребителски профил. При въвеждане на невалидни данни към потребителя се извежда съответната грешка.

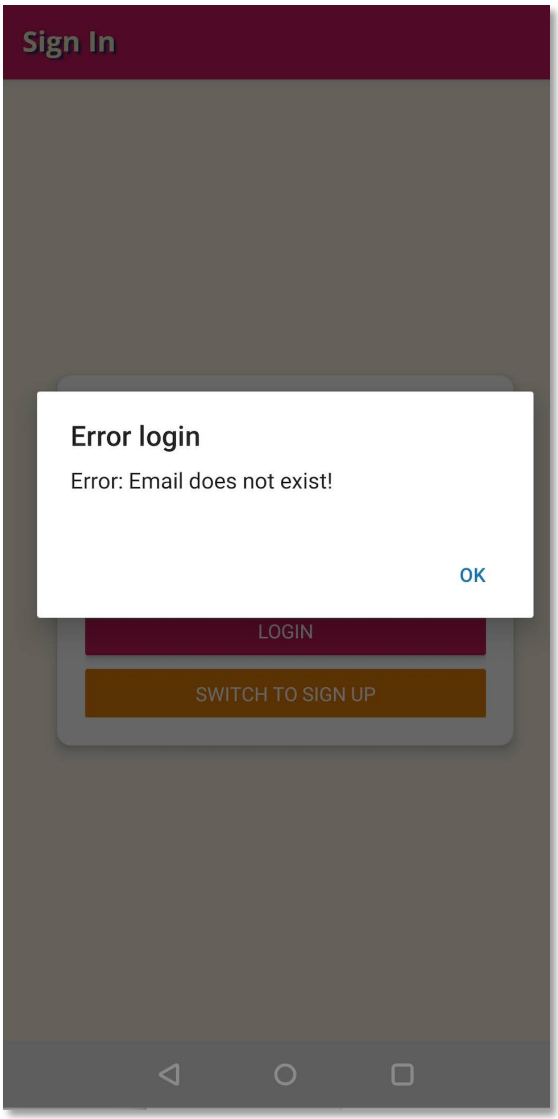

19:33
Sign In
E-Mail
testemail@abv.
Please enter valid email
Password
Confirm Password
SIGN UP
SWITCH TO LOGIN
GIF
English


19:33
Sign In
E-Mail
testemail@abv.bg
Password
Confirm Password
Please enter valid password
SIGN UP
SWITCH TO LOGIN
English

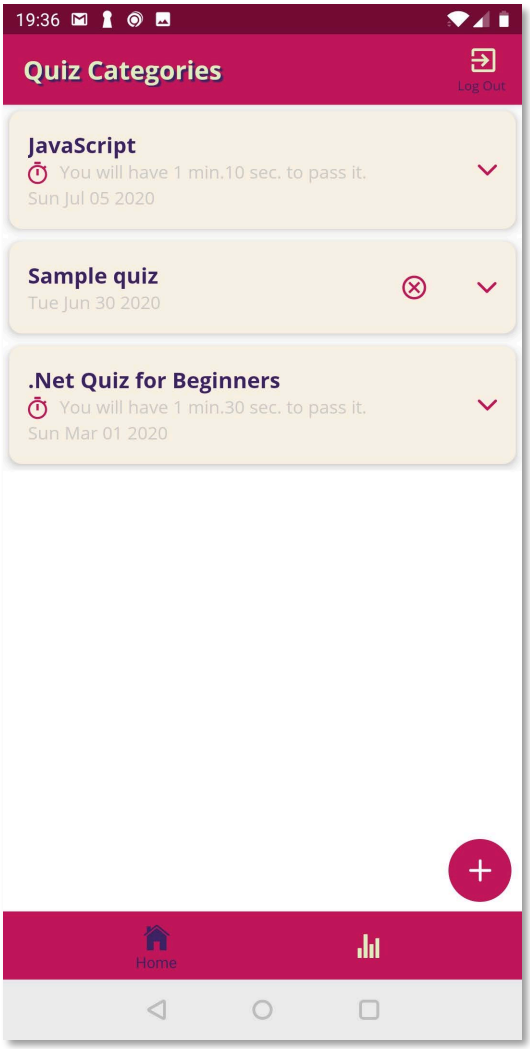

19:35
Sign In
E-Mail
mario@abv.bg
Password
.....
LOGIN
SWITCH TO SIGN UP
GIF
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
English

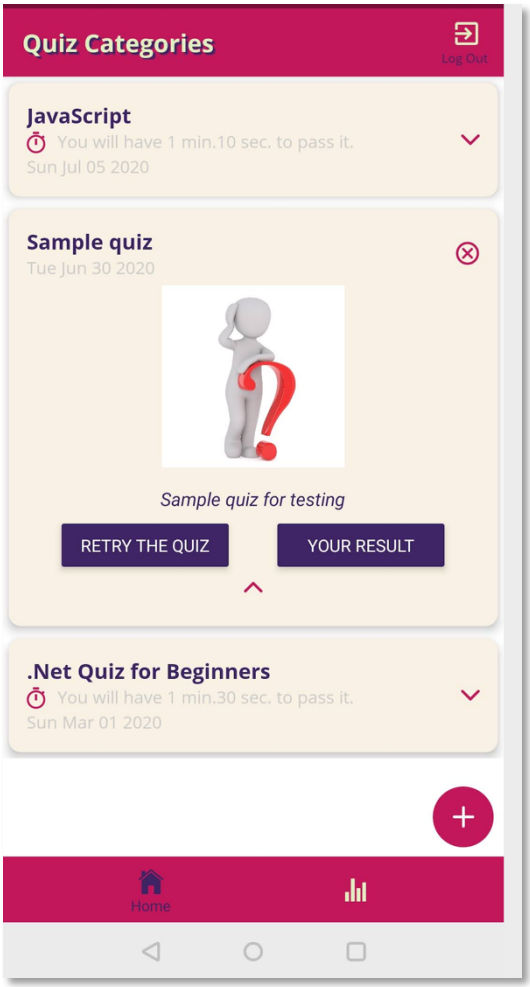

Sign In
E-Mail
Validation Error
Please enter valid authentication information!
OK
Please enter valid password
SIGN UP
SWITCH TO LOGIN





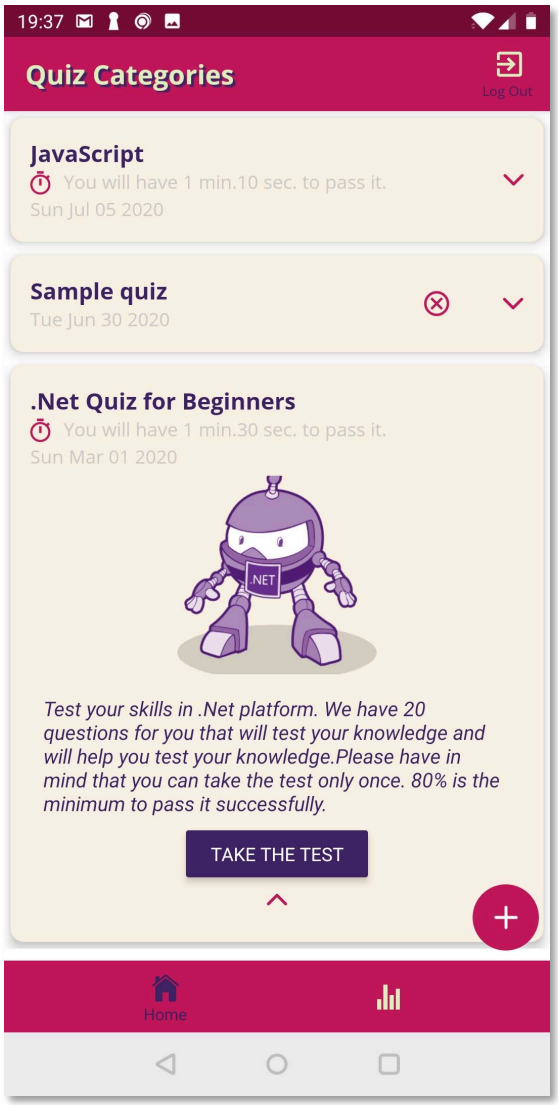
2. Начален екран – след процеса за удостоверяне на профила, потребителят отива към началният екран на приложението. Той е разделен на два таба, между които се навигира с бутоните в долната част на приложението. На първия таб е списък със всички тестове, а на вторият е списък с резултати от положени тестове. Всеки тест, който е бил положен от потребителя, има маркер, който указва дали е бил успешно издържан. Един тест може да бъде положен повече от един път, като индикаторът показва последния резултат. Списъкът с тестове съдържа съкратена информация за всеки от тях – заглавие, дата на публикуване, индикатор дали теста е с времеви лимит. При клик върху стрелката в дясно се отваря разширена информация за теста. Може да се види пълното описание на теста, неговото лого, както и бутон, с който може да се премине към самия тест. Ако потребителят вече е





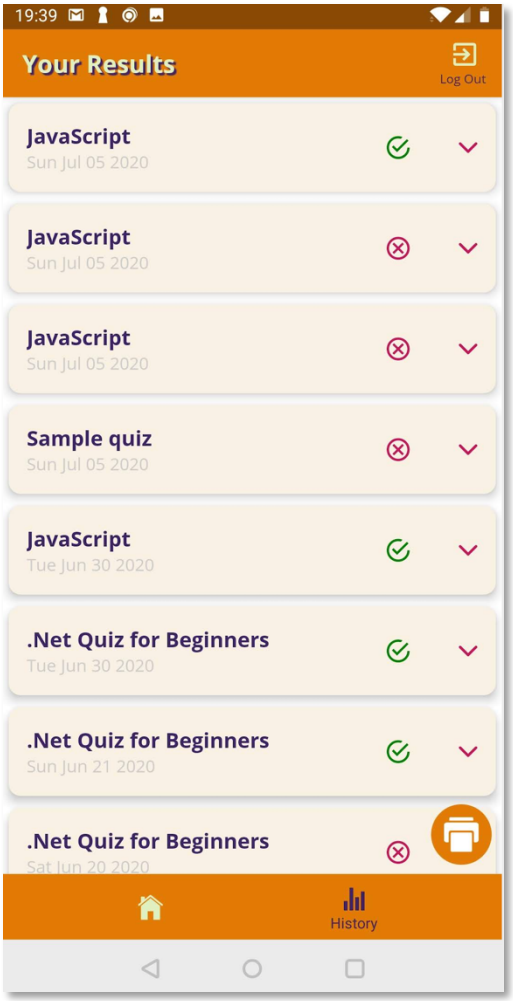
правил теста ще бъдат визуализирани два бутона – за преглед на резултата и за повторен опит на теста.

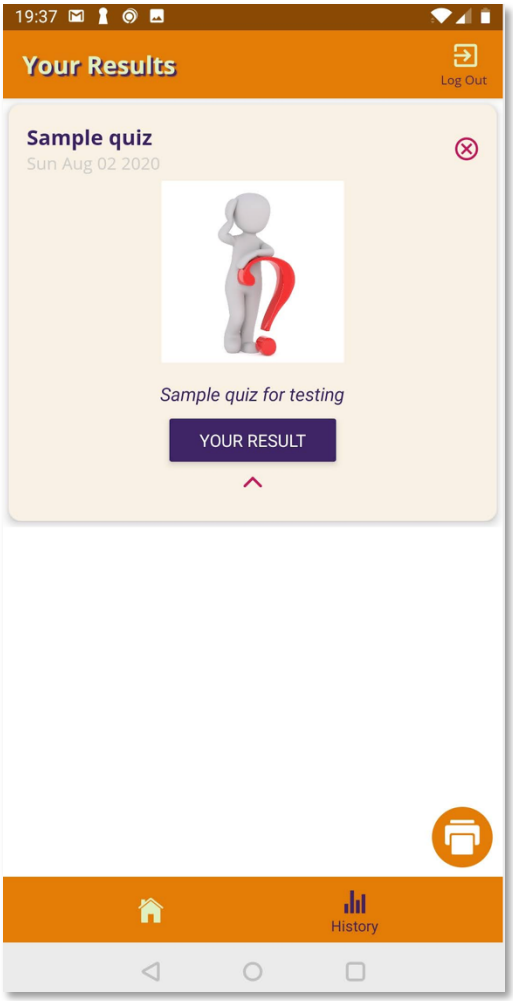
Бутонът  се използва за създаване на нов тест. Всеки потребител може да създава тестове, които са видими за всичко потребители на приложението след публикуването им.

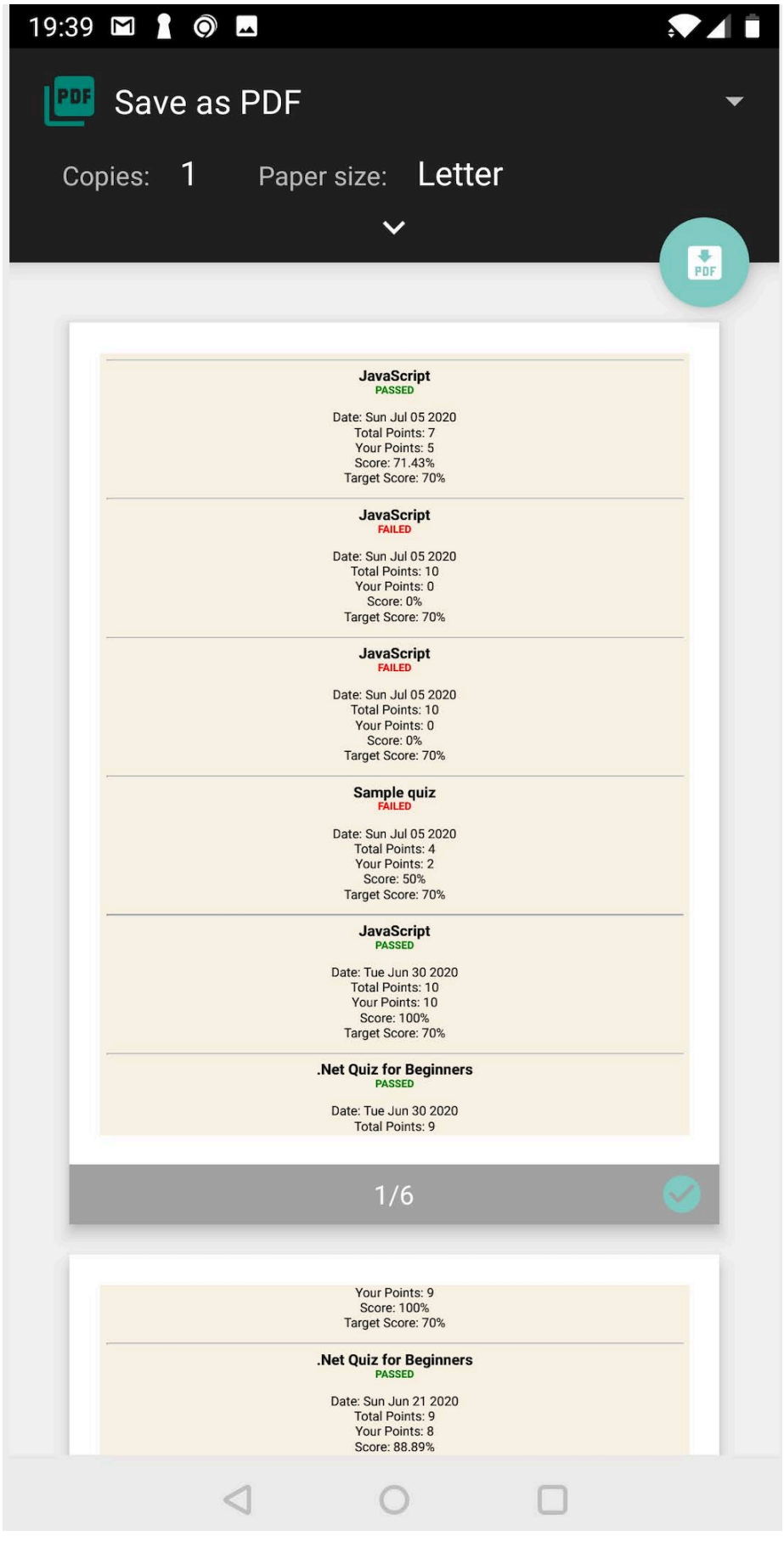


Бутонът  от списъка с резултати от тестове се използва за разпечатване на всички положени до момента тестове.

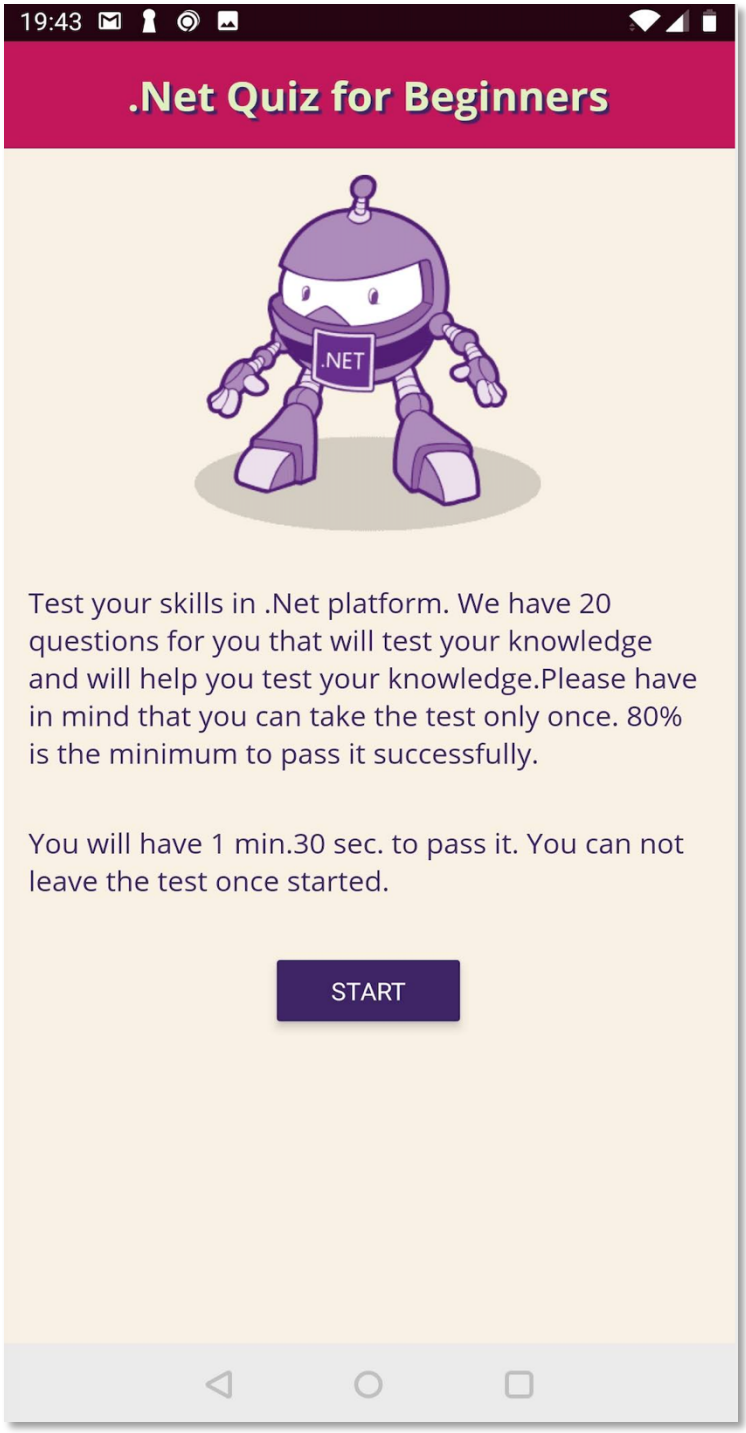
С бутон  потребителят може да излезе от профила си по всяко време.


