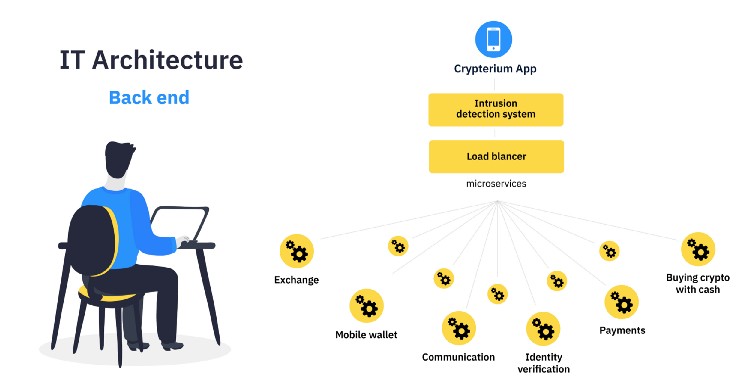
[Вы забыли свой пароль?](https://proglib.io/p/app-architecture-mom/)

Архитектура приложения

Front-end и Back-bend

Давайте в объяснении того, что есть архитектура приложения, отойдем от технических терминов и проведем аналогию с повседневной жизнью. Посмотрите на свое тело. Все, что находится снаружи, − голова и тело, − это front, а всё, что внутри, − сердце, мозг и внутренние органы, − back.

Ключевые принципы разработки Agile-приложения



Теперь, когда мы разобрались с различием front и back частей, давайте рассмотрим два ключевых подхода, которые используют современные разработчики: API First и Loose Coupling. Они позволяют программистам легко менять структуру приложения. Более того, они делают так, что каждая отдельная часть приложения может быть изменена без затрагивания остальных частей.

Метод API First отвечает за высокую скорость работы и нововведения. Идея в том, чтобы ввести данные и получить в ответ API, необходимый для Front-end и Back-end команд разработки: это позволяет им одновременно писать код и параллельно тестировать его. Преимущества метода заключаются в снижении издержек на разработку, увеличении скорости и снижении рисков.

Пример из жизни: когда вы готовите пасту Болоньезе, вам не нужна сначала паста, потом соус: вы можете готовить их параллельно. В таком случае, еда приготовится быстрее, ничего не успеет остыть, а друзья смогут оценить блюдо в том состоянии, в котором оно и должно быть (а не как обычно).

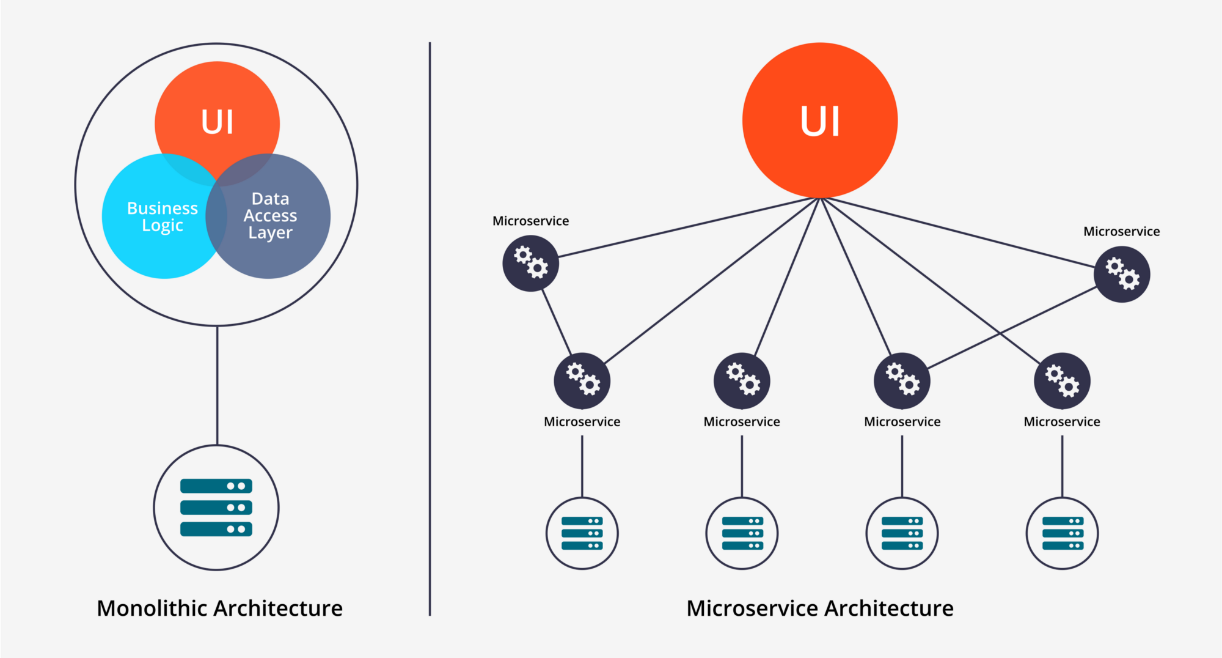


Одна из функций, за которую команда приложения любит подход API First, называется Swagger − это open-source фреймворк, который помогает разработчикам строить архитектуру, проектировать и создавать документацию для своих приложений. Swagger автоматически генерирует описание API для большинства языков и фреймворков, для обеих − Front-end и Back-end − команд.

Следующий подход называется Loose Coupling, в дословном переводе − слабая связь. И если в жизни примером Loose Coupling может быть отмена свидания в День святого Валентина, то в программировании это наоборот помогает. Если быть точнее, то эта функция упрощает соединение компонентов в сети.

