

Київський національний університет імені Тараса Шевченка
Факультет Комп'ютерних наук та кібернетики
Кафедра Математичної інформатики

Курсова робота

на тему:

“Створення мобільного додатку для соціальної мережі”

Виконав
Студент групи МІ-3
Казьмін Євген Валерійович

Науковий керівник
К. Ф-м. н. доцент
Дерев'янченко Олександр Валерійович

Київ - 2018

Зміст

- Вступ
- Вибір засобів розробки
- React Native
 - Загальна інформація
 - Додаток з React Native - це реальний мобільний додаток
 - Реальні приклади
- Налаштування середовища розробки
- Базова структура React
 - Компоненти
 - Props
 - State
 - Підсумок
- Redux
 - Загальна інформація
 - Actions та reducers
 - Підсумок
- Розробка додатку
 - Код додатку
 - Планування архітектури проекту
 - Створення екрану логіну
 - Компонент LoginScreen
 - Функція loginAttempt
 - Запит до api: auth.loginAttempt
 - Auth reducer
 - Навігація в додатку
 - Інші частини додатку
- Висновок
- Джерела

Вступ

У наші дні в інтернеті можна знайти безліч різноманітних мобільних додатків. Кожен з них був створений для деякої мети:

- Гра
- Клієнт деякого сервісу
- Корисна програма
- Інші варіанти

Зараз одним з найперспективніших напрямів ринку ІТ є розробка мобільних додатків. Тому в цій роботі я покажу як створити мобільний клієнт для інтернет-сервісу, описуючи власний досвід створення такого мобільного додатку.

Вибір засобів розробки

Зараз є два основні варіанти розробки мобільних додатків:

- Писати нативний код для кожної з платформ
- Використовувати кросплатформений фреймворк для розробки

Другий варіант є кращим, бо не потрібно переписувати весь код для переносу додатку на іншу платформу.

З мінусів такого підходу можна виділити низьку швидкість роботи додатку, але цей мінус вже майже усунено в сучасних фреймворках.

Моїм вибором став фреймворк React Native, бо я завжди вважав React перспективною розробкою, але не було часу розібратися в ній.

React Native

Загальна інформація

З офіційного сайту[1]: “React Native - інструмент, що дозволяє створювати мобільні додатки використовуючи лише Javascript. Використовує однаковий дизайн з React, маючи декларативний стиль опису користувацького інтерфейсу з компонентів”

