CQRS in NestJS

Viktoria Dolzhenko

Обо мне

TeamLead с более 10 летним опытом

Обо мне

TeamLead с более 10 летним опытом

Строю приложения с нуля последние 7 лет

Обо мне

TeamLead с более 10 летним опытом

Строю приложения с нуля последние 7 лет

На практике знаю что такое CQRS

Планы

1. Немного теории

Планы

1. Немного теории
2. Практика

CQRS – это архитектурный стиль

Layers

CQRS – это архитектурный стиль

Layers

Component Based

CQRS – это архитектурный стиль

Layers

Component Based

DDD (Domain Driven Design)

CQRS – это архитектурный стиль

Layers

Component Based

DDD (Domain Driven Design)

REST

CQRS – это архитектурный стиль

Layers

Component Based

DDD (Domain Driven Design)

REST

CQRS

CQS Principle

Command-Query Separation Principle

CQS Principle

Command-Query Separation Principle

Commands – изменяют состояние системы

CQS Principle

Command-Query Separation Principle

Commands – изменяют состояние системы

Queries – возвращают значение,
не имеют побочных эффектов

CQRS

Зачем

- Производительность

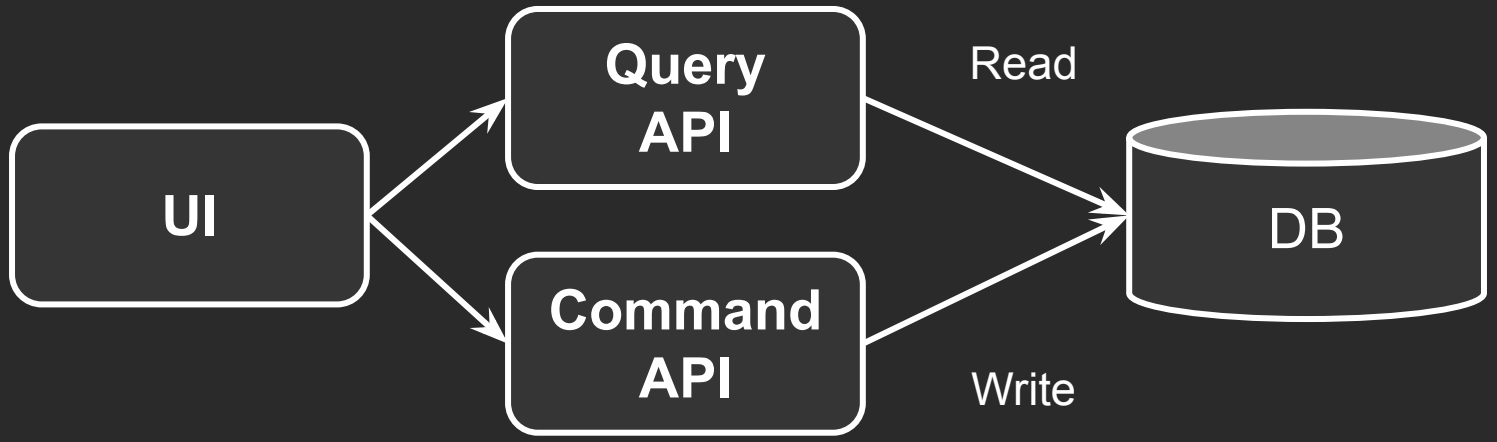
Зачем

- Производительность
- Масштабируемость

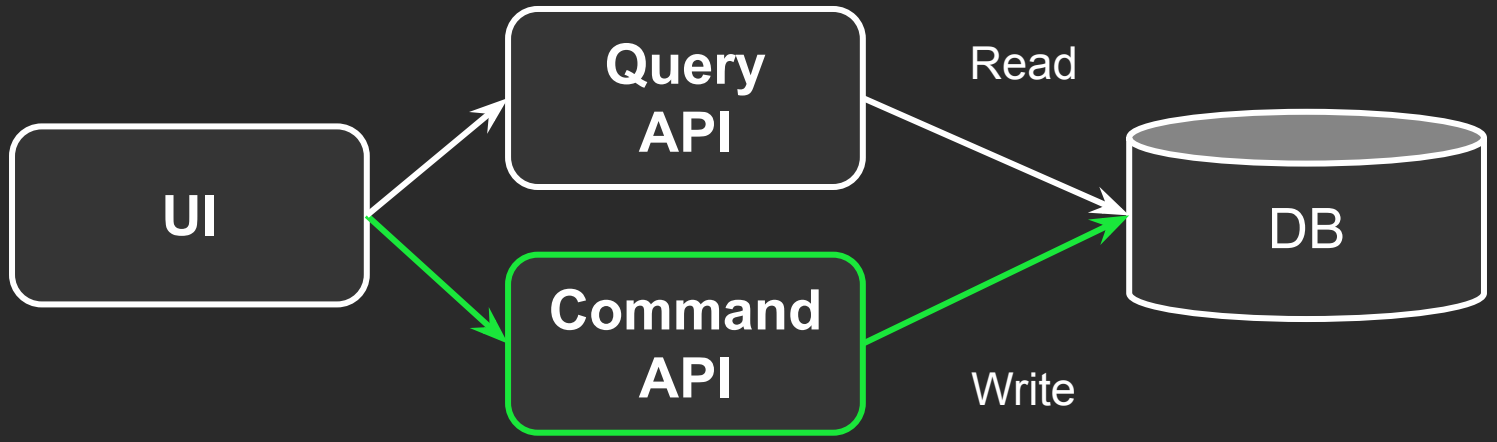
Зачем

- Производительность
- Масштабируемость
- Безопасность

