

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Егоров Мичил

Группа

К33401

Проверил:

Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Необходимо реализовать RESTful API средствами express + typescript.

Сайт криптобиржи:

- Вход
- Регистрация
- Портфель пользователя с указанием различных криптовалют и их количеством
- Графики роста криптовалют
- Поиск по криптовалютам с возможностью фильтрации по дате добавления на биржу

Ход работы

1. Структура приложения

```
(~/b/ITMO-ICT-Backend-2022/l/K33401/M/lab2) ➜ main tree
.
├── package-lock.json
└── package.json
src
├── controllers
│   ├── AuthController.ts
│   ├── PortfolioController.ts
│   ├── StockController.ts
│   └── UserController.ts
├── errors
│   └── ResponseError.ts
├── index.ts
└── orm
    ├── db.ts
    └── models
        ├── Portfolio.ts
        ├── PortfolioStock.ts
        ├── Stock.ts
        ├── StockHistory.ts
        ├── Token.ts
        └── User.ts
├── routes
│   ├── auth.ts
│   ├── index.ts
│   ├── portfolios.ts
│   ├── stocks.ts
│   └── users.ts
└── services
    ├── AuthService.ts
    ├── PortfolioService.ts
    ├── StockService.ts
    └── UserService.ts
tsconfig.json

7 directories, 25 files
```

2. Модели

The screenshot shows the VS Code interface with the User.ts file open in the editor. The Explorer sidebar on the left displays the project structure under the LAB2 folder, including src, orm, models, errors, db.ts, routes, auth.ts, index.ts, portfolios.ts, stocks.ts, users.ts, services, and various configuration files. The User.ts file is selected in the Explorer.

```
src > orm > models > User.ts > User
1 import {Column, Entity, OneToMany, OneToOne, PrimaryGeneratedColumn} from "typeorm";
2 import {Portfolio} from "./Portfolio";
3 import {AuthToken} from "./Token";
4
5 @Entity('User')
6 export class User {
7     @PrimaryGeneratedColumn()
8     id: number;
9
10    @Column({ unique: true })
11    email: string;
12
13    @Column()
14    password: string;
15
16    @OneToMany(() => Portfolio, (portfolio) => portfolio.user, {onDelete: "CASCADE"})
17    portfolios: Portfolio[];
18
19    @OneToOne(() => AuthToken, (token) => token.user, {onDelete: "CASCADE"})
20    authToken: AuthToken;
21 }
```

Bottom status bar: Ln 21, Col 2 | Spaces: 4 | UTF-8 | LF | TypeScript | Go Live | Spell | Prettier

The screenshot shows the VS Code interface with the Token.ts file open in the editor. The Explorer sidebar on the left displays the project structure under the LAB2 folder, including src, orm, models, errors, db.ts, routes, auth.ts, index.ts, portfolios.ts, stocks.ts, users.ts, services, and various configuration files. The Token.ts file is selected in the Explorer.

```
src > orm > models > Token.ts > AuthToken
1 import {Column, Entity, OneToOne, JoinColumn, PrimaryGeneratedColumn} from "typeorm";
2 import {User} from "./User";
3
4 @Entity('authToken')
5 export class AuthToken {
6     @PrimaryGeneratedColumn()
7     id: number;
8
9     @Column({ type: 'int' })
10    userId: number;
11
12    @ManyToOne(() => User, (user) => user.authToken, {onDelete: "CASCADE"})
13    @JoinColumn({ name: "userId" })
14    user: User;
15
16    @Column()
17    token: string;
18 }
```

Bottom status bar: Ln 15, Col 1 | Spaces: 4 | UTF-8 | LF | TypeScript | Go Live | Spell | Prettier

The screenshot shows the VS Code interface with the Stock.ts file open in the editor. The Explorer pane on the left displays the project structure under the LAB2 folder, including src, orm, models, db.ts, routes, services, and various controller and service files. The Stock.ts file is selected in the Explorer. The Editor pane shows the following TypeScript code:

```
src > orm > models > Stock.ts > Stock
1 import {PortfolioStock} from "./PortfolioStock";
2 import {Portfolio} from "./Portfolio";
3
4 @Entity('stock')
5 export class Stock {
6     @PrimaryGeneratedColumn()
7     id: number;
8
9     @Column()
10    name: string;
11
12     @Column()
13    description: string;
14
15     @Column({type: 'timestamp'})
16    created_at: Date;
17
18     @Column()
19    lastPrice: number;
20
21     @OneToMany(() => StockHistory, (stockHistory) => stockHistory.stock, {onDelete: "CASCADE"})
22    history: StockHistory[];
23
24     @OneToMany(() => PortfolioStock, (ps) => ps.stock, {onDelete: "CASCADE"})
25    portfolios: PortfolioStock[];
26
27 }
28 }
```

The status bar at the bottom indicates: Ln 22, Col 1 | Spaces: 4 | UTF-8 | LF | TypeScript | Go Live | 1 Spell | Prettier.

