САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 2: RESTful API

Выполнил:

Конев Антон

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 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

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

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

    @AllowNull( allowNull: false)
    @Column
    price: number;

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

5+ usages  Ant
export default Currency;
```

```ts
@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull( allowNull: false)
    @Column
    token: string;

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

5+ usages  Ant
export default RefreshToken;
```

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

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

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

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

5+ usages  Ant
export default Portfolio;
```

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

    @Unique
    @Column
    username: string;

    @AllowNull( allowNull: false)
    @Unique
    @Column
    email: string;

    @AllowNull( allowNull: false)
    @Column
    password: string;

    no usages  Ant
    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) : void  {
        const {password : string } = instance;

        if (instance.changed( key: 'password')) {
            instance.password = hashPassword(password);
        }
    }
}
```

Эндпоинты

User

```
router.route( prefix: '/create')
    .post(controller.post);

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

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

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

router.route( prefix: '/:id')
    .get(controller.get);

router.route( prefix: '/')
    .get(controller.getAllUsers);
```

Portfolio

```
router.route( prefix: '/:id')
    .get(controller.findByUserId)

router.route( prefix: '/buy')
    .post(controller.buyCurrency)

router.route( prefix: '/sell')
    .post(controller.sellCurrency)
```

Chart

```
router.route( prefix: '/')
    .get(controller.getChartData)
```

Currency

```
router.route( prefix: '/load')
    .get(controller.loadAll)

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

router.route( prefix: '/:id')
    .get(controller.getById)

router.route( prefix: '/name')
    .get(controller.getByName)

router.route( prefix: '/date_filter')
    .get(controller.filterByDate)
```

Контроллеры

User

```
1 usage  Ant
get = async (request: any, response: any) : Promise<void>  => {
    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});
    }
};

1 usage  Ant
post = async (request: any, response: any) : Promise<void>  => {
    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) : Promise<void>  => {
    response.send(request.user);
};

1 usage  Ant
auth = async (request: any, response: any) : Promise<void>  => {
    const {body} = request;
    const {email, password} = body;

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

        if (checkPassword) {
            const payload : {id: any}  = {id: user.id};
            console.log('payload is', payload);
            const accessToken : string  = jwt.sign(payload, jwtOptions.secretOrKey);
            const refreshTokenService : RefreshTokenService  = new RefreshTokenService(user);
            const refreshToken : string  = 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});
    }
};

1 usage  Ant
refreshToken = async (request: any, response: any) : Promise<void>  => {
    const {body} = request;
    const {refreshToken} = body;
    const refreshTokenService : RefreshTokenService  = new RefreshTokenService();

    try {
        const {userId : number | null , isExpired : boolean } = await refreshTokenService
            .isRefreshTokenExpired(refreshToken);

        if (!isExpired && userId) {
            const user : User  = await this.userService.getById(userId);
            const payload : {id: number}  = {id: user.id};
            const accessToken : string  = jwt.sign(payload, jwtOptions.secretOrKey);
            const refreshTokenService : RefreshTokenService  = new RefreshTokenService(user);
            const refreshToken : string  = await refreshTokenService.generateRefreshToken();
            response.send({accessToken, refreshToken});
        } else {
            throw new Error('Invalid credentials');
        }
    } catch (e) {
        response.status(401).send({'error': 'Invalid credentials'});
    }
};
```

```
1 usage  Ant
getAllUsers = async (request: any, response: any) : Promise<void>  => {
    try {
        const users : User[]  = await this.userService.getAllUsers();

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

Portfolio

```
findByUserId = async (request: any, response: any) : Promise<any>  => {
    try {
        const portfolio : Portfolio[] | undefined  = await this.portfolioService.getByUserId(request.params.id);
        return response.status(200).send(portfolio);
    } catch (error: any) {
        response.status(500).send({"error": error.message})
    }
}

1 usage  Ant
buyCurrency = async (request: any, response: any) : Promise<any>  => {
    try {
        const {userId, currencyId, amount} = request.body;
        const currency : Portfolio  = await this.portfolioService.buyCurrency(userId, currencyId, amount);
        return response.status(200).send(currency);
    } catch (error: any) {
        response.status(500).send({"error": error.message})
    }
}

1 usage  Ant
sellCurrency = async (request: any, response: any) : Promise<any>  => {
    try {
        const {userId, currencyId, amount} = request.body;
        const currency : Portfolio | null  = await this.portfolioService.sellCurrency(userId, currencyId, amount);
        return response.status(200).send(currency);
    } catch (error: any) {
        response.status(500).send({"error": error.message})
    }
}
```

Currency

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

1 usage  Ant
getAll = async (request: any, response: any) : Promise<void>  => {
    try {
        const currencies : Currency[]  = await this.currencyService.getAll();
        response.send(currencies)
    } catch (error: any) {
        response.status(404).send({"error": error.message})
    }
}

1 usage  Ant
getById = async (request: any, response: any) : Promise<void>  => {
    try {
        const currency : Currency  = await this.currencyService.getById(request.params.id);
        response.send(currency);
    } catch (error: any) {
        response.status(404).send({"error": error.message})
    }
}

1 usage  Ant *
getByName = async (request: any, response: any) : Promise<any>  => {
    try {
        const {name} = request.body
        if (!name) {
            return response.status(400).send('Currency name is missing');
        }
        const currencies : Currency[] | undefined  = await this.currencyService.search(name)
        response.send(currencies)
    } catch (error: any) {
        response.status(500).send({"error": error.message})
    }
}

