

Лекция 13

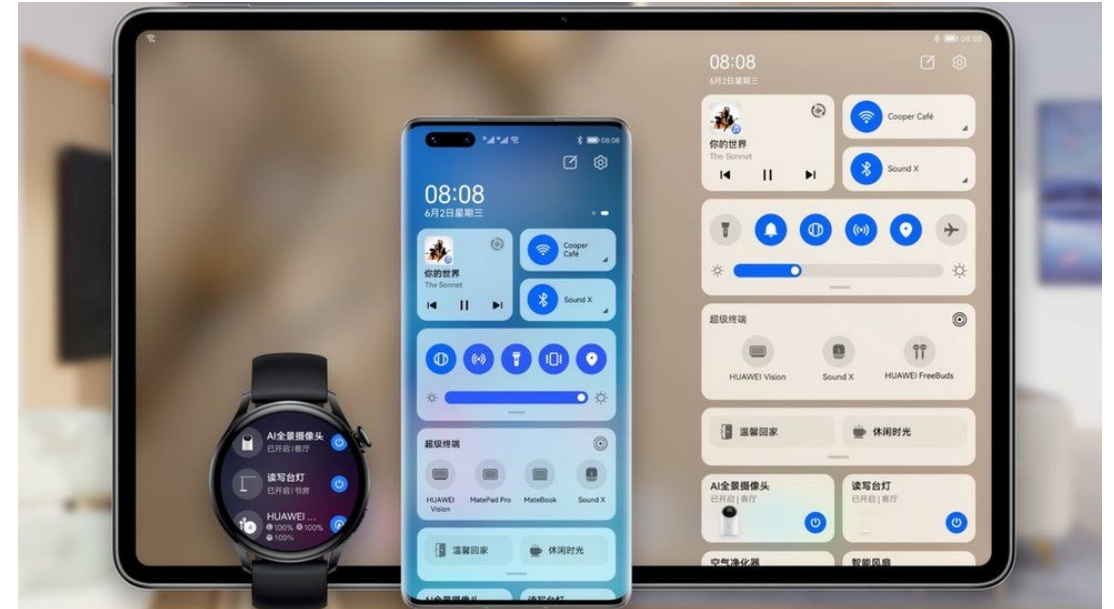
Мобильные приложения

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

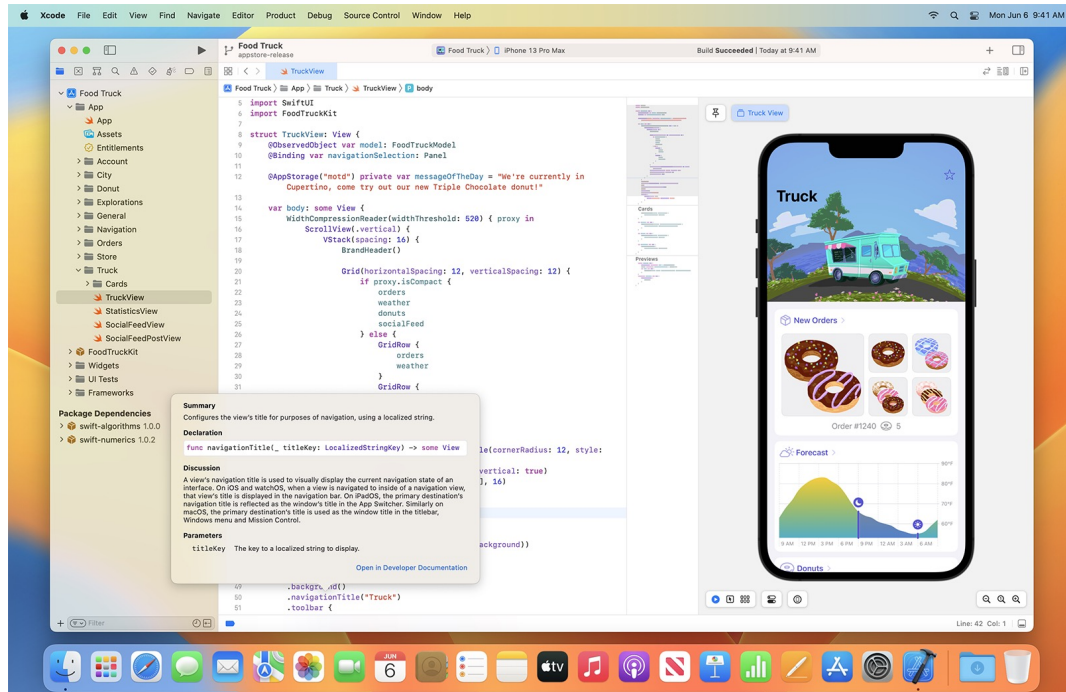


ЯЗЫКИ

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

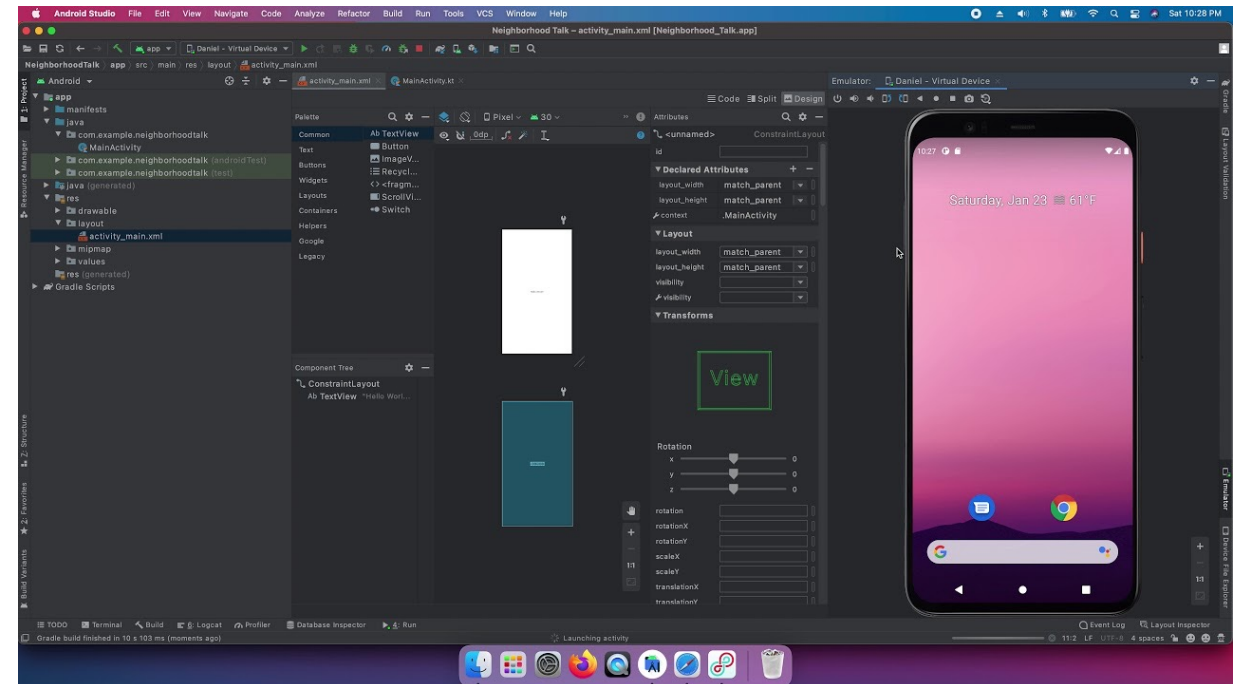


