Base Application infrastructure

Матвей Попов

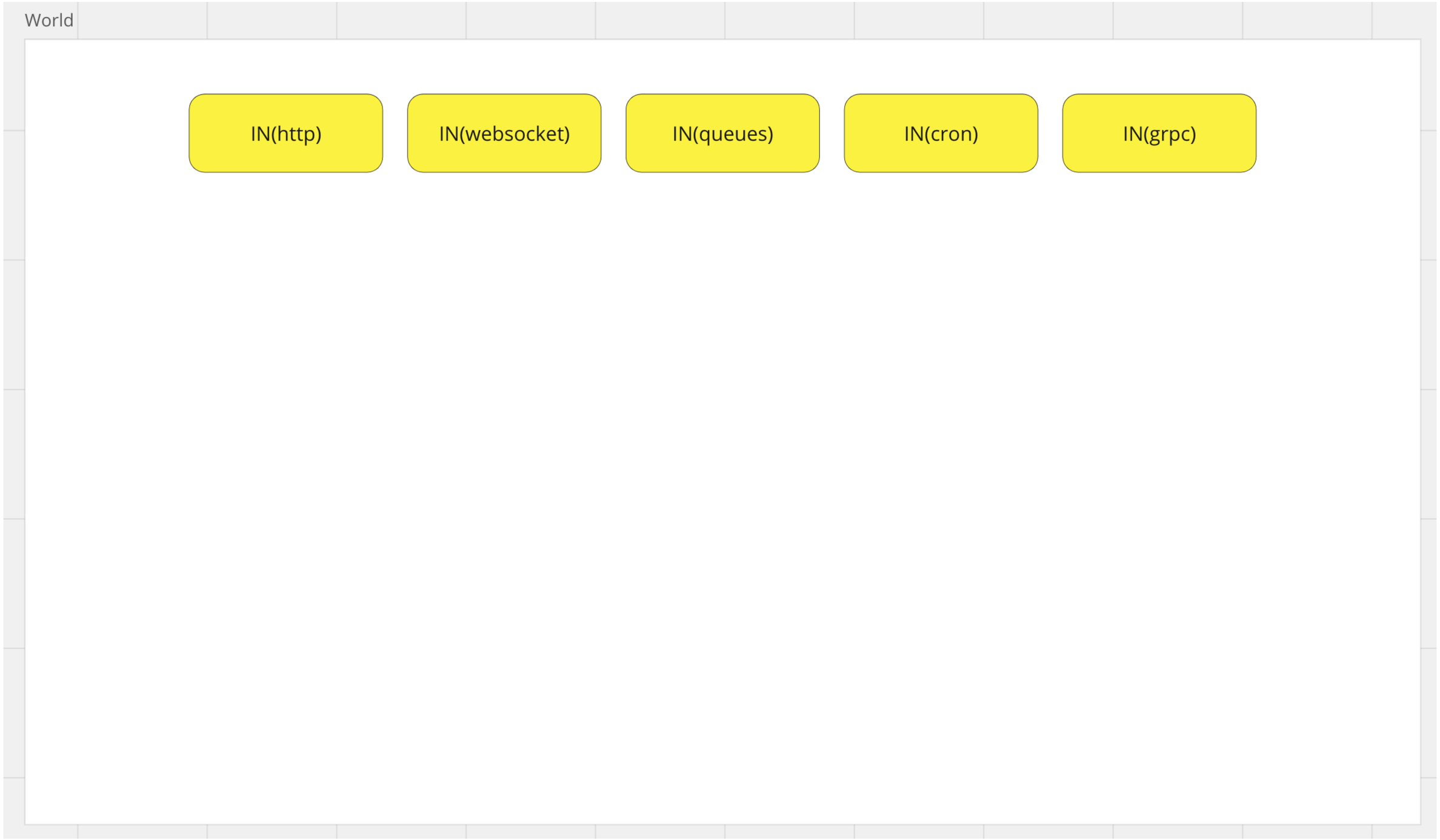
План

