Node.js Stack for Enterprise (Part 3)

Что делаем сегодня

- Смотрим код (который недосмотрели)
- Где нам помогли GRASP, SOLID, DI, DDD
- Отделяем модель от сервисов

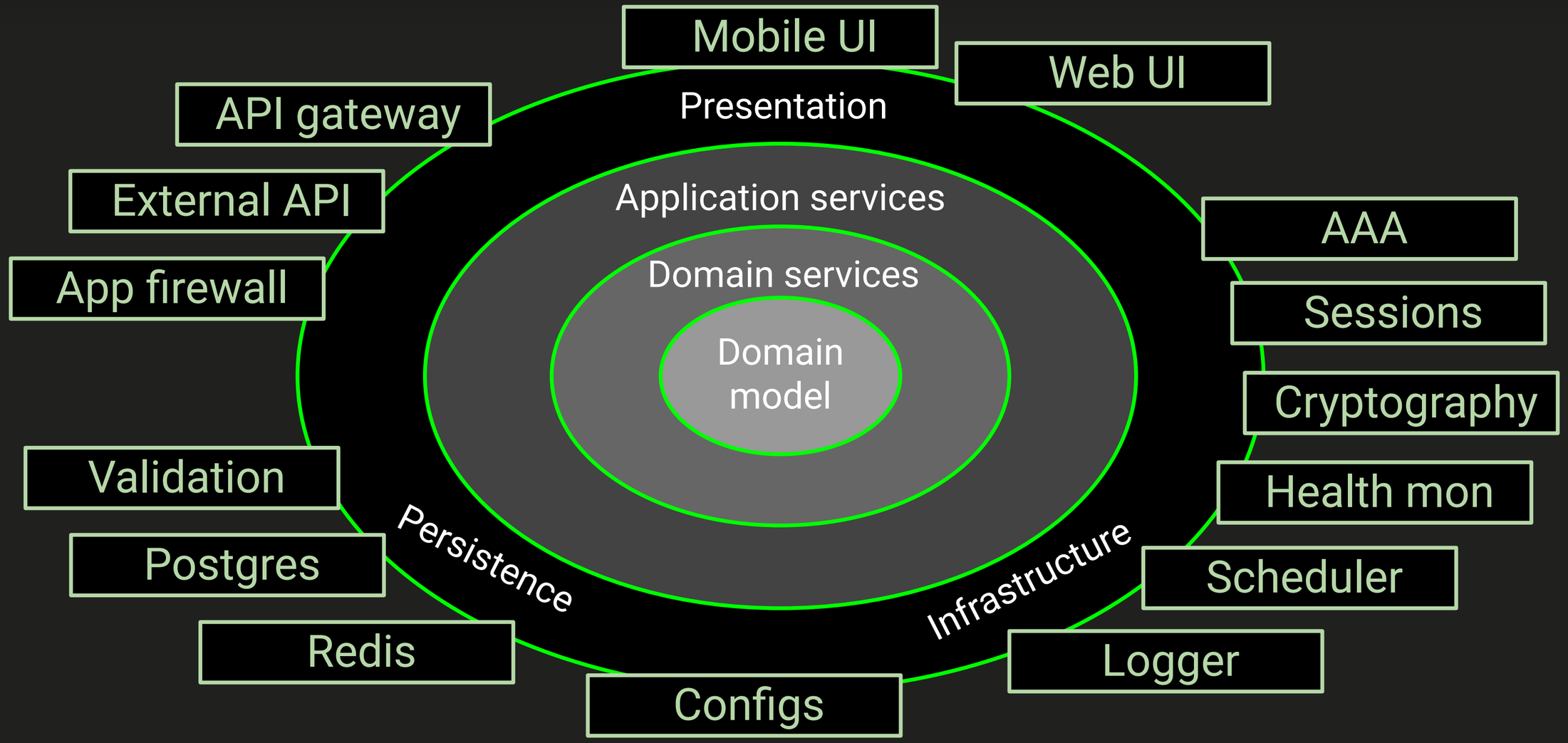
В предыдущих сериях

- Фреймворк-агностик, транспорт-агностик
- Изоляция кода и безопасность
- Нет состояний гонки в асинхронном коде
- Нет утечек памяти и ресурсов
- Надежно обрабатываются ошибки
- DI (внедрение зависимостей)
- Авто-роутинг на файловой системе