Среды разработки



- 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»»



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



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 X

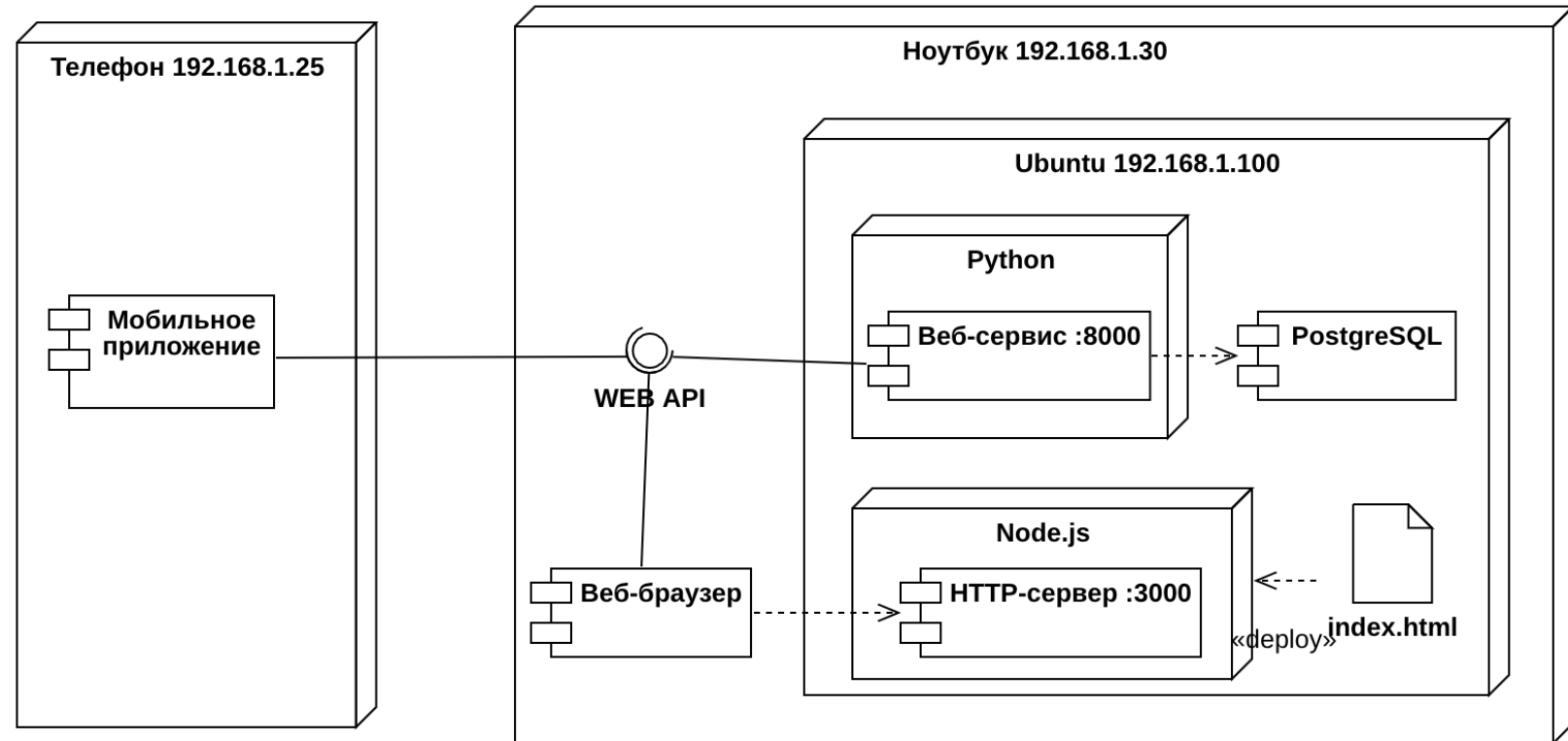
Не можете войти?

