Пловдивски университет „Паисий Хилендарски”
Факултет по математика и информатика

Курсова работа

По дисциплина “Мобилни приложения”

На тема: „Списък за пазаруване“

Изготвили:
Тони Бекирски 1801321001
Петър Маламов 1801321031

Специалност:
Софтуерно Инженерство
3-ти курс, редовно

Проверил:
доц. д-р Станка Хаджиколева

Подпис:
……

Пловдив, 2020

Съдържание

Име: …… Факултетен номер: ……

1. Увод

Целта на курсовия проект е да се изгради мобилно приложение показващо списъка за пазаруване , като продуктите са разделени в 2 списъка (такива които са купени и такива които не са)

Дизайн на приложението - Тони Бекирски

Връзки и модификации на базата – Петър Маламов

2. Основни функционалности

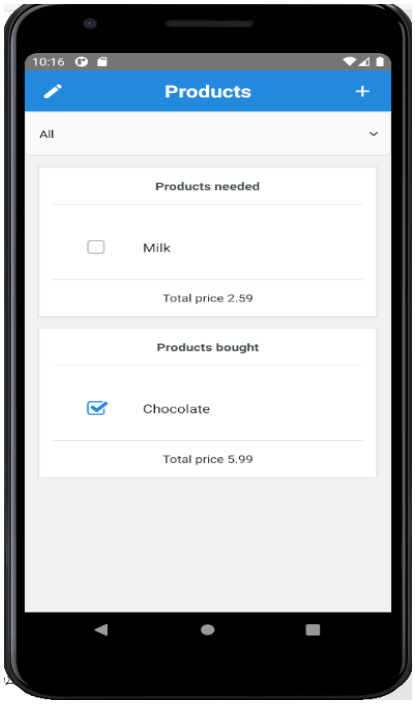
- Показване на списъка с продукти
- Добавяне на продукти
- Редактиране на продукт
- Премахване на продукт
- Филтриране на продукти
- Пресмятане на обща цена на продуктите

3. Използвани технологии и библиотеки

- React Native CLI
- JavaScript
- Android SDK
- Redux
- React-redux
- Redux-Saga
- React-navigation/native
- React-navigation/stack
- React-native-elements
- React-nativ-vector-icons
- React-native-dropdown-picker
- SQLite – react-native-sqlite-storage