19:39
Your Results
Log Out
JavaScript
Sun Jul 05 2020
JavaScript
Sun Jul 05 2020
JavaScript
Sun Jul 05 2020
Sample quiz
Sun Jul 05 2020
JavaScript
Tue Jun 30 2020
.Net Quiz for Beginners
Tue Jun 30 2020
.Net Quiz for Beginners
Sun Jun 21 2020
.Net Quiz for Beginners
Sat Jun 20 2020
History

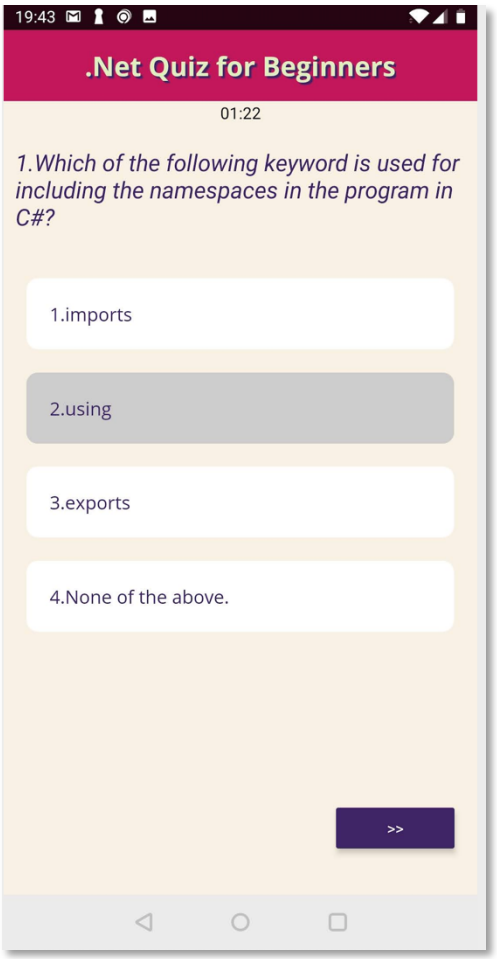

19:37
Your Results
Log Out
Sample quiz
Sun Aug 02 2020
Sample quiz for testing
YOUR RESULT
History

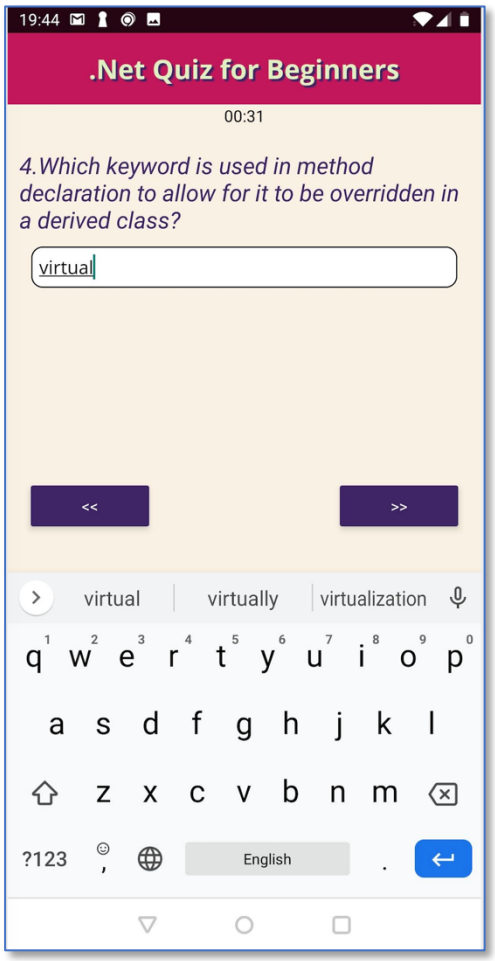

19:39
Save as PDF
Copies: 1
Paper size: Letter
PDF
JavaScript
PASSED
Date: Sun Jul 05 2020
Total Points: 7
Your Points: 5
Score: 71.43%
Target Score: 70%
JavaScript
FAILED
Date: Sun Jul 05 2020
Total Points: 10
Your Points: 0
Score: 0%
Target Score: 70%
JavaScript
FAILED
Date: Sun Jul 05 2020
Total Points: 10
Your Points: 0
Score: 0%
Target Score: 70%
Sample quiz
FAILED
Date: Sun Jul 05 2020
Total Points: 4
Your Points: 2
Score: 50%
Target Score: 70%
JavaScript
PASSED
Date: Tue Jun 30 2020
Total Points: 10
Your Points: 10
Score: 100%
Target Score: 70%
.Net Quiz for Beginners
PASSED
Date: Tue Jun 30 2020
Total Points: 9
1/6
Your Points: 9
Score: 100%
Target Score: 70%
.Net Quiz for Beginners
PASSED
Date: Sun Jun 21 2020
Total Points: 9
Your Points: 8
Score: 88.89%

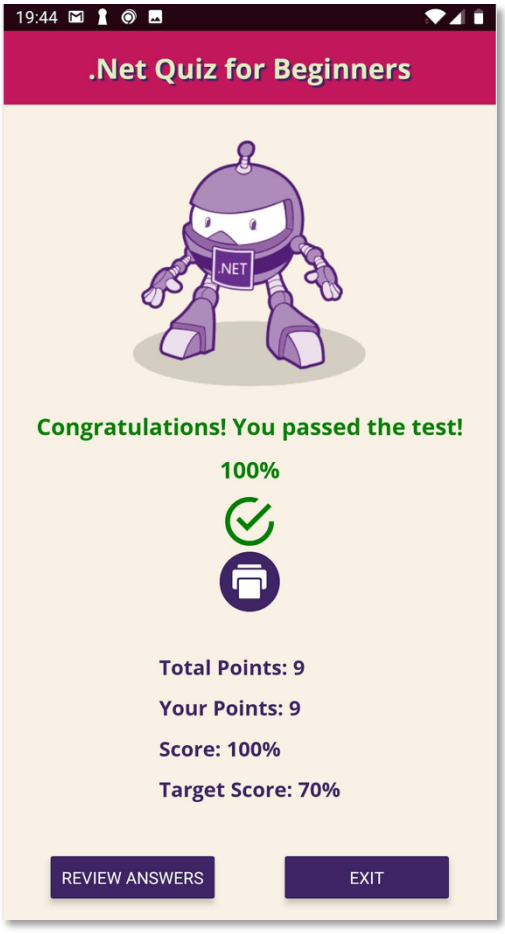
3. Провеждане на тест. При отваряне на тест се визулизира екран с детайлна информация за теста.



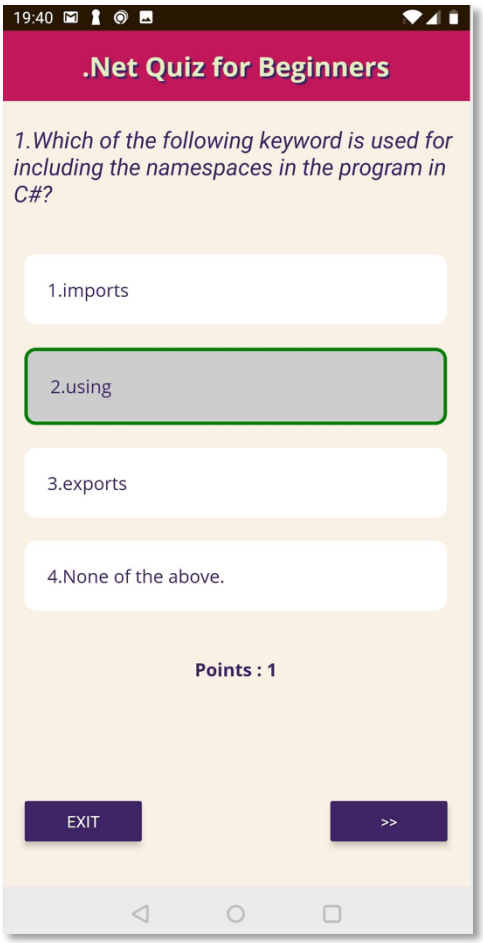
След натискане на бутон START се визуализират въпросите, както и таймер, ако тестът е за определено време. След започване, изпълнението на теста не може да бъде прекъснато. Има два типа въпроси – тестови и с отворен отговор. Възможно е да се премине към следващия въпрос без да е маркиран отговор на текущия, както и да се навигира между въпросите от теста свободно. Също така потребителя може да се върне към предишен въпрос и да смени посочения отговор.
В горната част на екрана се визуализира таймер, който отмерва оставащото време. След изтичане на времето теста се прекратява, като се запазват дадените до момента отговори от потребителя и се оценяват.

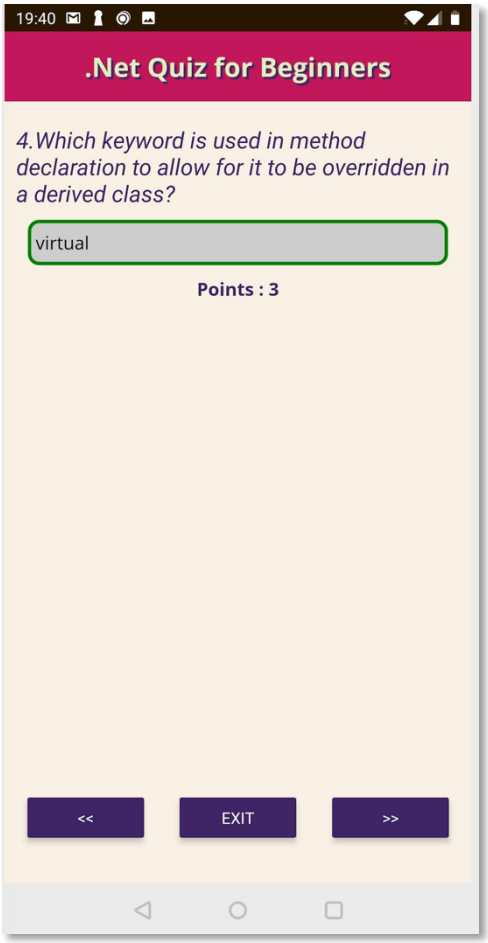


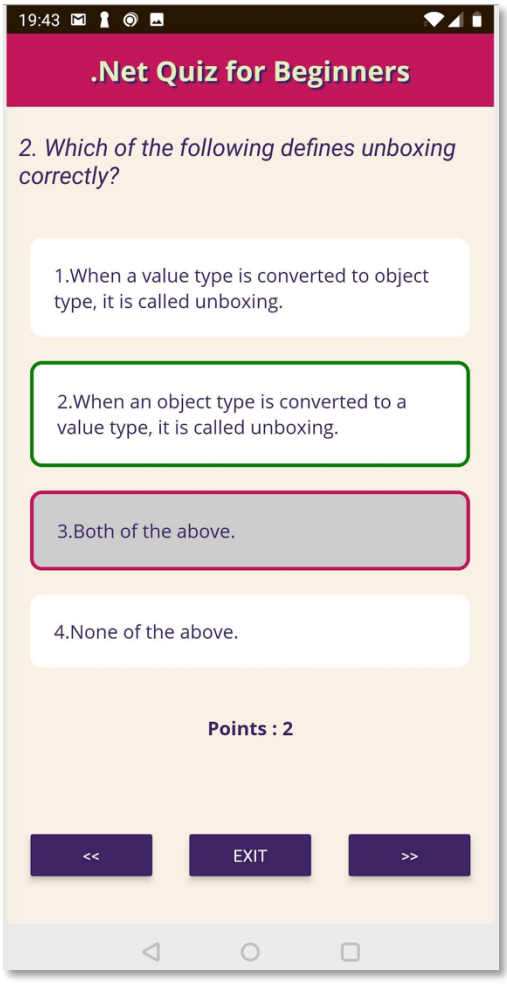


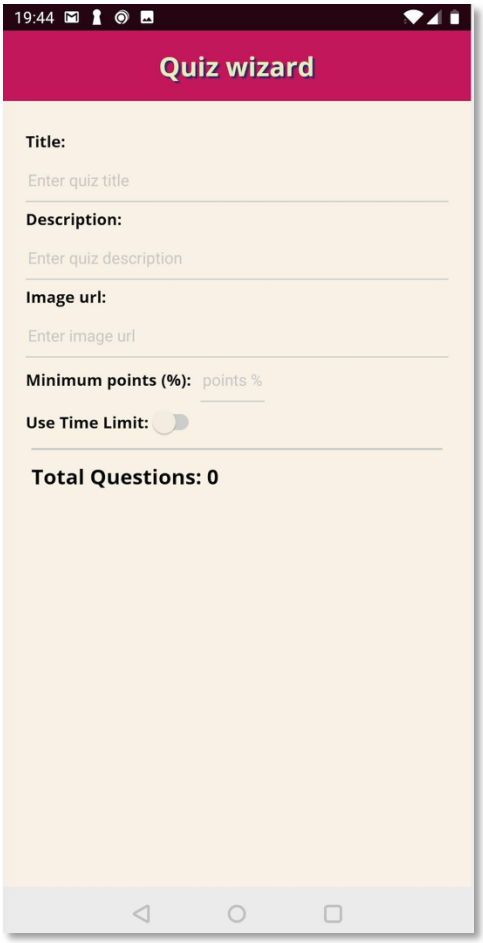


След приключване на теста се визуализира екран, който показва резултата от положения тест. С бутон EXIT се отива обратно на начален екран, а с бутон REVIEW ANSWERS може да се направи преглед на теста и да се видят верните отговори. Има и бутон „печат“ , с който може да се разпечати справка за положения тест.


19:40
.Net Quiz for Beginners
1.Which of the following keyword is used for including the namespaces in the program in C#?
1.imports
2.using
3.exports
4.None of the above.
Points : 1
EXIT
>>

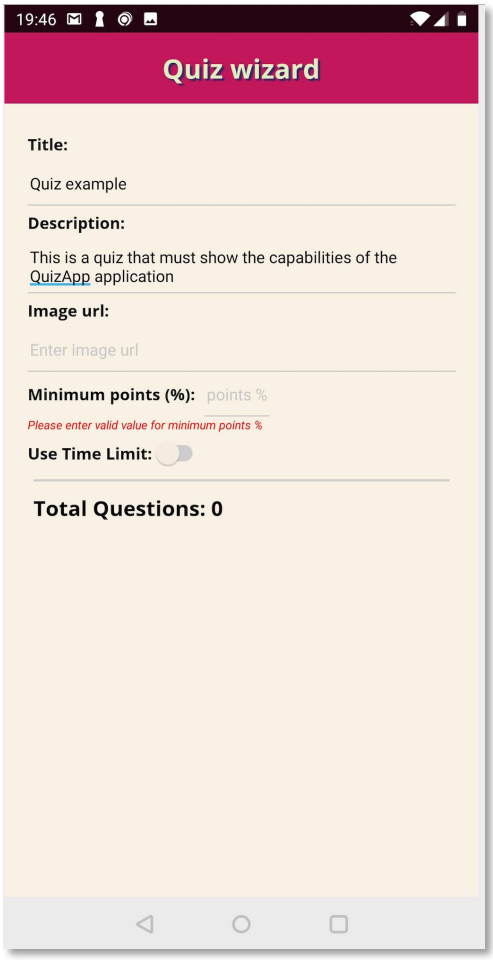

19:40
.Net Quiz for Beginners
4.Which keyword is used in method declaration to allow for it to be overridden in a derived class?
virtual
Points : 3
<<
EXIT
>>


19:43
.Net Quiz for Beginners
2. Which of the following defines unboxing correctly?
1.When a value type is converted to object type, it is called unboxing.
2.When an object type is converted to a value type, it is called unboxing.
3.Both of the above.
4.None of the above.
Points : 2
<<
EXIT
>>

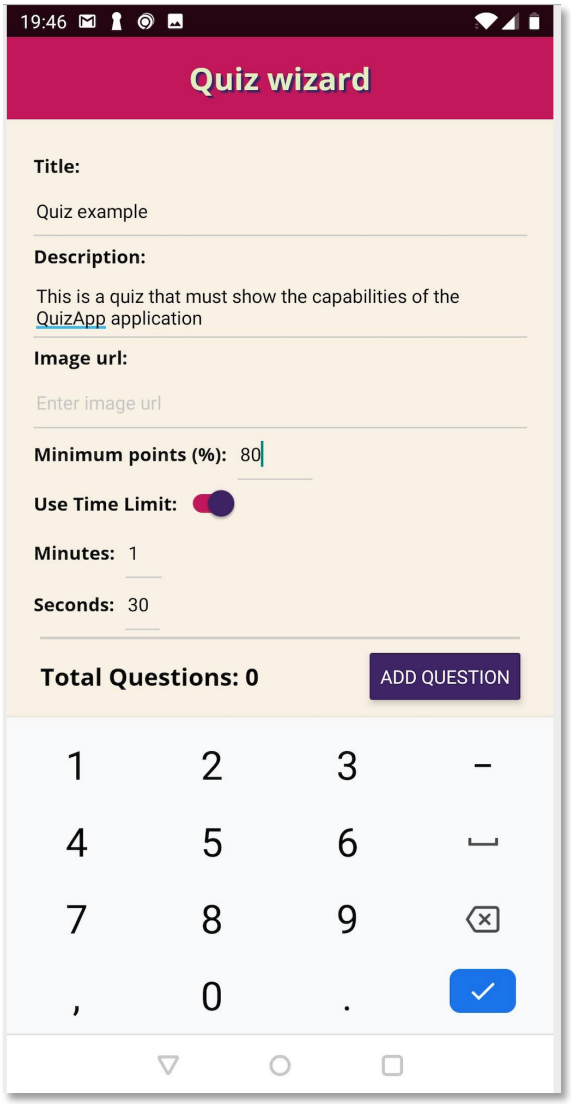


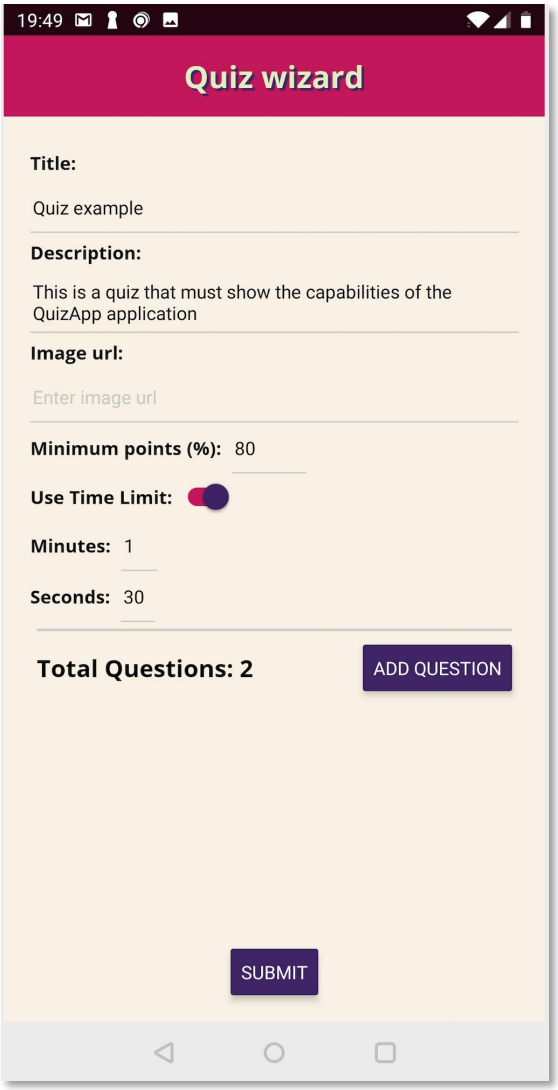
4. Създаване на тест.

При създаване на тест първо се визуализира прозорец за въвеждане на основна информация за теста. Заглавие, описание, миминален брой точки са задължителни полета. Всяко едно от тях има валидационен текст.

