

Лекция 14

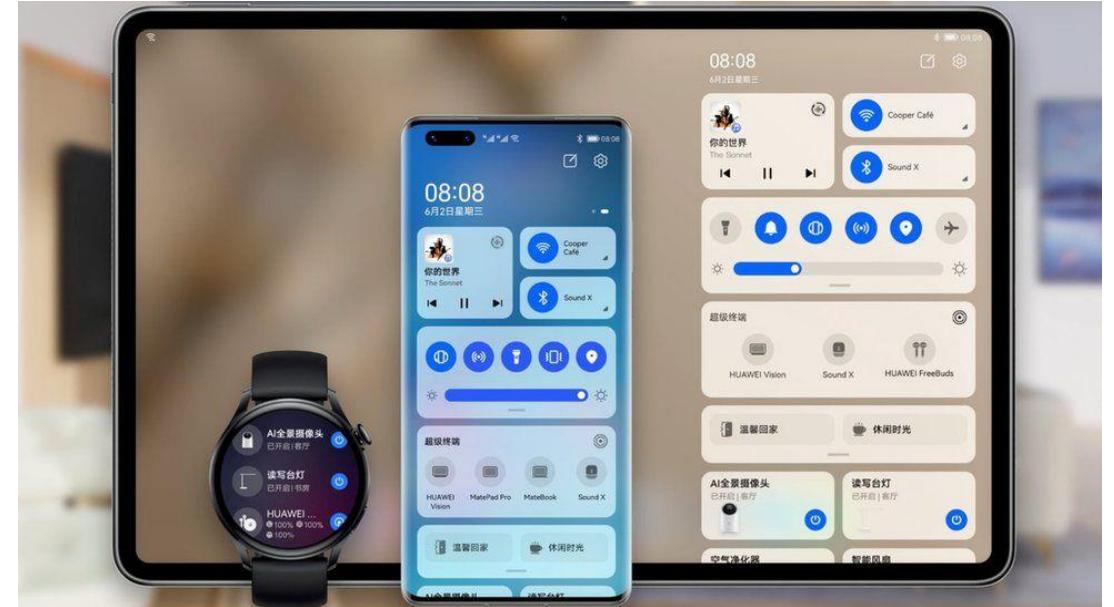
Мобильные приложения

Разработка интернет приложений

Канев Антон Игоревич

Мобильные приложения

- **Мобильное приложение** («Mobile application») — программное изделие, разновидность прикладного программного обеспечения, предназначенная для работы на смартфонах, планшетах и других мобильных (портативных, переносных, карманных) устройствах

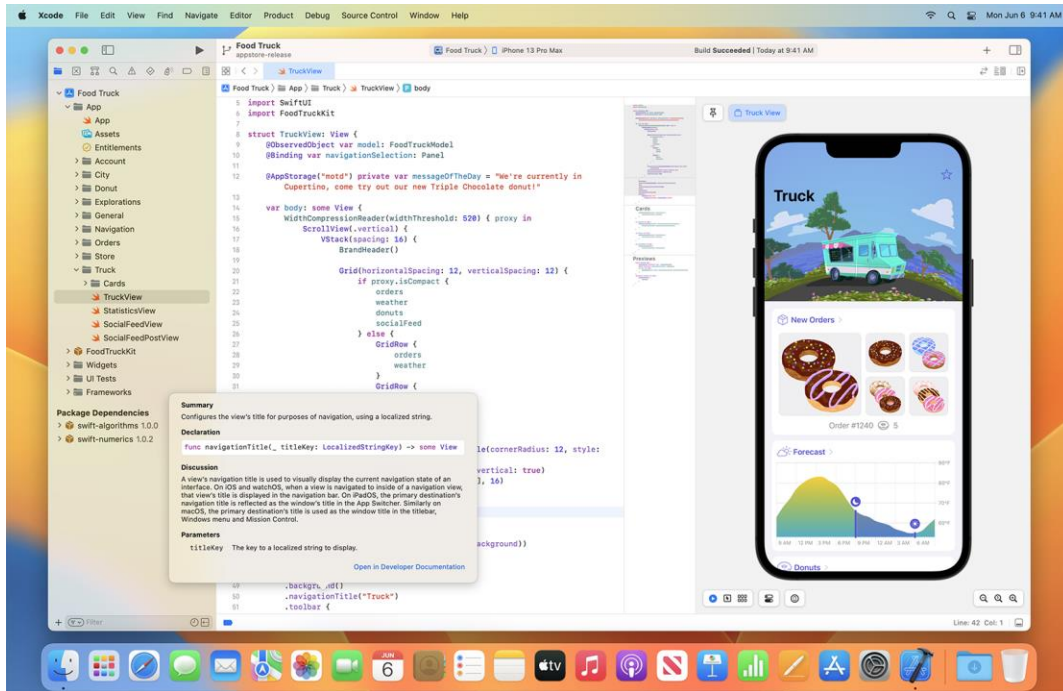


ЯЗЫКИ

- iOS: Objective-C, Swift
- Android: Java, Kotlin

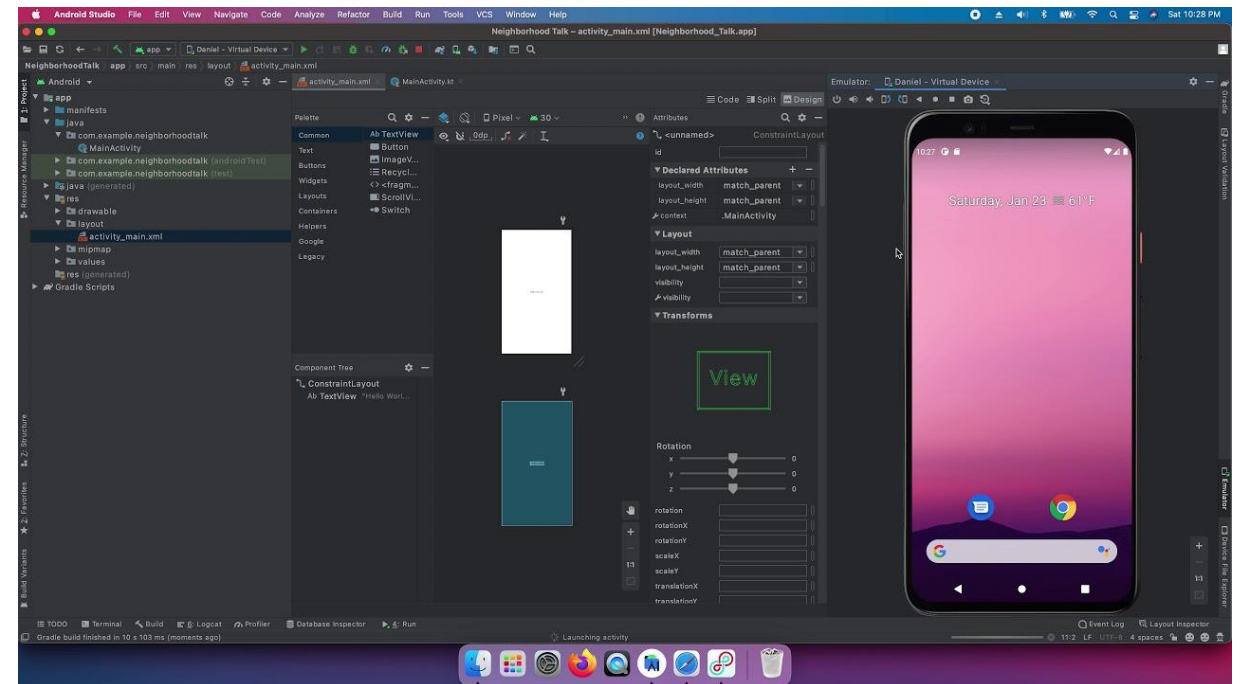


Среды разработки



- Xcode


- Android Studio




Web vs мобильные приложения

11:13 LTE 79

AA online.sberbank.ru

 Добавить ярлык СберБанк Онлайн на экран «Домой»
Нажмите на иконку и в меню выберите «На экран «Домой»»

 СБЕР БАНК

Логин [вход по номеру телефона](#)

Пароль [изменить пароль](#)




☒ Запомнить меня

[Продолжить](#)


СберБанк обрабатывает Cookies с целью персонализации сервисов и для того, чтобы пользоваться сайтом было удобнее. Пожалуйста, ознакомьтесь с [Политикой использования Cookies](#)

[Подробнее](#)

[Принять](#)

< >   

11:13 LTE 79

 СБЕР БАНК

Логин [вход по номеру телефона](#)

Пароль [изменить пароль](#)

☒ Запомнить меня

[Продолжить](#)

[Зарегистрироваться](#)

[Восстановить доступ](#)




[Нет карты Сбера](#)

Правила безопасности

Никому не говорите свои коды из СМС, даже если к вам обращаются от имени сотрудников Сбербанка. Это мошенники.