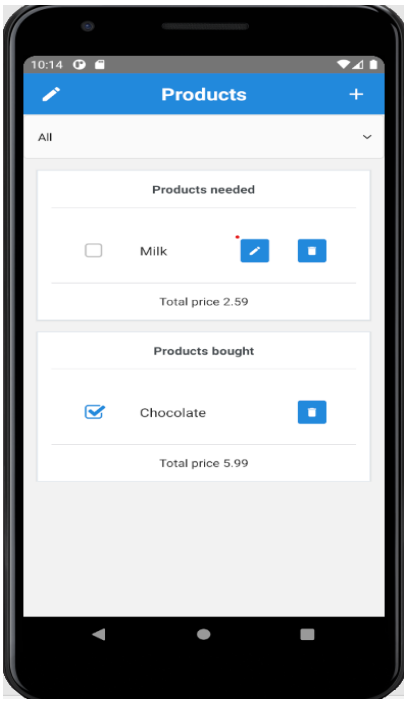
Име: …… Факултетен номер: ……

4. Потребителски инструктаж



Фигура 1

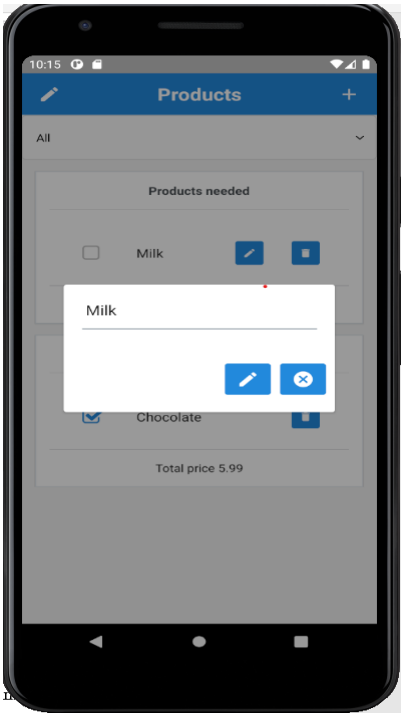
При стартиране, приложението се показват двата списъка с продукти (закупени и такива който трябва да се купят), заглавна част с 2 бутона (за промяна и за добавяне на продукт), филтър за продуктите по категория и поле с отметка . При натискане а отметката продукта сменя състоянието си в обратното състояние (‚закупен‘ > ‚трябва да се купи‘ или ‚трябва да се купи‘ > ‚закупен‘) . При селектиране на стойност от листа под заглавната част с филтрират продуктите по избраната от потребителя категория.



Фигура 2

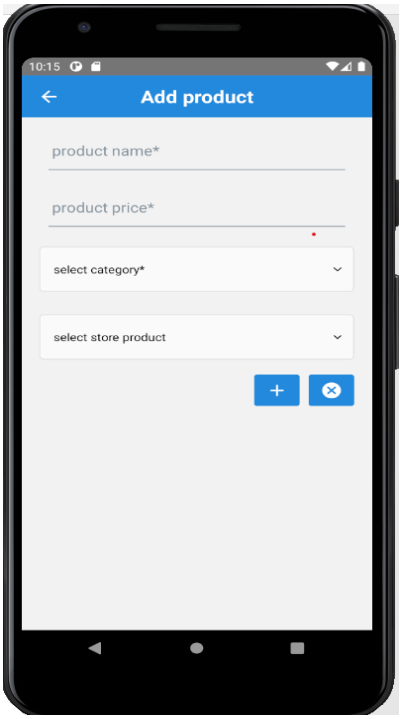
При натискане на бутона за промяна в заглавната част на всеки продукт от списъците се показват допълнителни два бутона – за промяна на името на продукта и за изтриване на продукта . При натискане на бутона за изтриване продуктът се премахва от списъка

Име: …… Факултетен номер: ……



Фигура 3

При натискане на бутона за „промяна“ на продукта излиза изскачащ прозорец . В него може да се смени името на избрания продукт.