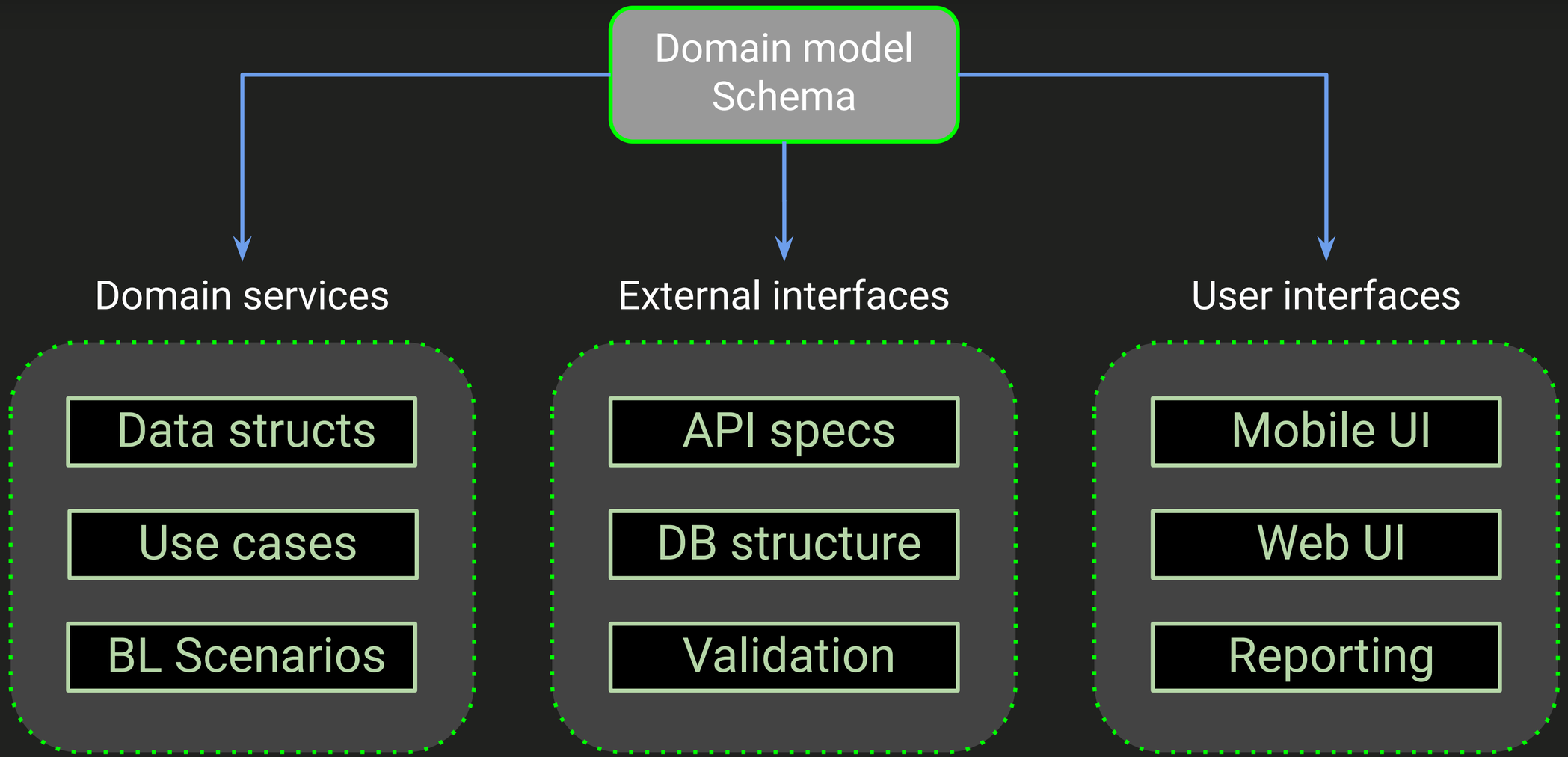
Давайте посмотрим код

graceful shutdown
metawatch
semaphore