Если вас просят установить программу на ваше устройство под предлогом

11:13 LTE 79


Введите пароль

• • • • •

1 2 3
ABC DEF




4 5 6
GHI JKL MNO

7 8 9
PQRS TUV WXYZ

0 

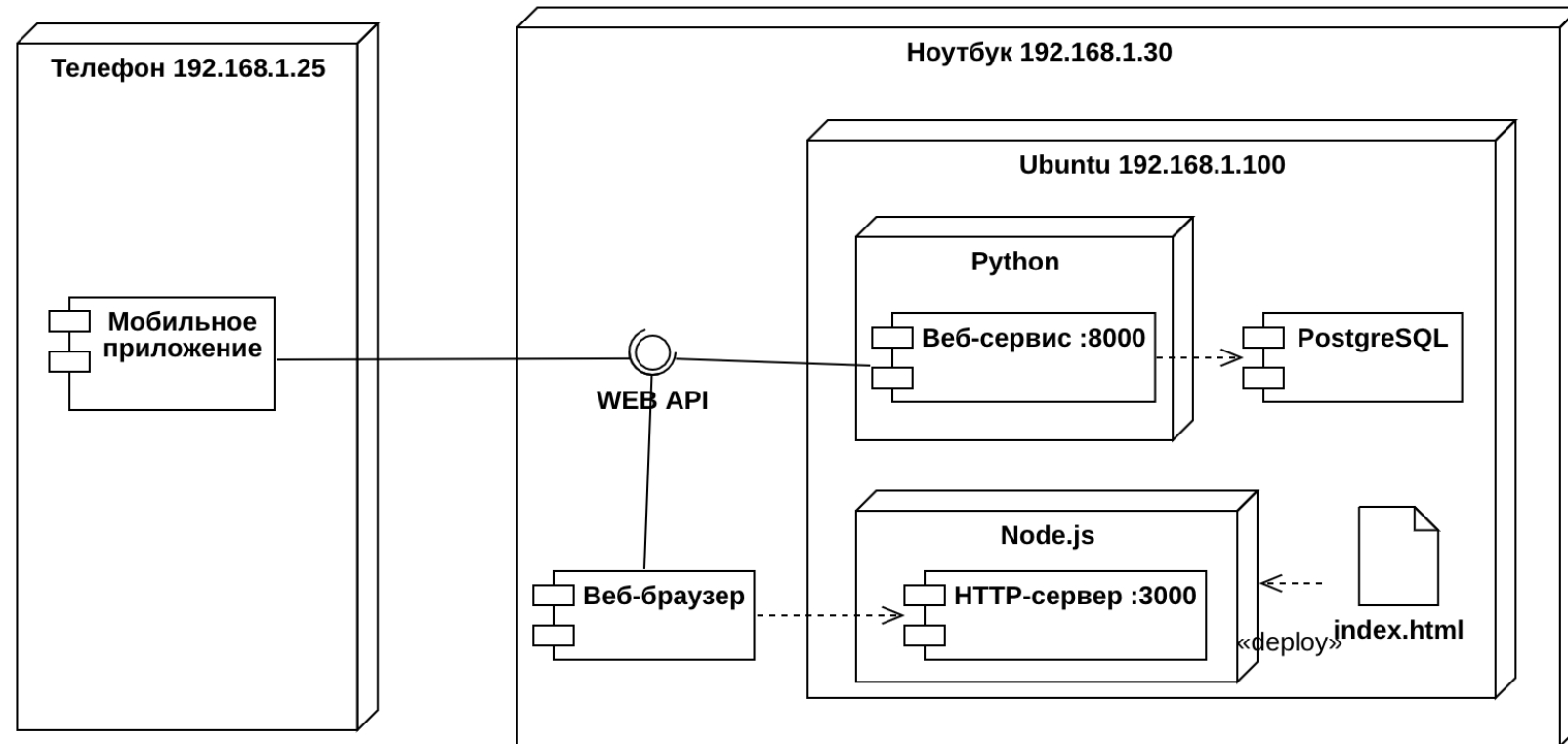
[Не можете войти?](#)

Версия 12.15.0

[Вход](#)  [На карте](#)  [Сменить](#) 

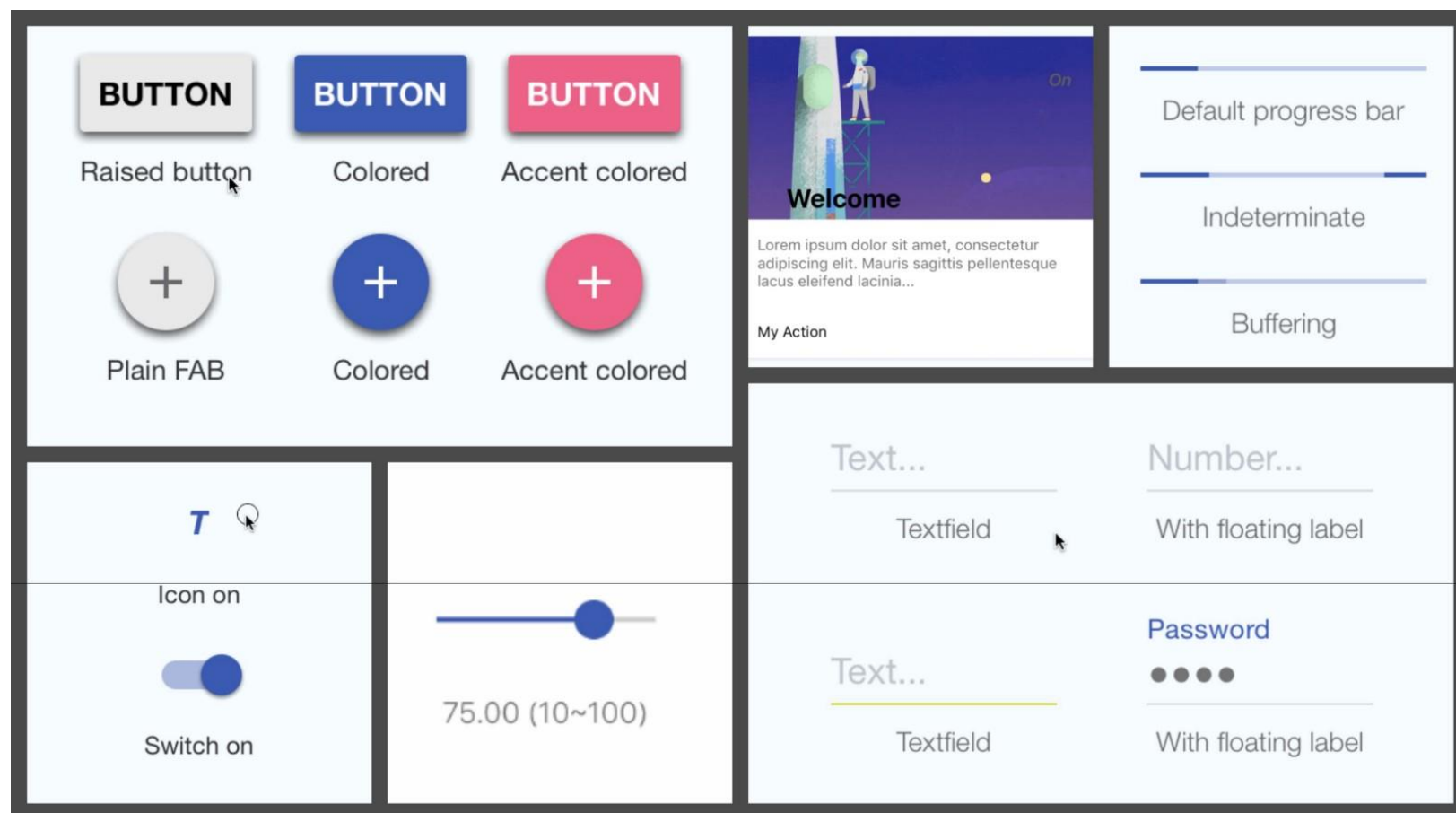
Трехзвенная архитектура. API

- Наше десктопное, кроссплатформенное или мобильное приложение должно обращаться к разработанному нами API
- Использовать два запроса: GET списка услуг и GET одной услуги



