2k20

ESSENTIALS WITH TYPESCRIPT

EXPRESS API

О Чем Поговорим?

> CRUD над моделью

> Валидация, фильтрация, сортировка, логирование, CORS

> Связанные модели

> Универсальный обработчик запросов

> Docker

О Себе

Full-stack JS разработчик

> Angular 8 (NgRx)

> Express + TS

> Nest

EXPRESS

Структура запросов

```
>> GET     /users
>> GET     /users/:id
>> POST    /users
>> PUT     /users/:id
>> DELETE  /users/:id
```

```
>> GET      /users?limit=10
>> GET      /users?offset=10
>> GET      /users?filter={"name": "John"}
>> GET      /users?sort=name
>> GET      /users?limit=10&offset=10&filter={"name": "John"}&order=1
```

Минимальная конфигурация

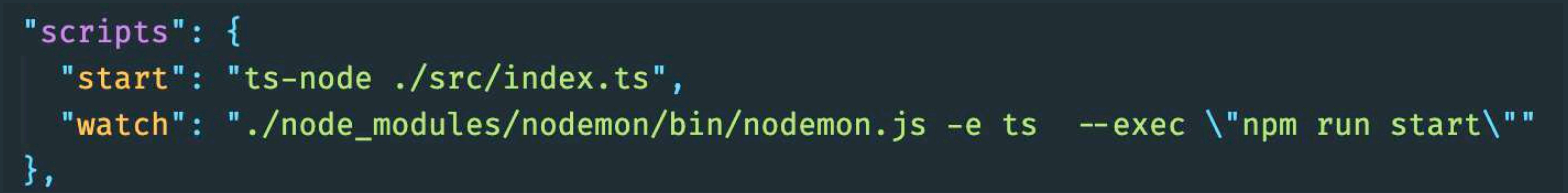
```
"dependencies": {
  "express": "^4.17.1"
},
"devDependencies": {
  "nodemon": "^2.0.2",
  "typescript": "^3.7.4",
  "@types/express": "^4.17.2"
}
```

Минимальная конфигурация

```
"scripts": {
  "start": "node ./build/index.js",
  "compile": "tsc && node ./build/index.js",
  "watch": "./node_modules/nodemon/bin/nodemon.js -e ts  --exec \"npm run compile\""
},
```

Минимальная конфигурация

TS-NODE



```
"scripts": {
  "start": "ts-node ./src/index.ts",
  "watch": "./node_modules/nodemon/bin/nodemon.js -e ts  --exec \"npm run start\""
},
```

TYPESCRIPT

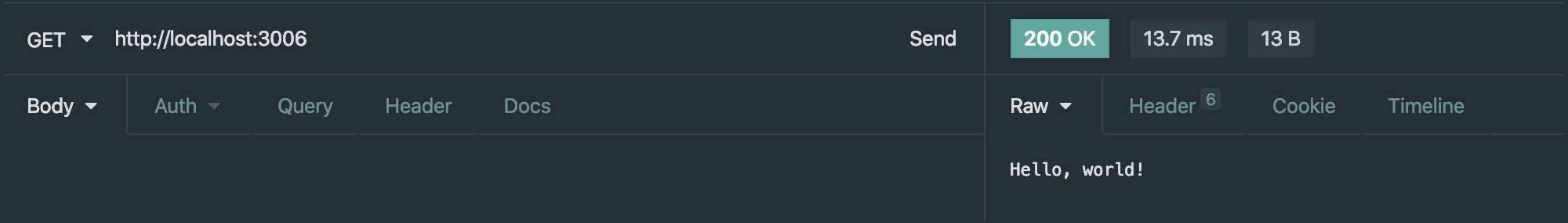
Настройка компиляции TS

```json
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es2015",
    "outDir": "./build"
  },
  "include": [
    "./src"
  ],
  "exclude": [
    "node_modules"
  ]
}
```

Hello, world!

```
import * as express from 'express';
import { Request, Response, Application } from 'express';


const app: Application = express();

app.get('/', (req: Request, res: Response) => {
    res.send('Hello, world!');
});

app.listen(3006, () => console.log('Сервер стартовал на порту 3006!'));
```

GET /



http://localhost:3006

200 OK 13.7 ms 13 B

```
Hello, world!
```

MONGODB

MONGOOSE ODM

Mongoose

```
"dependencies": {
  "express": "^4.17.1",
  "mongoose": "^5.8.7"
},
"devDependencies": {
  "@types/express": "^4.17.2",
  "@types/mongoose": "^5.5.41",
  "nodemon": "^2.0.2",
  "typescript": "^3.7.4"
}
```

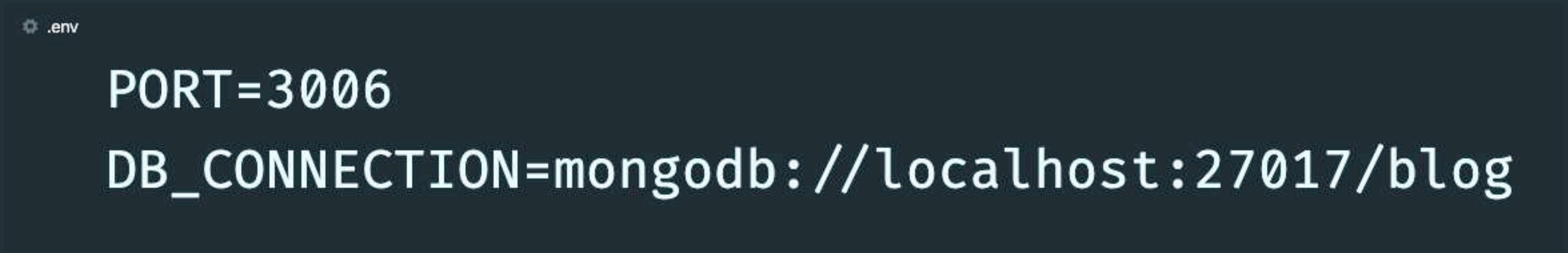
MongoDB

```yaml
version: '3'

services:
  mongo:
    image: mongo
    ports:
        - "27017:27017"
```

Подключение к MongoDB

```
import * as mongoose from 'mongoose';



mongoose
    .connect('mongodb://localhost:27017/blog',  { useNewUrlParser: true })
    .then(() => console.log('Успешное подключение к БД'))
    .catch(err => console.log('Произошла ошибка\n', err));
```

