

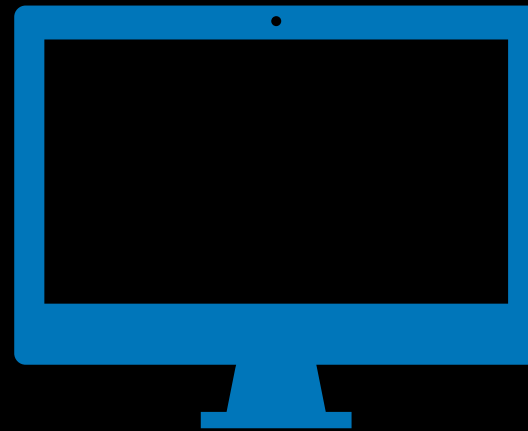
Backend for Frontend with



Дмитрий Фукс
Ведущий Frontend-разработчик
Maximaster
d.fuks@maximaster.ru

- Проблемы, которые привели к выбору паттерна BFF (Api gateway)
- Обзор возможностей фреймворка NestJS

SSR



HTML, JS, CSS

Core backend
(Symfony / Laravel / Yii / Bitrix / etc.)

JSON

CSV

XML

GraphQL

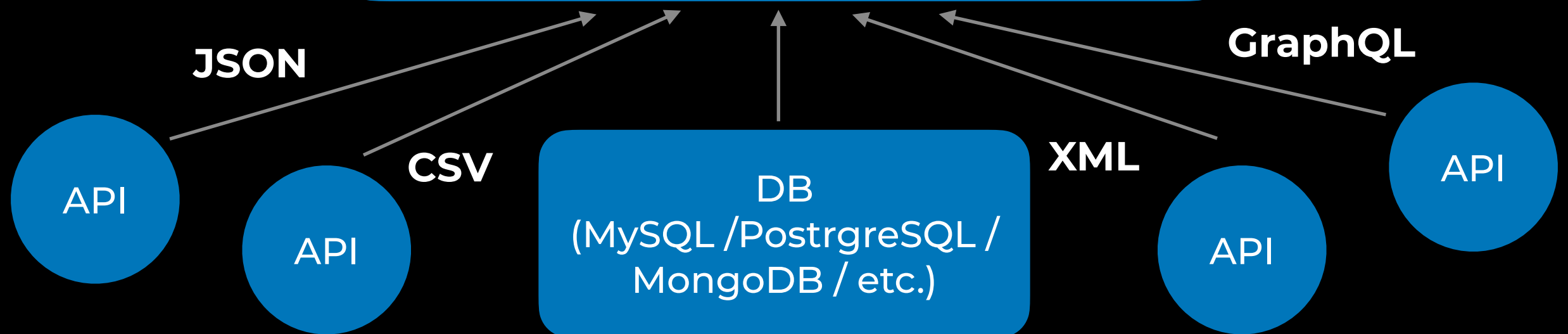
API

API

DB
(MySQL / PostgreSQL /
MongoDB / etc.)

API

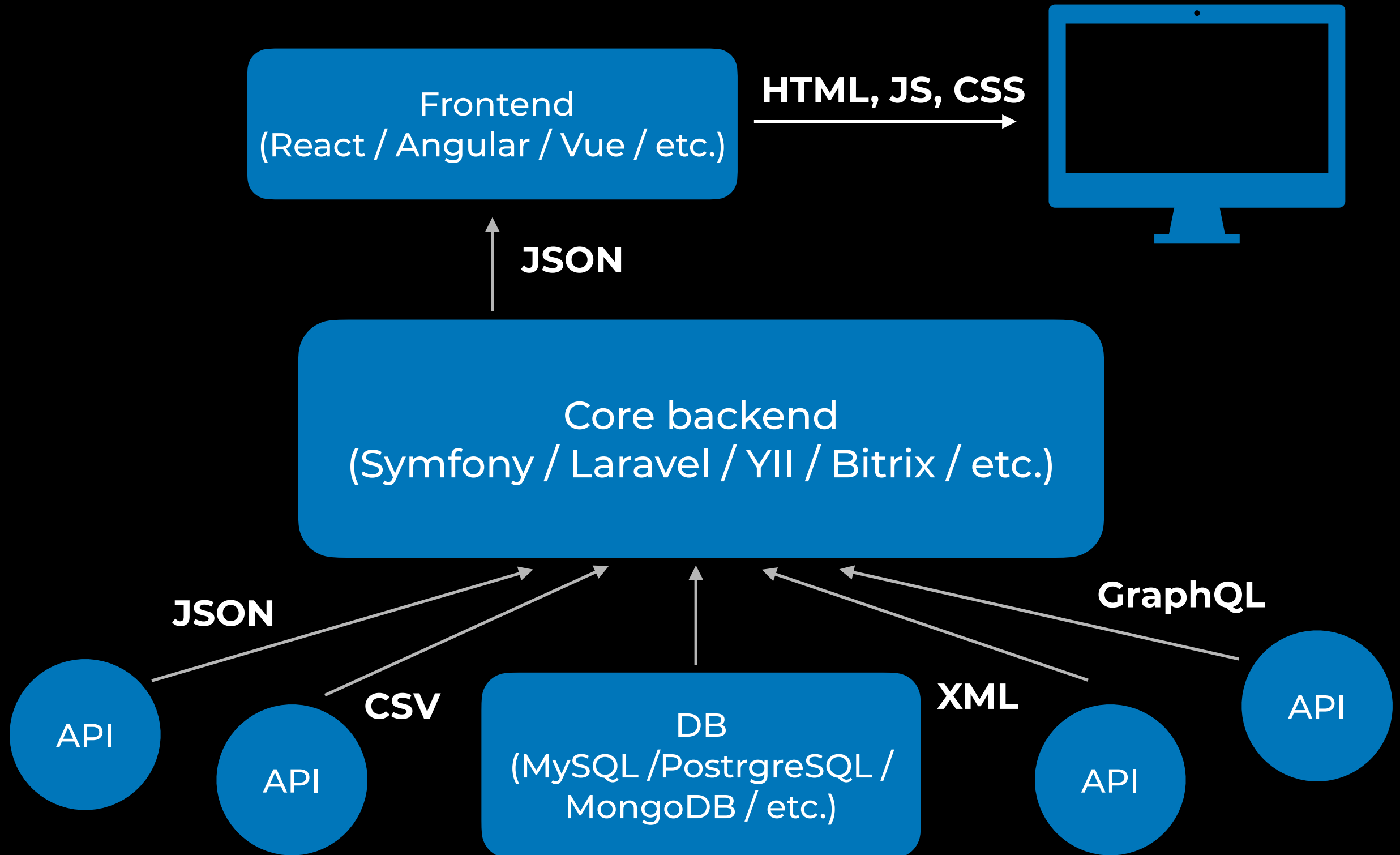
API



Проблемы

- На backend-разработчика возлагается ответственность за представление данных (лишняя логика, малая гибкость)
- У frontend-разработчика отсутствует согласованность между HTML, JS и CSS
- Backend'у приходится выступать как прокси к другим сервисам

