САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Домашнее задание №4

Выполнил:
Горбатов Дмитрий Алексеевич
гр. К33402

Преподаватель:
Добряков Давид Ильич

Санкт-Петербург
2024 г.

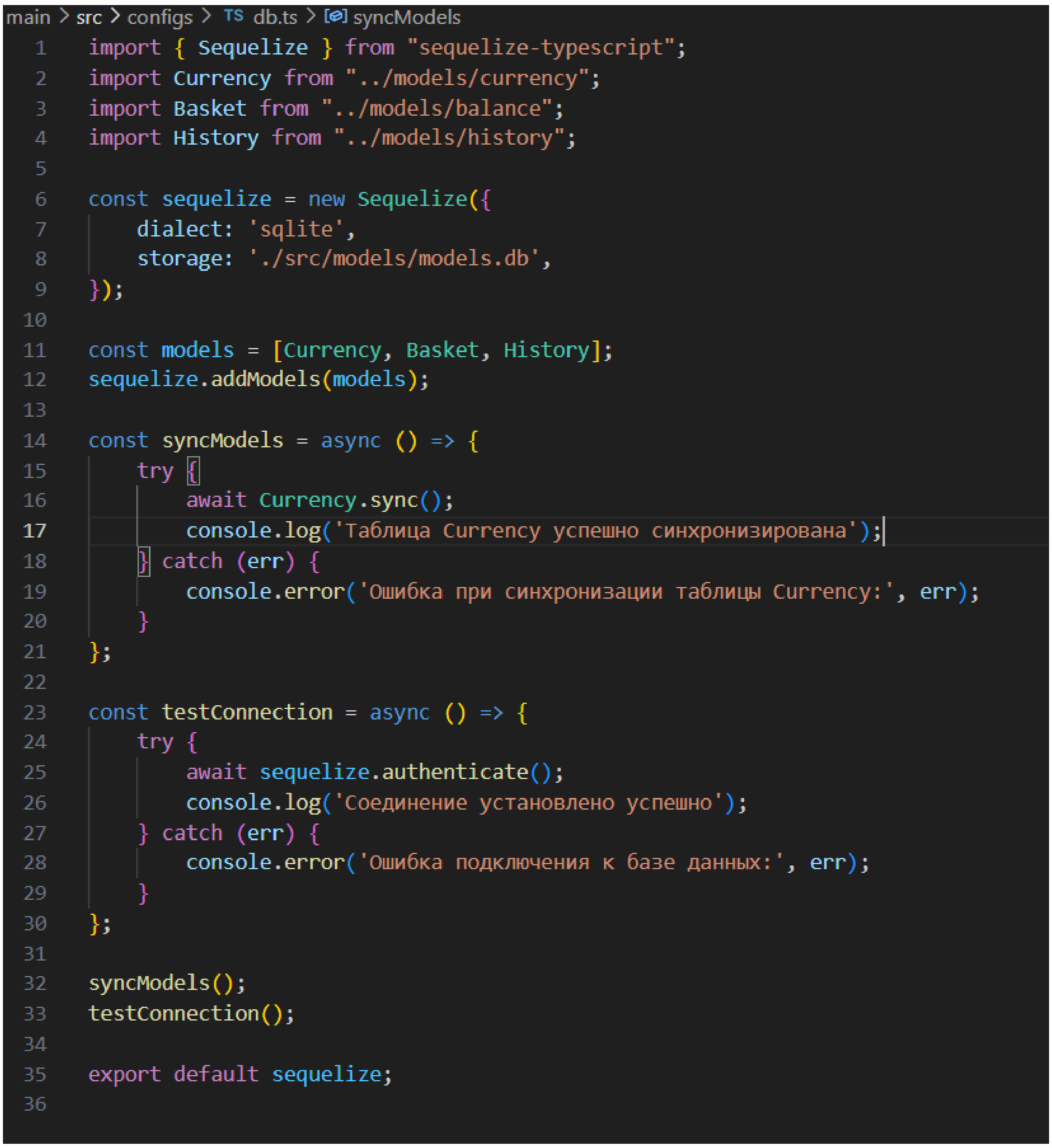
Задание

Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения. Мой вариант бэкенд для сайта криптобиржи.

Ход работы

Мой проект содержит такие папки как: configs, controllers, models, routes, service.