Версия 12.15.0

Вход Уведомления На карте Оплата Сменить

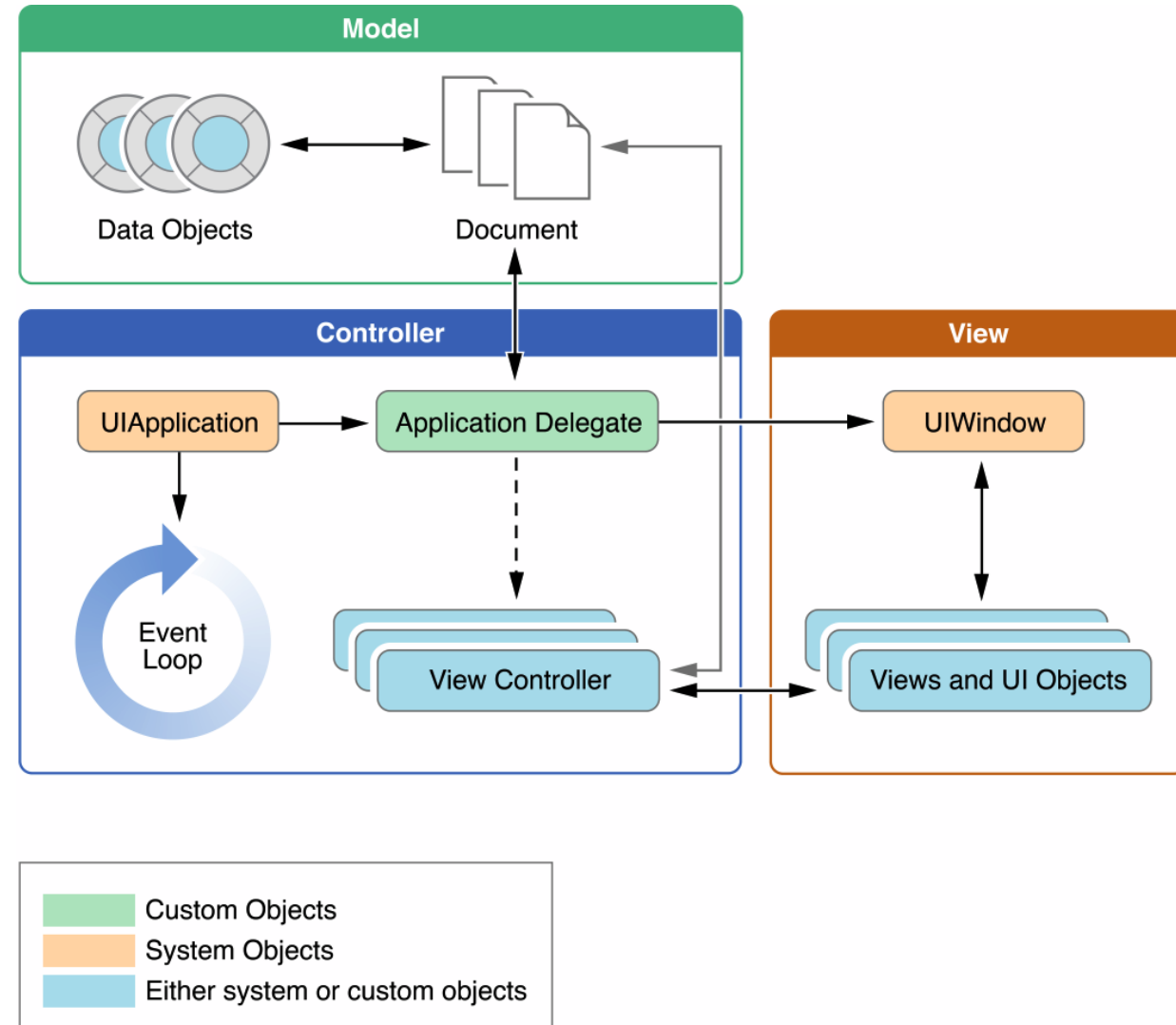
Трехзвенная архитектура. API

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



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.