валидация контракта
metaschema

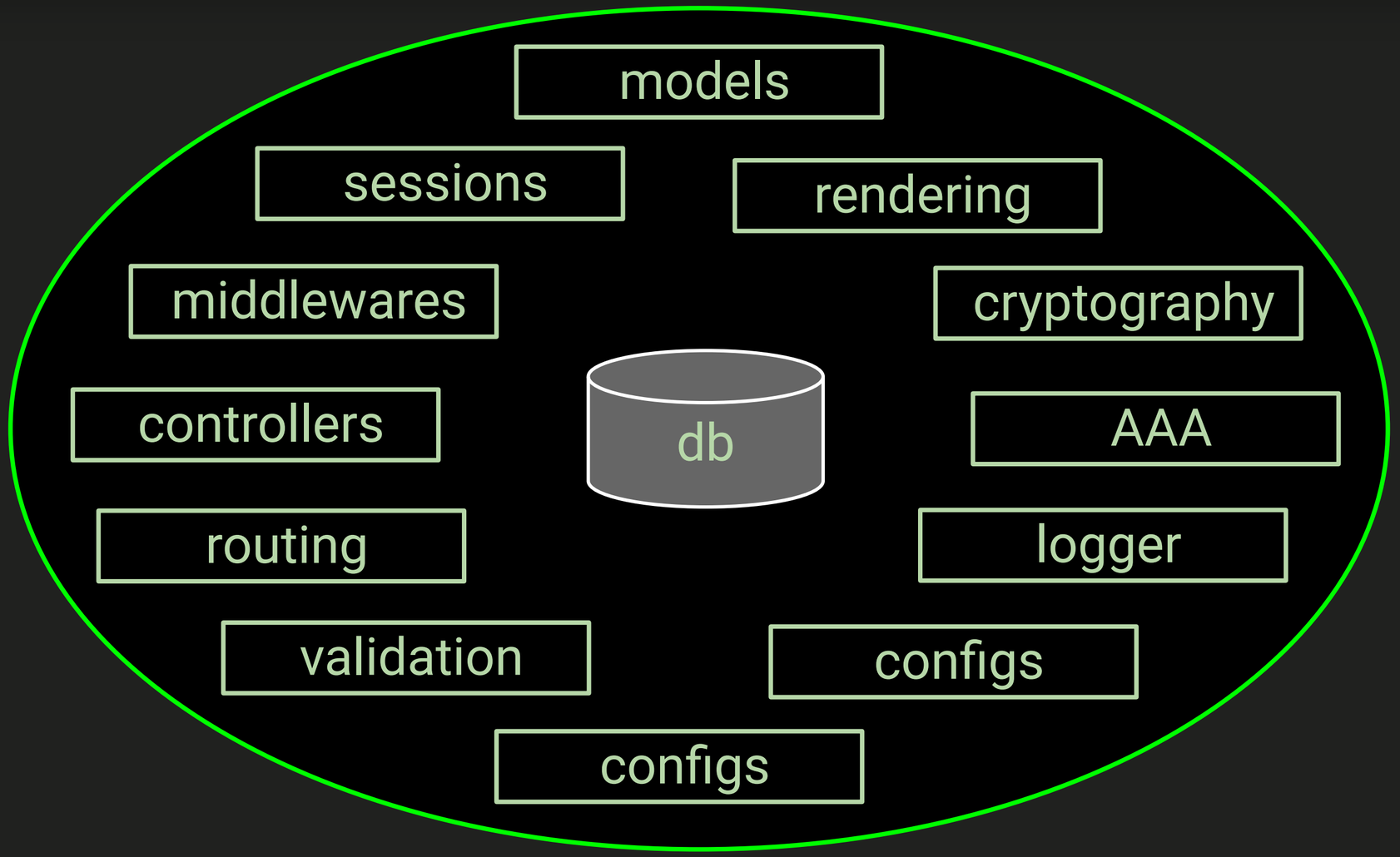
Application layered (onion) Architecture