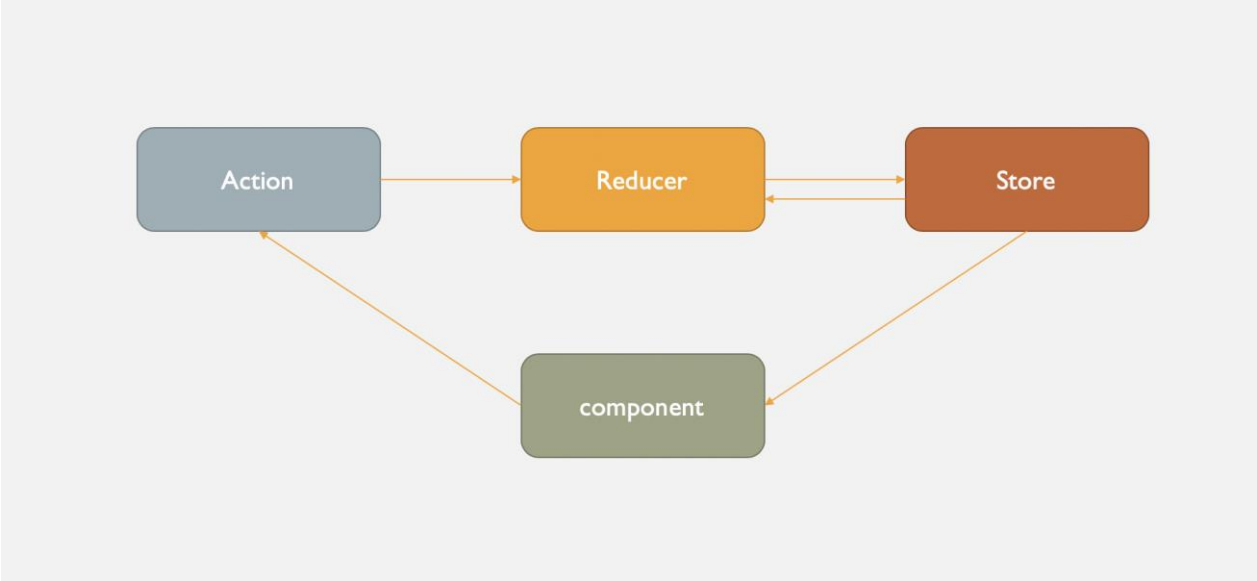
Фигура 4

При натискане на бутона за добавяне в заглавната част се отваря нов екран , в който може да попълним данни за нов продукт и да го добавим или да откажем и да се върнем назад . Полета които съдържат * са задължителни , без тях добавянето няма да се извърши . Има два начина на попълване на полета , първият е ръчно да се попълни всяко поле , а вторият е чрез избиране на вече готов продукт от последния списък . С избирането на продукт от списъка автоматично се попълват останалите полета на екрана с данни от избрания продукт . За успешното добавяне на продукт се разбира след като се покаже изскачащ прозорец в който пише „Product was added”