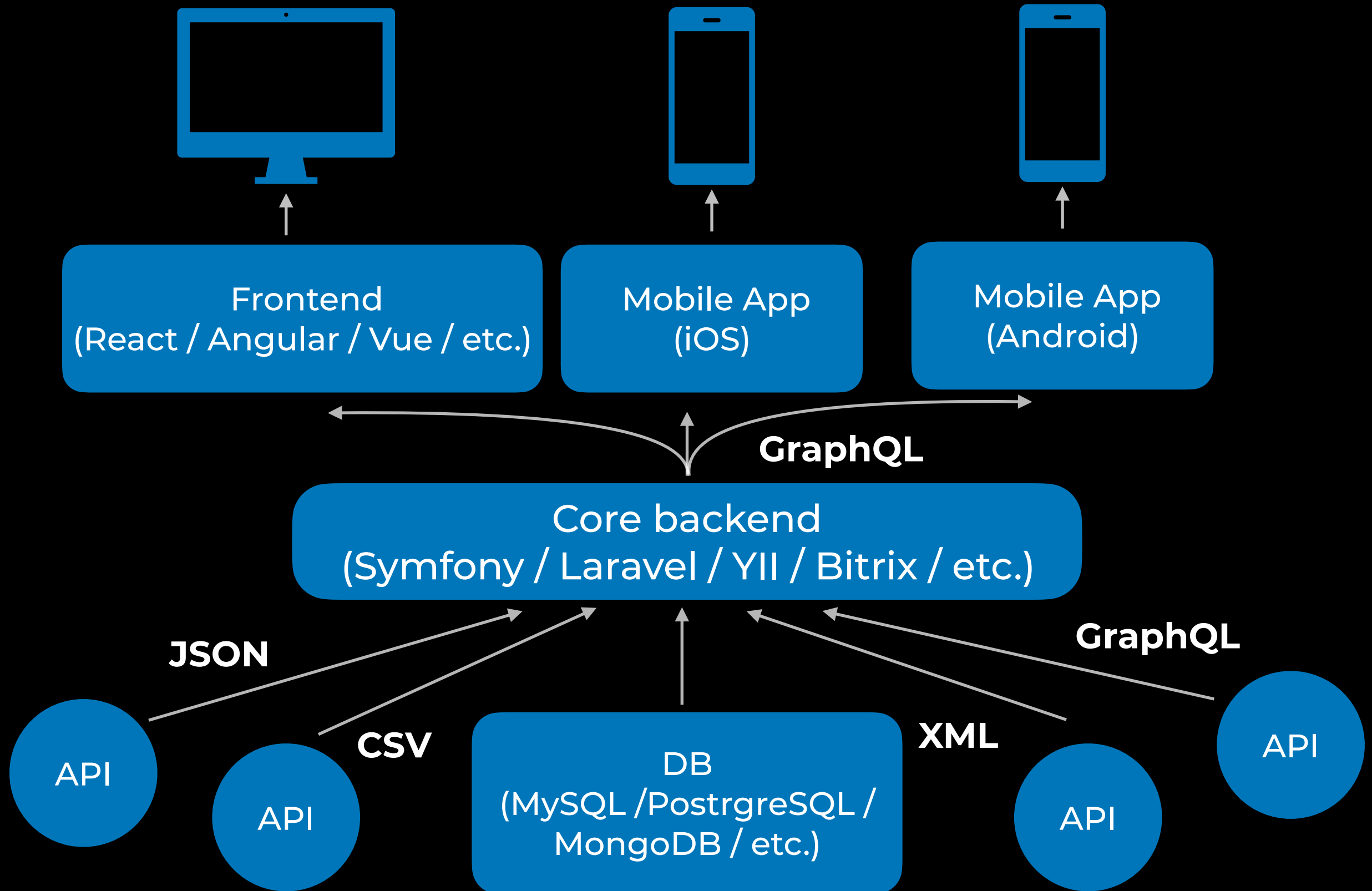
CSR



Проблемы

- На backend-разработчика возлагается ответственность за представление данных (лишняя логика, малая гибкость)
- ~~У frontend-разработчика отсутствует согласованность между HTML, JS и CSS~~
- Backend'у приходится выступать как прокси к другим сервисам

