Т БАНК

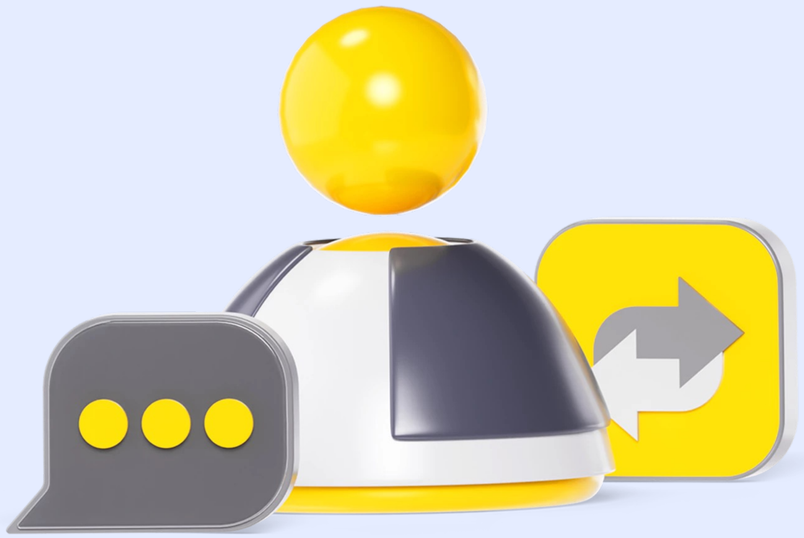
Чистая архитектура

Ещё один доклад про архитектуру

Игорь Антонов

21.11.2024

Привет 👋



Игорь Антонов

TeamLead в Т-Бизнес

Помогаю делать конференции Podlodka Java Crew, Podlodka TechLead Crew, Infostart Event

Читаю лекции в HTML Academy, ИТМО

Веду канал на YouTube и Telegram — https://youtube.com/@antonovjs и https://t.me/antonovjs

Про опыт

- 19 лет в IT
- Четыре компании
- Разный стэк

О чём пойдёт речь

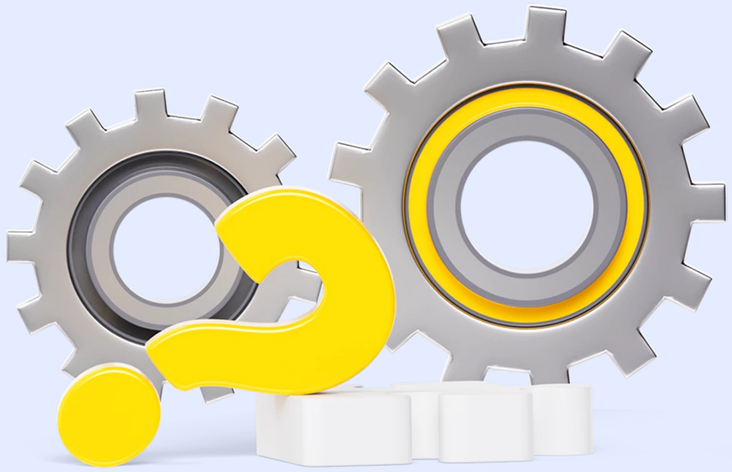
1. Архитектура ПО
2. Разные архитектуры
3. Чистая архитектура
4. Архитектура и реальный мир
5. Эпилог

Введение в архитектуру

Жизнь...

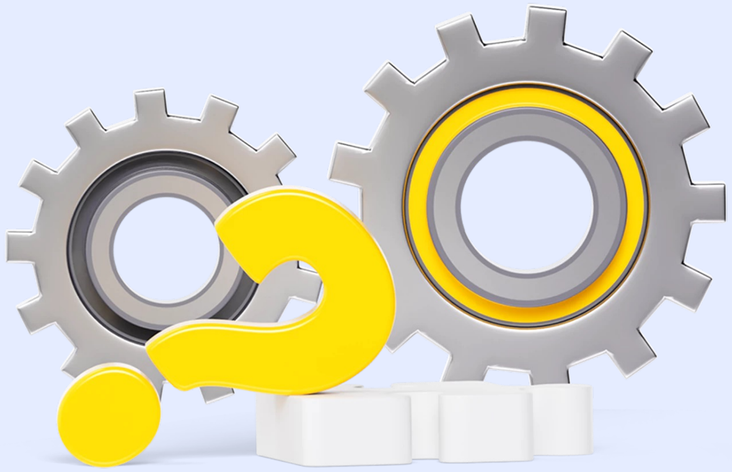
- Программировать, чтобы оно «заработало»
- Программировать, чтобы оно работало «правильно»
- Программировать, чтобы не испытывать боль с поддержкой
- Перестать программировать ☺️