Име: ……

Факултетен номер: ……

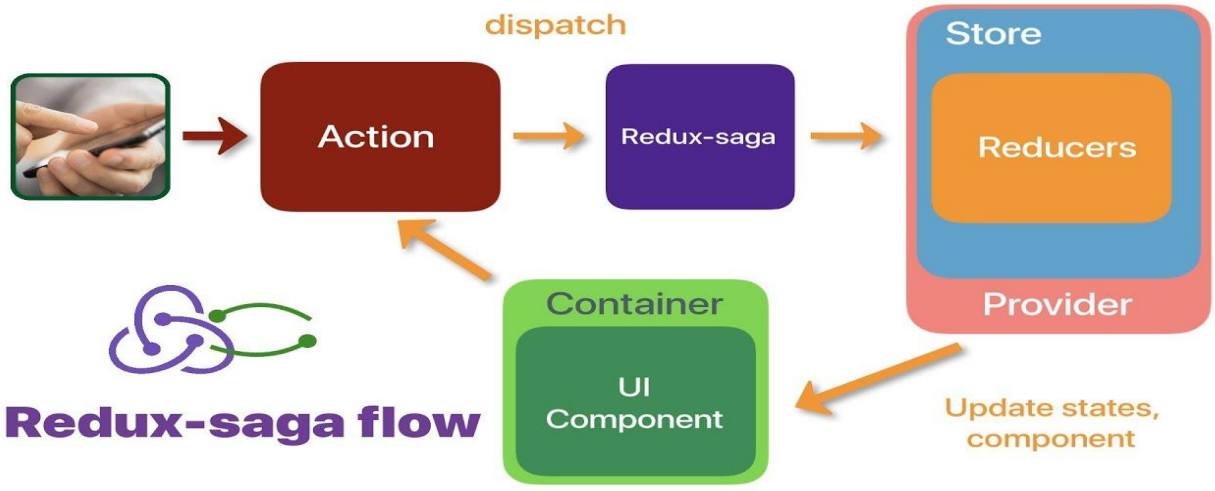
5. Архитектура на приложението



Фигура 5

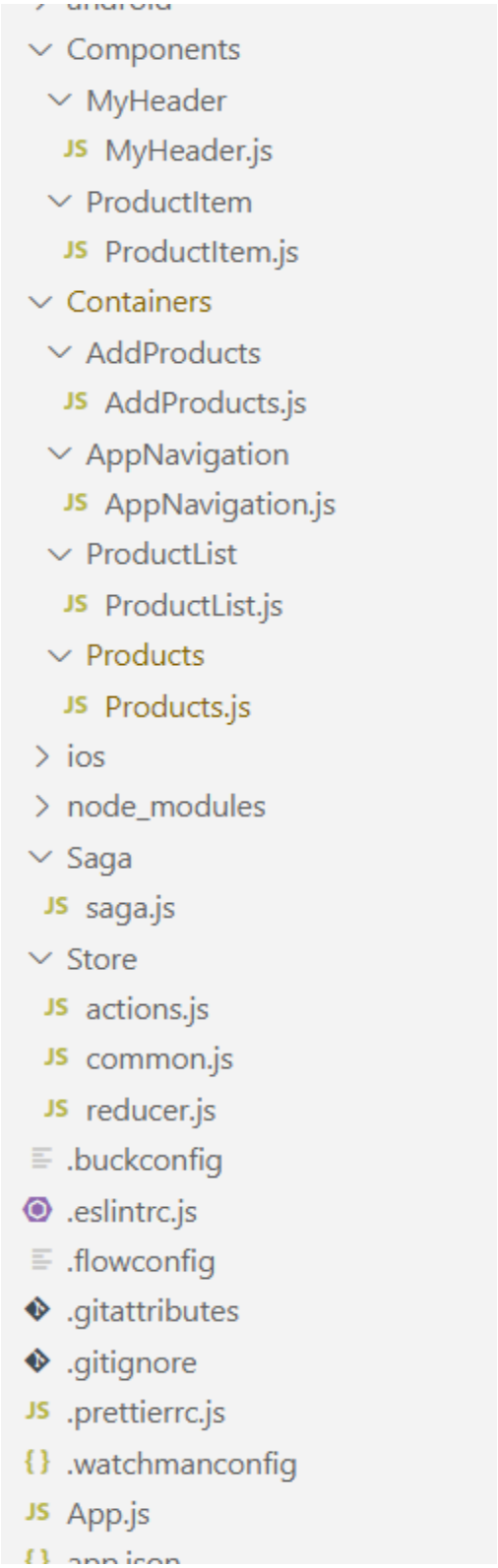
Приложението използва „флекс“ архитектурата , разработена от фейсбук, при нея има единичен поток на данни . При тази архитектура компонентите не могат директно да променят данните в магазин. Начина по който се променят данните е чрез изпращане на действие (action) от компонента . Това действие се прихваща от редюсерът. В зависимост от действието , редюсерът прави различни промени върху магазина . След като магазина вече е променен компонента получава новите данни и ги показва на екрана .

В това приложение обаче променяме малко тази архитектура като между действието и магазина добавяме посредник , в който може да извършваме заявки към АПИ или към база данни.



Фигура 6

Име: …… Факултетен номер: ……



Фигура 7

Име: …… Факултетен номер: ……

- Components – директория която съхранява компоненти, които могат да се ползват многократно
    - MyHeader – заглавната част на екраните
    - ProductItem – структурата на всеки продукт

- Containers – директория която съхранява контейнери, които съдържат в себе си по-малки компоненти или други контейнери.
    - AddProduct – съхранява екрана за добавяне на продукти
    - AppNavigation – съхранява стек навигацията на приложението
    - ProductList – съхранява продукти отнасящи се към конкретния лист (купени или нужни продукти)
    - Products - съхранява началния екрана на приложението
- node_modules – съхранява всички нужни библиотеки необходими на приложението за правилно функциониране.
- Saga – съхранява посредника (middleware) между приложението и базата данни
    - saga.js – в този файл се извършват заявките към базата данни , прихваща действия от други компоненти/ контейнери . След като извлече данни от базата пуска действие съм магазина.
- Store(магазин) – съхранява данните (извлечени от базата) с които работи приложението
    - actions.js – съхранява декларации на всички действия на приложението
    - common.js – съхранява два списъка – категории и продукти от които може да се избира при добавяне.
    - reducer.js – съхранява данните извлечени от базата данни, прихваща действия, пуснати (dispatched) от други компоненти и определя какво да прави
- App.js – в този файл се свързват магазина и посредника с приложението.