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class WhyReactNativeIsSoGreat extends Component {
  render() {
    return (
      <View>
        <Text>
          If you like React on the web, you'll like React Native.
        </Text>
        <Text>
          You just use native components like 'View' and 'Text',
          instead of web components like 'div' and 'span'.
        </Text>
      </View>
    );
  }
}
```

Додаток з React Native - це реальний мобільний додаток

Використовуючи React Native, ви створюєте не “мобільний веб додаток”, “HTML5 додаток” або “гібридний додаток”. Ви створюєте справжній додаток, який не відрізнити від додатку, написаного на Java або Objective-C.

React Native використовує ті самі фундаментальні компоненти користувацького інтерфейсу, що й звичайні iOS або Android додатки. Ви просто кладете до купи ці нативні компоненти, використовуючи JavaScript та React.

Реальні приклади

Ось приклади мобільних додатків, побудованих на React Native:

- Facebook
- Instagram
- Bloomberg
- Pinterest
- Skype
- Tesla
- Uber
- Walmart

Налагодження середовища розробки

Для написання коду можна використовувати будь-який текстовий редактор. Я використовував Sublime Text.

Для розробки на React Native треба встановити Nodejs та npm. Версія JavaScript - ES2015 (також відома як ES6).

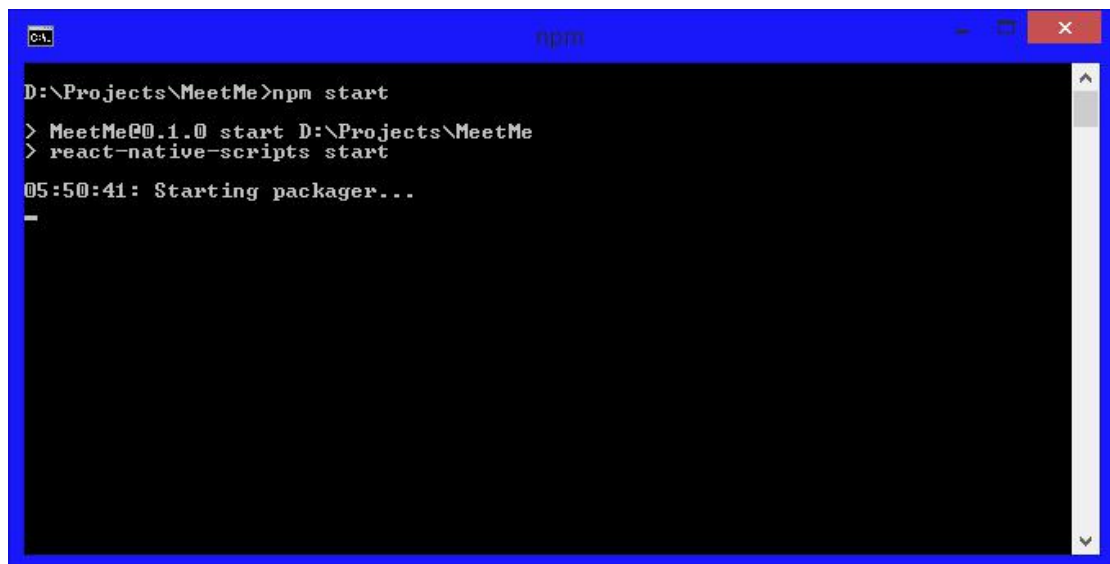
Тепер використовуємо наступні команди:

```
npm install -g create-react-native-app
```

Щоб створити проект з назвою "AwesomeProject" достатньо написати:

```
create-react-native-app AwesomeProject  
  
cd AwesomeProject  
npm start
```

Останній рядок запустить сервер для розробки.



```
CA. npm
D:\Projects\MeetMe>npm start
> MeetMe@0.1.0 start D:\Projects\MeetMe
> react-native-scripts start
05:50:41: Starting packager...
```

Через деякий час з'явиться інформація про успішний запуск. Якщо довго нічого не з'являється, то можна просто натиснути клавішу "q".

React Native дозволяє зручно тестувати додаток на мобільному пристрої. Для цього його не треба постійно перевстановлювати. Після запуску останньої команди ми побачимо наступну картину:



```

Your app is now running at URL: exp://192.168.0.103:80
View your app with live reloading:

Android device:
  -> Point the Expo app to the QR code above.
      <You'll find the QR scanner on the Projects tab of the app.>
iOS device:
  -> Press s to email/text the app URL to your phone.
Emulator:
  -> Press a to start an Android emulator.

Your phone will need to be on the same local network as this computer.
For links to install the Expo app, please visit https://expo.io.

Logs from serving your app will appear here. Press Ctrl+C at any time to stop.

> Press a to open Android device or emulator.
> Press s to send the app URL to your phone number or email address
> Press q to display QR code.
> Press r to restart packager, or R to restart packager and clear cache.
> Press d to toggle development mode. <current mode: development>
```

Тепер на мобільному пристрої встановлюємо додаток Ехро, запускаємо його, там є можливість сканувати QR код. Скануємо код з консолі і через декілька секунд бачимо працюючий прототип додатку на мобільному пристрої.

При зміні коду та збереженні змін (натискання Ctrl+S у редакторі) до Ехро буде надіслано останню версія додатку, і через декілька секунд можна тестувати оновлену версію. Дуже зручно.

Базова структура React

Компоненти

React Native дуже схожий на React[2], просто використовує інші компоненти для побудови інтерфейсу.

Взагалі увесь React - це інструмент для побудови реактивних користувацьких інтерфейсів. Усе будується через компоненти, які є своєрідними будівельними блоками. Ось приклад коду компоненту, який виводить текст "Hello world!":

