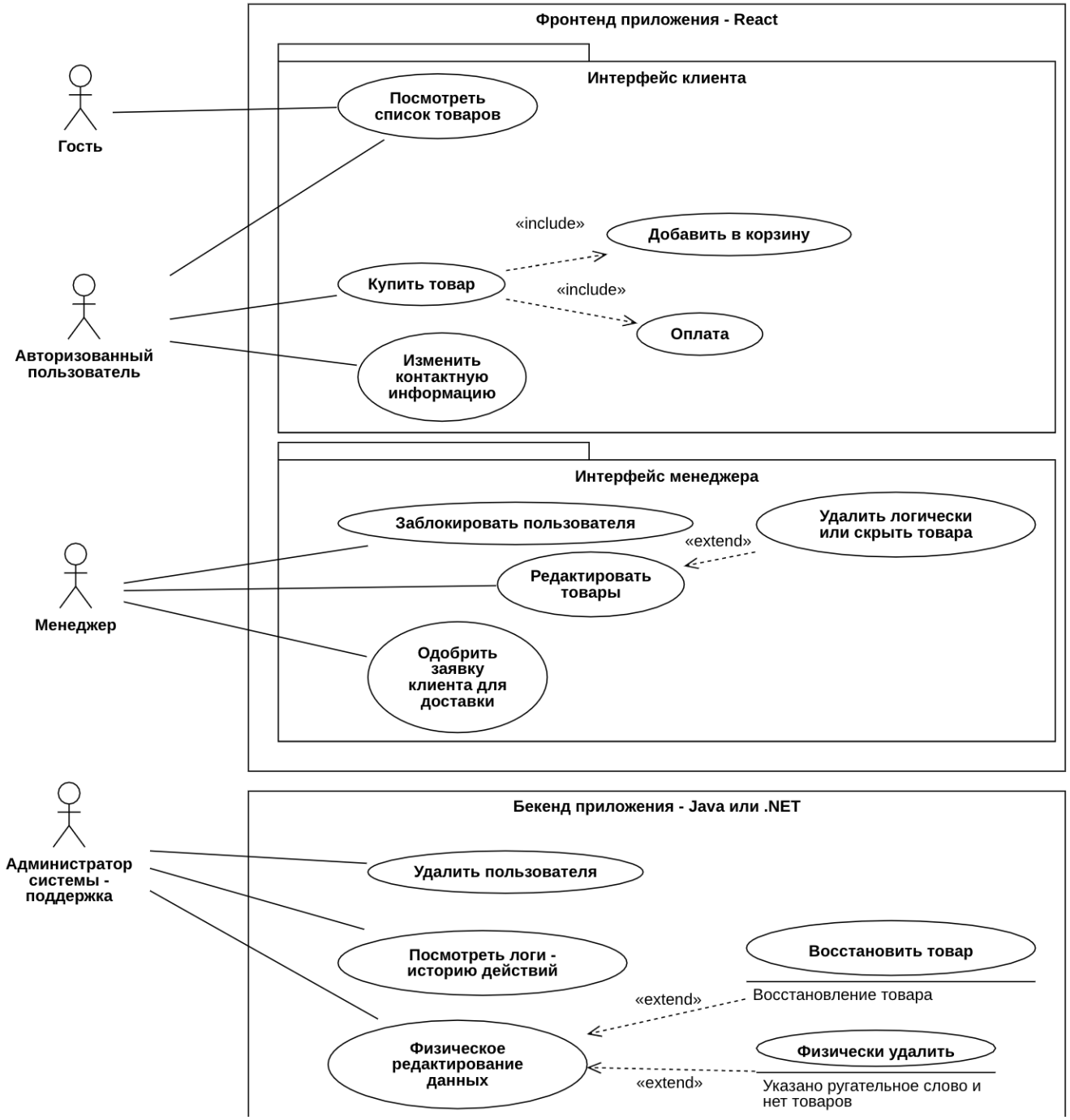
Лекция 13
Мобильные приложения

Разработка интернет приложений

Канев Антон Игоревич

Прецеденты

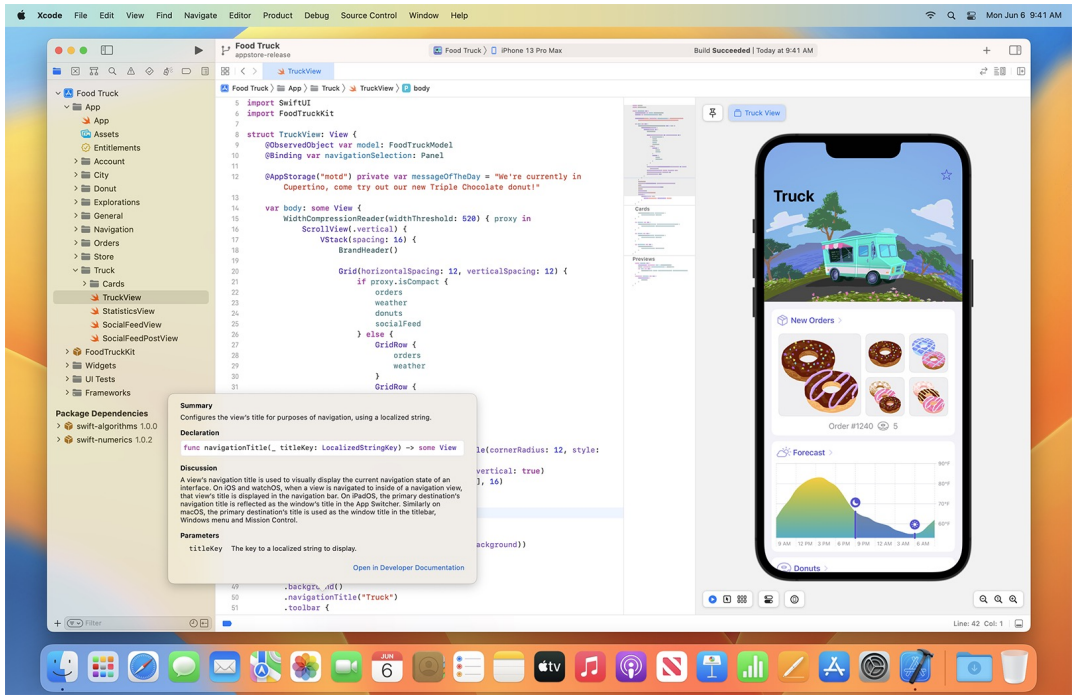
- Можем указать группы пользователей и функции в системе
- На диаграмме можно указать несколько интерфейсов пользователя