Конфигурация

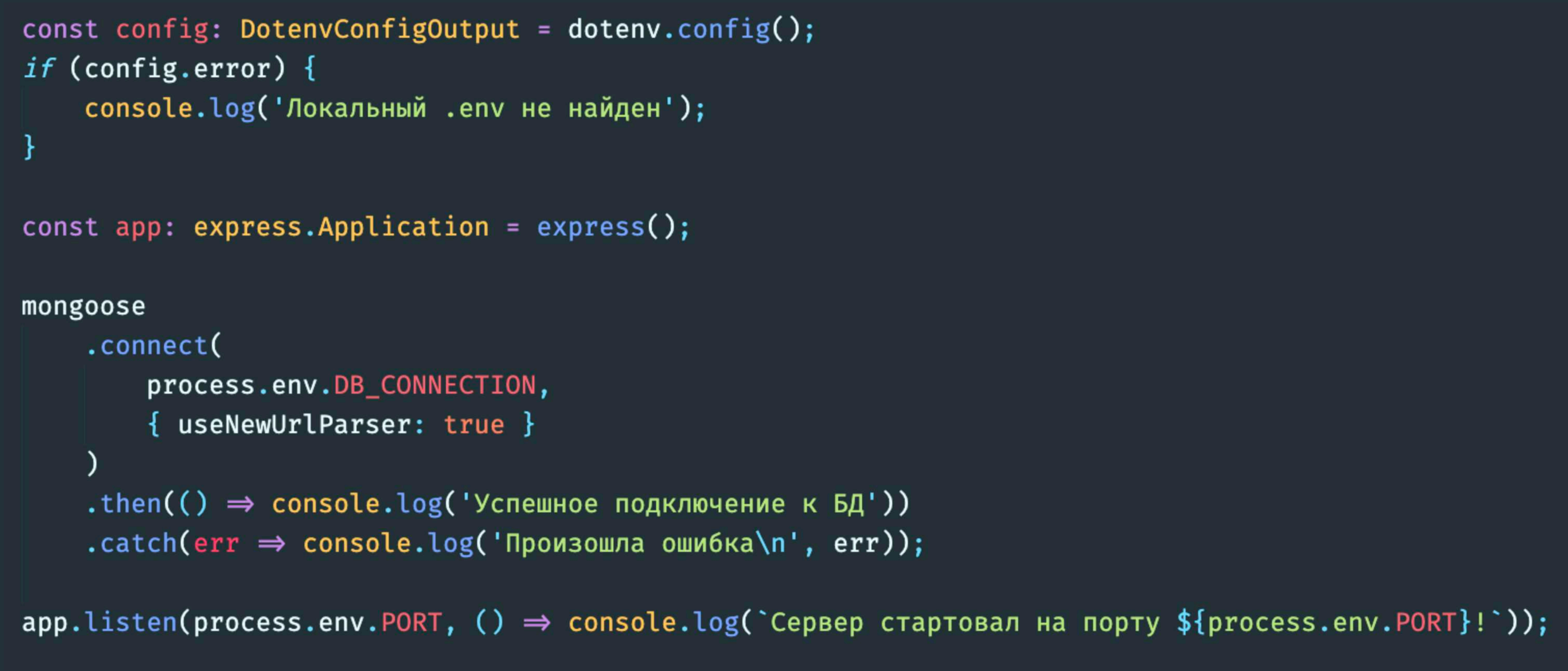

.env

```
PORT=3006
DB_CONNECTION=mongodb://localhost:27017/blog
```

dotenv

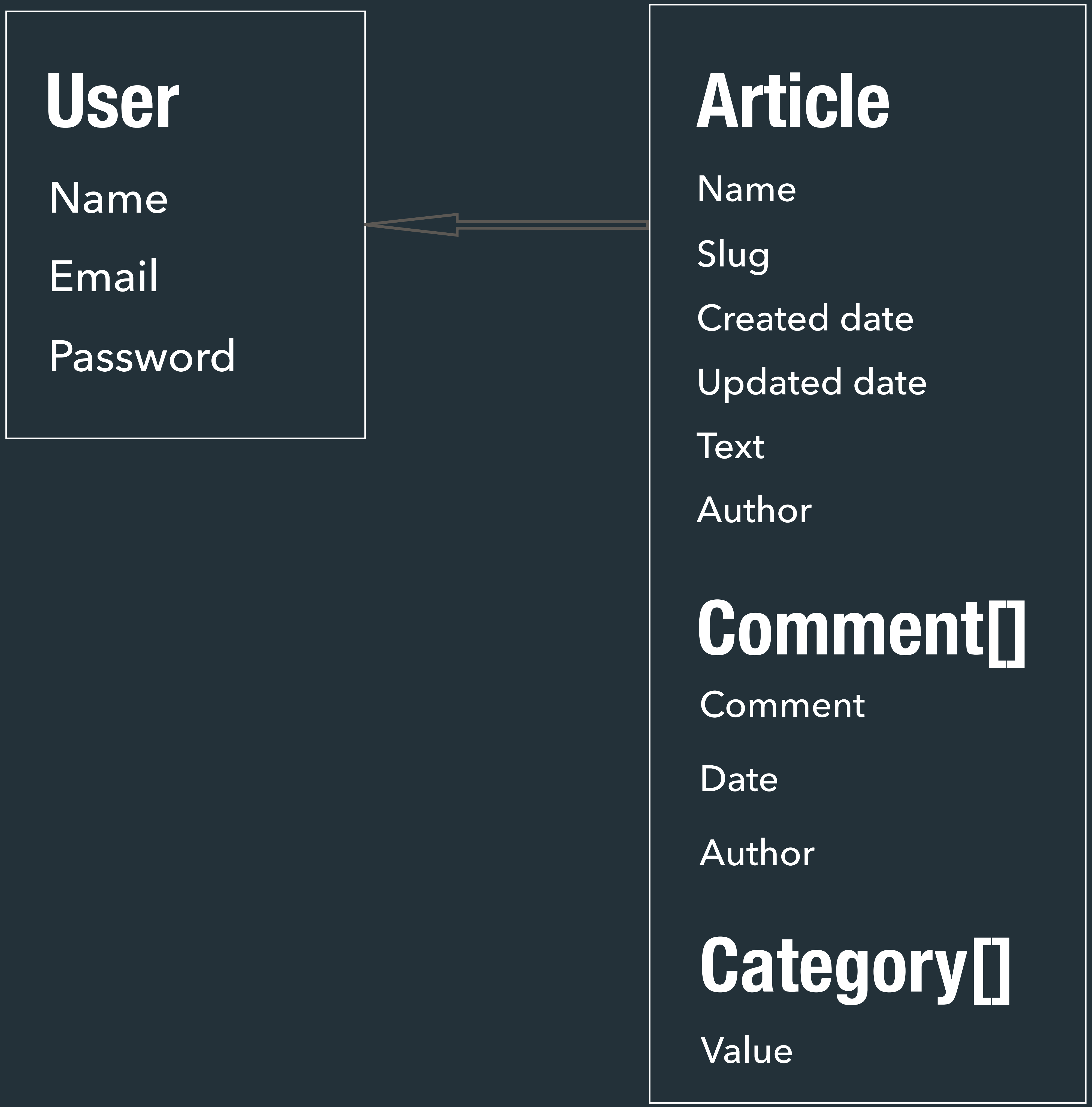
```
"dependencies": {
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "mongoose": "^5.8.7"
},
"devDependencies": {
  "@types/dotenv": "^8.2.0",
  "@types/express": "^4.17.2",
  "@types/mongoose": "^5.5.41",
  "nodemon": "^2.0.2",
  "typescript": "^3.7.4"
}
```

Конфигурация



```
const config: DotenvConfigOutput = dotenv.config();
if (config.error) {
    console.log('Локальный .env не найден');
}

const app: express.Application = express();

mongoose
    .connect(
        process.env.DB_CONNECTION,
        { useNewUrlParser: true }
    )
    .then(() => console.log('Успешное подключение к БД'))
    .catch(err => console.log('Произошла ошибка\n', err));

app.listen(process.env.PORT, () => console.log(`Сервер стартовал на порту ${process.env.PORT}!`));
```

Структура документов MongoDB



Структура проекта

```
▶ build
▶ node_modules
▼ src
    ▶ interfaces
    ▶ models
    ▶ routers
      index.ts
  .env
  docker-compose.yml
  package.json
  tsconfig.ts
```

Интерфейс User

```
import { Document } from "mongoose";


export interface User extends Document {
    name: string;
    email: string;
    password: string;
}
```

Mongoose Schema для User

```
import { Schema, model } from "mongoose";
import { User } from "../interfaces/user";


const userSchema: Schema = new Schema({
    name: String,
    email: String,
    password: String,
});

export default model<User>('User', userSchema);
```

Mongoose Schema

- Типы
- Хуки
- Геттеры по путям
- Вложенные схемы
- Виртуальные свойства
- Валидация, кастомные валидаторы, асинхронные валидаторы

User router

```
export default class UserRouterHandler {

    private _router: Router;

    public get router(): Router {
        return this._router;
    }

}
```

User router

```
private async _find(req: Request, res: Response): Promise<void> {

}

private async _create(req: Request, res: Response): Promise<void> {

}

private async _findOne(req: Request, res: Response): Promise<void> {

}

private async _update(req: Request, res: Response): Promise<void> {

}

private async _delete(req: Request, res: Response): Promise<void> {

}
```

User router

```
constructor() {
    this._router = Router();

    this._router.get('/', this._find.bind(this));
    this._router.post('/', this._create.bind(this));
    this._router.get('/:id', this._findOne.bind(this));
    this._router.put('/:id', this._update.bind(this));
    this._router.delete('/:id', this._delete.bind(this));
}
```

Body parser

```
"dependencies": {
  "body-parser": "^1.19.0",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "mongoose": "^5.8.7"
},
"devDependencies": {
  "@types/body-parser": "^1.17.1",
  "@types/dotenv": "^8.2.0",
  "@types/express": "^4.17.2",
  "@types/mongoose": "^5.5.41",
  "nodemon": "^2.0.2",
  "typescript": "^3.7.4"
}
```

Подключаем Body parser и User router handler

```
const config: DotenvConfigOutput = dotenv.config();
if (config.error) {
    throw config.error;
}

const app: Application = express();

mongoose
    .connect(process.env.DB_CONNECTION,  { useNewUrlParser: true })
    .then(() ⇒ console.log('Успешное подключение к БД'))
    .catch(err ⇒ console.log('Произошла ошибка\n', err));

app.use(bodyParser.json());

app.use('/api/users', new UserRouterHandler().router);

app.listen(3006, () ⇒ console.log(`Сервер стартовал на порту ${process.env.PORT}!`));
```

Middleware

```
app.get('/', (req: Request, res: Response) ⇒ {
    res.send('Hello, world!');
});
```

```
this._router = Router();

this._router.get('/', this._find.bind(this));
```

```
app.use(bodyParser.json());

app.use('/api/users', new UserRouterHandler().router);
```

Middleware

ЗАДАЧИ ФУНКЦИЙ ПРОМЕЖУТОЧНОЙ ОБРАБОТКИ

> Выполнение любого кода
> Внесение изменений в объекты запросов и ответов
> Завершение цикла “запрос-ответ”
> Вызов следующей функции промежуточной обработки из стека

Middleware

ВИДЫ ФУНКЦИЙ ПРОМЕЖУТОЧНОЙ ОБРАБОТКИ

- Обработчик уровня приложения
- Обработчик уровня маршрутизатора
- Обработчик для обработки ошибок
- Встроенные промежуточные обработчики
- Обработчики сторонних поставщиков

Middleware

СИНТАКСИС

Уровень	Тип	Аргументы
	get()	(req, res, next)
	put()	‘/endponit’, (req, res)
app	use()	‘/endponit’, (req, res, next)
router	post()	‘/endponit’, (err, req, res, next)
	delete()	
	…	‘/endponit’, (…), (…), (…)