The screenshot shows the VS Code interface with the StockHistory.ts file open in the editor. The Explorer pane on the left displays the project structure under the LAB2 folder, including src, orm, models, db.ts, routes, services, and various controller and service files. The StockHistory.ts file is selected in the Explorer. The Editor pane shows the following TypeScript code:

```
src > orm > models > StockHistory.ts > StockHistory > price
1 import {Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn} from "typeorm";
2 import {Stock} from "./Stock";
3
4 @Entity('stockHistory')
5 export class StockHistory {
6     @PrimaryGeneratedColumn()
7     id: number;
8
9     @Column({ type: 'int' })
10    stock_id: number;
11
12     @ManyToOne(() => Stock, (stock) => stock.history, {onDelete: "CASCADE"})
13    @JoinColumn({ name: 'stock_id' })
14    stock: Stock;
15
16     @Column()
17    price: number;
18
19     @Column({type: 'timestamp'})
20    timestamp: Date;
21 }
```

The status bar at the bottom indicates: Ln 17, Col 10 (5 selected) | Spaces: 4 | UTF-8 | LF | TypeScript | Go Live | 1 Spell | Prettier.

Portfolio.ts

```

src > orm > models > Portfolio.ts > Portfolio > stocks
1 import {Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn} from "typeorm";
2 import {User} from "./User";
3 import {PortfolioStock} from "./PortfolioStock";
4 import {Stock} from "./Stock";
5
6 @Entity('portfolio')
7 export class Portfolio {
8     @PrimaryGeneratedColumn()
9     id: number;
10
11     @Column()
12     name: string;
13
14     @Column({type: 'int'})
15     user_id: number;
16
17     @Column({default: 0})
18     balance: number;
19
20     @ManyToOne(() => User, (user) => user.portfolios, {onDelete: "CASCADE"})
21     @JoinColumn({name: 'user_id'})
22     user: User;
23
24     @OneToMany(() => PortfolioStock, (ps) => ps.portfolio, {onDelete: "CASCADE"})
25     stocks: PortfolioStock[];
26 }

```

PortfolioStock.ts

```

src > orm > models > PortfolioStock.ts > PortfolioStock
1 import {Column, Entity, JoinColumn, ManyToOne, PrimaryGeneratedColumn} from "typeorm";
2 import {Portfolio} from "./Portfolio";
3 import {Stock} from "./Stock";
4
5 @Entity('PortfolioStock')
6 export class PortfolioStock {
7     @PrimaryGeneratedColumn()
8     id: number;
9
10    @Column({ type: 'int' })
11    portfolio_id: number;
12
13    @ManyToOne(() => Portfolio)
14    @JoinColumn({name: 'portfolio_id'})
15    portfolio: Portfolio;
16
17    @Column({ type: 'int' })
18    stock_id: number;
19
20    @ManyToOne(() => Stock, (stock) => stock.portfolios, {onDelete: "CASCADE"})
21    @JoinColumn({name: 'stock_id'})
22    stock: Stock;
23
24    @Column({type: 'timestamp'})
25    timestamp: Date;
26
27    @Column()
28    price: number;
29
30    @Column({default: 0})
31    amount: number;
32 }

```

3. Контроллеры

The screenshot shows the VS Code interface with the file `AuthController.ts` open in the editor. The code defines a class `AuthController` with methods for token authentication, changing tokens, and deleting entries.

```
4  class AuthController {
5    private entityService: AuthService
6
7    constructor() {
8      this.entityService = new AuthService()
9    }
10
11   auth = async (request: any, response: any) => {
12     try {
13       const token = await this.entityService.auth(request.body)
14       response.send({token: token.token})
15     } catch (error: any) {
16       response.status(404).send({
17         "error": error.message
18       })
19     }
20   }
21
22   changeToken = async (request: any, response: any) => {
23     try {
24       const token = await this.entityService.changeToken(request.body)
25       response.send({token: token.token})
26     } catch (error: any) {
27       response.status(404).send({
28         "error": error.message
29       })
30     }
31   }
32
33   delete = async (request: any, response: any) => {
34     try {
35       const token = await this.entityService.delete(request.body)
36       response.send({status: 'ok'})
37     } catch (error: any) {
38       response.status(404).send({
39         "error": error.message
40       })
41     }
42   }
}
```

At the bottom, status bar indicators show: Ln 13, Col 54 | Spaces: 4 | UTF-8 | LF | (TypeScript | Go Live | Spell | Prettier).