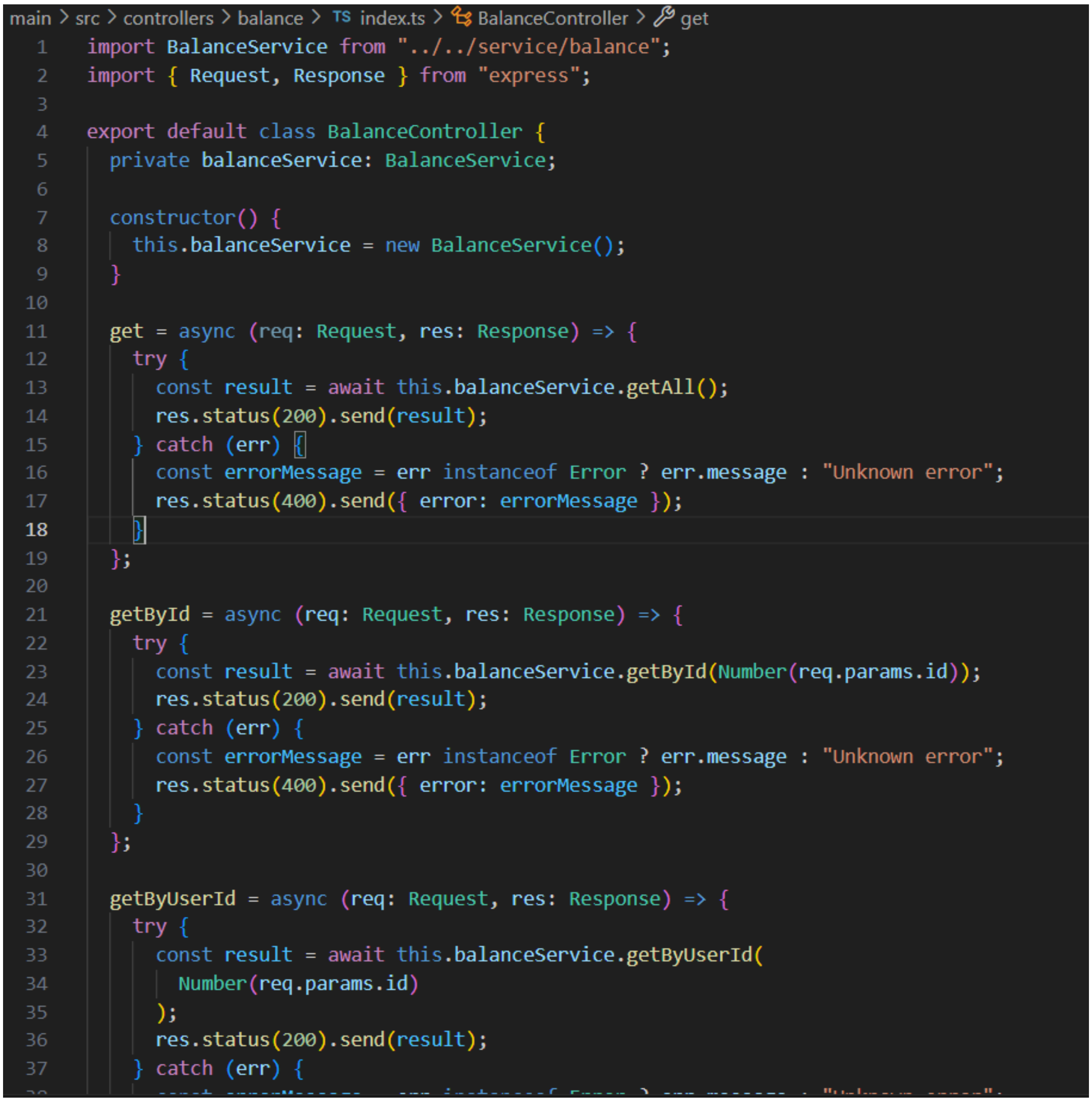
В папке configs располагается база данных db.ts:



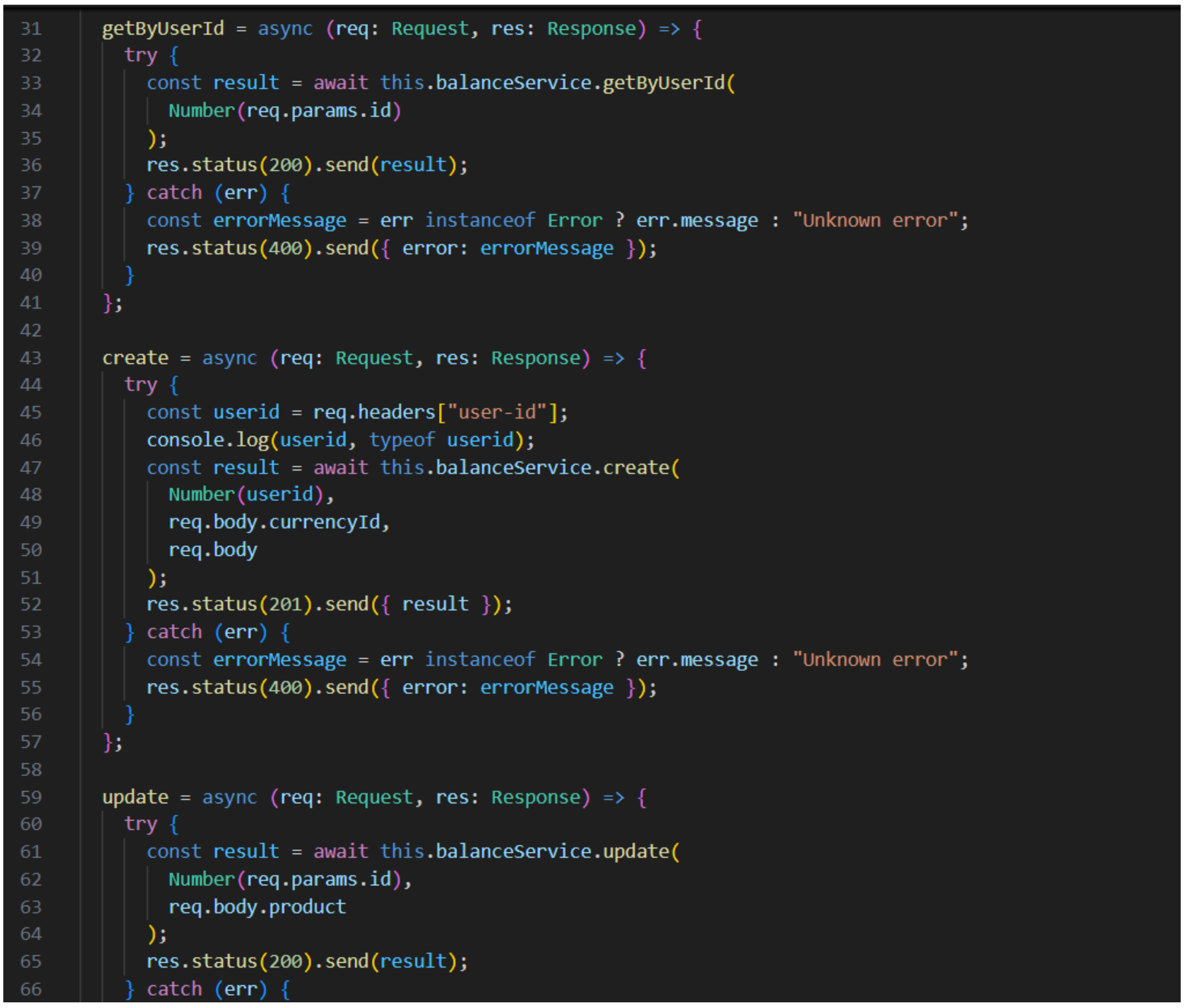
```typescript
import { Sequelize } from "sequelize-typescript";
import Currency from "../models/currency";
import Basket from "../models/balance";
import History from "../models/history";

const sequelize = new Sequelize({
    dialect: 'sqlite',
    storage: './src/models/models.db',
});

const models = [Currency, Basket, History];
sequelize.addModels(models);

const syncModels = async () => {
    try {
        await Currency.sync();
        console.log('Таблица Currency успешно синхронизирована');
    } catch (err) {
        console.error('Ошибка при синхронизации таблицы Currency:', err);
    }
};

const testConnection = async () => {
    try {
        await sequelize.authenticate();
        console.log('Соединение установлено успешно');
    } catch (err) {
        console.error('Ошибка подключения к базе данных:', err);
    }
};

syncModels();
testConnection();

export default sequelize;
```

