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

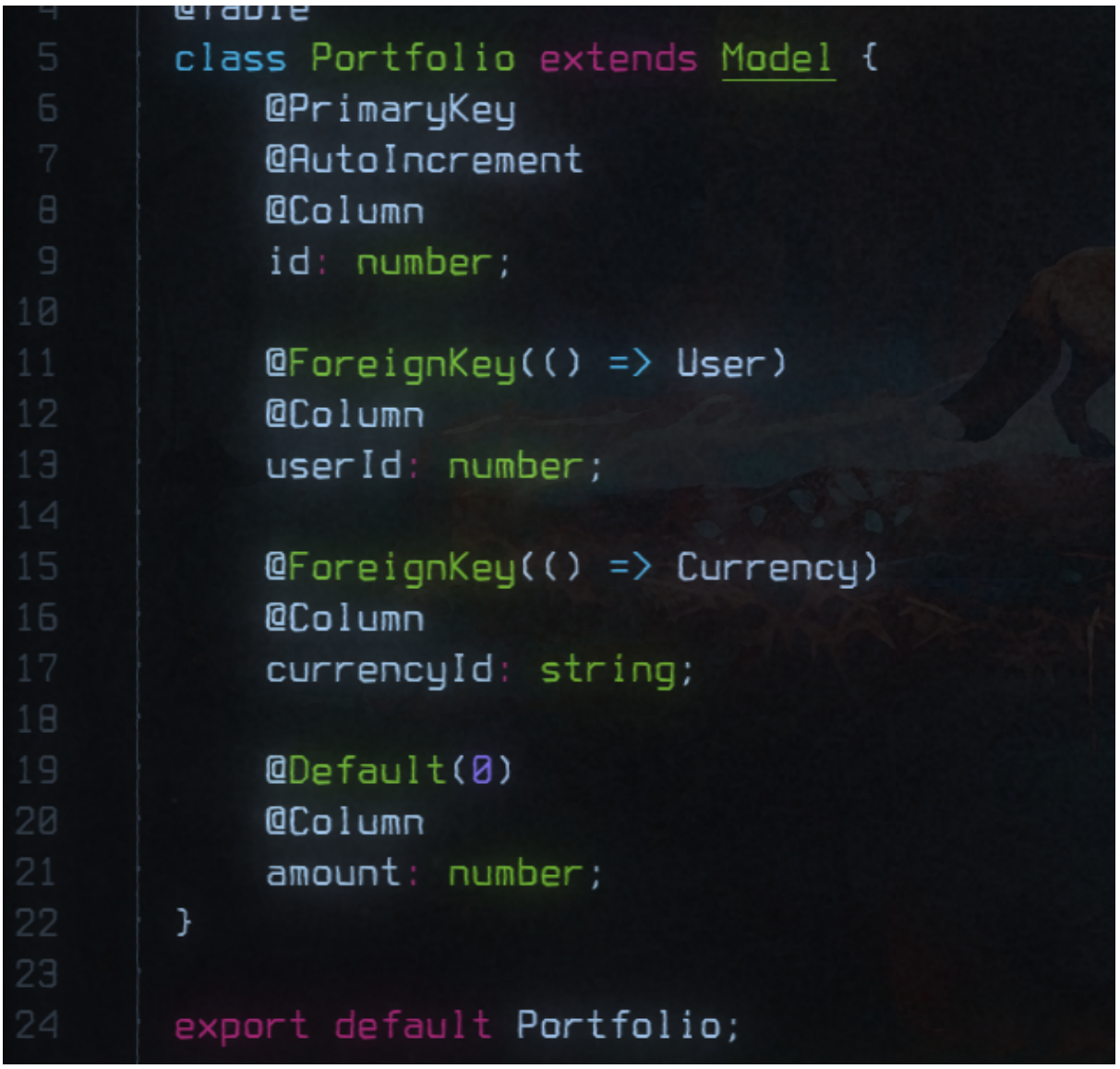
- Вход
- Регистрация
- Портфель пользователя с указанием различных криптовалют и их количеством
- Графики роста криптовалют
- Поиск по криптовалютам с возможностью фильтрации по дате добавления на биржу