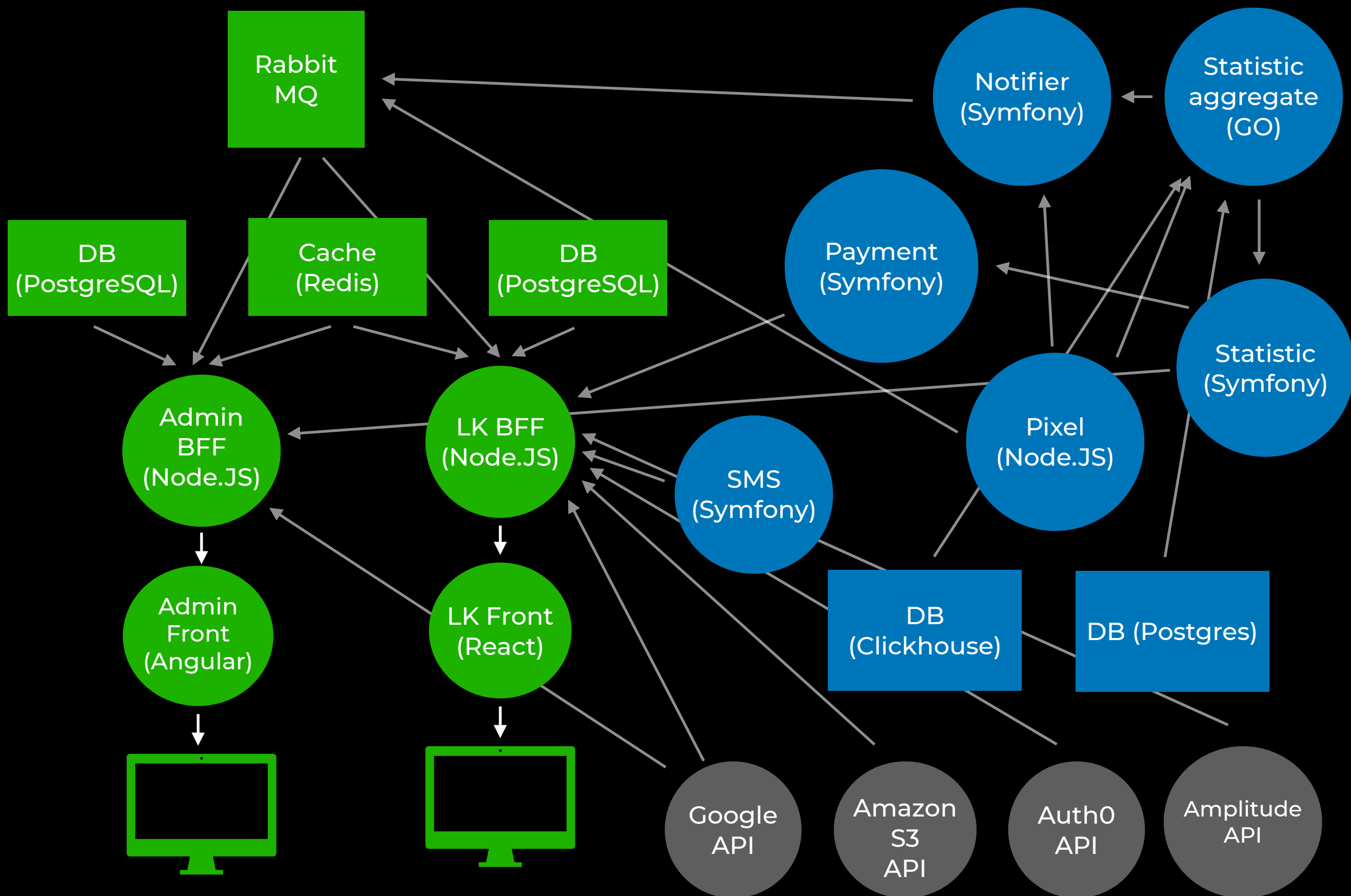
GraphQL



Проблемы

- ~~На backend-разработчика возлагается ответственность за представление данных (лишняя логика, малая гибкость)~~
- ~~У frontend-разработчика отсутствует согласованность между HTML, JS и CSS~~
- Backend'у приходится выступать как прокси к другим сервисам

Микросервисы и BFF



Преимущества

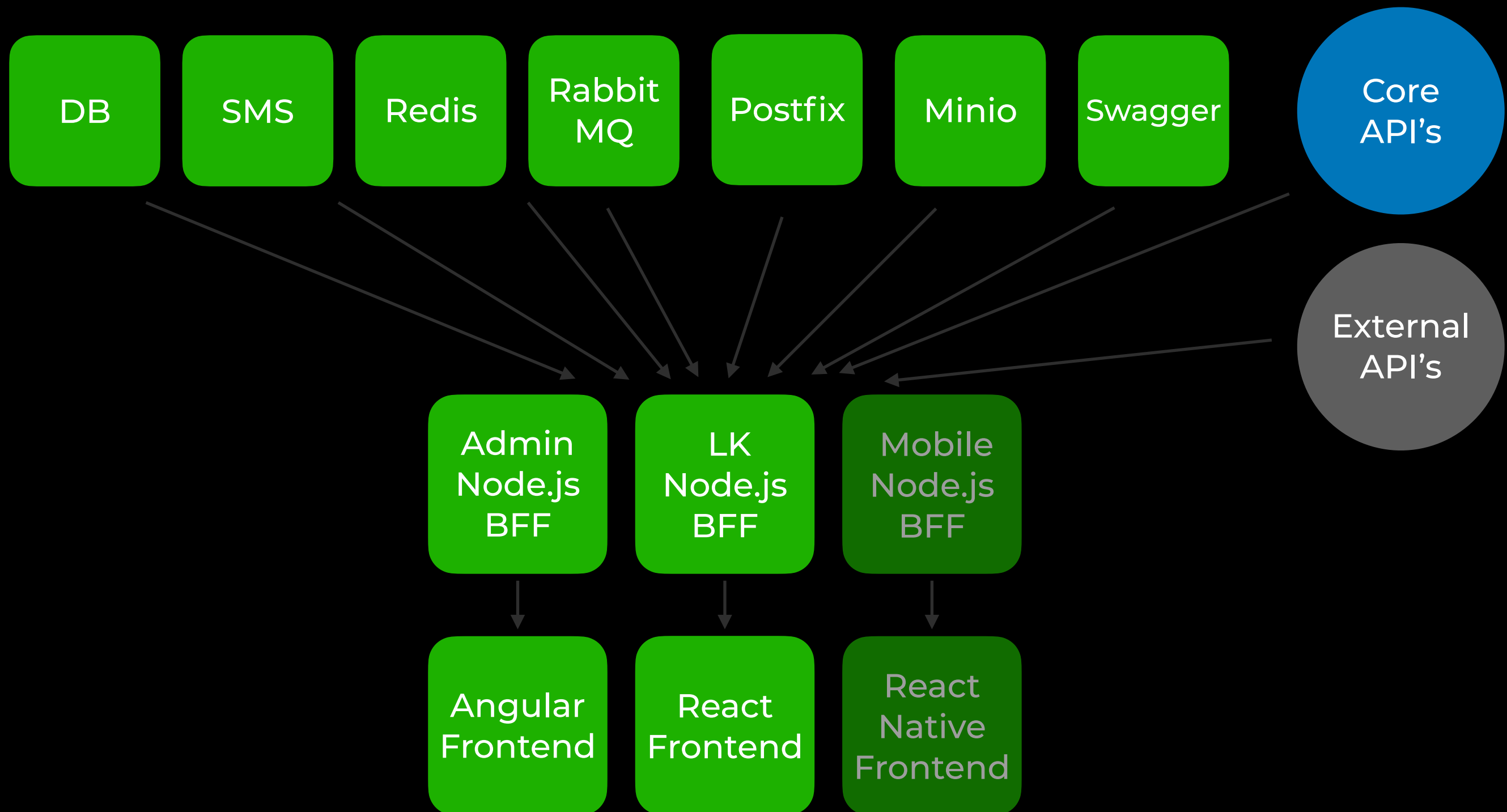
- Отсутствие зависимости от языка и фреймворка в каждом из сервисов
- Для взаимодействия с сервисами backend-разработчику достаточно реализовать REST-API
- Backend-разработчик не взаимодействует с сервисами, которые не требуются конкретно ему