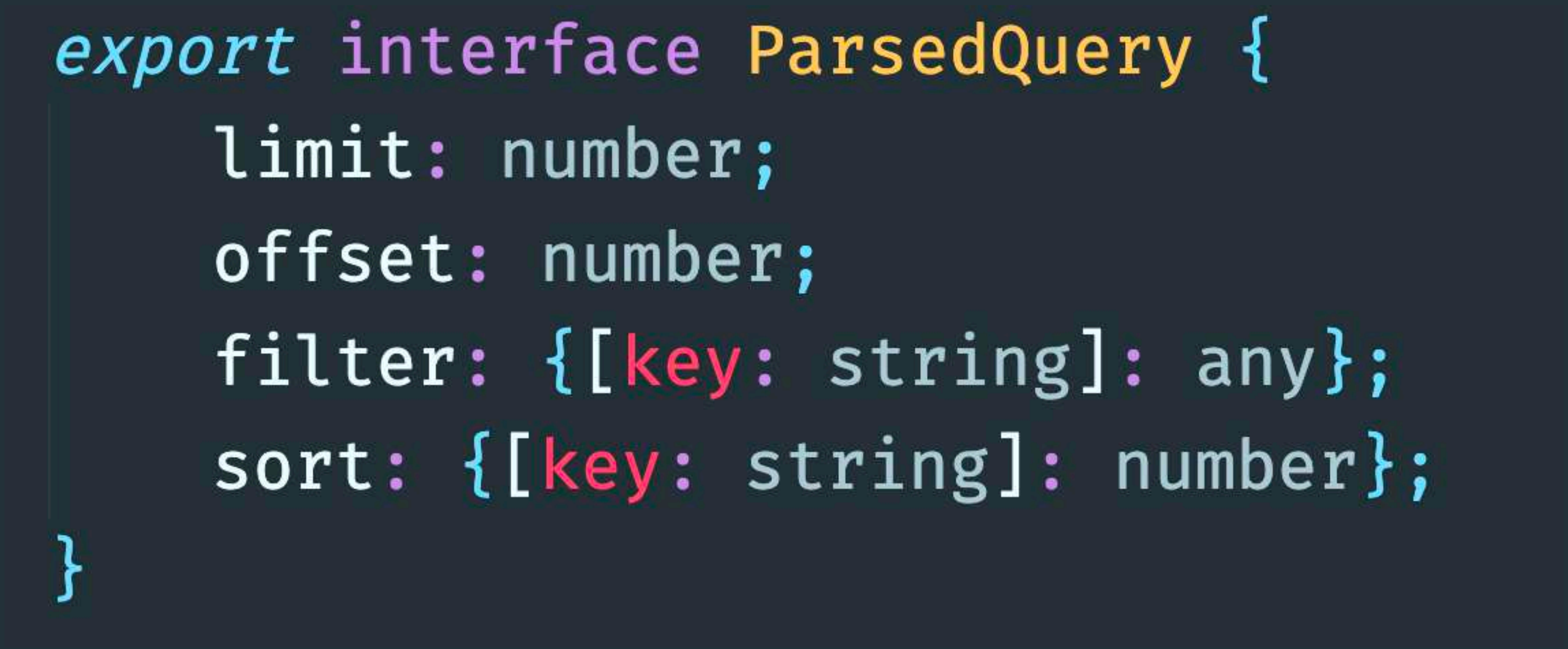
User router

```
private async _find(req: Request, res: Response): Promise<void> {

}

private async _create(req: Request, res: Response): Promise<void> {

}

private async _findOne(req: Request, res: Response): Promise<void> {

}

private async _update(req: Request, res: Response): Promise<void> {

}

private async _delete(req: Request, res: Response): Promise<void> {

}
```

Find user

```
private async _find(req: Request, res: Response): Promise<void> {
    try {
      const users: IUser[] = await User.find();

      res.json(users);
    } catch (error) {
      res.status(500).json({ error });
    }
}
```

Пагинация. Фильтрация. Сортировка.



```
export interface ParsedQuery {
    limit: number;
    offset: number;
    filter: {[key: string]: any};
    sort: {[key: string]: number};
}
```

Парсим данные query string

```
private _parseRequsetQuery(query: {[key: string]: any}): ParsedQuery {
    const limit: number = query.limit ? parseInt(query.limit) : 0;
    const offset: number = query.offset ? parseInt(query.offset) : 0;

    const filter: {[key: string]: any} = query.filter
        ? JSON.parse(query.filter)
        : {};

    const sort: {[key: string]: number} = (query.sort && query.order)
        ? {[query.sort]: parseInt(query.order)}
        : {};

    return { limit, offset, filter, sort };
}
```

Find user

```
  private async _find(req: Request, res: Response): Promise<void> {
    try {
1     const { limit, offset, filter, sort }: ParsedQuery = this._parseRequsetQuery(req.query);

2     const users: IUser[] = await User
        .find(filter)
        .skip(offset)
        .limit(limit)
        .sort(sort);

3     if (limit > 0) {
        //Content-Range: <unit> <range-start>-<range-end>/<size>
        const total: number = await User.count(filter);
        const contenRangeHeader: string = `users ${users.length ? offset + 1 : offset}-${offset + limit < total ? offset + limit : total}/${total}`;
        res.setHeader('Content-Range', contenRangeHeader);
      }

4     res.json(users);
    } catch (error) {
      res.status(500).json({ error });
    }
  }
```

Create user

```
private async _create(req: Request, res: Response): Promise<void> {
    try {
        const savedUser: IUser = await new User(req.body).save();
        res.json(savedUser);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

Find one user

```
private async _findOne(req: Request, res: Response): Promise<void> {
    try {
        const user: IUser = await User.findById(req.params.id)

        if (!user) {
            res.sendStatus(404);
            return;
        }

        res.json(user);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

Update user

```
private async _update(req: Request, res: Response): Promise<void> {
    try {
        const savedUser: IUser = await User.findOneAndUpdate(
            { _id: req.params.id },
            req.body,
            { new: true }
        );
        res.json(savedUser);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

Delete user

```typescript
private async _delete(req: Request, res: Response): Promise<void> {
    try {
        const removedUser: {ok?: number, n?: number} = await User.deleteOne({ _id: req.params.id });

        if (removedUser.n === 0) {
            res.sendStatus(404);
            return;
        }

        res.json(removedUser);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

NEW ENTITY

UNIVERSAL REQUEST HANDLER

Универсальный обработчик запросов

```
import { Request, Response } from "express";


export interface BaseRouterMethods {
    find(requset: Request, response: Response): Promise<void>;
    findOne(requset: Request, response: Response): Promise<void>;
    create(requset: Request, response: Response): Promise<void>;
    update(requset: Request, response: Response): Promise<void>;
    delete(requset: Request, response: Response): Promise<void>;
}
```

Конфигурирование обработчика

```
import { QueryPopulateOptions } from "mongoose";


export interface CommonServiceConfig {
    /**
     * Needs for build Content-Range header
     */
    entityName?: string;

    /**
     * Checking duplicate entity with this field
     */
    checkExists?: string;

    /**
     * Populate embedded entity
     */
    populate?: QueryPopulateOptions | QueryPopulateOptions[];
}
```

Универсальный обработчик запросов

```
export default abstract class CommonService<T extends Document> implements BaseRouterMethods {

    private _model: Model<T>;
    private _entityName: string;
    private _checkExists: string;
    private _populate: QueryPopulateOptions | QueryPopulateOptions[];

    constructor(
        model: Model<T>,
        config: CommonServiceConfig
    ) {
        this._model = model;

        this._checkExists = config.checkExists;
        this._entityName = config.entityName || 'entity';
        this._populate = config.populate || { path: '' };
    }
```

Универсальный обработчик запросов

```
async find(req: Request, res: Response): Promise<void> {
    try {
        const { limit, offset, filter, sort }: ParsedQuery = this._parseRequsetQuery(req.query);

        const entities: T[] = await this._model
            .find(filter)
            .skip(offset)
            .limit(limit)
            .sort(sort)
            .populate(this._populate);
```

