

Base Application infrastructure

Матвей Попов

План

- Из каких компонент состоит приложение?
- Аутентификация и авторизация
- Хранение данных
- Как организовывать архитектуру приложения?
- Как тестировать приложение?

**Из чего же состоит наше
приложение?**

- Framework(Ktor)
- Зависимости
- BuildManagementTool(Gradle)
- Наш код

Как наш код выглядит?

World

IN(http)

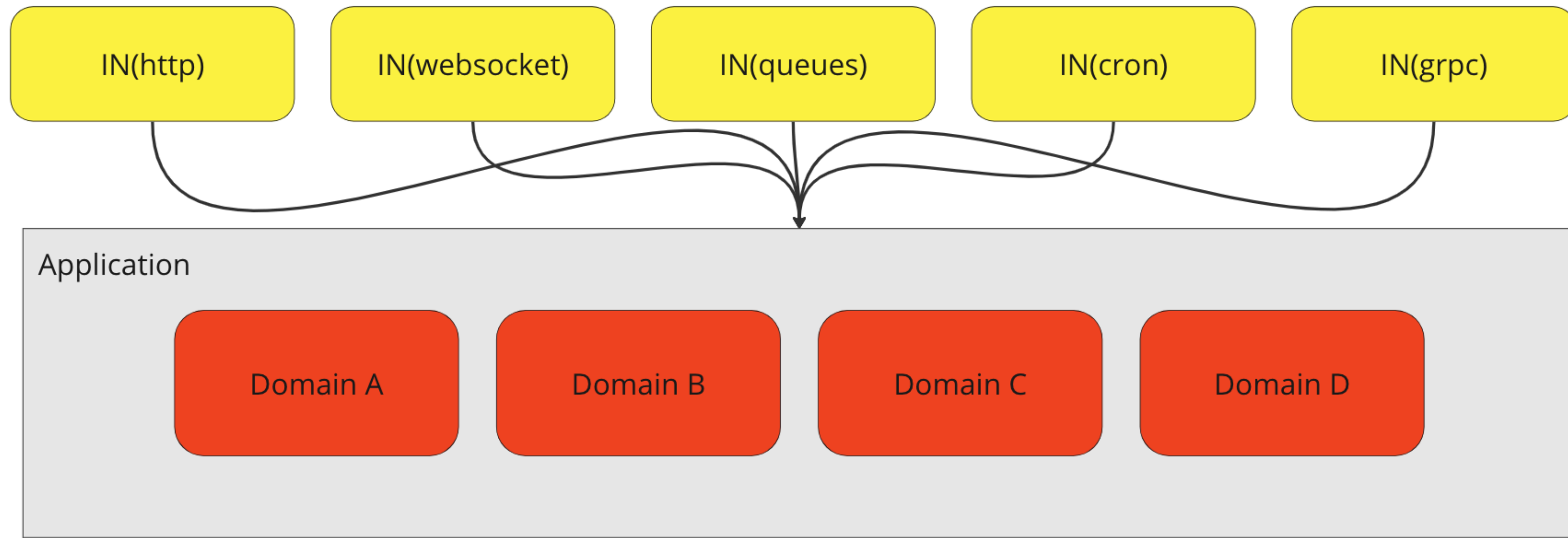
IN(websocket)

IN(queues)