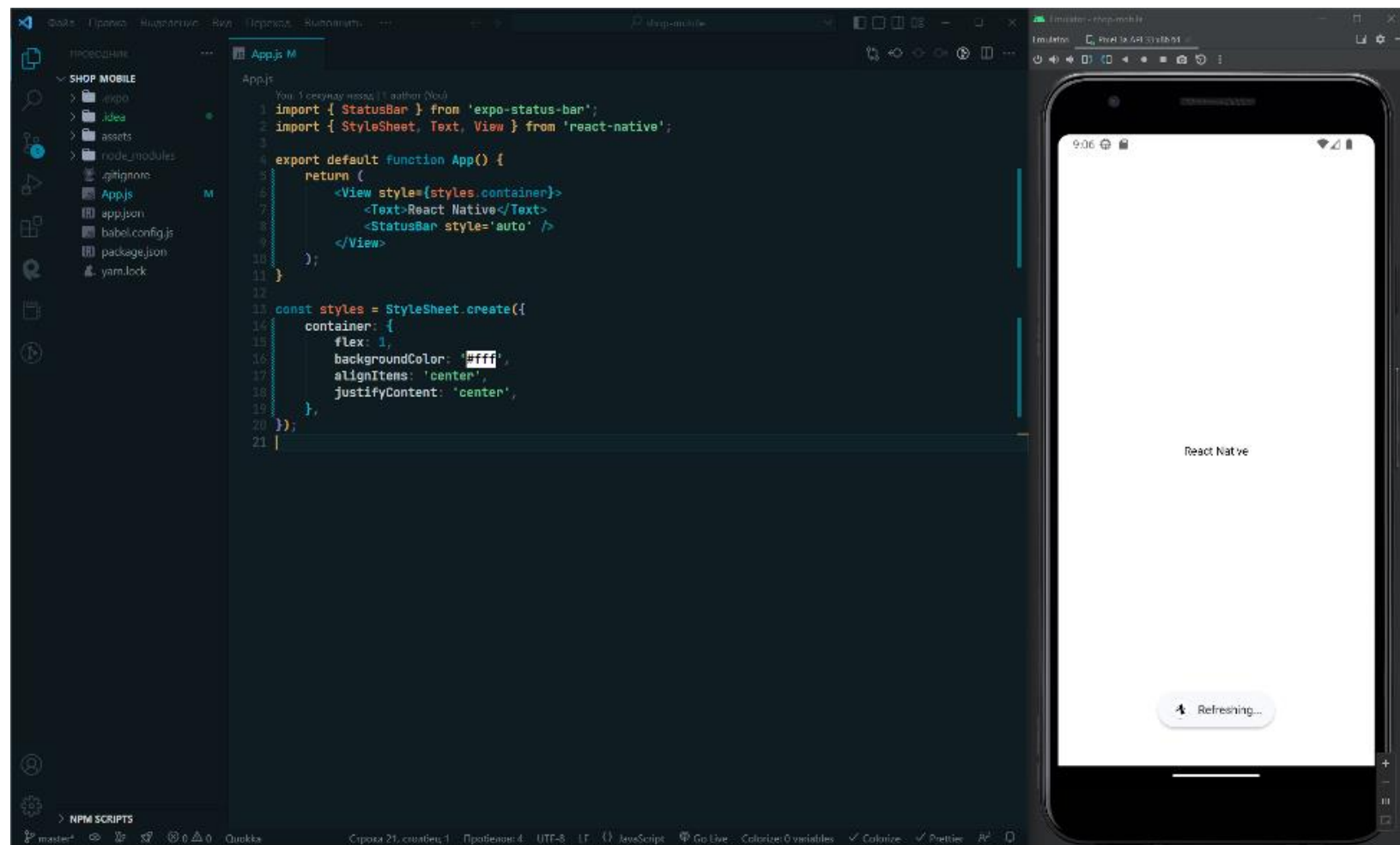
React Native

- React Native – фреймворк для кроссплатформенной разработки на JavaScript
- Позволяет создавать нативные приложения с помощью известных нам технологий: axios, redux-toolkit и тд



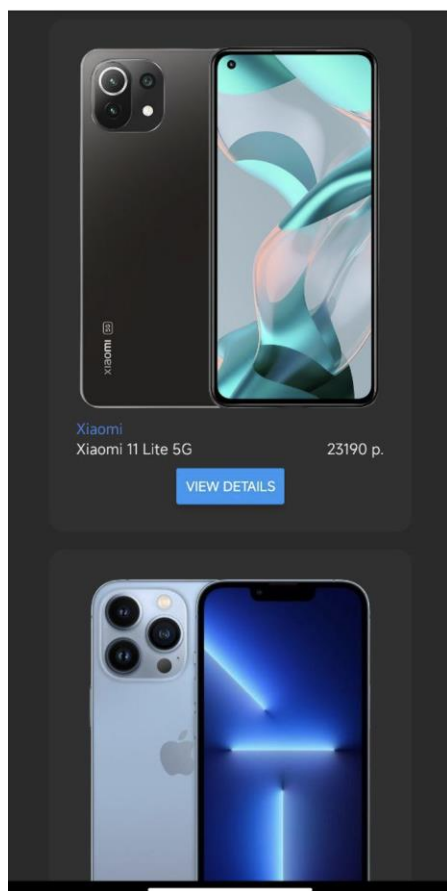
React Native

- Мы можем вести разработку в VS Code и смотреть изменения в телефоне через QR
- Или в Android Studio и эмуляторе