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

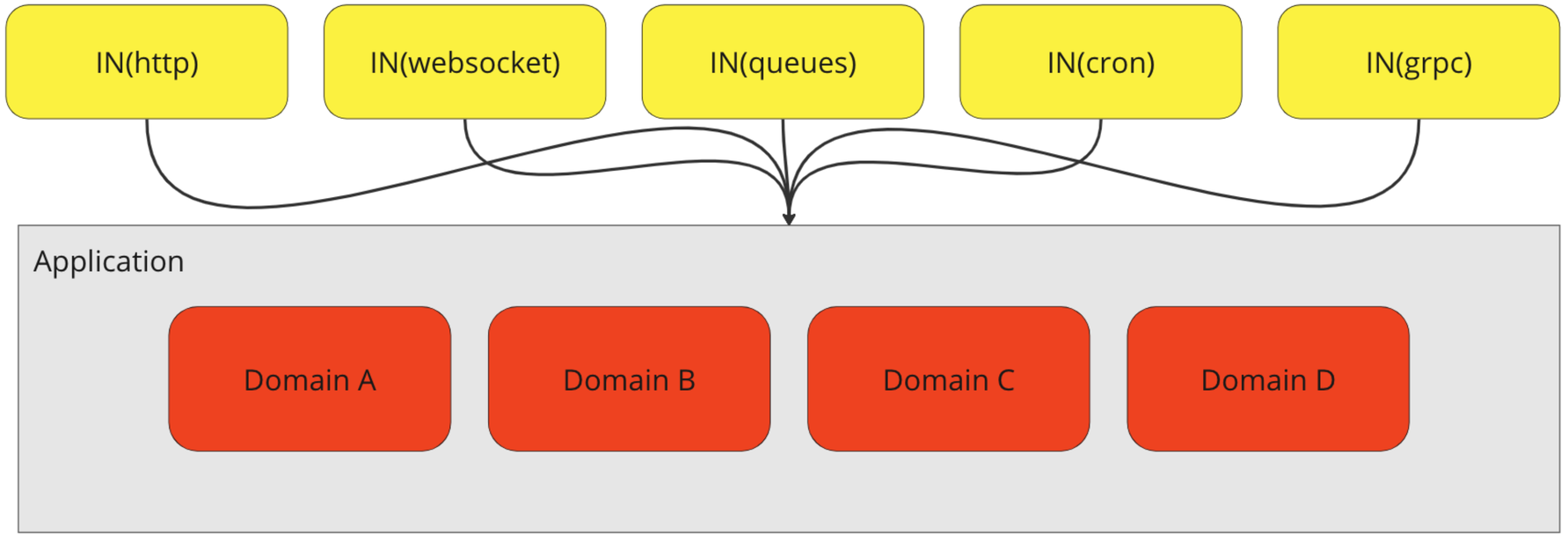
Из чего же состоит наше приложение?