В следующей папке controllers располагается еще 3 папки: balance, currency, history – три сущности.

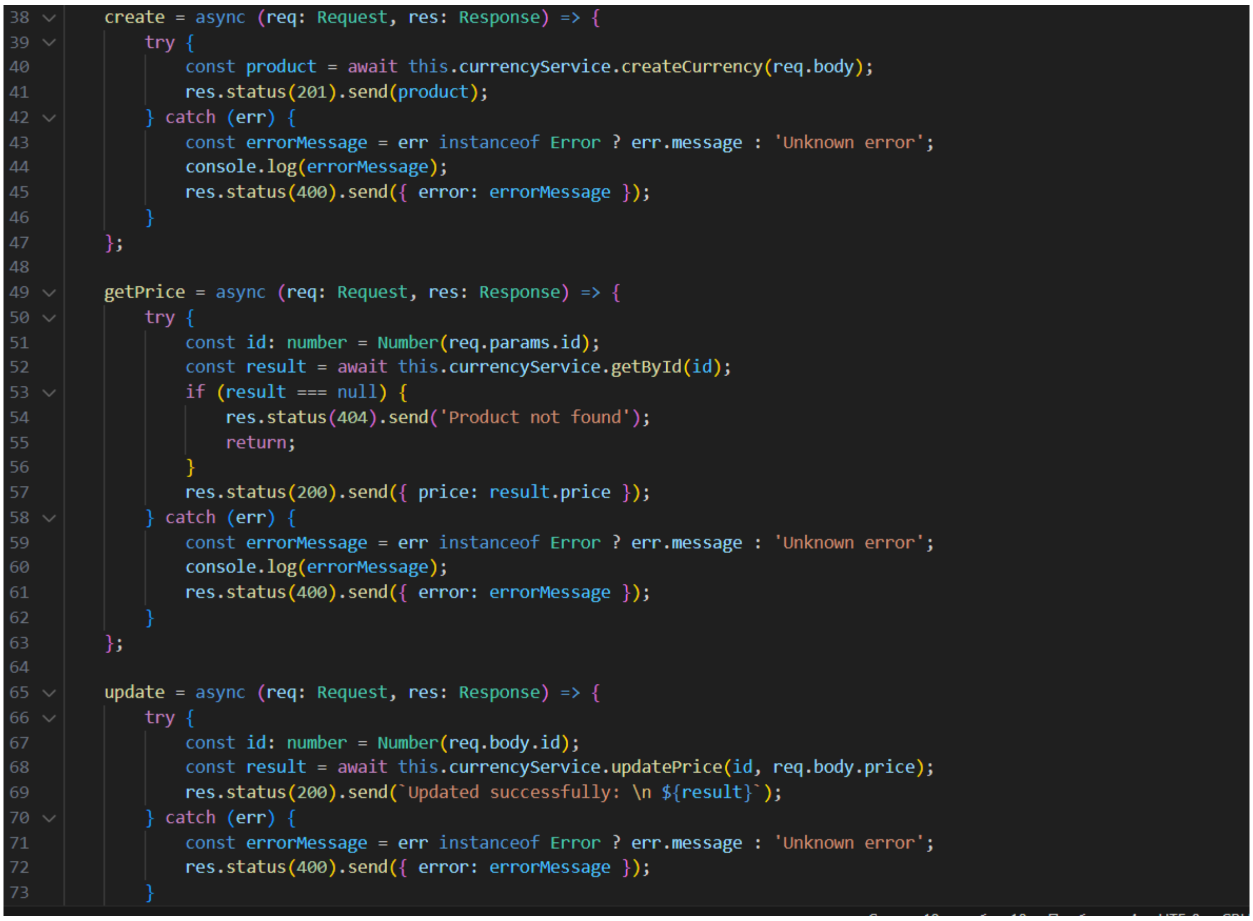
Balance:


main > src > controllers > balance > TS index.ts > BalanceController > get

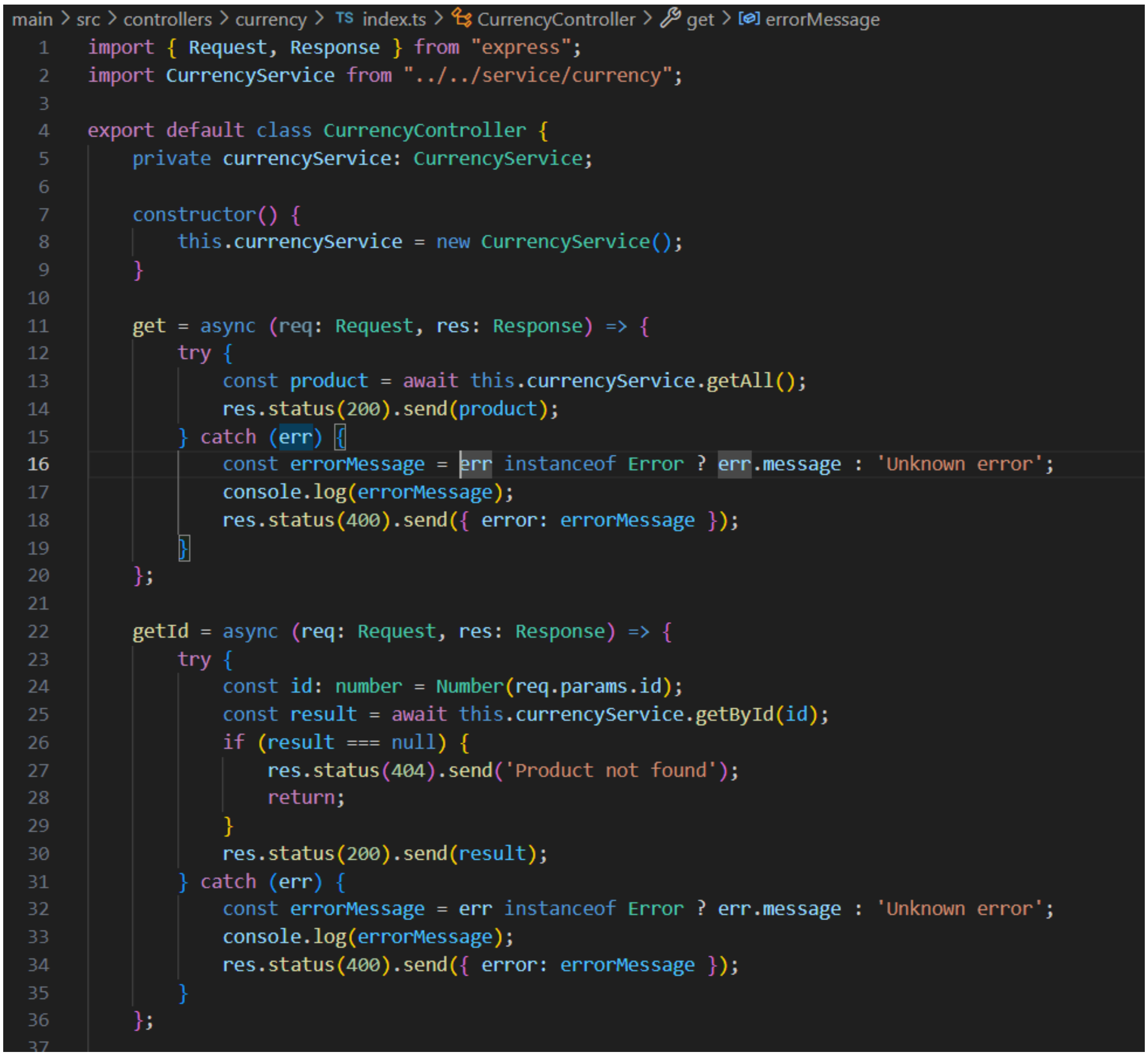
```typescript
import BalanceService from "../../service/balance";
import { Request, Response } from "express";

export default class BalanceController {
  private balanceService: BalanceService;

  constructor() {
    this.balanceService = new BalanceService();
  }

  get = async (req: Request, res: Response) => {
    try {
      const result = await this.balanceService.getAll();
      res.status(200).send(result);
    } catch (err) {
      const errorMessage = err instanceof Error ? err.message : "Unknown error";
      res.status(400).send({ error: errorMessage });
    }
  };

  getById = async (req: Request, res: Response) => {
    try {
      const result = await this.balanceService.getById(Number(req.params.id));
      res.status(200).send(result);
    } catch (err) {
      const errorMessage = err instanceof Error ? err.message : "Unknown error";
      res.status(400).send({ error: errorMessage });
    }
  };

  getByUserId = async (req: Request, res: Response) => {
    try {
      const result = await this.balanceService.getByUserId(
        Number(req.params.id)
      );
      res.status(200).send(result);
    } catch (err) {
```


```
getByUserId = async (req: Request, res: Response) => {
  try {
    const result = await this.balanceService.getByUserId(
      Number(req.params.id)
    );
    res.status(200).send(result);
  } catch (err) {
    const errorMessage = err instanceof Error ? err.message : "Unknown error";
    res.status(400).send({ error: errorMessage });
  }
};

create = async (req: Request, res: Response) => {
  try {
    const userid = req.headers["user-id"];
    console.log(userid, typeof userid);
    const result = await this.balanceService.create(
      Number(userid),
      req.body.currencyId,
      req.body
    );
    res.status(201).send({ result });
  } catch (err) {
    const errorMessage = err instanceof Error ? err.message : "Unknown error";
    res.status(400).send({ error: errorMessage });
  }
};

update = async (req: Request, res: Response) => {
  try {
    const result = await this.balanceService.update(
      Number(req.params.id),
      req.body.product
    );
    res.status(200).send(result);
  } catch (err) {
```