Архитектура



- Это файловая структура? Чтобы разложить все файлы по полочкам.
- Это «правильный» фреймворк, лежащий в основе приложения?
- Микросервисы vs Монолиты?
- Доступность? Надёжность? Тестируемость? Отказоустойчивость?

Архитектура



- ~150 определений
- Наиболее важные характеристики системы

«**Архитектура** — самые важные решения, которые принимаются перед началом разработки»

Хорошая и плохая архитектура

Верхнеуровневый дизайн

Архитектура приложения

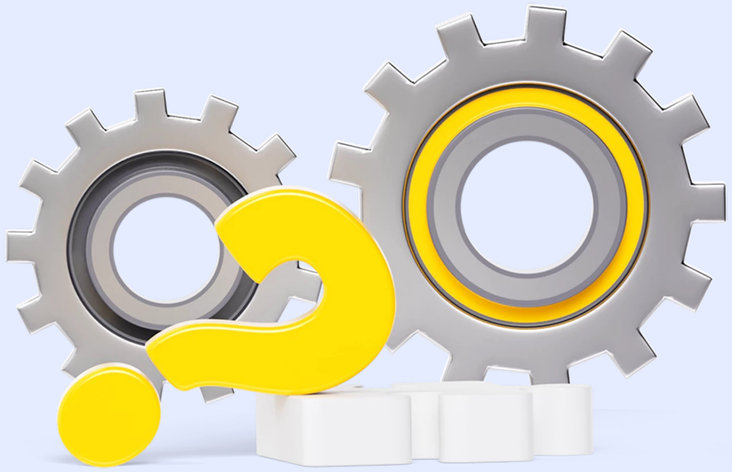
Производительность

Безопасность

Доступность

Код

Разработчик и архитектура

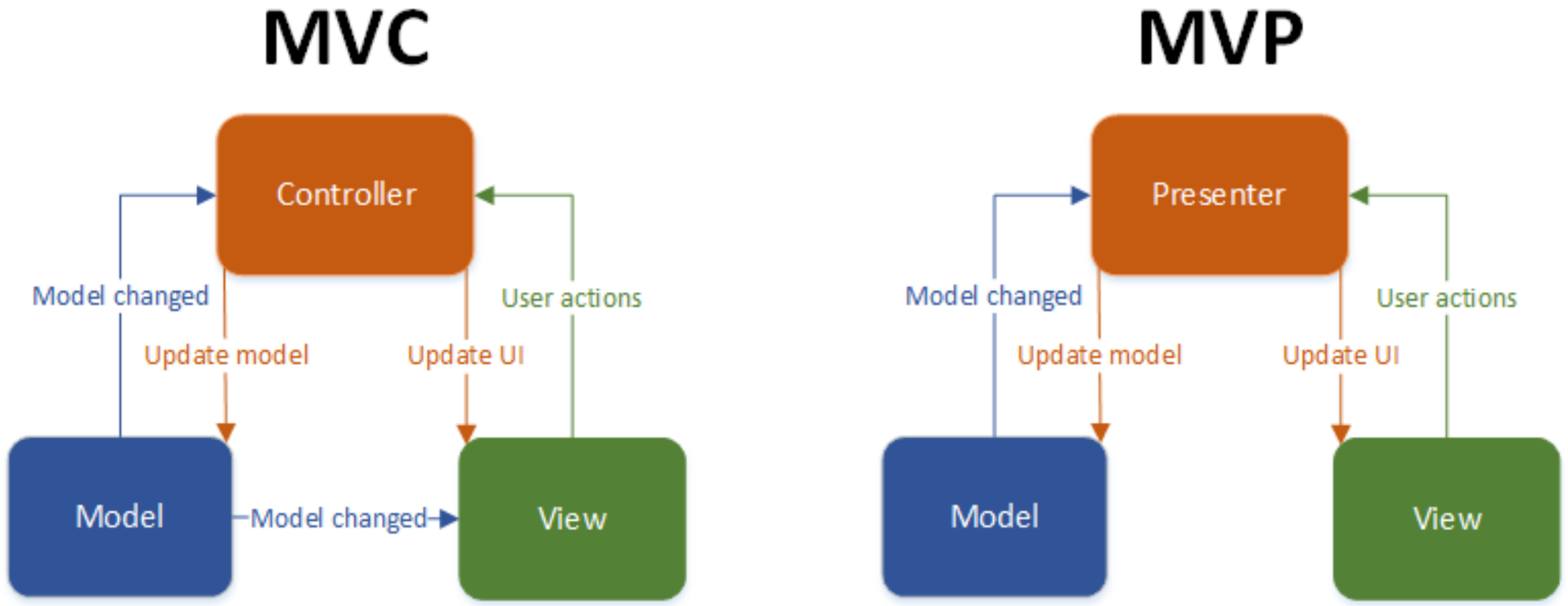


- Простая поддержка
- Отсутствие хрупкости
- Расширяемость
- Тестируемость

Разные подходы к архитектуре

Архитектура

- Разделение ответственности
- Модули (компоненты)
- Публичные контракты
- Связи

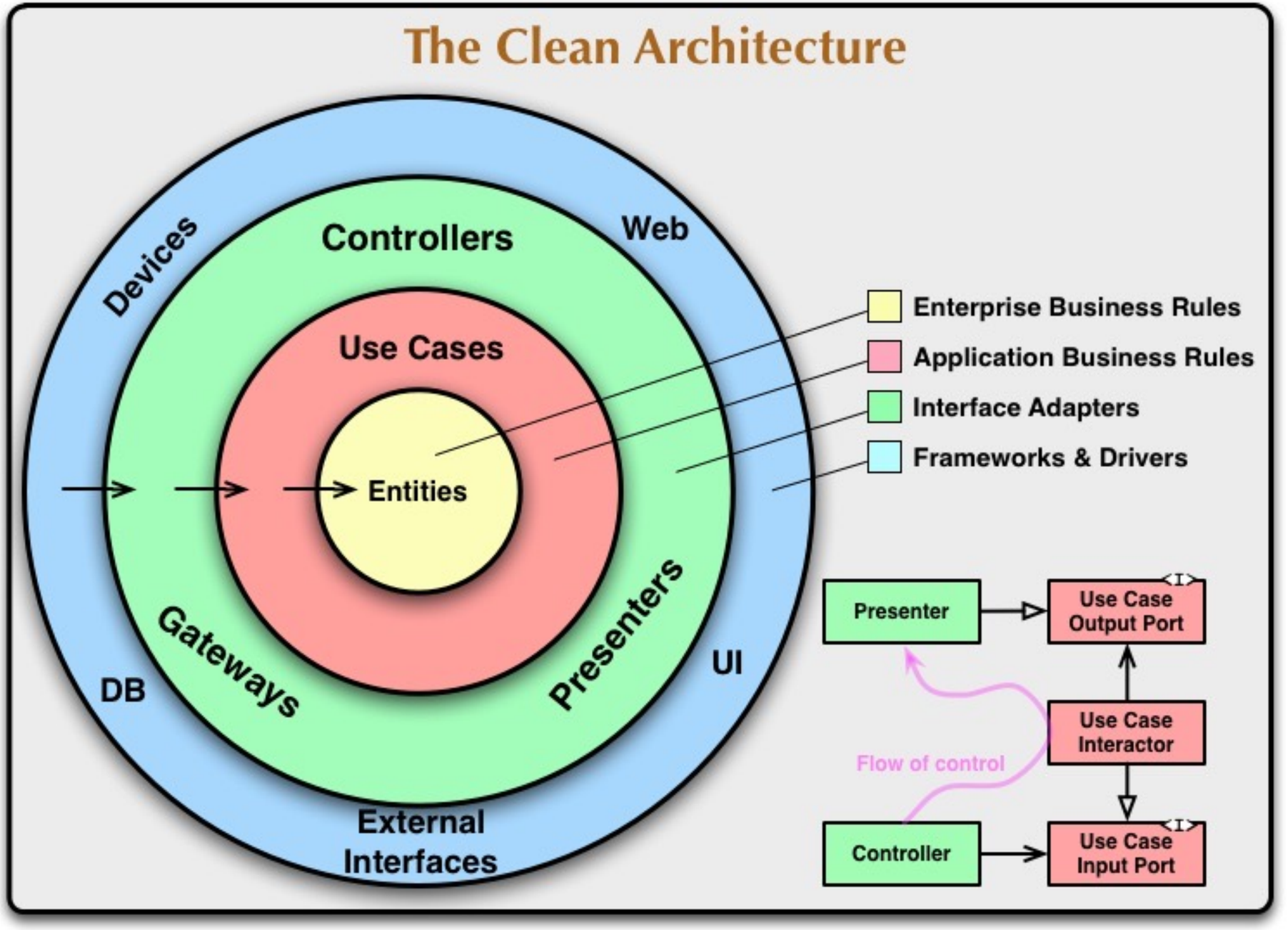

MVC
Controller
Model changed
Update model
Update UI
User actions
Model
Model changed
View
MVP
Presenter
Model changed
Update model
Update UI
User actions
Model
View

Т БАНК

Чистая архитектура

Роберт Мартин

Чистая архитектура — это методология проектирования



https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Чистая архитектура