React Native

- Создадим карточки и заполним их данными из API



```
export default function ShopScreen({ navigation }) {
  const dispatch = useDispatch();
  const { devices } = useSelector((store) => store.device);

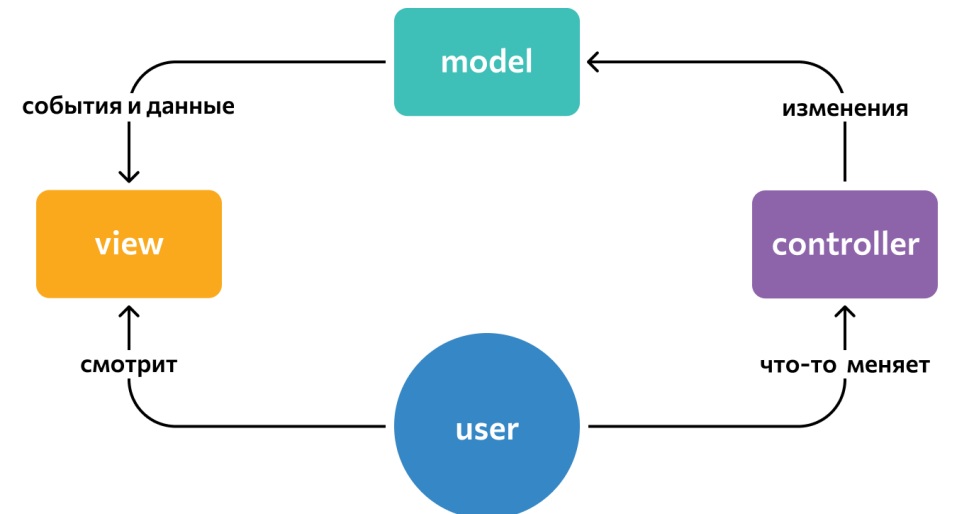
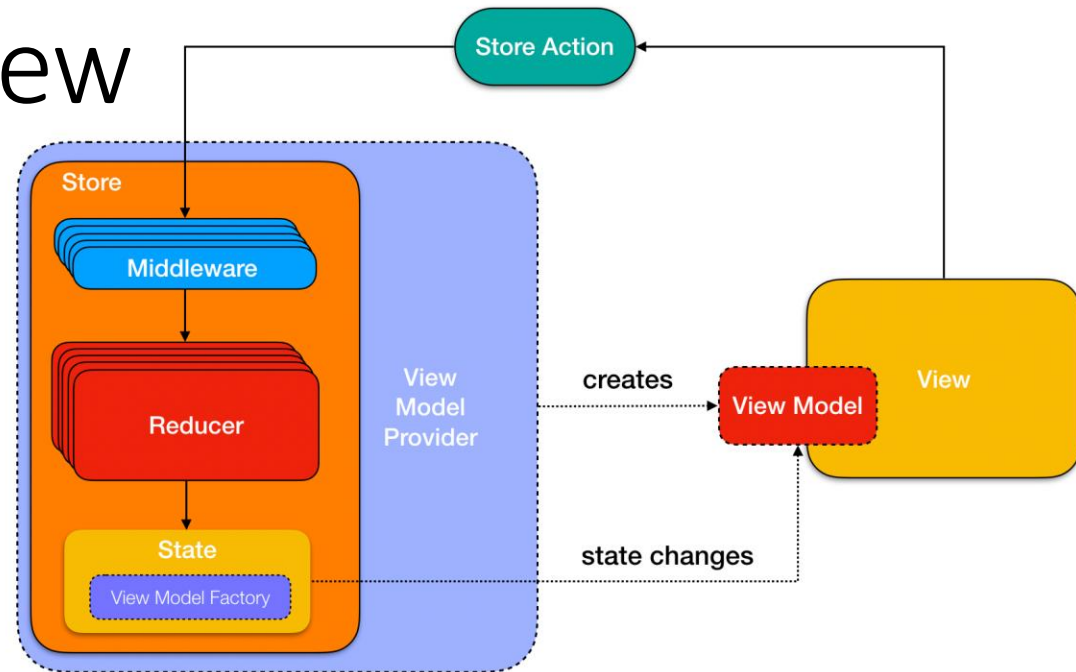
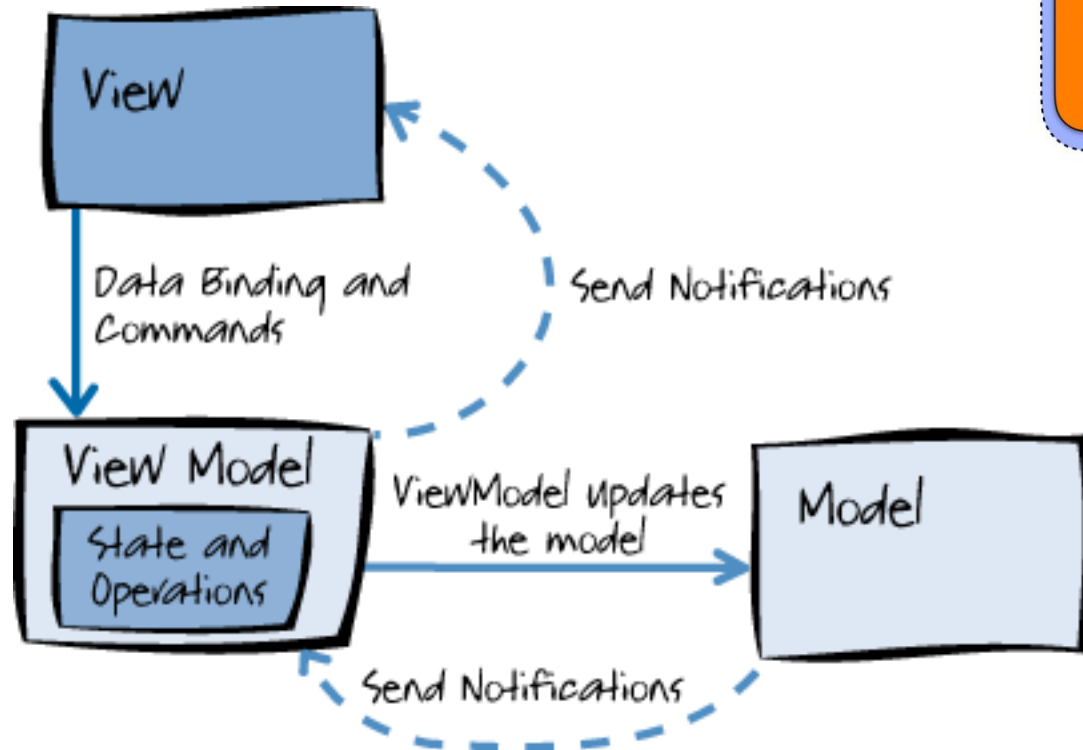
  useEffect(() => {
    async function getAllDevices() {
      await axiosInstance.get('/device').then((response) => dispatch(setDevices(response?.data)));
    }
    getAllDevices();
  }, [dispatch]);

  return (
    <ScrollView>
      <View style={styles.page}>
        {!!devices &&
          devices.map((device) => <DeviceCard key={device.id} {...device} navigation={navigation} />)}
      </View>
    </ScrollView>
  );
}

const styles = StyleSheet.create({
  page: {
    display: 'flex',
    width: '100%',
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#2a2a2a',
  },
});
```

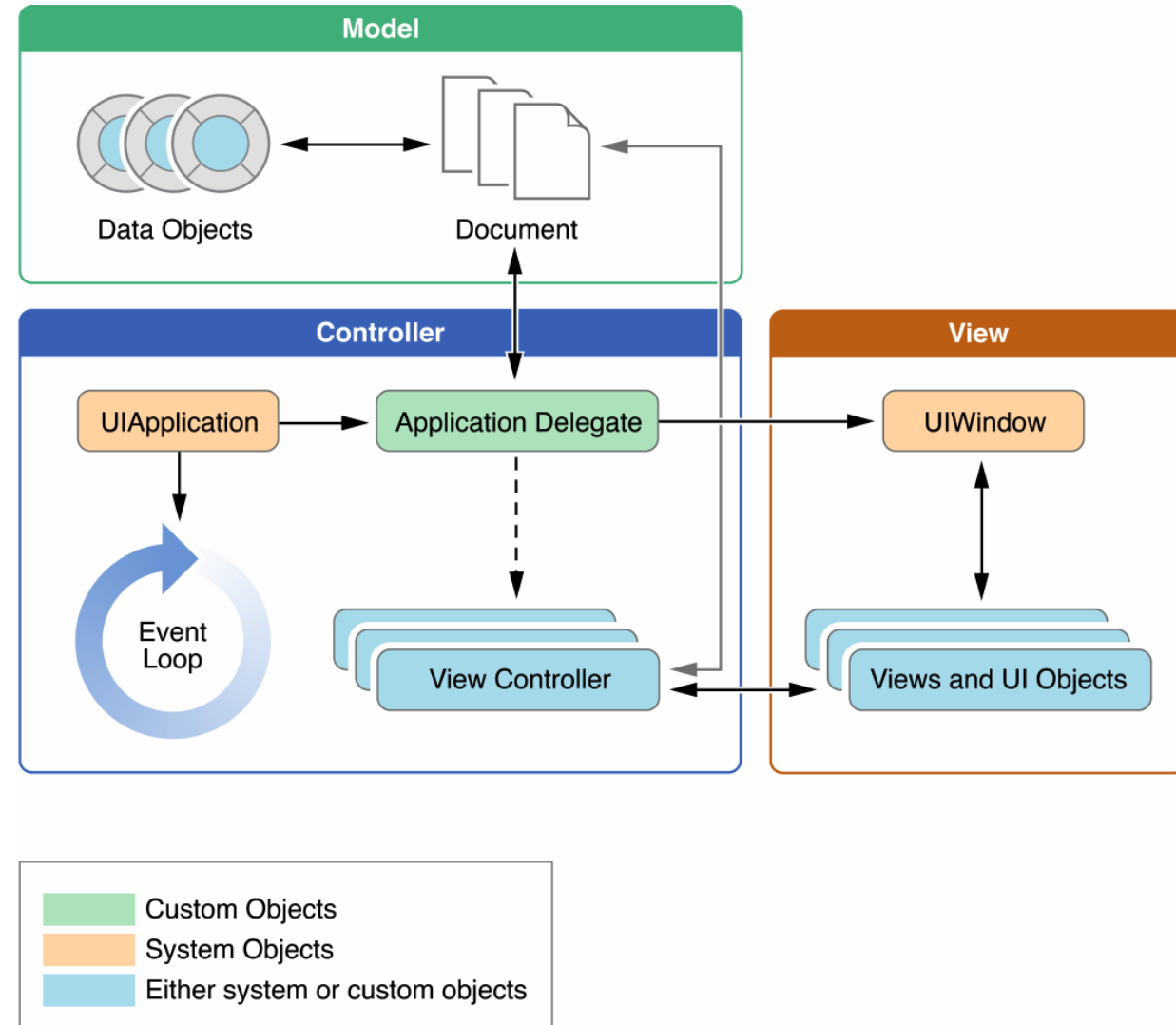
Model ViewModel View

- Паттерн MVVM должен быть нам уже знаком на примере React
- MVVM отличается от MVC