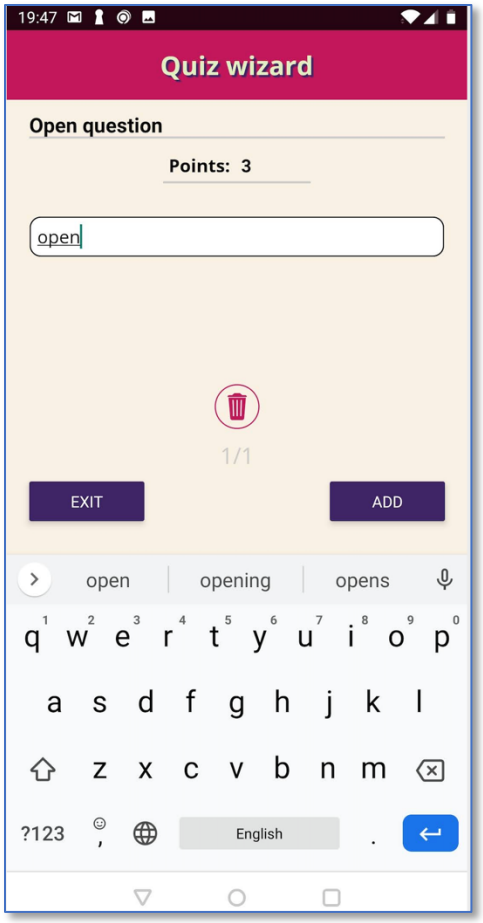


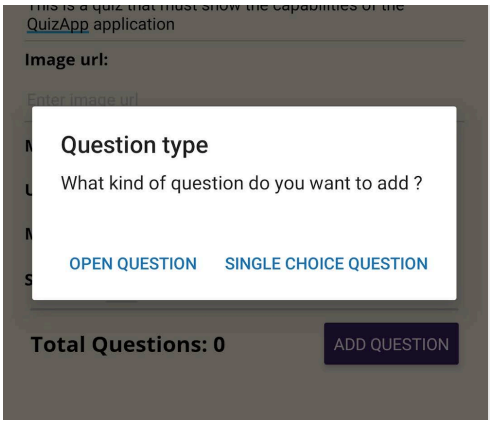
Ако въведената информация е валидна се появява бутон за добавяне на въпроси към теста. При въвеждане на един или повече въпроси се появява и бутон SUBMIT, който се използва за публикуване на теста. По този начин той става видим за останалите потребители на приложението.





При избиране на бутон ADD QUESTION се извежда съобщение за избор на типа на въпроса- с оворен отговор или тестови.

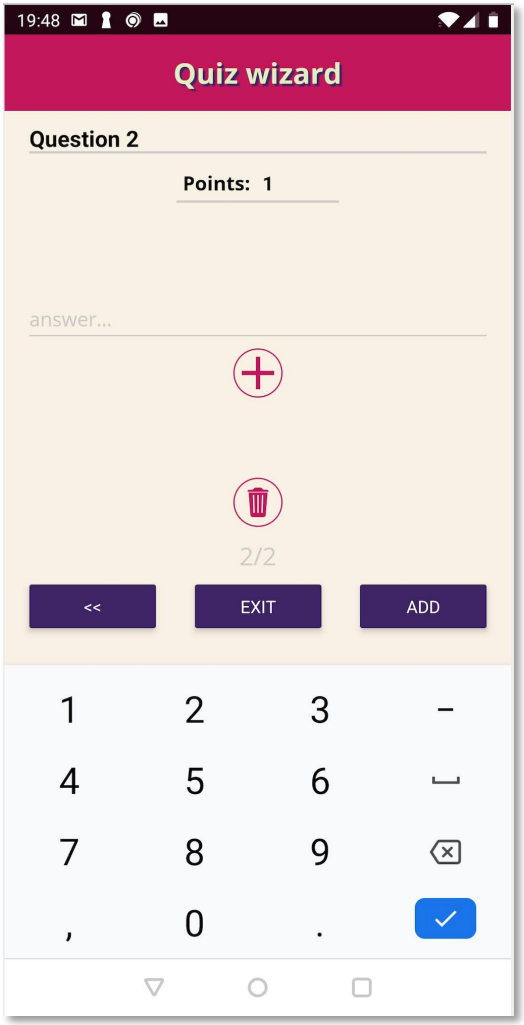


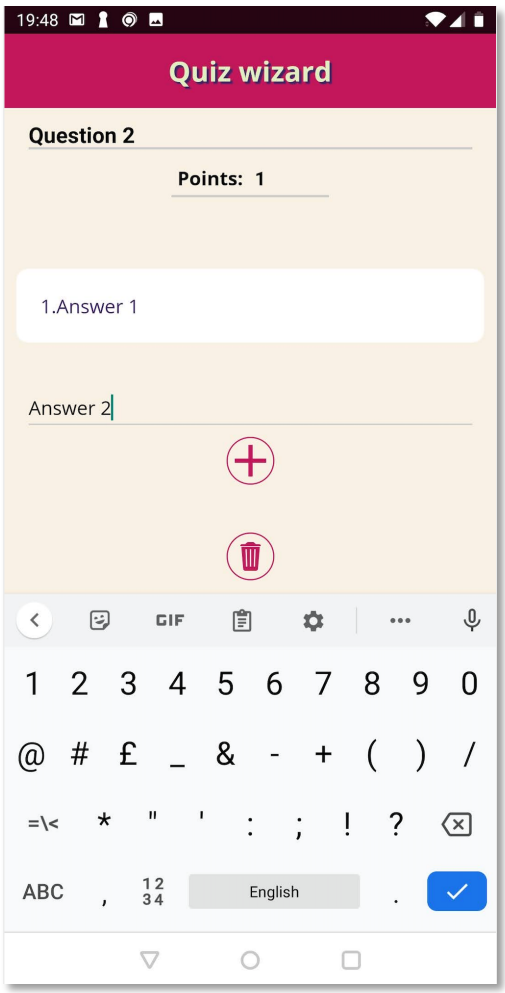


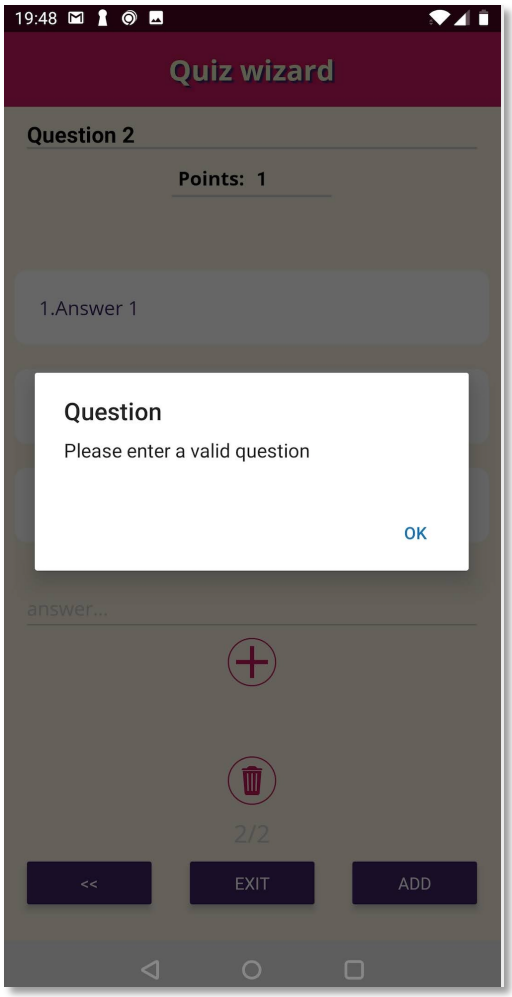
При създаване на въпрос с отворен отговор трябва да се въведе въпрос, брой точки, които носи правилният отговор и верен отговор на въпроса.

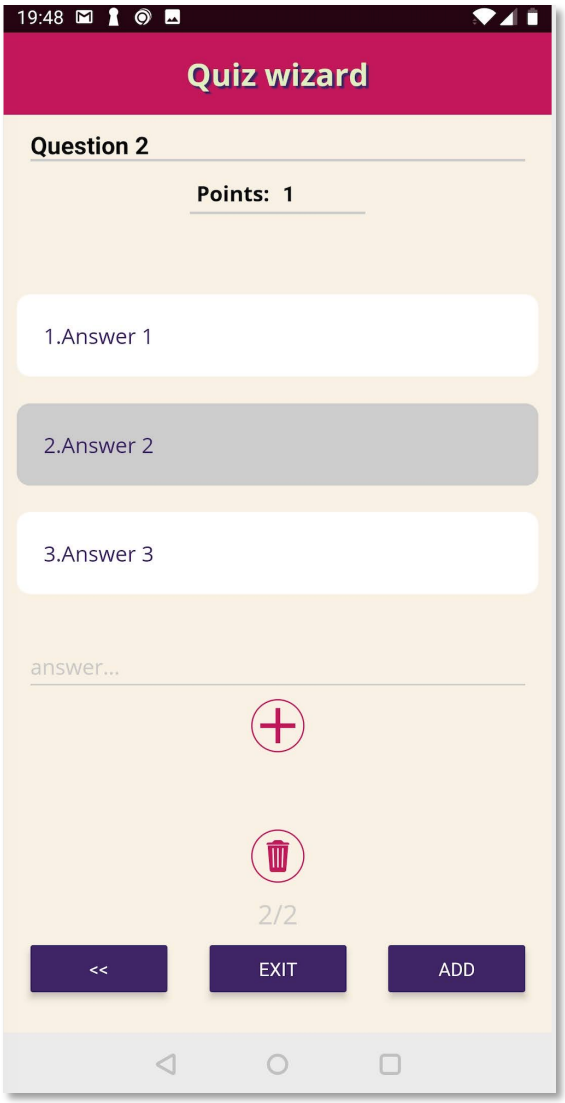
При въпрос от тестови тип има възможност да се добавят няколко възможни отговора, като само един от тях може да се посочи като верен.

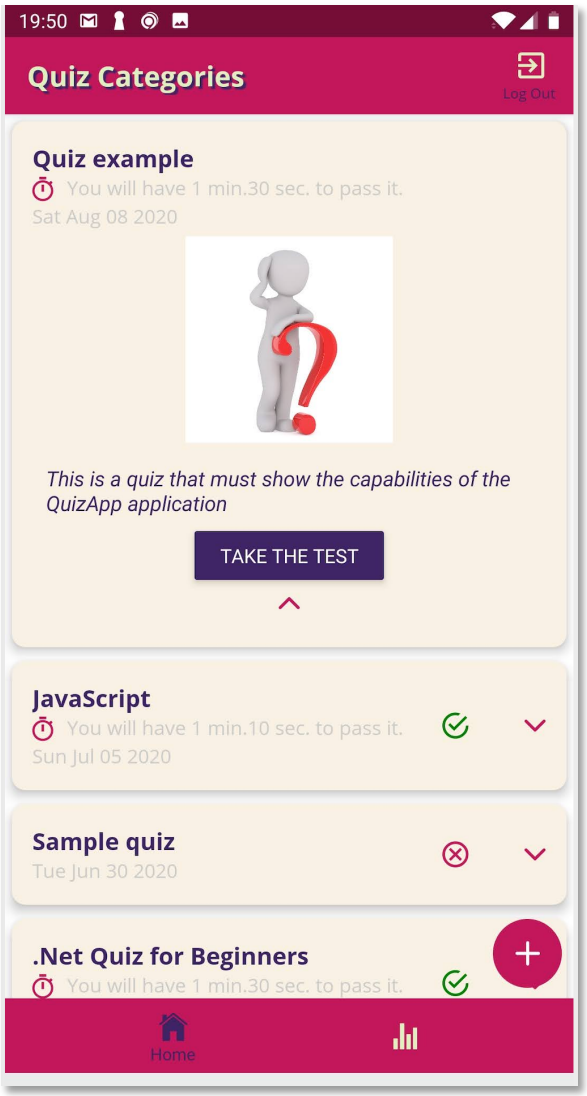
С бутон ADD може да се добави нов въпрос, ако текущия е валиден (има повече от един възможен отговор, въведен е броят точки и е посочен верният отговор).
Има възможност да се навигира между въпросите и да се изтрива въпрос. Бутон EXIT връща потребителя към екрана за попълване на информация за теста.


19:48
Quiz wizard
Question 2
Points: 1
answer...
2/2
<<
EXIT
ADD
1
2
3
4
5
6
7
8
9
,
0
.


19:48
Quiz wizard
Question 2
Points: 1
1.Answer 1
Answer 2
GIF
1 2 3 4 5 6 7 8 9 0
@ # £ _ & - + ( ) /
=\< * " ' : ; ! ?
ABC
English


19:48
Quiz wizard
Question 2
Points: 1
1.Answer 1
Question
Please enter a valid question
OK
answer...
2/2
<<
EXIT
ADD





Програмен код на приложението

1. Приложението стартира от App.js файла. Визуализира се loading компонент – AppLoading докато се заредят необходимите ресурси. След това се зарежда NavigationContainer компонента, който е обвит в state sore-а на приложението.

```
import React, { useState } from "react";
import { AppLoading } from "expo";
import { Provider } from "react-redux";
import * as Font from "expo-font";
import NavigationContainer from "./navigation/NavigationContainer";
import store from "./store/store";
```

```
const fetchFonts = () => {
  return Font.loadAsync({
    "open-sans": require("./assets/fonts/OpenSans-Regular.ttf"),
    "open-sans-bold": require("./assets/fonts/OpenSans-Bold.ttf"),
  });
};

export default function App() {
  const [fontLoaded, setFontLoaded] = useState(false);
  if (!fontLoaded) {
    return (
      <AppLoading
        startAsync={fetchFonts}
        onFinish={() => {
          setFontLoaded(true);
        }}
      />
    );
  }
  return (
    <Provider store={store}>
      <NavigationContainer />
    </Provider>
  );
}
```

2. NavigationContainer компонента дефинира и управлява връзката между отделните екрани на приложението.

```
import * as React from "react";
import { Platform, Alert } from "react-native";
import { useSelector } from "react-redux";
import { NavigationContainer } from "@react-navigation/native";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { createMaterialBottomTabNavigator } from "@react-navigation/material-bottom-
tabs";
import { createStackNavigator } from "@react-navigation/stack";
import HomeScreen from "../screens/quiz/Home";
import ResultsScreen from "../screens/user/Results";
import AuthScreen from "../screens/user/AuthScreen";
import StartupScreen from "../screens/StartupScreen";
import colors from "../constants/colors";
import { Ionicons } from "@expo/vector-icons";
import LogoutButton from "../components/UI/LogoutButton";
```

```
//const defaultNavOptions = {};
const defaultNavOptions = {
  headerStyle: {
    backgroundColor: Platform.OS === "android" ? colors.primary : "white",
  },
  headerTitleStyle: {
    fontFamily: "open-sans-bold",
    textShadowColor: colors.activeColor,
    textShadowOffset: { width: 2, height: 2 },
    textShadowRadius: 1,
  },
  headerBackTitleStyle: {
    fontFamily: "open-sans",
  },

  headerTintColor:
    Platform.OS === "android" ? colors.inactiveColor : colors.primary,
};
const Tab = //createBottomTabNavigator();
  Platform.OS === "android"
    ? createMaterialBottomTabNavigator()
    : createBottomTabNavigator();
const Stack = createStackNavigator();

const QuizNavigator = (props) => {
  return (
    <Stack.Navigator initialRouteName="Home" screenOptions={defaultNavOptions}>
      <Stack.Screen
        options={{
          title: "Quiz Categories",
          headerRight: () => <LogoutButton />,
        }}
        name="Home"
        component={HomeScreen}
      />
    </Stack.Navigator>
  );
};

const ResultsNavigator = (props) => {
  return (
    <Stack.Navigator
      initialRouteName="History"
      screenOptions={{
        ...defaultNavOptions,
        headerStyle: {
          backgroundColor: Platform.OS === "android" ? colors.accent : "white",
```

```
        },
      }}
    >
      <Stack.Screen
        options={{
          title: "Your Results",
          headerRight: () => <LogoutButton />,
        }}
        name="History"
        component={ResultsScreen}
      />
    </Stack.Navigator>
  );
};

export default NavContainer = (props) => {
  let isLoading = useSelector((state) => state.loading.loading);
  let isAuth = useSelector((state) => !!state.auth.userId);
  if (isLoading) return <StartupScreen />;

  return (
    <NavigationContainer>
      {isAuth ? (
        <Tab.Navigator
          initialRouteName="Home"
          screenOptions={defaultNavOptions}
          shifting={true}
          activeColor={colors.activeColor}
          inactiveColor={colors.inactiveColor}
          barStyle={{ backgroundColor: "#694fad" }}
        >
          <Tab.Screen
            options={{
              title: "Home",
              tabBarColor: colors.primary,
              tabBarIcon: (tabInfo) => {
                return (
                  <Ionicons name="ios-home" size={25} color={tabInfo.color} />
                );
              },
            }}
            name="Home"
            component={QuizNavigator}
          />
          <Tab.Screen
            options={{
              title: "History",
              tabBarColor: colors.accent,
```

```
            tabBarIcon: (tabInfo) => {
              return (
                <Ionicons name="ios-stats" size={25} color={tabInfo.color} />
              );
            },
          }}
          name="History"
          component={ResultsNavigator}
        />
      </Tab.Navigator>
    ) : (
      <Stack.Navigator
        initialRouteName="SignIn"
        screenOptions={defaultNavOptions}
      >
        <Stack.Screen
          options={{ title: "Sign In" }}
          name="SignIn"
          component={AuthScreen}
        />
      </Stack.Navigator>
    )}
  </NavigationContainer>
 );
};
```

3. В store директорията се намира логиката за управление на централното състояние на приложението. Използван е общ патърн за работа с react-redux – логиката се разделя на actions и reducers. Всички reducer-и се обединяват в общ state store в store.js:

```
import authReducer from "./reducers/auth";
import quizReducer from "./reducers/quiz";
import loadingReducer from "./reducers/loading";
import { AUTHENTICATE } from "./actions/auth";
import * as authService from "../service/authService";
import ReduxThunk from "redux-thunk";
import { createStore, combineReducers, applyMiddleware } from "redux";
import { composeWithDevTools } from "redux-devtools-extension";

const saveAuthInfo = (store) => (next) => (action) => {
  if (action.type === AUTHENTICATE) {
    authService.setDispatcher(store.dispatch);
    authService.setUserId(action.userId);
  }
```

```
  return next(action);
};

const middleware = [ReduxThunk, saveAuthInfo];

const rootReducer = combineReducers({
  auth: authReducer,
  quiz: quizReducer,
  loading: loadingReducer,
});

export default store = createStore(
  rootReducer,
  //applyMiddleware(ReduxThunk),
  applyMiddleware(...middleware),
  composeWithDevTools()
);
```

Actions:

auth.js:

```
import * as authService from "../../service/authService";

export const AUTHENTICATE = authService.AUTHENTICATE;
export const LOGOUT = authService.LOGOUT;

export const storeAuthentication = (userId) => {
  return {
    type: AUTHENTICATE,
    userId: userId,
  };
};

export const logout = () => {
  return async (dispatch) => {
    await authService.logout();
    dispatch({ type: LOGOUT });
  };
};

export const signup = (email, password) => {
  return async (dispatch) => {
    const userId = await authService.signUp(email, password);
    dispatch(storeAuthentication(userId));
  };
};
```

```
export const login = (email, password) => {
  return async (dispatch) => {
    const userId = await authService.login(email, password);
    dispatch(storeAuthentication(userId));
  };
};
```

loading.js:

```
export const LOADED = "LOADED";

export const endLoading = () => {
  return {
    type: LOADED
  };
};
```

quiz.js

```
import * as quizService from "../../service/quizService";
export const GET_QUIZZES = "GET_QUIZZES";
export const ADD_QUIZ = "ADD_QUIZ";
export const ADD_RESULTS = "ADD_RESULTS";

export const getQuizzes = () => {
  return async (dispatch) => {
    const loadedQuizzes = await quizService.fetchQuizzes();
    const loadedUserQuizzes = await quizService.fetchUserQuizzes();
    dispatch({
      type: GET_QUIZZES,
      quizzes: loadedQuizzes,
      userQuizzes: loadedUserQuizzes,
    });
  };
};

export const addQuiz = (quiz) => {
  return {
    type: ADD_QUIZ,
    quiz: {
      title: quiz.title,
      imageUrl: quiz.imageUrl,
      description: quiz.description,
      timeLimit: quiz.timeLimit,
```

```
      date: quiz.date,
      minimumPointsPrc: quiz.minimumPointsPrc,
      ownerId: quiz.ownerId,
    },
    id: quiz.id,
  };
};

export const addResults = (passedQuiz) => {
  return { type: ADD_RESULTS, quiz: passedQuiz };
};
```