The screenshot shows the VS Code interface with the file `UserController.ts` open in the editor. The code defines a class `UserController` with methods for getting a user by ID, listing all users, and creating a new user.

```
1  import {UserService} from '../services/UserService'
2  import {Request} from "express";
3
4  class UserController {
5    private userService: UserService
6
7    constructor() {
8      this.userService = new UserService()
9    }
10
11   get = async (request: any, response: any) => {
12     try {
13       const user = await this.userService.getById(
14         +request.params.id
15       )
16       response.send(user)
17     } catch (error: any) {
18       response.status(404).send({
19         "error": error.message
20       })
21     }
22   }
23
24   list = async (request: any, response: any) => {
25     try {
26       const users = await this.userService.list()
27       response.send({users: users})
28     } catch (error: any) {
29       response.status(404).send({
30         "error": error.message
31       })
32     }
33   }
34
35   create = async (request: any, response: any) => {
36     try {
37       const user = await this.userService.create({
38         email: request.body.email,
39         password: request.body.password
40       })
41       response.send(user)
42     } catch (error: any) {
43       response.status(404).send({
44         "error": error.message
45       })
46     }
47   }
48
49 }
```

At the bottom, status bar indicators show: Ln 27, Col 42 (29 selected) | Spaces: 4 | UTF-8 | LF | (TypeScript | Go Live | Spell | Prettier).

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying the project structure under 'LAB2'. The 'src' folder contains several controller files: AuthController.ts, PortfolioController.ts, StockController.ts, and UserController.ts. The UserController.ts file is currently selected and open in the main editor area. The code implements four methods: 'create', 'delete', 'update', and 'list'. Each method uses an asynchronous function to interact with a 'userService'. The 'create' method handles email and password validation. The 'delete' method removes a user by ID. The 'update' method updates a user's information. The 'list' method retrieves all users.

```
UserController.ts
35     create = async (request: any, response: any) => {
36         try {
37             const user = await this.userService.create({ email: request.body.email, password: request.body.password });
38             response.send(user);
39         } catch (error: any) {
40             response.status(400).send({
41                 "error": error.message
42             })
43         }
44     }
45
46     delete = async (request: any, response: any) => {
47         try {
48             const token = await this.userService.delete(+request.params.id);
49             response.send({ status: 'ok' });
50         } catch (error: any) {
51             response.status(404).send({
52                 "error": error.message
53             })
54         }
55     }
56
57     update = async (request: any, response: any) => {
58         try {
59             const user = await this.userService.update({
60                 id: +request.params.id,
61                 ...request.body,
62             });
63             response.send(user);
64         } catch (error: any) {
65             response.status(400).send({
66                 "error": error.message
67             })
68         }
69     }
70 }
71
72 export default UserController
```

The screenshot shows the VS Code interface with the Explorer sidebar open, displaying the project structure under 'LAB2'. The 'src' folder contains several controller files: AuthController.ts, StockController.ts, UserController.ts, and PortfolioController.ts. The PortfolioController.ts file is currently selected and open in the main editor area. The code defines a 'PortfolioController' class with methods for 'get', 'list', 'create', and 'buy'. The 'get' method retrieves a stock by ID. The 'list' method retrieves all stocks. The 'create' method adds a new stock. The 'buy' method is partially implemented.

```
PortfolioController.ts
1 import PortfolioService from '../services/PortfolioService'
2
3 class PortfolioController {
4     private entityService: PortfolioService
5
6     constructor() {
7         this.entityService = new PortfolioService()
8     }
9
10    get = async (request: any, response: any) => {
11        try {
12            const stock = await this.entityService.getById(
13                +request.params.id
14            );
15            response.send(stock);
16        } catch (error: any) {
17            response.status(404).send({
18                "error": error.message
19            })
20        }
21    }
22
23    list = async (request: any, response: any) => {
24        try {
25            const portfolios = await this.entityService.list();
26            response.send({ portfolios });
27        } catch (error: any) {
28            response.status(404).send({
29                "error": error.message
30            })
31        }
32    }
33
34    create = async (request: any, response: any) => {
35        try {
36            const stock = await this.entityService.create(
37                request.body
38            );
39        } catch (error: any) {
40            response.status(404).send({
41                "error": error.message
42            })
43        }
44    }
45
46    buy = async (request: any, response: any) => {
47        try {
48            const stock = await this.entityService.buy(
49                +request.params.id
50            );
51            response.send(stock);
52        } catch (error: any) {
53            response.status(404).send({
54                "error": error.message
55            })
56        }
57    }
58}
```