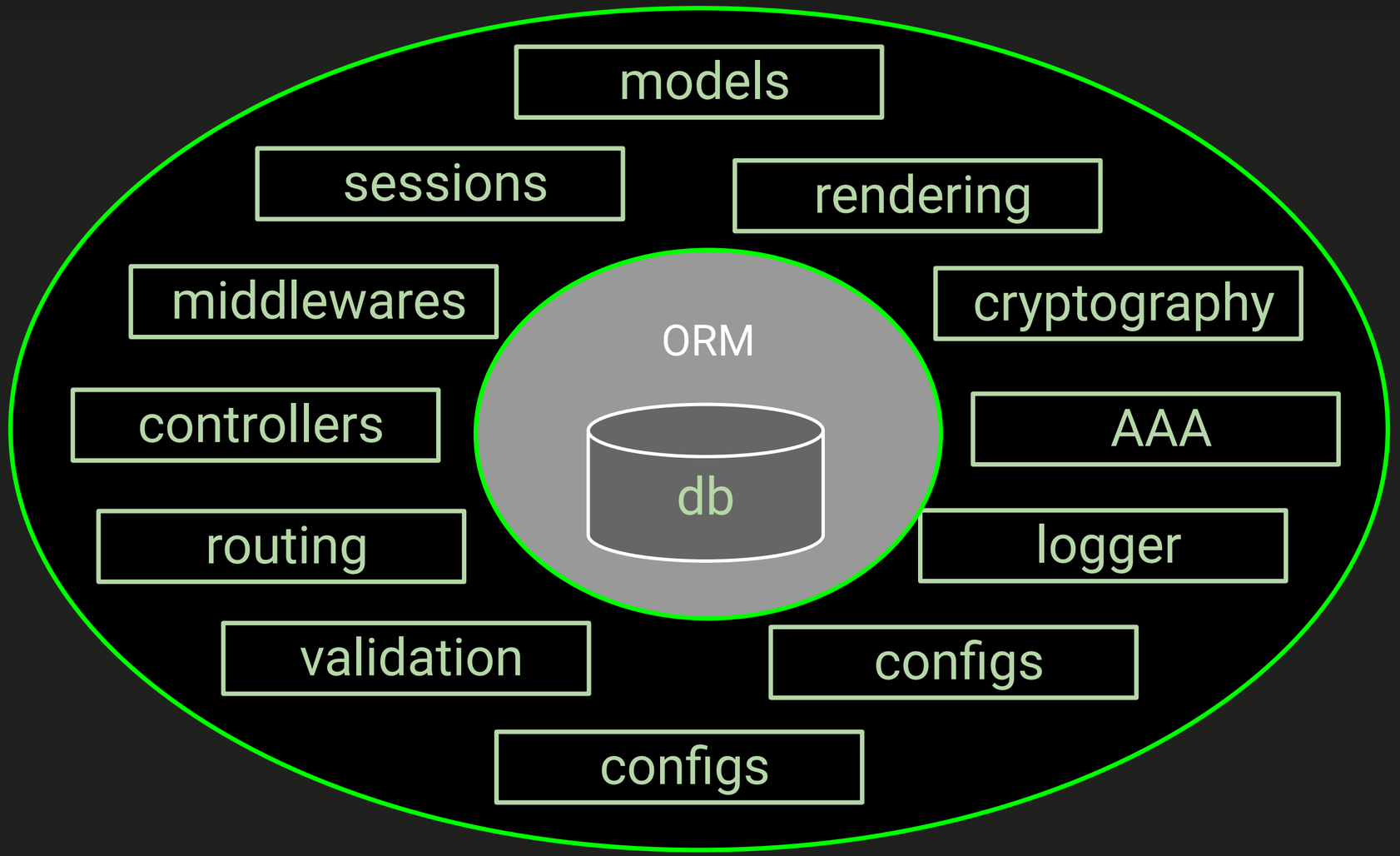
Schema-centric approach



Express architecture: Big ball of mud



Express architecture: Big ball of mud



Пример схем

Schemas/SystemUser.js

```
({
  login: { type: 'string', unique: true, length: 30 },
  password: { type: 'string', length: 10 },
  fullName: { 'string' required: false },
});
```

Пример схем

Schemas/SystemGroup.js

```
({
  name: { type: 'string', unique: true },
  users: { many: 'SystemUser' },
});
```

Пример схем

Schemas/SystemSession.js

```
({
  user: 'SystemUser',
  token: { type: 'string', unique: true },
  ip: 'string',
  data: 'json',
});
```

Генерация структуры базы

```
CREATE TABLE "SystemUser" (
  "systemUserId" bigint generated always as identity,
  "login" varchar(30) NOT NULL,
  "password" varchar(10) NOT NULL,
  "fullName" varchar
);

ALTER TABLE "SystemUser"
ADD CONSTRAINT "pkSystemUser"
PRIMARY KEY ("systemUser");
```

Генерация структуры базы

```
CREATE TABLE "SystemGroup" (
  "systemGroupId" bigint generated always as identity,
  "name" varchar NOT NULL
);

ALTER TABLE "SystemGroup"
ADD CONSTRAINT "pkSystemGroup"
PRIMARY KEY ("systemGroup");
```

Генерация структуры базы

```
CREATE TABLE "SystemGroupSystemUser" (
  "systemGroupId" bigint NOT NULL,
  "systemUserId" bigint NOT NULL
);

ALTER TABLE "SystemGroupSystemUser"
ADD CONSTRAINT "pkSystemGroupSystemUser"
PRIMARY KEY ("systemGroupId", "systemUserId");
```