Reducers:

auth.js:

```
import { AUTHENTICATE, LOGOUT } from "../actions/auth";

const initialState = {
  userId: "",
};

export default (state = initialState, action) => {
  switch (action.type) {
    case LOGOUT:
      return initialState;
    case AUTHENTICATE:
      return {
        userId: action.userId,
      };
    default:
      return state;
  }
};
```

loading.js:

```
import { LOADED } from "../actions/loading";

const initialState = {
  loading: true
};

export default (state = initialState, action) => {
  switch (action.type) {
    case LOADED:
```

```
      return {
        loading: false
      };
    default:
      return state;
  }
};
```

quiz.js:

```
import { QUIZZES, HISTORY_QUIZZES } from "../../data/dummyData";
import { GET_QUIZZES, ADD_QUIZ, ADD_RESULTS } from "../actions/quiz";
import { Quiz, QuizResult } from "../../models/quiz";

const initialState = {
  availableQuizzes: [],
  passedQuizzes: [],
};

export default (state = initialState, action) => {
  switch (action.type) {
    case GET_QUIZZES:
      return {
        ...state,
        availableQuizzes: action.quizzes,
        passedQuizzes: action.userQuizzes,
      };
    case ADD_QUIZ:
      console.log(`ADD_QUIZ: ${action.quiz}`);
      const quiz = action.quiz;
      const newQuiz = new Quiz(
        action.id,
        quiz.title,
        quiz.imageUrl,
        quiz.description,
        quiz.timeLimit,
        quiz.date,
        quiz.minimumPointsPrc
      );
      return {
        ...state,
        availableQuizzes: state.availableQuizzes.concat(newQuiz),
      };

    case ADD_RESULTS:
      console.log(`ADD_QUIZ: ${action.quiz}`);
```

```
      const readyQuiz = action.quiz;
      const newUserQuiz = new QuizResult(
        readyQuiz.id,
        readyQuiz.title,
        readyQuiz.imageUrl,
        readyQuiz.description,
        readyQuiz.timeLimit,
        readyQuiz.date,
        readyQuiz.minimumPointsPrc,
        readyQuiz.quizId,
        readyQuiz.passed,
        readyQuiz.questions
      );
      return {
        ...state,
        passedQuizzes: state.passedQuizzes.concat(newUserQuiz),
      };
  }
  return state;
};
```

4. В service директорията е логиката свързана с комуникацията с бекенд частта (Firebase Realtime Database) , аутентикацията (Firebase Auth) и функционалността за отпечатване на резултати от тестове.

authService.js:

```
import { AsyncStorage, Alert } from "react-native";
import USER_DATA from "../constants/userData";
import { REFRESH_TOKEN_URL, API_KEY, AUTH_URL } from "../constants/api";
//import reduxStore from "../store/store";
export const AUTHENTICATE = "AUTHENTICATE";
export const LOGOUT = "LOGOUT";

let userId = "";
let dispatch = null;

export const setDispatcher = (reduxDispatcher) => {
  dispatch = reduxDispatcher;
};

export const setUserId = (currentUserId) => {
  userId = currentUserId;
};

export const getUserId = () => {
```

```
  return userId; //userId;
};

export const getDispatch = () => {
  return dispatch; //userId;
};

export const getTokenId = async () => {
  const { idToken } = await getUserData();
  return idToken;
};
export const validateAuthentication = async () => {
  try {
    const {
      userId,
      idToken,
      email,
      expirationDate,
      refreshToken,
    } = await getUserData();

    if (!userId || !idToken || !refreshToken || !email || !expirationDate) {
      throw new Error("User information not found");
    }
  } catch (err) {
    throw new Error(`Could not authenticate.${err.message}`);
  }
};

export const getUserData = async () => {
  const userDataString = await AsyncStorage.getItem(USER_DATA);
  if (!userDataString) {
    throw new Error("User information not found");
  }

  const userData = JSON.parse(userDataString);
  if (!userData) {
    throw new Error("User information not found");
  }
  return userData;
};

export const isTokenExpired = (expirationDate) => {
  if (new Date() >= new Date(expirationDate)) {
    return true;
  }
  return false;
};
```

```
export const refreshTokenIfExpired = async () => {
  try {
    const { expirationDate, refreshToken } = await getUserData();
    if (!isTokenExpired(expirationDate)) {
      return true;
    }

    await refreshAuthToken(refreshToken);
  } catch (err) {
    await handleAuthError(err);
    return false;
  }
  return true;
};

const handleAuthError = async (err) => {
  Alert.alert(`Authentication Error`, `${err.message} The App will logout`, [
    {
      text: "Okay",
      onPress: async () => {
        await logout();
        dispatch({ type: LOGOUT });
      },
    },
  ]);
};

const refreshAuthToken = async (refreshToken) => {
  const requestDate = new Date();
  const response = await fetch(`${REFRESH_TOKEN_URL}/token?key=${API_KEY}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      grant_type: "refresh_token",
      refresh_token: refreshToken,
    }),
  });

  if (!response.ok) {
    const errorData = await response.json();
    let message = "Something went wrong";
    switch (errorData.error.message) {
      case "TOKEN_EXPIRED":
        message = "Credentials are no longer valid. Please sign in again.";
        break;
```

```
      case "MISSING_REFRESH_TOKEN":
        message = "No refresh token provided.";
        break;
      case "USER_DISABLED":
        message = "The user account has been disabled by an administrator.";
        break;
      case "USER_NOT_FOUND":
        message = "The user corresponding to the refresh token was not found.";
        break;
      case "INVALID_REFRESH_TOKEN":
        message = "An invalid refresh token is provided.";
        break;
      case "INVALID_GRANT_TYPE":
        message = "The grant type specified is invalid.";
    }
    throw new Error(message);
  }

  const respData = await response.json();

  const expirationDate = new Date(
    requestDate.getTime() + parseInt(respData.expires_in) * 1000
  ).toISOString();

  await AsyncStorage.mergeItem(
    USER_DATA,
    JSON.stringify({
      idToken: respData.id_token,
      refreshToken: respData.refresh_token,
      expirationDate: expirationDate,
    })
  );
};

export const signUp = async (email, password) => {
  const requestDate = new Date();

  const response = await fetch(`${AUTH_URL}/accounts:signUp?key=${API_KEY}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      email: email,
      password: password,
      returnSecureToken: true,
    }),
  });
```

```
  if (!response.ok) {
    const errorData = await response.json();
    let message = "Something went wrong";
    switch (errorData.error.message) {
      case "EMAIL_EXISTS":
        message = "Email already used!";
        break;
      case "TOO_MANY_ATTEMPTS_TRY_LATER":
        message =
          "We have blocked all requests from this device due to unusual activity. Try
again later.";
        break;
    }
    throw new Error(message);
  }

  const respData = await response.json();

  const expirationDate = new Date(
    requestDate.getTime() + parseInt(respData.expiresIn) * 1000
  ).toISOString();

  await AsyncStorage.setItem(
    USER_DATA,
    JSON.stringify({
      userId: respData.localId,
      idToken: respData.idToken,
      email: respData.email,
      refreshToken: respData.refreshToken,
      expirationDate: expirationDate,
    })
  );

  return respData.localId;
};

export const login = async (email, password) => {
  const requestDate = new Date();
  const response = await fetch(
    `${AUTH_URL}/accounts:signInWithPassword?key=${API_KEY}`,
    {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        email: email,
```

```
        password: password,
        returnSecureToken: true,
      }),
    }
  );

  if (!response.ok) {
    const errorData = await response.json();
    let message = "Something went wrong";
    switch (errorData.error.message) {
      case "EMAIL_NOT_FOUND":
        message = "Email does not exist!";
        break;
      case "INVALID_PASSWORD":
        message = "Password is not valid!";
        break;
      case "USER_DISABLED":
        message = "Account has been disabled!";
        break;
    }
    throw new Error(message);
  }

  const respData = await response.json();
  const expirationDate = new Date(
    requestDate.getTime() + parseInt(respData.expiresIn) * 1000
  ).toISOString();

  await AsyncStorage.setItem(
    USER_DATA,
    JSON.stringify({
      userId: respData.localId,
      idToken: respData.idToken,
      email: respData.email,
      refreshToken: respData.refreshToken,
      expirationDate: expirationDate,
    })
  );
  return respData.localId;
};

export const logout = async () => {
  await AsyncStorage.removeItem(USER_DATA);
};
```

printingService.js:

```
import colors from "../constants/colors";
import * as quizService from "./quizService";
import * as Print from "expo-print";

export const printUserScores = async (quizzes) => {
  let content = "";
  quizzes.forEach((q) => {
    const totalPoints = quizService.calculateQuizTotalPoints(q.questions);
    const userPoints = quizService.calculateQuizEarnedPoints(q.questions);
    let score = 100;
    if (totalPoints) {
      score = Math.round((userPoints / totalPoints) * 10000, 2) / 100;
    }
    const passed = quizService.isPassed(q, q.questions);

    content += `<h3 style="margin:0">${q.title}</h3>`;
    if (passed) {
      content += `<h5 style="color:green;margin:0">PASSED</h5>`;
    } else {
      content += `<h5 style="color:red;margin:0">FAILED</h5>`;
    }
    content += `<p>`;
    content += `Date: ${q.date.toDateString()}<br>`;
    content += `Total Points: ${totalPoints}<br>`;
    content += `Your Points: ${userPoints}<br>`;
    content += `Score: ${score}%<br>`;
    content += `Target Score: ${q.minimumPointsPrc}%<br>`;
    content += `</p>`;
    content += `<hr>`;
  });
  content = packageHTML(content);
  return await Print.printAsync({ html: content });
};

export const printSingleQuizResult = async (quiz) => {
  let content = "";
  const totalPoints = quizService.calculateQuizTotalPoints(quiz.questions);
  const userPoints = quizService.calculateQuizEarnedPoints(quiz.questions);
  let score = 100;
  if (totalPoints) {
    score = Math.round((userPoints / totalPoints) * 10000, 2) / 100;
  }
  const passed = quizService.isPassed(quiz, quiz.questions);

  content += `<h3 style="margin:0">${quiz.title}</h3>`;
```

```
  if (passed) {
    content += `<h5 style="color:green;margin:0">PASSED</h5>`;
  } else {
    content += `<h5 style="color:red;margin:0">FAILED</h5>`;
  }
  content += `<img src=${quiz.imageUrl} width="200" height="auto">`;
  content += `<p>`;
  content += `Date: ${quiz.date.toDateString()}<br>`;
  content += `Total Points: ${totalPoints}<br>`;
  content += `Your Points: ${userPoints}<br>`;
  content += `Score: ${score}%<br>`;
  content += `Target Score: ${quiz.minimumPointsPrc}%<br>`;
  content += `</p>`;
  content += `<hr>`;
  content = packageHTML(content);
  return await Print.printAsync({ html: content });
};

const packageHTML = (content) => {
  let html = `<!DOCTYPE html><head><title>User Scores</title></head>`;
  html += `<body style="text-align:center;background-
color:${colors.backColor}"><hr>${content}</body>`;
  html += `</html>`;
  return html;
};
```

quizService.js:

```
import {
  QUESTION_TYPE,
  SingleChoiceQuestion,
  OpenQuestion,
} from "../models/question";
import { Quiz, QuizResult } from "../models/quiz";
import * as authService from "./authService";
import { URL } from "../constants/api";

export const QUIZ_STATUS = {
  NONE: "NONE",
  INIT: "INIT",
  STARTED: "STARTED",
  FINISHED: "FINISHED",
  REVIEW: "REVIEW",
  ERROR: "ERROR",
};
```

```
export const calculateQuizTotalPoints = (questions) => {
  let totalPoints = 0;
  questions.forEach((q) => {
    totalPoints += q.points;
  });
  return totalPoints;
};

export const calculateQuizEarnedPoints = (questions) => {
  let userPoints = 0;
  questions.forEach((q) => {
    if (q.userAnswer) {
      switch (q.type) {
        case QUESTION_TYPE.SINGLE_CHOICE: {
          userPoints += q.possibleAnswers[q.userAnswer].truthy ? q.points : 0;
          break;
        }
        case QUESTION_TYPE.OPEN_QUESTION: {
          userPoints += q.userAnswer === q.correctAnswer ? q.points : 0;
        }
      }
    }
  });
  return userPoints;
};

export const isPassed = (quiz, questions) => {
  const totalPoints = calculateQuizTotalPoints(questions);
  const userPoints = calculateQuizEarnedPoints(questions);
  const minimumPoints = (quiz.minimumPointsPrc / 100) * totalPoints;
  return !quiz.minimumPointsPrc || minimumPoints <= userPoints;
};

export const createQuestionFromJson = (id, jsonData) => {
  switch (jsonData.type) {
    case QUESTION_TYPE.OPEN_QUESTION: {
      const question = new OpenQuestion(
        id,
        jsonData.number,
        jsonData.question,
        jsonData.points,
        jsonData.quizId
      );
      question.setCorrectAnswer(jsonData.correctAnswer);
      return question;
    }

    case QUESTION_TYPE.SINGLE_CHOICE: {
```

```
      const question = new SingleChoiceQuestion(
        id,
        jsonData.number,
        jsonData.question,
        jsonData.points,
        jsonData.quizId
      );
      question.setPossibleAnswers(jsonData.possibleAnswers);
      return question;
    }
  }
};

export const createAnsweredQuestionFromJson = (id, jsonData) => {
  const question = createQuestionFromJson(id, jsonData);
  return question.setAnswer(jsonData.userAnswer);
};

export const createJsonFromQuestion = (question) => {
  const jsonQuestion = {
    number: question.number,
    question: question.question,
    points: question.points,
    quizId: question.quizId,
    type: question.type,
  };
  switch (question.type) {
    case QUESTION_TYPE.OPEN_QUESTION: {
      return JSON.stringify({
        ...jsonQuestion,
        correctAnswer: question.correctAnswer,
      });
    }
    case QUESTION_TYPE.SINGLE_CHOICE: {
      return JSON.stringify({
        ...jsonQuestion,
        possibleAnswers: question.possibleAnswers,
      });
    }
  }
};

export const insertFullQuiz = async (quiz, questions) => {
  const newQuiz = await insertQuiz(quiz);
  questions.forEach(async (q) => {
    q.quizId = newQuiz.id;
    await insertQuestion(q);
  });
```

```
  return newQuiz;
};

export const insertQuiz = async (quiz) => {
  if (!(await authService.refreshTokenIfExpired())) {
    return false;
  }
  const tokenId = await authService.getTokenId();
  const userId = authService.getUserId();
  const response = await fetch(`${URL}/quiz.json?auth=${tokenId}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      title: quiz.title,
      imageUrl: quiz.imageUrl
        ? quiz.imageUrl
        : "https://cdn.pixabay.com/photo/2017/05/13/09/04/question-2309040_1280.jpg",
      description: quiz.description,
      timeLimit: quiz.timeLimit,
      date: quiz.date,
      minimumPointsPrc: quiz.minimumPointsPrc,
      ownerId: userId,
    }),
  });
  if (!response.ok) {
    console.log(`ADD_QUIZ: Error response: ${JSON.stringify(response)}`);
    throw new Error(
      `Something went wrong. ${response.statusText ? response.statusText : ""}`
    );
  }
  const resData = await response.json();
  const newQuiz = new Quiz(
    resData.name,
    quiz.title,
    quiz.imageUrl,
    quiz.description,
    quiz.timeLimit,
    quiz.date,
    quiz.minimumPointsPrc,
    userId
  );
  return newQuiz;
};

export const insertQuestion = async (question) => {
  if (!(await authService.refreshTokenIfExpired())) {
```

```
    return false;
  }
  const tokenId = await authService.getTokenId();

  const response = await fetch(`${URL}/question.json?auth=${tokenId}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: createJsonFromQuestion(question),
  });

  if (!response.ok) {
    console.log(`ADD_QUESTION: Error response: ${JSON.stringify(response)}`);
    throw new Error(
      `Something went wrong. ${response.statusText ? response.statusText : ""}`
    );
  }
  const resData = await response.json();
  return resData.name;
};

export const fetchQuestions = async (quizId) => {
  if (!(await authService.refreshTokenIfExpired())) {
    return false;
  }
  const tokenId = await authService.getTokenId();
  const response = await fetch(
    `${URL}/question.json?auth=${tokenId}&orderBy="quizId"&equalTo="${quizId}"`
  );
  if (!response.ok) {
    console.log(`GET_QUESTIONS: Error response: ${JSON.stringify(response)}`);
    throw new Error(
      `Something went wrong. ${response.statusText ? response.statusText : ""}`
    );
  }
  const resData = await response.json();
  let loadedQuestions = [];
  for (const key in resData) {
    loadedQuestions.push(createQuestionFromJson(key, resData[key]));
  }
  return loadedQuestions;
};

export const fetchQuizzes = async () => {
  if (!(await authService.refreshTokenIfExpired())) {
    return false;
  }
```

```
  const tokenId = await authService.getTokenId();
  const response = await fetch(`${URL}/quiz.json?auth=${tokenId}`);
  if (!response.ok) {
    console.log(`GET_QUIZZES: Error response: ${JSON.stringify(response)}`);
    throw new Error(
      `Something went wrong. ${response.statusText ? response.statusText : ""}`
    );
  }
  const resData = await response.json();
  let loadedQuizzes = [];
  for (const key in resData) {
    const quiz = new Quiz(
      key,
      resData[key].title,
      resData[key].imageUrl,
      resData[key].description,
      resData[key].timeLimit,
      new Date(resData[key].date),
      resData[key].minimumPointsPrc,
      resData[key].ownerId
    );
    loadedQuizzes.push(quiz);
  }
  return loadedQuizzes;
};

export const insertQuizResults = async (quiz, questions) => {
  if (!(await authService.refreshTokenIfExpired())) {
    return false;
  }
  const tokenId = await authService.getTokenId();
  const userId = authService.getUserId();
  const date = new Date();
  const passed = isPassed(quiz, questions);
  const response = await fetch(
    `${URL}/quizResults/${userId}.json?auth=${tokenId}`,
    {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        title: quiz.title,
        imageUrl: quiz.imageUrl,
        description: quiz.description,
        timeLimit: quiz.timeLimit,
        date: date,
        minimumPointsPrc: quiz.minimumPointsPrc,
```

```
        quizId: quiz.id,
        passed: passed,
        questions: questions,
      }),
    }
  );
  if (!response.ok) {
    console.log(`ADD_RESULTS: Error response: ${JSON.stringify(response)}`);
    throw new Error(
      `Something went wrong. ${response.statusText ? response.statusText : ""}`
    );
  }
  const resData = await response.json();
  const newQuizResults = new QuizResult(
    resData.name,
    quiz.title,
    quiz.imageUrl,
    quiz.description,
    quiz.timeLimit,
    date,
    quiz.minimumPointsPrc,
    quiz.id,
    passed,
    questions
  );
  return newQuizResults;
};

export const fetchUserQuizzes = async () => {
  if (!(await authService.refreshTokenIfExpired())) {
    return false;
  }
  const tokenId = await authService.getTokenId();
  const userId = authService.getUserId();
  const response = await fetch(`${URL}/quizResults/${userId}.json?auth=${tokenId}`);
  if (!response.ok) {
    console.log(
      `GET_USER_QUIZZES: Error response: ${JSON.stringify(response)}`
    );
    throw new Error(
      `Something went wrong. ${response.statusText ? response.statusText : ""}`
    );
  }
  const resData = await response.json();
  let loadedQuizzes = [];
  for (const key in resData) {
    const quizQuestions = [];
    resData[key].questions.forEach((q) => {
```

```
      quizQuestions.push(createAnsweredQuestionFromJson(q.id, q));
    });

    const quiz = new QuizResult(
      key,
      resData[key].title,
      resData[key].imageUrl,
      resData[key].description,
      resData[key].timeLimit,
      new Date(resData[key].date),
      resData[key].minimumPointsPrc,
      resData[key].quizId,
      resData[key].passed,
      quizQuestions
    );
    loadedQuizzes.push(quiz);
  }
  return loadedQuizzes;
};
```

5. В models са класовете, които използва приложението, за структурен модел на данните.

question.js :

```
export const QUESTION_TYPE = {
  OPEN_QUESTION: "OPEN_QUESTION",
  SINGLE_CHOICE: "SINGLE_CHOICE",
};

class Question {
  constructor(id, number, question, points, quizId) {
    this.id = id;
    this.number = number;
    this.question = question;
    this.points = points;
    this.quizId = quizId;
    this.type = null;
  }
}

export class OpenQuestion extends Question {
  constructor(id, number, question, points, quizId) {
    super(id, number, question, points, quizId);
    this.type = QUESTION_TYPE.OPEN_QUESTION;
  }
  setCorrectAnswer(correctAnswer) {
```

```
    this.correctAnswer = correctAnswer;
  }

  setAnswer(answer) {
    return new AnsweredOpenQuestion(this, answer);
  }
}

export class SingleChoiceQuestion extends Question {
  constructor(id, number, question, points, quizId) {
    super(id, number, question, points, quizId);
    this.type = QUESTION_TYPE.SINGLE_CHOICE;
  }

  addPossibleAnswer(id, answer) {
    this.possibleAnswers = {
      ...this.possibleAnswers,
      [id]: {
        answer: answer,
        truthy: false,
      },
    };
  }

  setPossibleAnswers(possibleAnswers) {
    this.possibleAnswers = null;
    let id = 0;
    possibleAnswers.forEach((a) => {
      if (a) {
        id++;
        this.possibleAnswers = {
          ...this.possibleAnswers,
          [id]: {
            answer: a.answer,
            truthy: a.truthy,
          },
        };
      }
    });
  }

  setCorrectAnswer(correctAnswerId) {
    this.possibleAnswers[correctAnswerId].truthy = true;
  }

  setAnswer(answer) {
    return new AnsweredSingleChoiceQuestion(this, answer);
  }
```

```
}

export class AnsweredSingleChoiceQuestion extends SingleChoiceQuestion {
  constructor(question, answer) {
    super(
      question.id,
      question.number,
      question.question,
      question.points,
      question.quizId
    );
    this.possibleAnswers = question.possibleAnswers;
    this.userAnswer = answer;
  }
}

export class AnsweredOpenQuestion extends OpenQuestion {
  constructor(question, answer) {
    super(
      question.id,
      question.number,
      question.question,
      question.points,
      question.quizId
    );
    this.correctAnswer = question.correctAnswer;
    this.userAnswer = answer;
  }
}
```

quiz.js:

```
export class Quiz {
  constructor(
    id,
    title,
    imageUrl,
    description,
    timeLimit,
    date,
    minimumPointsPrc,
    ownerId
  ) {
    this.id = id;
    this.title = title;
    this.imageUrl = imageUrl;
```

```
    this.description = description;
    this.date = date;
    this.timeLimit = timeLimit;
    this.minimumPointsPrc = minimumPointsPrc;
    this.ownerId = ownerId;
  }
}

export class QuizResult {
  constructor(
    id,
    title,
    imageUrl,
    description,
    timeLimit,
    date,
    minimumPointsPrc,
    quizId,
    passed,
    questions
  ) {
    this.id = id;
    this.title = title;
    this.imageUrl = imageUrl;
    this.description = description;
    this.date = date;
    this.timeLimit = timeLimit;
    this.minimumPointsPrc = minimumPointsPrc;
    this.quizId = quizId;
    this.passed = passed;
    this.questions = questions;
  }
}
```

6. В screens са екраните, които изграждат мобилното приложение.