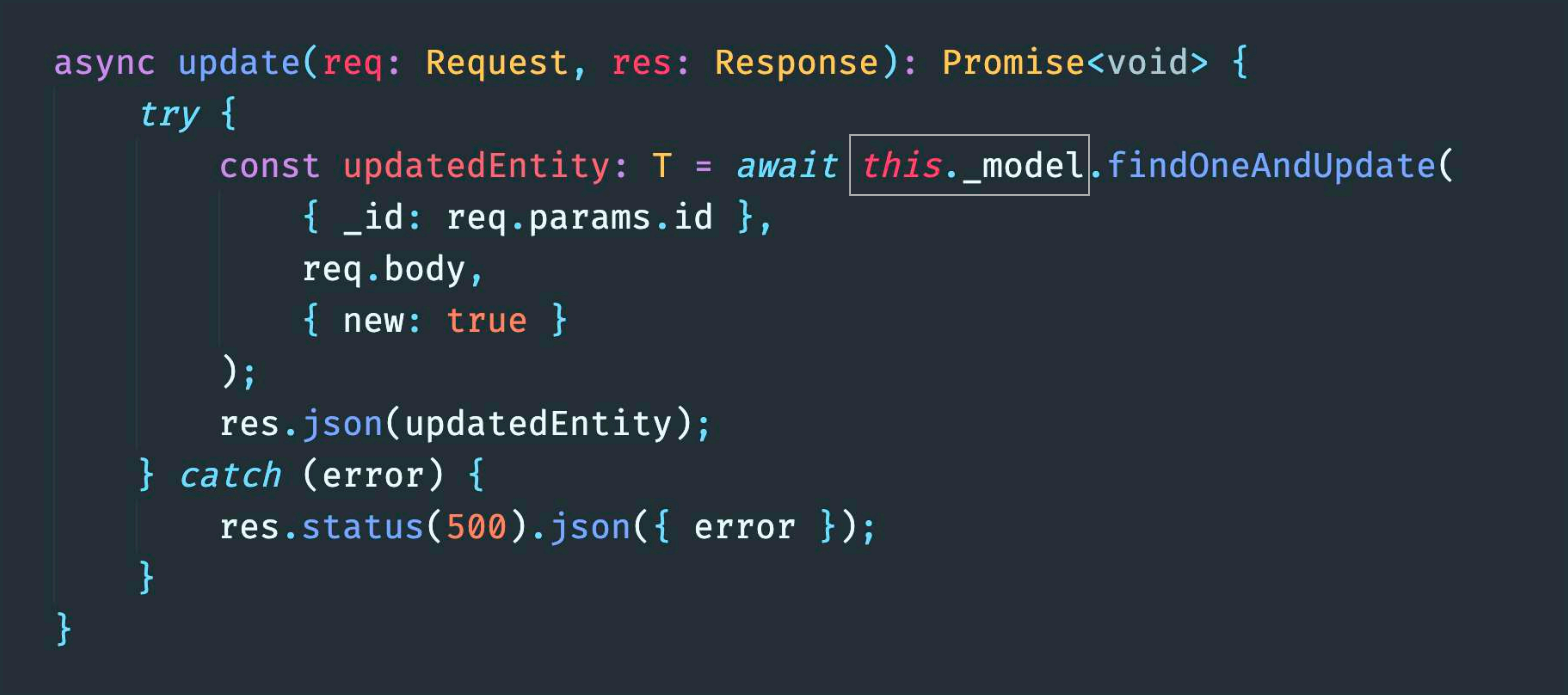
Универсальный обработчик запросов

```
async create(req: Request, res: Response): Promise<void> {
  try {
1   if (this._checkExists) {
      const exists: boolean = !!(await this._model.find({ [this._checkExists]: req.body[this._checkExists] })).length

      if (exists) {
       res.status(400).json({
        error: true,
        message: `${this._entityName} with ${req.body[this._checkExists]} ${this._checkExists} already exists.`
       });

       return;
      }
    }

2   const savedEntity: T = await new this._model(req.body).save();
    res.json(savedEntity);
  } catch (error) {
    res.status(500).json({ error });
  }
}
```

Универсальный обработчик запросов

```
async findOne(req: Request, res: Response): Promise<void> {
    try {
        const entity: T = await this._model
            .findById(req.params.id)
            .populate(this._populate);

        if (!entity) {
            res.sendStatus(404);
            return;
        }

        res.json(entity);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

Универсальный обработчик запросов



```
async update(req: Request, res: Response): Promise<void> {
    try {
        const updatedEntity: T = await this._model.findOneAndUpdate(
            { _id: req.params.id },
            req.body,
            { new: true }
        );
        res.json(updatedEntity);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

Универсальный обработчик запросов

```
async delete(req: Request, res: Response): Promise<void> {
    try {
        const removedEntity: {ok?: number, n?: number} = await this._model.deleteOne({ _id: req.params.id });

        if (removedEntity.n === 0) {
            res.sendStatus(404);
            return;
        }

        res.json(removedEntity);
    } catch (error) {
        res.status(500).json({ error });
    }
}
```

Универсальный обработчик запросов

```
export default class RouterHandler<Q extends Document> extends CommonService<Q> implements BaseRouterMethods

    protected _router: Router;

    public get router(): Router {
        return this._router;
    }

    constructor(
        model: Model<Q>,
        config: CommonServiceConfig
    ) {
        super(model, config);

        this._router = Router();

        this._router.get('/', this.find.bind(this));
        this._router.post('/', this.create.bind(this));
        this._router.get('/:id', this.findOne.bind(this));
        this._router.put('/:id', this.update.bind(this));
        this._router.delete('/:id', this.delete.bind(this));
    }
}
```

Обработчик запросов пользователей

```
import user from "../models/user";
import { User } from "../interfaces/user";
import RouterHandler from "./router-handler";


export default class UserRouterHandler extends RouterHandler<User> {
    constructor() {
        super(
            user,
            {
                entityName: 'users',
                checkExists: 'email'
            }
        );
    }
}
```

NEW ENTITY

Шаг 1 – Интерфейс Article

```
import { User } from "./user";
import { Document } from "mongoose";

export interface Article extends Document {
    name: string;
    slug: string;
    text: string;
    author: User;
    createdAt: string;
    updatedAt: string;
    category: { value: string }[];
    comments: { comment: string, date: string, author: User }[];
}
```

Шаг 2 – Схема Article

```
import { Schema, model } from "mongoose";
import { Article } from "../interfaces/article";


const articleSchema: Schema = new Schema(
    {
        name: { type: String, required: true },
        slug: { type: String, required: true, unique: true },
        text: { type: String, minlength: 50, maxlength: 5000 },
        author: { type: Schema.Types.ObjectId, ref: 'User' },
        category: [{ value: String }],
        comments: [
            {
                comment: { type: String, required: true },
                date: { type: Date, default: Date.now },
                author: { type: Schema.Types.ObjectId, ref: 'User' }
            }
        ],
    },
    { timestamps: true }
);

export default model<Article>('Article', articleSchema);
```

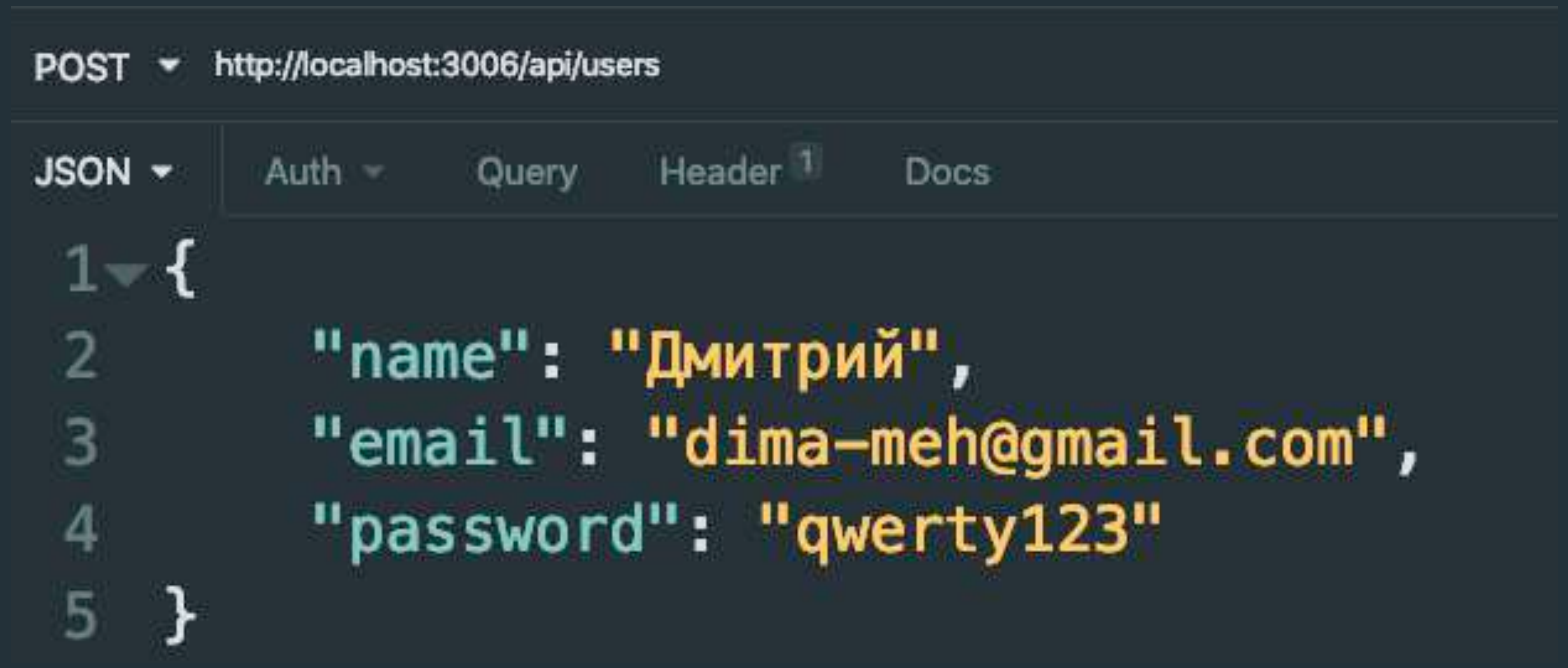
Шаг 3 – Маршрутизатор Article

```
import article from "../models/article";
import RouterHandler from "./router-handler";
import { Article } from "../interfaces/article";


export default class ArticleRouterHandler extends RouterHandler<Article> {
    constructor() {
        super(
            article,
            {
                entityName: 'articles',
                populate: {
                    path: 'author',
                    select: '-password'
                }
            }
        );
    }
}
```