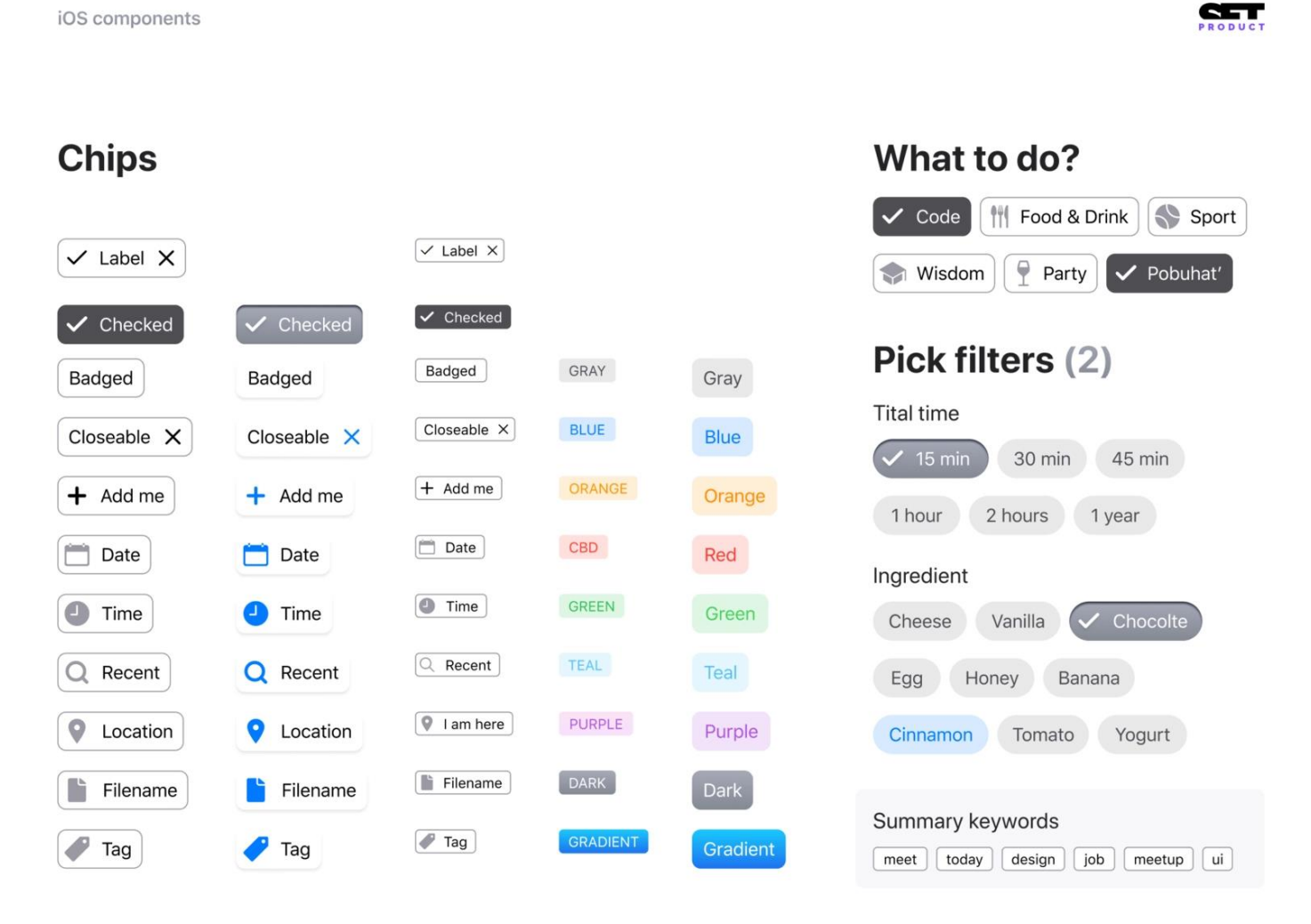
Архитектура Swift приложения

- **Модель** – отвечает за использование предметных (бизнес, domain) данных. Активные модели умеют уведомлять окружающих об изменениях в себе, а пассивные — нет.
- **Представление** (Вид, View) – отвечает за слой представления (GUI). Не обязательно должен быть связан с UI отрисовкой. Помимо представления пользователю данных, у него есть ещё одна важная задача: принять событие пользователя.
- **Контроллер/Презентер/ViewModel** – так или иначе отвечают за связь модели с контроллером. В основном занимаются тем, что пробрасывают события модели в представление, а события представления – в модель, соответствующим образом их преобразуя и обрабатывая.



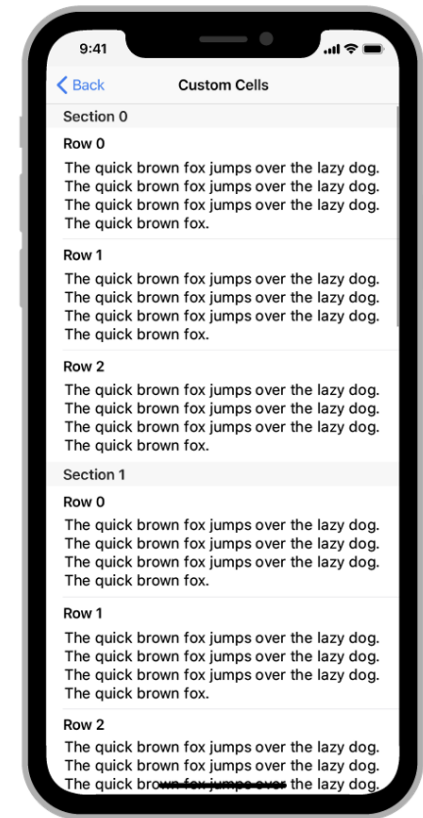
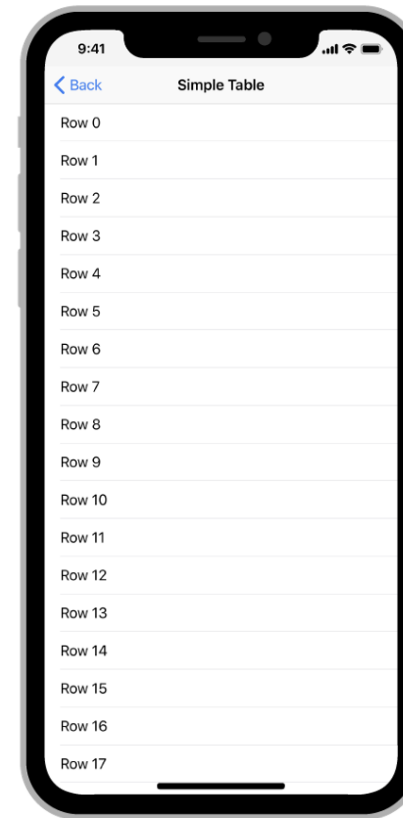
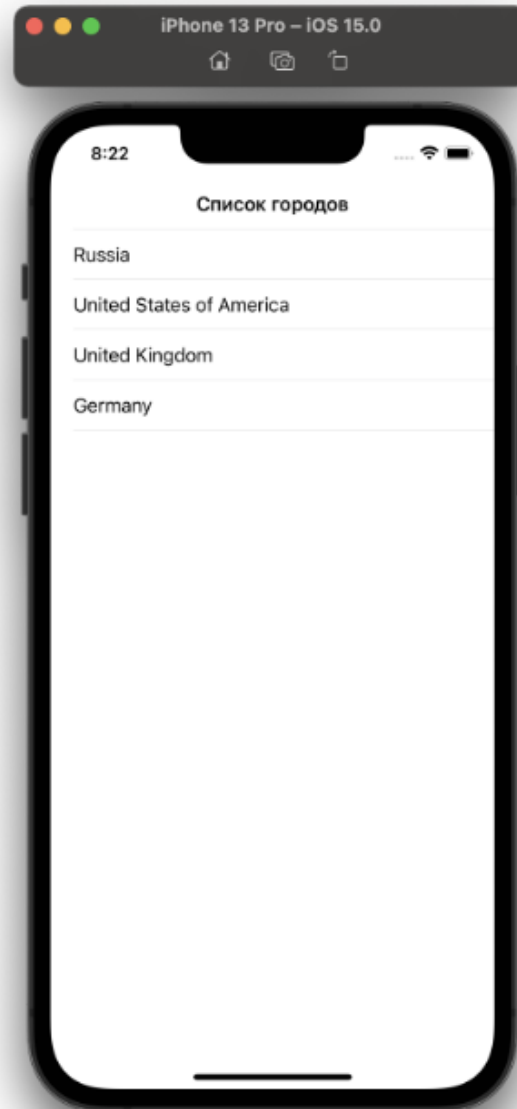
UI компоненты