StartupScreen.js:

```
import React, { useEffect } from "react";
import { useDispatch } from "react-redux";
import LoadingControl from "../components/UI/LoadingControl";
import { validateAuthentication, getUserData } from "../service/authService";
import * as authActions from "../store/actions/auth";
import * as loadingActions from "../store/actions/loading";

export default StartupScreen = (props) => {
```

```
  const dispatch = useDispatch();

  useEffect(() => {
    const checkAuth = async () => {
      try {
        await validateAuthentication();
        const { userId } = await getUserData();
        dispatch(authActions.storeAuthentication(userId));
        dispatch(loadingActions.endLoading());
      } catch (err) {
        console.log(err.message);
        dispatch(authActions.logout());
        dispatch(loadingActions.endLoading());
      }
    };
    checkAuth();
  }, [validateAuthentication, dispatch]);
  return <LoadingControl />;
};
```

quiz/wizard/Question.js

```
import React, { useState, useEffect } from "react";
import {
  StyleSheet,
  View,
  Text,
  TouchableNativeFeedback,
  TextInput,
  Button,
  Dimensions,
  Alert,
} from "react-native";
import { Ionicons } from "@expo/vector-icons";
import * as componentUtils from "../../../components/utils";
import colors from "../../../constants/colors";
import { QUESTION_TYPE } from "../../../models/question";
const Question = (props) => {
  const [answerValue, setAnswerValue] = useState("");
  const [question, setQuestion] = useState({
    question: "",
    points: "",
    possibleAnswers: {},
    correctAnswer: "",
  });
  const [questionInputHeight, setQuestionInputHeight] = useState(30);
  const [correctAnswer, setCorrectAnswer] = useState(null);
```

```
const QuestionBodyComponent = componentUtils.getQuestionComponentByQuestionType(
  props.type
);
const handleAction = (action) => {
  setAnswerValue("");
  setQuestion({
    ...question,
    question: "",
    points: "",
    possibleAnswers: {},
    correctAnswer: "",
  });
  setQuestionInputHeight(30);
  setCorrectAnswer(null);
  action();
};

useEffect(() => {
  if (props.question) {
    setQuestion(props.question);
    const correctAns =
      props.type === QUESTION_TYPE.OPEN_QUESTION
        ? props.question.correctAnswer
        : Object.keys(props.question.possibleAnswers).find((key) => {
            return props.question.possibleAnswers[key].truthy;
          });
    setCorrectAnswer(correctAns);
  }
}, [props.question]);

const isQuestionValid = (question) => {
  let isValid = true;
  isValid = isValid && question.question;
  isValid = isValid && question.points && +question.points > 0;
  if (props.type === QUESTION_TYPE.SINGLE_CHOICE) {
    isValid = isValid && Object.keys(question.possibleAnswers).length >= 2;
    isValid =
      isValid &&
      Object.keys(question.possibleAnswers).some((key) => {
        return question.possibleAnswers[key].truthy;
      });
  } else {
    isValid = isValid && question.correctAnswer;
  }

  return isValid;
};
const addAnswer = (answer) => {
```

```
  if (!question.question) {
    Alert.alert("Question", "Please add your question");
    return;
  }
  if (!question.points) {
    Alert.alert("Number of points", "Please add number of points");
    return;
  }
  if (!answerValue) {
    Alert.alert("Answer", "Please add possible answer");
    return;
  }
  const updatedQuestion = question;
  const ansNum = Object.keys(updatedQuestion.possibleAnswers).length + 1;
  updatedQuestion.possibleAnswers = {
    ...updatedQuestion.possibleAnswers,
    [ansNum]: {
      answer: answer,
      truthy: false,
    },
  };
  setQuestion(updatedQuestion);
  setAnswerValue("");
};

const setPoints = (points) => {
  if (!+points) {
    points = 0;
  }
  setQuestion({
    ...question,
    points: +points,
  });
};
const setQuestionText = (text) => {
  setQuestion({
    ...question,
    question: text,
  });
};
const updateCorrectAnswer = (value) => {
  const updatedQuestion = question;
  if (props.type === QUESTION_TYPE.SINGLE_CHOICE) {
    Object.keys(updatedQuestion.possibleAnswers).forEach((key) => {
      updatedQuestion.possibleAnswers[key].truthy = false;
    });
    updatedQuestion.possibleAnswers[value].truthy = true;
  } else {
```

```
    updatedQuestion.correctAnswer = value;
  }

  setQuestion(updatedQuestion);
  setCorrectAnswer(value);
};

const removeAnswer = (index) => {
  let updatedQuestion = {
    ...question,
    possibleAnswers: {},
  };
  let ansNum = 0;
  setCorrectAnswer(null);
  Object.keys(question.possibleAnswers).forEach((key) => {
    if (key != index) {
      ansNum++;
      updatedQuestion.possibleAnswers[ansNum] = question.possibleAnswers[key];
      updatedQuestion.possibleAnswers[ansNum].truthy = false;
    }
  });
  setQuestion(updatedQuestion);
};

return (
  <View style={styles.container}>
    <View style={styles.inputContainer}>
      <View style={styles.titleContainer}>
        <Text style={styles.title}>Quiz wizard</Text>
      </View>
      <View style={styles.questionContainer}>
        <TextInput
          style={{ ...styles.question, height: questionInputHeight }}
          placeholder="Question..."
          multiline={true}
          onContentSizeChange={(e) =>
            setQuestionInputHeight(e.nativeEvent.contentSize.height)
          }
          onChangeText={(text) => setQuestionText(text)}
          value={question.question}
        />
      </View>
      <View style={styles.pointsContainer}>
        <Text style={styles.points}>Points:</Text>
        <TextInput
          style={styles.points}
          placeholder="points..."
          keyboardType="numeric"
```

```
            onChangeText={(text) => setPoints(text)}
            value={question.points === 0 ? "" : "" + question.points}
          />
        </View>
        <QuestionBodyComponent
          question={question}
          onSelectAnswer={updateCorrectAnswer}
          selectedAnswer={correctAnswer}
          onLongPress={removeAnswer}
          createMode
        />
        {props.type === QUESTION_TYPE.SINGLE_CHOICE && (
          <View style={styles.answerContainer}>
            <TextInput
              style={styles.answer}
              placeholder="answer..."
              onChangeText={(text) => setAnswerValue(text)}
              value={answerValue}
            />
          </View>
        )}
        {props.type === QUESTION_TYPE.SINGLE_CHOICE && (
          <TouchableNativeFeedback
            onPress={() => {
              addAnswer(answerValue);
            }}
          >
            <View style={styles.addQuestionButton}>
              <Ionicons name="md-add" size={42} color={colors.primary} />
            </View>
          </TouchableNativeFeedback>
        )}
      </View>
      <View style={styles.bottomContainer}>
        <TouchableNativeFeedback
          onPress={() => {
            handleAction(props.onRemoveQuestion);
          }}
        >
          <View style={styles.addQuestionButton}>
            <Ionicons name="ios-trash" size={32} color={colors.primary} />
          </View>
        </TouchableNativeFeedback>
        <View style={styles.counterContainer}>
          <Text style={styles.counter}>
            {props.questionNumber}/{props.questionsCount}
          </Text>
        </View>
```

```
        <View style={styles.buttonContainer}>
          {props.isLast && (
            <View style={styles.button}>
              <Button
                color={colors.activeColor}
                title="Add"
                onPress={() => {
                  if (!isQuestionValid(question)) {
                    Alert.alert("Question", "Please enter a valid question");
                  } else {
                    if (props.question) {
                      props.onUpdateQuestion({ ...question, type: props.type });
                    } else {
                      props.onCreateQuestion({ ...question, type: props.type });
                    }
                    handleAction(props.onAddQuestion);
                  }
                }}
              />
            </View>
          )}
          {!props.isLast && (
            <View style={styles.button}>
              <Button
                color={colors.activeColor}
                title=">>"
                onPress={() => {
                  if (!isQuestionValid(question)) {
                    Alert.alert("Question", "Please enter a valid question");
                  } else {
                    if (props.question) {
                      props.onUpdateQuestion({ ...question, type: props.type });
                    } else {
                      props.onCreateQuestion({ ...question, type: props.type });
                    }
                    handleAction(props.onNext);
                  }
                }}
              />
            </View>
          )}
          <View style={styles.button}>
            <Button
              color={colors.activeColor}
              title="EXIT"
              onPress={() => {
                if (!isQuestionValid(question)) {
                  Alert.alert(
```

```
                    "Question",
                    "Entered question is not valid and will be deleted if you
continue. Do you want to continue?",
                    [
                      {
                        text: "Ok",
                        onPress: () => {
                          handleAction(props.onExit);
                        },
                      },
                      { text: "Cancel" },
                    ]
                  );
                } else {
                  if (props.question) {
                    props.onUpdateQuestion({ ...question, type: props.type });
                  } else {
                    props.onCreateQuestion({ ...question, type: props.type });
                  }
                  handleAction(props.onExit);
                }
              }}
            />
          </View>
          {!props.isFirst && (
            <View style={styles.button}>
              <Button
                color={colors.activeColor}
                title="<<"
                onPress={() => {
                  if (!isQuestionValid(question)) {
                    Alert.alert("Question", "Please enter a valid question");
                  } else {
                    if (props.question) {
                      props.onUpdateQuestion({ ...question, type: props.type });
                    } else {
                      props.onCreateQuestion({ ...question, type: props.type });
                    }
                    handleAction(props.onPrevious);
                  }
                }}
              />
            </View>
          )}
        </View>
      </View>
    </View>
  );
```

```
};

const styles = StyleSheet.create({
  container: {
    flexGrow: 1,
    backgroundColor: colors.backColor,
    alignItems: "center",
  },
  inputContainer: {
    flexGrow: 1,
    alignItems: "center",
    justifyContent: "flex-start",
    width: "100%",
  },
  bottomContainer: {
    margin: 20,
    alignItems: "center",
    paddingHorizontal: 20,
    paddingVertical: 10,
    width: "100%",
  },
  buttonContainer: {
    flexDirection: "row-reverse",
    justifyContent: "space-between",
    alignItems: "center",
    width: "100%",
  },
  titleContainer: {
    width: "100%",
    backgroundColor: colors.primary,
    justifyContent: "flex-start",
    alignItems: "center",
  },
  title: {
    fontFamily: "open-sans-bold",
    fontSize: 23,
    color: colors.inactiveColor,
    padding: 15,
    textShadowColor: colors.activeColor,
    textShadowOffset: { width: 2, height: 2 },
    textShadowRadius: 1,
  },
  question: {
    fontSize: 18,
    fontFamily: "open-sans-bold",
    borderBottomColor: colors.greyish,
    borderBottomWidth: 2,
    //marginVertical: 5,
```

```
  //textAlign: "center",
},
questionContainer: {
  margin: 10,
  paddingHorizontal: 20,
  //paddingVertical: 10,
  width: "100%",
},
pointsContainer: {
  flexDirection: "row",
  justifyContent: "center",
  alignItems: "center",
  borderBottomColor: colors.greyish,
  borderBottomWidth: 2,
  marginBottom: 20,
},
points: {
  fontSize: 16,

  fontFamily: "open-sans-bold",

  paddingHorizontal: 5,
},
counterContainer: {
  paddingVertical: 10,
},
counter: {
  fontFamily: "open-sans",
  fontSize: 20,
  color: colors.greyish,
},
answer: {
  fontSize: 16,
  fontFamily: "open-sans",
  borderBottomColor: colors.greyish,
  borderBottomWidth: 1,
  //textAlign: "center",
},
answerContainer: {
  //margin: 10,
  paddingHorizontal: 20,
  //paddingVertical: 10,
  width: "100%",
},
addQuestionButton: {
  justifyContent: "center",
  alignItems: "center",
  borderColor: colors.primary,
```

```
    borderWidth: 1,
    borderRadius: 20,
    width: 40,
    height: 40,
    marginTop: 10,
  },

  button: {
    width: Dimensions.get("window").width / 4,
  },
});

export default Question;
```

quiz/wizard/QuizHeader.js

```
import React, { useCallback, useReducer, useRef, useEffect } from "react";
import {
  View,
  Text,
  Button,
  StyleSheet,
  Dimensions,
  Switch,
} from "react-native";
import colors from "../../../constants/colors";
import Input from "../../../components/UI/Input";

const FORM_UPDATE = "UPDATE";
const FORM_LOAD = "FORM_LOAD";
const formReducer = (state, action) => {
  switch (action.type) {
    case FORM_UPDATE:
      const updatedInputValues = {
        ...state.inputValues,
        [action.input]: action.value,
      };
      const updatedInputValidities = {
        ...state.inputValidities,
        [action.input]: action.isValid,
      };
      let updatedFormIsValid = true;
      for (const key in updatedInputValidities) {
        updatedFormIsValid = updatedFormIsValid && updatedInputValidities[key];
      }

      return {
```

```
        inputValues: updatedInputValues,
        inputValidities: updatedInputValidities,
        formIsValid: updatedFormIsValid,
      };
    case FORM_LOAD:
      const loadedState = {
        inputValues: {
          title: action.quiz.title,
          description: action.quiz.description,
          imageUrl: action.quiz.imageUrl,
          minimumPoints: action.quiz.minimumPointsPrc,
          withTimeLimit: !!action.quiz.timeLimit,
          timeLimitMinutes: !!action.quiz.timeLimit
            ? "" + action.quiz.timeLimit.min
            : 0,
          timeLimitSeconds: !!action.quiz.timeLimit
            ? "" + action.quiz.timeLimit.sec
            : 0,
        },
        inputValidities: {
          title: true,
          description: true,
          imageUrl: true,
          minimumPoints: true,
          withTimeLimit: true,
          timeLimitMinutes: true,
          timeLimitSeconds: true,
        },
        formIsValid: true,
      };
      console.log(loadedState);
      return loadedState;
    default:
      return state;
  }
};

const QuizHeader = (props) => {
  const [formState, dispatchFormState] = useReducer(formReducer, {
    inputValues: {
      title: "",
      description: "",
      imageUrl: "",
      minimumPoints: 0,
      withTimeLimit: false,
      timeLimitMinutes: 0,
      timeLimitSeconds: 0,
    },
```

```
  inputValidities: {
    title: false,
    description: false,
    imageUrl: true,
    minimumPoints: false,
    withTimeLimit: true,
    timeLimitMinutes: true,
    timeLimitSeconds: true,
  },
  formIsValid: false,
});

const inputChangeHandler = useCallback(
  (inputIdentifier, inputValue, inputIsValid) => {
    dispatchFormState({
      type: FORM_UPDATE,
      value: inputValue,
      isValid: inputIsValid,
      input: inputIdentifier,
    });
  },
  [dispatchFormState]
);

useEffect(() => {
  console.log(props.quiz);
  if (props.quiz) {
    dispatchFormState({
      type: FORM_LOAD,
      quiz: props.quiz,
    });
  }
}, [props]);
const timeLimitMinutesInput = useRef(null);
const timeLimitSecondsInput = useRef(null);

const setWithTimeLimit = (withTimeLimit) => {
  inputChangeHandler("withTimeLimit", withTimeLimit, true);
  inputChangeHandler("timeLimitMinutes", 0, !withTimeLimit);
  inputChangeHandler("timeLimitSeconds", 0, !withTimeLimit);
  if (timeLimitMinutesInput.current && !withTimeLimit) {
    timeLimitMinutesInput.current.resetInput();
  }
  if (timeLimitSecondsInput.current && !withTimeLimit) {
    timeLimitSecondsInput.current.resetInput();
  }
};
```

```
return (
  <View style={styles.container}>
    <View style={styles.titleContainer}>
      <Text style={styles.title}>Quiz wizard</Text>
    </View>
    <View style={styles.bodyContainer}>
      <View style={styles.inputContainer}>
        <Input
          id="title"
          label="Title:"
          maxLength={50}
          minLength={5}
          placeholder="Enter quiz title"
          style={styles.input}
          onInputChange={inputChangeHandler}
          validationText="Please enter valid quiz title"
          initialValue={formState.inputValues.title}
          initiallyValid={formState.inputValidities.title}
        />
        <Input
          id="description"
          label="Description:"
          placeholder="Enter quiz description"
          style={styles.input}
          onInputChange={inputChangeHandler}
          validationText="Please enter valid quiz description"
          multiline={true}
          initialValue={formState.inputValues.description}
          initiallyValid={formState.inputValidities.description}
        />
        <Input
          id="imageUrl"
          autoCapitalize={"none"}
          label="Image url:"
          placeholder="Enter image url"
          style={styles.input}
          validationText="Please enter valid image url"
          onInputChange={inputChangeHandler}
          url
          initialValue={formState.inputValues.imageUrl}
          initiallyValid={formState.inputValidities.imageUrl}
        />
        <View style={styles.rowContainer}>
          <Input
            id="minimumPoints"
            label="Minimum points (%):"
            min={0}
            validationText="Enter valid minimum points %"
```

```
            required={true}
            max={100}
            positionHorizontal={true}
            keyboardType="numeric"
            placeholder="points %"
            onInputChange={inputChangeHandler}
            validationText="Please enter valid value for minimum points %"
            style={styles.input}
            initialValue={
              formState.inputValues.minimumPoints === 0
                ? ""
                : "" + formState.inputValues.minimumPoints
            }
            initiallyValid={formState.inputValidities.minimumPoints}
          />
        </View>
        <View style={styles.rowContainer}>
          <Text style={styles.label}>Use Time Limit:</Text>
          <Switch
            trackColor={{ false: colors.greyish, true: colors.primary }}
            thumbColor={
              formState.inputValues.withTimeLimit
                ? colors.activeColor
                : colors.backColor
            }
            ios_backgroundColor="#ccc"
            onValueChange={setWithTimeLimit}
            value={formState.inputValues.withTimeLimit}
          />
        </View>
        {formState.inputValues.withTimeLimit && (
          <View>
            <Input
              getRef={(ref) => (timeLimitMinutesInput.current = ref.current)}
              id="timeLimitMinutes"
              label="Minutes:"
              min={0}
              required={true}
              positionHorizontal={true}
              keyboardType="numeric"
              placeholder="min"
              onInputChange={inputChangeHandler}
              validationText="Please enter valid value"
              initialValue={formState.inputValues.timeLimitMinutes}
              initiallyValid={formState.inputValidities.timeLimitMinutes}
              style={styles.input}
            />
            <Input
```

```
            getRef={(ref) => (timeLimitSecondsInput.current = ref.current)}
            id="timeLimitSeconds"
            label="Seconds:"
            min={0}
            required={true}
            positionHorizontal={true}
            keyboardType="numeric"
            placeholder="sec"
            onInputChange={inputChangeHandler}
            validationText="Please enter valid value"
            initialValue={formState.inputValues.timeLimitSeconds}
            initiallyValid={formState.inputValidities.timeLimitSeconds}
            style={styles.input}
            style={styles.input}
          />
        </View>
      )}
      <View style={styles.infoContainer}>
        <Text style={styles.infoLabel}>
          Total Questions: {props.questionsCount ?? 0}
        </Text>
        {formState.formIsValid && (
          <View style={styles.button}>
            <Button
              color={colors.activeColor}
              title="Add Question"
              onPress={() => {
                console.log(`ADD QUIZ: ${JSON.stringify(formState)}`);
                props.onCreateUpdateQuizHeader({
                  title: formState.inputValues.title,
                  imageUrl: formState.inputValues.imageUrl,
                  description: formState.inputValues.description,
                  timeLimit: formState.inputValues.withTimeLimit
                    ? {
                        min: +formState.inputValues.timeLimitMinutes,
                        sec: +formState.inputValues.timeLimitSeconds,
                      }
                    : null,
                  minimumPointsPrc: formState.inputValues.minimumPoints,
                });
                props.onAddQuestion();
              }}
            />
          </View>
        )}
      </View>
    </View>
    {formState.formIsValid && props.questionsCount > 1 && (
```

```
          <View style={styles.buttonContainer}>
            <View style={styles.button}>
              <Button
                color={colors.activeColor}
                title="Submit"
                onPress={() => {
                  props.onCreateUpdateQuizHeader({
                    title: formState.inputValues.title,
                    imageUrl: formState.inputValues.imageUrl,
                    description: formState.inputValues.description,
                    timeLimit: formState.inputValues.withTimeLimit
                      ? {
                          min: +formState.inputValues.timeLimitMinutes,
                          sec: +formState.inputValues.timeLimitSeconds,
                        }
                      : null,
                    minimumPointsPrc: formState.inputValues.minimumPoints,
                  });
                  props.onSubmit();
                }}
              />
            </View>
          </View>
        )}
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flexGrow: 1,
    backgroundColor: colors.backColor,
  },
  bodyContainer: {
    flexGrow: 1,
    backgroundColor: colors.backColor,
    justifyContent: "space-between",
  },
  inputContainer: {
    margin: 20,
  },
  titleContainer: {
    width: "100%",
    backgroundColor: colors.primary,
    justifyContent: "flex-start",
    alignItems: "center",
  },
```

```
  title: {
    fontFamily: "open-sans-bold",
    fontSize: 23,
    color: colors.inactiveColor,
    padding: 15,
    textShadowColor: colors.activeColor,
    textShadowOffset: { width: 2, height: 2 },
    textShadowRadius: 1,
  },
  text: {
    fontFamily: "open-sans",
    fontSize: 18,
  },
  label: {
    fontFamily: "open-sans-bold",
    marginVertical: 8,
  },
  infoLabel: {
    fontFamily: "open-sans-bold",
    fontSize: 18,
  },
  input: {
    borderBottomColor: "#ccc",
    borderBottomWidth: 1,
    paddingHorizontal: 2,
    marginVertical: 10,
    paddingVertical: 5,
  },
  rowContainer: {
    flexDirection: "row",
    justifyContent: "flex-start",
    alignItems: "center",
  },
  infoContainer: {
    flexDirection: "row",
    justifyContent: "space-between",
    paddingVertical: 10,
    margin: 5,
    alignItems: "center",
    borderColor: colors.greyish,
    borderTopWidth: 2,
  },
  buttonContainer: {
    flexGrow: 1,
    flexDirection: "row-reverse",
    padding: 20,
    justifyContent: "center",
    width: Dimensions.get("window").width,
```

```
    alignItems: "flex-end",
    color: colors.activeColor,
  },
  button: {
    // width: Dimensions.get("window").width / 4,
  },
});

export default QuizHeader;
```