Подключение маршрутизаторов

```
app.use('/api/users', new UserRouterHandler().router);
app.use('/api/articles', new ArticleRouterHandler().router);
```

Запись нового пользователя

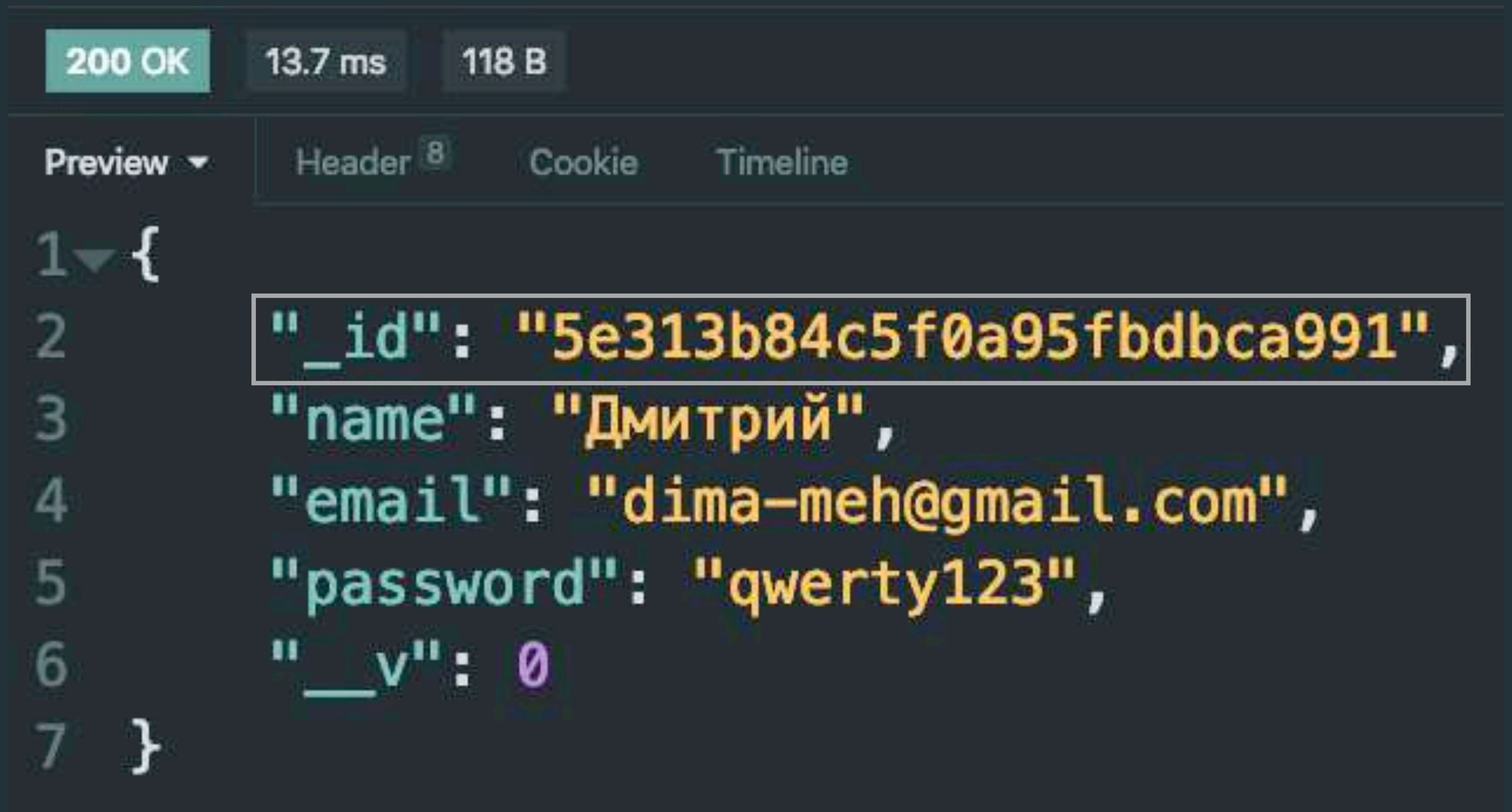


POST http://localhost:3006/api/users

JSON Auth Query Header 1 Docs

```
{
      "name": "Дмитрий",
      "email": "dima-meh@gmail.com",
      "password": "qwerty123"
}
```

$\Longrightarrow$

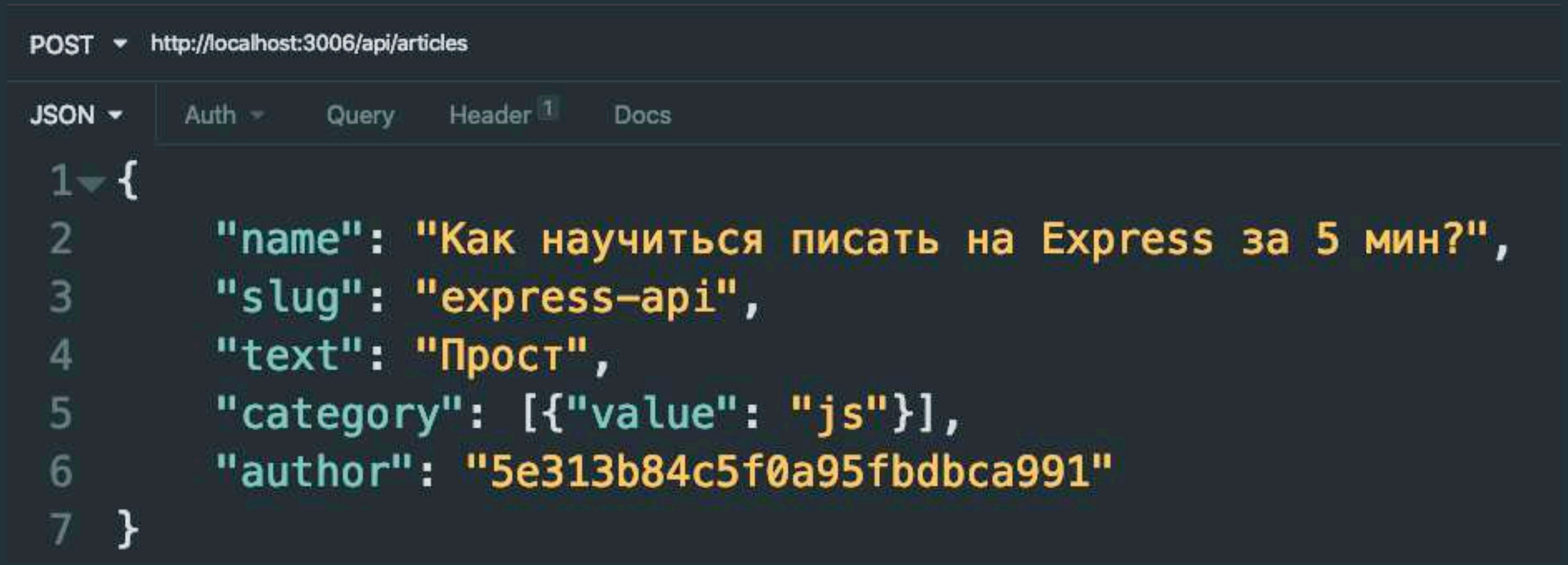


200 OK 13.7 ms 118 B

Preview Header 8 Cookie Timeline

```
{
      "_id": "5e313b84c5f0a95fbdbca991",
      "name": "Дмитрий",
      "email": "dima-meh@gmail.com",
      "password": "qwerty123",
      "__v": 0
}
```