PortfolioController.ts

```
src > controllers > PortfolioController.ts > PortfolioController > buy > stock
```

```
35  create = async (request: any, response: any) => {
36    try {
37      const stock = await this.entityService.create(
38        request.body
39      )
40      response.send(stock)
41    } catch (error: any) {
42      response.status(404).send({
43        "error": error.message
44      })
45    }
46  }

47 exchangeMoney = async (request: any, response: any) => {
48  try {
49    const portfolio = await this.entityService.exchange(
50      request.body
51    )
52    response.send(portfolio)
53  } catch (error: any) {
54    response.status(404).send({
55      "error": error.message
56    })
57  }
58}

59 buy = async (request: any, response: any) => {
60  try {
61    const stock = await this.entityService.buyStock(
62      request.body
63    )
64    response.send(stock)
65  } catch (error: any) {
66    response.status(404).send({
67      "error": error.message
68    })
69  }
70}

71
72}
73
```

Ln 63, Col 56 Spaces: 4 UTF-8 LF ⓘ TypeScript ⓘ Go Live ⓘ Spell ⓘ Prettier ⓘ

PortfolioController.ts

```
src > controllers > PortfolioController.ts > PortfolioController > sell > stock
```

```
74  sell = async (request: any, response: any) => {
75    try {
76      const stock = await this.entityService.sellStock(
77        request.body
78      )
79      response.send(stock)
80    } catch (error: any) {
81      response.status(404).send({
82        "error": error.message
83      })
84    }
85  }

86 delete = async (request: any, response: any) => {
87  try {
88    const portfolio = await this.entityService.delete(+request.params.id)
89    response.send({status: 'ok'});
90  } catch (error: any) {
91    response.status(404).send({
92      "error": error.message
93    })
94  }
95}

96 update = async (request: any, response: any) => {
97  try {
98    const portfolio = await this.entityService.update({
99      id: +request.params.id,
100     ...request.body,
101   })
102   response.send(portfolio);
103 } catch (error: any) {
104   response.status(404).send({
105     "error": error.message
106   })
107 }
108}

109
110}
111}
112}
```

Ln 63, Col 56 Spaces: 4 UTF-8 LF ⓘ TypeScript ⓘ Go Live ⓘ Spell ⓘ Prettier ⓘ

The screenshot shows the VS Code interface with the StockController.ts file open in the editor. The code implements a StockController class with methods for getting, listing, and creating stocks.

```
1 import StockService from '../services/StockService'
2
3
4 class StockController {
5   private stockService: StockService
6
7   constructor() {
8     this.stockService = new StockService()
9   }
10
11   get = async (request: any, response: any) => {
12     try {
13       const stock = await this.stockService.getById(
14         +request.params.id
15     )
16     response.send(stock)
17   } catch (error: any) {
18     response.status(404).send({
19       "error": error.message
20     })
21   }
22 }
23
24   list = async (request: any, response: any) => {
25     try {
26       const stocks = await this.stockService.list(
27         new Date(request.params.at_least | 0)
28     )
29     response.send({stocks: stocks})
30   } catch (error: any) {
31     response.status(404).send({
32       "error": error.message
33     })
34   }
35 }
36
37   create = async (request: any, response: any) => {
38     try {
39       const stock = await this.stockService.create(
40         request.body
41     )
42     response.send(stock)
43   } catch (error: any) {
44     response.status(404).send({
45       "error": error.message
46     })
47   }
48 }
49
50   getStockHistory = async (request: any, response: any) => {
51     try {
52       const stocks = await this.stockService.getStockHistory(
53         +request.params.id
54     )
55     response.send({stocks})
56   } catch (error: any) {
57     response.status(404).send({
58       "error": error.message
59     })
60   }
61 }
62
63   updatePrice = async (request: any, response: any) => {
64     try {
65       const stock = await this.stockService.updatePrice(
66         request.body
67     )
68     response.send(stock)
69   } catch (error: any) {
70     response.status(404).send({
71       "error": error.message
72     })
73   }
74 }
```