- **Цель архитектуры** — управлять сложностью. Чем сложнее система, тем важнее иметь чистую и понятную архитектуру.
- Архитектура **не должна зависеть** от фреймворков, баз данных, UI и других деталей реализации
- Хорошую архитектуру **легко изменять и поддерживать**.
- Архитектура должна поддерживать принципы инверсии зависимостей. Зависимости направлены от абстракций к реализациям, а не наоборот.
- Чистая архитектура **не должна зависеть** от языка программирования

Чего хотелось лично мне

- Стандартизация
- Переиспользовать подходы между проектами
- Упростить поддержку
- Модульность, гибкость

Чистая архитектура изнутри

Entities

- **Сущности** — самые важные компоненты приложения. Объекты предметной области, бизнес-процессов.
- Сущности **не зависят** от других компонентов приложения
- Сущности должны легко тестироваться
- Сущности могут содержать бизнес-логику и определять правила обработки данных.

Use Cases

- **Оркестратор** над Entities
- Слой определяет **действия**, которые могут быть выполнены с помощью сущностей
- Проще говоря — описывают сценарии использования системы
- Логика приложения, сервисы или обработчики CQRS

Interface Adapters

- **Преобразуют** данные из формата сценариев и сущностей для использования с внешними системами.
- Доступ к данными, контроллеры, внешние сервисы, фоновые задания и так далее.

Frameworks и Drivers

- Внешние инструменты и библиотеки для выполнения специфичных задач.
- Базы данных, фреймворки и так далее.

Т БАНК

Архитектура и реальный мир


Registration
Dapge::
Cacup
Opy:

Базовая Entity

```typescript
export abstract class Entity<T> {
    protected readonly _id: T;

    constructor(id: T) {
      this._id = id;
    }

    get id(): T {
      return this._id;
    }

    equals(object?: Entity<T>): boolean {
      if (object == null || object == undefined) {
        return false;
      }

      if (this === object) {
        return true;
      }

      return this._id === object.id;
    }
  }

```

Базовый ValueObject

```typescript
export abstract class ValueObject<T> {
    protected readonly props: T;

    constructor(props: T) {
      this.props = Object.freeze(props);
    }

    equals(value?: ValueObject<T>): boolean {
      if (value == null || value == undefined) {
        return false;
      }

      if (this === value) {
        return true;
      }

      return JSON.stringify(this.props) === JSON.stringify(value.props);
    }
  }