Ход работы

Модели

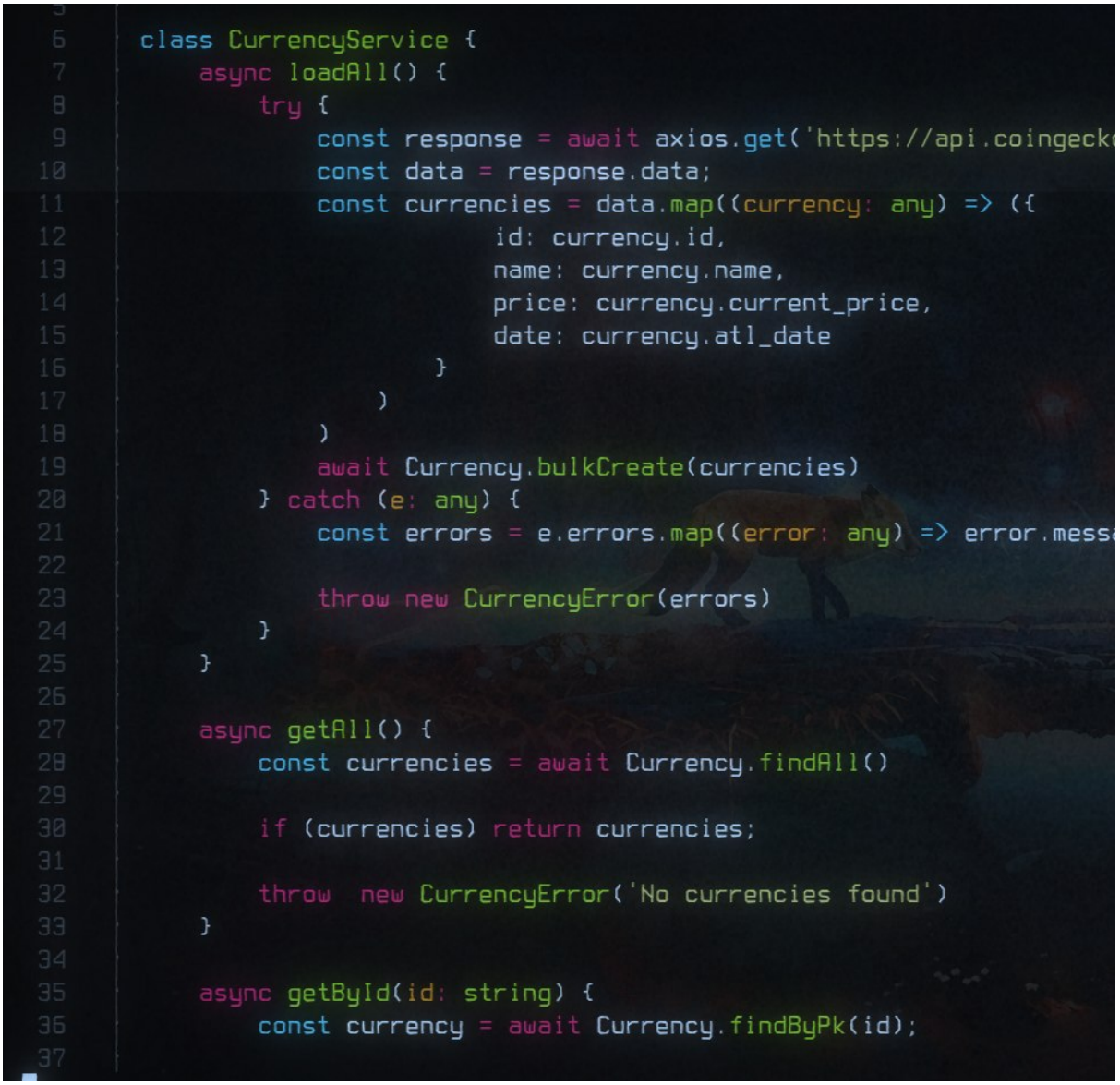
File: ./app/src/models/currency/Currency.ts

```ts
import {AllowNull, Column, Model, PrimaryKey, Table, Unique} from "sequelize-typescript";

@Table
class Currency extends Model {
    @PrimaryKey
    @AllowNull(false)
    @Column
    id: string;

    @AllowNull(false)
    @Unique
    @Column
    name: string;

    @AllowNull(false)
    @Column
    price: number;

    @AllowNull(false)
    @Column
    date: Date;
}

export default Currency;
```



```
@Table
class Portfolio extends Model {
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number;

    @ForeignKey(() => User)
    @Column
    userId: number;

    @ForeignKey(() => Currency)
    @Column
    currencyId: string;

    @Default(0)
    @Column
    amount: number;
}

export default Portfolio;
```