- Уже знакомый нам термин UI kit
- Использование кнопок, изображений, списков и других компонентов



UITableView

- Для того, чтобы на экране отобразилась таблица, необходимо создать переменную класса `WeatherViewController` типа `UITableView` и задать там первичные настройки



Модель данных

- В данном пункте мы создадим модель данных, которая соответствует тому, что вы уже создали на бэкенде.
- В эту модель данных будет парситься json. Также мы создадим запрос к вашему сервису и сам парсинг ответа.
- Прежде чем приступить к созданию подключения сервиса необходимо задать модель с данными, которые придут в ответе от сервиса.

```
import Foundation

struct WeatherData: Codable {
    var location: Location
    var current: Current
}

struct Location: Codable {
    var name: String
    var country: String
    var region: String
}

struct Current: Codable {
    var observation_time: String
    var temperature: Int
    var wind_speed: Int
    var pressure: Int
    var feelslike: Int
}
```

Генерация запроса

- Добавляем обращение в внешнему API
- В вашей лабораторной вы заменяете URL на ваш API
- Для эмулятора можно указать localhost
- Для показа IP в локальной сети, например 192.168.100.108

```
func configureURLRequest(city: String) -> URLRequest {
    var request: URLRequest
    let accessToken: String = "b849bbbe085e655065bb8546ec2a8dd5" // нужен для weather-api

    let queryItems = [
        URLQueryItem(name: "access_key", value: accessToken),
        URLQueryItem(name: "query", value: "'\((city)')")
    ]
    guard var urlComponents = URLComponents(string: "http://api.weatherstack.com/current") else {
        // если не получится создать компоненты из своих query параметров, то переходим на google
        return URLRequest(url: URL(string: "https://google.com")!)
    }

    urlComponents.queryItems = queryItems

    guard let url = urlComponents.url else {
        // если не получится создать url из своего адреса, то переходим на google
        return URLRequest(url: URL(string: "https://google.com")!)
    }

    request = URLRequest(url: url)
    request.httpMethod = ApiMethods.post.rawValue // устанавливаем метод запроса через enum
    return request
}
```

Запросы к API

- Создание обработчиков запросов к собственному API сервису в отдельном файле
- Указываем в какие переменные мы должны положить полученные файлы

```
import Foundation

final class ApiService {

    func getWeatherData(city: String, completion: @escaping (WeatherData?, Error?) -> ()) {
        let request = configureURLRequest(city: city) // конфигурация кастомного запроса

        URLSession.shared.dataTask(with: request, completionHandler: { data, response, error in // completionHandler

            if let error = error {
                print("error")
                completion(nil, error)
            }
            if let response = response {
                print(response)
            }
            guard let data = data else {
                completion(nil, error)
                return
            }

            do {
                let weatherData = try JSONDecoder().decode(WeatherData.self, from: data) // декодируем json в объект
                completion(weatherData, nil)
            } catch let error {
                completion(nil, error)
            }
        }).resume() // запускаем задачу
    }
}
```

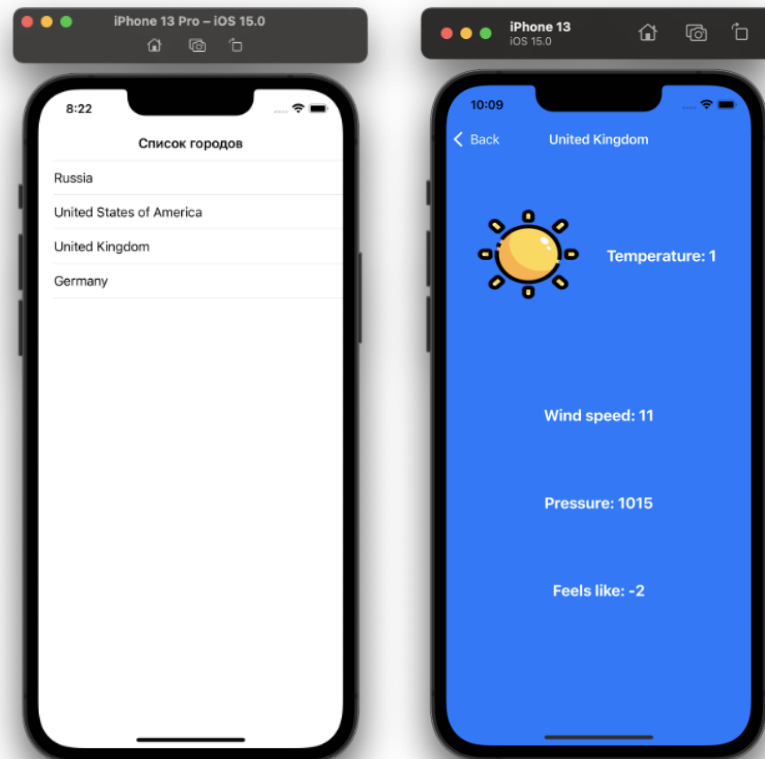

Заполнение страницы данными

```
private func loadWeatherData(cities: [String]) {
    guard let apiService = apiService else { // раскрытие опциональной переменной apiService
        return
    }

    cities.forEach {
        apiService.getWeatherData(city: $0, completion: { [weak self] (weatherData, error) in // weak self для
            DispatchQueue.main.async { // запуск асинхронной задачи на main потоке из-за обработки на ui !!!
                guard let self = self else { return }
                if let error = error {
                    // показ ошибки
                    self.present(UIAlertController(title: "ERROR", message: error.localizedDescription, preferredStyle: .alert))
                    return
                }
                if let weatherData = weatherData {
                    self.weatherListData.append(weatherData) // массив с данными о погоде
                }
                self.weatherListTableView.reloadData() // перезагрузка таблицы для отображения новых данных
            }
        })
    }
}
```

Переходы между страницами

- Далее необходимо добавить переход на данный экран с основного



```
import Foundation
import UIKit
```

```
final class WeatherInfoViewController: UIViewController {
    private var weatherData: WeatherData

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    init(weatherData: WeatherData) {
        self.weatherData = weatherData
        super.init(nibName: nil, bundle: nil)
    }
}
```

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    let weatherInfoViewController = WeatherInfoViewController(weatherData: self.weatherListData[indexPath.row])
    navigationController?.pushViewController(weatherInfoViewController, animated: true)
}
```

Заполнение детальной информации

- Создадим функцию, которая будет сохранять в текстовые лейблы значения строк с детальной информацией об объекте, которые мы передали с первого экрана.
- которая вызывается из инициализатора контроллера

```
private var weatherData: WeatherData

init(weatherData: WeatherData) {
    self.weatherData = weatherData
    super.init(nibName: nil, bundle: nil)
    fillData(withModel: weatherData)
}
```

```
func fillData(withModel: WeatherData) {
    degreeLabel.text = "Temperature: " + String(withModel.current.temperature)
    windLabel.text = "Wind speed: " + String(withModel.current.wind_speed)
    pressureLabel.text = "Pressure: " + String(withModel.current.pressure)
    feelslikeLabel.text = "Feels like: " + String(withModel.current.feelslike)
}
```

Верстка страницы с деталями

```
final class WeatherInfoViewController: UIViewController {
    //добавим на экран элементы, которые хотим отобразить на экране
    private let imageView = UIImageView()
    private let degreeLabel = UILabel()
    private let windLabel = UILabel()
    private let pressureLabel = UILabel()
    private let feelslikeLabel = UILabel()