```

Разные котики

```ts
import { ValueObject } from './ValueObject';

interface CatTypeProps {
  type: string;
  ageRequirement: number;
}

export abstract class CatType extends ValueObject<CatTypeProps> {
  abstract toString(): string;
  abstract isEligibleForAge(age: number): boolean;
}

export class ScientistCatType extends CatType {
  constructor() {
    super({ type: 'scientist', ageRequirement: 5 });
  }

  toString() {
    return this.props.type;
  }

  isEligibleForAge(age: number) {
    return age >= this.props.ageRequirement;
  }
}

export class FighterCatType extends CatType {
  constructor() {
    super({ type: 'fighter', ageRequirement: 2 });
  }

  toString() {
    return this.props.type;
  }

  isEligibleForAge(age: number) {
    return age >= this.props.ageRequirement;
  }
}

export class LazyCatType extends CatType {
  constructor() {
    super({ type: 'lazy', ageRequirement: 2 });
  }

  toString() {
    return this.props.type;
  }

  isEligibleForAge(age: number) {
    return age >= this.props.ageRequirement;
  }
}

```

Тестируем Entity

```test/Cat.test.ts
import { describe, it } from 'node:test';
import assert from 'node:assert';

import { Cat } from './Cat';
import { ScientistCatType, FighterCatType, LazyCatType } from '../valueObjects/CatType';

describe('Cat Domain Tests', () => {
  const today = new Date();
  const twoYearsAgo = new Date(today.getFullYear() - 2, today.getMonth(), today.getDate());
  const fiveYearsAgo = new Date(today.getFullYear() - 5, today.getMonth(), today.getDate());
  const sixYearsAgo = new Date(today.getFullYear() - 6, today.getMonth(), today.getDate());

  it('should calculate age correctly', () => {
    const cat = new Cat('1', 'test@example.com', 'Whiskers', fiveYearsAgo, new LazyCatType());
    assert.strictEqual(cat.age, 5);
  });

  it('should not register a kitten younger than 2 years', () => {
    const kitten = new Cat('2', 'kitten@example.com', 'Kitty', today, new LazyCatType());
    assert.strictEqual(kitten.isEligibleForRegistration(), false);
  });

  it('should register a lazy cat older than 2 years', () => {
    const cat = new Cat('3', 'lazycat@example.com', 'Lazy', twoYearsAgo, new LazyCatType());
    assert.strictEqual(cat.isEligibleForRegistration(), true);
  });

  it('should not register a scientist cat younger than 5 years', () => {
    const youngScientistCat = new Cat('4', 'youngscientist@example.com', 'Young Scientist', fiveYearsAgo, new ScientistCatType());
    assert.strictEqual(youngScientistCat.isEligibleForRegistration(), false);
  });

  it('should register a scientist cat older than 5 years', () => {
    const oldScientistCat = new Cat('5', 'oldscientist@example.com', 'Old Scientist', sixYearsAgo, new ScientistCatType());
    assert.strictEqual(oldScientistCat.isEligibleForRegistration(), true);
  });

  it('should register a fighter cat older than 2 years', () => {
    const fighterCat = new Cat('6', 'fighter@example.com', 'Fighter', twoYearsAgo, new FighterCatType());
    assert.strictEqual(fighterCat.isEligibleForRegistration(), true);
  });
});

```

Infrastructure Data Models

```
export interface CatDataModel {
    id: string;
    email: string;
    name: string;
    birthDate: string;
    type: string;
}
```

Infrastructure Data Models

```typescript
import { Cat } from '../entities/Cat';
import { CatDataModel } from '../dataModels/CatDataModel';
import { ScientistCatType, FighterCatType, LazyCatType, CatType } from '../valueobjects/CatType';

export class CatDataMapper {
  static toDomain(catDataModel: CatDataModel): Cat {
    const { id, email, name, birthDate, type } = catDataModel;
    let catType: CatType;

    switch (type) {
      case 'scientist':
        catType = new ScientistCatType();
        break;
      case 'fighter':
        catType = new FighterCatType();
        break;
      case 'lazy':
        catType = new LazyCatType();
        break;
      default:
        throw new Error('Unknown cat type');
    }

    return new Cat(id, email, name, new Date(birthDate), catType);
  }

  static toData(cat: Cat): CatDataModel {
    return {
      id: cat.id,
      email: cat.email,
      name: cat.name,
      birthDate: cat.birthDate.toISOString(),
      type: cat.type.toString(),
    };
  }
}