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

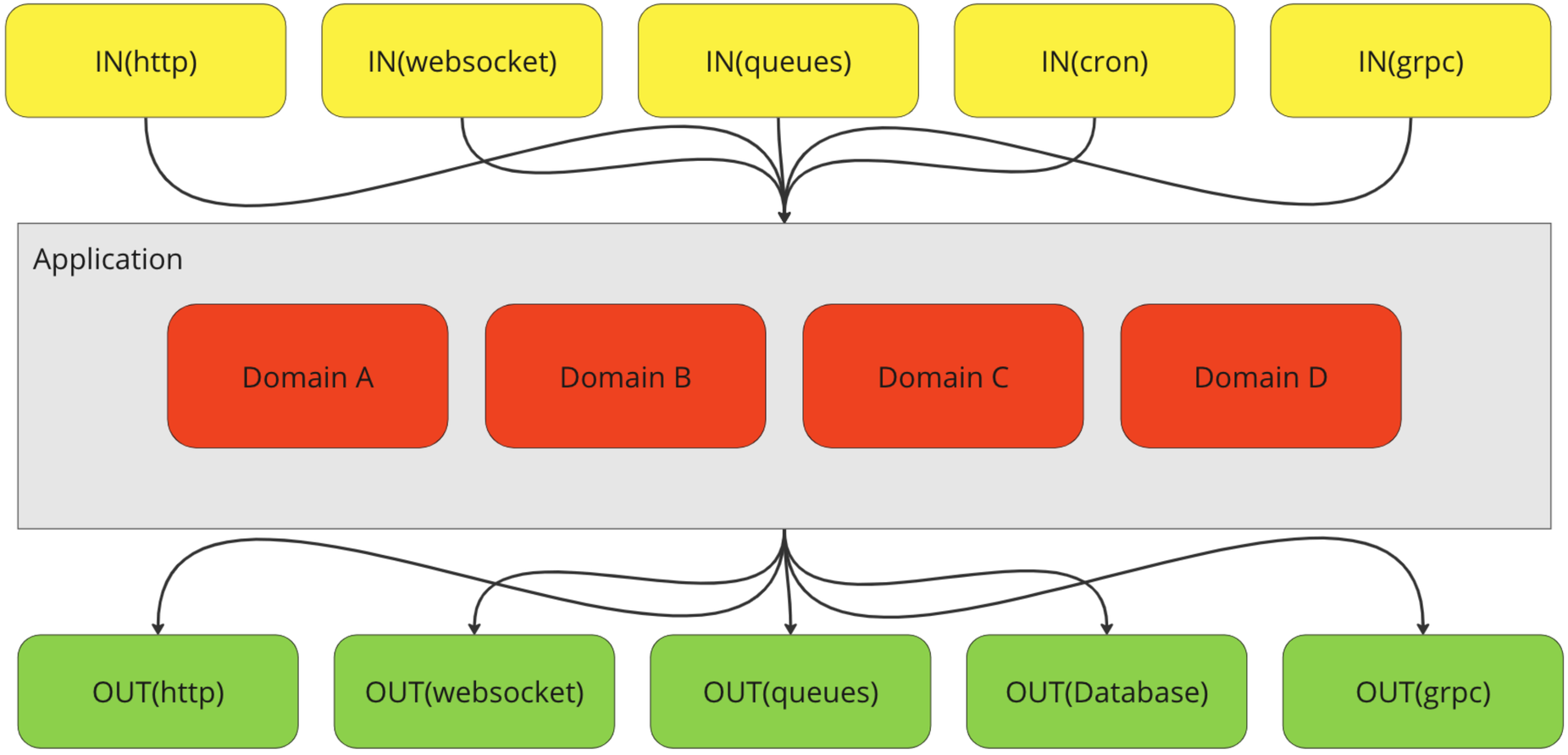
Как наш код выглядит?


World
IN(http)
IN(websocket)
IN(queues)
IN(cron)
IN(grpc)

World


IN(http)
IN(websocket)
IN(queues)
IN(cron)
IN(grpc)
Application
Domain A
Domain B
Domain C
Domain D

World


IN(http)
IN(websocket)
IN(queues)
IN(cron)
IN(grpc)
Application
Domain A
Domain B
Domain C
Domain D
OUT(http)
OUT(websocket)
OUT(queues)
OUT(Database)
OUT(grpc)

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

Валидиция 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 данных(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 - atomicity
- 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 SharingSphere