```
import React, { Component } from 'react';
import { Text } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View>
        <Text>Hello world!</Text>
      </View>

    );
  }
}
```

Тут <View> - це теж деякий компонент. Щоб потім десь використати написаний компонент HelloWorldApp треба лише написати:

```
<View>
  <HelloWorldApp/>
</View>
```

У коді компоненту можна було помітити метод render(). Він викликається, коли потрібно перемалювати компонент. У методі render() бачимо трохи незвичайний return. У нього всередині так званий JSX[3]. Формально, JSX - це просто синтаксичний цукор для функції React.createElement(component, props, ...children).

Наступний код:

```
<MyButton color="blue" shadowSize={2}>  
  Click Me  
</MyButton>
```

Перетворюється в:

```
React.createElement(  
  MyButton,  
  {color: 'blue', shadowSize: 2},  
  'Click Me'  
)
```

Крім `render()` у будь-якого компоненту є ще інші методи та конструктор[4].

Також у будь-якого React компоненту є дві базові складові:

- Props
- State

Props

Props - це параметри які буди передані в компонент.

```
<View>  
  <HelloWorldApp name='John' />  
</View>
```

Тепер у кодї компоненту можна використовувати параметр `this.props.name` (у цьому прикладі, параметр буде мати значення 'John')

Щоб не помилитися з заданням цих параметрів та їх типів існує чудова бібліотека `PropTypes`. Після опису компоненту можна додати наступний блок:

```
HelloWorldApp.propTypes = {  
  name: PropTypes.string  
}
```


Так для чого ж потрібні ці props? Взагалі props - це зовнішні параметри компонента, при їх зміні компонент буде перемальовано. Якщо б ми написали у компоненті HelloWorldApp такий код:

```
render() {  
  return (  
    <View>  
      <Text>Hello, {this.props.name}!</Text>  
    </View>  
  )  
}
```

То при зміні параметру name, змінювався б і відображуваний текст. (У нашому випадку було б "Hello, John!").

Також в props можна передавати функції. Тоді компонент при виникненні деяких ситуацій викликувати ці функції. (Наприклад callback-функція натиску на кнопку).

State

State - внутрішній стан компоненту. По суті, this.state - це об'єкт, який можна використовувати для збереження яких-небудь даних, які впливають на компонент. Наприклад, у компонента checkBox у стані можна зберігати статус чекбокса (checked або unchecked). При зміні state компонент буде перемальовано.

Є один нюанс. Напрямку змінювати this.state неможна (можна, але компонент не перемалюється). Для цього є метод this.setState(newState), який замінює поточний state на newState (поля які не встановлені в newState будуть взяті зі старого state).

```
this.setState({  
  key1: newValue1,  
  key2: newValue2  
})
```

Наприклад приклад простого компоненту - лічильника:

```
class Counter extends Component {
  constructor(props) {
    this.state = {
      currentNumber: 0
    }
  }
  render() {
    return (
      <View>
        <Text>{this.state.currentNumber}</Text>
        <Button
          title='Increment'
          onPress={() => {this.setState({
            currentNumber: this.state.currentNumber + 1
          })}}/>
      </View>
    )
  }
}
```

Підсумок

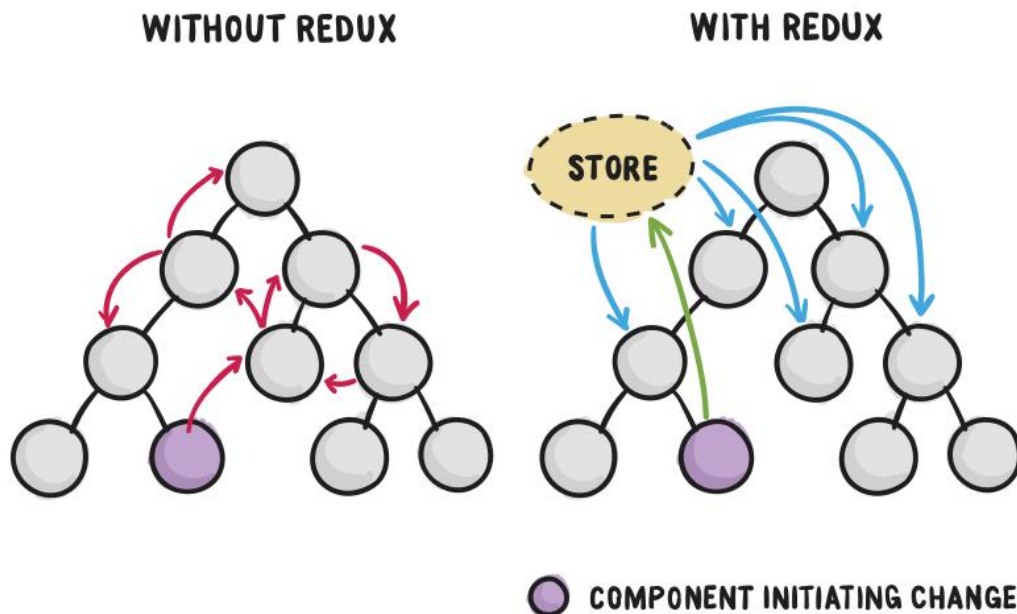
Використовуючи props і state можна написати компонент будь-якої складності, просто розбивши його на дрібніші компоненти. Це основа розробки на React.

Redux

Загальна інформація

Як можна було помітити, до цього часу ми навчилися лише будувати компоненти, які не несли загальної бізнес-логіки. Для цього потрібен якийсь “загальний стан програми”, щоб його можна було змінювати відповідно до ідей бізнес-логіки та реагувати на ці зміни. Накаращою бібліотекою, яка дає такий “глобальний стан” є Redux[5].

А для чого взагалі цей “глобальний стан” потрібен?
Порівняйте:



Тобто бачимо, що без Redux зміна деякого компоненту викликає хаотичну зміну в інших, а з Redux - спочатку змінюється store, а потім усі залежні від цієї зміни компонентів.

Щоб використовувати Redux треба зрозуміти, як проходить взаємодія з глобальним store.

Є дві базові складові розробки на Redux:

- Actions
- Reducers

Actions та reducers

Actions - опис дії, яка змінює store. В загальному випадку це просто об'єкт з параметром type та іншими додатковими параметрами.

Щоб виконати деякий action є метод `dispatch(action)`.

Reducers - такі блоки коду, які на основі отриманого action змінюють store.

Підсумок

Redux - потужна бібліотека для підвищення швидкості розробки та масштабованості програми. Data flow стає значно зрозумілішим. Детальніше можна почитати на офіційному сайті.

Розробка додатку

Код додатку

Увесь код можна знайти на GitHub[6].

Планування архітектури проекту

Темою мобільного додатку було вибрано клієнт для соціальної мережі. Основні екрани, які користувач зможе побачити, було вибрано наступними:

- Екран завантаження
- Екран логіну
- Екран реєстрації
- Екран списку людей в мережі, з якими у користувача є хоча б одна мова
- Екран списку людей в мережі, які знаходяться у зазначеному радіусі від користувача
- Екран профайлу користувача
- Екран налаштувань додатку
- Екран діалогів
- Екран діалогу з іншим користувачем

Задання цих екранів відбувається в головному файлі App.js

Створення екрану логіну

У цьому звіті, для прикладу, буде розказано про створення екрану логіну.

Код екрану (у кодї це називається контейнером) можна знайти у файлі `/containers/LoginContainer.js`

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
import { connect } from 'react-redux'
import LoginScreen from '../components/LoginScreen'
import { loginAttempt } from '../actions'
import { getLastErrorSelector, isLoggedInSelector } from '../reducers/auth'

const mapStateToProps = (state) => ({
  lastError: getLastErrorSelector(state),
  isLoggedIn: isLoggedInSelector(state)
})

const mapDispatchToProps = (dispatch) => ({
  onLoginAttempt: (username, password) => dispatch(loginAttempt(username, password))
})

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(LoginScreen)
```

Спочатку бачимо блок `import`, який підключає необхідні файли.