Сервис


class CurrencyService {
async loadAll() {
try {
const response = await axios.get('https://api.coingeck
const data = response.data;
const currencies = data.map((currency: any) => ({
id: currency.id,
name: currency.name,
price: currency.current_price,
date: currency.atl_date
}
)
)
await Currency.bulkCreate(currencies)
} catch (e: any) {
const errors = e.errors.map((error: any) => error.mess
throw new CurrencyError(errors)
}
}
async getAll() {
const currencies = await Currency.findAll()
if (currencies) return currencies;
throw new CurrencyError('No currencies found')
}
async getById(id: string) {
const currency = await Currency.findByPk(id);

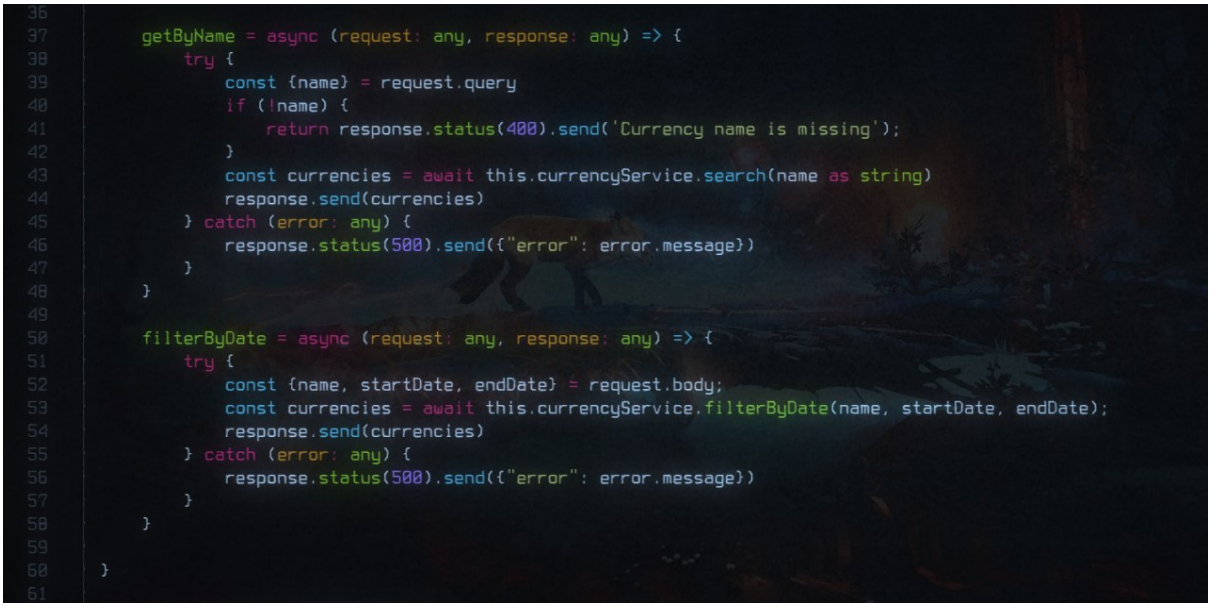
```
    async getById(id: string) {
        const currency = await Currency.findByPk(id);

        if (currency) return currency;

        throw new CurrencyError(`Currency with id = ${id} not found`)
    }

    async search(name: string) {
        try {
            const currencies = await Currency.findAll({
                where: {
                    name: name,
                }
            });
            if (currencies) return currencies;
        } catch (e: any) {
            throw new Error(`Failed to find currency ${name}. Such a sad story :(`)
        }
    }

    async filterByDate(name: string, startDate: Date, endDate: Date) {
        try {
            const currencies = await Currency.findAll({
                where: {
                    date: {
                        [Op.between]: [startDate, endDate]
                    },
                    name: name
                }
            })
            if(currencies) return currencies;
        }catch (e: any){
```

```
class PortfolioService {
    async getByUserId(userId: string) {
        try {
            const portfolio = await Portfolio.findAll({
                where: {
                    userId: userId,
                }
            })
            if (portfolio) return portfolio;
        } catch (e: any) {
            throw  new Error(`Failed to load portfolio of user with id = ${userId} :/`)
        }
    }

    async buyCurrency(userId: number, currencyId: string, amount: number) {
        const t = await sequelize.transaction();

        try {
            let currency = await Portfolio.findOne({
                where: {
                    userId: userId,
                    currencyId: currencyId,
                },
                lock: true,
                transaction: t
            });

            if (currency) {
                currency.amount += amount;
                await currency.save();
            } else {
                currency = await Portfolio.create({
                    userId: userId,
                    currencyId: currencyId,
                    amount: amount
                }, { transaction: t })
            }
            return currency;
        } catch (e: any) {
            await t.rollback()
            throw new Error('Failed to buy currency :(')
        }
    }

    async sellCurrency(userId: number, currencyId: string, amount: number) {
        const t = await sequelize.transaction();

        try {
            // for update lock in transaction
            const currency = await Portfolio.findOne({ where: { userId: userId, currencyId: currencyId }, lock: true, transaction: t });

            if (currency) {
                if (amount < currency.amount) {
                    currency.amount -= amount;
                    await currency.save();
                    return currency;
                } else if (amount == currency.amount) {
                    await currency.destroy();
                    return null;
                } else {
                    throw new Error("Can't sell because amount is > than you have. Do you want to sell more than you have? Crazy");
                }
            } else {
                throw new Error('Currency not found. You can not sell currency that you do not have. Stop it');
            }