Операционные системы

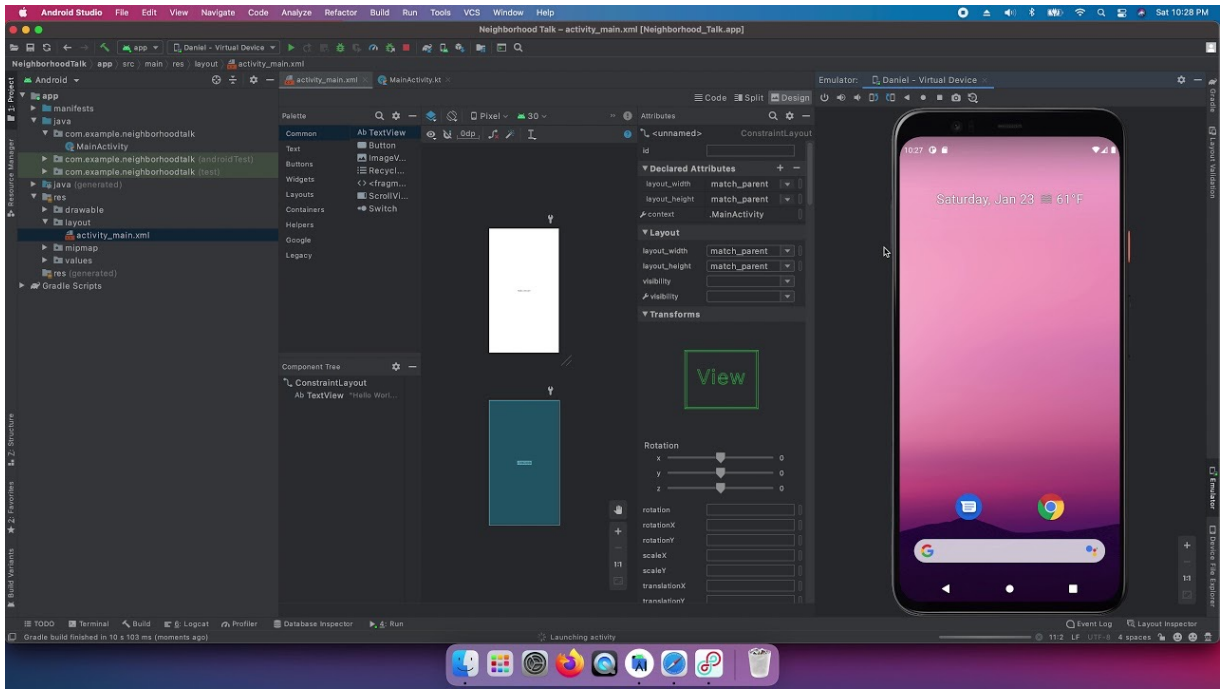
- **Мобильное приложение** («Mobile application») — программное изделие, разновидность прикладного программного обеспечения, предназначенная для работы на смартфонах, планшетах и других мобильных (портативных, переносных, карманных) устройствах

