САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет
по лабораторной работе №2
«REST, RESTful, SOAP, GraphQL»

Выполнила:

Киреева М.С.

Группа
К3333

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Реализовать RESTful API средствами express + typescript

Сайт криптобиржи со следующим функционалом:

- 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. Модели

1) Users:
    a. FIO
    b. email
    c. password
    d. username
2) Currency
    a. id
    b. name
    c. price
    d. createdAt (создается автоматически)
3) Portfolio
    a. id
    b. userId
    c. currencyId
    d. sum

Currency

```
10+ usages  dikiydinozavrik *
@Table
class Currency extends Model {
    @Column( options: { primaryKey: true, autoIncrement: true })
    5+ usages  dikiydinozavrik
    id: number;

    @AllowNull( allowNull: false)
    @Column
    no usages  dikiydinozavrik
    name: string;

    @AllowNull( allowNull: false)
    @Column
    no usages  new *
    price: number;
}
5+ usages  dikiydinozavrik
export default Currency;
```

User

```
@Table
class User extends Model {
    @Column( options: { primaryKey: true, autoIncrement: true })
    5+ usages  new *
    id: number;

    @Unique
    @Column
    no usages  new *
    username: string

    @Column
    1 usage  new *
    firstName: string

    @Column
    1 usage  new *
    lastName: string

    @Unique
    @Column
    1 usage  new *
    email: string

    @AllowNull( allowNull: false)
    @Column
    5+ usages  new *
    password: string
```

Portfolio

```
@Table
class Portfolio extends Model {
    @Column( options: { primaryKey: true, autoIncrement: true })
    5+ usages  new *
    id: number;

    @ForeignKey( relatedClassGetter: () => User)
    @Column
    3 usages  new *
    userId: number

    @ForeignKey( relatedClassGetter: () => Currency)
    @Column
    no usages  new *
    currencyId: number

    @Default( value: 0)
    @Column
    3 usages  new *
    sum: number

}
```

2. Роуты

Currency

```
//Добавление валюты на биржу
router.route( prefix: '/add_currency')
    .post(controller.post)

//Удаление валюты
router.route( prefix: '/delete/:id')
    .delete(controller.deleteById)

//Все валюты на бирже
router.route( prefix: '/all')
    .get(controller.getAll)

//Фильрация по дате
router.route( prefix: '/date_filter')
    .get(controller.byDate)

//Информация о валюте
router.route( prefix: '/name_filter')
    .get(controller.ByName)
```

User

```
//Регистрация
router.route( prefix: '/create_account')
    .post(controller.post)

//Вход
router.route( prefix: '/login')
    .post(controller.auth)

//Поиск пользователя по токену
router.route( prefix: '/auth')
    .get(passport.authenticate( strategy: 'jwt',  options: { session: false }), controller.me)

//Инфомарция об аккаунте
router.route( prefix: '/account/:id')
    .get(controller.get)

//Обновление токена
router.route( prefix: '/refresh')
    .post(controller.refreshToken)

//Все профили
router.route( prefix: '/profiles')
    .get(controller.getAll)

//Удаление аккаунта
router.route( prefix: '/delete/:id')
    .delete(controller.deleteById)
```

Portfolio

```
//Покупка валюты
router.route( prefix: '/buy_currency')
    .post(controller.buyCurrency)

//Весь портфель пользователя
router.route( prefix: '/user_currencies')
    .get(controller.findByUser)

//Информация об одной валюте в портфеле пользователя
router.route( prefix: '/currency_info')
    .get(controller.oneByUser)

//Продажа валюты
router.route( prefix: '/sell')
    .post(controller.sell)

//Все портфели
router.route( prefix: '/all')
    .get(controller.getAll)
```

Plot

```
//График за год
router.route( prefix: '/year')
    .get(controller.oneYear)

//График за месяц
router.route( prefix: '/month')
    .get(controller.oneMonth)
```