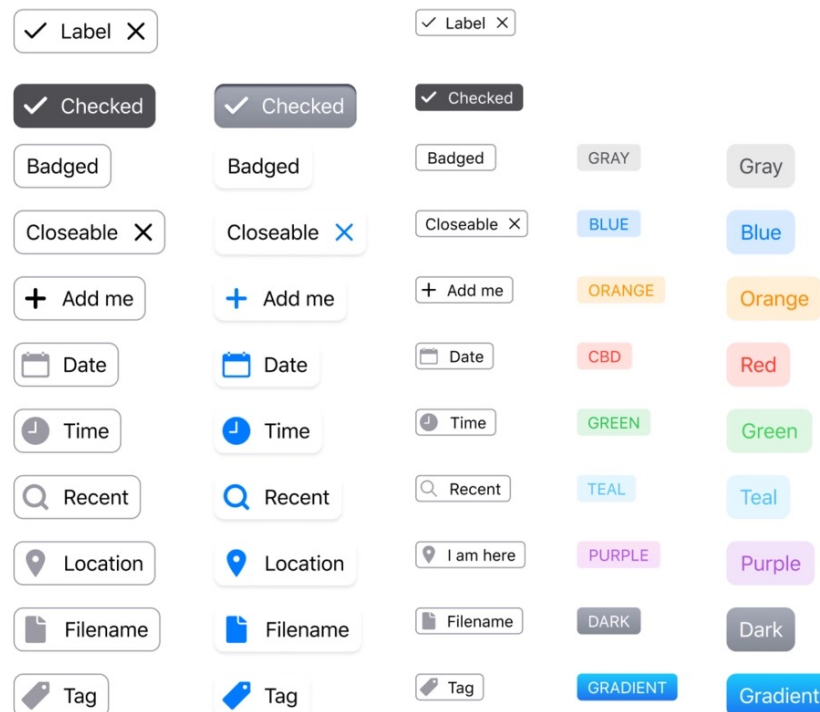
UI компоненты

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

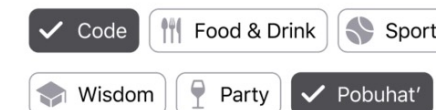
iOS components



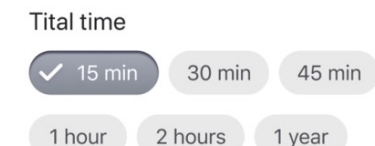
Chips



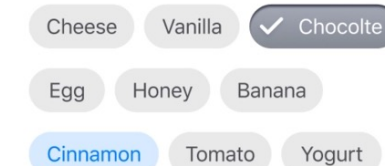
What to do?



Pick filters (2)