Среды разработки



- Xcode

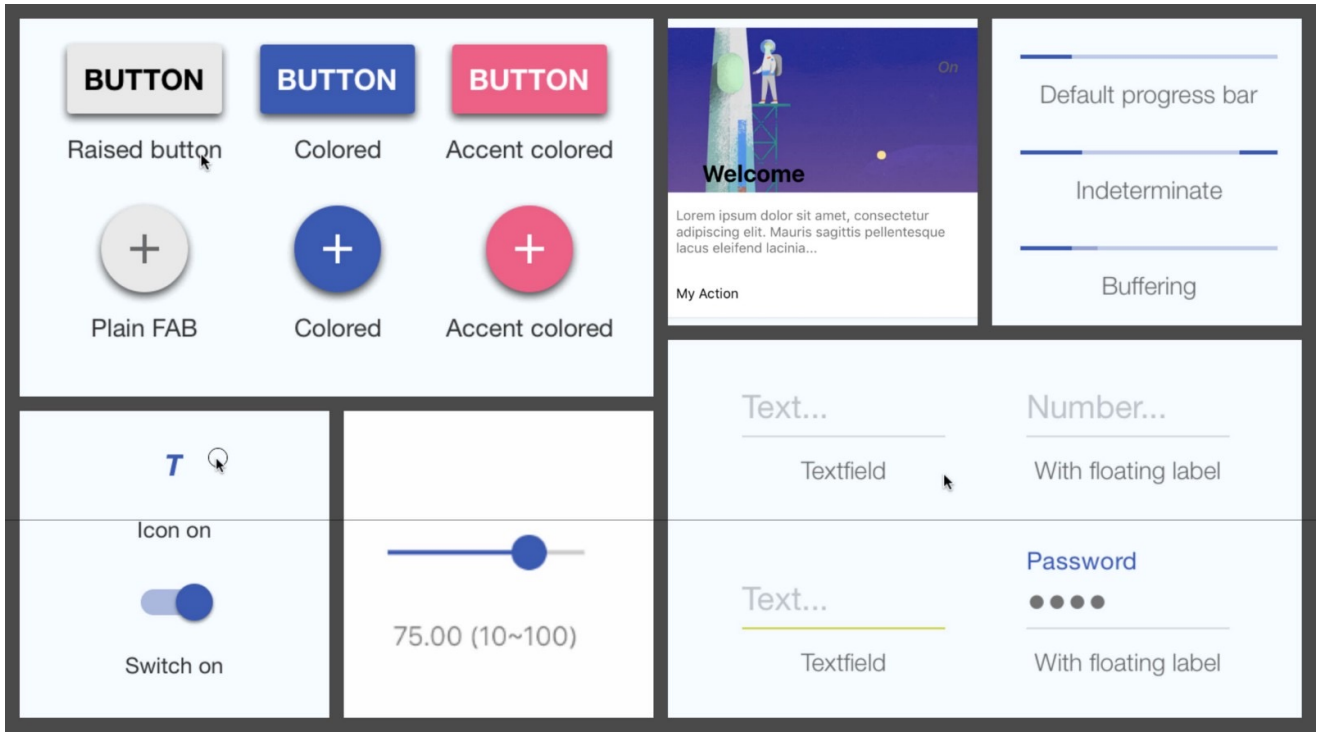
- Android Studio

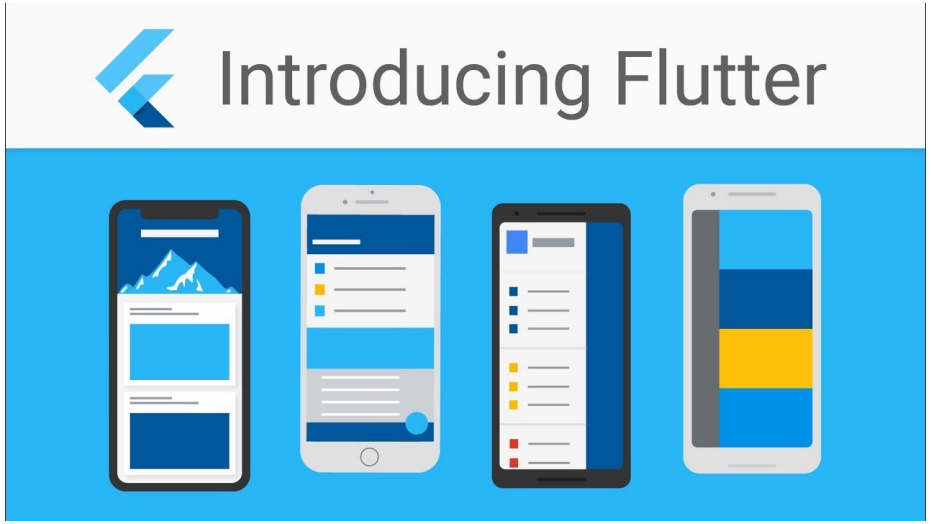


Языки

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

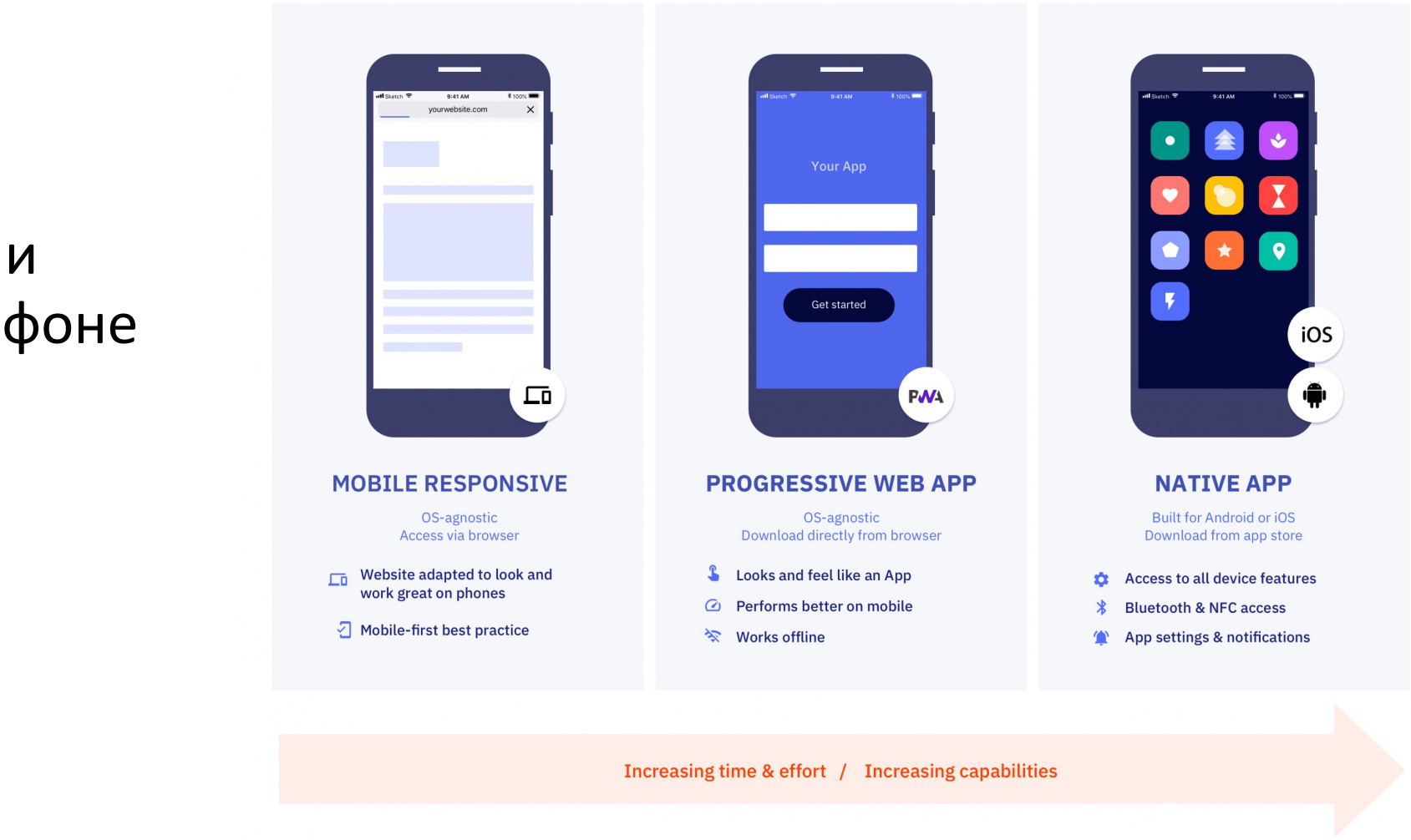
Flutter, React Native





PWA

- Выглядит как приложение
- Работает лучше и быстрее на телефоне
- Работает offline



systemseed.com

manifest.json

```
{
  "name": "Tile Notes",