Функциональность, реализуемая BFF

- Аутентификация, авторизация, хранение сессий
- Валидация запросов
- Хранение справочников
- Взаимодействие с другими сервисами и формирование данных в необходимом фронтенду формате
- Кеширование
- Логирование
- Отправка email/sms
- Хранение файлов (аватарки, выгрузки и т. д.)
- Отправка уведомлений
- Генерация OpenAPI

Если какая-то функциональность
требуется стороннему сервису, то
её необходимо унести из BFF
И ей не должен заниматься
Fontend-разработчик

Архитектура Frontend-приложения



Почему



?

Требования к backend-фреймворку для BFF

- Наличие готовой архитектуры
- Наличие реализованной указанной раньше функциональности
- Популярность (регулярные обновления и поддержка)
- Простота и следование современным подходам
- Написан на JavaScript/TypeScript

master

Go to file

Add file

Code

About

A progressive Node.js framework for building efficient, scalable, and enterprise-grade server-side applications on top of TypeScript & JavaScript (ES6, ES7, ES8) 🚀

[nestjs.com](#)

nest

javascript

typescript

nodejs

framework

nodejs-framework

typescript-framework











javascript-framework

microservices

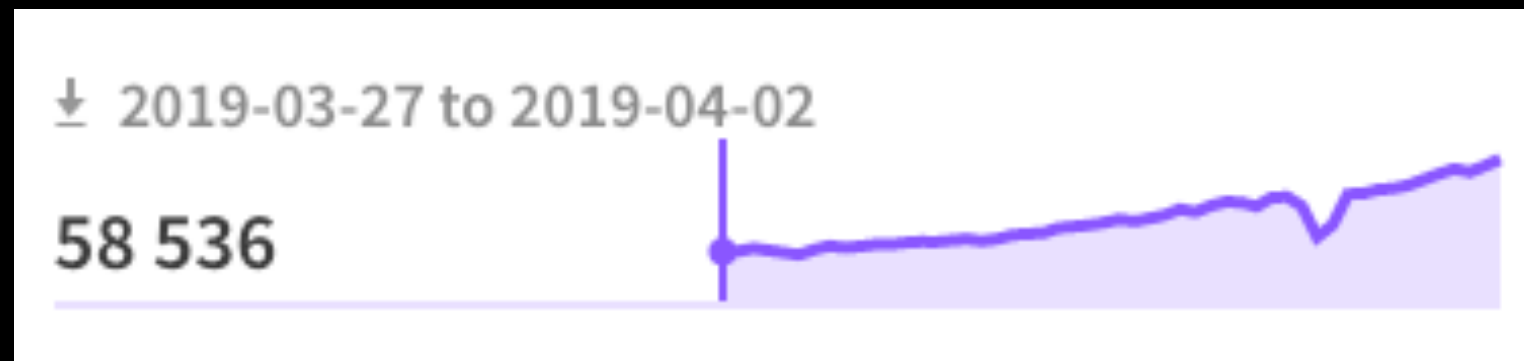
websockets

node

nestjs

 kamilmysliwiec fix(common): respect NO_COLOR e... 3 hours ago 6,422 .circleci Update config.yml 5 months ago .github Create Dependabot config file 4 days ago benchmarks refactor(): Rewrite benchmark script to TS 15 months ago integration test: gateway-ack when no data provided o... 7 days ago packages fix(common): respect NO_COLOR env varia... 3 hours ago sample chore(deps): update babel monorepo to v7.... 14 hours ago scripts fix(): integration tests cleanup, update deps... 9 months ago tools ci(): temporarily switch to npm i instead of c... 22 days ago .commitlintrc.json chore(): bump package version 13 months ago .eslintignore chore(): remove lint from lint-staged 10 months ago

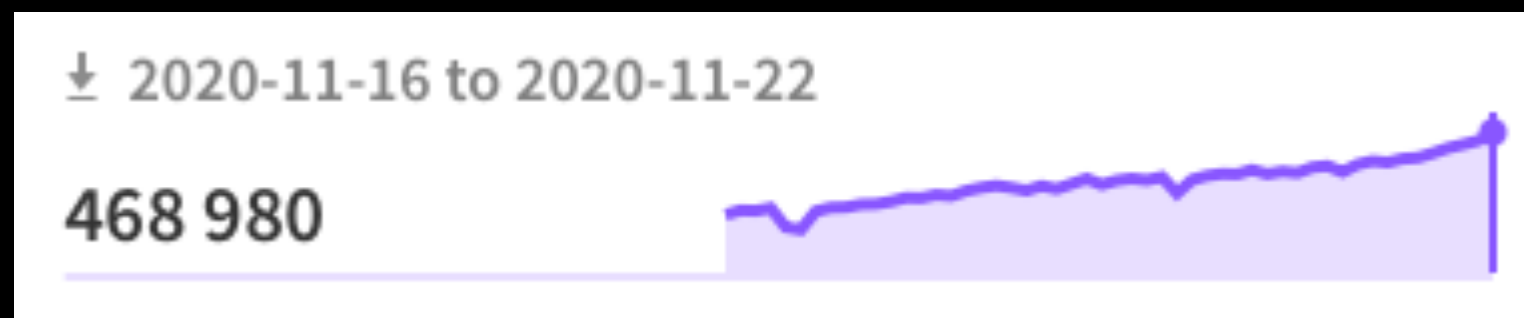
Март 2019



Март 2020



Ноябрь 2020



Архитектура



Архитектура

- Объектно-ориентированное программирование (с ФП и ФРП)
- Декораторы
- Express
- TypeScript
- Dependency Injection
- TypeORM
- Контроллеры
- Guards
- Pipes
- Кеширование
- Логирование, обработка ошибок, очереди, middleware и т. д.
- CLI