6. Имплементация

1. Инициализиране на базата

Когато компонентът Products се зареди се изпраща действие за инициализиране на базата .

```
componentDidMount() {
  this.props.initDB().then(() => this.props.getProducts());
}

handleFilterChange(item) {
```

Фигура 8

```
0
1   const mapDispatchToProps = (dispatch) => {
2     return {
3       getProducts: () => {
4         return dispatch({type: 'GETPRODUCTS'});
5       },
6       initDB: async () => {
7         return await dispatch({type: 'initDB'});
8       },
9     };
0   };
1
```

Фигура 9

Изпратеното действие за инициализиране на базата се прихваща от Saga


```
5   export default function* rootSaga() {
6     yield takeEvery(initDB, initDatabase);
7     yield takeEvery(NEWPRODUCT, addProduct);
8     yield takeEvery(GETPRODUCTS, getProducts);
9     yield takeEvery(UPDATEPRODUCT, updatedProduct);
0     yield takeEvery(DELETEPRODUCT, deleteProduct);
1   }
2
```

Фигура 10

```
41
42  function* initDatabase(action) {
43    const init = initProducts();
44    const products = yield take(init);
45    yield put(update(products));
46  }
47
```

Фигура 11

При действие за инициализиране на базата се проверява дали тази база съществува и ако не тя се създава след което се извличат продуктите от нея (ако има такива) се изпащат на магазина

```
function initProducts() {
  const channel = new eventChannel((emiter) => {
    const listener = database.transaction((transaction) => {
      transaction.executeSql(
        'CREATE TABLE IF NOT EXISTS myProducts(id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(40),category VARCHAR(30),price DECIMAL(8,2), pur
        [],
        (transaction, result) => {
          txn.executeSql('SELECT * FROM myProducts', [], function (tx, res) {
            emiter({data: res.rows.raw() || {}});
          });
        },
      );
    });
    return () => {
      listener.off();
    };
  });
  return channel;
}
```

Фигура 12

Редюсерът прихваща това действие и променя магазина с новите данни

```
};
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case UPDATEPRODUCTS:
      return {
        ...state,
        myProducts: action.payload.products,
        fetchingData: false,
      };
    case GETPRODUCTS:
      return {...state, fetchingData: true};
    case ADDINGPRODUCT:
      return {...state, addingProduct: action.value};
    default:
      return state;
  }
};

export default reducer;
```

Фигура 13

Име: …… Факултетен номер: ……

2. Извличане на данни от базата

   След като действието за инициализиране на базата е приключило се изпраща действие за взимане на продуктите от базата

```
componentDidMount() {
  this.props.initDB().then(() => this.props.getProducts());
}

handleFilterChange(item) {
```

Фигура 14

Това действие се прихваща в Saga и в Reducer-а

```
yield takeEvery(GETPRODUCTS, getProducts);
```

SAGA

Фигура 15

```
    };
  case GETPRODUCTS:
    return {...state, fetchingData: true};
  case ADDINGPRODUCT:
```

Reducer

Фигура 16

В редюсера това действа пуска зареждача

В сага-та това действие извлича данните от базата и изпраща ново действие към редюсера

Име: …… Факултетен номер: ……

```
73
74  function getProductsEventChannel() {
75    const channel = new eventChannel((emiter) => {
76      const listener = database.transaction(function (txn) {
77        txn.executeSql('SELECT * FROM myProducts', [], function (tx, res) {
78          emiter({data: res.rows.raw() || {}});
79        });
80      });
81      return () => {
82        listener.off();
83      };
84    });
85    return channel;
86  }
87
88  function* getProducts() {
89    const updateChannel = getProductsEventChannel();
90    const item = yield take(updateChannel);
91    yield put(update(item));
92  }
93
```

Фигура 17

Новото действие прихванато в редюсера записва данните в магазина и спира зареждачката


```
case UPDATEPRODUCTS:
  return {
    ...state,
    myProducts: action.payload.products,
    fetchingData: false,
  };
```