Ingredient

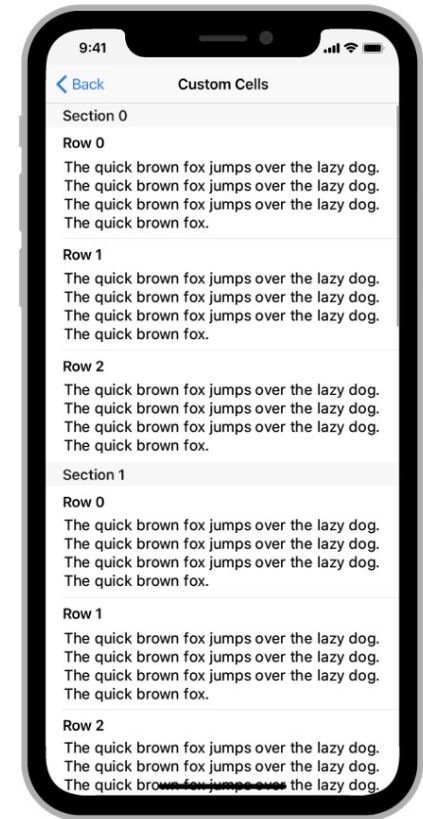
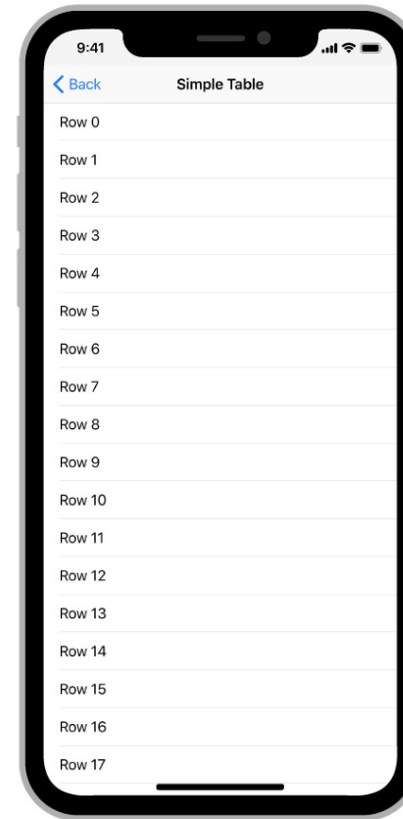
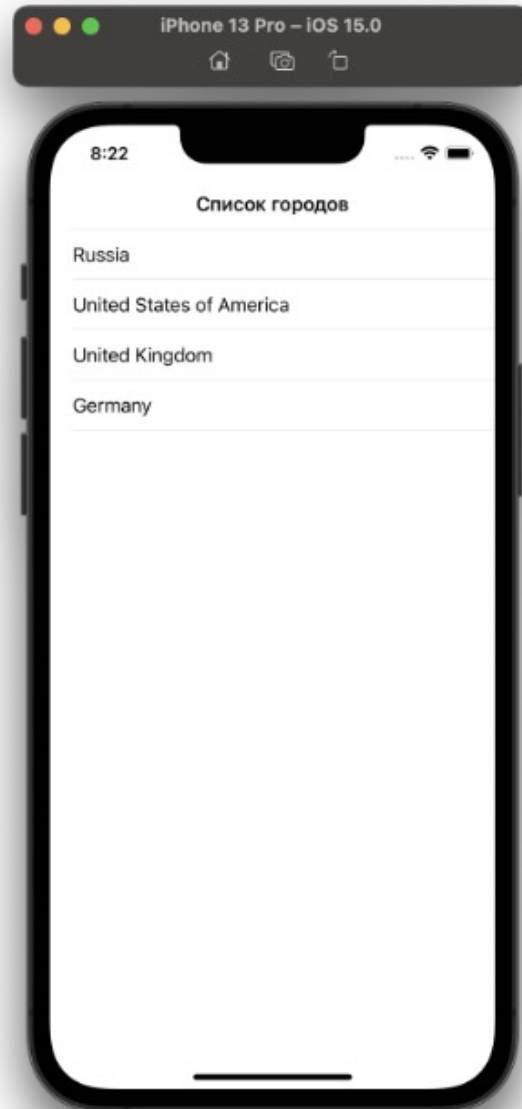


Summary keywords



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
    }
}
```

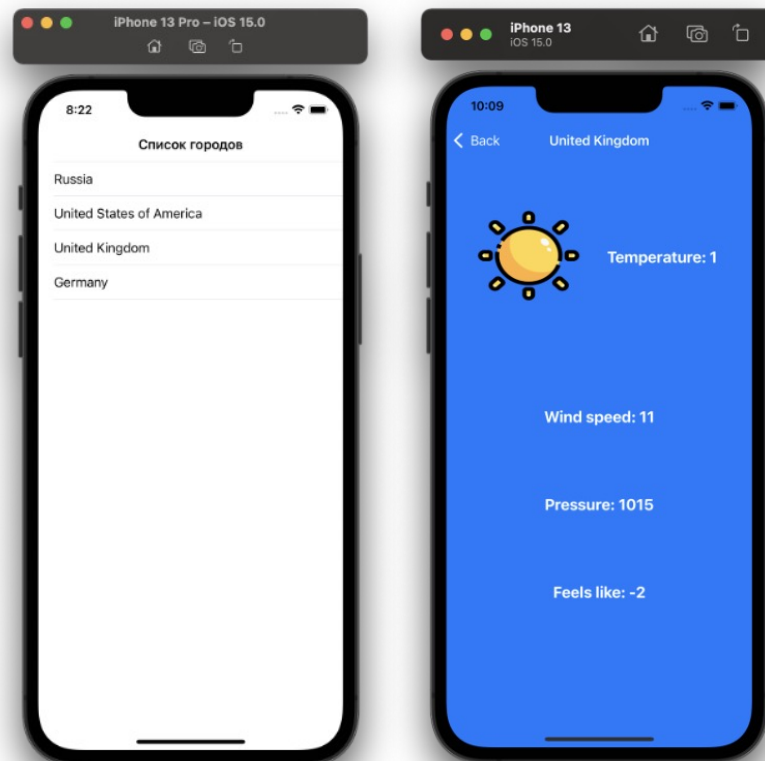
Заполнение страницы данными