    //создадим переменную для хранения детальной информации об объекте
    private var weatherData: WeatherData

    override func viewDidLoad() {
        super.viewDidLoad()
        configure()
        configureDataElements()
    }

    //зададим базовые настройки для текстовых полей и добавим их на экран
    private func configureDataElements() {
        [degreeLabel, windLabel, pressureLabel, feelslikeLabel].forEach {
            $0.translatesAutoresizingMaskIntoConstraints = false
            $0.font = UIFont.systemFont(ofSize: 20, weight: .bold)
            $0.textColor = .white
            view.addSubview($0)
        }

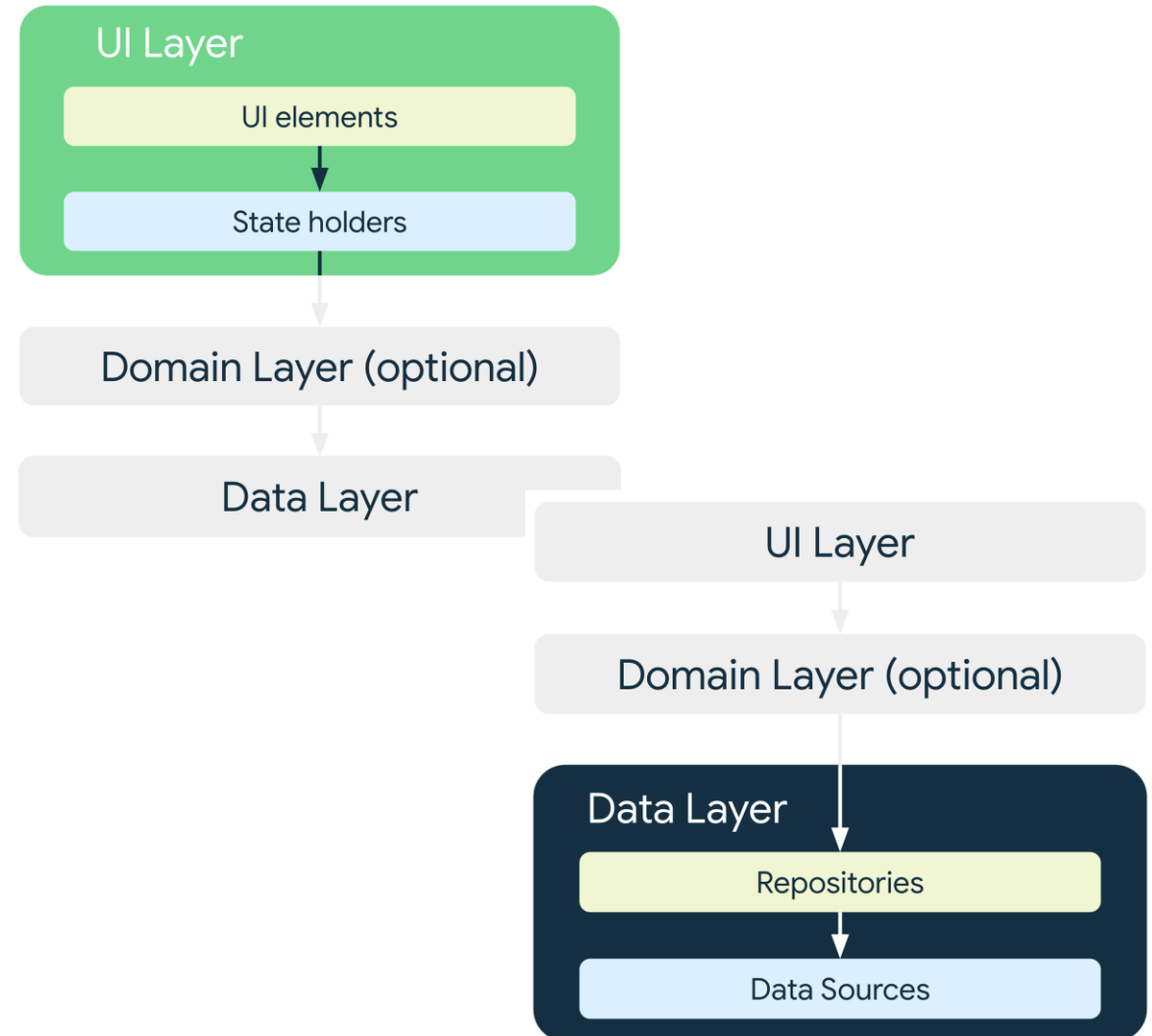
        //зададим констрейнты и базовые настройки для картинки
        imageView.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(imageView)
        imageView.heightAnchor.constraint(equalToConstant: 250).isActive = true
        imageView.widthAnchor.constraint(equalToConstant: 200).isActive = true
        imageView.leftAnchor.constraint(equalTo: view.leftAnchor, constant: 5).isActive = true
        imageView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor).isActive = true