3. Контроллеры

Currency

```
    get = async (request: any, response: any) => {
        try {
            const currency: Currency|CurrencyError = await this.currencyService.getById(
                Number(request.params.id)
            )

            response.send(currency)
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }

    2 usages  new *
    post = async (request: any, response: any) => {
        const { body } = request

        try {
            const currency : Currency|CurrencyError = await this.currencyService.create(body)

            response.status(201).send(currency)
        } catch (error: any) {
            response.status(400).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 })
        }
    }

    1 usage  new *
    deleteById = async (request: any, response: any)=> {
        try {
            const { id } = request.params;

            const deletedCount = await Currency.destroy( options: {
                where: {id: id}
            });

            if (deletedCount === 0) {
                throw new CurrencyError(`Currency with id ${id} not found`);
            }
            else {
                response.send(`Currency with id ${id} was deleted`)
            }

            response.status(204).send();
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
```

```
ByName = async (request: any, response: any) => {
    try {
        const { name } = request.query;
        if (!name) {
            return response.status(400).send('Currency name is required.');
        }

        const currencies = await this.currencyService.ByName(name as string);
        response.send(currencies);
    } catch (error) {
        response.status(500).send('An error occurred while fetching currencies by name.');
    }
};

1 usage  new *
ByDate = async (request: any, response: any) => {
    try {
        const { startDate,endDate } = request.query;
        if (!startDate) {
            return response.status(400).send('Currency name is required.');
        }
        const startDateObj = new Date(startDate as string);
        const endDateObj = new Date(endDate as string);
        const currencies = await this.currencyService.ByDate(startDateObj,endDateObj);
        response.send(currencies);
    } catch (error) {
        response.status(500).send('An error occurred while fetching currencies by name.');
    }
};
}
```

User

```
get = async (request: any, response: any) => {
    try {
        const user: User | UserError = await this.userService.getById(
            Number(request.params.id)
        )

        response.send(user)
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}

2 usages  new *
post = async (request: any, response: any) => {
    const { body } = request

    try {
        const user : User|UserError = await this.userService.create(body)

        response.status(201).send(user)
    } catch (error: any) {
        response.status(400).send({ "error": error.message })
    }
}
```

```
    me = async (request: any, response: any) => {
        response.send(request.user)
    }

    2 usages   new *
    auth = async (request: any, response: any) => {
        const { body } = request

        const { email, password } = body

        try {
            const { user, checkPassword } = await this.userService.checkPassword(email, password)

            if (checkPassword) {
                const payload = { id: user.id }

                console.log('payload is', payload)

                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)

                const refreshTokenService = new RefreshTokenService(user)

                const refreshToken = await refreshTokenService.generateRefreshToken()

                response.send({ accessToken, refreshToken })
            } else {
                throw new Error('Login or password is incorrect!')
            }
        } catch (e: any) {
            response.status(401).send({ "error": e.message })
        }
    }
```

```
    refreshToken = async (request: any, response: any) => {
        const { body } = request

        const { refreshToken } = body

        const refreshTokenService = new RefreshTokenService()

        try {
            const { userId, isExpired } = await refreshTokenService
                .isRefreshTokenExpired(refreshToken)

            if (!isExpired && userId) {
                const user = await this.userService.getById(userId)

                const payload = { id: user.id }

                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)

                const refreshTokenService = new RefreshTokenService(user)

                const refreshToken = await refreshTokenService.generateRefreshToken(

                response.send({ accessToken, refreshToken })
            } else {
                throw new Error('Invalid credentials')
            }
        } catch (e) {
            response.status(401).send({ 'error': 'Invalid credentials' })
        }
    }
```

```
getAll = async (request: any, response: any) => {
    try {
        const users = await this.userService.getAll()

        response.send(users)
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}

1 usage  new *
deleteById = async (request: any, response: any)=> {
    try {
        const { id } = request.params;

        const deletedCount = await User.destroy( options: {
            where: {id: id}
        });

        if (deletedCount === 0) {
            throw new UserError(`User with id ${id} not found`);
        }
        else {
            response.send(`User with id ${id} was deleted`)
        }

        response.status(204).send();
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}
2 usages  new *
```