  "short_name": "Tile Notes",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#fdfdfd",
  "theme_color": "#db4938",
  "orientation": "portrait-primary",
  "icons": [
    {
      "src": "/logo192.png",
      "type": "image/png", "sizes": "192x192"
    },
    {
      "src": "/logo512.png",
      "type": "image/png", "sizes": "512x512"
    }
  ]
}
```

Service Worker

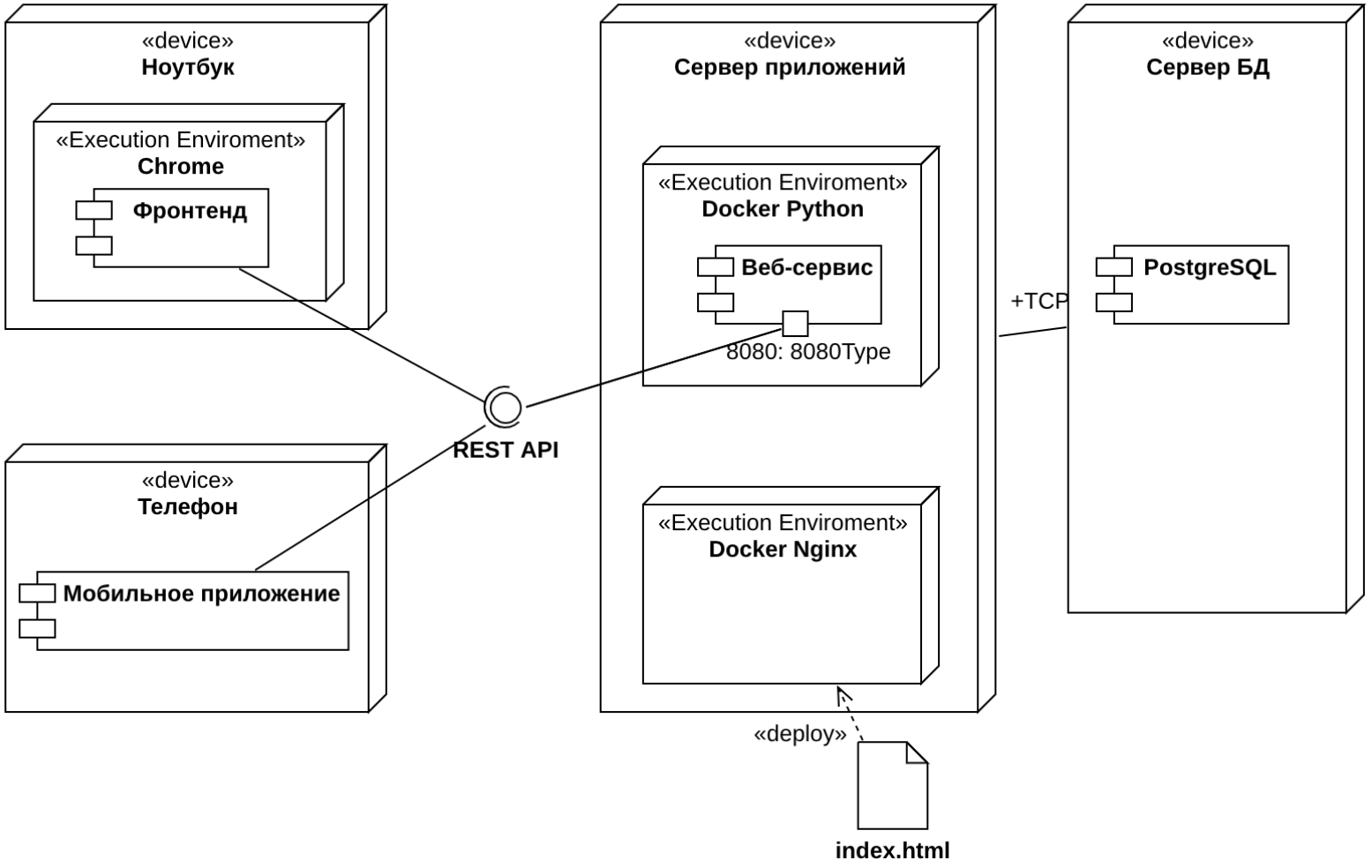
Регистрируем service worker, делаем это в файле `index.js` после рендера корневого компонента:

```
if ("serviceWorker" in navigator) {
  window.addEventListener("load", function() {
    navigator.serviceWorker
      .register("/serviceWorker.js")
      .then(res => console.log("service worker registered"))
      .catch(err => console.log("service worker not registered", err))
  })
}
```