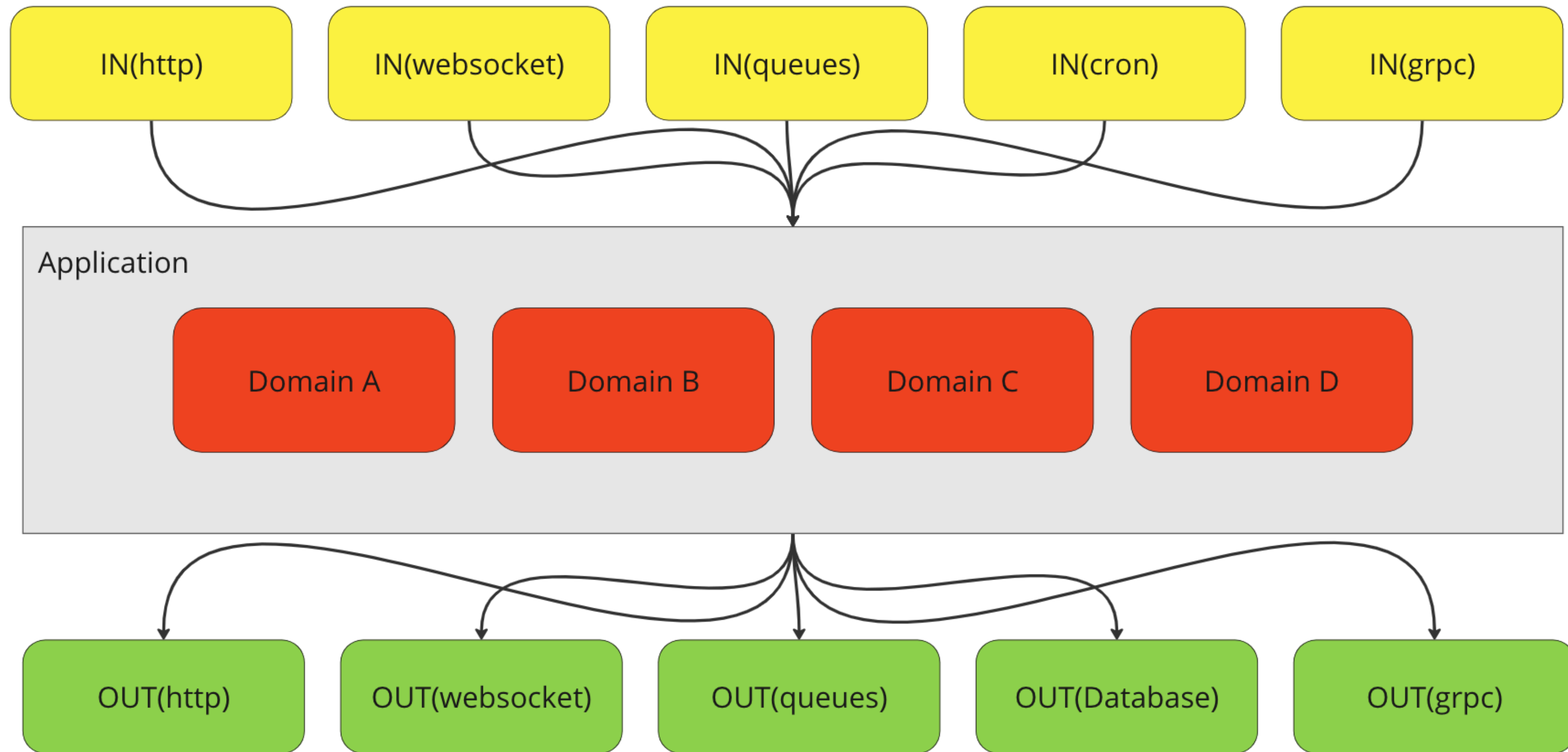
IN(cron)

IN(grpc)

World



World



Что мы всегда делаем?

1. Принимаем данные
2. Манипулируем с данными
3. Отдаем данные

Что мы делаем с входящими данными?

- Преобразуем в необходимый формат
- Валидируем данные
- Идентифицируем источник данных
- Аутентифицируем источник данных
- Авторизуем источник данных

Форматы данных

Виды форматов

- Бинарные
- Не бинарные

- JSON
- XML
- YAML
- Protobuf
- Avro

Валидация данных

- Проверяем соответствие некоторым бизнес требованиям
- Проверяем корректность данных
- Проверяем целостность данных

Идентификация источника

**Цель - понять, кто “стучится в
наше приложение”**

Примеры

- Login
- Email
- Телефон
- Какой-то токен

**Любая сущность по которой
можно однозначно определить
клиента**

Аутентификация

**Цель - проверить клиента на
подлинность**

Необходимо проверить:

- Клиент представлен в нашей системе
- Клиент вводит валидные данные
- Клиент пытается представиться так как мы его знаем

**Существуют 3 фактора,
которые задействуются в
процессе аутентификации**

1. Знание

2. Владение

3. Свойство

Виды аутентификации

- Однофакторная
- Многофакторная

Авторизация

**Цель - наделение
пользователя определенными
правами**

Примеры

- Обычный пользователь с доступом только до своих данных
- Пользователь с доступом ко всем данным(администратор)

**Какие есть способы
реализации?**

Сессии

1. Клиент отправляет идентификационные данные
2. Сервер проверяет данные
3. Сервер создает сессию и генерирует `cookie`
4. Клиент отправляет `cookie` при каждом запросе
5. Сервер валидирует и выдает доступ
6. По истечению времени или каких-то условий `cookie` очищается

Cookie

Небольшой **фрагмент** данных, отправленный клиентом и **хранимый** на стороне клиента.