The screenshot shows the VS Code interface with the StockController.ts file open in the editor. The code has been modified to include three new methods: create, getStockHistory, and updatePrice.

```
37   create = async (request: any, response: any) => {
38     try {
39       const stock = await this.stockService.create(
40         request.body
41     )
42     response.send(stock)
43   } catch (error: any) {
44     response.status(404).send({
45       "error": error.message
46     })
47   }
48 }
49
50   getStockHistory = async (request: any, response: any) => {
51     try {
52       const stocks = await this.stockService.getStockHistory(
53         +request.params.id
54     )
55     response.send({stocks})
56   } catch (error: any) {
57     response.status(404).send({
58       "error": error.message
59     })
60   }
61 }
62
63   updatePrice = async (request: any, response: any) => {
64     try {
65       const stock = await this.stockService.updatePrice(
66         request.body
67     )
68     response.send(stock)
69   } catch (error: any) {
70     response.status(404).send({
71       "error": error.message
72     })
73   }
74 }
```

```

src / controllers > StockController.ts > StockController > updatePrice > stock
68     )
69     response.send(stock)
70   } catch (error: any) {
71     response.status(404).send({
72       "error": error.message
73     })
74   }
75
76   delete = async (request: any, response: any) => {
77     try {
78       const stock = await this.stockService.delete(+request.params.id)
79       response.send({status: 'ok'});
80     } catch (error: any) {
81       response.status(404).send({
82         "error": error.message
83       })
84     }
85   }
86
87   update = async (request: any, response: any) => {
88     try {
89       const stock = await this.stockService.update({
90         id: +request.params.id,
91         ...request.body,
92       })
93       response.send(stock);
94     } catch (error: any) {
95       response.status(404).send({
96         "error": error.message
97       })
98     }
99   }
100 }
101
102 export default StockController

```

Ln 65, Col 58 Spaces: 4 UTF-8 LF ⓘ TypeScript ⓘ Go Live ⓘ Spell ⓘ Prettier ⓘ

4. Routes

```

src > routes > index.ts > ...
1 import express from 'express';
2 import userRoutes from './users';
3 import stockRoutes from './stocks';
4 import portfolioRoutes from './portfolios';
5 import authRoutes from './auth';
6
7 const router = express.Router();
8
9 router.use('/users', userRoutes);
10 router.use('/stocks', stockRoutes);
11 router.use('/portfolios', portfolioRoutes);
12 router.use('/auth', authRoutes);
13
14 export default router;

```

```

src > routes > portfolios.ts > ...
1 import express from 'express';
2 import PortfolioController from './controllers/PortfolioController';
3
4 const router = express.Router();
5
6 const controller = new PortfolioController();
7
8 router.route('/:id')
9   .get(controller.get)
10  .post(controller.create)
11  .put(controller.update)
12  .patch(controller.patch)
13  .delete(controller.delete)
14
15 router.route('/')
16   .post(controller.list)
17
18 router.route('/exchange')
19   .post(controller.exchange)
20
21 router.route('/buy')
22   .post(controller.buy)
23
24 router.route('/sell')
25   .post(controller.sell)
26
27 router.route('/:id')
28   .delete(controller.delete)
29
30 router.route('/:id')
31   .post(controller.update)
32
33 export default router;

```

```

src > routes > auth.ts > ...
1 import express from 'express';
2 import AuthController from './controllers/AuthController';
3
4 const router = express.Router();
5
6 const controller = new AuthController();
7
8 router.route('/')
9   .post(controller.login)
10  .post(controller.register)
11
12 router.route('/change-password')
13   .post(controller.changePassword)
14
15 router.route('/logout')
16   .post(controller.logout)
17
18 export default router;

```

```

src > routes > stocks.ts > ...
1 import express from 'express';
2 import StockController from './controllers/StockController';
3
4 const router = express.Router();
5
6 const controller = new StockController();
7
8 router.route('/')
9   .post(controller.create)
10  .put(controller.update)
11  .patch(controller.patch)
12  .delete(controller.delete)
13
14 router.route('/:id')
15   .get(controller.get)
16
17 export default router;

```

```

src > routes > users.ts > ...
2 import UserController from './controllers/UserController';
3
4 const router = express.Router();
5
6 const controller = new UserController();
7
8 router.route('/:id')
9   .get(controller.get)
10  .put(controller.update)
11  .patch(controller.patch)
12  .delete(controller.delete)
13
14 router.route('/')
15   .post(controller.create)
16
17 router.route('/histories')
18   .get(controller.getHistories)
19
20 router.route('/update-profile')
21   .post(controller.updateProfile)
22
23 router.route('/:id')
24   .patch(controller.patch)
25
26 export default router;