quiz/wizard/QuizWizard.js

```
import React, { useState } from "react";
import { Modal, Alert, StyleSheet, ScrollView } from "react-native";
import QuizHeader from "./QuizHeader";
import Question from "./Question";
import { Quiz } from "../../../models/quiz";
import {
  QUESTION_TYPE,
  SingleChoiceQuestion,
  OpenQuestion,
} from "../../../models/question";
export const STATE = {
  CREATE_UPDATE_QUIZ: "CREATE_UPDATE_QUIZ",
  ADD_QUESTIONS: "ADD_QUESTIONS",
};

const QuizWizard = (props) => {
  const [screenState, setScreenState] = useState(STATE.CREATE_UPDATE_QUIZ);
  const [quiz, setQuiz] = useState(null);
  const [questions, setQuestions] = useState([]);
  const [questionType, setQuestionType] = useState(QUESTION_TYPE.SINGLE_CHOICE);
  const [position, setPosition] = useState(-1);
  const [currQuestion, setCurrQuestion] = useState(null);
  const [questionsCount, setQuestionsCount] = useState(0);

  const createUpdateQuizHeaderHandler = (quiz) => {
    console.log(`Minimum points: ${quiz.minimumPointsPrc}`);
    const newQuiz = new Quiz(
      0,
      quiz.title,
      quiz.imageUrl,
      quiz.description,
      quiz.timeLimit,
      new Date(Date.now()),
      quiz.minimumPointsPrc,
```

```
    ""
  );
  console.log(newQuiz);
  setQuiz(newQuiz);
};

const addQuestionHandlerFromHeader = () => {
  if (questions.length === 0) {
    addQuestionHandler();
  } else {
    setScreenState(STATE.ADD_QUESTIONS);
    setCurrQuestion(questions[position]);
    setQuestionType(questions[position].type);
  }
};

const addQuestionHandler = () => {
  Alert.alert("Question type", "What kind of question do you want to add ?", [
    {
      text: "Open question",
      onPress: () => {
        setQuestionType(QUESTION_TYPE.OPEN_QUESTION);
        setScreenState(STATE.ADD_QUESTIONS);
        setQuestionsCount(questionsCount + 1);
        setPosition(position + 1);
        setCurrQuestion(null);
      },
    },
    {
      text: "Single choice question",
      onPress: () => {
        setQuestionType(QUESTION_TYPE.SINGLE_CHOICE);
        setScreenState(STATE.ADD_QUESTIONS);
        setQuestionsCount(questionsCount + 1);
        setPosition(position + 1);
        setCurrQuestion(null);
      },
    },
  ]);
};

const createQuestionHandler = (question) => {
  setQuestions(questions.concat(question));
};

const updateQuestionHandler = (question) => {
  const updatedQuestions = questions;
  updatedQuestions[position] = question;
```

```
  setQuestions(updatedQuestions);
};

const removeQuestionHandler = () => {
  const updatedQuestions = questions.filter((q, indx) => {
    return indx != position;
  });
  let newPosition = position;
  if (position >= updatedQuestions.length && position > 0) {
    newPosition = updatedQuestions.length - 1;
  }
  setQuestions(updatedQuestions);
  setPosition(newPosition);
  setCurrQuestion(updatedQuestions[newPosition] ?? null);
  setQuestionType(
    updatedQuestions[newPosition]
      ? updatedQuestions[newPosition].type
      : QUESTION_TYPE.SINGLE_CHOICE
  );
  setQuestionsCount(questionsCount - 1);
};

const exitHandler = () => {
  setScreenState(STATE.CREATE_UPDATE_QUIZ);
  setCurrQuestion(null);
};

const previousHandler = () => {
  const updatedQuestions = questions;
  const newPosition = position - 1;
  setPosition(newPosition);
  setCurrQuestion(updatedQuestions[newPosition] ?? null);
  setQuestionType(updatedQuestions[newPosition].type);
};

const nextHandler = () => {
  const newPosition = position + 1;
  setPosition(newPosition);
  setCurrQuestion(questions[newPosition] ?? null);
  setQuestionType(questions[newPosition].type);
};

let content = null;
switch (screenState) {
  case STATE.CREATE_UPDATE_QUIZ:
    content = (
      <QuizHeader
        quiz={quiz}
```

```
        onCreateUpdateQuizHeader={createUpdateQuizHeaderHandler}
        onAddQuestion={addQuestionHandlerFromHeader}
        onSubmit={async () => {
          let number = 0;
          let quizQuestions = questions.map((q) => {
            let question = null;
            number += 1;
            switch (q.type) {
              case QUESTION_TYPE.SINGLE_CHOICE:
                question = new SingleChoiceQuestion(
                  0,
                  number,
                  q.question,
                  q.points,
                  ""
                );
                question.possibleAnswers = q.possibleAnswers;
                break;
              case QUESTION_TYPE.OPEN_QUESTION:
                question = new OpenQuestion(
                  0,
                  number,
                  q.question,
                  q.points,
                  ""
                );
                question.setCorrectAnswer(q.correctAnswer);
                break;
            }
            return question;
          });
          await props.onSubmitQuiz(quiz, quizQuestions);
        }}
        questionsCount={questions.length}
      />
    );
    break;
  case STATE.ADD_QUESTIONS:
    content = (
      <Question
        type={questionType}
        isFirst={position === 0}
        isLast={questions.length <= position + 1}
        questionNumber={position + 1}
        questionsCount={questionsCount}
        onPrevious={previousHandler}
        onExit={exitHandler}
        onNext={nextHandler}
```

```
          question={currQuestion}
          onAddQuestion={addQuestionHandler}
          onCreateQuestion={createQuestionHandler}
          onUpdateQuestion={updateQuestionHandler}
          onRemoveQuestion={removeQuestionHandler}
        />
      );
  }
  console.log(questions);
  console.log(position);
  console.log(currQuestion);
  return (
    <Modal
      transparent={false}
      animationType={"slide"}
      onRequestClose={() => {
        Alert.alert(
          "Exit quiz wizard",
          "If you exit the wizard your changes would not be saved. Do you want to exit
?",
          [
            { text: "Cancel" },
            {
              text: "Exit",
              onPress: () => {
                props.onGoBack();
              },
            },
          ]
        );
      }}
    >
      <ScrollView contentContainerStyle={styles.scrollViewContent}>
        {content}
      </ScrollView>
    </Modal>
  );
};

const styles = StyleSheet.create({
  scrollViewContent: {
    flexGrow: 1,
  },
});
export default QuizWizard;
```

quiz/ Home.js

```
import React, { useCallback, useEffect, useState } from "react";
import { useSelector, useDispatch } from "react-redux";
import { View, Text, StyleSheet, Button } from "react-native";
import QuizList from "./QuizList";
import Quiz from "./Quiz";
import * as quizActions from "../../store/actions/quiz";
import * as quizService from "../../service/quizService";
import { getQuizExample } from "../../data/dummyData";
import colors from "../../constants/colors";
import LoadingControl from "../../components/UI/LoadingControl";
import { QUIZ_STATUS } from "../../service/quizService";
import QuizWizard from "./wizard/QuizWizard";

export const STATE = {
  NONE: "NONE",
  TAKE_QUIZ: "TAKE_QUIZ",
  REVIEW_QUIZ: "REVIEW_QUIZ",
  CREATE_QUIZ: "CREATE_QUIZ",
};

export default HomeScreen = (props) => {
  const [quiz, setQuiz] = useState(null);
  const [screenState, setScreenState] = useState(STATE.NONE);

  const dispatch = useDispatch();
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState();
  const passedQuizzes = useSelector((state) => state.quiz.passedQuizzes).sort(
    (q1, q2) => {
      if (q1.date < q2.date) return 1;
      if (q1.date > q2.date) return -1;
      return 0;
    }
  );
  const quizzes = useSelector((state) => state.quiz.availableQuizzes)
    .sort((q1, q2) => {
      if (q1.date < q2.date) return 1;
      if (q1.date > q2.date) return -1;
      return 0;
    })
    .map((q) => {
      if (passedQuizzes.some((p) => p.quizId === q.id)) {
        let passed = passedQuizzes.find((p) => p.quizId === q.id).passed;
        return {
          ...q,
          passed: passed,
```

```
      };
    }
    return q;
  });

const loadQuizzes = useCallback(async () => {
  setError(null);
  try {
    await dispatch(quizActions.getQuizzes());
  } catch (err) {
    setError(err);
  }
}, [dispatch, setIsLoading]);

useEffect(() => {
  setIsLoading(true);
  loadQuizzes().then(() => {
    setIsLoading(false);
  });
}, [loadQuizzes]);

if (isLoading) {
  return <LoadingControl />;
}

if (error) {
  return (
    <View style={styles.centered}>
      <Text>{error.message}</Text>
      <Button
        title="Try again"
        onPress={loadQuizzes}
        color={colors.primary}
      />
    </View>
  );
}

if (quizzes.length === 0) {
  return (
    <View style={styles.centered}>
      <Text>There is no quizzes</Text>
      <Button
        title="ADD DEFAULT"
        onPress={async () => {
          try {
            const data = getQuizExample();
            const newQuiz = await quizService.insertFullQuiz(
```

```
              data.quiz,
              data.questions
            );
            dispatch(quizActions.addQuiz(newQuiz));
          } catch (e) {
            setError(e);
          }
        }}
      />
    </View>
  );
}

const takeQuizHandler = (quiz) => {
  setQuiz(quiz);
  setScreenState(STATE.TAKE_QUIZ);
};
const createQuizHandler = () => {
  console.log("createQuizHandler");
  setScreenState(STATE.CREATE_QUIZ);
};
const reviewQuizHandler = (quiz) => {
  quiz = passedQuizzes.find((p) => p.quizId === quiz.id);
  setScreenState(STATE.REVIEW_QUIZ);
  setQuiz(quiz);
};
const returnToMainScreen = () => {
  setScreenState(STATE.NONE);
  setQuiz(null);
};

const handleSubmitQuiz = async (quiz, questions) => {
  const newQuiz = await quizService.insertFullQuiz(quiz, questions);
  dispatch(quizActions.addQuiz(newQuiz));
  setScreenState(STATE.NONE);
  setQuiz(null);
};
let content = null;
switch (screenState) {
  case STATE.NONE:
    content = (
      <QuizList
        quizzes={quizzes}
        onTakeQuiz={takeQuizHandler}
        onViewResults={reviewQuizHandler}
        onCreateQuiz={createQuizHandler}
      />
    );
```

```
      break;
    case STATE.TAKE_QUIZ:
      content = (
        <Quiz
          requestedState={QUIZ_STATUS.INIT}
          quiz={quiz}
          onGoBack={returnToMainScreen}
        />
      );
      break;
    case STATE.REVIEW_QUIZ:
      content = (
        <Quiz
          requestedState={QUIZ_STATUS.FINISHED}
          quiz={quiz}
          onGoBack={returnToMainScreen}
        />
      );
      break;
    case STATE.CREATE_QUIZ:
      content = (
        <QuizWizard
          onGoBack={returnToMainScreen}
          onSubmitQuiz={handleSubmitQuiz}
        />
      );
      break;
  }
  return <View style={styles.container}>{content}</View>;
};
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center",
  },
});
```

quiz/ Quiz.js

```
import React, { useState, useEffect, useCallback } from "react";
import { useDispatch } from "react-redux";
import {
  View,
  Text,
  StyleSheet,
```

```
  Button,
  Image,
  Dimensions,
  Modal,
  ScrollView,
  Alert,
} from "react-native";
import colors from "../../constants/colors";
import Question from "../../components/quiz/question/Question";
import QuizResult from "./QuizResult";
import LoadingControl from "../../components/UI/LoadingControl";
import Timer from "../../components/UI/Timer";
import {
  QUIZ_STATUS,
  fetchQuestions,
  insertQuizResults,
} from "../../service/quizService";
import { addResults } from "../../store/actions/quiz";

const Quiz = (props) => {
  const dispatch = useDispatch();
  const [quizQuestions, setQuizQuestions] = useState([]);
  const [position, setPosition] = useState(0);
  const [quizStatus, setQuizStatus] = useState(QUIZ_STATUS.NONE);
  const [currQuestion, setCurrQuestion] = useState(null);
  const [startingTime, setStartingTime] = useState(Date.now());

  const loadQuestions = useCallback(async () => {
    try {
      const loadedQuestions = await fetchQuestions(props.quiz.id);
      loadedQuestions.sort((q1, q2) => {
        if (q1.number < q2.number) return -1;
        if (q1.number > q2.number) return 1;
        return 0;
      });
      setQuizQuestions(loadedQuestions);
    } catch (err) {
      setQuizStatus(QUIZ_STATUS.ERROR);
    }
  }, [fetchQuestions, setQuizQuestions, setQuizStatus]);

  useEffect(() => {
    setQuizStatus(QUIZ_STATUS.NONE);
    if (props.requestedState === QUIZ_STATUS.FINISHED) {
      setQuizQuestions(props.quiz.questions);
      setQuizStatus(QUIZ_STATUS.FINISHED);
    } else {
      loadQuestions().then(() => {
```

```
      setQuizStatus(QUIZ_STATUS.INIT);
    });
  }
}, [loadQuestions, props.requestedState]);

if (quizStatus == QUIZ_STATUS.NONE) {
  return <LoadingControl />;
}

if (quizStatus == QUIZ_STATUS.ERROR) {
  return (
    <View style={styles.centered}>
      <Text>{error.message}</Text>
      <Button
        title="Try again"
        onPress={loadQuestions}
        color={colors.primary}
      />
    </View>
  );
}

const dim = Dimensions.get("window");
const timeMessage = props.quiz.timeLimit
  ? (props.quiz.timeLimit.min ? `${props.quiz.timeLimit.min} min.` : "") +
    (props.quiz.timeLimit.sec ? `${props.quiz.timeLimit.sec} sec.` : "")
  : "";

const startQuiz = () => {
  if (!quizQuestions || quizQuestions.length === 0) {
    setQuizStatus(QUIZ_STATUS.FINISHED);
    return;
  }
  setPosition(0);
  setCurrQuestion(quizQuestions[0] ?? null);
  setQuizStatus(QUIZ_STATUS.STARTED);
  setStartingTime(Date.now());
};

const onNextQuestionHandler = (answer) => {
  if (!quizStatus !== QUIZ_STATUS.REVIEW) {
    const updatedQuestions = quizQuestions;
    updatedQuestions[position] = currQuestion.setAnswer(answer);
    setQuizQuestions(updatedQuestions);
  }
  const newPosition = position + 1;
  setPosition(newPosition);
  setCurrQuestion(quizQuestions[newPosition] ?? null);
```

```
};
const onPreviousQuestionHandler = (answer) => {
  if (!quizStatus !== QUIZ_STATUS.REVIEW) {
    const updatedQuestions = quizQuestions;
    updatedQuestions[position] = currQuestion.setAnswer(answer);
    setQuizQuestions(updatedQuestions);
  }
  const newPosition = position - 1;
  setPosition(newPosition);
  setCurrQuestion(quizQuestions[newPosition] ?? null);
};
const onSubmitHandler = (answer) => {
  const updatedQuestions = quizQuestions;
  updatedQuestions[position] = currQuestion.setAnswer(answer);
  setQuizQuestions(updatedQuestions);
  setCurrQuestion(updatedQuestions[position]);
  setQuizStatus(QUIZ_STATUS.NONE);
  insertQuizResults(props.quiz, updatedQuestions)
    .then((newQuizResult) => {
      dispatch(addResults(newQuizResult));
      setQuizStatus(QUIZ_STATUS.FINISHED);
    })
    .catch((err) => {
      Alert.alert("Something went wrong", err.message, [
        {
          text: "OK",
          onPress: () => {
            setQuizStatus(QUIZ_STATUS.INIT);
            props.onGoBack();
          },
        },
      ]);
    });
};

const endQuizHandler = () => {
  setQuizStatus(QUIZ_STATUS.INIT);
  props.onGoBack();
};

const reviewQuestionsHandler = () => {
  setPosition(0);
  setCurrQuestion(quizQuestions[0] ?? null);
  setQuizStatus(QUIZ_STATUS.REVIEW);
};

const timeoutHandler = () => {
  const updatedQuestions = quizQuestions;
```

```
  setQuizStatus(QUIZ_STATUS.NONE);
  insertQuizResults(props.quiz, updatedQuestions)
    .then((newQuizResult) => {
      Alert.alert("Time is up ", "Time is up.Your result has been saved.", [
        {
          text: "OK",
          onPress: () => {
            dispatch(addResults(newQuizResult));
            setQuizStatus(QUIZ_STATUS.FINISHED);
          },
        },
      ]);
    })
    .catch((err) => {
      Alert.alert("Something went wrong", `${err.message}`, [
        {
          text: "OK",
          onPress: () => {
            setQuizStatus(QUIZ_STATUS.INIT);
            props.onGoBack();
          },
        },
      ]);
    });
};

const onQuestionExitHandler = () => {
  if (quizStatus === QUIZ_STATUS.REVIEW) {
    setQuizStatus(QUIZ_STATUS.FINISHED);
    return;
  }
  endQuizHandler();
};

console.log(`TIME MESSAGE: ${timeMessage}`);
console.log(!!timeMessage);
let content = (
  <View style={styles.container}>
    <Image
      style={styles.image}
      source={{
        uri: props.quiz.imageUrl,
        height: dim.width / 2,
        width: dim.height / 2,
      }}
    />
    <View style={styles.infoContainer}>
      <Text style={styles.info}>{`${props.quiz.description}`}</Text>
```

```
        {!!timeMessage && (
          <Text
            style={styles.info}
          >{`You will have ${timeMessage} to pass it. You can not leave the test once
started.`}</Text>
        )}
      </View>
      <View style={styles.startButton}>
        <Button color={colors.activeColor} title="START" onPress={startQuiz} />
      </View>
    </View>
  );

  if (quizStatus == QUIZ_STATUS.STARTED || quizStatus == QUIZ_STATUS.REVIEW) {
    content = (
      <Question
        question={currQuestion}
        isLast={quizQuestions.length === position + 1}
        isFirst={position === 0}
        reviewMode={quizStatus === QUIZ_STATUS.REVIEW}
        onNext={onNextQuestionHandler}
        onPrevious={onPreviousQuestionHandler}
        onSubmit={onSubmitHandler}
        onExit={onQuestionExitHandler}
      />
    );
  }
  if (quizStatus == QUIZ_STATUS.FINISHED) {
    content = (
      <QuizResult
        userQuestions={quizQuestions}
        quiz={props.quiz}
        onExit={endQuizHandler}
        onReview={reviewQuestionsHandler}
      />
    );
  }
  return (
    <Modal
      transparent={false}
      animationType={"slide"}
      onRequestClose={() => {
        if (quizStatus !== QUIZ_STATUS.STARTED) props.onGoBack();
      }}
    >
      <ScrollView contentContainerStyle={styles.scrollViewContent}>
        <View style={styles.container}>
          <View style={styles.titleContainer}>
```

```
            <Text style={styles.title}>{`${props.quiz.title}`}</Text>
          </View>
          {quizStatus === QUIZ_STATUS.STARTED && props.quiz.timeLimit ? (
            <Timer
              timeInSeconds={
                props.quiz.timeLimit.min * 60 + props.quiz.timeLimit.sec
              }
              startingTime={startingTime}
              onTimeout={timeoutHandler}
            />
          ) : null}
          {content}
        </View>
      </ScrollView>
    </Modal>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "flex-start",
    alignItems: "center",
    backgroundColor: colors.backColor,
  },
  scrollViewContent: {
    flexGrow: 1,
  },
  title: {
    fontFamily: "open-sans-bold",
    fontSize: 23,
    color: colors.inactiveColor,
    padding: 15,
    textShadowColor: colors.activeColor,
    textShadowOffset: { width: 2, height: 2 },
    textShadowRadius: 1,
  },
  titleContainer: {
    width: "100%",
    backgroundColor: colors.primary,
    justifyContent: "flex-start",
    alignItems: "center",
  },
  info: {
    fontFamily: "open-sans",
    fontSize: 16,
    color: colors.activeColor,
    padding: 15,
```

```
    textAlignVertical: "center",
  },
  infoContainer: {
    justifyContent: "center",
    alignItems: "center",
  },
  image: {
    resizeMode: "contain",
    margin: 15,
  },
  startButton: {
    width: Dimensions.get("window").width / 4,
    margin: 20,
  },
});

export default Quiz;
```