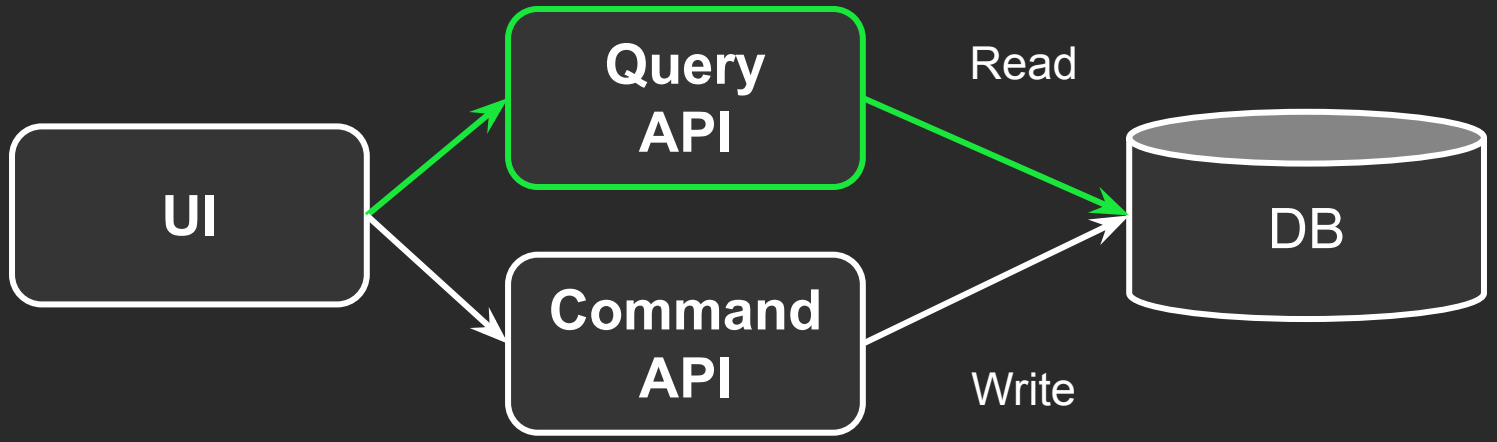
CQRS



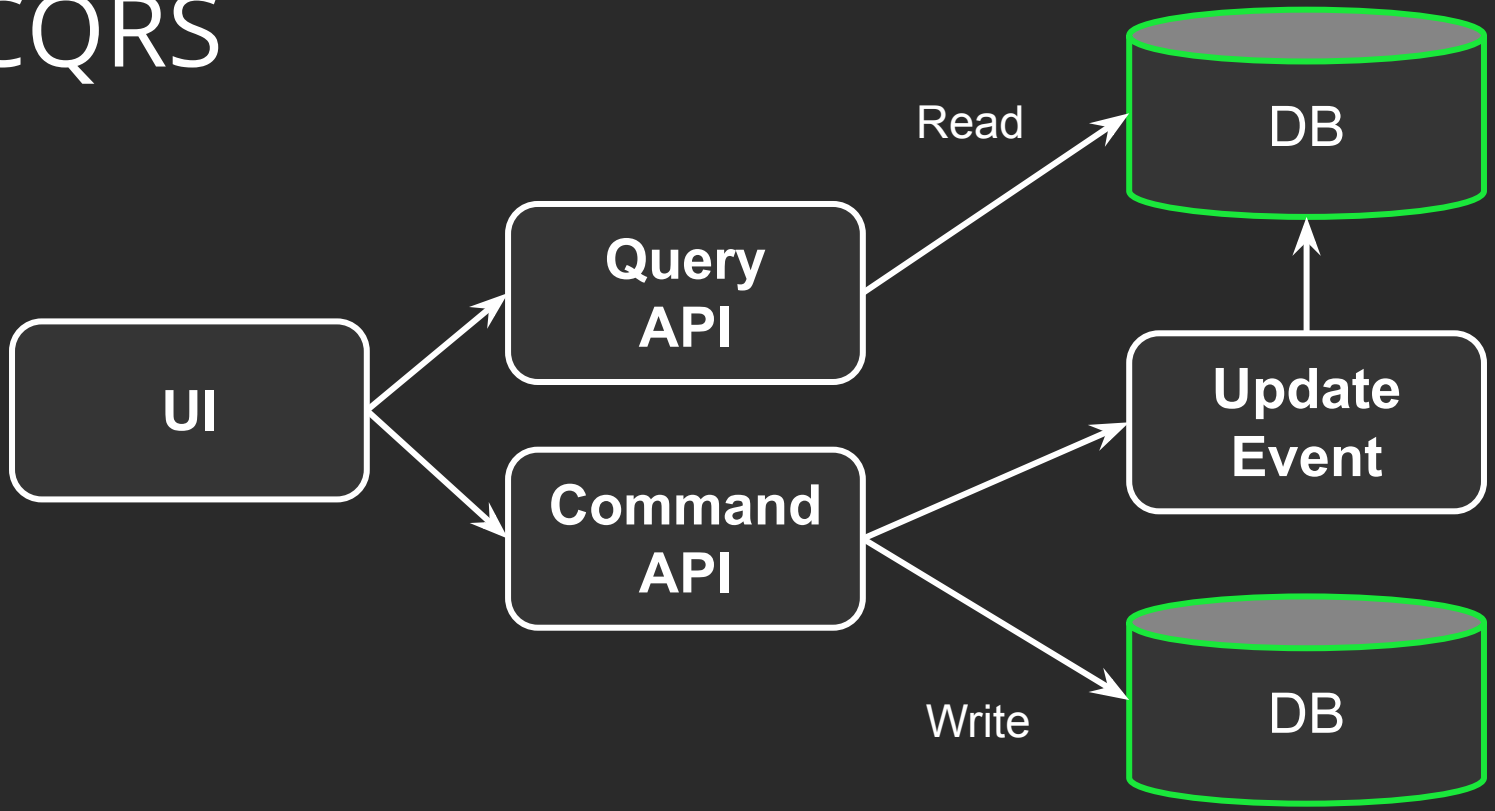
CQRS



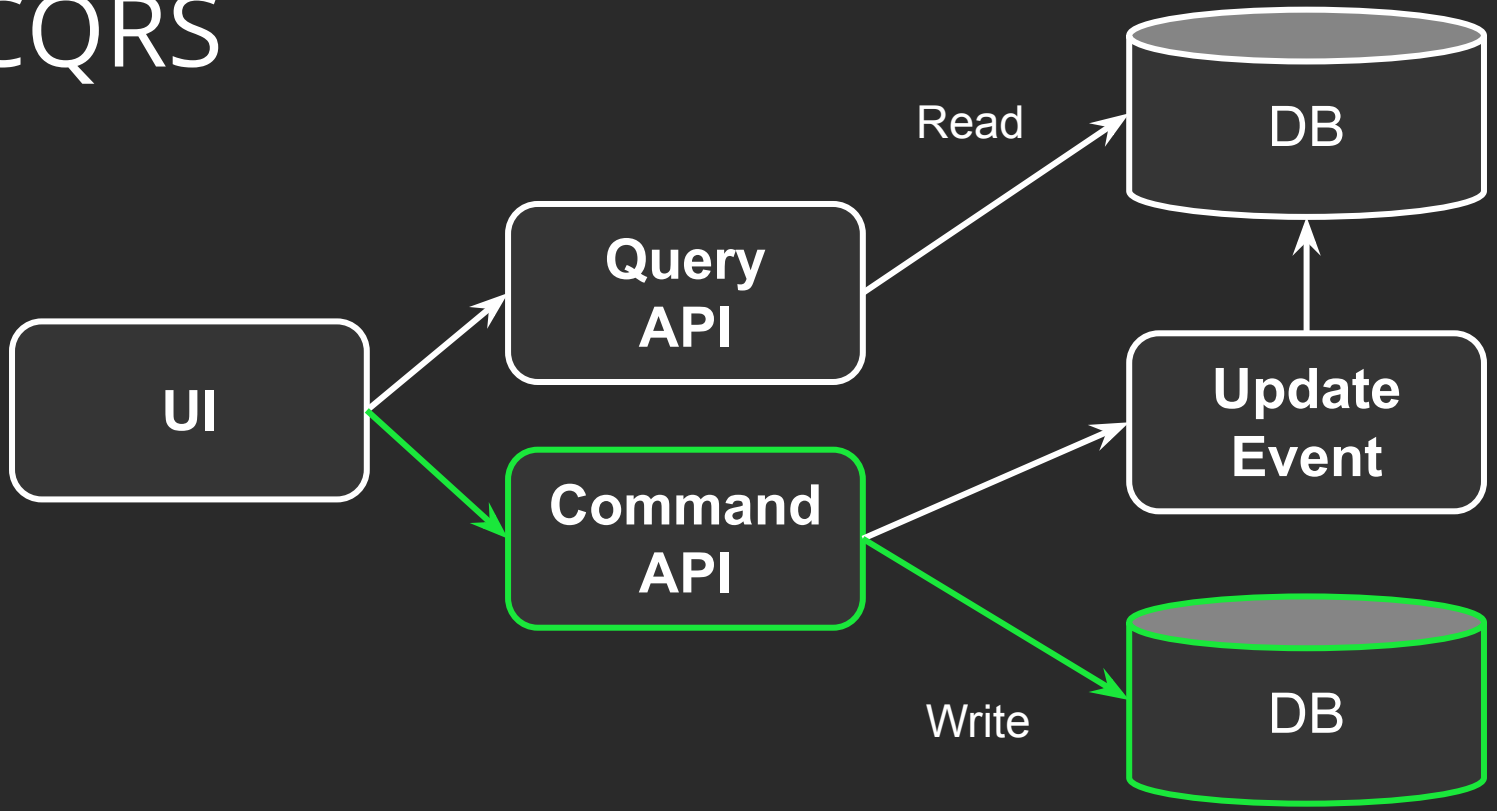
CQRS



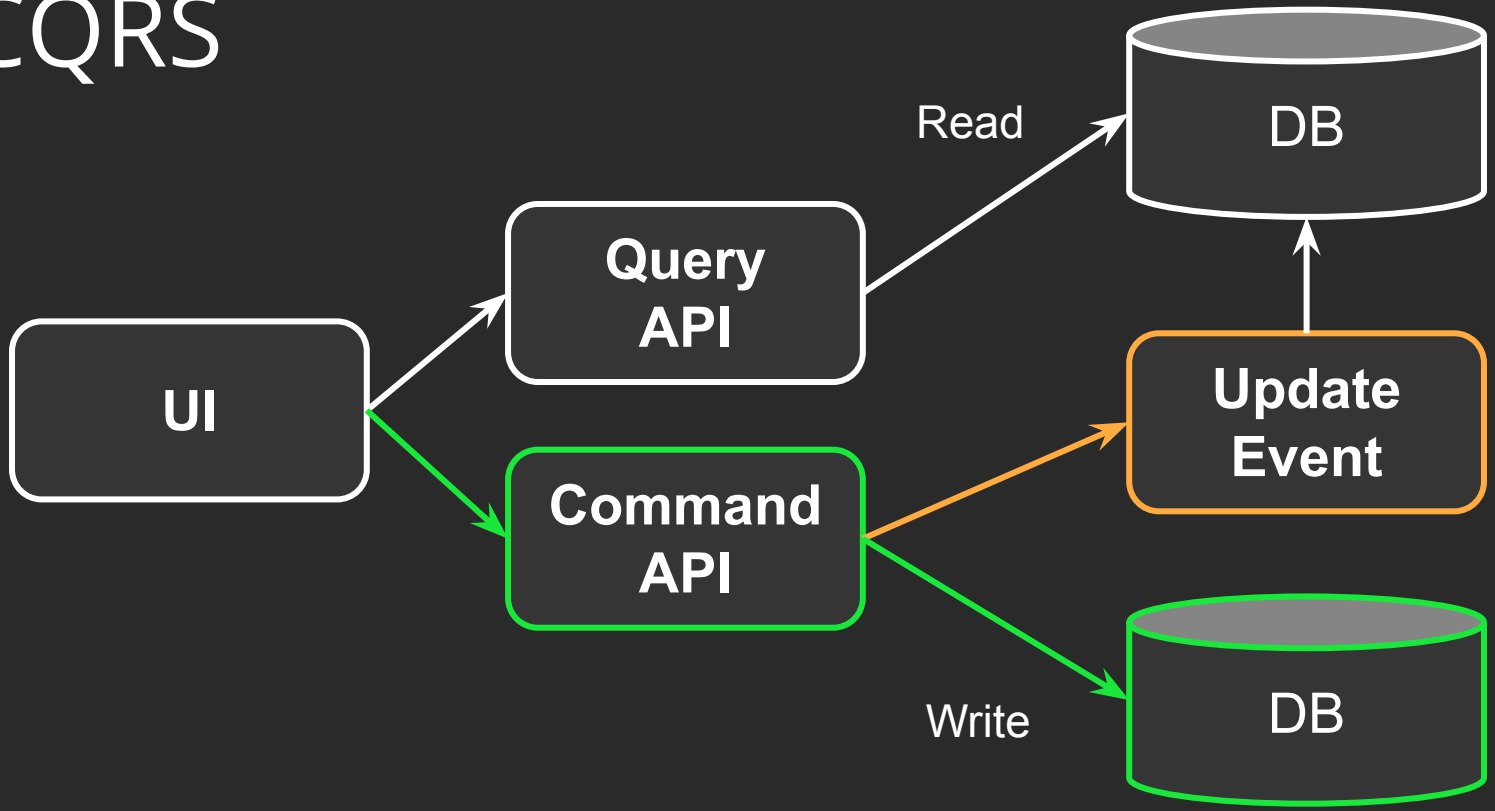
CQRS



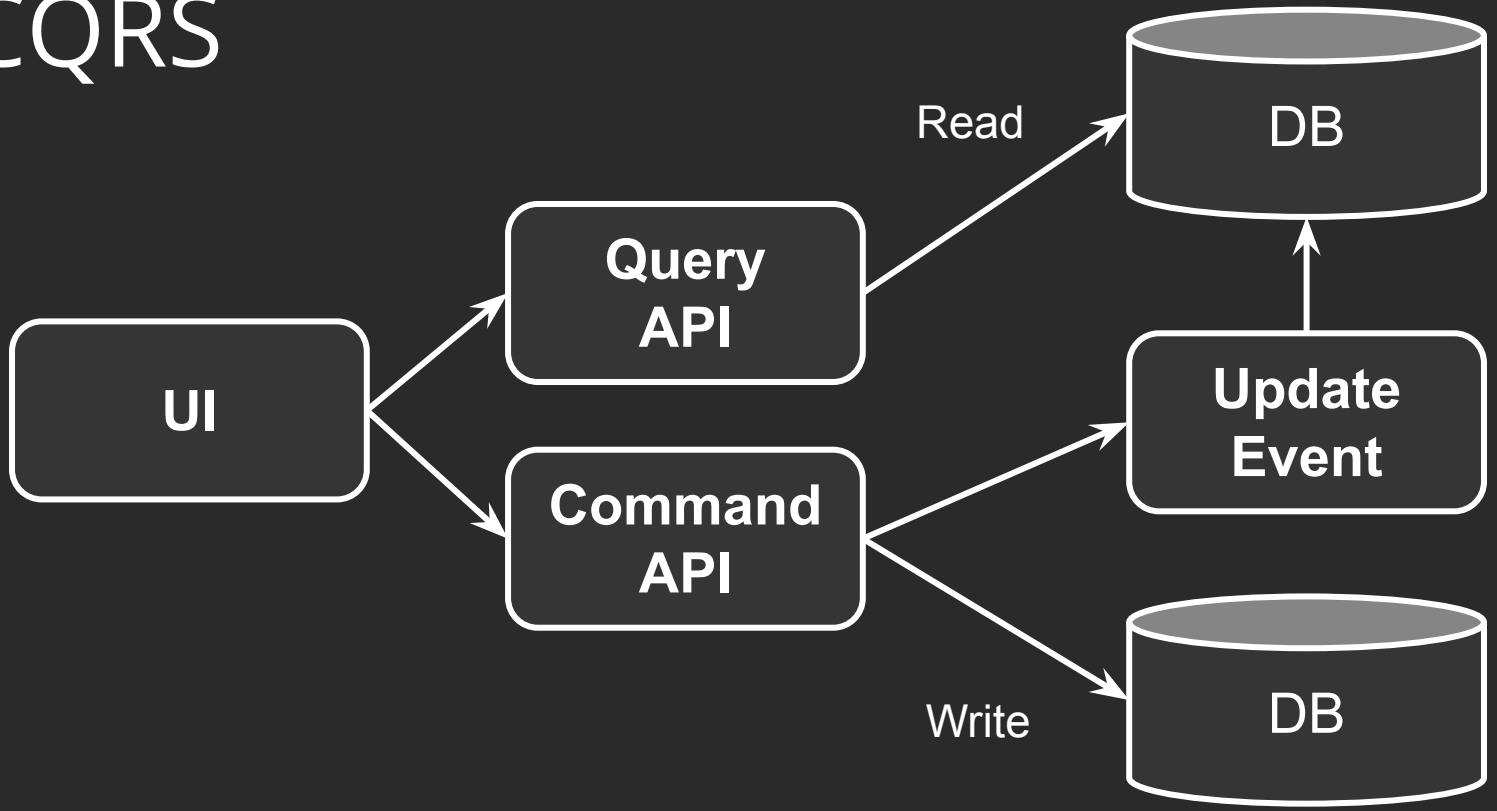
CQRS

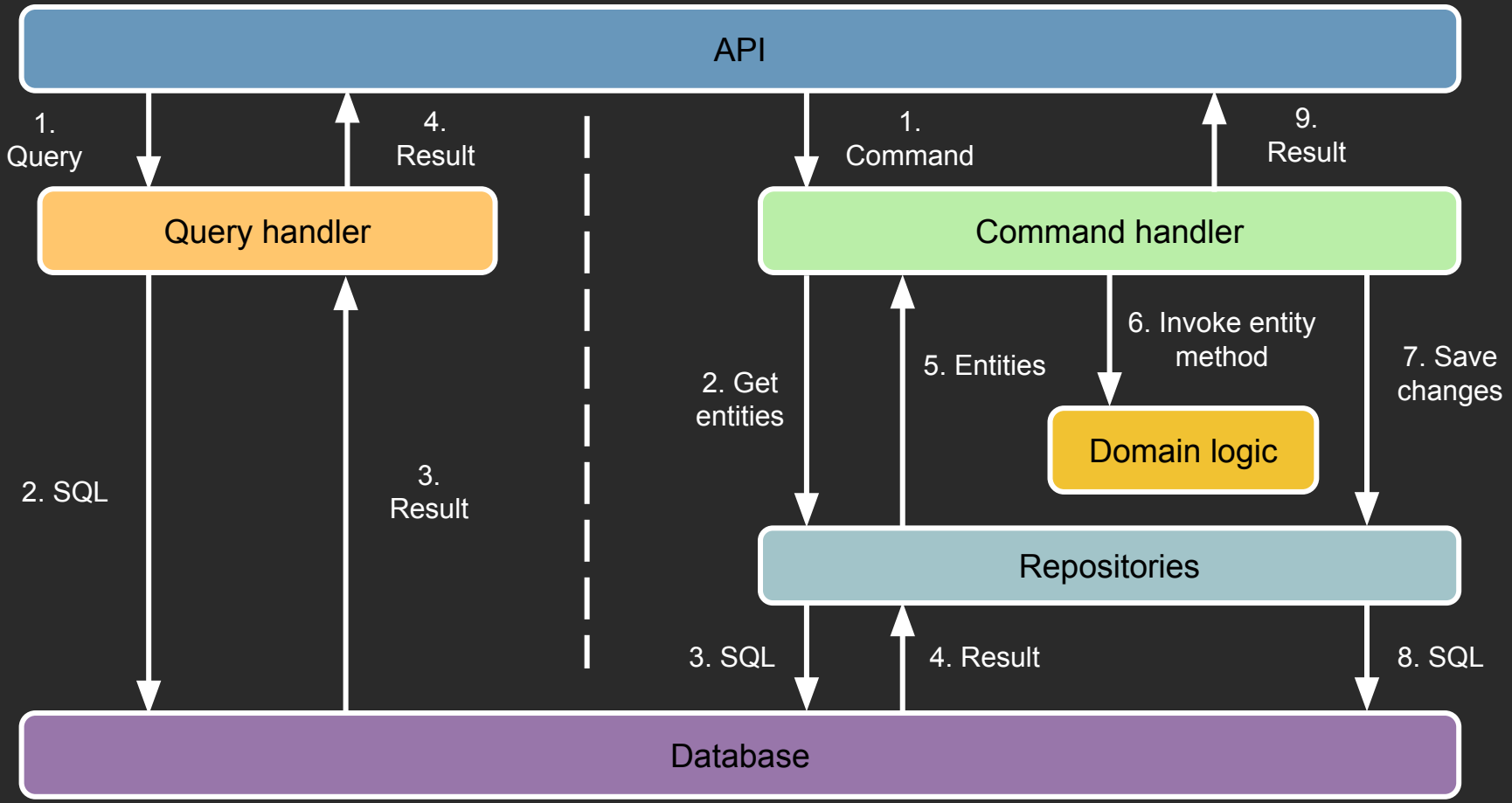


CQRS

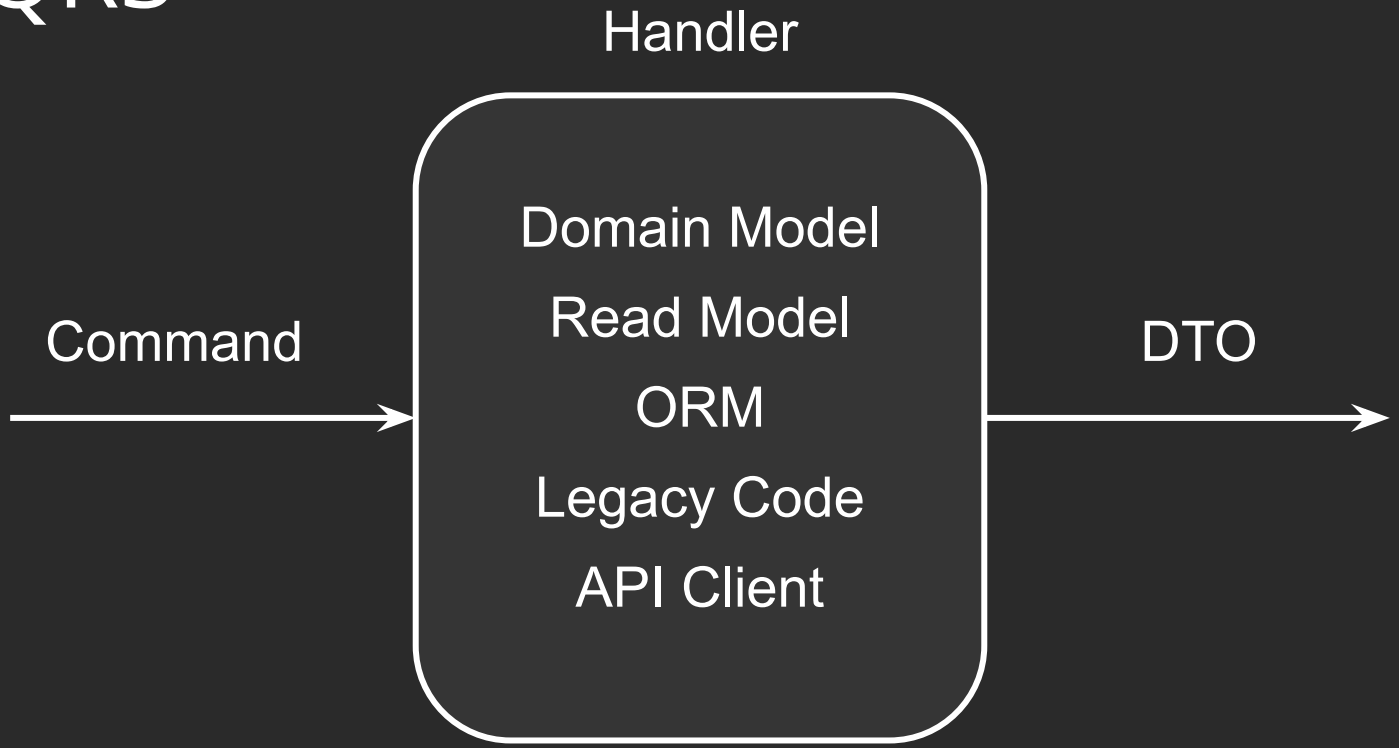