Создаем файл serviceWorker.js и кладем его в директорию public:

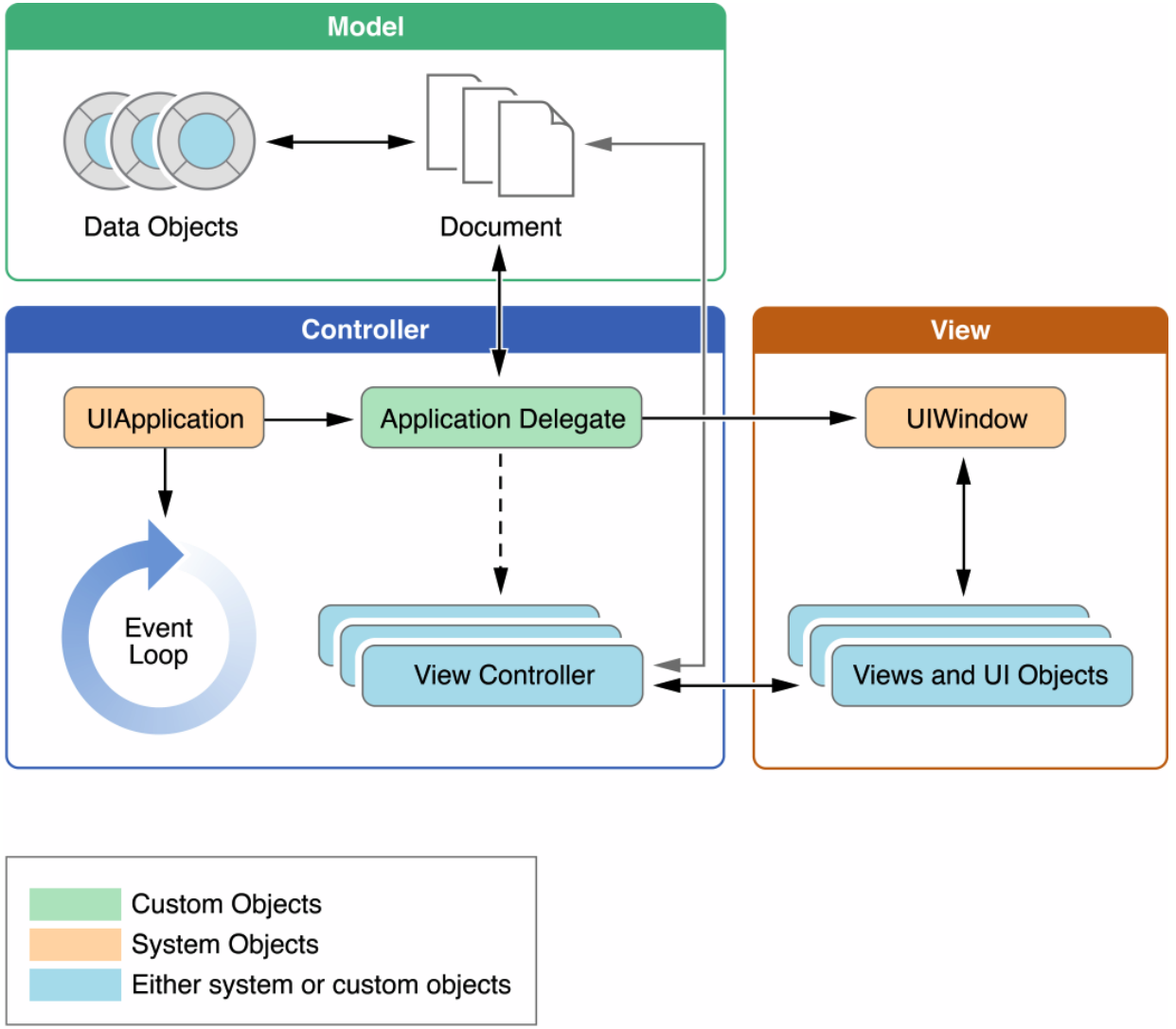
```
self.addEventListener('fetch',() => console.log("fetch"));
```