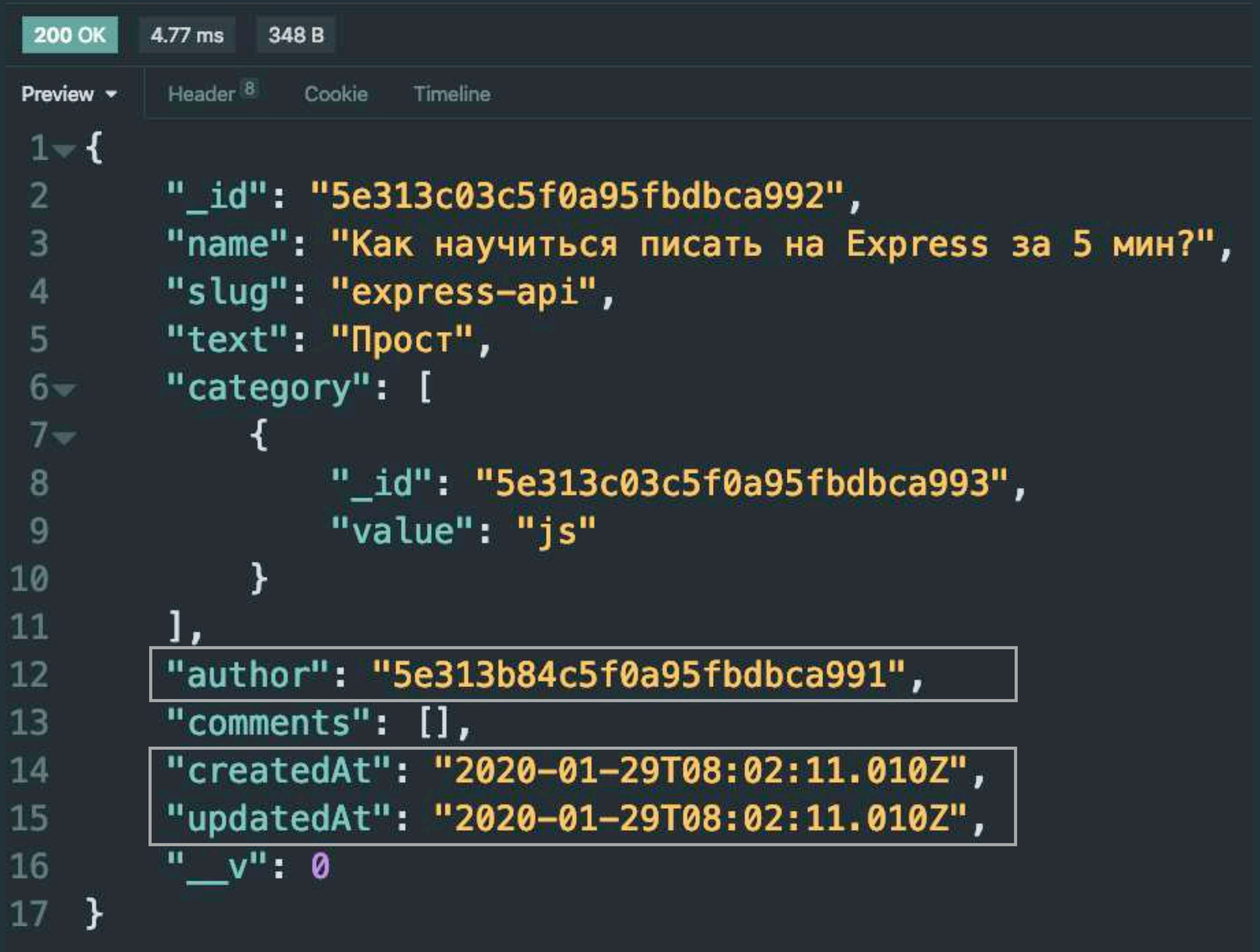
Запись новой статьи



http://localhost:3006/api/articles

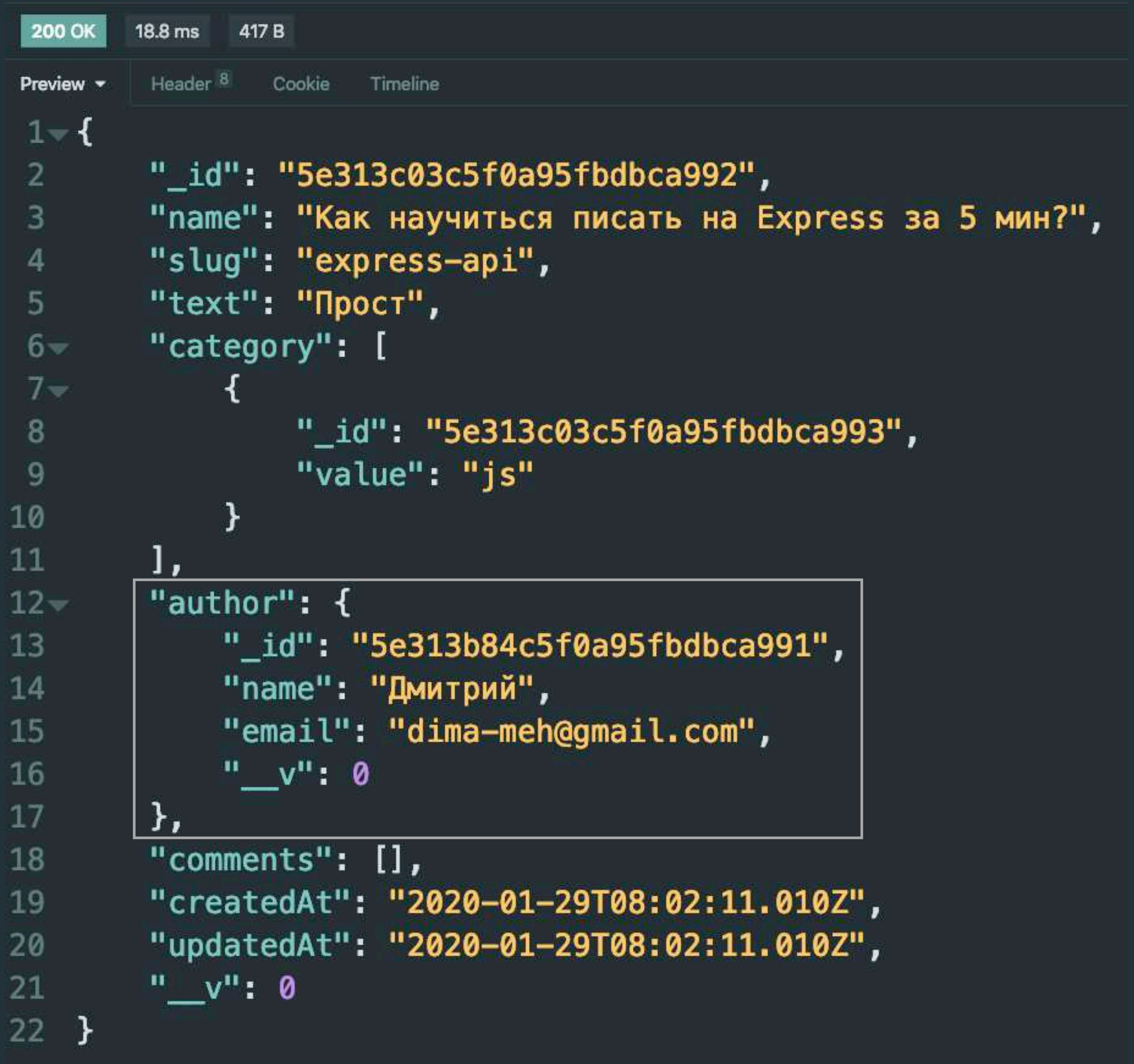
```
{
    "name": "Как научиться писать на Express за 5 мин?",
    "slug": "express-api",
    "text": "Прост",
    "category": [{"value": "js"}],
    "author": "5e313b84c5f0a95fbdbca991"
}
```

$\Longrightarrow$



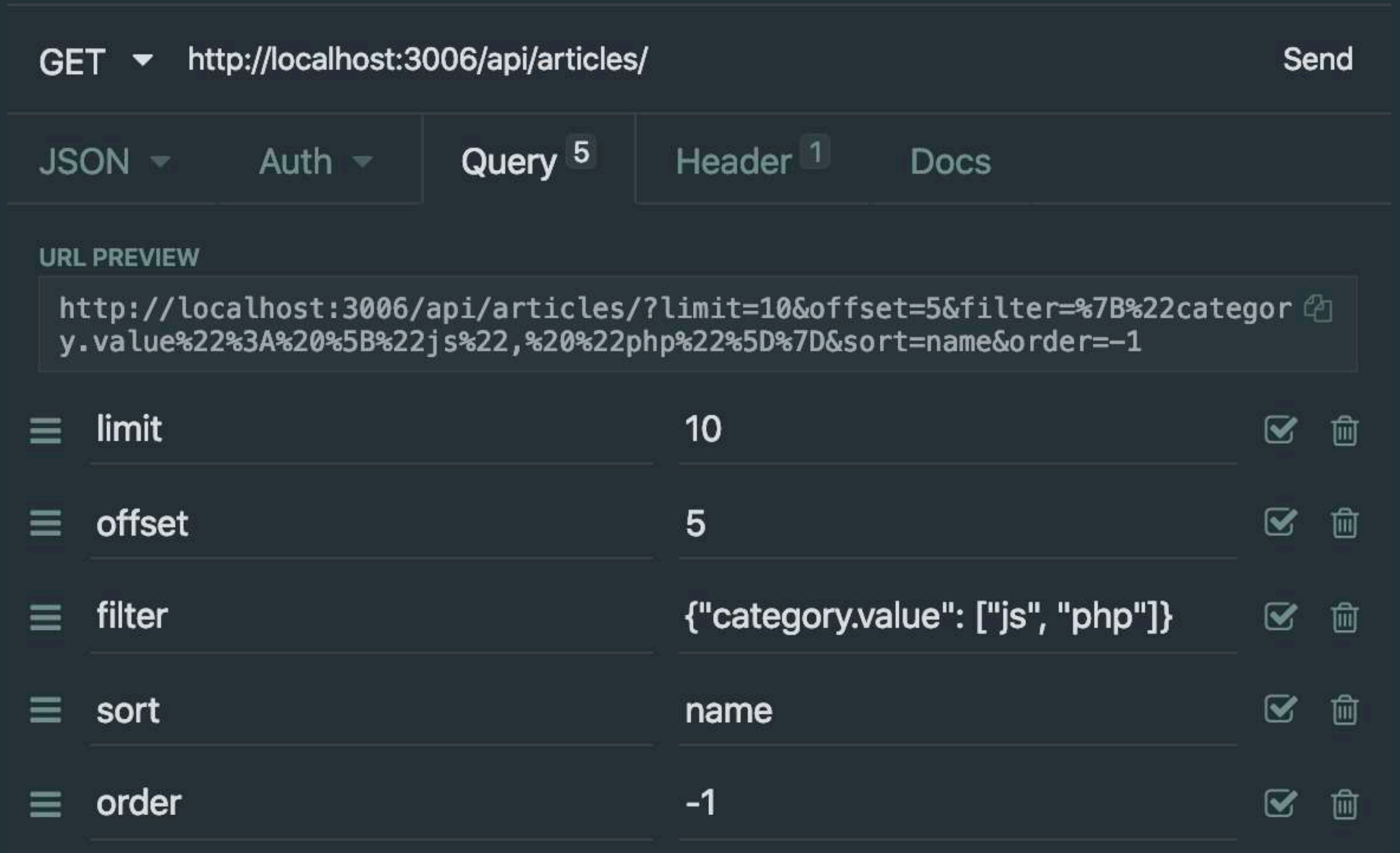
```
{
    "_id": "5e313c03c5f0a95fbdbca992",
    "name": "Как научиться писать на Express за 5 мин?",
    "slug": "express-api",
    "text": "Прост",
    "category": [
        {
            "_id": "5e313c03c5f0a95fbdbca993",
            "value": "js"
        }
    ],
    "author": "5e313b84c5f0a95fbdbca991",
    "comments": [],
    "createdAt": "2020-01-29T08:02:11.010Z",
    "updatedAt": "2020-01-29T08:02:11.010Z",
    "__v": 0
}
```