```

Ln 9, Col 21 Spaces: 2 UTF-8 LF ⓘ TypeScript ⓘ Go Live ⓘ Spell ⓘ Prettier ⓘ

5. Сервисы

The screenshot shows the VS Code interface with the file `AuthService.ts` open in the editor. The code implements an `AuthService` class with methods for authentication, changing tokens, and deleting user data. It uses `TypeScript` and `TypeORM` for database interactions.

```
AuthService.ts
src > services > AuthService.ts ...
1 import { UserService } from './UserService';
2 import { StockService } from './StockService';
3 import { PortfolioService } from './PortfolioService';
4 import { StockHistory } from './StockHistory';
5 import { Token } from './Token';
6 import { User } from './User';
7 import { getRepository } from 'typeorm';
8 import { AuthToken } from './Auth';
9 import { Connection } from 'typeorm';
10 import { MinKey } from 'typeorm';
11
12 class AuthService {
13   public async auth(userData: {email: string, password: string}): Promise<AuthToken> {
14     const user = await new UserService().getByEmail(userData.email);
15
16     if(user.password !== userData.password) {
17       throw new Error("Password is not correct");
18     }
19
20     const tokenRepo = getConnection().getRepository(AuthToken);
21
22     return await tokenRepo.findOne({where: {userId: user.id}}).then(async (token) => {
23       if(token) {
24         return token;
25       }
26
27       return tokenRepo.save({
28         user: user,
29         token: uuid.v4()
30     });
31     });
32   }
33
34   public async changeToken(userData: {email: string, password: string}): Promise<AuthToken> {
35     const token = await this.auth(userData);
36     token.token = uuid.v4();
37     return getConnection().getRepository(AuthToken).save(token);
38   }
39
40   public async delete(userData: {email: string, password: string}): Promise<void> {
41     const entity = await this.auth(userData);
42     await getConnection().getRepository(AuthToken).remove(entity);
43   }
44 }
45
46 export default AuthService;
```

The screenshot shows the VS Code interface with the file `PortfolioService.ts` open in the editor. The code implements a `PortfolioService` class with methods for getting portfolios by ID, creating new portfolios, and manipulating stocks. It uses `TypeScript` and `TypeORM` for database interactions.

```
PortfolioService.ts M
src > services > PortfolioService.ts ...
1 import { Portfolio } from '../orm/models/Portfolio';
2 import { PortfolioStock } from '../orm/models/PortfolioStock';
3 import { UserService } from './UserService';
4 import { StockService } from './StockService';
5 import { getConnection, MinKey } from 'typeorm';
6
7 class PortfolioService {
8   public getPortfolioRepo() {
9     return getConnection().getRepository(Portfolio);
10 }
11
12 public async getById(id: number): Promise<Portfolio> {
13   return await this.getPortfolioRepo().findOne({where: {id: id}}).then(async (portfolio) => {
14     if (!portfolio) {
15       throw new Error(`Portfolio (id=${id}) does not exist`);
16     }
17     portfolio['stocks'] = await getConnection().getRepository(PortfolioStock).find({
18       where: {
19         portfolio_id: portfolio.id
20       }
21     });
22     return portfolio;
23   });
24 }
25
26 public async create(portfolioData: {name: string, user_id: number}): Promise<Portfolio> {
27   const user = await new UserService().getById(portfolioData.user_id);
28   const entityRepo = getConnection().getRepository(Portfolio);
29
30   return entityRepo.save({user: user, name: portfolioData.name});
31 }
32
33 private async manipulateStock(dealInfo: {portfolio_id: number, stock_id: number, amount: number, isSold: boolean}) {
34   const stock = await new StockService().getById(dealInfo.stock_id);
35   const portfolio = await this.getById(dealInfo.portfolio_id);
36
37   portfolio.balance += stock.lastPrice * dealInfo.amount * (dealInfo.isSold ? 1 : -1);
38 }
```