Система Loose Coupling уменьшает риск случайного изменения отдельных объектов, без изменения других − так как в приложении всё взаимосвязано, это может привести к поломкам и уязвимостям. Так вот, благодаря возможности ограничения работы отдельных соединений, система помогает найти и решить проблему быстрее, прямо во время тестирования.

Микросервисы против монолита



Благодаря принципам API First и Loose Coupling, приложение может выступать микросервисом − приложением, состоящем из независимых служб, способных работать самостоятельно, свободно масштабироваться на разных устройствах.

Микросервисные архитектуры лучше организованы, так как у каждого микросервиса есть определенная задача. Их преимущество ещё и в легкой реконфигурации и перестройке для различных целей. Кроме того, они характеризуются быстрым развертыванием, отказоустойчивостью, горизонтальным масштабированием, низким порогом входа и простотой управления.

Представьте себе умный дом, где все можно контролировать и управлять с помощью одного устройства. Допустим, это устройство − \* core \*, а управляемыми элементами являются \* services \*. С помощью основного устройства вы можете открывать окна, включать телевизор или даже закрывать шторы. Так работает архитектура микросервисов.

Но всегда есть альтернативный вариант, верно? Второй тип архитектуры − монолитная архитектура. Это означает, что приложение написано как одна единица кода, чьи компоненты предназначены для совместной работы, используют одни и те же ресурсы и место на диске. Службы в таких приложениях тесно связаны, и при изменении одной из них проблемы могут возникнуть у остальных.

Представьте себе многослойный шоколадный торт. Каждый новый слой делает торт ещё вкуснее, но вы не можете добавить слой с клубникой в середину, не изменив вкус и структуру торта. Можно считать, что у торта − монолитная архитектура.

.NET Core против JVM-платформ

Мультифункциональные приложения, например, мобильные кошельки, обычно связаны ещё с сотнями различных служб. Чтобы структурировать работу приложения, в Crypterium разделили команду Back-end разработчиков на две. Одна работает только над ядром продукта, вторая − над всем остальным, то есть авторизацией, коммуникацией и так далее.

Каждая команда использует собственные фреймворки. Основная выбрала .NET Core − платформу, которая характеризуется быстрой разработкой, отладкой и тестированием. Вдобавок, она высокопроизводительна, подходит для работы с кросс-платформенными приложениями и ориентирована на микросервисы. В то же время, остальные сервисы разрабатываются с помощью JVM-фреймворка, который, кстати, является прямым конкурентом продукту от Oracle.

Использование сразу двух популярных фреймворков позволяет выбирать из большего количества специалистов на рынке. Для .NET мы используем языки C, а для JVM − Kotlin и Java. Кроме того, эти же языки используются Android-разработчиками.

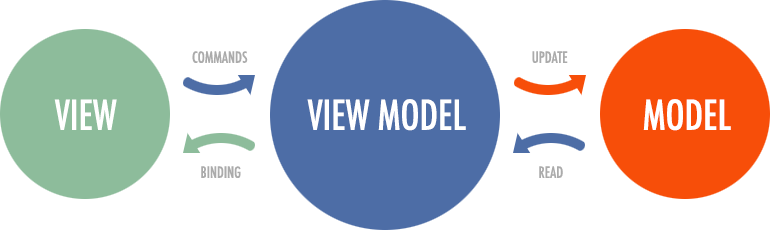
Функции Front-end команды

Команда Front-end специалистов следит за тем, чтобы приложение было удобным, а интерфейс − интуитивно-понятным и быстрым.

Android-версия приложения Crypterium основана на языках Java и Kotlin (как и среда JVM), а приложение iOS − на новом, простом в использовании языке программирования Swift. Функции языка включают в себя контроль доступа, управление памятью, отладку, цепочку вызовов и протокол-ориентированное программирование.

MVVM и роутинг для iOS

Команда разработчиков Crypterium для iOS, выбрала стиль архитектуры MVVM и роутинг. Благодаря структуре, архитектуры удобны и для разработчиков, и для пользователей.



MVVM − это Model-View-ViewModel, где Model означает информацию о продукте, а View показывает, как клиенты видят продукт. В MVVM есть структура слоев: первый уровень − UI (пользовательский интерфейс). Другие уровни содержат сетевые и логические сервисы. Роутинг отвечает за технические процессы − действия пользователей, перемещения внутри приложения, регулируются именно им.

Давайте разберем пример, когда пользователь хочет отправить криптовалюту на другой адрес. Слой сетевых сервисов содержит информацию о количестве отправленных монет данных и адресе. Когда пользователь подтверждает транзакцию, следующий слой проверяет, достаточно ли монет для отправки на счету, и предоставляет положительный или отрицательный ответ.

Чистая архитектура для Android

Чтобы повысить простоту обслуживания и гибкость приложений, команда Android решила использовать метод под названием «Чистая архитектура». Он гарантирует отсутствие ненужных связей и делает приложение более тестируемым.

Результатом является чистое, новое, свежее, простое в использовании приложение для Android с четырьмя уровнями:

* веб, базы данных, пользовательский интерфейс;
* шлюзы, презентаторы;
* варианты использования;
* юридическая информация.

Заключение

Архитектура приложений − очень сложная тема, и все, что написано выше, является лишь верхушкой айсберга.

Если вам понравился материал о том, что такое архитектура приложения, посмотрите следующее:

* [Как стать архитектором ПО](https://proglib.io/p/become-software-architect/)
* [Что такое микросервисная архитектура и когда ее применять](https://proglib.io/p/microservices/)
* [3 лучших книги по архитектуре программного обеспечения](https://proglib.io/p/computer-architecture-books/)