Трехзвенная архитектура. API



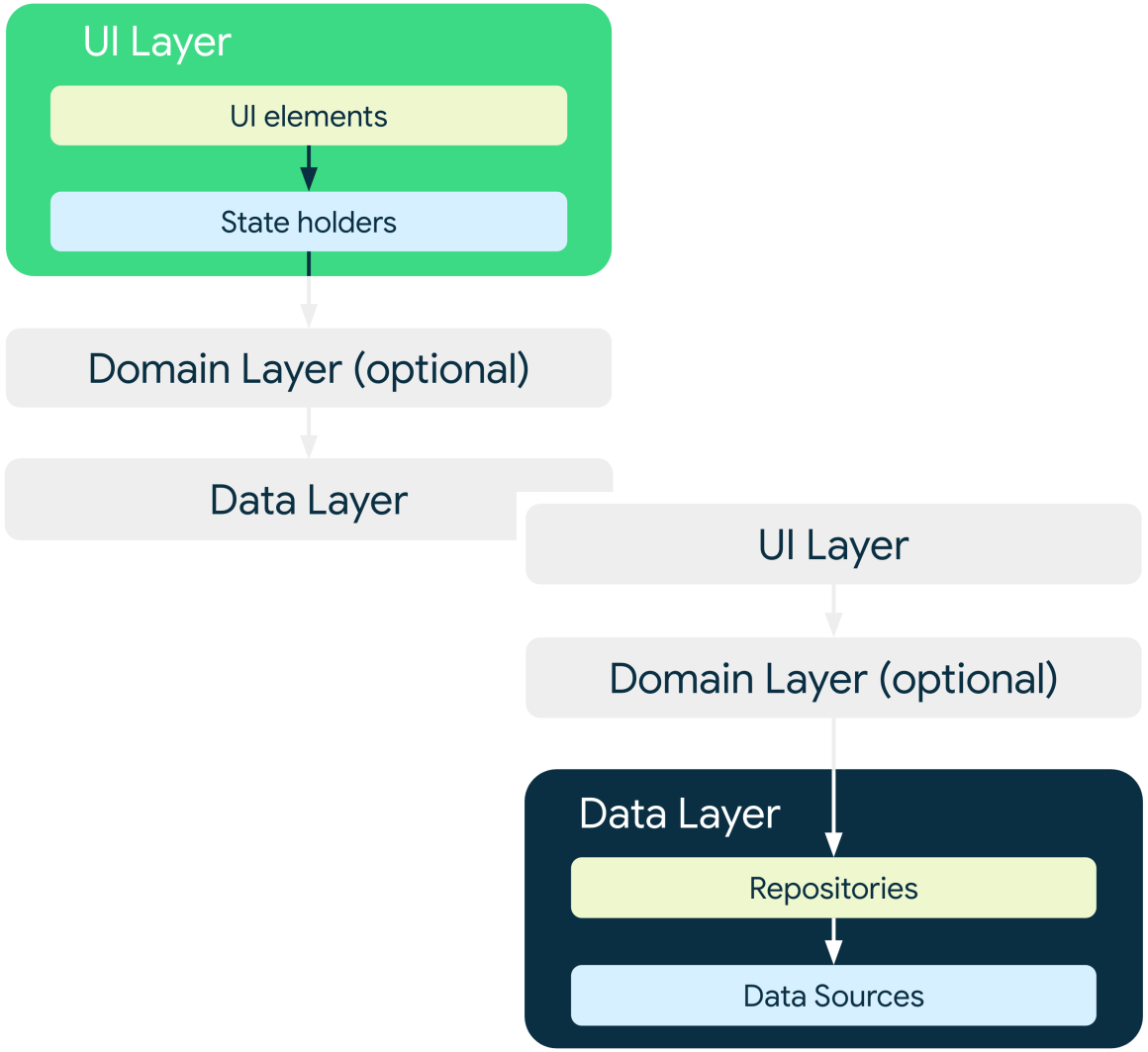
Архитектура Swift приложения

- **Модель** – отвечает за использование предметных (бизнес, domain) данных. Активные модели умеют уведомлять окружающих об изменениях в себе, а пассивные — нет.
- **Вид** (Представление, View) – отвечает за слой представления (GUI). Вид не обязательно должен быть связан с UI отрисовкой. Помимо представления пользователю данных, у Вида 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 вид, а события вида – в модель, соответствующим образом их преобразуя и обрабатывая.