<https://docs.nestjs.com/>



<https://angular.io/docs>

Установка

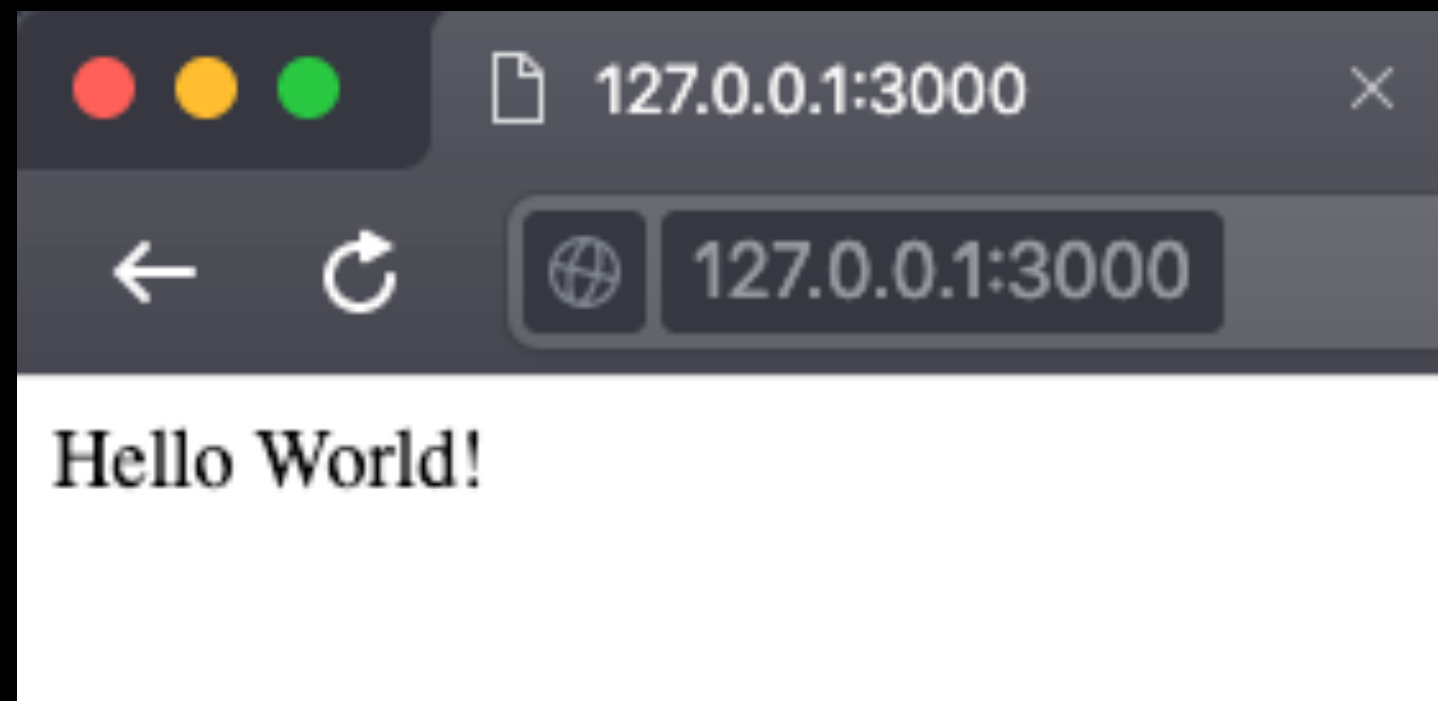
```
$ npm i -g @nestjs/cli
```

```
$ nest new hello-world
```

Запуск

```
$ cd hello-world
```

```
$ npm run start
```



```
/* AppController.ts */
```

```
import { Controller, Get } from '@nestjs/common';  
import { AppService } from '../app.service';
```

```
@Controller('/')  
export class AppController {  
  constructor(  
    private readonly appService: AppService,  
  ) {}
```

```
  @Get()  
  public getHello(): string {  
    return this.appService.getHello();  
  }  
}
```

```
/* AppService.ts */
```

```
import { Injectable } from '@nestjs/common';
```

```
@Injectable()
```

```
export class AppService {  
  public getHello(): string {  
    return 'Hello World!';  
  }  
}
```



```
/* AppModule.ts */
```

```
import { Module } from '@nestjs/common';  
import { AppController } from './app.controller';  
import { AppService } from './app.service';
```

```
@Module({  
  providers: [AppService],  
  controllers: [AppController],  
  imports: [],  
  exports: [],  
})
```

```
export class AppModule {}
```

```
/* main.ts */
```

```
import { NestFactory } from '@nestjs/core';  
import { AppModule } from './app.module';
```

```
async function bootstrap() {  
  const app = await NestFactory.create(AppModule);  
  
  await app.listen(3_000);  
}
```

```
bootstrap();
```

UserModule

providers: [UserFinder, UserCreator, UserUpdater]

controllers: [UserController]

UserFinder

UserCreator

UserUpdater

UserController

exports: [UserFinder]

AppModule

imports: [UserModule]

AppService

AppController

providers: [AppService]

controllers: [AppController]

Дочерние модули

- user
 - controllers
 - UserController.ts
 - dto
 - UserCreateDto.ts
 - entities
 - User.ts
 - factories
 - UserFactory.ts
 - services
 - UserCreator.ts
 - UserModule.ts
 - AppModule.ts
 - main.ts

```
@Controller(prefix: '/user')  
export class UserController {  
  constructor(  
    private readonly userCreator: UserCreator,  
  ) {}  
  
  @Post()  
  public create(@Body() dto: UserCreatedDto): Promise<User> {  
    return this.userCreator.create(dto);  
  }  
}
```

```
export class UserCreatedDto {  
    public name!: string;  
}
```

```
@Entity( options: {  
  name: 'users',  
})  
  
export class User {  
  @PrimaryGeneratedColumn( strategy: 'increment')  
  public id!: number;  
  
  @Column( options: {  
    type: 'varchar',  
    length: 255,  
    unique: true,  
    nullable: false,  
  })  
  public name!: string;  
  
  @Column( options: {  
    type: 'int',  
    default: 0,  
    nullable: false,  
  })  
  public balance!: number;  
}
```



```
@Injectable()
export class UserFactory {
  public createFromCreatedDto(dto: UserCreatedDto): User {
    const user = new User();

    user.name = dto.name;

    return user;
  }
}
```