Получение статьи и автора



```
{
    "_id": "5e313c03c5f0a95fbdbca992",
    "name": "Как научиться писать на Express за 5 мин?",
    "slug": "express-api",
    "text": "Прост",
    "category": [
        {
            "_id": "5e313c03c5f0a95fbdbca993",
            "value": "js"
        }
    ],
    "author": {
        "_id": "5e313b84c5f0a95fbdbca991",
        "name": "Дмитрий",
        "email": "dima-meh@gmail.com",
        "__v": 0
    },
    "comments": [],
    "createdAt": "2020-01-29T08:02:11.010Z",
    "updatedAt": "2020-01-29T08:02:11.010Z",
    "__v": 0
}
```

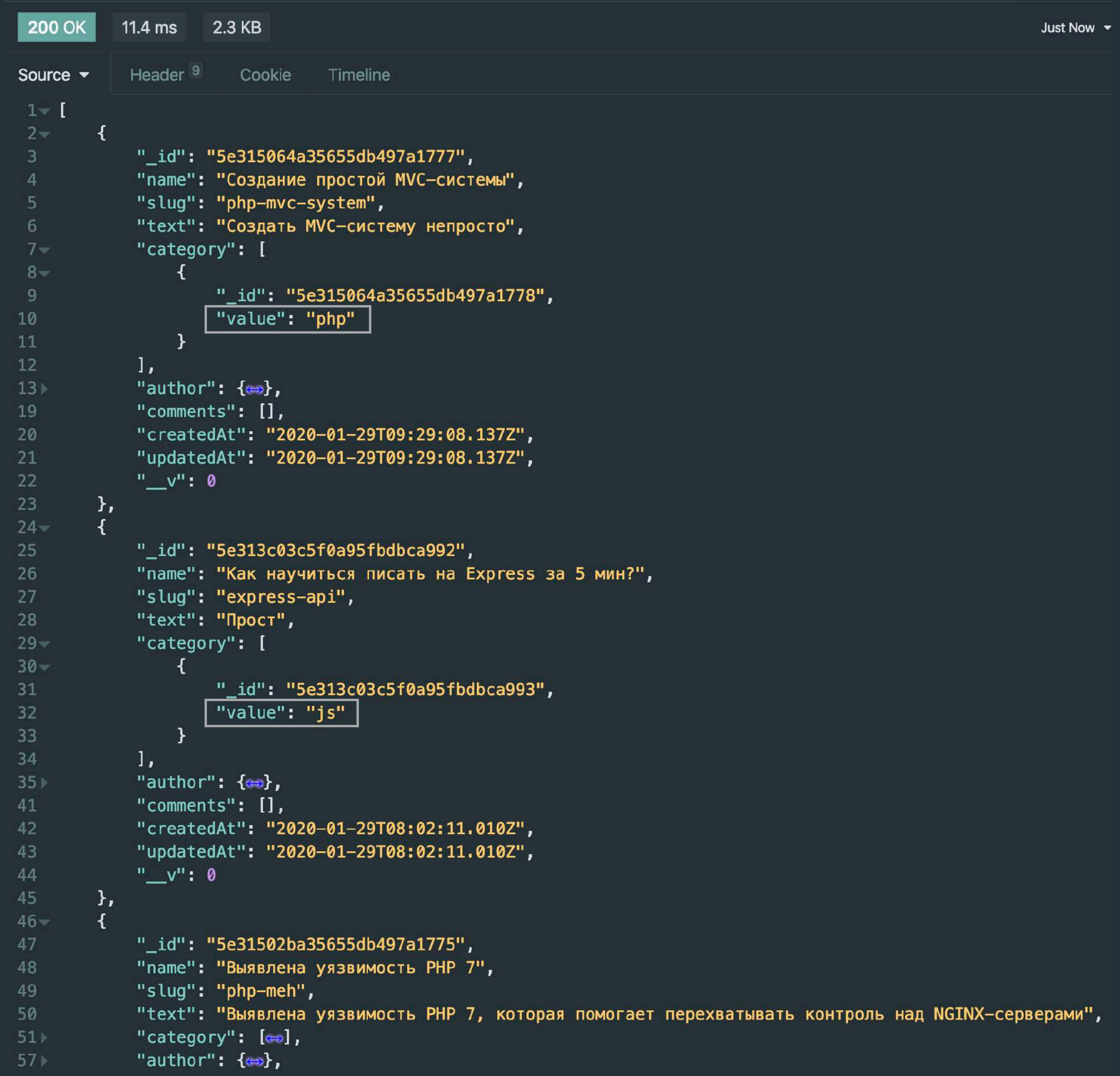
Сложная выборка



http://localhost:3006/api/articles/

```
http://localhost:3006/api/articles/?limit=10&offset=5&filter=%7B%22categor
y.value%22%3A%20%5B%22js%22,%20%22php%22%5D%7D&sort=name&order=-1
```

limit	10
offset	5
filter	{"category.value": ["js", "php"]}
sort	name
order	-1



```
[
    {
        "_id": "5e315064a35655db497a1777",
        "name": "Создание простой MVC-системы",
        "slug": "php-mvc-system",
        "text": "Создать MVC-систему непросто",
        "category": [
            {
                "_id": "5e315064a35655db497a1778",
                "value": "php"
            }
        ],
        "author": {↔},
        "comments": [],
        "createdAt": "2020-01-29T09:29:08.137Z",
        "updatedAt": "2020-01-29T09:29:08.137Z",
        "__v": 0
    },
    {
        "_id": "5e313c03c5f0a95fbdbca992",
        "name": "Как научиться писать на Express за 5 мин?",
        "slug": "express-api",
        "text": "Прост",
        "category": [
            {
                "_id": "5e313c03c5f0a95fbdbca993",
                "value": "js"
            }
        ],
        "author": {↔},
        "comments": [],
        "createdAt": "2020-01-29T08:02:11.010Z",
        "updatedAt": "2020-01-29T08:02:11.010Z",
        "__v": 0
    },
    {
        "_id": "5e31502ba35655db497a1775",
        "name": "Выявлена уязвимость PHP 7",
        "slug": "php-meh",
        "text": "Выявлена уязвимость PHP 7, которая помогает перехватывать контроль над NGINX-серверами",
        "category": [↔],
        "author": {↔},
```

Morgan

```
"dependencies": {
  "morgan": "^1.9.1",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "mongoose": "^5.8.7",
  "body-parser": "^1.19.0"
},
"devDependencies": {
  "nodemon": "^2.0.2",
  "typescript": "^3.7.4",
  "@types/dotenv": "^8.2.0",
  "@types/morgan": "^1.7.37",
  "@types/express": "^4.17.2",
  "@types/mongoose": "^5.5.41",
  "@types/body-parser": "^1.17.1"
}
```

Логирование

```
app.use(morgan('tiny'));
```

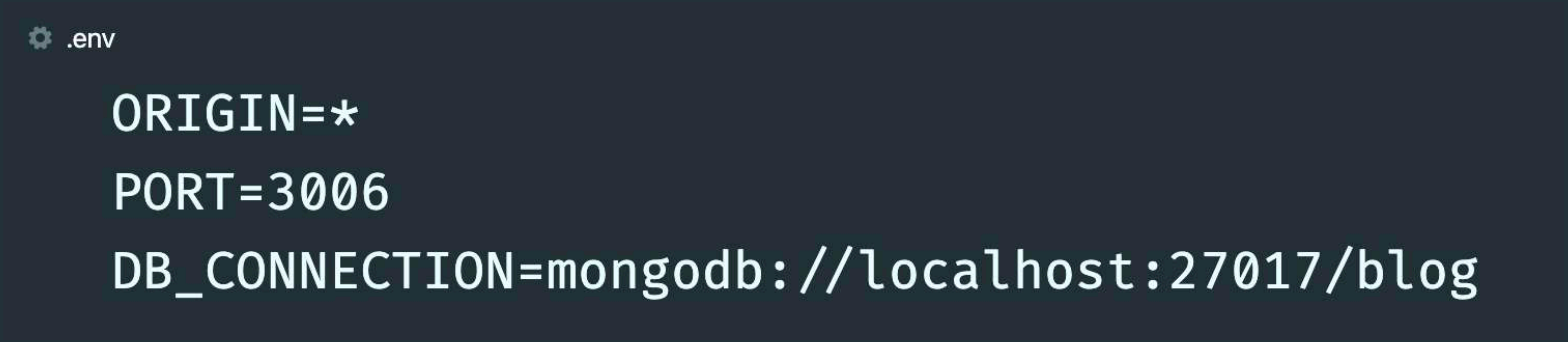
Логирование

```
/**
 * Выводим в консоль только 4xx и 5xx ответы
 */
this._app.use(morgan('dev', { skip: (req, res) ⇒ res.statusCode < 400 }));

/**
 * Записываем все логи в файл /log/access.log,
 * с интервалом в 3 дня
 */
const accessLogStream: WriteStream = rfs('access.log', {
    interval: '3d',
    path: path.join(__dirname, 'log')
});
this._app.use(morgan('common', { stream: accessLogStream }));
```