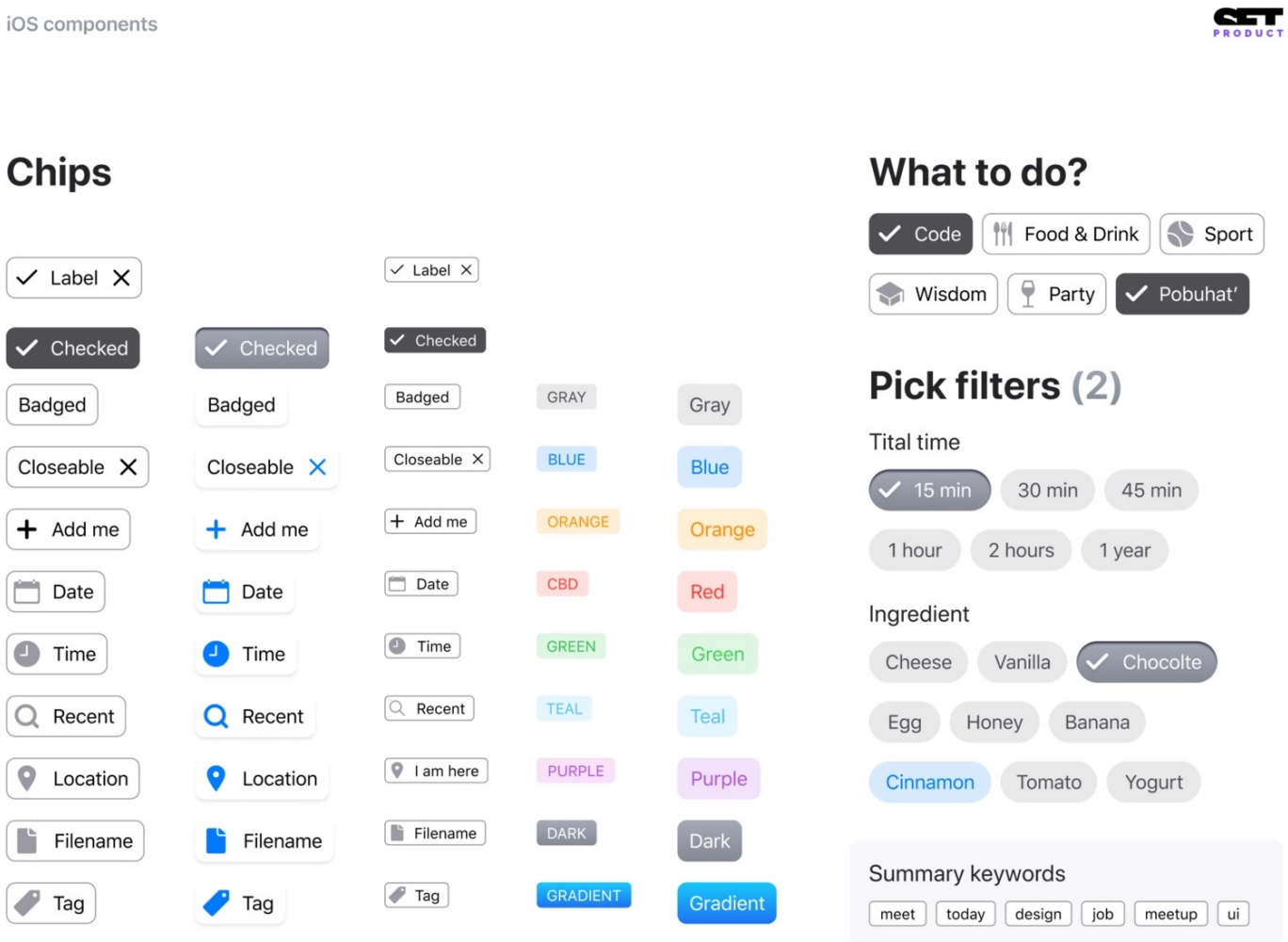


Архитектура Android приложения

- **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.



UI компоненты



UITableView

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