```
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)
        }

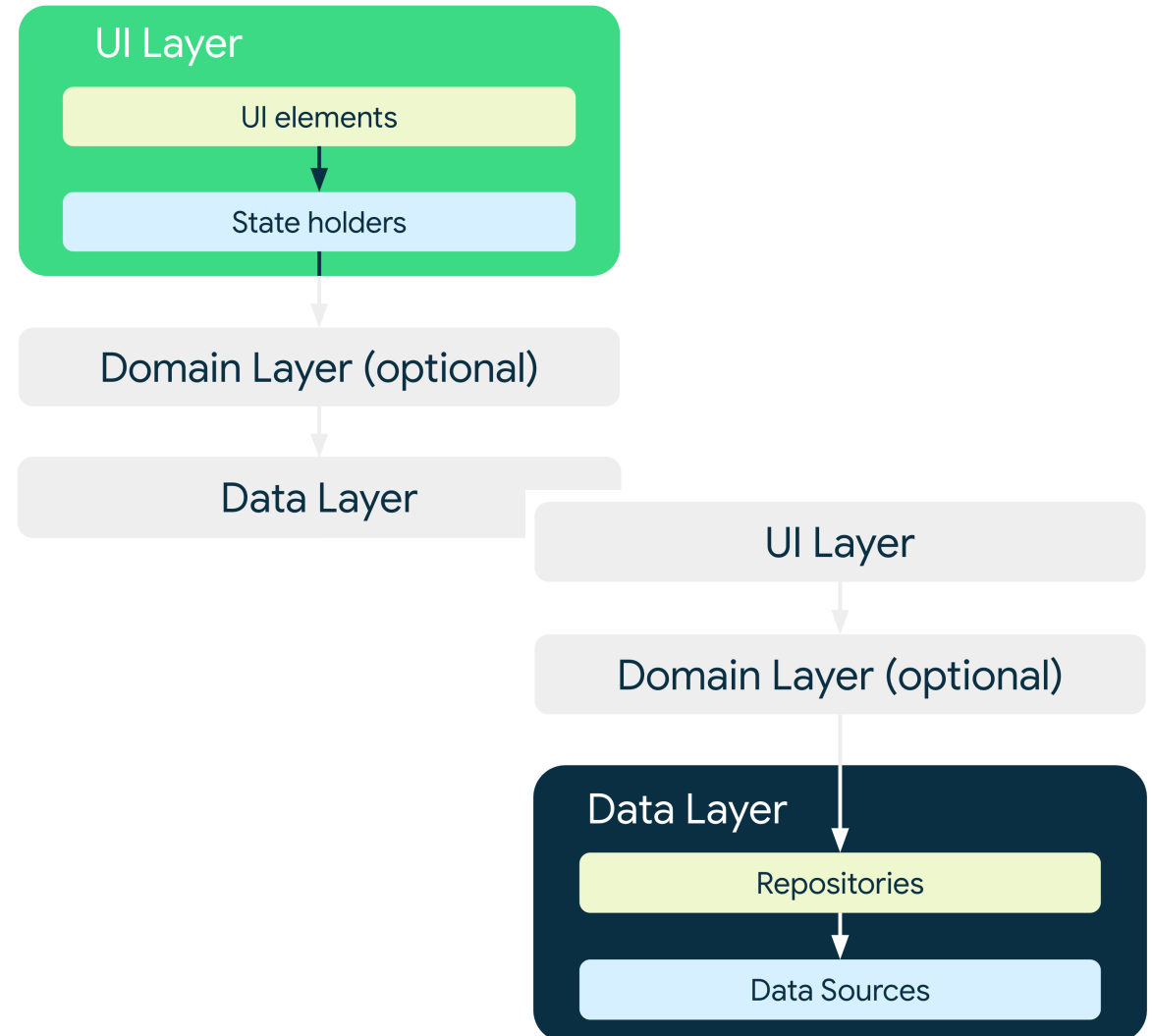
        //зададим констрейнты и базовые настройки для картинки
        imageView.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(imageView)
        imageView.heightAnchor.constraint(equalToConstant: 250).isActive = true
        imageView.widthAnchor.constraint(equalToConstant: 200).isActive = 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