Потім визначається функція `mapStateToProps`. Вона служить для того, щоб визначити які з `props` цього екрану будуть братися з `Redux store` і яким чином вони будуть братися.

Далі оголошується функція `mapDispatchToProps`. Вона служить для того, щоб в деякі `props` цього екрану покласти функції для зміну `Redux store`, тобто функції, які роблять `dispatch(action)`.

Остання частина - виклик функції `connect(mapStateToProps, mapDispatchToProps)(LoginScreen)`. Ця функція повертає компонент `LoginScreen`, `props` якого було встановлено функціями `mapStateToProps` та `mapDispatchToProps`.

Компонент `LoginScreen`

Цей компонент займає всю область екрану логіну. На ньому є поля для вводу логіна та пароля, а також кнопка "Логін".

У `state` цього компоненту будемо зберігати поточний введений логін і пароль.

Роглянемо `render()` цього компоненту:

```
render() {  
  
  return (  
    <KeyboardAvoidingView>  
      <Text>Username</Text>  
      <TextInput  
        value={this.state.username}  
        onChangeText={(newUsername) => {this.setState({username: newUsername})}}/>  
      <Text>Password</Text>  
      <TextInput  
        secureTextEntry  
        value={this.state.password}  
        onChangeText={(newPassword) => {this.setState({password: newPassword})}}/>  
      <Button  
        title="Login"  
        onPress={() => {  
          this.props.onLoginAttempt(this.state.username, this.state.password);  
        }}/>  
      <Text>{this.props.lastError + ""}</Text>  
      <Text>Are you new here? <Text style={{fontWeight: "bold"}} onPress={()=>{this.props.na  
    </KeyboardAvoidingView>  
  )  
}
```

Спочатку створюється `<KeyboardAvoidingView>` - компонент, який відображає свій зміст таким чином, що клавіатура мобільного додатку не спричиняє незручностей при користуванні додатком.

Потім `<Text>` - компонент для виводу тексту.

`<TextInput>` - компонент для вводу тексту. Один такий компонент створено для логіну, ішний для паролю. Як бачимо в `onChangeText` передаємо функцію, яка змінює поточний state компоненту.

`<Button>` - компонент кнопки, яка робить спробу авторизації. Функція `onLoginAttempt` була передана з `LoginContainer` і по суті є `Redux action`, бо ця функція має зробити реквест на API та перевести додаток у стан очікування відповіді від сервера.

Функція loginAttempt

Розглянемо код цієї функції:

```
export const loginAttempt = (username, password) => (dispatch) => {  
  
  dispatch({  
    type: types.LOGIN_ATTEMPT_REQUEST,  
    username  
  })  
  
  auth.loginAttempt(username, password,  
    (accessToken, refreshToken) => {  
    dispatch({  
      type: types.LOGIN_ATTEMPT_SUCCESS,  
      accessToken,  
      refreshToken,  
    })  
  },  
    (error) => {  
    dispatch({  
      type: types.LOGIN_ATTEMPT_FAILURE,  
      error,  
    })  
  })  
  )  
}
```

Спочатку викликається action про те, що програма очікує відповіді з сервера.

Потім виконується асинхронний запит на сервер, та на випадок успішного/невдалого результату створюється ще по одному з action.