        imageView.image = UIImage(named: "sunny")
    }
}
```

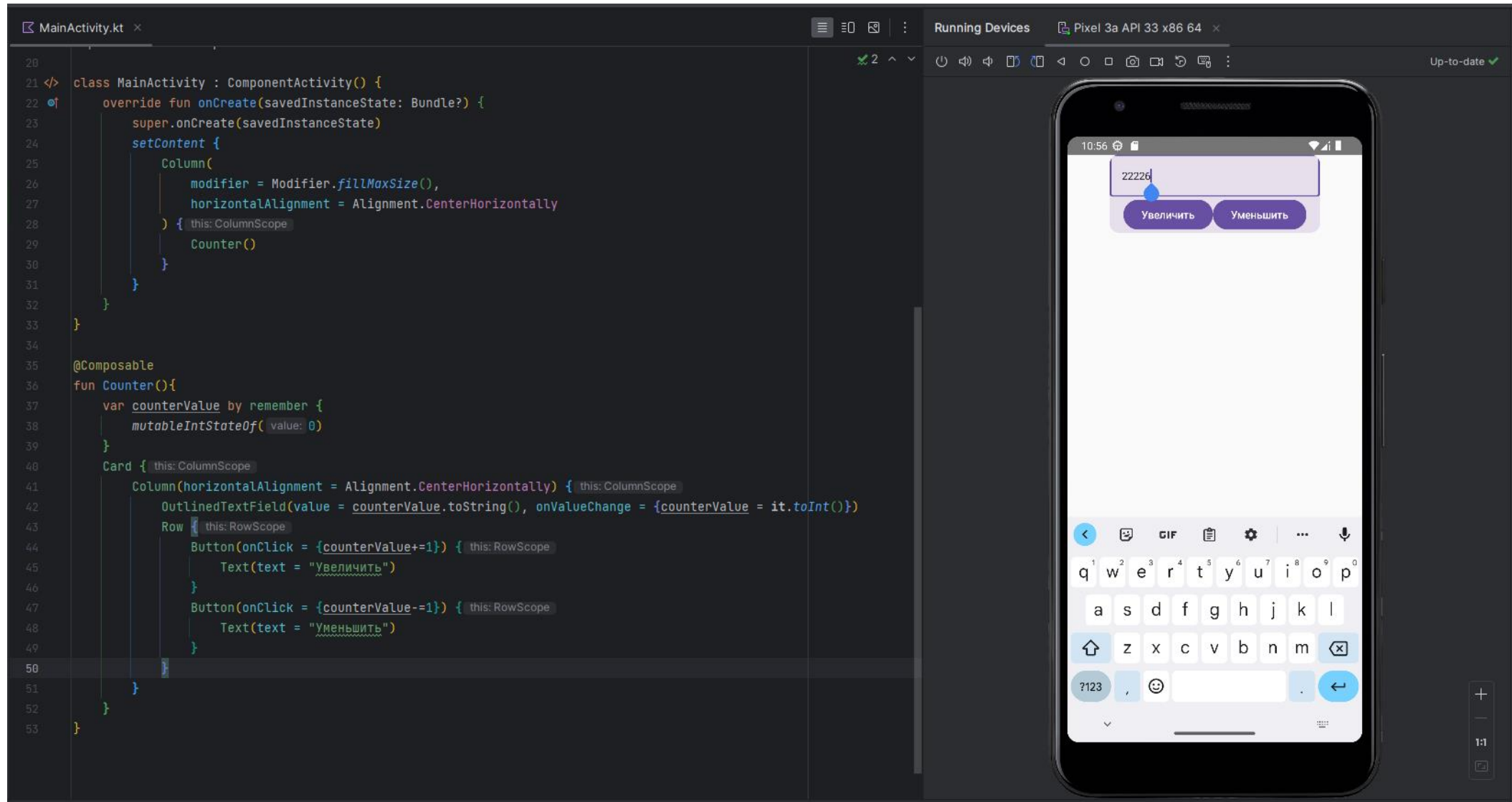


Архитектура Android приложения

- **Роль слоя UI** (или *слоя представления*) — отображать на экране данные приложения.
- **Слой данных** в приложении содержит *бизнес-логику* — правила, по которым приложение создаёт, хранит и изменяет данные.
- **Доменный слой** располагается между слоями UI и данных. Доменный слой отвечает за инкапсуляцию сложной бизнес-логики или простой бизнес-логики, которую переиспользуют несколько ViewModel.

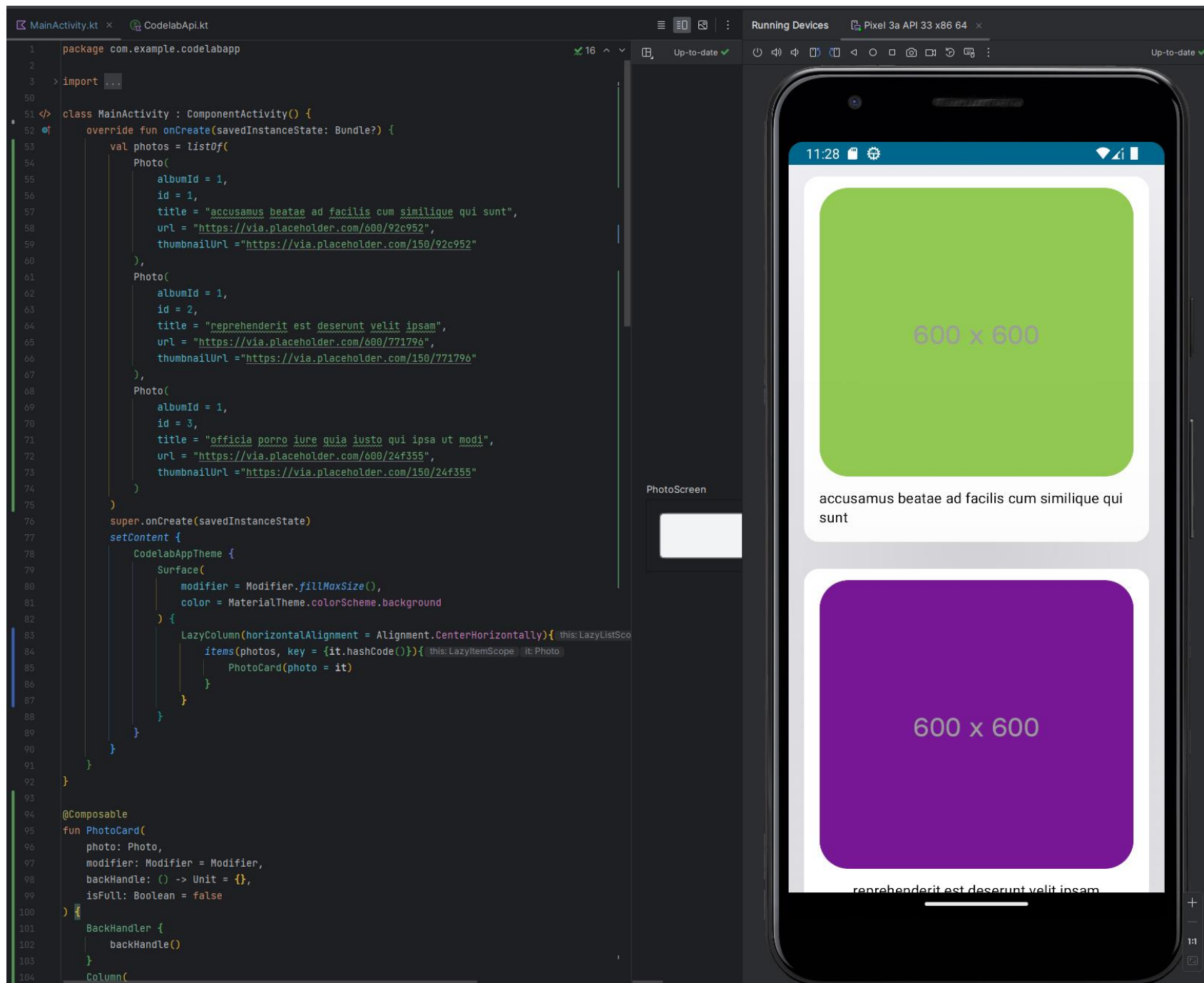


Счетчик в Android Studio



Окно списка

- Создаем карточки на основе mock
- Помещаем данные из модели в компоненты нашего экрана



Модель для полученных данных

- Сделаем структуру – модель для полученных от API данных

```
Body Cookies Headers (24) Test Results
Pretty Raw Preview Visualize JSON ↕
1
2 {
3   "albumId": 1,
4   "id": 1,
5   "title": "accusamus beatae ad facilis cum similique qui s
6   "url": "https://via.placeholder.com/600/92c952",
7   "thumbnailUrl": "https://via.placeholder.com/150/92c952"
8 },
9 {
10  "albumId": 1,
11  "id": 2,
12  "title": "reprehenderit est deserunt velit ipsam",
13  "url": "https://via.placeholder.com/600/771796",
14  "thumbnailUrl": "https://via.placeholder.com/150/771796"
15 },
16 {
17  "albumId": 1,
18  "id": 3,
19  "title": "officia porro iure quia iusto qui ipsa ut modi"
20  "url": "https://via.placeholder.com/600/24f355",
21  "thumbnailUrl": "https://via.placeholder.com/150/24f355"
22 },
23 {
24  "albumId": 1,
25  "id": 4,
26  "title": "culpa odio esse rerum omnis laboriosam voluptate repudiandae",
27  "url": "https://via.placeholder.com/600/d32776",
28  "thumbnailUrl": "https://via.placeholder.com/150/d32776"
29 },
30 {
31  "albumId": 1,
```

```
// Добавляем необходимые импорты, используя `Alt +`
import com.google.gson.annotations.SerializedName
```

```
// Класс `Photo` представляет собой модель данных
// соответствующие полям в JSON-структуре.
```

```
data class Photo(
    @SerializedName("albumId")
    val albumId: Int,
    @SerializedName("id")
    val id: Int,
    @SerializedName("title")
    val title: String,
    @SerializedName("url")
    val url: String,
    @SerializedName("thumbnailUrl")
    val thumbnailUrl: String
)
```


Обращение к API

- Используется REST клиент для Android Retrofit для выполнения HTTP запросов к API
- Для эмулятора можно указать localhost
- Для показа IP в локальной сети, например 192.168.100.108

```
interface CodelabApi {  
  
    @GET("photos")  
    suspend fun getAllPhotos(): List<Photo>  
  
    companion object RetrofitBuilder{  
        private const val BASE_URL = "https://jsonplaceholder.typicode.com/"  
  
        private fun getRetrofit(): Retrofit {  
            return Retrofit.Builder()  
                .baseUrl(BASE_URL)  
                .addConverterFactory(GsonConverterFactory.create())  
                .build()  
        }  
        val api: CodelabApi = getRetrofit().create()  
    }  
}
```

Окно детализации и поиск

- Поиск нужно переделать на бэкенд: мобильное приложение должно работать вместе с веб-сервисом
- Также необходимо добавить второй экран с детальной информацией

```
//Добавляем поле для ввода
OutlinedTextField(
  value = filterText,
  textStyle = TextStyle(
    fontSize = 16.sp,
    lineHeight = 20.sp,
    fontWeight = FontWeight(400),
    color = Color(0xFF818C99),
    letterSpacing = 0.1.sp,
  ),
  //При изменении состояния поля (ввод символов), ищем карточки
  onChange = {
    filterText = it
    scope.launch {
      filteredPhotos = photos.filter { photo ->
        photo.title.contains(filterText)
          || photo.id.toString()
            .contains(filterText) || photo.albumId.toString()
              .contains(filterText)
      }
    }
  }
)
```