The image shows a TypeScript code snippet with a function signature. Two red boxes highlight the type annotations: box 1 highlights 'UserCreatedDto' and box 2 highlights 'User'.

```
@Injectable()
export class UserCreator {
  constructor(
    @InjectRepository(User)1
    private readonly userRepository: Repository<User>,
    private readonly userFactory: UserFactory,
  ) {}

  public create(dto: UserCreatedDto): Promise<User> {
    const user = this.userFactory.createFromCreatedDto(dto);

    return this.userRepository.save(user);
  }
}
```

```
@Module(metadata: {  
  imports: [  
    TypeOrmModule.forFeature(entities: [User]),  
  ],  
  providers: [  
    UserCreator,  
    UserFactory,  
  ],  
  controllers: [  
    UserController,  
  ],  
})  
export class UserModule {}
```

```
@Module(metadata: {  
  imports: [  
    TypeOrmModule.forRoot(  
      type: 'postgres',  
      host: 'localhost',  
      port: 5432,  
      username: 'example',  
      password: 'example',  
      database: 'example',  
      synchronize: true,  
      namingStrategy: new SnakeNamingStrategy(),  
      entities: ["src/**/*.entities/*.ts"],  
    ),  
    UserModule,  
  ],  
})  
export class AppModule {}
```

Валидация (pipes)

POST localhost:3000/user

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

1 {}

Body Cookies Headers (7) Test Results Status: 500 Internal Server Error

Pretty Raw Preview Visualize JSON ▼ ↺

```
1 {  
2   "statusCode": 500,  
3   "message": "Internal server error"  
4 }
```

[Nest] 8509 - 25.11.2020, 16:33:44 [ExceptionHandler]
null value in column "name" of relation "users" violates
not-null constraint +26361ms

`@Patch(path: ':id')`

`public update(`

`@Param(property: 'id') id: number,`

`@Body() dto: UserUpdatedDto,`

`): Promise<User> {`

`return this.userUpdater.update(id, dto);`

`}`

```
@Patch(path: ':id')
public update(
  @Param(property: 'id') id: string,
  @Body() dto: UserUpdateDto,
): Promise<User> {
  const parsedId = parseInt(id, radix: 10);

  if (isNaN(parsedId)) {
    throw new Error('Id must be a number');
  }

  return this.userUpdater.update(parsedId, dto);
}
```



```
export class UserUpdatedto {  
  @IsString()  
  @Length(min: 1, max: 255)  
  @IsOptional()  
  public name?: string;  
  
  @IsInt()  
  @IsPositive()  
  @Max(maxValue: 100_000)  
  @IsOptional()  
  public balance?: number;  
}
```

```
export class UserCreatedto {  
  @IsString()  
  @Length(min: 1, max: 255)  
  @IsDefined()  
  public name!: string;  
}
```

```
@Patch(path: ':id')
public update(
    @Param(property: 'id', ParseIntPipe) id: number,
    @Body() dto: UserUpdatedDto,
): Promise<User> {
    return this.userUpdater.update(id, dto);
}
```

POST

localhost:3000/user

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON ▼

1 {}

Body

Cookies

Headers (7)

Test Results

Status: 400 Bad Request

Pretty

Raw

Preview

Visualize

JSON ▼



1 {
2
3
4
5
6
7
8
9 }

```
"statusCode": 400,  
"message": [  
  "name should not be null or undefined",  
  "name must be longer than or equal to 1 characters",  
  "name must be a string"  
],  
"error": "Bad Request"
```

Авторизация (guards)

```
@Injectable()
export class AuthGuard implements CanActivate {
  public canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest<Request>();

    return !!request.query.admin;
  }
}
```

```
@Controller(prefix: '/user')  
@UseGuards(new AuthGuard())  
export class UserController {...}
```

POST

localhost:3000/user

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON ▼

1

{}

Body

Cookies

Headers (7)

Test Results

Status: 403 Forbidden

Pretty

Raw

Preview

Visualize

JSON ▼



1

{

2

"statusCode": 403,

3

"message": "Forbidden resource",

4

"error": "Forbidden"

5

}

Преобразование ответа (interceptors)


```
export interface IResponse<Data = {}> {  
  code: number;  
  message: string;  
  payload: Data;  
}
```

@Injectable()

export class FormatResponse implements **NestInterceptor** {

public intercept(context: ExecutionContext, **next: CallHandler**): **Observable<unknown>** {

return next.handle().pipe(

map(

project: (payload): IResponse => ({
 code: HttpStatus.OK,
 message: '',
 payload: payload || {},
}),

),

);

}

}

```
@Controller(prefix: '/user')  
@UseGuards(new AuthGuard())  
@UseInterceptors(new FormatResponse())  
export class UserController {...}
```

1

Обработка ошибок (exception filters)

```
@Get(path: ':id')
public async findById(@Param(property: 'id', ParseIntPipe) id: number): Promise<User> {
    const user = await this.userFinder.findById(id);

    if (!user) {
        throw new NotFoundException(objectOrError: 'User not found');
    }

    return user;
}
```

@Catch(HttpException) ¹

export class HttpExceptionFilter implements **ExceptionHandler** ² {

public catch(exception: HttpException, host: ArgumentsHost) {

const ctx = host.switchToHttp();

const **response** ³ = ctx.getResponse<Response>();

const status = exception.getStatus();

const responseDto: **IResponse** ⁴ = {

code: status,

message: exception.message,

stack: exception.stack,

payload: {},

}

response

.**status**(status)

.**json**(responseDto);

}

}

```
@Controller( prefix: '/user')  
@UseGuards(new AuthGuard())  
@UseInterceptors(new FormatResponse())  
@UseFilters(new HttpExceptionHandler())  
export class UserController {...}
```

Логирование


```
export class LoggerMiddleware implements NestMiddleware {  
  public use(req: Request, res: Response, next: NextFunction): void {  
    console.log('New request has come', {  
      extra: {  
        request: {  
          ip: req.ip,  
          baseUrl: req.baseUrl,  
          headers: req.headers,  
          method: req.method,  
          query: req.query,  
          body: req.body,  
        },  
      },  
    });  
    next();  
  }  
}
```

```
@Module({...})  
export class UserModule implements NestModule {  
  public configure(consumer: MiddlewareConsumer): void {  
    consumer.apply(LoggerMiddleware).forRoutes(UserController);  
  }  
}
```