```

Interface Adapters Repository

```typescript
import { Cat } from '../entities/Cat';

export interface CatRepository {
  save(cat: Cat): Promise<void>;
  getAll(): Promise<Cat[]>;
}

```

```typescript
import { Cat } from '../entities/Cat';
import { CatRepository } from './CatRepository.interface';
import { CatDataMapper } from '../mappers/CatDataMapper';
import { CatDataModel } from '../dataModels/CatDataModel';

export class InMemoryCatRepository implements CatRepository {
  private cats: CatDataModel[] = [];

  async save(cat: Cat): Promise<void> {
    const catDataModel = CatDataMapper.toData(cat);
    this.cats.push(catDataModel);
  }

  async getAll(): Promise<Cat[]> {
    return this.cats.map(catDataModel => CatDataMapper.toDomain(catDataModel));
  }
}

```

Use Cases
RegisterCat

```ts
import { Cat, CatType } from '../entities/Cat';
import { CatRepository } from '../repositories/CatRepository.interface';
import { NotificationService } from '../services/NotificationService';
import { CatRegistrationError } from '../errors/CatRegistrationError';

export class RegisterCat {
  constructor(
    private catRepository: CatRepository,
    private notificationService: NotificationService
  ) {}

  async execute(data: { id: string; email: string; name: string; birthDate: Date; type: CatType }) {
    const cat = new Cat(data.id, data.email, data.name, data.birthDate, data.type);

    if (!cat.isEligibleForRegistration()) {
      throw new CatRegistrationError('Cat is not eligible for registration');
    }

    await this.catRepository.save(cat);
    this.notificationService.notify(cat);
  }
}

```

Use Cases
GetAllCats

```ts
import { CatRepository } from '../repositories/CatRepository.interface';
import { CatDTO } from '../dto/CatDTO';

export class GetAllCats {
  constructor(private catRepository: CatRepository) {}

  async execute(): Promise<CatDTO[]> {
    const cats = await this.catRepository.getAll();
    return cats.map(cat => new CatDTO(cat));
  }
}

```

Use Cases Тестирование

```test_RegisterCat.ts
import assert from 'node:assert';
import { describe, it } from 'node:test';

import { RegisterCat } from './RegisterCat';
import { InMemoryCatRepository } from '../repositories/InMemoryRepository';
import { NotificationService } from '../services/NotificationService';
import { CatRegistrationError } from '../errors/CatRegistrationError';
import { ScientistCatType, LazyCatType } from '../valueObjects/CatType';

describe('RegisterCat Use Case Tests', () => {
  it('should register a valid cat', async () => {
    const catRepository = new InMemoryCatRepository();
    const notificationService = new NotificationService([]);
    const registerCat = new RegisterCat(catRepository, notificationService);

    const catData = {
      id: '1',
      email: 'valid@example.com',
      name: 'Valid Cat',
      birthDate: new Date(new Date().getFullYear() - 3, 0, 1),
      type: new LazyCatType(),
    };

    await registerCat.execute(catData);

    const cats = await catRepository.getAll();
    assert.strictEqual(cats.length, 1);
    assert.strictEqual(cats[0].email, catData.email);
  });

  it('should not register a kitten', async () => {
    const catRepository = new InMemoryCatRepository();
    const notificationService = new NotificationService([]);
    const registerCat = new RegisterCat(catRepository, notificationService);

    const kittenData = {
      id: '2',
      email: 'kitten@example.com',
      name: 'Kitten',
      birthDate: new Date(),
      type: new LazyCatType(),
    };

    await assert.rejects(
      async () => {
        await registerCat.execute(kittenData);
      },
      CatRegistrationError
    );
  });

  it('should not register a scientist cat younger than 5 years', async () => {
    const catRepository = new InMemoryCatRepository();
    const notificationService = new NotificationService([]);
    const registerCat = new RegisterCat(catRepository, notificationService);

    const youngScientistData = {
      id: '3',
      email: 'youngscientist@example.com',
      name: 'Young Scientist',
      birthDate: new Date(new Date().getFullYear() - 4, 0, 1),
      type: new ScientistCatType(),
    };

    await assert.rejects(
      async () => {
        await registerCat.execute(youngScientistData);
      },
      CatRegistrationError
    );
  });
});