Фигура 18

3. Визуализация
   Продуктите се взимат от магазина

```
106
107  const mapStateToProps = (state) => {
108    return {
109      myProducts: state.myProducts,
110      fetchingData: state.fetchingData,
111      categories: [
112        {
113          label: 'All',
114          value: 'All',
115        },
116        ...state.categories,
117      ],
118    };
119  };
120
```

Фигура 19

След това минават през филтър по категория


```
render() {
  const filterdPoducts = this.getProductByCategory(
    this.props.myProducts,
    this.state.category,
  );
```

Фигура 20

```
4    }
5
6    getProductByCategory(products, category) {
7      if (category !== 'All') {
8        return products.filter((product) => product.category === category);
9      }
0      return products;
1    }
2
```

Фигура 21

След филтъра по категория продуктите се разделят на два листа в зависимост от това дали продукта е купен или не

```
const productNeeded = this.getProductsByPurchased(filterdPoducts, 0);

const productBought = this.getProductsByPurchased(filterdPoducts, 1);
```

Фигура 22

```
getProductsByPurchased(products, needed) {
  return products.filter((product) => product.purchased === needed);
}
```

Фигура 23

Това дали ще се покажат данните се определя от това дали в момента извличаме данни , ако извличаме ще се покаже зареждачката

```
const body = () => {
  return this.props.fetchingData ? (
    <View style={[styles.container]}>
      <ActivityIndicator size="large" color="red" />
    </View>
  ) : (
    <ScrollView>
      <ProductList
        title="Products needed"
        showButtons={this.state.editProduct}
        products={productNeeded}
      />
      <ProductList
        title="Products bought"
        showButtons={this.state.editProduct}
        products={productBought}
      />
    </ScrollView>
  );
};
```

Фигура 24

След като данните вече са се заредили подавам двата списъка на контейнера ProductList

В ProductList минаваме през всички продукти в приетия списък и ги подаваме на ProductItem

```
const items = products.map((product, index) => {
  return (
    <ProductItem
      product={product}
      subtitle="title"
      showButtons={showButtons}
      updateProduct={updateProduct}
      deleteProduct={deleteProduct}
    />
  );
});
```

Фигура 25

В ProductList също така изчислява общата цена на продуктите.

```
20    });
21    const calcTotalPrice = () => {
22      let total = 0;
23      for (let i = 0; i < products.length; i++) {
24        total += products[i].price;
25      }
26      return total.toFixed(2);
27    };
```


Фигура 26

```
  return (
    <>
      <Card>
        <Card.Title>{title}</Card.Title>
        <Card.Divider />
        {items.length > 0 ? (
          items
        ) : (
          <Text style={{textAlign: 'center'}}>No products</Text>
        )}
        <Card.Divider />
        <Text style={{textAlign: 'center'}}>
          Total price {calcTotalPrice()}
        </Text>
      </Card>
    </>
  );
```

Фигура 27

В ProductItem компонета показваме данните за конкретния продукт

```
return (
  <ListItem key={product.id} bottomDivider>
    <CheckBox checked={Boolean(purchased)} onPress={() => changeBought()} />
    <ListItem.Content>
      <ListItem.Title>{name}</ListItem.Title>
    </ListItem.Content>
    {showButtons && buttons()}
    <Overlay
      isVisible={modalVisibility}
      onBackdropPress={() => setModalVisibility(false)}
      overlayStyle={{width: 300, height: 170}}>
      <Input
        placeholder="product"
        value={editValue}
        onChangeText={(value) => setEditValue(value)}
      />
      <View
        style={{
          marginTop: 20,
          flexDirection: 'row',
          justifyContent: 'flex-end',
        }}>
        <Button
          icon={{
            name: 'edit',
            size: 25,
            color: 'white',
          }}
          buttonStyle={{marginLeft: 10}}
          onPress={() => updateName()}
        />
        <Button--
          icon={{
            name: 'cancel',
            size: 25,
            color: 'white',
          }}
          buttonStyle={{marginLeft: 10}}
          onPress={() => setModalVisibility(false)}
        />
      </View>
    </Overlay>
  </ListItem>
```