Currency:

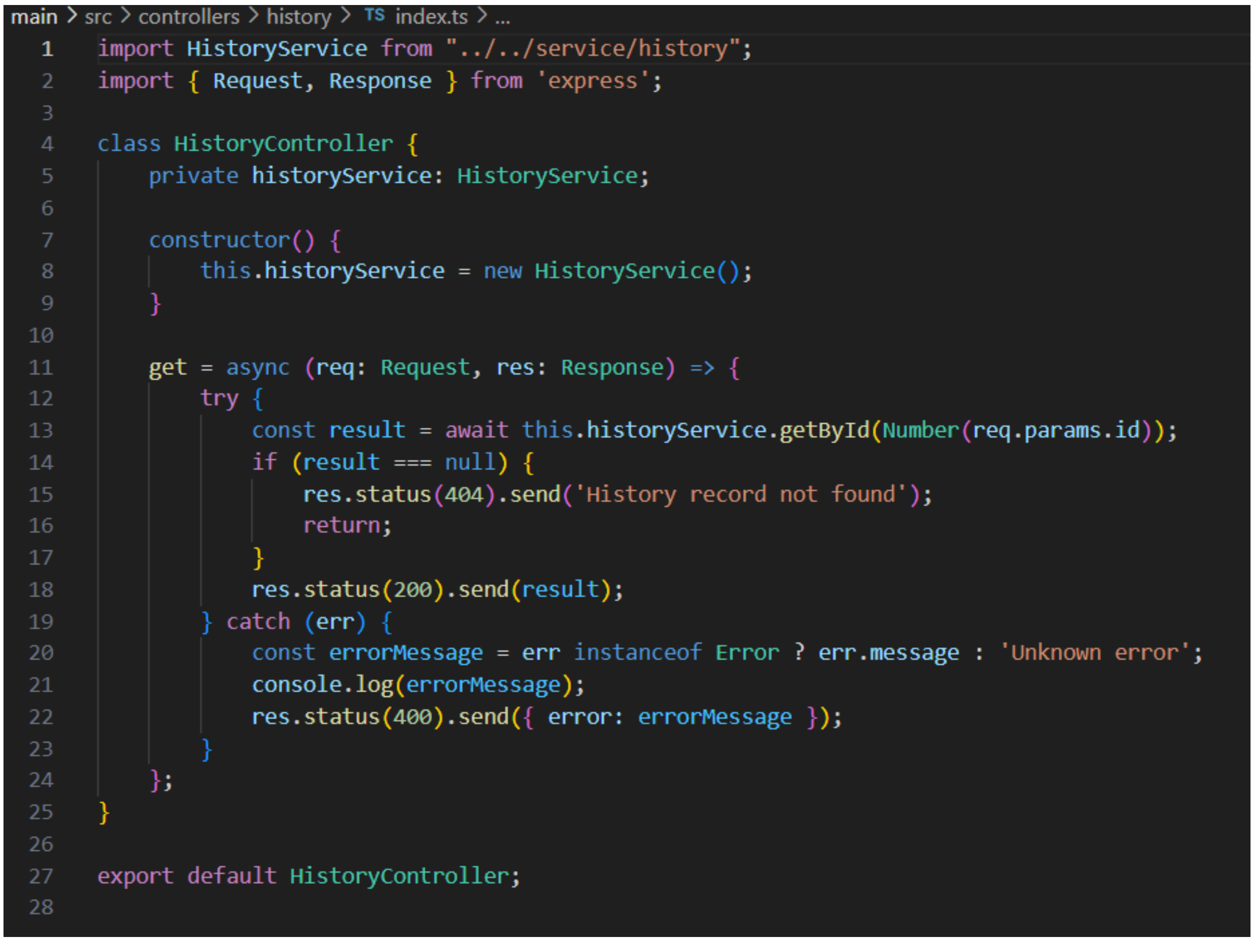


```typescript
    create = async (req: Request, res: Response) => {
        try {
            const product = await this.currencyService.createCurrency(req.body);
            res.status(201).send(product);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'Unknown error';
            console.log(errorMessage);
            res.status(400).send({ error: errorMessage });
        }
    };

    getPrice = async (req: Request, res: Response) => {
        try {
            const id: number = Number(req.params.id);
            const result = await this.currencyService.getById(id);
            if (result === null) {
                res.status(404).send('Product not found');
                return;
            }
            res.status(200).send({ price: result.price });
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'Unknown error';
            console.log(errorMessage);
            res.status(400).send({ error: errorMessage });
        }
    };

    update = async (req: Request, res: Response) => {
        try {
            const id: number = Number(req.body.id);
            const result = await this.currencyService.updatePrice(id, req.body.price);
            res.status(200).send(`Updated successfully: \n ${result}`);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'Unknown error';
            res.status(400).send({ error: errorMessage });
        }
```