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

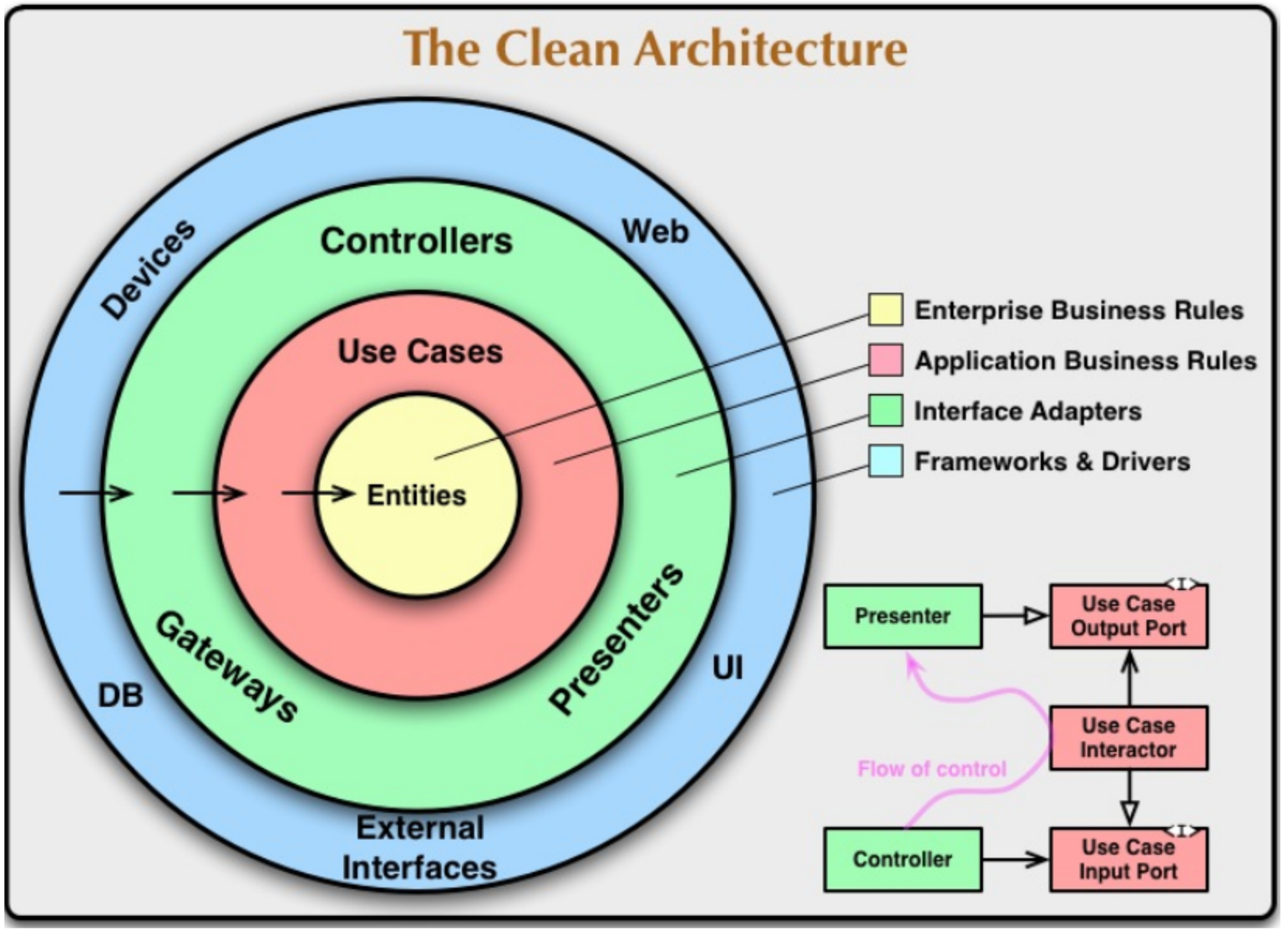
Признаки успешной архитектуры

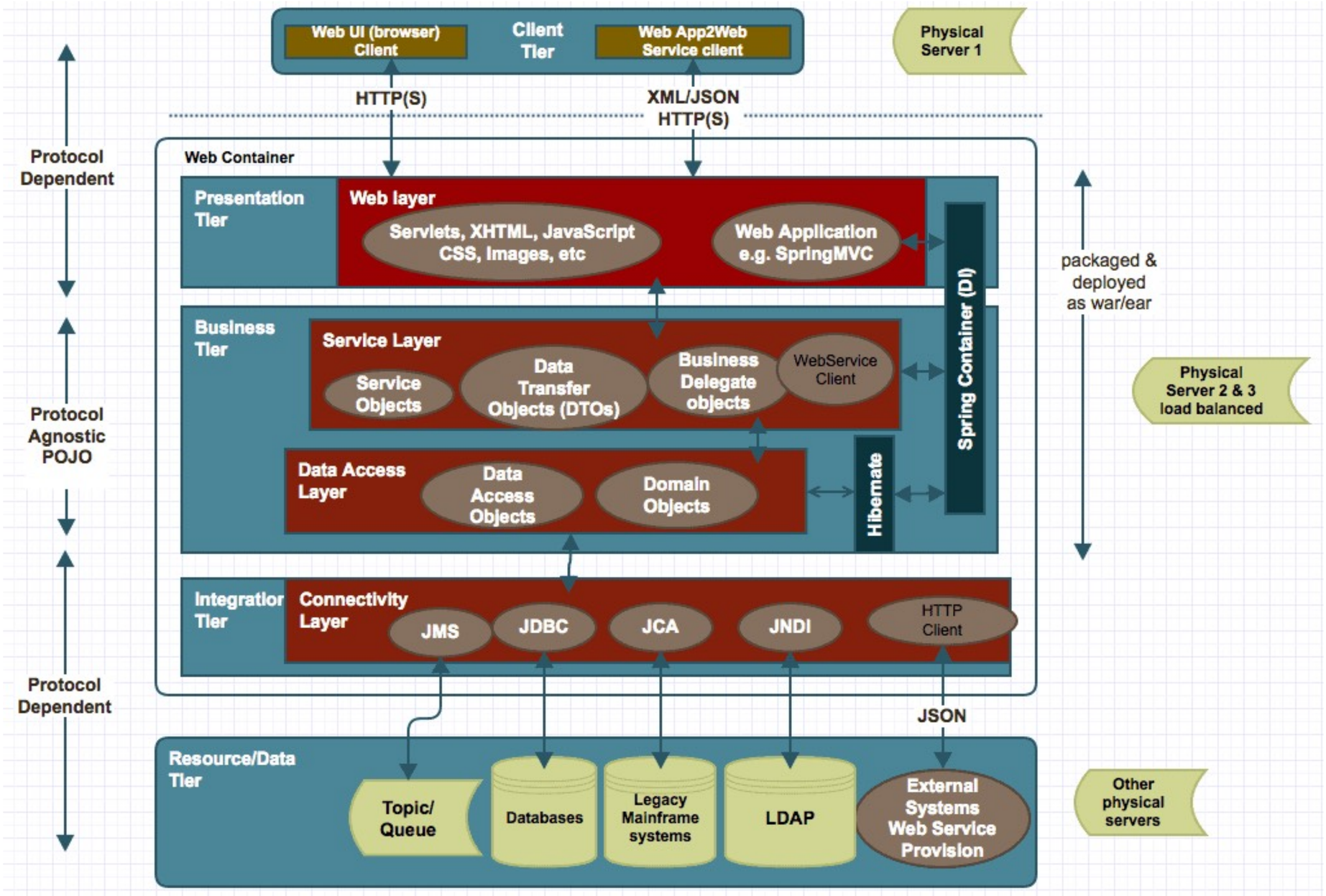
- Разделение между кодом архитектуры и кодом имплементации
- Большинство кода выглядит как взаимодействие поставщика и потребителя услуг
- Все критические компоненты можно легко заменить

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

Есть ли что-то конкретнее?