CORS

```
"dependencies": {
  "cors": "^2.8.5",
  "morgan": "^1.9.1",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "mongoose": "^5.8.7",
  "body-parser": "^1.19.0"
},
"devDependencies": {
  "nodemon": "^2.0.2",
  "typescript": "^3.7.4",
  "@types/cors": "^2.8.6",
  "@types/dotenv": "^8.2.0",
  "@types/morgan": "^1.7.37",
  "@types/express": "^4.17.2",
  "@types/mongoose": "^5.5.41",
  "@types/body-parser": "^1.17.1"
}
```

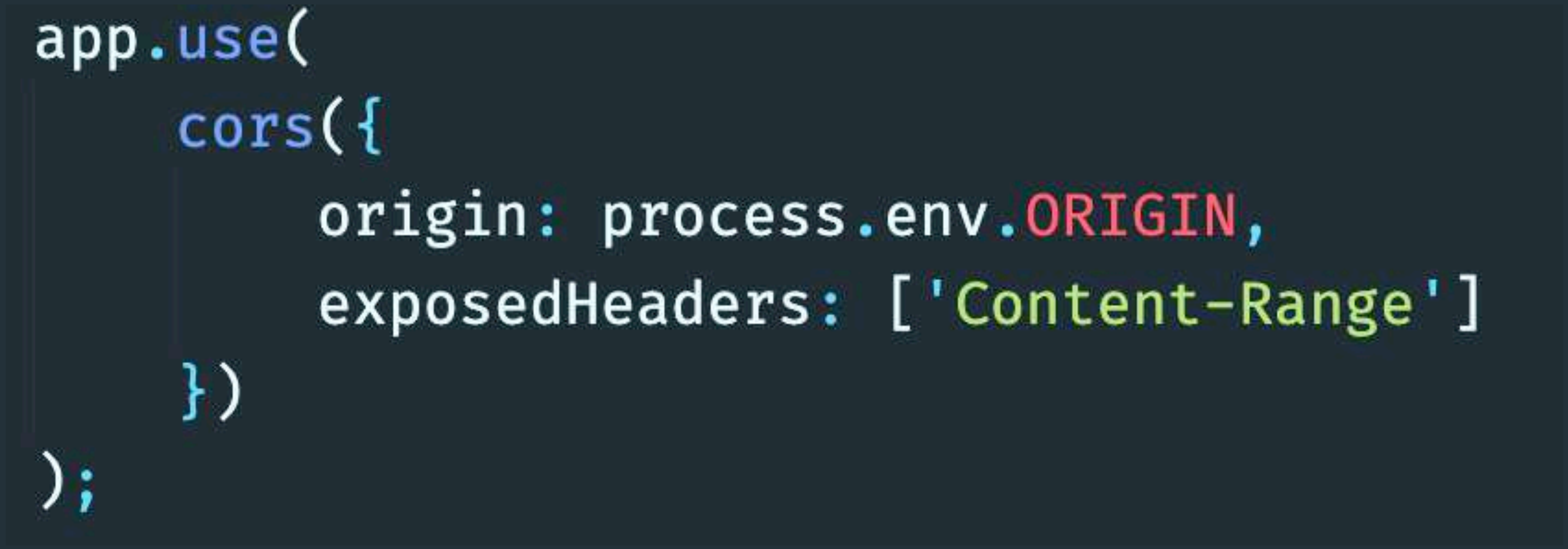
CORS



.env

```
ORIGIN=*
PORT=3006
DB_CONNECTION=mongodb://localhost:27017/blog
```

CORS



```
app.use(
    cors({
        origin: process.env.ORIGIN,
        exposedHeaders: ['Content-Range']
    })
);
```

Server

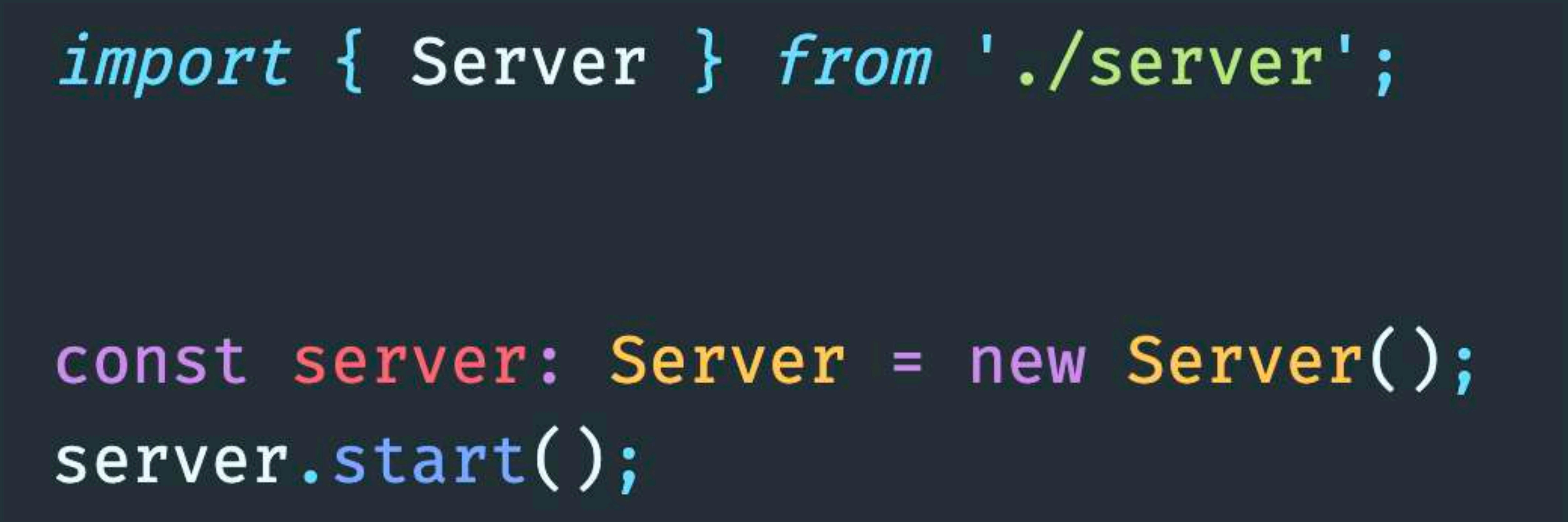
```
1  const config: DotenvConfigOutput = dotenv.config();
   if (config.error) {
       console.log('Локальный .env не найден')
   }

   const app: Application = express();

2  mongoose
       .connect(process.env.DB_CONNECTION,  { useNewUrlParser: true })
       .then(() => console.log('Успешное подключение к БД'))
       .catch(err => console.log('Произошла ошибка\n', err));

3  app.use(cors({ origin: process.env.ORIGIN, exposedHeaders: ['Content-Range'] }));

   app.use(bodyParser.json());

   app.use(morgan('tiny'));

4  app.use('/api/users', new UserRouterHandler().router);
   app.use('/api/articles', new ArticleRouterHandler().router);

5  app.listen(process.env.PORT, () => console.log(`Сервер стартовал на порту ${process.env.PORT}!`));
```

Server

```
  export class Server {
1     private _app: Application;

      constructor() {
          this._app = express();
      }

2     start(): void {
          this._loadEnv();
          this._setConnection();
          this._setMiddleware();
          this._setRouters();

          this._app.listen(process.env.PORT, () => console.log(`Сервер стартовал на порту ${process.env.PORT}!`));
      }

3     private _loadEnv(): void { }

4     private _setRouters(): void { }

5     private _setConnection(): void { }

6     private _setMiddleware(): void { }
  }
```

Сервер

ТОЧКА ВХОДА



```
import { Server } from './server';



const server: Server = new Server();
server.start();
```

Многоконтейнерное Docker-окружение

```yaml
version: '3'

services:
  mongo:
    image: mongo
    volumes:
      - ../databases/mongodb:/data/db

  api:
    image: node
    command: node /home/node/app/index.js
    volumes:
      - "./build:/home/node/app"
      - "./node_modules:/home/node/app/node_modules"
    ports:
      - "3006:3006"
    environment:
      - ORIGIN=*
      - PORT=3006
      - DB_CONNECTION=mongodb://mongo:27017/blog
```

Резюме

РЕПОЗИТОРИЙ