Генерация структуры базы

```
ALTER TABLE "SystemGroupSystemUser"
ADD CONSTRAINT "fkSystemGroupSystemUserSystemGroup"
FOREIGN KEY ("systemGroupId")
REFERENCES "SystemGroup" ("systemGroupId");

ALTER TABLE "SystemGroupSystemUser"
ADD CONSTRAINT "fkSystemGroupSystemUserSystemUser"
FOREIGN KEY ("systemUserId")
REFERENCES "SystemUser" ("systemUserId");
```

Генерация структуры базы

```
CREATE TABLE "SystemSession" (
  "systemSessionId" bigint generated always as identity,
  "userId" bigint NOT NULL,
  "token" varchar NOT NULL,
  "ip" varchar NOT NULL,
  "data" jsonb NOT NULL
);

ALTER TABLE "SystemSession"
ADD CONSTRAINT "pkSystemSession"
PRIMARY KEY ("systemSession");
```

Генерация структуры базы

```
ALTER TABLE "SystemSession"
ADD CONSTRAINT "fkSystemSessionUser"
FOREIGN KEY ("userId")
REFERENCES "SystemUser" ("systemUserId");
```

Генерация .d.ts тайпингов

```
interface SystemUser {
  systemUserId: number;
  login: string;
  password: string;
  fullName: string;
}

interface SystemGroup {
  systemGroupId: number;
  name: string;
}
```

Генерация .d.ts тайпингов

```
interface SystemSession {
  systemSessionId: number;
  userId: number;
  token: string;
  ip: string;
  data: string;
}
```

Генерируем из схемы

- Валидация контрактов и данных
- Тайпинги .d.ts для кода
- SQL DDL структура базы
- Авто-миграции и версии БД
- Скаффолдинг доступа к БД
- UI (веб и мобильный)
- Schema + Flow-chart