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under the 'src' folder. The 'services' folder contains 'AuthService.ts', 'PortfolioService.ts', 'StockService.ts', and 'UserService.ts'. Other files like 'index.ts', 'portfolio.ts', 'stocks.ts', and 'users.ts' are also listed.
- Editor View:** Displays the code for 'PortfolioService.ts'. The code handles portfolio transactions, including buying and selling stocks, updating balances, and managing deals. It uses promises and async/await syntax.
- Bottom Status Bar:** Shows 'Ln 1, Col 1' and other status indicators like 'Spaces: 4', 'UTF-8', 'LF', 'TypeScript', 'Go Live', 'Spell', and 'Prettier'.

```
if(portfolio.balance < 0) {
    throw new Error("Not enough balance")
}

// start transaction
await getConnection().transaction(async transactionalEntityManager => {
    const stocksRepo = getConnection().getRepository(PortfolioStock);

    if(!dealInfo.isSold) {
        await stocksRepo.save({
            portfolio: portfolio,
            stock: stock,
            timestamp: new Date(),
            price: stock.lastPrice,
            amount: dealInfo.amount,
        });
    } else {
        let amount = dealInfo.amount;
        let toRemoveStocks = [];

        for(let portfolio_stock of await stocksRepo.find({where: {
            portfolio_id: portfolio.id,
            stock_id: stock.id,
        }})) {
            if(amount == 0) {
                break;
            }

            let minus_amount = Math.min(portfolio_stock.amount, amount);
            portfolio_stock.amount -= minus_amount;
            amount -= minus_amount;

            stocksRepo.save(portfolio_stock);

            if(portfolio_stock.amount == 0) {
                toRemoveStocks.push(portfolio_stock);
            }
        }
    }
}
```

This screenshot shows the same project structure and editor view as the first one, but with more code visible in the editor pane. The code includes methods for buying, selling, and updating portfolios, as well as a list method and an update method that performs a save operation.

```
public async buyStock(dealInfo: {portfolio_id: number, stock_id: number, amount: number}): Promise<Portfolio> {
    return await this.manipulateStock({...dealInfo, isSold: false});
}

public async sellStock(dealInfo: {portfolio_id: number, stock_id: number, amount: number}): Promise<Portfolio> {
    return await this.manipulateStock({...dealInfo, isSold: true});
}

public async exchange(exchangeInfo: {portfolio_id: number, balance: number}): Promise<Portfolio> {
    const portfolio = await this.getById(exchangeInfo.portfolio_id);
    portfolio.balance += exchangeInfo.balance;

    return await this.getPortfolioRepo().save(portfolio);
}

public async delete(id: number): Promise<void> {
    const entity = await this.getById(id);
    await getConnection().getRepository(Portfolio).remove(entity);
}

public async list(): Promise<Portfolio[]> {
    return await getConnection().getRepository(Portfolio).find();
}

public async update(portfolioData: {id: number, balance: number, user_id: number}): Promise<Portfolio> {
    const user = await new UserService().getById(portfolioData.user_id);
    const portfolio = await this.getById(portfolioData.id);

    return await getConnection().getRepository(Portfolio).save({
        id: portfolioData.id,
        balance: portfolioData.balance + portfolio.balance,
        user: user,
    });
}

export default PortfolioService
```



EXPLORER ... StockService.ts X

src > services > StockService.ts ...

OPEN EDITORS

LAB2

controllers

- AuthController.ts
- PortfolioController.ts
- StockController.ts
- UserController.ts

errors

orm

- models
- Portfolio.ts
- PortfolioStock.ts
- Stock.ts
- StockHistory.ts
- Token.ts
- User.ts

db.ts

routes

- auth.ts
- index.ts
- portfolios.ts
- stocks.ts
- users.ts

services

- AuthService.ts
- PortfolioService.ts
- StockService.ts
- UserService.ts

index.ts

* .env

* .env.sample

.gitignore

package-lock.json

package.json

tsconfig.json

OUTLINE

TIMELINE

StockService.ts

```
38    }
39
40    public async create(stockData: { name: string, description: string, lastPrice: number}): Promise<Stock> {
41        const entityRepo = getConnection().getRepository(Stock);
42
43        await entityRepo.findOne({ where: { name: stockData.name } }).then((stock) => {
44            if (stock) {
45                throw new Error('Stock with such name already exists');
46            }
47        });
48
49        const created_at = new Date();
50        const stock = await entityRepo.save({...stockData, created_at: created_at});
51        this.addHistory(stock);
52
53        return stock;
54    }
55
56    public async delete(id: number): Promise<void> {
57        const entity = await this.getById(id);
58        await getConnection().getRepository(Stock).remove(entity);
59    }
60
61    public async list(at_least: Date): Promise<Stock[]> {
62        return await getConnection().getRepository(Stock).find({
63            where: { created_at: MoreThan(at_least) }
64        });
65    }
66
67    public async update(stockData: { id: number, name: string, description: string }): Promise<Stock> {
68        return await getConnection().getRepository(Stock).save({
69            ...stockData
70        });
71    }
72}
73
74 export default StockService
75
```