Токены

1. Клиент отправляет идентификационные данные
2. Сервер проверяет данные
3. Сервер генерирует, подписывает и “зашивает” данные для идентификации пользователя
4. Клиент сохраняет токен на своей стороне и использует его при каждом запросе
5. Сервер проверяет токен

Что мы будем использовать?

JWT - JSON Web Token

JWT

- Открытый стандарт для создания токенов, основан на формате JSON
- Поддерживает симметричную и асимметричную подпись
- Стоит из 3-х частей: `headers`, `payload`, `signature`

**Как мы будем использовать
JWT?**

С помощью Ktor Plugin Authentication

**Теперь мы должны данные
куда-то отдать**

Хранение данных

Поговорим про базы данных

**На какие виды можно
разделить все базы данных?**

- Sql
- NoSql
- NewSql

SQL

Декларативный язык программирования, применяемый для создания, модификации и управления **данным** в **реляционной** базе данных

SQL Databases

- Oracle
- MySql
- PostgreSQL
- Clickhouse*
- MariaDB
- FoundationDB
- И другие....

Основные различия

- Оптимизированы под разные действия
- Оптимизированы под разный паттерн нагрузки
- Дают разные гарантии сохранности данных
- Оптимизированы под разные типы данных
- По разному поддерживают распределённость данных

Вспомнил лучик вот и солнце:
ACID

Свойство транзакций

- A - atomically
- C - consistency
- I - isolation
- D - durability

Работает только в НЕ
распределенной системе

В распределенной всегда BASE

- Basically Available
- Soft state
- Eventual consistent

NoSql Databases

- MongoDB
- Cassandra
- Redis
- Memcached
- Zookeeper
- Vertica
- Neo4j

Отличия:

- Обычно данные не привязаны к какой-то схеме
- Нацелены на специфичные форматы данных
- Имеют бОльшую пропускную способность
- Могут быстро масштабироваться
- Отсутствует сильная согласованность

NewSql Databases

Предпосылки

- Очень большой рост данных
- Необходимо автоматически и быстро масштабироваться
- Шардирование и партиционирование из коробки
- Нужна строгая согласованность

- YDB
- Amazon Aurora
- Apache ShardingSphere

Миграции

Проблемы

- Схема данных имеет свойство меняться
- Необходимо применять изменения на нескольких окружениях
- Необходимо уметь эти изменения откатывать
- Хотелось бы мигрировать схемы без привязки к платформе

Решение - MigrationTool

Liquibase

- Поддерживает 59 баз данных
- Можно описывать миграции без привязки к конкретной платформе
- Поддерживает версионирование
- Очень просто интегрируется с Gradle

**Теперь посмотрим
непосредственно на наш код**

Архитектура и организация кода

Что такое архитектура?

Архитектура ПО

Совокупность решений об организации программной
системы

**Как нужно проектировать
систему?**

Нужно проектировать так, чтобы
максимизировать количество
объектов и **минимизировать**
количество связей между ними