Фигура 28

4. Добавяне на нов продукт
   Добавянето на нов продукт се извършва в AddProduct контейнера , чрез изпращане действие (сам когато всички полета са попълнени)

```
40    handleAdd = () => {
41      if (
42        this.state.productName &&
43        !isNaN(this.state.productPrice) &&
44        this.state.category
45      ) {
46        const newProduct = {
47          name: this.state.productName,
48          purchased: false,
49          price: this.state.productPrice,
50          category: this.state.category,
51        };
52        this.props.addProduct(newProduct);
53        this.setState({
54          ...this.state,
55          modalVisibility: true,
56          productName: '',
57          productPrice: '',
58        });
59      }
60    };
```

Фигура 29

Това действие се прихваща от сагата

```
yield takeEvery(NEWPRODUCT, addProduct);
```

Фигура 30

```
47  function addProductEventChannel(payload) {
48    const channel = new eventChannel((emiter) => {
49      const listener = database.transaction(function (tx) {
50        tx.executeSql(
51          'INSERT INTO myProducts (name,category,price,purchased) VALUES (?,?,?,?)',
52          [payload.name, payload.category, payload.price, payload.purchased],
53          (tx, results) => {
54            tx.executeSql('SELECT * FROM myProducts', [], function (tx, res) {
55              emiter({data: res.rows.raw() || {}});
56            });
57          },
58        );
59      });
60      return () => {
61        listener.off();
62      };
63    });
64    return channel;
65  }
66
67  function* addProduct(action, dispatch) {
68    const updatedProduct = addProductEventChannel(action.payload);
69    const item = yield take(updatedProduct);
70    yield put(update(item));
71    yield put({type: ADDINGPRODUCT, value: true});
72  }
73
```

Фигура 31

След като протокът е добавен се извличат всички продукти от базата и се изпраща действие към редюсера за да модифицира писъка с продуктите.

5. Модифицира на единичен продукт

Има два начина на модифицира на продукт

1.Смяна на състоянието на продукта (купен , нужен)

В компонента ProductItem при натискането на отметката се изпраща действие за модифициране на продукта

```
<CheckBox checked={Boolean(purchased)} onPress={() => changeBought()} />
<ListItem.Content>
```

```
const changeBought = () => {
  const newProduct = {...product};
  newProduct.purchased = !newProduct.purchased;
  updateProduct(newProduct);
};
```

Фигура 32

2.Смяна на името на продукта

В компонента ProductItem при натискането на бутона за модифициране се отваря нов прозорец в който може да се промени името на продукта

```
const editButton = () => {
  return (
    <Button
      icon={<Icon name="edit" size={15} color="white" />}
      onPress={() => handleOpenModal()}
    />
  );
};
```

Фигура 33

При отварянето на прозореца се задава стойността на полето в него (името на продукта)


```
const handleOpenModal = () => {
  setEditValue(product.name);
  setModalVisibility(true);
};
```

Фигура 34

Новият прозорец се състои от поле за писане и два бутона – за модифициране и за отказ

```
<Overlay
  isVisible={modalVisibility}
  onBackdropPress={() => setModalVisibility(false)}
  overlayStyle={{width: 300, height: 170}}>
  <Input
    placeholder="product"
    value={editValue}
    onChangeText={(value) => setEditValue(value)}
  />
  <View
    style={{
      marginTop: 20,
      flexDirection: 'row',
      justifyContent: 'flex-end',
    }}>
    <Button
      icon={{
        name: 'edit',
        size: 25,
        color: 'white',
      }}
      buttonStyle={{marginLeft: 10}}
      onPress={() => updateName()}
    />
    <Button
      icon={{
        name: 'cancel',
        size: 25,
        color: 'white',
      }}
      buttonStyle={{marginLeft: 10}}
      onPress={() => setModalVisibility(false)}
    />
  </View>
</Overlay>
```

Фигура 35

При натискането на бутона за модифициране се изпраща действие за промяна на продукта

```
const updateName = () => {
  const newProduct = {...product};
  newProduct.name = editValue;
  updateProduct(newProduct);
  setModalVisibility(false);
};
```