Ln 1, Col 1 Spaces: 4 UTF-8 LF ⓘ TypeScript ⓘ Go Live ⓘ Spell ⓘ Prettier

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under the 'LAB2' folder. The 'UserService.ts' file is selected.
- Code Editor:** Displays the code for the 'UserService' class. The code includes methods for getting a user by ID or email, creating a new user, and deleting a user.
- Bottom Status Bar:** Shows 'Ln 1, Col 1' and other settings like 'Spaces: 4', 'UTF-8', 'LF', 'TypeScript', 'Go Live', 'Spell', and 'Prettier'.

```
UserService.ts
src > services > UserService.ts > ...
1 import { User } from "../orm/models/User"
2 import { getConnection } from "typeorm";
3
4 class UserService {
5   public async getById(id: number): Promise<User> {
6     return await getConnection().getRepository(User).findOne({ where: { id: id } }).then((user) => {
7       if (!user) {
8         throw new Error(`User (id=${id}) does not exist`);
9       }
10      return user;
11    });
12  }
13
14  public async getByEmail(email: string): Promise<User> {
15    return await getConnection().getRepository(User).findOne({ where: { email: email } }).then((user) => {
16      if (!user) {
17        throw new Error(`User (email=${email}) does not exist`);
18      }
19      return user;
20    });
21  }
22
23  public async create(userData: { email: string, password: string }): Promise<User> {
24    const userRepo = getConnection().getRepository(User);
25
26    await userRepo.findOne({ where: { email: userData.email } }).then((user) => {
27      if (user !== null) {
28        throw new Error('User with such email already exists');
29      }
30    });
31
32    // todo hash pass
33    return userRepo.save(userData);
34  }
35
36  public async delete(id: number): Promise<void> {
37    const user = await this.getById(id);
38
39    const user = await this.getById(id);
40    await getConnection().getRepository(User).remove(user);
41  }
42
43  public async list(): Promise<User[]> {
44    return await getConnection().getRepository(User).find();
45  }
46
47  public async update(userData: { id: number, email: string, password: string }): Promise<User> {
48    const user = await this.getById(userData.id);
49
50    try {
51      await this.getByEmail(userData.email);
52      throw new Error('email is already taken');
53    } catch {
54      return await getConnection().getRepository(User).save({
55        id: user.id,
56        ...userData
57      });
58    }
59  }
60
61  export default UserService
62
```

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under the 'LAB2' folder. The 'UserService.ts' file is selected.
- Code Editor:** Displays the updated code for the 'UserService' class. The code includes methods for deleting, listing, updating, and exporting the service.
- Bottom Status Bar:** Shows 'Ln 1, Col 1' and other settings like 'Spaces: 4', 'UTF-8', 'LF', 'TypeScript', 'Go Live', 'Spell', and 'Prettier'.

```
UserService.ts
src > services > UserService.ts > ...
37
38  public async delete(id: number): Promise<void> {
39    const user = await this.getById(id);
40    await getConnection().getRepository(User).remove(user);
41  }
42
43  public async list(): Promise<User[]> {
44    return await getConnection().getRepository(User).find();
45  }
46
47  public async update(userData: { id: number, email: string, password: string }): Promise<User> {
48    const user = await this.getById(userData.id);
49
50    try {
51      await this.getByEmail(userData.email);
52      throw new Error('email is already taken');
53    } catch {
54      return await getConnection().getRepository(User).save({
55        id: user.id,
56        ...userData
57      });
58    }
59  }
60
61  export default UserService
62
```

6. Тестирование с помощью Postman

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections like Home, Workspaces, API Network, Reports, and Explore. Below these are collections for APIs, Environments, Mock Servers, Monitors, Flows, and History. A prominent 'My Workspace' section is open, showing a 'New Collection' named 'stock_delete'. This collection contains several requests: GET user_get, POST user_creation, DEL user_deleting, PATCH user_updating, GET user_list, POST auth_auth, POST auth_change_token, DEL auth_delete_token, POST stock_create, GET stock_get, GET stock_list, POST stock_updateprice, GET stock_history, and a selected 'DEL stock_delete'. The main workspace shows a 'DELETE' request to '127.0.0.1:3000/stocks/5'. The 'Params' tab is active, showing a single parameter 'Key' with 'Value'. Other tabs include Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings. A 'Send' button is at the top right. Below the request, there's a 'Response' section with a placeholder image of a rocket launching. At the bottom, there's a progress bar labeled 'Start working with APIs' and a note 'Next: Save a request. Show me'.

Вывод

В ходе данной лабораторной работы мы реализовали RESTful API приложение сайта криптобиржи.