main > src > controllers > currency > TS index.ts > CurrencyController > get > errorMessage

```typescript
import { Request, Response } from "express";
import CurrencyService from "../../service/currency";

export default class CurrencyController {
    private currencyService: CurrencyService;

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

    get = async (req: Request, res: Response) => {
        try {
            const product = await this.currencyService.getAll();
            res.status(200).send(product);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'Unknown error';
            console.log(errorMessage);
            res.status(400).send({ error: errorMessage });
        }
    };

    getId = async (req: Request, res: Response) => {
        try {
            const id: number = Number(req.params.id);
            const result = await this.currencyService.getById(id);
            if (result === null) {
                res.status(404).send('Product not found');
                return;
            }
            res.status(200).send(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'Unknown error';
            console.log(errorMessage);
            res.status(400).send({ error: errorMessage });
        }
    };
```

History:

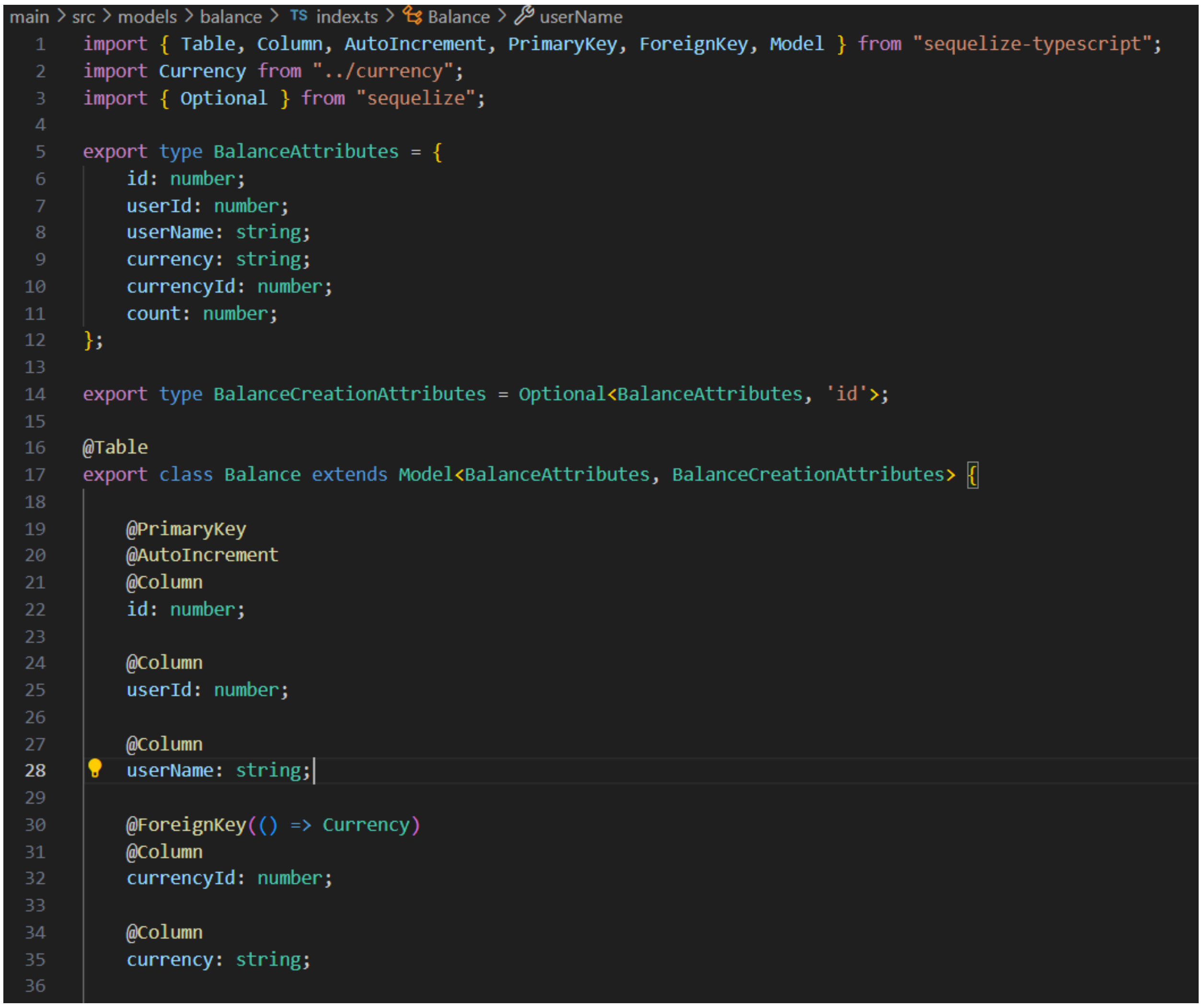


main > src > controllers > history > TS index.ts > ...

```
import HistoryService from "../../service/history";
import { Request, Response } from 'express';

class HistoryController {
    private historyService: HistoryService;

    constructor() {
        this.historyService = new HistoryService();
    }

    get = async (req: Request, res: Response) => {
        try {
            const result = await this.historyService.getById(Number(req.params.id));
            if (result === null) {
                res.status(404).send('History record not found');
                return;
            }
            res.status(200).send(result);
        } catch (err) {
            const errorMessage = err instanceof Error ? err.message : 'Unknown error';
            console.log(errorMessage);
            res.status(400).send({ error: errorMessage });
        }
    };
}

export default HistoryController;
```