Portfolio

```
buyCurrency = async (request: any, response: any) => {
    try {
        const { user_id, currency_id, amount } = request.query;
        await this.portfolioService.buyCurrency(user_id, currency_id, amount);
        response.status(200).json({ message: 'Currency added to portfolio successfully.' });
    } catch (error) {
        response.status(500).json({ error: 'Failed to add currency to portfolio.' });
    }
};

1 usage  new *
findByUser = async (request: any, response: any) => {
    try {
        const { user_id } = request.query;
        if (!user_id) {
            return response.status(400).send('User id is required.');
        }

        const currencies = await this.portfolioService.ByUser(user_id as string);
        response.send(currencies);
    } catch (error) {
        response.status(500).send('An error occurred while loading portfolio.');
    }
};
```

```
oneByUser= async (request: any, response: any) => {
    try {
        const { user_id, currency_id } = request.query;
        if (!user_id || !currency_id) {
            return response.status(400).send('Params are required.');
        }
        const currency = await this.portfolioService.oneByUser(user_id as string, currency_id as string);
        response.send(currency);
    } catch (error) {
        response.status(404).send('Currency not found.');
    }
}
```

```
2 usages  new *
getAll = async (request: any, response: any) => {
    try {
        const portfolio = await this.portfolioService.getAll()

        response.send(portfolio)
    } catch (error: any) {
        response.status(404).send({ "error": error.message })
    }
}
```

```
sell = async (request: any, response: any) => {
    try {
        const { user_id, currency_id, amount } = request.query;
        if (!user_id || !currency_id || !amount) {
            return response.status(400).send('Params are required.');
        }
        await this.portfolioService.sell(user_id, currency_id, amount);

        response.send('Currency has been sold successfully.');
    } catch (error) {
        response.status(500).send('An error occurred during selling.');
    }
};
```

Plot

```
    oneYear = async (request: any, response: any) => {
        try {
            const { currency_id } = request.query;

            if (!currency_id) {
                return response.status(400).send('Currency id is required.');
            }

            const endDate: Date = new Date();
            const startDate: Date = new Date();
            startDate.setFullYear( year: endDate.getFullYear() - 1);

            const currencies = await this.plotService.byDate(startDate,endDate,currency_id as number);
            response.send(currencies);
        } catch (error) {
            response.status(500).send('An error occurred while loading data.');
        }
    };

    oneMonth = async (request: any, response: any) => {
        try {
            const { currency_id } = request.query;

            if (!currency_id) {
                return response.status(400).send('Currency id is required.');
            }

            const endDate: Date = new Date();
            const startDate: Date = new Date();
            startDate.setMonth( month: endDate.getMonth() - 1);


            const currencies = await this.plotService.byDate(startDate,endDate,currency_id as number);
            response.send(currencies);
        } catch (error) {
            response.status(500).send('An error occurred while loading data.');
        }
    };
```

4. Сервисы

Currency

```
async getById(id: number) : Promise<Currency> {
    const currency = await Currency.findByPk(id)

    if (currency) return currency.toJSON()

    throw new CurrencyError('Not found!')
}

3 usages  new *
async create(currencyData: Partial<Currency>): Promise<Currency> {
    try {
        const currency = await Currency.create(currencyData)

        return currency.toJSON()
    } catch (e: any) {
        const errors = e.errors.map((error: any) => error.message)

        throw new CurrencyError(errors)
    }
}

2 usages  new *
async getAll() {
    const currencies = await Currency.findAll()

    if (currency) return currencies

    throw new CurrencyError('Currencies are not found')
}

ByName = async (name: string): Promise<Currency[]> => {
    try {
        const currencies = await Currency.findAll( options: {
            where: {
                name: name,
            },
        });
        return currencies;
    } catch (error) {
        throw new Error('Failed to fetch currencies by name.');
    }
};

1 usage  dikiydinozavrik *
async byDate(startDate: Date, endDate: Date): Promise<Currency[]> {
    try {
        const currencies = await Currency.findAll( options: {
            attributes: ['name', 'price', [Sequelize.fn( fn: 'MIN', Sequelize.col( col: 'createdAt')), 'firstCreatedAt']]
            group: ['name'],
            where: {
                createdAt: {
                    [Op.between]: [startDate, endDate],
                },
            },
            order: [[Sequelize.fn( fn: 'MIN', Sequelize.col( col: 'createdAt')), 'ASC']],
        });
        return currencies;
    } catch (error) {
        throw new Error('Failed to retrieve currencies.');
    }
}
```

