

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа: restful api

**Выполнил:
Хайрнасов А.К.
К33402**

**Проверил:
Добряков Д. И.**

Санкт-Петербург

2022 г.

Задача

Необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate)

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

Модели

```
File: ./app/src/models/currency/Currency.ts

1 import {AllowNull, Column, Model, PrimaryKey, Table, Unique} from "sequelize-typescript";
2
3 @Table
4 class Currency extends Model {
5     @PrimaryKey
6     @AllowNull(false)
7     @Column
8     id: string;
9
10    @AllowNull(false)
11    @Unique
12    @Column
13    name: string;
14
15    @AllowNull(false)
16    @Column
17    price: number;
18
19    @AllowNull(false)
20    @Column
21    date: Date;
22
23
24    export default Currency;
```

```
4  @Table
5  class Portfolio extends Model {
6      @PrimaryKey
7      @AutoIncrement
8      @Column
9      id: number;
10
11     @ForeignKey(() => User)
12     @Column
13     userId: number;
14
15     @ForeignKey(() => Currency)
16     @Column
17     currencyId: string;
18
19     @Default(0)
20     @Column
21     amount: number;
22 }
23
24 export default Portfolio;
```

Сервис

```
5
6 class CurrencyService {
7     async loadAll() {
8         try {
9             const response = await axios.get('https://api.coingecko.com/api/v3/coins/list');
10            const data = response.data;
11            const currencies = data.map((currency: any) => ({
12                id: currency.id,
13                name: currency.name,
14                price: currency.current_price,
15                date: currency.last_updated
16            })
17        )
18        await Currency.bulkCreate(currencies)
19    } catch (e: any) {
20        const errors = e.errors.map((error: any) => error.message);
21
22        throw new CurrencyError(errors)
23    }
24
25
26
27     async getAll() {
28         const currencies = await Currency.findAll();
29
30         if (currencies) return currencies;
31
32         throw new CurrencyError('No currencies found')
33     }
34
35     async getById(id: string) {
36         const currency = await Currency.findByPk(id);
37     }
}
```

```
35     async getById(id: string) {
36         const currency = await Currency.findById(id);
37
38         if (currency) return currency;
39
40         throw new CurrencyError(`Currency with id = ${id} not found`)
41     }
42
43     async search(name: string) {
44         try {
45             const currencies = await Currency.findAll({
46                 where: {
47                     name: name,
48                 }
49             });
50             if (currencies) return currencies;
51         } catch (e: any) {
52             throw new Error(`Failed to find currency ${name}. Such a sad story :(`)
53         }
54     }
55
56     async filterByDate(name: string, startDate: Date, endDate: Date) {
57         try {
58             const currencies = await Currency.findAll({
59                 where: {
60                     date: {
61                         [Op.between]: [startDate, endDate]
62                     },
63                     name: name
64                 }
65             })
66             if(currencies) return currencies;
67         }catch (e: any){

```

```

4  class PortfolioService {
5      async getById(userId: string) {
6          try {
7              const portfolio = await Portfolio.findAll({
8                  where: {
9                      userId: userId,
10                 }
11             })
12             if (portfolio) return portfolio;
13         } catch (e: any) {
14             throw new Error(`Failed to load portfolio of user with id = ${userId} :/${}`);
15         }
16     }
17
18     async buyCurrency(userId: number, currencyId: string, amount: number) {
19         const t = await sequelize.transaction();
20
21         try {
22             let currency = await Portfolio.findOne({
23                 where: {
24                     userId: userId,
25                     currencyId: currencyId,
26                 },
27                 lock: true,
28                 transaction: t
29             });
30
31             if (currency) {
32                 currency.amount += amount;
33                 await currency.save();
34             } else {
35                 currency = await Portfolio.create({
36                     userId: userId,
37
38                     currencyId: currencyId,
39                     amount: amount
40                 }, { transaction: t })
41             return currency;
42         } catch (e: any) {
43             await t.rollback();
44             throw new Error('Failed to buy currency :(')
45         }
46     }
47
48     async sellCurrency(userId: number, currencyId: string, amount: number) {
49         const t = await sequelize.transaction();
50
51         try {
52             // for update lock in transaction
53             const currency = await Portfolio.findOne({ where: { userId: userId, currencyId: currencyId }, lock: true, transaction: t });
54
55             if (currency) {
56                 if (amount < currency.amount) {
57                     currency.amount -= amount;
58                     await currency.save();
59                     return currency;
60                 } else if (amount == currency.amount) {
61                     await currency.destroy();
62                     return null;
63                 } else {
64                     throw new Error("Can't sell because amount is > than you have. Do you want to sell more than you have? Crazy");
65                 }
66             } else {
67                 throw new Error('Currency not found. You can not sell currency that you do not have. Stop it');
68             }
69         }

```

Контроллеры

```
3  class ChartController {
4      private chartService: ChartService;
5
6      constructor() {
7          this.chartService = new ChartService();
8      }
9
10     getChartData = async (request: any, response: any) => {
11         try {
12             const {currencyId, days} = request.body;
13             const chart = await this.chartService.getChartData(currencyId, days);
14             return response.status(200).send(chart);
15         } catch (error: any) {
16             response.status(500).send({ "error": error.message })
17         }
18     }
19 }
20
21 export default ChartController;
```

```
3  class CurrencyController {
4      private currencyService: CurrencyService;
5
6      constructor() {
7          this.currencyService = new CurrencyService();
8      }
9
10     loadAll = async (request: any, response: any) => {
11         try {
12             await this.currencyService.loadAll();
13             response.status(200).send("OK!");
14         } catch (error: any) {
15             response.status(404).send({ "error": error.message })
16         }
17     }
18
19     getAll = async (request: any, response: any) => {
20         try {
21             const currencies = await this.currencyService.getAll();
22             response.send(currencies)
23         } catch (error: any) {
24             response.status(404).send({ "error": error.message })
25         }
26     }
27
28     getById = async (request: any, response: any) => {
29         try {
30             const currency = await this.currencyService.getById(request.params.id);
31             response.send(currency);
32         } catch (error: any) {
33             response.status(404).send({ "error": error.message })
34         }
35     }
}
```

```
36      getByName = async (request: any, response: any) => {
37        try {
38          const {name} = request.query
39          if (!name) {
40            return response.status(400).send('Currency name is missing');
41          }
42          const currencies = await this.currencyService.search(name as string)
43          response.send(currencies)
44        } catch (error: any) {
45          response.status(500).send({ "error": error.message })
46        }
47      }
48
49      filterByDate = async (request: any, response: any) => {
50        try {
51          const {name, startDate, endDate} = request.body;
52          const currencies = await this.currencyService.filterByDate(name, startDate, endDate);
53          response.send(currencies)
54        } catch (error: any) {
55          response.status(500).send({ "error": error.message })
56        }
57      }
58    }
59  }
60}
61
```

Вывод

В ходе работы реализован RESTful API для криптобиржи, включая функции входа, регистрации, отображения портфеля пользователя, графиков роста криптовалют и поиска по криптовалютам. API обеспечивает полный набор операций для взаимодействия с системой и был создан на основе ранее написанного boilerplate.