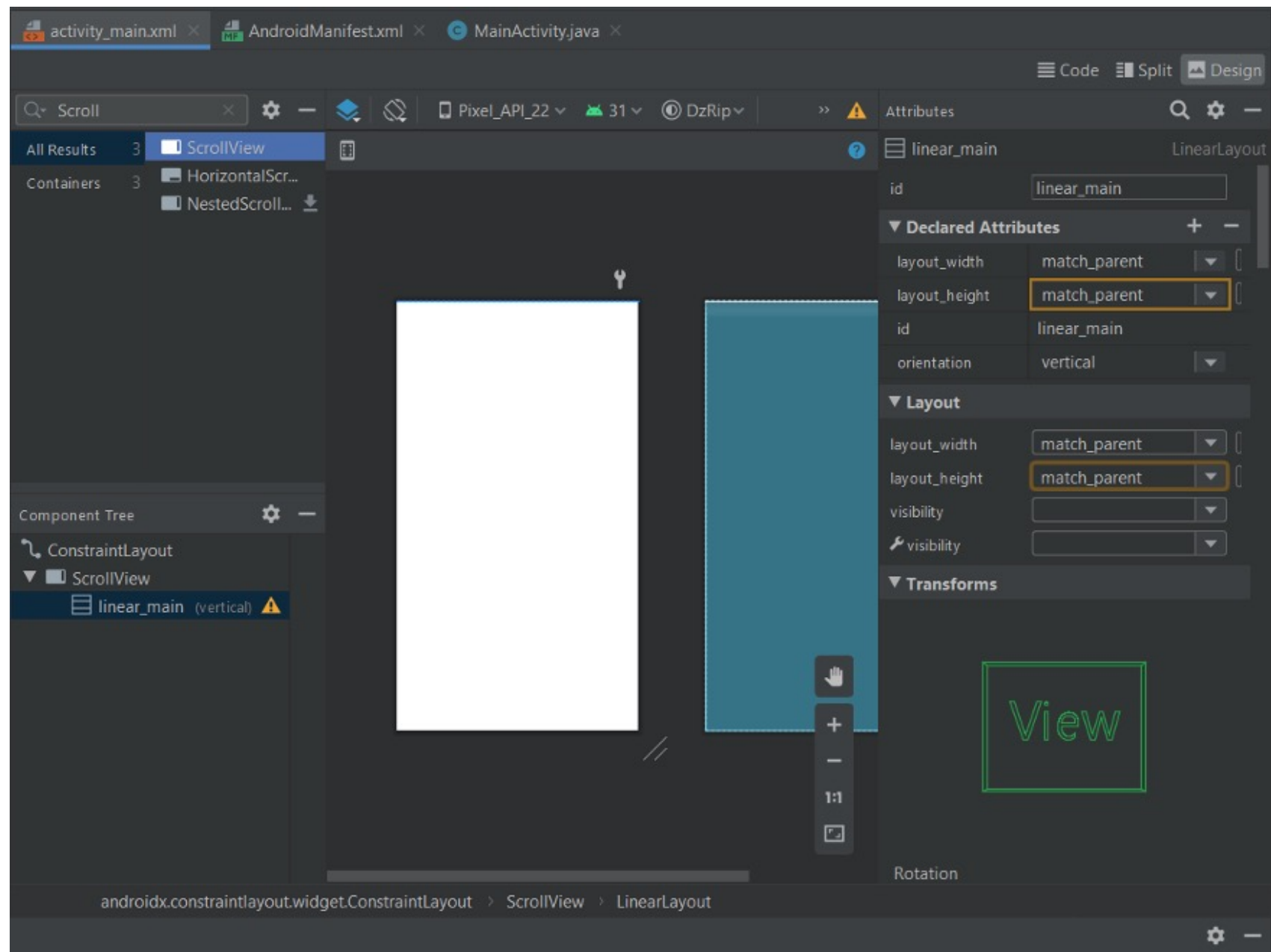
- **Роль слоя UI** (или *слоя представления*) — отображать на экране данные приложения.
- **Слой данных** в приложении содержит *бизнес-логику* — правила, по которым приложение создаёт, хранит и изменяет данные.
- **Доменный слой** располагается между слоями UI и данных. Доменный слой отвечает за инкапсуляцию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



Верстка

- Разработку проводим в Android Studio
- Описываем верстку в файле представления activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:background="#FF6200EE"
        >
        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="match_parent"
            android:layout_margin="8dp"
            android:layout_height="200dp"
            />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```



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

```
public class NetworkService {  
    private static NetworkService mInstance;  
    private static final String BASE_URL = "http://192.168.100.108:8000";  
    private Retrofit mRetrofit;  
  
    private NetworkService() {  
        mRetrofit = new Retrofit.Builder()  
            .baseUrl(BASE_URL)  
            .addConverterFactory(GsonConverterFactory.create())  
            .build();  
    }  
  
    public static NetworkService getInstance() {  
        if (mInstance == null) {  
            mInstance = new NetworkService();  
        }  
        return mInstance;  
    }  
}
```

Модель для полученных данных

- Сделаем структуру – модель для полученных от API данных

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
  {
    "pk": 6,
    "os_name": "Windows",
    "last_version": "11.00",
    "os_descript": "The most popular computer os",
    "src": "https://gold-nm.biz/files/products/otklyuchit-kombinatsii-s-klavishej-win-v-windows-10.800x600.jpg"
  },
  {
    "pk": 7,
    "os_name": "Android",
    "last_version": "12.00",
    "os_descript": "The most popular mobile os",
    "src": "https://itc.ua/wp-content/uploads/2020/04/android_logo_stacked__rgb_.5.jpg"
  },
  {
    "pk": 8,
    "os_name": "IOS",
    "last_version": "14.00",
    "os_descript": "Mobile OS for Apple devices",
    "src": "https://upload.wikimedia.org/wikipedia/commons/thumb/c/ca/IOS_logo.svg/300px-IOS_logo.svg.png"
  }
]
```