Архитектура - это про баланс

Признаки **успешной** архитектуры

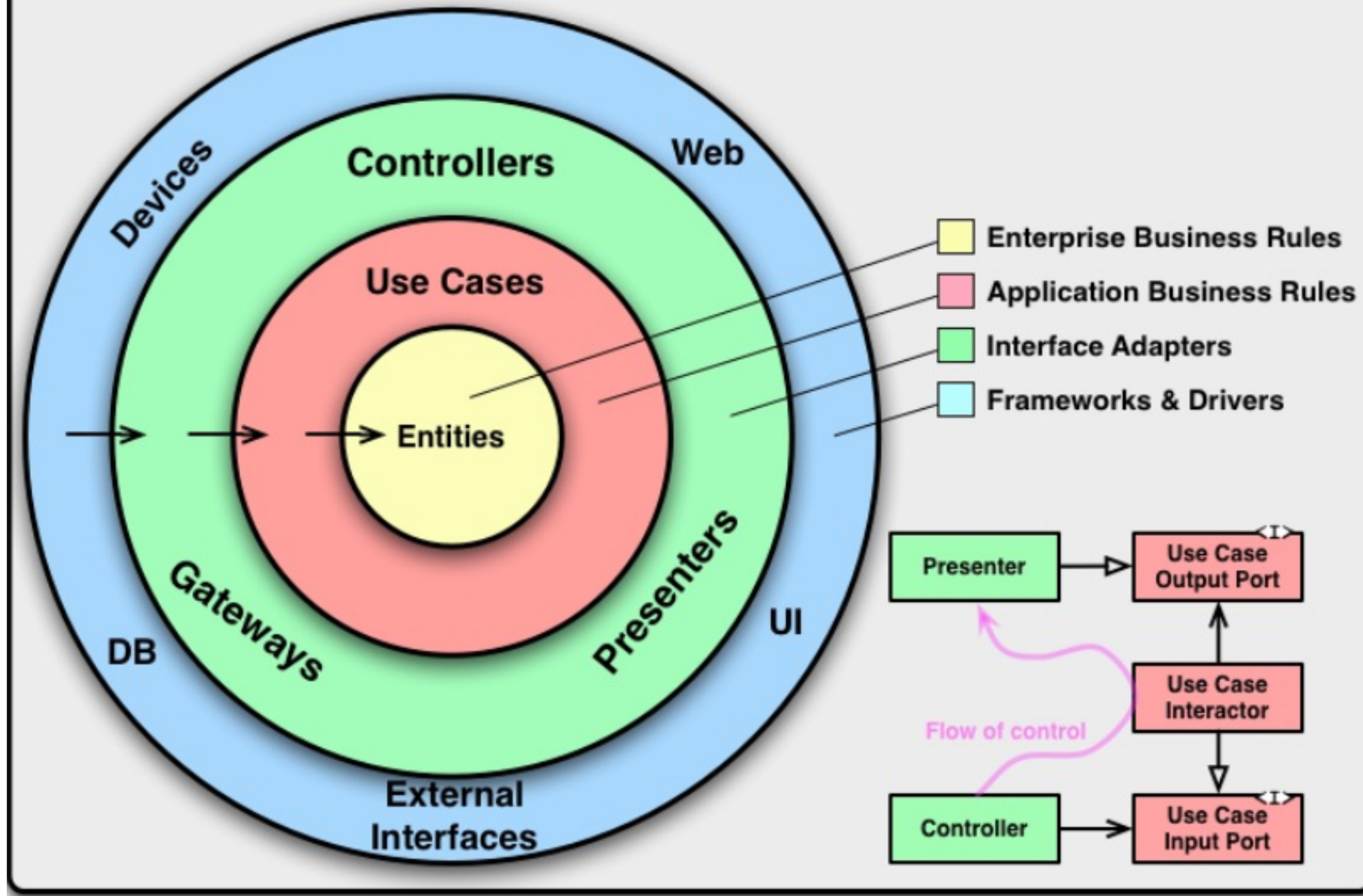
- Разделение между кодом архитектуры и кодом имплементации
- Большинство кода выглядит как взаимодействие поставщика и потребителя услуг
- Все критические компоненты можно легко заменить