Запит до api: auth.loginAttempt

Розглянемо код цієї функції:

```
loginAttempt: (username, password, successCallback, failureCallback) => {
  fetch('https://s-n.herokuapp.com/api/auth/token/obtain/', {
    method: 'POST',
    headers: {
      "Content-type": "application/x-www-form-urlencoded; charset=UTF-8"
    },
    body: 'username=' + username + '&password=' + password
  })
  .then(
    function (response) {
      console.log(response);
      if (response.status !== 200) {
        response.json().then((data) => {
          let error = ''
          for (let i in data)
            error += i + ':' + data[i] + '\n'
          failureCallback(error)
        })
        return
      }
      response.json().then((data) => {
        if (data.access !== 'undefined')
          successCallback(data.access, data.refresh)
        else
          failureCallback('Unknown error')
      })
    }
  )
}
```

Як бачимо тут немає React/Redux специфічного коду, просто виконується запит на API і обробляється відповідь. Для запитів використовується `fetch[7]`.

Auth reducer

Розглянемо код reducer-а, який змінює Redux store при виклику певного action:

```
const authReducer = (state = initialState, action) => {
  switch (action.type) {
    case REGISTER_ATTEMPT_REQUEST:
    case LOGIN_ATTEMPT_REQUEST:
      return Object.assign({}, state, {
        isAttempting: true,
        username: action.username
      })
    case REGISTER_ATTEMPT_SUCCESS:
    case LOGIN_ATTEMPT_SUCCESS:
      return Object.assign({}, state, {
        loggedIn: true,
        isAttempting: false,
        accessToken: action.accessToken,
        refreshToken: action.refreshToken,
        lastError: ""
      })
    case REGISTER_ATTEMPT_FAILURE:
    case LOGIN_ATTEMPT_FAILURE:
      console.log(action)
      return Object.assign({}, state, {
        loggedIn: false,
        isAttempting: false,
        lastError: action.error,
      })
    default:
      return initialState;
  }
}
```

Цей код одночасно обробляє action-и і на логін, і на реєстрацію.

Можна виділити 3 типи action:

- REQUEST
- SUCCESS
- FAILURE

На кожен з них особливим особом змінюється Redux store.

Поля `loggedIn` та `lastError` можна було помітити в компоненті `LoginScreen`. Це саме ці поля передаються в `props` компоненту, тому якщо у цьому `reducer`-і ці поля буде змінено, то компонент перемалюється. Також в компоненті `LoginScreen` є такий шматок коду:

```
componentDidUpdate() {  
  if (this.props.isLoggedIn)  
    this.props.navigation.navigate('drawer');  
}
```

Цей метод викликається після оновлення компоненту, тут я перевіряю поле `isLoggedIn`. Якщо воно `true`, це означає, що користувач залогінився і його можна направити на основну частину додатку.

Навігація в додатку

Для навігації (зміни екранів) у додатку я використовую бібліотеку `React Navigation`[8]. Завдяки ній, я можу позначити кожен екран своїм словом-ключем і легко переходити на екрани наступним способом:

```
this.props.navigation.navigate('drawer');
```

Інші частини додатку

Також у додатку було реалізовано декілька інших екранів описаним вище способом. Щоб не витратити місце, опис приводити не буду, код можна подивитися у репозиторії.

Висновок

У ході даної роботи я навчився створювати мобільні додатки за допомогою фреймворку `React Native` та бібліотек `Redux`, `React Navigation`. Як можна було помітити, процес створення був дуже легким та покроковим. Я вважаю, що такий спосіб створення кросплатформених додатків - майбутнє галузі `mobile-development`.

Джерела

1. React Native (<http://facebook.github.io/react-native/>)
2. React (<https://reactjs.org/docs/components-and-props.html>)
3. JSX - деталі (<https://habr.com/post/319270/>)
4. Lifecycle hooks
(<https://www.fullstackreact.com/30-days-of-react/day-7/>)
5. Redux (<https://redux.js.org/>)
6. Source code (<https://github.com/aangairbender/meetme>)
7. Вступ до fetch (<https://habr.com/post/252941/>)
8. React Navigation (<https://reactnavigation.org/>)