GRASP

General responsibility assignment software patterns (распределение ответственности)

Книга “Применение UML и шаблонов проектирования” // **Крэг Ларман**

GRASP

General responsibility assignment software patterns (распределение ответственности)

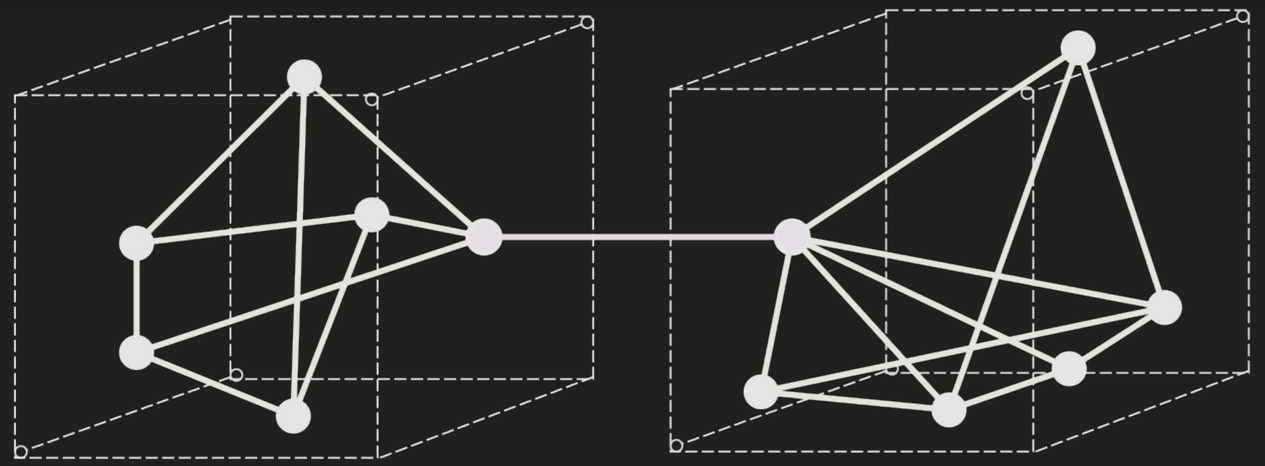
- Low Coupling
- Information Expert
- Controller
- Pure Fabrication
- Protected Variations
- High Cohesion
- Creator
- Polymorphism
- Indirection

GRASP: Information Expert

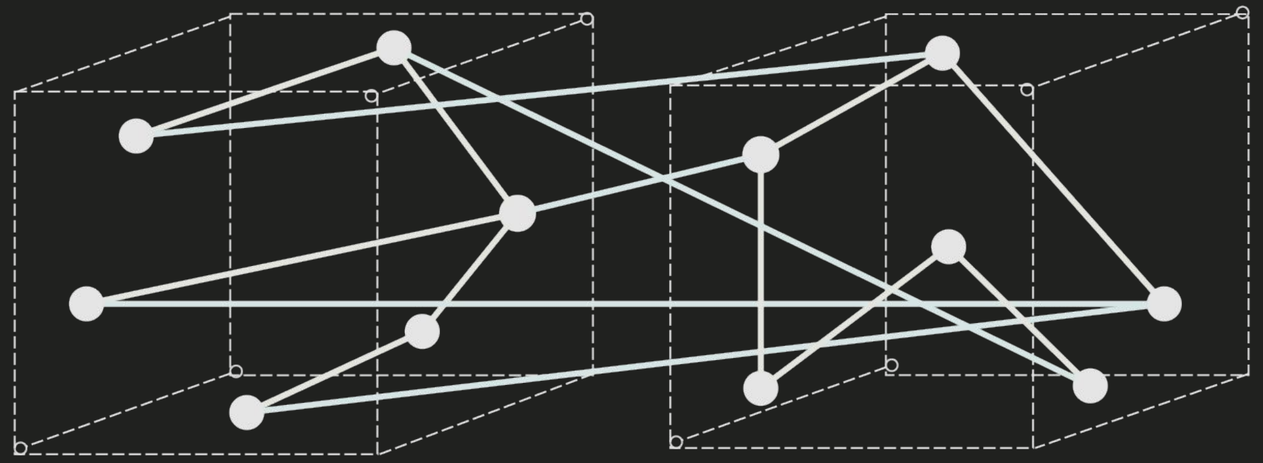
Проблема: как распределить ответственность между классами?
Решение: назначить ответственность классу, который имеет всю необходимую информацию для работы (принятия решения).
Зачем: снижает зацепление, упрощает код, повышает инкапсуляцию, переиспользование.