quiz/ QuizList.js

```
import React from "react";

import { View, Text, FlatList, StyleSheet } from "react-native";

import QuizItem from "../../components/quiz/QuizItem";
import FloatingPlusButton from "../../components/UI/FloatingPlusButton";

const QuizList = (props) => {
  return (
    <View style={styles.list}>
      <FlatList
        data={props.quizzes}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
          <QuizItem
            quiz={item}
            showTimeLimit={true}
            isTaken={(item.passed ?? null) === null ? false : true}
            onTakeQuiz={props.onTakeQuiz}
            onViewResults={props.onViewResults}
          />
        )}
      />
      <FloatingPlusButton onPress={props.onCreateQuiz} iconName="ios-add" />
    </View>
  );
};
```

```
const styles = StyleSheet.create({
  list: {
    flex: 1,
    width: "100%",
  },
  centered: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
});

export default QuizList;
```

quiz/ QuizResult.js

```
import React, { useState } from "react";
import {
  View,
  Text,
  StyleSheet,
  Image,
  Button,
  Dimensions,
  Alert,
} from "react-native";
import colors from "../../constants/colors";
import { Ionicons } from "@expo/vector-icons";
import * as quizService from "../../service/quizService";
import * as printService from "../../service/printingService";
import FloatingPlusButton from "../../components/UI/FloatingPlusButton";

const QuizResult = (props) => {
  const dim = Dimensions.get("window");
  const [showPrint, setShowPrint] = useState(true);
  let totalPoints = quizService.calculateQuizTotalPoints(props.userQuestions);
  let userPoints = quizService.calculateQuizEarnedPoints(props.userQuestions);

  let score = 100;
  if (totalPoints) {
    score = Math.round((userPoints / totalPoints) * 10000, 2) / 100;
  }
  const passed = quizService.isPassed(props.quiz, props.userQuestions);
  const printQuizHandler = () => {
    setShowPrint(false);
    printService
```

```
    .printSingleQuizResult(props.quiz)
    .then(() => {
      setShowPrint(true);
    })
    .catch((err) => {
      setShowPrint(true);
      console.log(err);
      //  Alert.alert("Error Printing", err);
    });
};
const message = passed
  ? "Congratulations! You passed the test!"
  : "Sorry ! You didn't pass the test!";
const iconName = passed
  ? "md-checkmark-circle-outline"
  : "md-close-circle-outline";

console.log(props.quiz);
return (
  <View style={styles.screen}>
    <View style={{ alignItems: "center" }}>
      <Image
        style={styles.image}
        source={{
          uri: props.quiz.imageUrl,
          height: dim.width / 2,
          width: dim.height / 2,
        }}
      />
      <Text style={passed ? styles.successText : styles.failedText}>
        {message}
      </Text>
      <Text style={passed ? styles.successText : styles.failedText}>
        {score}%
      </Text>
      <Ionicons
        name={iconName}
        color={passed ? colors.greenish : colors.primary}
        size={50}
      />
      {showPrint && (
        <FloatingPlusButton
          onPress={printQuizHandler}
          iconName="ios-print"
          color={colors.activeColor}
          static
        />
      )}
```

```
      </View>
      <View>
        <Text style={styles.text}>Total Points: {totalPoints}</Text>
        <Text style={styles.text}>Your Points: {userPoints}</Text>
        <Text style={styles.text}>Score: {score}%</Text>
        <Text style={styles.text}>
          Target Score: {props.quiz.minimumPointsPrc}%
        </Text>
      </View>
      <View style={styles.buttonsContainer}>
        <View style={styles.button}>
          {props.userQuestions.length > 0 && (
            <Button
              title="Review Answers"
              color={colors.activeColor}
              onPress={props.onReview}
            />
          )}
        </View>
        <View style={styles.button}>
          <Button
            title="Exit"
            color={colors.activeColor}
            onPress={props.onExit}
          />
        </View>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  screen: {
    flex: 1,
    alignItems: "center",
    justifyContent: "space-around",
  },
  image: {
    resizeMode: "contain",
    margin: 15,
  },
  buttonsContainer: {
    flexDirection: "row",
    justifyContent: "space-around",
    padding: 10,
    width: Dimensions.get("window").width,
  },
  button: {
```

```
    width: Dimensions.get("window").width / 3,
  },
  text: {
    padding: 5,
    fontSize: 17,
    fontFamily: "open-sans-bold",
    color: colors.activeColor,
  },
  successText: {
    padding: 5,
    fontSize: 19,
    fontFamily: "open-sans-bold",
    color: colors.greenish,
  },
  failedText: {
    padding: 5,
    fontSize: 19,
    fontFamily: "open-sans-bold",
    color: colors.primary,
  },
});

export default QuizResult;
```

user/ AuthScreen.js

```
import React, { useState, useCallback, useReducer } from "react";
import {
  ScrollView,
  View,
  KeyboardAvoidingView,
  StyleSheet,
  Button,
  Alert,
  Platform
} from "react-native";
import Card from "../../components/UI/Card";
import Input from "../../components/UI/Input";
import { LinearGradient } from "expo-linear-gradient";
import { useDispatch } from "react-redux";
import * as authActions from "../../store/actions/auth";
import LoadingControl from "../../components/UI/LoadingControl";
import colors from "../../constants/colors";

const FORM_UPDATE = "UPDATE";
```

```
const formReducer = (state, action) => {
  if (action.type === FORM_UPDATE) {
    const updatedInputValues = {
      ...state.inputValues,
      [action.input]: action.value
    };
    const updatedInputValidities = {
      ...state.inputValidities,
      [action.input]: action.isValid
    };
    let updatedFormIsValid = true;
    for (const key in updatedInputValidities) {
      updatedFormIsValid = updatedFormIsValid && updatedInputValidities[key];
    }

    return {
      inputValues: updatedInputValues,
      inputValidities: updatedInputValidities,
      formIsValid: updatedFormIsValid
    };
  } else {
    return state;
  }
};

const AuthScreen = props => {
  const dispatch = useDispatch();
  const [isSignup, setIsSignup] = useState(false);
  const [isLoading, setIsLoading] = useState(false);
  const [formState, dispatchFormState] = useReducer(formReducer, {
    inputValues: {
      email: "",
      password: "",
      confirmPassword: ""
    },
    inputValidities: {
      email: false,
      password: false,
      confirmPassword: !isSignup
    },
    formIsValid: false
  });

  const inputChangeHandler = useCallback(
    (inputIdentifier, inputValue, inputIsValid) => {
      dispatchFormState({
        type: FORM_UPDATE,
        value: inputValue,
```

```
      isValid: inputIsValid,
      input: inputIdentifier
    });
  },
  [dispatchFormState]
);

const authHandler = useCallback(async () => {
  if (!formState.formIsValid) {
    Alert.alert(
      "Validation Error",
      "Please enter valid authentication information!",
      [{ text: "OK" }]
    );
    return;
  }
  try {
    setIsLoading(true);
    if (isSignup) {
      await dispatch(
        authActions.signup(
          formState.inputValues.email,
          formState.inputValues.password
        )
      );
    } else {
      await dispatch(
        authActions.login(
          formState.inputValues.email,
          formState.inputValues.password
        )
      );
    }
  } catch (err) {
    setIsLoading(false);
    Alert.alert(`Error ${isSignup ? "sing up" : "login"}`, `${err}`, [
      { text: "OK" }
    ]);
  }
}, [dispatch, isSignup, formState]);

// if (isLoading) {
//   return <LoadingControl />;
// }
let passwordInput = null;

return (
```

```
    <LinearGradient
      style={styles.screen}
      //colors={["#D9C9D0", "#D4C7CD", "#EAD1DC"]}
      colors={[colors.backColor, colors.backColor, colors.backColor]}
    >
      <KeyboardAvoidingView
        style={styles.container}
        behavior={Platform.OS == "ios" ? "padding" : "height"}
        keyboardVerticalOffset={50}
      >
        <Card style={styles.authContainer}>
          <ScrollView>
            <Input
              id="email"
              onSubmitEditing={() => {
                passwordInput.focus();
              }}
              autoCapitalize="none"
              autoCorrect={false}
              validationText="Please enter valid email"
              label="E-Mail"
              onInputChange={inputChangeHandler}
              required={true}
              email
            />
            <Input
              id="password"
              childRef={ref => {
                passwordInput = ref;
              }}
              onSubmitEditing={() => { }}
              secureTextEntry={true}
              autoCapitalize="none"
              autoCorrect={false}
              validationText="Please enter valid password"
              label="Password"
              onInputChange={inputChangeHandler}
              required={true}
              minLength={5}
            />
            {isSignup && (
              <Input
                id="confirmPassword"
                childRef={ref => {
                  passwordInput = ref;
                }}
                onSubmitEditing={() => { }}
                secureTextEntry={true}
```

```
            autoCapitalize="none"
            autoCorrect={false}
            validationText="Please enter valid password"
            label="Confirm Password"
            onInputChange={inputChangeHandler}
            required={true}
            minLength={5}
            equal={formState.inputValues.password}
          />
        )}
        <View style={styles.buttonContainer}>
          {isLoading ? (
            <LoadingControl />
          ) : (
            <Button
              title={isSignup ? "Sign Up" : "Login"}
              color={colors.primary}
              onPress={authHandler}
            />
          )}
        </View>
        <View style={styles.buttonContainer}>
          <Button
            title={`Switch to ${isSignup ? "Login" : "Sign Up"}`}
            color={colors.accent}
            onPress={() => {
              setIsSignup(prevState => !prevState);
              if (!isSignup) {
                dispatchFormState({
                  type: FORM_UPDATE,
                  value: "",
                  isValid: false,
                  input: "confirmPassword"
                });
              } else {
                dispatchFormState({
                  type: FORM_UPDATE,
                  value: "",
                  isValid: true,
                  input: "confirmPassword"
                });
              }
            }}
          />
        </View>
      </ScrollView>
    </Card>
  </KeyboardAvoidingView>
```

```
    </LinearGradient>

  );
};

AuthScreen.navigationOptions = {
  headerTitle: "Authenticate"
};

const styles = StyleSheet.create({
  screen: {
    flex: 1
  },
  container: {
    width: "100%",
    height: "100%",
    justifyContent: "center",
    alignItems: "center"
  },
  authContainer: {
    width: "80%",
    maxWidth: 400,
    maxHeight: 400
  },
  buttonContainer: {
    marginTop: 10
  }
});

export default AuthScreen;
```

user/ ResultList.js

```
import React, { useState } from "react";

import { View, FlatList, StyleSheet, Alert } from "react-native";

import QuizItem from "../../components/quiz/QuizItem";
import FloatingPlusButton from "../../components/UI/FloatingPlusButton";
import colors from "../../constants/colors";
import * as printService from "../../service/printingService";

const ResultList = (props) => {
  const [showPrint, setShowPrint] = useState(true);
  const printQuizHandler = () => {
    setShowPrint(false);
    printService
```

```
      .printUserScores(props.passedQuizzes)
      .then(() => {
        setShowPrint(true);
      })
      .catch((err) => {
        setShowPrint(true);
        Alert.alert("Error Printing", err);
      });
  };
  return (
    <View style={styles.list}>
      <FlatList
        data={props.passedQuizzes}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
          <QuizItem
            quiz={item}
            showTimeLimit={false}
            isTaken={true}
            onViewResults={props.onViewResults}
          />
        )}
      />
      {showPrint && (
        <FloatingPlusButton
          onPress={printQuizHandler}
          iconName="ios-print"
          color={colors.accent}
        />
      )}
    </View>
  );
};

const styles = StyleSheet.create({
  list: {
    flex: 1,
    width: "100%",
  },
});

export default ResultList;
```

user/ Results.js

```
import React, { useCallback, useEffect, useState } from "react";
```

```
import { useSelector, useDispatch } from "react-redux";
import { View, Text, StyleSheet, Button } from "react-native";
import ResultList from "../user/ResultList";
import Quiz from "../quiz/Quiz";
import { QUIZ_STATUS } from "../../service/quizService";
import LoadingControl from "../../components/UI/LoadingControl";
import * as quizActions from "../../store/actions/quiz";

export default ResultsScreen = (props) => {
  const [quiz, setQuiz] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState();
  const passedQuizzes = useSelector((state) => state.quiz.passedQuizzes).sort(
    (q1, q2) => {
      if (q1.date < q2.date) return 1;
      if (q1.date > q2.date) return -1;
      return 0;
    }
  );
  const dispatch = useDispatch();
  const loadQuizzes = useCallback(async () => {
    setError(null);
    try {
      await dispatch(quizActions.getQuizzes());
    } catch (err) {
      setError(err);
    }
  }, [dispatch, setIsLoading]);

  useEffect(() => {
    setIsLoading(true);
    loadQuizzes().then(() => {
      setIsLoading(false);
    });
  }, [loadQuizzes]);

  if (isLoading) {
    return <LoadingControl />;
  }

  if (error) {
    return (
      <View style={styles.centered}>
        <Text>{error.message}</Text>
        <Button
          title="Try again"
          onPress={loadQuizzes}
          color={colors.primary}
```

```
        />
      </View>
    );
  }

  const reviewQuizHandler = (quiz) => {
    setQuiz(quiz);
  };

  const goBackFromQuizDetailsHandler = () => {
    setQuiz(null);
  };

  let content = (
    <ResultList
      onViewResults={reviewQuizHandler}
      passedQuizzes={passedQuizzes}
    />
  );
  if (quiz) {
    content = (
      <Quiz
        requestedState={QUIZ_STATUS.FINISHED}
        quiz={quiz}
        onGoBack={goBackFromQuizDetailsHandler}
      />
    );
  }
  return <View style={styles.container}>{content}</View>;
};
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#fff",
    alignItems: "center",
    justifyContent: "center",
  },
});
```

7. Общите компоненти, които се използват за изграждане на екраните на приложението се намират в директория components.