Кеширование

```
import { RedisService1 } from 'nestjs-redis';
import * as Redis from 'ioredis';
import { Injectable } from '@nestjs/common';

@Injectable()
export class CacheService {
  private readonly client: Redis.Redis;

  constructor(
    private readonly redisService: RedisService,
  ) {
2    this.client = this.redisService.getClient();
  }

3  public set<Value>(key: string, value: Value) {
    return this.client.set(key, JSON.stringify(value));
  }

  public async get<Value>(key: string): Promise<Value | undefined> {
4    const data = await this.client.get(key);

    if (!data) {
      return;
    }

5    return JSON.parse(data);
  }
}
```

```
@Injectable()
export class UserRepo {
  private readonly cachePrefix = 'user-';

  constructor(
    @InjectRepository(User)
    private readonly baseUserRepository: Repository<User>,
    private readonly cacheService: CacheService,
  ) {}

  private getCacheKey(id: number): string {
    return `${this.cachePrefix}${id}`;
  }

  public async findById(id: number): Promise<User | undefined> {...}

  public async create(user: User): Promise<User> {...}

  public async update(user: User): Promise<User> {...}

  public async findAll(): Promise<User[]> {...}
}
```

```
public async findById(id: number): Promise<User | undefined> {  
  const cacheKey = this.getCacheKey(id);  
  
  const cachedUser = await this.cacheService.get<User>(cacheKey);  
  
  if (cachedUser) {  
    return cachedUser;  
  }  
  
  const user = await this.baseUserRepository.findOne(id);  
  
  await this.cacheService.set(cacheKey, user);  
  
  return user;  
}
```

```
@Module(metadata: {  
  imports: [  
    TypeOrmModule.forRoot({type: 'postgres'...}),  
    RedisModule.register(  
      options: {  
        host: 'localhost',  
        port: 6379,  
      }  
    ),  
    UserModule,  
  ],  
})  
export class AppModule {}
```

OpenAPI (swagger)

PATCH /user/{id}

Parameters

Try it out

Name	Description
------	-------------

id * required

number
(path)

id

Request body required

application/json

Example Value | Schema

```
{
  "name": "string",
  "balance": 0
}
```

Responses

Code	Description	Links
------	-------------	-------

201

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
  "name": "string",
  "balance": 0
}
```

```
export class UserUpdatedDto {  
  @ApiModelProperty()  
  @IsString()  
  @Length(min: 1, max: 255)  
  @IsOptional()  
  public name?: string;  
  
  @ApiModelProperty()  
  @IsInt()  
  @IsPositive()  
  @Max(maxValue: 100_000)  
  @IsOptional()  
  public balance?: number;  
}
```

```
@Entity({name: 'users'...})
export class User {
    @ApiProperty()
    @PrimaryGeneratedColumn(strategy: 'increment')
    public id!: number;

    1
    @ApiProperty()
    @Column({type: 'varchar'...})
    public name!: string;

    2
    @ApiProperty()
    @Column({type: 'int'...})
    public balance!: number;
}
```

```
@Patch(path: ':id')
@ApiCreatedResponse(options: { type: User })
public update(
    @Param(property: 'id', ParseIntPipe) id: number,
    @Body() dto: UserUpdatedto,
): Promise<User> {
    return this.userUpdater.update(id, dto);
}
```

```
@Get()
@ApiOkResponse(options: { type: User, isArray: true })
public findAll(): Promise<User[]> {
    return this.userFinder.findAll();
}
```

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(new ValidationPipe());

  const options = new DocumentBuilder()
    .setTitle('Api example')
    .setDescription('API description')
    .setVersion('1.0')
    .addTag({ name: 'example' })
    .build();

  const document = SwaggerModule.createDocument(app, options);

  SwaggerModule.setup(path: 'api', app, document);

  await app.listen(port: 3_000);
}
```

Кастомные декораторы

Code	Description
201	<div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "id": 0, "name": "string", "balance": 0 }</pre></div>

Code	Description
201	<div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "code": 0, "message": "string", "stack": "string", "payload": { "id": 0, "name": "string", "balance": 0 } }</pre></div>


```
export const ApiBaseResponse = <TModel extends Type<any>>(  
  model: TModel,  
  options?: ApiResponseMetadata,  
) => {  
  const schema = { $ref: getSchemaPath(model) };  
  
  const payload = options?.isArray  
    ? { type: 'array', items: schema }  
    : schema;
```

```
  return applyDecorators(  
    ApiResponse({ options: {  
      ...options,  
      schema: {  
        allOf: [  
          { $ref: getSchemaPath(BaseResponse) },  
          {  
            properties: {  
              payload,  
            },  
          },  
        ],  
      },  
    }),  
  );  
}
```

```
@Get()
```

```
@ApiResponse(User, options: { status: HttpStatus.OK, isArray: true })
```