Многослойная архитектура

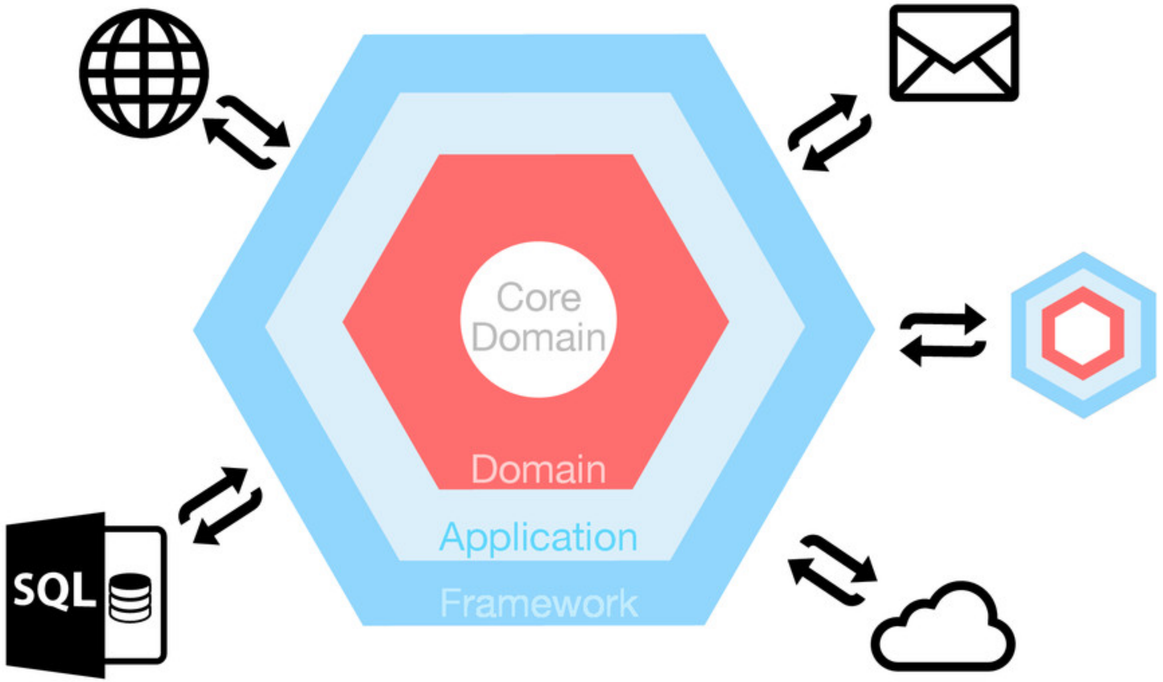

The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port


Web UI (browser) Client
Client Tier
Web App2Web Service client
Physical Server 1
HTTP(S)
XML/JSON HTTP(S)
Protocol Dependent
Web Container
Presentation Tier
Web layer
Servlets, XHTML, JavaScript CSS, Images, etc
Web Application e.g. SpringMVC
Spring Container (DI)
packaged & deployed as war/ear
Business Tier
Service Layer
Service Objects
Data Transfer Objects (DTOs)
Business Delegate objects
WebService Client
Physical Server 2 & 3 load balanced
Protocol Agnostic POJO
Data Access Layer
Data Access Objects
Domain Objects
Hibernate
Integration Tier
Connectivity Layer
JMS
JDBC
JCA
JNDI
HTTP Client
Protocol Dependent
JSON
Resource/Data Tier
Topic/ Queue
Databases
Legacy Mainframe systems
LDAP
External Systems Web Service Provision
Other physical servers

Круто конечно, но как код организовывать?

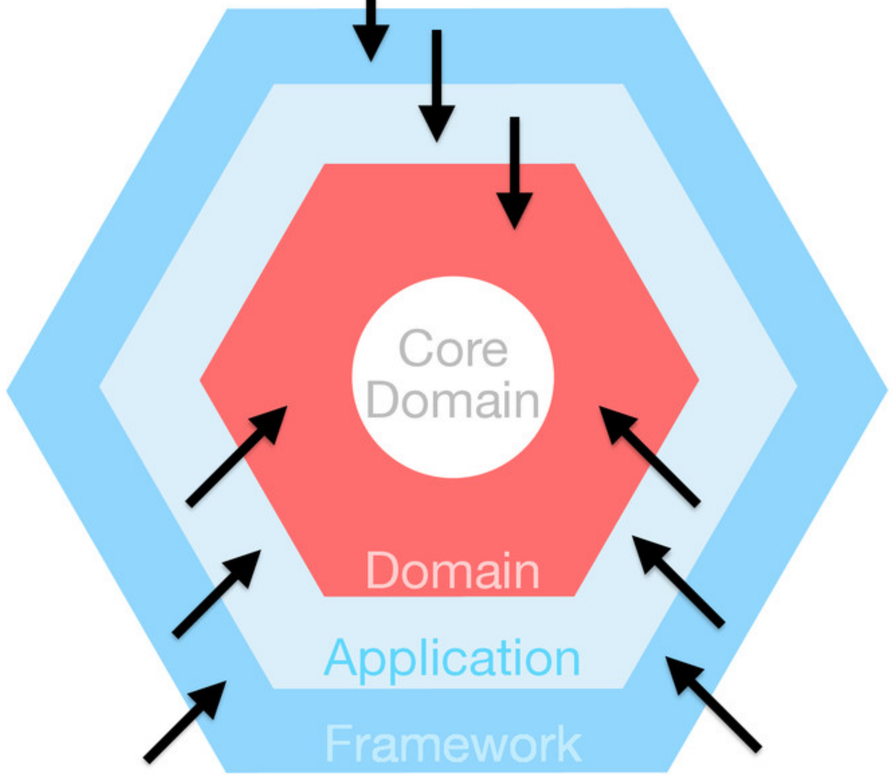
Гексагональная архитектура

The Hexagon



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

Dependencies



Советы

- Архитектура в первую очередь должна помогать
- Придерживаемся SOLID
- Создаем папочки 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
- Исчерпывающие 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?


JUnit
5

- 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!