```
public class Post {
    @SerializedName("pk")
    @Expose
    private int pk;
    @SerializedName("os_name")
    @Expose
    private String osName;
    @SerializedName("last_version")
    @Expose
    private float lastVersion;
    @SerializedName("os_descript")
    @Expose
    private String osDescript;
    @SerializedName("src")
    @Expose
    private String src;

    public int getPk() {
        return pk;
    }

    public String getOsName() {
        return osName;
    }

    public float getLastVersion() {
        return lastVersion;
    }

    public String getOsDescript() {
        return osDescript;
    }

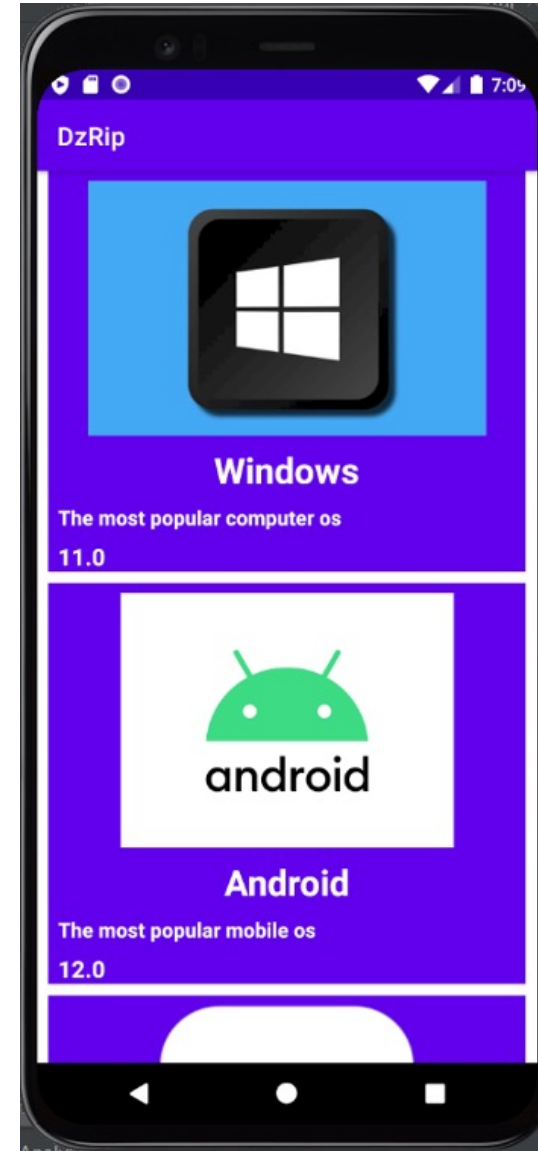
    public String getSrc() {
        return src;
    }
}
```

Окно списка услуг

- Создаем карточки из полученных от API данных
- Помещаем данные из модели в компоненты нашего экрана

```
public void addCardView(Post post, int i) {  
    final View view = getLayoutInflater().inflate(R.layout.item_view, null);  
    LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(  
        LinearLayout.LayoutParams.MATCH_PARENT,  
        LinearLayout.LayoutParams.WRAP_CONTENT  
    );  
    params.setMargins(24, 0, 24, 24);  
  
    TextView title = view.findViewById(R.id.title);  
    TextView descr = view.findViewById(R.id.descr);  
    TextView version = view.findViewById(R.id.version);  
    ImageView image = view.findViewById(R.id.imageView2);  
    Glide.with(this).load(post.getSrc()).into(image);  
    title.setText(post.getOsName());  
    descr.setText(post.getOsDescript());  
    version.setText(post.getLastVersion() + "");  
    view.setLayoutParams(params);  
}
```

```
NetworkService.getInstance()  
    .getJSONApi()  
    .getAllPosts()  
    .enqueue(new Callback<List<Post>>() {  
        @Override  
        public void onResponse(@NonNull Call<List<Post>> call, @NonNull Response<List<Post>> response)  
            List<Post> postList = response.body();  
  
        for (int i = 0; i < postList.size(); i++) {  
            addCardView(postList.get(i), i);  
        }  
    })
```



Окно детализации услуги

- Добавим второй экран с детальной информацией
- В методичке вы передаете данные из основного экрана
- В лабораторной вы обращаетесь ко второму методу сервиса

```
public class DetailedActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_detailed);  
        //Инициализация компонентов активити  
        TextView title=findViewById(R.id.title);  
        TextView description=findViewById(R.id.description);  
        TextView version=findViewById(R.id.version);  
        ImageView image=findViewById(R.id.imageView);  
        //Получение объекта Bundle и проверка получения.  
        Bundle bundle =getIntent().getExtras();  
        if(bundle!=null)  
        { //Заполнение компонентов активити из Bundle.  
            Glide.with(this).load(bundle.getString("src")).into(image);  
            title.setText(bundle.getString("os_name"));  
            description.setText(bundle.getString("os_descript"));  
            version.setText("Версия: "+bundle.getString("last_version"));  
        }  
    }  
}
```