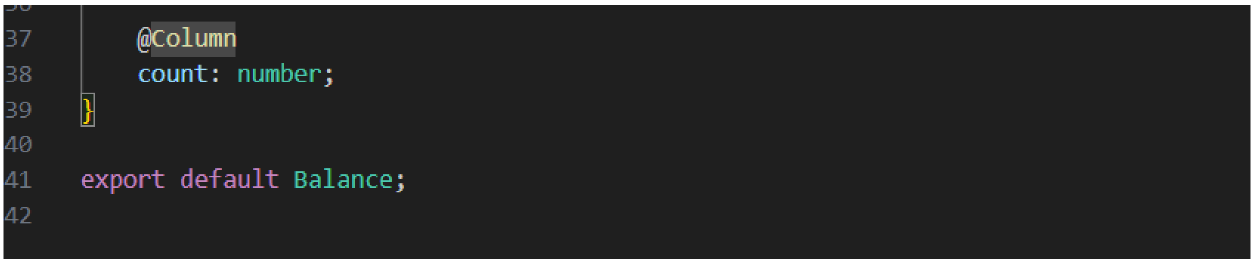
Следующая папка – модели.

Balance:



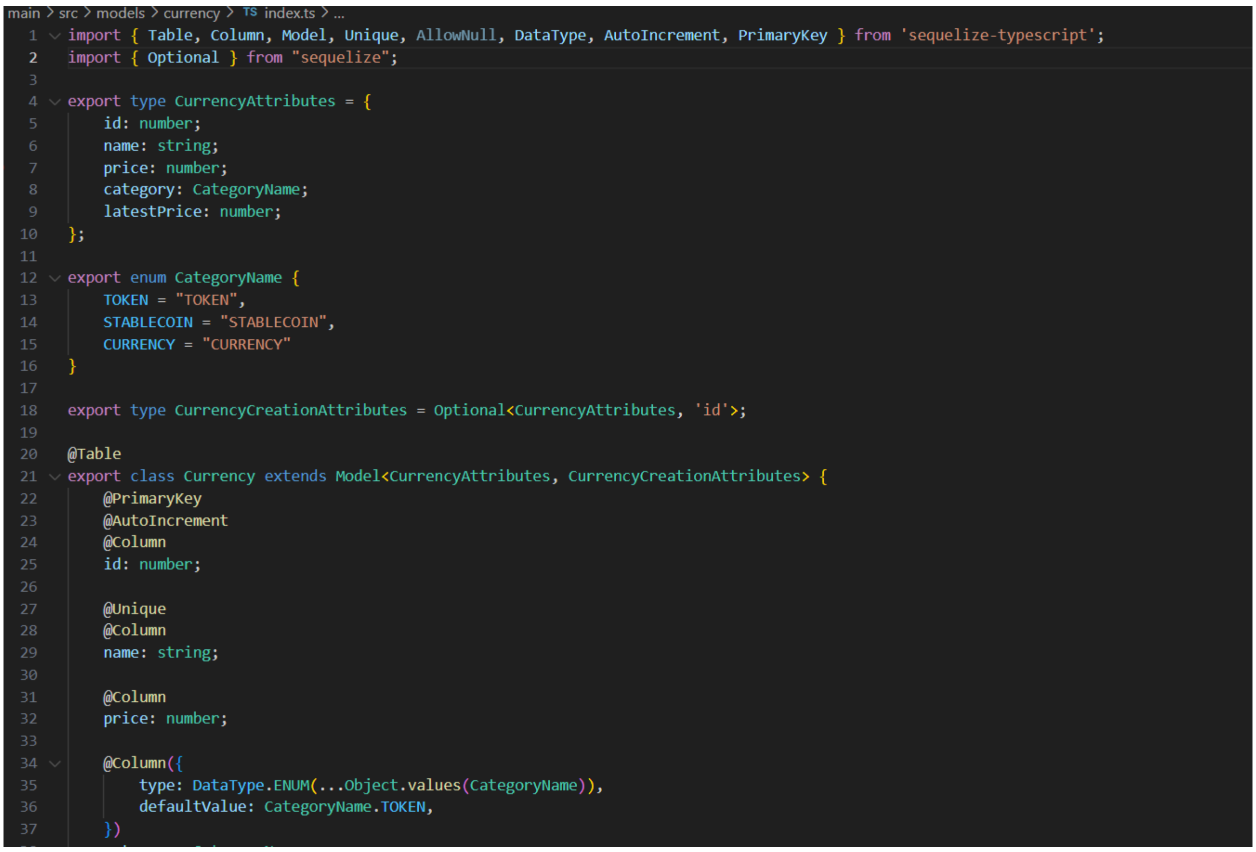
main > src > models > balance > TS index.ts > Balance > userName

```
import { Table, Column, AutoIncrement, PrimaryKey, ForeignKey, Model } from "sequelize-typescript";
import Currency from "../currency";
import { Optional } from "sequelize";

export type BalanceAttributes = {
    id: number;
    userId: number;
    userName: string;
    currency: string;
    currencyId: number;
    count: number;
};

export type BalanceCreationAttributes = Optional<BalanceAttributes, 'id'>;

@Table
export class Balance extends Model<BalanceAttributes, BalanceCreationAttributes> {

    @PrimaryKey
    @AutoIncrement
    @Column
    id: number;

    @Column
    userId: number;

    @Column
    userName: string;

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

    @Column
    currency: string;