Фигура 36

Действието за модифициране на продукта се прихваща в сагата

```
yield takeEvery(UPDATEPRODUCT, updatedProduct);
```

Фигура 37

Новият продукт се добавя в базата, след което се извличат всички продукти и се изпраща действие към редюсера , за да запази новия списък

```
105  function updateProductEventChannel(payload) {
106    console.log('payload', payload);
107    const channel = new eventChannel((emiter) => {
108      const listener = database.transaction(function (tx) {
109        tx.executeSql(
110          'UPDATE myProducts SET name = ?, category = ? , price = ?, purchased = ? WHERE id = ?;',
111          [
112            payload.name,
113            payload.category,
114            payload.price,
115            payload.purchased,
116            payload.id,
117          ],
118          (tx, results) => {
119            tx.executeSql('SELECT * FROM myProducts', [], function (tx, res) {
120              emiter({data: res.rows.raw() || {}});
121            });
122          },
123        );
124      });
125      return () => {
126        listener.off();
127      };
128    });
129    return channel;
130  }
131
132  function* updatedProduct(action) {
133    const updateChannel = updateProductEventChannel(action.product);
134    const item = yield take(updateChannel);
135    yield put(update(item));
136  }
```

Фигура 38

6. Изтриване на продукт

В компонента ProductItem при натискането на бутона за изтриване се изпраща действие

```
35    const deleteButton = () => {
36      return (
37        <Button
38          icon={<Icon name="delete" size={15} color="white" />}
39          onPress={() => deleteProduct(product.id)}
40        />
41      );
42    };
43
```

Фигура 39

Това действие се прихваща сагата

```
yield takeEvery(DELETEPRODUCT, deleteProduct);
```

Фигура 40

Продуктът се премахва от базата , след което се извличат всички продукти от базата и се праща действие, което модифицира продуктите в магазина


```
138  function deleteProductEventChannel(id) {
139    const channel = new eventChannel((emiter) => {
140      const listener = database.transaction(function (tx) {
141        tx.executeSql(
142          'DELETE FROM myProducts WHERE id=?;',
143          [id],
144          (tx, results) => {
145            tx.executeSql('SELECT * FROM myProducts', [], function (tx, res) {
146              emiter({data: res.rows.raw() || {}});
147            });
148          },
149        );
150      });
151      return () => {
152        listener.off();
153      };
154    });
155    return channel;
156  }
157
158  function* deleteProduct(action) {
159    const updateChannel = deleteProductEventChannel(action.id);
160    const item = yield take(updateChannel);
161    yield put(update(item));
162  }
```

Фигура 41

7. Заключение

Курсовият проект даде добра основа за разработване на мобилни приложения с помощта на React Native и целите на приложението бяха постигнати.

Приложението предоставя интуитивен и лесен за ползване интерфейс, с помощта на който потребителите лесно могат да управляват списъка си за пазаруване.

Идеята за развитие на приложението е :

1: Базата да не е локална

2: Да се създадат профили , което да позволи на повече от един потребител да добавя , модифицира и премахва от един общ списък (Пример: По този начин , ако едно семейство ползва приложението , всеки един член на семейството ще може да управлява същия списък)

8. Библиография

- React Native CLI - https://reactnative.dev/docs/environment-setup
- Redux - https://redux.js.org/
- React-redux - https://react-redux.js.org/
- Redux-Saga - https://redux-saga.js.org
- React-navigation/native - https://reactnavigation.org/
- React-navigation/stack - https://reactnavigation.org/
- React-native-elements - https://reactnativeelements.com/
- React-nativ-vector-icons - https://www.npmjs.com/package/react-native-vector-icons
- React-native-dropdown-picker - https://www.npmjs.com/package/react-native-dropdown-picker
- SQLite – react-native-sqlite-storage - https://www.npmjs.com/package/react-native-sqlite-storage/v/5.0.0

9. Списък на фигурите

Име: …… Факултетен номер: ……