```

Контроллеры

```ts
class ChartController {
    private chartService: ChartService;

    constructor() {
        this.chartService = new ChartService();
    }

    getChartData = async (request: any, response: any) => {
        try {
            const {currencyId, days} = request.body;
            const chart = await this.chartService.getChartData(currencyId, days);
            return response.status(200).send(chart);
        } catch (error: any) {
            response.status(500).send({"error": error.message})
        }
    }
}

export default ChartController;
```

```ts
class CurrencyController {
    private currencyService: CurrencyService;

    constructor() {
        this.currencyService = new CurrencyService();
    }

    loadAll = async (request: any, response: any) => {
        try {
            await this.currencyService.loadAll();
            response.status(200).send("OK!")
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const currencies = await this.currencyService.getAll();
            response.send(currencies)
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }

    getById = async (request: any, response: any) => {
        try {
            const currency = await this.currencyService.getById(request.params.id);
            response.send(currency);
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }
```



```
    getByName = async (request: any, response: any) => {
        try {
            const {name} = request.query
            if (!name) {
                return response.status(400).send('Currency name is missing');
            }
            const currencies = await this.currencyService.search(name as string)
            response.send(currencies)
        } catch (error: any) {
            response.status(500).send({"error": error.message})
        }
    }

    filterByDate = async (request: any, response: any) => {
        try {
            const {name, startDate, endDate} = request.body;
            const currencies = await this.currencyService.filterByDate(name, startDate, endDate);
            response.send(currencies)
        } catch (error: any) {
            response.status(500).send({"error": error.message})
        }
    }

}
```

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.