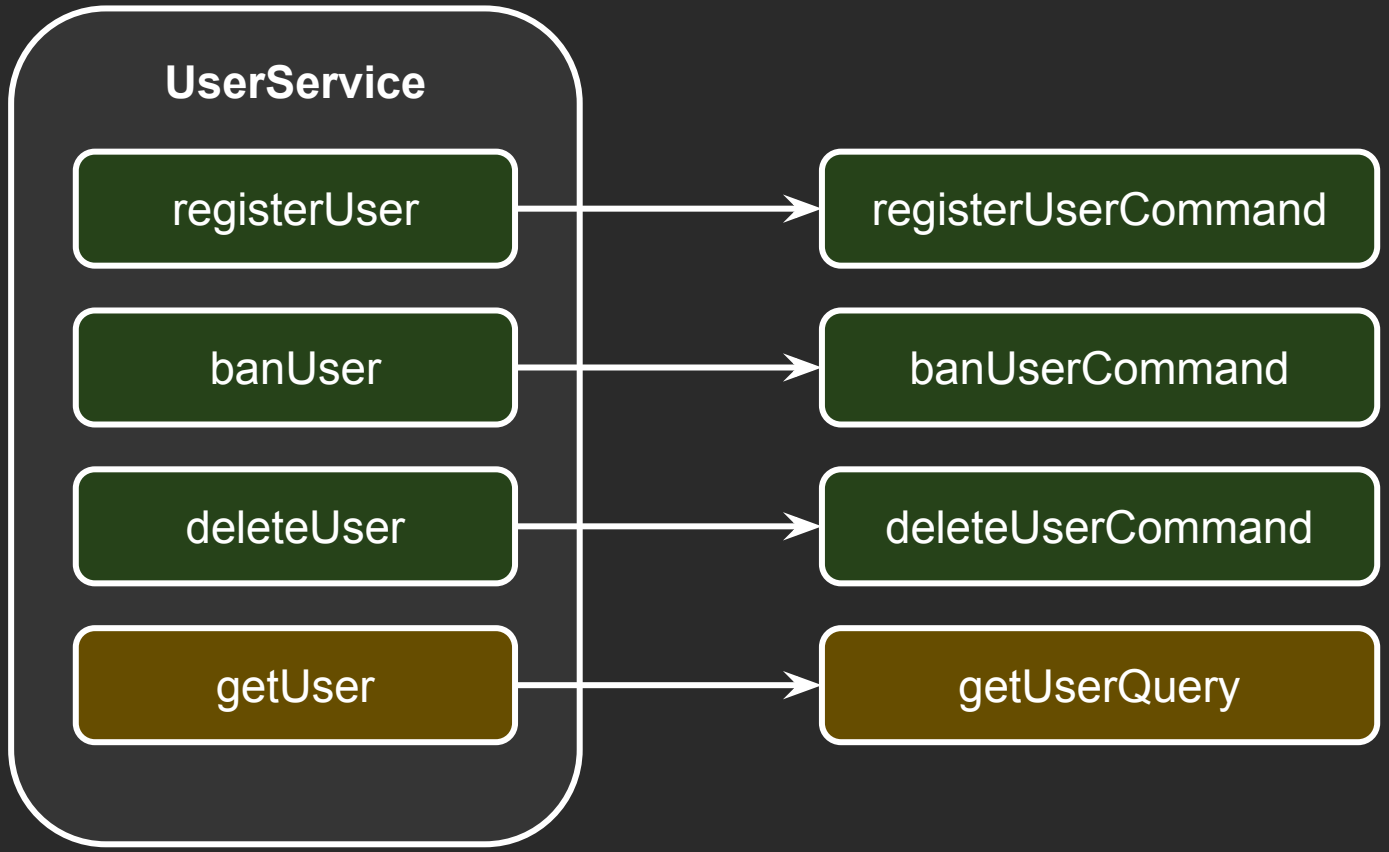
CQRS



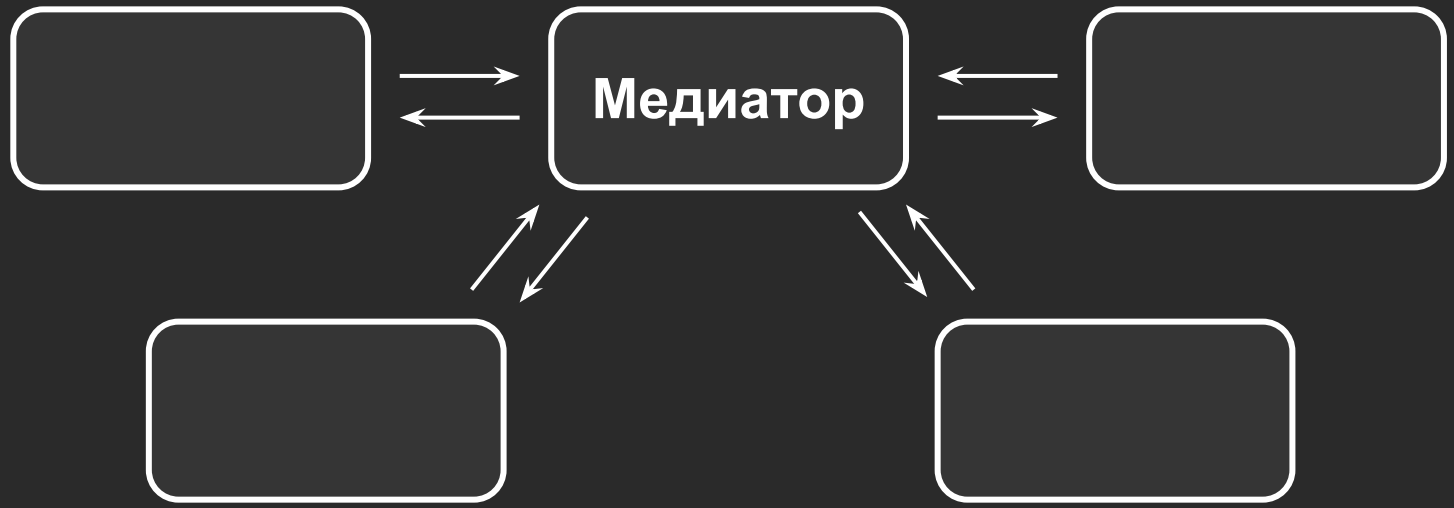

API
1.
Query
4.
Result
Query handler
2. SQL
3.
Result
1.
Command
9.
Result
Command handler
2. Get
entities
5. Entities
6. Invoke entity
method
7. Save
changes
Domain logic
Repositories
3. SQL
4. Result
8. SQL
Database