quiz/question/ OpenQuestion.js

```
import React, { useState, useEffect } from "react";
import { View, Text, StyleSheet, TextInput, Dimensions } from "react-native";
```

```
import colors from "../../../constants/colors";

const OpenQuestion = (props) => {
  const [answer, setAnswer] = useState(null);
  const currQuestion = props.question;

  useEffect(() => {
    setAnswer(props.selectedAnswer);
  }, [props.selectedAnswer]);

  const changeTextHandler = (text) => {
    if (props.reviewMode) {
      return;
    }
    setAnswer(text);
  };

  let frameStyle = null;
  if (!props.reviewMode) {
    frameStyle = { borderWidth: 1, borderColor: "black" };
  } else {
    if (currQuestion.correctAnswer === (answer ?? props.selectedAnswer)) {
      frameStyle = styles.rightAnswer;
    } else {
      frameStyle = styles.wrongAnswer;
    }
  }

  return (
    <View style={styles.container}>
      <TextInput
        placeholder={
          props.createMode ? "Enter correct answer" : "Enter your answer"
        }
        style={{
          ...styles.textInput,
          backgroundColor: props.reviewMode ? "#ccc" : "white",
          ...frameStyle,
        }}
        editable={!props.reviewMode}
        autoCapitalize="none"
        multiline={true}
        onChangeText={changeTextHandler}
        onBlur={() => props.onSelectAnswer(answer)}
        value={answer ?? props.selectedAnswer}
      />
      {props.reviewMode && currQuestion.correctAnswer !== answer && (
        <Text style={styles.correctAnswerText}>
```

```
          Correct Answer: {currQuestion.correctAnswer}
        </Text>
      )}
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    alignItems: "center",
  },
  textInput: {
    width: 0.9 * Dimensions.get("window").width,
    paddingHorizontal: 7,
    paddingVertical: 3,
    marginVertical: 10,
    borderRadius: 10,
    fontSize: 16,
    fontFamily: "open-sans",
  },
  rightAnswer: {
    borderWidth: 3,
    borderColor: colors.greenish,
  },
  wrongAnswer: {
    borderWidth: 3,
    borderColor: colors.primary,
  },
  correctAnswerText: {
    fontFamily: "open-sans",
    fontSize: 16,
    color: colors.greenish,
    paddingBottom: 10,
  },
});

export default OpenQuestion;
```

quiz/question/ Question.js

```
import React, { useState, useEffect } from "react";
import { View, Text, Button, StyleSheet, Dimensions } from "react-native";
import colors from "../../../constants/colors";
import * as componentUtils from "../../utils";

const Question = (props) => {
  const [selectedAnswer, setSelectedAnswer] = useState(null);
```

```
const currQuestion = props.question;
const userAnswer = currQuestion.userAnswer;
const QuestionBodyComponent = componentUtils.getQuestionComponentByQuestionType(
  currQuestion.type
);

useEffect(() => {
  setSelectedAnswer(userAnswer);
}, [props, userAnswer]);

const selectAnswer = (ans) => {
  if (props.reviewMode) {
    return;
  }
  setSelectedAnswer(ans);
};

const handleAction = (action) => {
  const ans = selectedAnswer;
  setSelectedAnswer(null);
  action(ans);
};

const actionButton = (
  <View style={styles.actionButton}>
    {!props.isLast && (
      <View style={styles.button}>
        <Button
          color={colors.activeColor}
          title=">>"
          onPress={() => {
            handleAction(props.onNext);
          }}
        />
      </View>
    )}
    {props.isLast && !props.reviewMode && (
      <View style={styles.button}>
        <Button
          color={colors.activeColor}
          title="SUBMIT"
          onPress={() => {
            handleAction(props.onSubmit);
          }}
        />
      </View>
    )}
    {props.reviewMode && (
```

```
        <View style={styles.button}>
          <Button
            color={colors.activeColor}
            title="EXIT"
            onPress={props.onExit}
          />
        </View>
      )}
      {!props.isFirst && (
        <View style={styles.button}>
          <Button
            color={colors.activeColor}
            title="<<"
            onPress={() => {
              handleAction(props.onPrevious);
            }}
          />
        </View>
      )}
    </View>
  );

  return (
    <View style={styles.container}>
      <Text style={styles.question}>
        {currQuestion.number}.{currQuestion.question}
      </Text>
      <QuestionBodyComponent
        {...props}
        onSelectAnswer={selectAnswer}
        selectedAnswer={selectedAnswer ?? userAnswer}
      />
      {props.reviewMode && (
        <Text style={styles.pointsText}>Points : {currQuestion.points}</Text>
      )}
      {actionButton}
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    margin: 10,
    padding: 10,
    flex: 1,
    width: "100%",
    height: "100%",
    alignItems: "center",
```

```
    justifyContent: "space-between",
    backgroundColor: colors.backColor,
  },
  question: {
    fontFamily: "open-sans-bold",
    fontSize: 20,
    fontStyle: "italic",
    color: colors.activeColor,
  },
  answers: {
    flex: 1,
    alignItems: "stretch",
    width: "100%",
    paddingVertical: 30,
  },
  answer: {
    fontFamily: "open-sans",
    fontSize: 16,
    color: colors.activeColor,
  },
  answerCard: {
    margin: 10,
    borderRadius: 10,
    overflow: "hidden",
  },
  actionButton: {
    flex: 1,
    flexDirection: "row-reverse",
    padding: 20,
    justifyContent: "space-between",
    width: Dimensions.get("window").width,
    alignItems: "flex-end",
    color: colors.activeColor,
  },
  button: {
    width: Dimensions.get("window").width / 4,
  },
  rightAnswer: {
    borderWidth: 3,
    borderColor: colors.greenish,
  },
  wrongAnswer: {
    borderWidth: 3,
    borderColor: colors.primary,
  },
  pointsText: {
    fontFamily: "open-sans-bold",
    fontSize: 16,
```

```
    color: colors.activeColor,
  },
});
export default Question;
```

quiz/question/ SingleChoiceQuestion.js

```
import React from "react";
import {
  View,
  Text,
  TouchableNativeFeedback,
  TouchableOpacity,
  Platform,
  StyleSheet,
} from "react-native";
import colors from "../../../constants/colors";
import Card from "../../../components/UI/Card";

const SingleChoiceQuestion = (props) => {
  const selectedAnswer = props.selectedAnswer;
  const currQuestion = props.question;
  const Touchable =
    Platform.OS === "android" ? TouchableNativeFeedback : TouchableOpacity;
  if (!currQuestion) return null;

  return (
    <View style={styles.answers}>
      {Object.keys(currQuestion.possibleAnswers).map((val) => {
        let frameStyle = null;

        if (currQuestion.possibleAnswers[val].truthy && props.reviewMode) {
          frameStyle = styles.rightAnswer;
        }

        if (
          selectedAnswer === val &&
          !currQuestion.possibleAnswers[val].truthy &&
          props.reviewMode
        ) {
          frameStyle = styles.wrongAnswer;
        }

        return (
          <Touchable
            key={val}
            useForeground={true}
```

```
            onPress={() => {
              props.onSelectAnswer(val);
            }}
            onLongPress={() => {
              if (props.onLongPress) {
                props.onLongPress(val);
              }
            }}
          >
            <View style={styles.answerCard}>
              <Card
                style={{
                  ...{
                    backgroundColor: selectedAnswer === val ? "#ccc" : "#fff",
                  },
                  ...frameStyle,
                }}
              >
                <Text style={styles.answer} key={val}>
                  {val}.{currQuestion.possibleAnswers[val].answer}
                </Text>
              </Card>
            </View>
          </Touchable>
        );
      })}
    </View>
  );
};

const styles = StyleSheet.create({
  answers: {
    //flex: 1,
    alignItems: "stretch",
    width: "100%",
    paddingVertical: 30,
  },
  answer: {
    fontFamily: "open-sans",
    fontSize: 16,
    color: colors.activeColor,
  },
  answerCard: {
    margin: 10,
    borderRadius: 10,
    overflow: "hidden",
  },
  rightAnswer: {
```

```
    borderWidth: 3,
    borderColor: colors.greenish,
  },
  wrongAnswer: {
    borderWidth: 3,
    borderColor: colors.primary,
  },
});
export default SingleChoiceQuestion;
```

quiz/ QuizItem.js

```
import React, { useState } from "react";
import { Ionicons } from "@expo/vector-icons";
import {
  View,
  StyleSheet,
  Text,
  TouchableOpacity,
  TouchableNativeFeedback,
  Platform,
  Button,
  Image,
  Dimensions,
} from "react-native";
import Card from "../UI/Card";
import colors from "../../constants/colors";

const Touchable =
  Platform.OS === "android" ? TouchableNativeFeedback : TouchableOpacity;

const QuizItem = (props) => {
  const [isExpanded, setIsExpanded] = useState(false);
  const timeMessage = props.quiz.timeLimit
    ? (props.quiz.timeLimit.min ? `${props.quiz.timeLimit.min} min.` : "") +
      (props.quiz.timeLimit.sec ? `${props.quiz.timeLimit.sec} sec.` : "")
    : "";

  const passedIcon = props.isTaken ? (
    props.quiz.passed ? (
      <View style={styles.mark}>
        <Ionicons
          name={"md-checkmark-circle-outline"}
          color={colors.greenish}
          size={23}
        />
      </View>
```

```
  ) : (
    <View style={styles.mark}>
      <Ionicons
        name={"md-close-circle-outline"}
        color={colors.primary}
        size={23}
      />
    </View>
  )
) : null;
return (
  <View style={styles.container}>
    <Card style={styles.quizItem}>
      <View style={styles.sampleInfo}>
        <View style={styles.quizTitle}>
          <Text style={styles.title}>{props.quiz.title}</Text>

          {props.quiz.timeLimit && props.showTimeLimit && (
            <View style={styles.timeInfo}>
              <Ionicons
                style={{ paddingRight: 10 }}
                name={"md-stopwatch"}
                color={colors.primary}
                size={20}
              />
              <Text style={styles.dateField}>
                {`You will have ${timeMessage} to pass it.`}
              </Text>
            </View>
          )}
          <Text style={styles.dateField}>
            {props.quiz.date.toDateString()}
          </Text>
        </View>
        {passedIcon}
        {!isExpanded && (
          <Touchable onPress={() => setIsExpanded(true)}>
            <View style={styles.arrowDown}>
              <Ionicons
                name={"ios-arrow-down"}
                color={colors.primary}
                size={23}
              />
            </View>
          </Touchable>
        )}
      </View>
      <View>
```

```
          {isExpanded && (
            <View style={styles.details}>
              <View style={{ padding: 5 }}>
                <Image
                  style={{ width: 150, height: 150 }}
                  source={{ uri: props.quiz.imageUrl }}
                />
              </View>
              <Text style={styles.quizDescription}>
                {props.quiz.description}
              </Text>
              <View style={styles.buttonContainer}>
                {props.onTakeQuiz ? (
                  <View style={styles.button}>
                    <Button
                      style={styles.button}
                      color={colors.activeColor}
                      title={props.isTaken ? "Retry the quiz" : "Take the test"}
                      onPress={() => props.onTakeQuiz(props.quiz)}
                    />
                  </View>
                ) : null}
                {props.isTaken && props.onViewResults ? (
                  <View style={styles.button}>
                    <Button
                      color={colors.activeColor}
                      title="Your Result"
                      onPress={() => props.onViewResults(props.quiz)}
                    />
                  </View>
                ) : null}
              </View>
              <Touchable onPress={() => setIsExpanded(false)}>
                <View style={styles.arrowUp}>
                  <Ionicons
                    name={"ios-arrow-up"}
                    color={colors.primary}
                    size={23}
                  />
                </View>
              </Touchable>
            </View>
          )}
        </View>
      </Card>
    </View>
  );
};
```

```
const styles = StyleSheet.create({
  container: {
    margin: 5,
  },
  quizItem: {
    flex: 1,
    padding: 15,
    backgroundColor: colors.backColor,
  },
  quizTitle: {
    flex: 15,
    borderWidth: 0,
    borderColor: "red",
  },
  mark: {
    flex: 2,
    alignItems: "flex-end",
    justifyContent: "center",
  },
  arrowDown: {
    flex: 3,
    alignItems: "flex-end",
    justifyContent: "center",
    width: "100%",
  },
  arrowUp: {
    alignItems: "center",
    justifyContent: "center",
    paddingVertical: 5,
    width: "100%",
  },

  title: {
    fontSize: 17,

    color: colors.activeColor,
    fontFamily: "open-sans-bold",
  },
  quizDescription: {
    fontSize: 15,
    fontStyle: "italic",
    color: colors.activeColor,
    fontFamily: "open-sans",
    padding: 10,
  },
  dateField: {
    color: "#ccc",
```

```
    fontFamily: "open-sans",
  },
  timeInfo: {
    flexDirection: "row",
  },
  details: {
    alignItems: "center",
  },
  sampleInfo: {
    flexDirection: "row",
    justifyContent: "space-between",
  },
  buttonContainer: {
    flexDirection: "row",
    justifyContent: "center",
    width: "100%",
  },
  button: {
    width: Dimensions.get("window").width / 3,
    marginHorizontal: 20,
  },
});

export default QuizItem;
```

UI/ Card.js

```
import React from "react";
import { View, StyleSheet } from "react-native";

const Card = props => {
  return (
    <View style={{ ...styles.card, ...props.style }}>{props.children}</View>
  );
};

const styles = StyleSheet.create({
  card: {
    shadowColor: "black",
    shadowOpacity: 0.26,
    shadowRadius: 6,
    shadowOffset: {
      width: 0,
      height: 2
    },
    elevation: 6,
    backgroundColor: "white",
```

```
    padding: 20,
    borderRadius: 10
  }
});

export default Card;
```

UI/ FloatingPlusButton.js

```
import React from "react";
import {
  View,
  StyleSheet,
  TouchableNativeFeedback,
  Dimensions,
} from "react-native";
import colors from "../../constants/colors";
import { Ionicons } from "@expo/vector-icons";
const FloatingPlusButton = (props) => {
  return (
    <View
      style={
        props.static
          ? {
              ...styles.containerStatic,
              backgroundColor: props.color ?? colors.primary,
            }
          : {
              ...styles.containerFloat,
              backgroundColor: props.color ?? colors.primary,
            }
      }
    >
      <TouchableNativeFeedback
        onPress={() => {
          if (props.onPress) {
            props.onPress();
          }
        }}
      >
        <Ionicons name={props.iconName} size={35} color="white" />
      </TouchableNativeFeedback>
    </View>
  );
};

const styles = StyleSheet.create({
```

```
  containerFloat: {
    position: "absolute",
    width: 50,
    height: 50,
    borderRadius: 25,
    alignItems: "center",
    justifyContent: "center",
    right: Dimensions.get("window").width * 0.02,
    bottom: Dimensions.get("window").height * 0.01,
    overflow: "hidden",
    backgroundColor: colors.primary,
  },
  containerStatic: {
    width: 50,
    height: 50,
    borderRadius: 25,
    alignItems: "center",
    justifyContent: "center",
    backgroundColor: colors.primary,
  },
});

export default FloatingPlusButton;
```

UI/ HeaderButton.js

```
import React from "react";
import { Platform, Text, View, StyleSheet } from "react-native";
import { HeaderButton } from "react-navigation-header-buttons";
import { Ionicons } from "@expo/vector-icons";
import colors from "../../constants/colors";

const CustomHeaderButton = props => {
  return (
    <View style={styles.buttonContainer}>
      <HeaderButton
        {...props}
        IconComponent={Ionicons}
        iconSize={25}
        color={
          Platform.OS === "android" ? colors.inactiveColor : colors.primary
        }
      />
      <Text style={styles.title}>{props.title}</Text>
    </View>
  );
};
```

```
const styles = StyleSheet.create({
  buttonContainer: {
    padding: 10
  },
  title: {
    fontFamily: "open-sans",
    fontSize: 10,
    color: Platform.OS === "android" ? colors.activeColor : colors.primary
  }
});
export default CustomHeaderButton;
```

UI/ Input.js

```
import React, { useReducer, useEffect, useRef } from "react";
import { View, TextInput, Text, StyleSheet } from "react-native";

const INPUT_CHANGE = "INPUT_CHANGE";
const INPUT_BLUR = "INPUT_BLUR";
const RESET = "RESET";
const inputReducer = (state, action) => {
  switch (action.type) {
    case INPUT_CHANGE:
      return {
        ...state,
        value: action.value,
        isValid: action.isValid,
        touched: true,
      };
    case INPUT_BLUR:
      return {
        ...state,
        touched: true,
      };
    case RESET:
      return {
        value: action.value,
        isValid: action.isValid,
        touched: false,
      };
    default:
      return state;
  }
};
const isValidURL = (str) => {
  if (!str) {
```

```
    return true;
  }
  var pattern = new RegExp(
    "^(https?:\\/\\/)?" + // protocol
    "((([a-z\\d]([a-z\\d-]*[a-z\\d])*)\\.)+[a-z]{2,}|" + // domain name
    "((\\d{1,3}\\.){3}\\d{1,3}))" + // OR ip (v4) address
    "(\\:\\d+)?(\\/[-a-z\\d%_.~+]*)*" + // port and path
    "(\\?[;&a-z\\d%_.~+=-]*)?" + // query string
      "(\\#[-a-z\\d_]*)?$",
    "i"
  ); // fragment locator
  return !!pattern.test(str);
};

const Input = (props) => {
  const { onInputChange } = props;
  const inputRef = useRef(null);

  const [state, dispatch] = useReducer(inputReducer, {
    value: props.initialValue ? props.initialValue : "",
    isValid: !!props.initiallyValid,
    touched: false,
  });

  useEffect(() => {
    if (!props.getRef) {
      return;
    }
    inputRef.current = {
      state: state,
      resetInput: resetInput,
    };
    props.getRef(inputRef);
  }, [state]);

  useEffect(() => {
    if (!props.initialValue) {
      return;
    }
    if (props.initialValue === state.value) {
      return;
    }
    resetInput();
  }, [props.initialValue]);

  useEffect(() => {
    if (state.touched) {
      props.onInputChange(props.id, state.value, state.isValid);
```

```
    }
  }, [state, onInputChange]);

  const lostFocusHandler = () => {
    dispatch({
      type: INPUT_BLUR,
    });
  };

  const resetInput = () => {
    dispatch({
      type: RESET,
      value: props.initialValue ? props.initialValue : "",
      isValid: !!props.initiallyValid,
    });
  };
  const textChangeHandler = (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;
    if (props.required && text.trim().length === 0) {
      isValid = false;
    }
    if (props.email && !emailRegex.test(text.toLowerCase())) {
      isValid = false;
    }
    if (props.url && !isValidURL(text)) {
      isValid = false;
    }
    if (props.min != null && +text < props.min) {
      isValid = false;
    }
    if (props.max != null && (+text > props.max || !+text)) {
      isValid = false;
    }
    if (props.minLength != null && text.length < props.minLength) {
      isValid = false;
    }
    if (props.equal != null && text !== props.equal) {
      isValid = false;
    }
    dispatch({
      type: INPUT_CHANGE,
      value: text,
      isValid: isValid,
    });
  };
```

```
  return (
    <View style={styles.inputControl}>
      <View
        style={{
          flexDirection: props.positionHorizontal ? "row" : "column",
          alignItems: props.positionHorizontal ? "center" : "stretch",
        }}
      >
        <Text
          style={{
            ...styles.label,
            marginVertical: props.positionHorizontal ? 0 : 5,
            marginRight: props.positionHorizontal ? 8 : 0,
          }}
        >
          {props.label}
        </Text>
        <TextInput
          {...props}
          style={styles.input}
          value={state.value}
          onChangeText={textChangeHandler}
          onBlur={lostFocusHandler}
          ref={props.childRef}
        />
      </View>
      {!state.isValid && state.touched && (
        <Text style={styles.validationText}>{props.validationText}</Text>
      )}
    </View>
  );
};

const styles = StyleSheet.create({
  inputControl: {
    width: "100%",
  },
  label: {
    fontFamily: "open-sans-bold",
    marginVertical: 8,
  },
  input: {
    paddingHorizontal: 2,
    paddingVertical: 5,
    borderBottomColor: "#ccc",
    borderBottomWidth: 1,
  },
```

```
  validationText: {
    fontFamily: "open-sans",
    fontSize: 10,
    fontStyle: "italic",
    color: "red",
  },
});

export default Input;
```

UI/ LoadingControl.js

```
import React from "react";
import { View, ActivityIndicator, StyleSheet } from "react-native";
import Color from "../../constants/colors";

const LoadingControl = props => {
  return (
    <View style={{ ...styles.centered, ...props.style }}>
      <ActivityIndicator size="large" color={Color.primary} />
    </View>
  );
};

const styles = StyleSheet.create({
  centered: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center"
  }
});

export default LoadingControl;
```

UI/ LogoutButton.js

```
import React from "react";
import { Alert } from "react-native";
import { useDispatch } from "react-redux";
import * as authActions from "../../store/actions/auth";
import HeaderButton from "./HeaderButton";
import { HeaderButtons, Item } from "react-navigation-header-buttons";

const LogoutButton = props => {
  const dispatch = useDispatch();

  return (
```

```
    <HeaderButtons HeaderButtonComponent={HeaderButton} left={true}>
      <Item
        title="Log Out"
        iconName={Platform.OS === "android" ? "md-exit" : "ios-exit"}
        onPress={() => {
          Alert.alert("Log out", "Do you want to log out?", [
            {
              text: "Yes",
              onPress: () => {
                dispatch(authActions.logout());
              }
            },
            {
              text: "No"
            }
          ]);
        }}
      />
    </HeaderButtons>
  );
};

export default LogoutButton;
```

UI/ Timer.js

```
import React, { useEffect, useState } from "react";
import { View, Text, StyleSheet } from "react-native";

const formatNumber = (number) => `0${number}`.slice(-2);
const getRemaining = (time) => {
  console.log(time);
  const mins = Math.floor(time / 60);
  const secs = time - mins * 60;
  return { mins: formatNumber(mins), secs: formatNumber(secs) };
};

const Timer = (props) => {
  const [remainingSecs, setRemainingSecs] = useState(props.timeInSeconds);
  const { mins, secs } = getRemaining(remainingSecs);

  useEffect(() => {
    const interval = setInterval(() => {
      const seconds = Math.floor((Date.now() - props.startingTime) / 1000);
      let secondsLeft = props.timeInSeconds - seconds;
      if (secondsLeft <= 0) {
        props.onTimeout();
```

```
        clearInterval(interval);
      } else {
        setRemainingSecs(secondsLeft);
      }
    }, 1000);
    return () => {
      clearInterval(interval);
    };
  }, [remainingSecs]);

  return (
    <View>
      <Text>{`${mins}:${secs}`}</Text>
    </View>
  );
};

const styles = StyleSheet.create({});

export default Timer;
```