```

Interface Adapters Notification Service

```ts
import { Cat } from '../entities/Cat';

export interface NotificationStrategy {
  sendNotification(cat: Cat): void;
}

```

```ts
import { Cat } from '../entities/Cat';
import { NotificationStrategy } from './NotificationStrategy.interface';

export class EmailNotificationStrategy implements NotificationStrategy {
  sendNotification(cat: Cat): void {
    console.log(`Sending email to ${cat.email}: Welcome, ${cat.name}!`);
  }
}

```

```ts
import { Cat } from '../entities/Cat';
import {NotificationStrategy } from './NotificationStrategy.interface';

export class NotificationService {
  private strategies: NotificationStrategy[];

  constructor(strategies: NotificationStrategy[]) {
    this.strategies = strategies;
  }

  notify(cat: Cat): void {
    this.strategies.forEach(strategy => strategy.sendNotification(cat));
  }
}

```

Interface Adapters

CatController

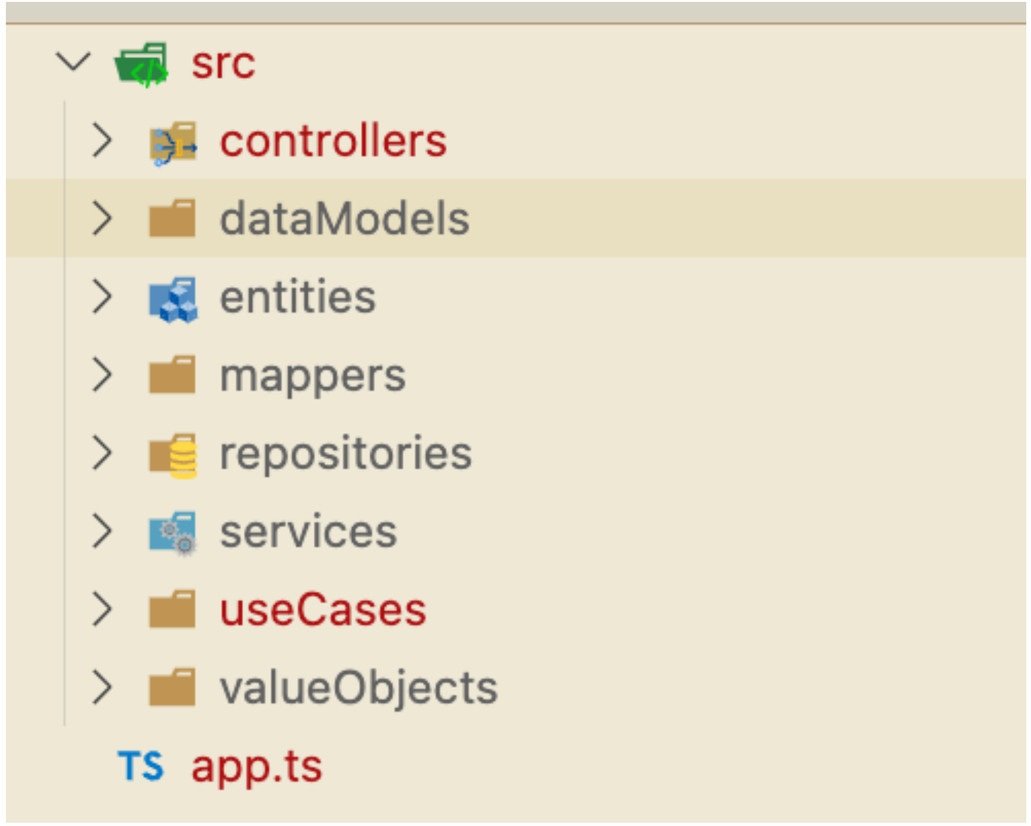
```ts
import { Request, Response } from 'express';

import { RegisterCat } from '../useCases/RegisterCat';
import { GetAllCats } from '../useCases/GetAllCats';
import { BusinessError } from '../errors/BusinessError';
import { CatTypeFactory } from '../factories/CatTypeFactory';

export class CatController {
  constructor(
    private registerCat: RegisterCat,
    private getAllCats: GetAllCats
  ) {}

  async registerCatHandler(req: Request, res: Response) {
    try {
      const { id, email, name, birthDate, type } = req.body;
      const catType = CatTypeFactory.create(type);
      await this.registerCat.execute({ id, email, name, birthDate: new Date(birthDate), type: catType });
      res.status(201).send({ message: 'Cat registered successfully' });
    } catch (error) {
      if (error instanceof BusinessError) {
        res.status(error.statusCode).send({ title: 'Invalid Request', detail: error.message });
      } else {
        res.status(500).send({ title: 'Internal Server Error', detail: error.message });
      }
    }
  }

  async getAllCatsHandler(req: Request, res: Response) {
    try {
      const cats = await this.getAllCats.execute();
      res.status(200).json(cats);
    } catch (error) {
      res.status(500).send({ title: 'Internal Server Error', detail: error.message });
    }
  }
}