CQRS


Handler
Domain Model
Read Model
ORM
Legacy Code
API Client
Command
DTO


UserService
registerUser
banUser
deleteUser
getUser
registerUserCommand
banUserCommand
deleteUserCommand
getUserQuery

Медиатор (посредник)



Подключение

```
$ npm i @nestjs/cqrs
```

/src/app/auth/auth.module.ts

```
@Module({
    imports: [
        DatabaseModule,
+       CqrsModule,
    ],
    providers: [
-       AuthService,
+       RegisterUserHandler,
    ],
    controllers: [
        AuthController,
    ],
})
export class AuthModule {}
```

```
export class CreateBookCommand {
    constructor(
        public readonly userId: string,
        public readonly title: string,
    ) {
    }
}
```

```ts
@CommandHandler(CreateBookCommand)
export class CreateBookHandler implements ICommandHandler<CreateBookCommand> {
    constructor(
        private readonly bookRepository: BookRepository,
    ) {
    }

    async execute(command: CreateBookCommand): Promise<void> {
        return await this.bookRepository.insert({
            userId: command.userId,
            title: command.title,
        });
    }
}
```

```ts
@CommandHandler(CreateBookCommand)
export class CreateBookHandler implements ICommandHandler<CreateBookCommand> {
    constructor(
        private readonly bookRepository: BookRepository,
    ) {
    }

    async execute(command: CreateBookCommand): Promise<void> {
        return await this.bookRepository.insert({
            userId: command.userId,
            title: command.title,
        });
    }
}
```

```ts
@CommandHandler(CreateBookCommand)
export class CreateBookHandler implements ICommandHandler<CreateBookCommand> {
     constructor(
          private readonly bookRepository: BookRepository,
     ) {
     }

     async execute(command: CreateBookCommand): Promise<void> {
          return await this.bookRepository.insert({
               userId: command.userId,
               title: command.title,
          });
     }
}
```

```
export class BooksController {

    constructor(private readonly commandBus: CommandBus) {
    }

    @Post()
    async create(@Body() request: CreateBookRequestDto): Promise<BaseResponse> {

        await this.commandBus
            .execute(new CreateBookCommand(
                request.userId,
                request.title));

        return new SuccessResponse();
    }
}
```

```
export class BooksController {

    constructor(private readonly commandBus: CommandBus) {
    }

    @Post()
    async create(@Body() request: CreateBookRequestDto): Promise<BaseResponse> {

        await this.commandBus
            .execute(new CreateBookCommand(
                request.userId,
                request.title));

        return new SuccessResponse();
    }
}
```

Providers

- CommandBus
- QueryBus

Providers

- CommandBus
- QueryBus
- EventBus

Регистрация пользователя

Регистрация пользователя

1. Получает код подтверждения

Регистрация пользователя

1. Получает код подтверждения
2. Подтверждает код

Регистрация пользователя

1. Сохранить статус в БД

Регистрация пользователя

1. Сохранить статус в БД
2. Начислить бонусы

Регистрация пользователя

1. Сохранить статус в БД
2. Начислить бонусы
3. Отправить письмо

/src/app/auth/auth.service.ts

```
async confirm(email: string, code: string): Promise<void> {
    {...}
    await this.userService.register(user);
    await this.walletService.addBonus(user.id, '10');
    await this.notificationService.sendEmail(user.email, 'Registered');
    return void 0;
}
```

/src/app/auth/auth.service.ts

```
async confirm(email: string, code: string): Promise<void> {
    {...}
    await this.userService.register(user);
    await this.walletService.addBonus(user.id, '10');
    await this.notificationService.sendEmail(user.email, 'Registered');
    return void 0;
}
```

/src/app/auth/auth.service.ts

```
async confirm(email: string, code: string): Promise<void> {
    {...}
    await this.userService.register(user);
    await this.walletService.addBonus(user.id, '10');
    await this.notificationService.sendEmail(user.email, 'Registered');
    return void 0;
}
```

/src/common/user/user.service.ts

```
async register(user: User): Promise<void> {

  {...}

  user.status = UserStatus.Registered;
  await this.userRepository.save(user);

  await this.walletService.addBonus(user.id, '10');

  await this.notificationService.sendEmail(user.email, 'Registered');
}
```

/src/common/user/user.service.ts

```
async register(user: User): Promise<void> {

    {...}

    user.status = UserStatus.Registered;
    await this.userRepository.save(user);

    await this.walletService.addBonus(user.id, '10');

    await this.notificationService.sendEmail(user.email, 'Registered');
}
```