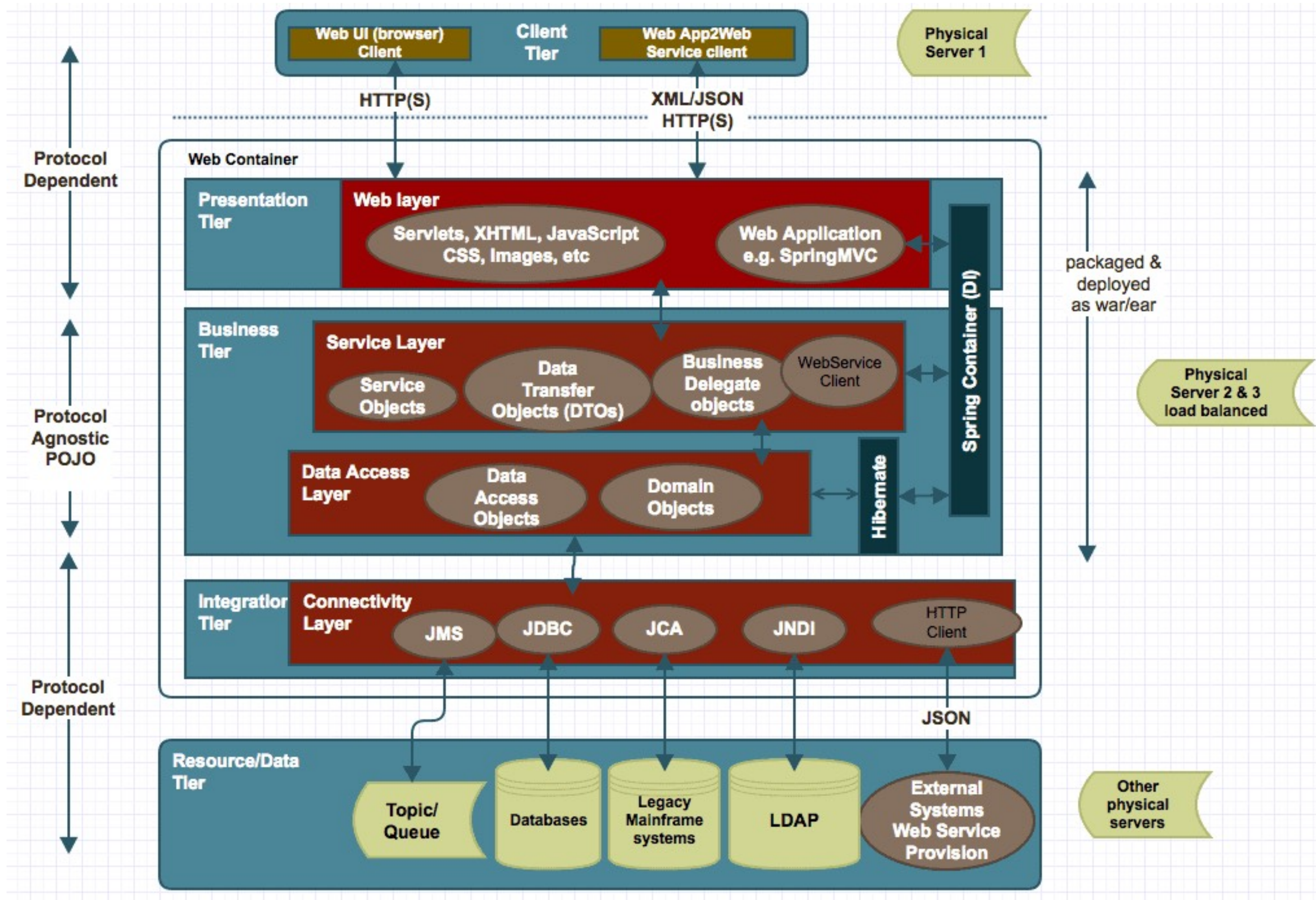
- Архитектуру легко объяснить словами
- Большая часть функционала становится доступна к концу цикла разработки

Есть ли что-то конкретнее?