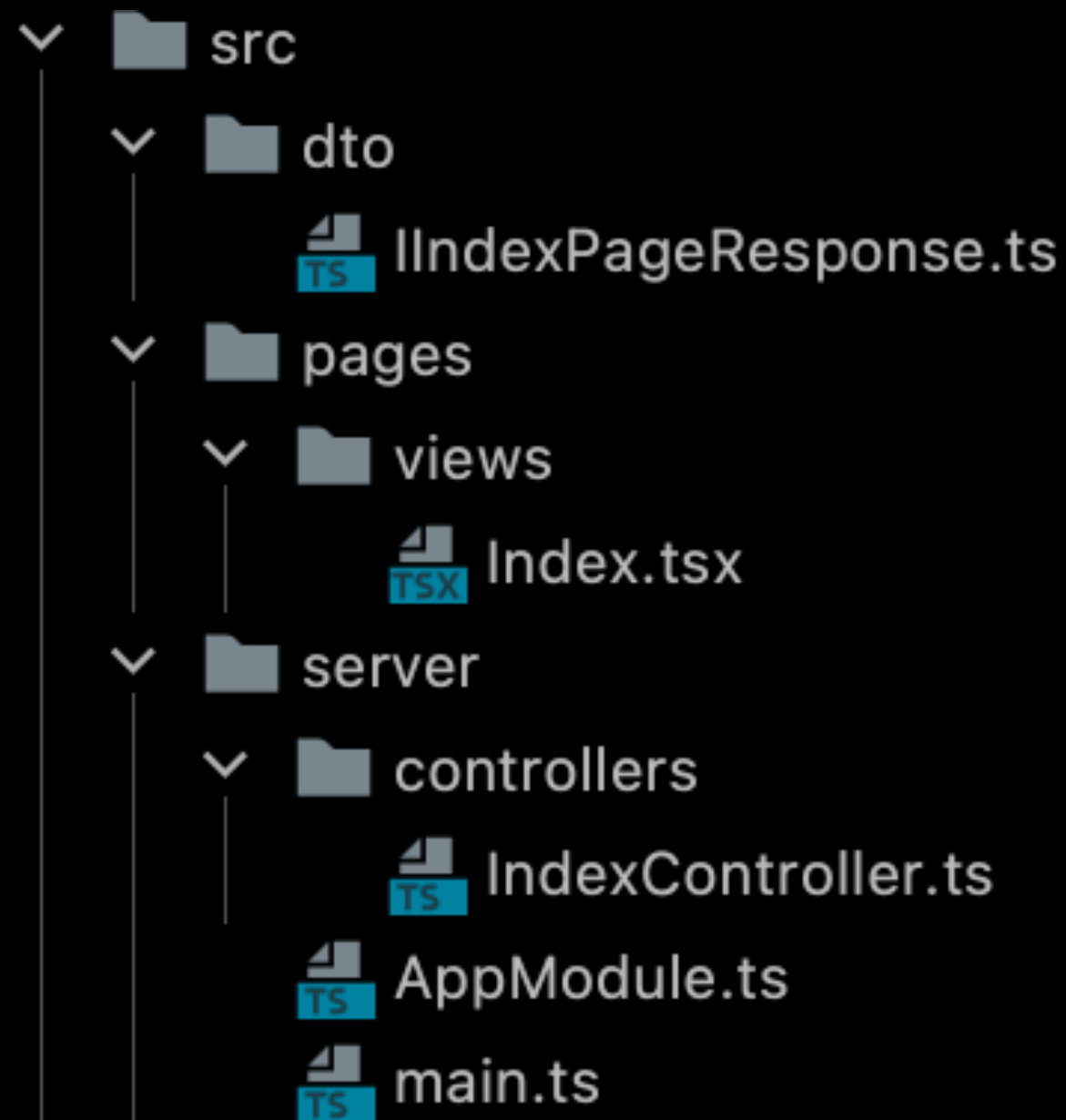
```
public findAll(): Promise<User[]> {
```

```
    return this.userFinder.findAll();
```

```
}
```

- >  docker-data
- >  node_modules library root
- ▼  src
 - ▼  cache
 - >  services
 -  CacheModule.ts
 - ▼  user
 - >  controllers
 - >  decorators
 - >  dto
 - >  entities
 - >  exceptionFilters
 - >  factories
 - >  guards
 - >  interceptors
 - >  middlewares
 - >  repositories
 - >  services
 -  UserModule.ts
 -  AppModule.ts
 -  main.ts
-  .gitignore
-  docker-compose.yml
-  nodemon.json
-  package.json
-  package-lock.json
-  tsconfig.json

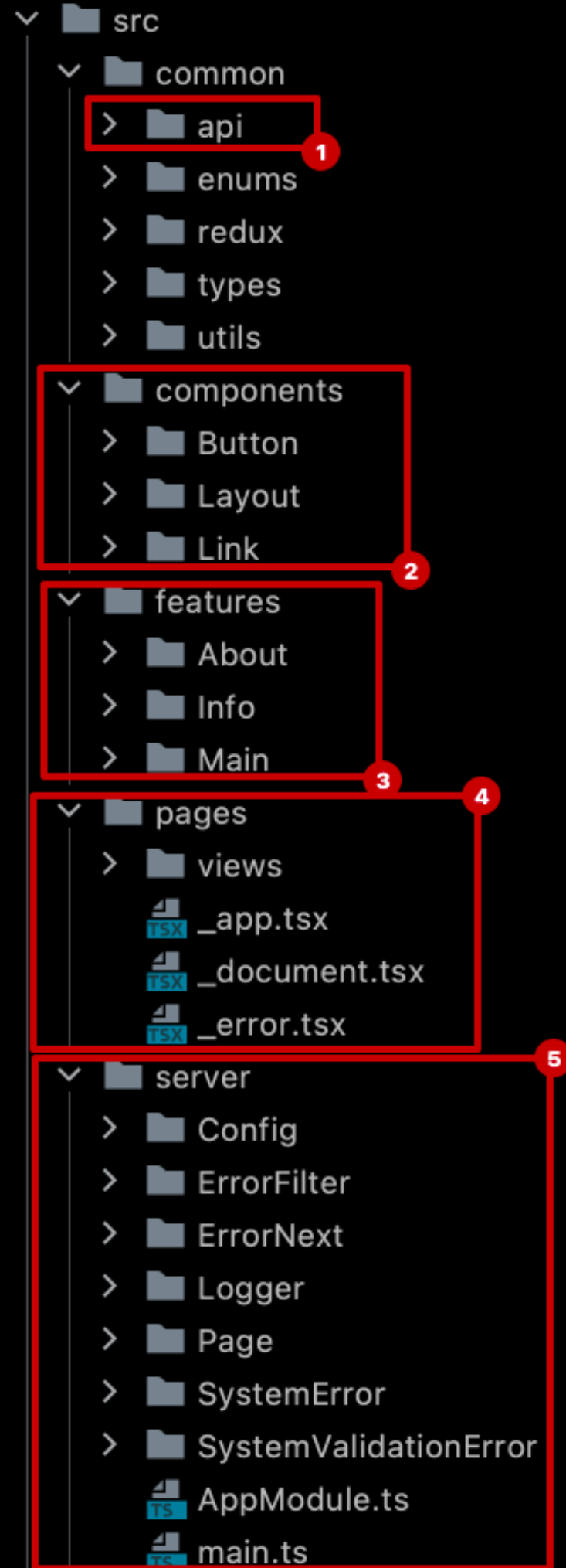
Взаимодействие с фронтендом



```
export interface IIndexPageResponse {  
    title: string;  
    content: string;  
}
```

```
@Controller()
export class IndexController {
  @Get(path: '/')
  @Render(template: 'Index')
  public get(): IIndexPageResponse {
    return {
      title: 'Index page',
      content: 'Hello world',
    }
  }
}
```

```
const Index: NextPage<IIndexPageResponse> = ({  
  title: string,  
  content: string,  
}) => (  
  <>  
    <Head>  
      <title>{title}</title>  
    </Head>  
    {content}  
  </>  
);
```

<https://github.com/DiFuks/hello-world>

<https://github.com/DiFuks/nest-next-minimal/tree/simple-sample>

<https://github.com/DiFuks/react-next-nest-boilerplate>