/src/common/user/user.service.ts

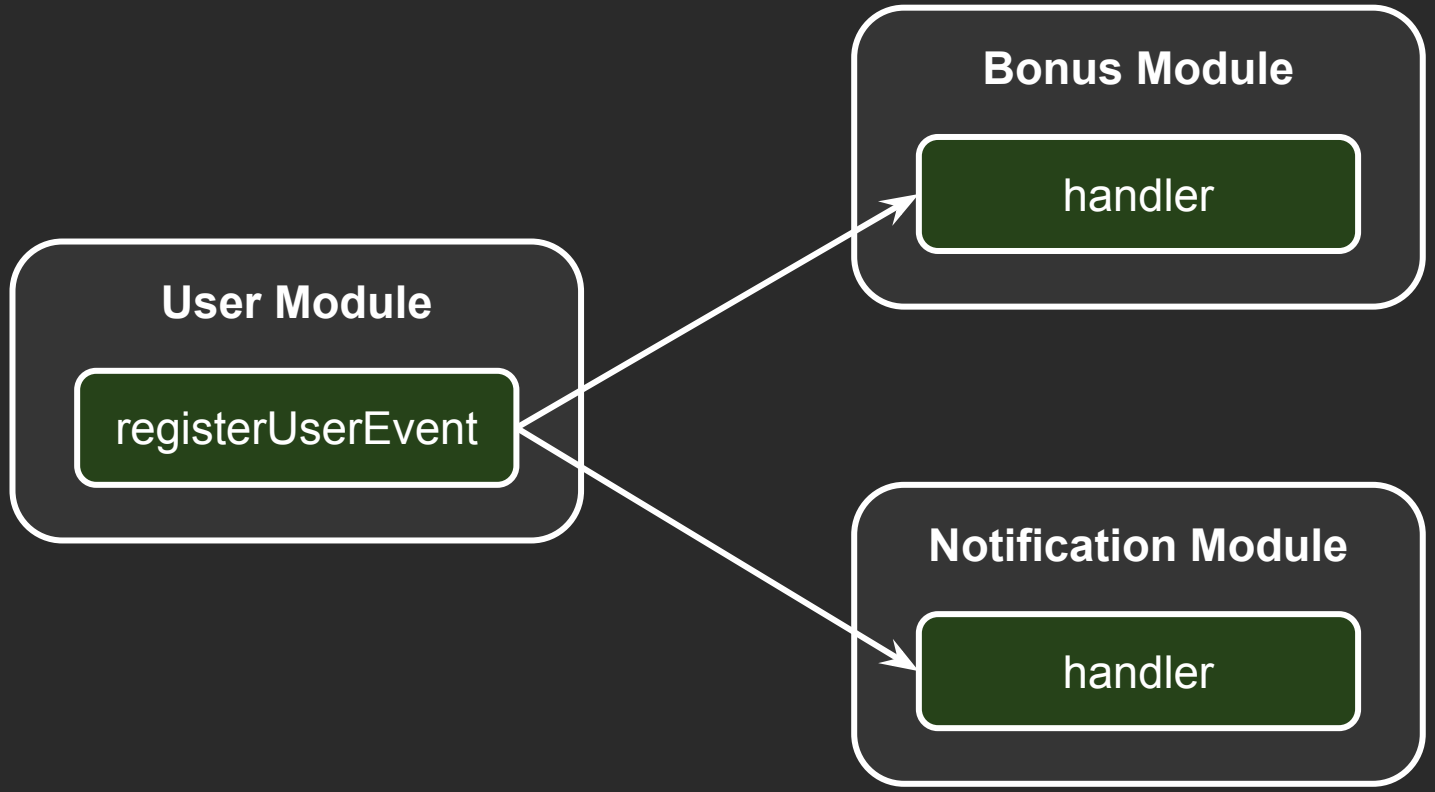
```
async register(user: User): Promise<void> {

    {...}

    user.status = UserStatus.Registered;
    await this.userRepository.save(user);

    await this.walletService.addBonus(user.id, '10');

    await this.notificationService.sendEmail(user.email, 'Registered');
}
```


Bonus Module
handler
User Module
registerUserEvent
Notification Module
handler

```
export class RegisterUserEvent {
    constructor(
        public readonly userId: string,
    ) {
    }
}
```

```
@CommandHandler(RegisterUserCommand)
export class RegisterUserHandler implements ICommandHandler<RegisterUserCommand>
{
     constructor(
          private readonly userRepository: UserRepository,
          private readonly eventBus: EventBus,
     ) {
     }

     async execute(command: RegisterUserCommand): Promise<void> {
          user.status = UserStatus.Registered;
          await this.userRepository.save(user);

          void this.eventBus.publish(new RegisterUserEvent(user.id));
     }
}
```

```
@CommandHandler(RegisterUserCommand)
export class RegisterUserHandler implements ICommandHandler<RegisterUserCommand>
{
     constructor(
          private readonly userRepository: UserRepository,
          private readonly eventBus: EventBus,
     ) {
     }

     async execute(command: RegisterUserCommand): Promise<void> {
          user.status = UserStatus.Registered;
          await this.userRepository.save(user);

          void this.eventBus.publish(new RegisterUserEvent(user.id));
     }
}
```

```typescript
@EventsHandler(RegisterUserEvent)
export class RegisterUserHandler implements IEventHandler<RegisterUserEvent> {
    constructor(
        private readonly commandBus: CommandBus,
        private readonly userRepository: UserRepository,
    ) {
    }

    async handle(event: RegisterUserEvent): Promise<void> {
        const user = await this.userRepository.getById(event.userId);

        return await this.commandBus.execute(
            new NotifyCommand(
                user!.email,
                'Registered',
                'You have successfully registered'));
    }
}
```

```
@EventsHandler(RegisterUserEvent)
export class RegisterUserHandler implements IEventHandler<RegisterUserEvent> {
     constructor(
          private readonly commandBus: CommandBus,
          private readonly userRepository: UserRepository,
     ) {
     }

     async handle(event: RegisterUserEvent): Promise<void> {
          const user = await this.userRepository.getById(event.userId);

          return await this.commandBus.execute(
               new NotifyCommand(
                    user!.email,
                    'Registered',
                    'You have successfully registered'));
     }
}
```

```
@EventsHandler(RegisterUserEvent)
export class RegisterUserHandler implements IEventHandler<RegisterUserEvent> {
     constructor(
          private readonly commandBus: CommandBus,
          private readonly userRepository: UserRepository,
     ) {
     }

     async handle(event: RegisterUserEvent): Promise<void> {
          const user = await this.userRepository.getById(event.userId);

          return await this.commandBus.execute(
               new NotifyCommand(
                    user!.email,
                    'Registered',
                    'You have successfully registered'));
     }
}
```

Книжный магазин

Спасибо за внимание

https://github.com/waksund/cqrs_ts