Многослойная архитектура

The Clean Architecture

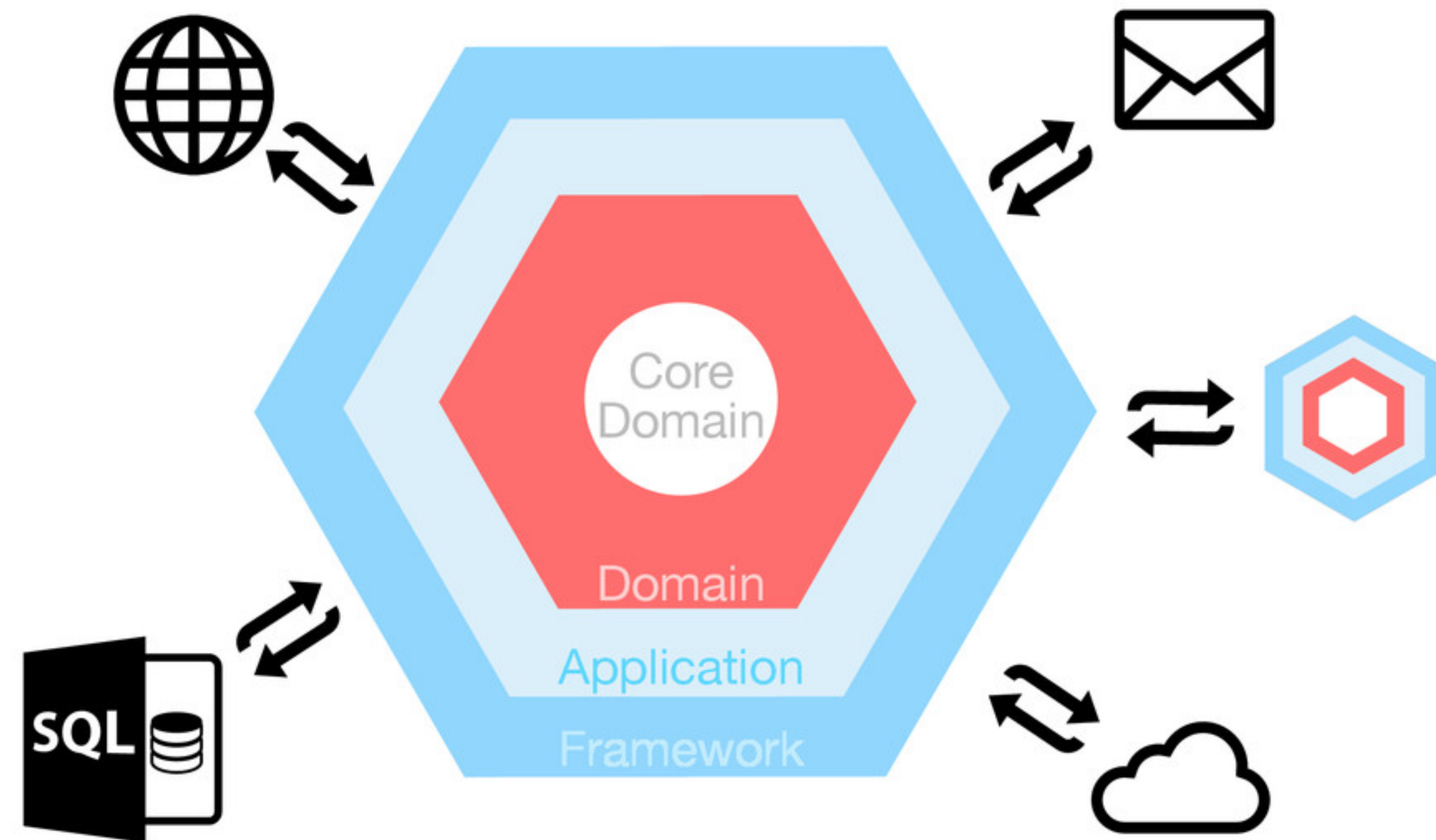




**Круто конечно, но как код
организовывать?**

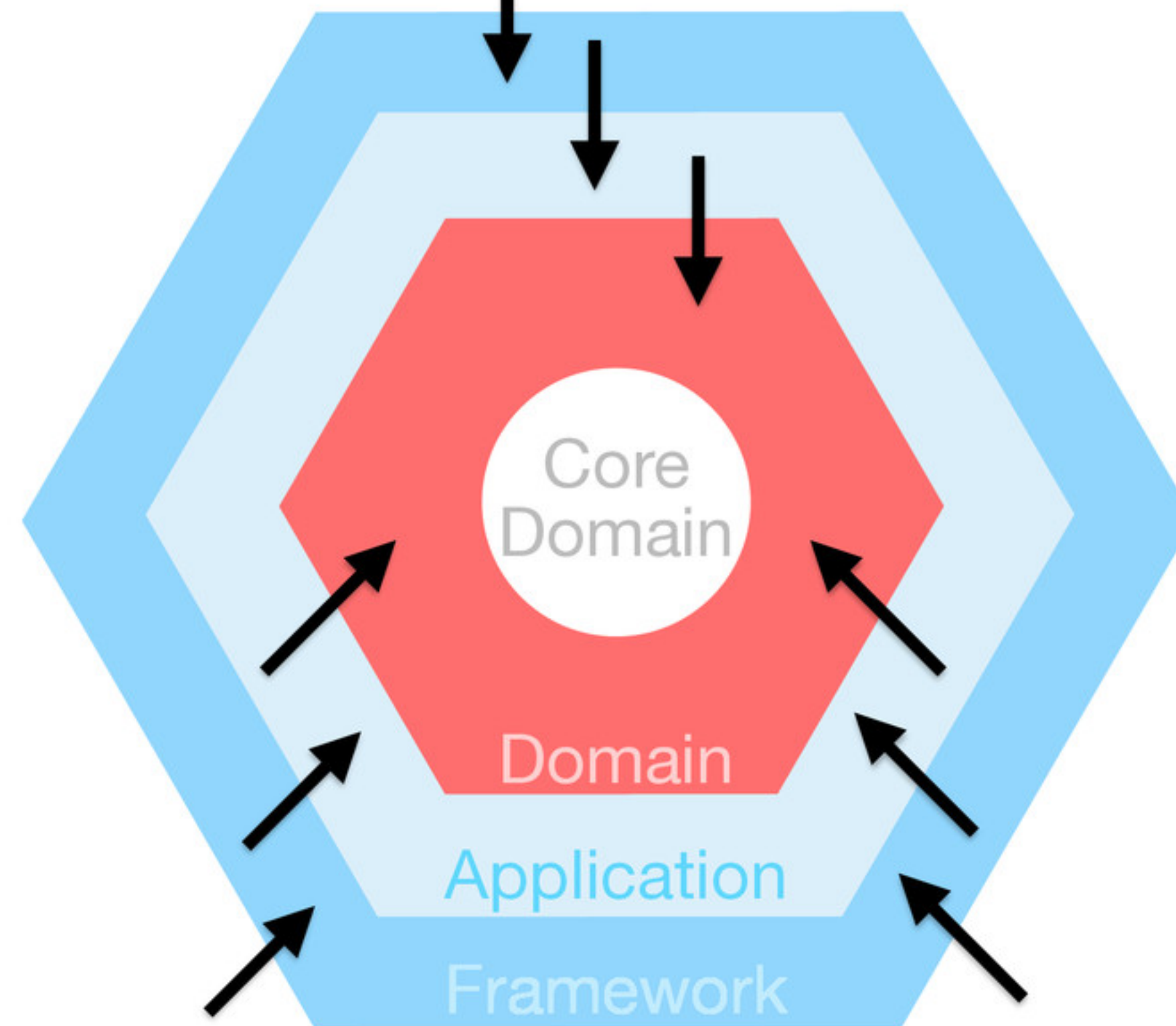
Гексагональная архитектура

The Hexagon



- Domain layer
- Application layer
- Framework layer
- Persistence layer
- Односторонний поток
зависимости

Dependencies



Советы

- Архитектура в первую очередь должна помогать
- Придерживаемся **SOLID**
- Создаем папки под каждый слой
- Дополнительно можем разделять по доменам
- Абстракции лучше имплементаций

Тестирование

Цель тестирования

Проверка соответствия ПО предъявляемым требованиям,
обеспечение уверенности в качестве ПО, поиск ошибок

Принципы тестирования

- Тестирование демонстрирует наличие дефектов
- Исчерпывающее тестирование невозможно
- Раннее тестирование
- Скопление дефектов
- Парадокс пестицида
- Тестирование зависит от контекста
- Заблуждение об отсутствии ошибок

Какие бывают тесты?

- Модульные (Unit)
- Интеграционные (Integration)
- Функциональные (Functional)
- Регрессионные (Regress)
- Нагрузочные (Load)
- Производительные (Performance)

Что мы будем использовать?

Unit tests

**Что мы для этого будем
использовать?**



- Подключаем как зависимость в Gradle
- В отдельном модуле пишем тесты
- Запускаем через Gradle

Домашнее задание

- Добавить аутентификацию и авторизацию
- Сделать в отдельном Pull Request
- Добавить меня в reviewers
- Подробное описание в отдельном файле

Материалы

- Про MVC: <https://clck.ru/36AoaC>
- Про аутентификацию: <https://clck.ru/36Aod6>
- Про слой доступа к данным: <https://clck.ru/36AogQ>
- Про гексогональную архитектуру: <https://clck.ru/36Aojc>

- Про unit тесты: <https://clck.ru/36Aon2>

Спасибо за внимание!