1 usage  Ant
filterByDate = async (request: any, response: any) : Promise<void>  => {
    try {
        const {name, startDate, endDate} = request.body;
        const currencies : Currency[] | undefined  = await this.currencyService.filterByDate(name, startDate, endDate);
        response.send(currencies)
    } catch (error: any) {
        response.status(500).send({"error": error.message})
    }
}
```

Chart

```
1 usage  Ant
getChartData = async (request: any, response: any) : Promise<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})
    }
}
```

Сервисы

Auth

```
generateRefreshToken = async (): Promise<string> => {
    const token : `${string}-${string}-${string}...  = randomUUID();
    const userId : number | undefined  = this.user?.id;
    await RefreshToken.create( values: {token, userId});

    return token;
};

1 usage  Ant
isRefreshTokenExpired = async (token: string): Promise<{ userId: number | null, isExpired: boolean }> => {
    const refreshToken : RefreshToken | null  = await RefreshToken.findOne( options: {where: {token}});

    if (refreshToken) {
        const tokenData = refreshToken.toJSON();
        const currentDate : Date  = new Date();
        const timeDelta : number  = currentDate.getTime() - tokenData.createdAt.getTime();

        if (timeDelta > 0 && timeDelta < parseInt(process.env.REFRESH_TOKEN_LIFETIME!)) {
            return {userId: tokenData.userId, isExpired: false};
        }

        return {userId: null, isExpired: true};
    }

    return {userId: null, isExpired: true};
}
```

Chart

```
async getChartData(id: string, days: number) : Promise<any>  {
    try {
        const response : AxiosResponse<any, any>  = await axios.get( url: `https://api.coingecko.com/api/v3/coins/${id}/market_chart?vs_currency=rub&days=${days-1}&interval=daily`);
        const {prices} = response.data;
        if (prices) return prices;
    } catch (e: any) {
        const errors = e.errors.map((error: any) => error.message)
        throw new ChartError(errors)
    }
}
```

User

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

    if (user) return user.toJSON();

    throw new UserError(`User with id = ${id} not found :(`);
}

1 usage  Ant
async create(userData: Partial<User>): Promise<User> {
    try {
        const user : User  = await User.create(userData);

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

        throw new UserError(errors);
    }
}

1 usage  Ant
async checkPassword(email: string, password: string): Promise<any> {
    const user : User | null  = await User.findOne( options: {where: {email}});

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

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

1 usage  Ant
async getAllUsers() : Promise<User[]>  {
    const users : User[]  = await User.findAll();

    if (users) return users;

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

Currency

```
async loadAll() : Promise<void>  {
    try {
        const response : AxiosResponse<any, any>  = await axios.get( url: 'https://api.coingecko.com/api/v3/coins/markets?vs_currency=rub&per_page=20&page=1');
        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.message)

        throw new CurrencyError(errors)
    }
}

1 usage   Ant
async getAll() : Promise<Currency[]>  {
    const currencies : Currency[]  = await Currency.findAll()

    if (currencies) return currencies;

    throw  new CurrencyError('No currencies found')
}

1 usage   Ant
async getById(id: string) : Promise<Currency>  {
    const currency : Currency | null  = await Currency.findByPk(id);

    if (currency) return currency;

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

```
async search(name: string) : Promise<Currency[] | undefined...  {
    try {
        const currencies : Currency[]  = await Currency.findAll( options: {
            where: {
                name: name,
            }
        });
        if (currencies) return currencies;
    } catch (e: any) {
        throw new Error(`Failed to find currency ${name}`)
    }
}


1 usage   Ant
async filterByDate(name: string, startDate: Date, endDate: Date) : Promise<Currency[] | undefined...  {
    try {
        const currencies : Currency[]  = await Currency.findAll( options: {
            where: {
                date: {
                    [Op.between]: [startDate, endDate]
                },
                name: name
            }
        })
        if(currencies) return currencies;
    }catch (e: any){
        throw new Error(`Failed to find currencies between dates`)
    }
}
```

Portfolio

```
async getByUserId(userId: string) : Promise<Portfolio[] | undefine...  {
    try {
        const portfolio : Portfolio[]  = await Portfolio.findAll( options: {
            where: {
                userId: userId,
            }
        })
        if (portfolio) return portfolio;
    } catch (e: any) {
        throw  new Error(`Failed to load portfolio of user with id = ${userId}`)
    }
}

1 usage   Ant
async buyCurrency(userId: number, currencyId: string, amount: number) : Promise<Portfolio>  {
    try {
        let currency : Portfolio | null  = await Portfolio.findOne( options: {
            where: {
                userId: userId,
                currencyId: currencyId,
            }
        });

        if (currency) {
            currency.amount += amount;
            await currency.save();
        } else {
            currency = await Portfolio.create( values: {
                userId: userId,
                currencyId: currencyId,
                amount: amount
            })
        }
        return currency;
    } catch (e: any) {
        throw new Error('Failed to buy currency')
    }
}
```

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

        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")
            }
        } else {
            throw new Error('Currency not found')
        }


    } catch (e: any) {
        throw new Error('Failed to sell currency')
    }
}
```

Вывод

В ходе лабораторной работы был реализован RESTful API по варианту сайта криптобиржи средствами sequelize+typescript и Express.js.