GRASP: Cohesion & Coupling

Cohesion - связность внутри класса, модуля, компонента



Coupling - зацепление между классами, модулями...



GRASP: Creator

Проблема: кто должен создавать инстанс? (кто держит ссылку или разрушает)
Решение: тот, кто содержит или агрегирует инстансы, кто интенсивно работает с ними, кто имеет информацию для инициализации.
Зачем: для снижения зацепления.
Примеры: конструктор, фабрика, пул.

GRASP: Controller

Controller - точка входа для внешнего взаимодействия, выполняет системные операции и делегирует бизнес-логику.
Проблема: кто и как взаимодействует с UI?
Зачем: защита от событий, конкурентности, асинхронности и параллельности.
Примеры: команда, фасад, изоляция слоев.

GRASP: Polymorphism

Polymorphism - альтернативное поведение на основании типа (см. SOLID: LSP, OCP).
Проблема: как быть, если в зависимости от типа нужно изменять поведение?
Решение: заменяем if и case на полиморфизм и наследование, обращаемся через интерфейс или абстрактный класс.

GRASP: Indirection

Indirection - Введение объекта-посредника снижает зацепление между абстракциями.
Зачем: снижает зацепление, улучшает переиспользование кода.
Пример: шаблон Mediator (посредник) из GoF, в шаблоне MVC, C - controller.

GRASP: Pure fabrication

Чистая выдумка - абстракция, которой нет в предметной области. Она часто позволяет снизить зацепление классов предметной области.
Примеры: Socket, DB Query, EventEmitter, Promise, список, асинхронная очередь.

GRASP: Protected variations

Устойчивость к изменениям:
защита абстракций от изменения путем спецификации интерфейсов или контрактов и взаимодействию через них.

Что мы применяли

- Архитектура: DDD и clean architecture, слоеная (layered/onion) архитектура, модель в центре
- Принципы и паттерны: GRASP, SOLID, GoF, IoC, DI, coupling/cohesion, LoD закон Деметры
- Структура проекта: не зависит от фреймворка
- Техники: асинхронное и параллельное программирование, метапрограммирование

LoD Закон Деметры

- Каждый модуль как можно меньше “знает” о других (low coupling), через интерфейсы (ISP)
- Каждый программный компонент, взаимодействует только с друзьями (явно)
- Метод класса обращается к своим аргументам, методам и свойствам инстанса или и его структурных частей первого уровня

Что мы получили

- Переиспользование, нет бойлерплейт-кода
- Очень мало зависимостей и они надежные (pg 803 kb, ws 118 kb = 921 kb)
- Системный код 6 kb, прикладной код: 837 b
- Metarhia (5 июня): 286 kb (JavaScript: 155 kb)
- Характеристики: переносимость, надежность, безопасность, поддерживаемость

Что дальше

- Динамическая генерация .d.ts тайпингов
- Скаффолдинг api интерфейсов на клиенте
- Скаффолдинг модели данных на клиенте
- Автоматические миграции из схем
- Состояние с автоматической защитой
- Визуальное моделирование процессов BPMN
- Визуальное моделирование данных в ERD

github.com/metarhia

github.com/tshemsedinov
youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks/Index

Largest Node.js and JavaScript course
https://habr.com/ru/post/485294/

t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com