```

Frameworks & Drivers

app.ts

```ts
const catRepository = new InMemoryCatRepository();
const notificationService = new NotificationService([
  new EmailNotificationStrategy(),
  new SMSNotificationStrategy()
]);

const registerCat = new RegisterCat(catRepository, notificationService);
const getAllCats = new GetAllCats(catRepository);

const catController = new CatController(registerCat, getAllCats);

const app = express();
app.use(express.json());

app.post('/cats', (req, res) => catController.registerCatHandler(req, res));
app.get('/cats', (req, res) => catController.getAllCatsHandler(req, res));

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Файловая структура



И это всё?

nTier, IoC

Code Blame 31 lines (25 loc) · 1.07 KB Raw

```ts
import { inject, injectable } from 'inversify';
import { Request, Response } from 'express';

import { BaseController } from '../../libs/rest/index.js';
import { CreateUserRequest } from './types/create-user.type.js';
import { ParamUserID } from './types/param-userid.type.js';
import { Component } from '../../types/components.enum.js';
import { UserService } from './user.service.js';


@injectable()
export class UserController extends BaseController {
  constructor(
    @inject(Component.UserService) private readonly userService: UserService,
  ) {
    super();

    this.addRoute({ path: '/register', method: 'post', handler: this.create })
    this.addRoute({ path: '/:userId', method: 'get', handler: this.show });
  }

  public async create({ body }: CreateUserRequest, res: Response): Promise<void> {
    const user = await this.userService.register(body);
    this.ok(res, user.toPOJO());
  }

  public async show({ params }: Rec
    const user = await this.userSer
    this.ok(res, user.toPOJO());
  }
}
```

Code Blame 24 lines (19 loc) · 743 Bytes Raw

```ts
import { inject, injectable } from 'inversify';

import { BaseMemoryRepository } from '../../libs/data-access/index.js';
import { UserEntity } from './user.entity.js';
import { UserFactory } from './user.factory.js';
import { Component } from '../../types/components.enum.js';

@injectable()
export class UserRepository extends BaseMemoryRepository<UserEntity> {
  constructor(@inject(Component.UserFactory) factory: UserFactory) {
    super(factory);
  }

  public async findByEmail(email: string): Promise<UserEntity | null> {
    const entities = Array.from(this.entities.values());
    const user = entities.find((entity) => entity.email === email);

    if (! user) {
      return null;
    }

    return this.factory.create(user);
  }
}
```

Материалы

- Пример nTier архитектуры — https://github.com/AntonovIgor/it-conf30
- Блог в Telegram — https://t.me/antonovjs
- Вопросы? https://t.me/antonov_i

Выводы

Не универсальное решение

Явное разделение ответственности

Проще тестировать

Гибкая разработка

Проще мигрировать

Т БАНК

Спасибо!