Plot

```
byDate = async (startDate: Date, endDate: Date, currency_id: number): Promise<CurrencyPrice[]> => {
    try {
        const prices = await CurrencyPrice.findAll( options: {
            where: {
                currencyId:  currency_id,
                createdAt: {
                    [Op.between]: [startDate, endDate]
                },
            },
        });
        return prices;
    } catch (error) {
        throw new Error('Failed to load data.');
    }
};
```

Portfolio

```
async buyCurrency(userId: number, currencyId: number, amount: number) {
    try {
        let existingCurrency = await Portfolio.findOne( options: {
            where: {
                userId: userId,
                currencyId: currencyId,
            },
        });

        if (existingCurrency) {
            existingCurrency.sum += amount;
            await existingCurrency.save();
        } else {
            existingCurrency = await Portfolio.create( values: {
                userId: userId,
                currencyId: currencyId,
                sum: amount,
            });
        }

        return existingCurrency.toJSON();
    } catch (error) {
        throw new Error('Failed to buy currency.');
    }
}
```

```
async getAll() {
    const portfolio = await Portfolio.findAll()

    if (portfolio) return portfolio

    throw new PortfolioError('Currencies are not found')
}

1 usage  new *
ByUser = async (userId: string): Promise<Portfolio[]> => {
    try {
        const currencies = await Portfolio.findAll( options: {
            where: {
                userId: userId,
            },
        });
        return currencies;
    } catch (error) {
        throw new Error('Failed to open the portfolio.');
    }
};

oneByUser = async (userId: string, currencyId: string): Promise<Portfolio | null> => {
    try {
        const currency = await Portfolio.findOne( options: {
            where: {
                userId: userId,
                currencyId: currencyId,
            },
        });

        if (!currency) {
            throw new Error('Currency not found.');
        }

        return currency;
    } catch (error) {
        throw new Error('Failed to find currency.');
    }
};
```

```
sell = async (userId: string, currencyId: string, amount: number): Promise<void> => {
    try {
        const currency = await Portfolio.findOne( options: {
            where: {
                userId: userId,
                currencyId: currencyId,
            },
        });

        if (currency) {
            const newSum = currency.sum - amount;

            if (newSum > 0) {
                currency.sum = newSum;
                await currency.save();
            } else {
                await currency.destroy();
            }
        } else {
            throw new Error('Currency not found.');
        }
    } catch (error) {
        throw new Error('Failed to sell currency.');
    }
};
```

User

```
async getById(id: number) : Promise<User> {
    const user = await User.findByPk(id)

    if (user) return user.toJSON()

    throw new UserError('Not found!')
}

4 usages  new *
async create(userData: Partial<User>): Promise<User> {
    try {
        const user = await User.create(userData)

        return user.toJSON()
    } catch (e: any) {
        const errors = e.errors.map((error: any) => error.message)

        throw new UserError(errors)
    }
}
```

```
    async getAll() {
        const users = await User.findAll()

        if (users) return users

        throw new UserError('Users are not found')
    }

    2 usages  new *
    async checkPassword(email: string, password: string) : Promise<any> {
        const user = await User.findOne( options: { where: { email } })

        if (user) return { user: user.toJSON(), checkPassword: checkPassword(user, password) }

        throw new UserError('Incorrect login/password!')
    }
}
```

Вывод

В ходе данной работы было реализовано RESful API приложение криптобиржи с использованием express + typescript.