```

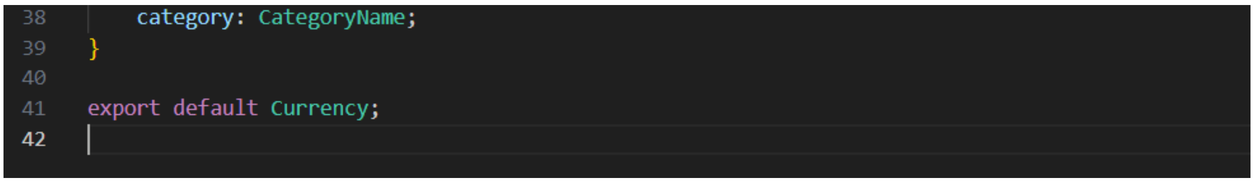


```
    @Column
    count: number;
}

export default Balance;

```

Currency:

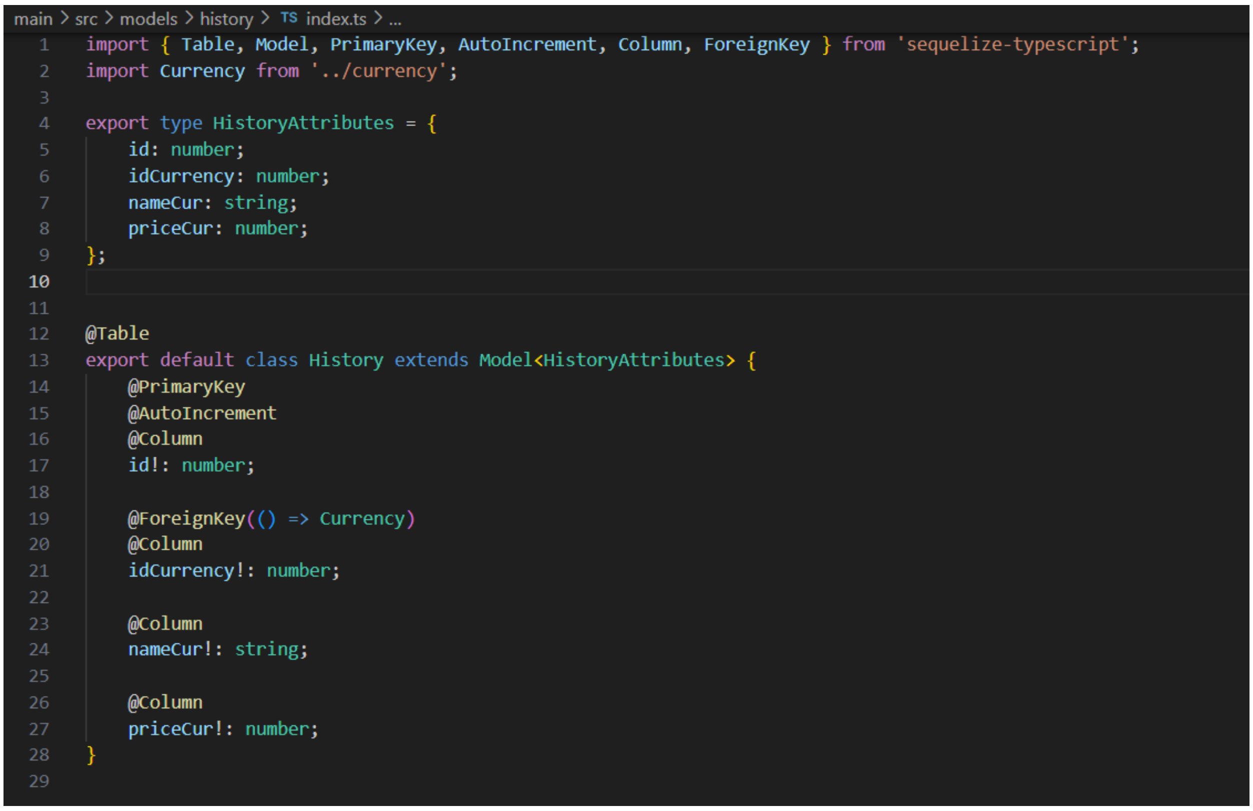


```typescript
import { Table, Column, Model, Unique, AllowNull, DataType, AutoIncrement, PrimaryKey } from 'sequelize-typescript';
import { Optional } from "sequelize";

export type CurrencyAttributes = {
    id: number;
    name: string;
    price: number;
    category: CategoryName;
    latestPrice: number;
};

export enum CategoryName {
    TOKEN = "TOKEN",
    STABLECOIN = "STABLECOIN",
    CURRENCY = "CURRENCY"
}

export type CurrencyCreationAttributes = Optional<CurrencyAttributes, 'id'>;

@Table
export class Currency extends Model<CurrencyAttributes, CurrencyCreationAttributes> {
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number;

    @Unique
    @Column
    name: string;

    @Column
    price: number;

    @Column({
        type: DataType.ENUM(...Object.values(CategoryName)),
        defaultValue: CategoryName.TOKEN,
    })
```



```typescript
    category: CategoryName;
}

export default Currency;

```

History:

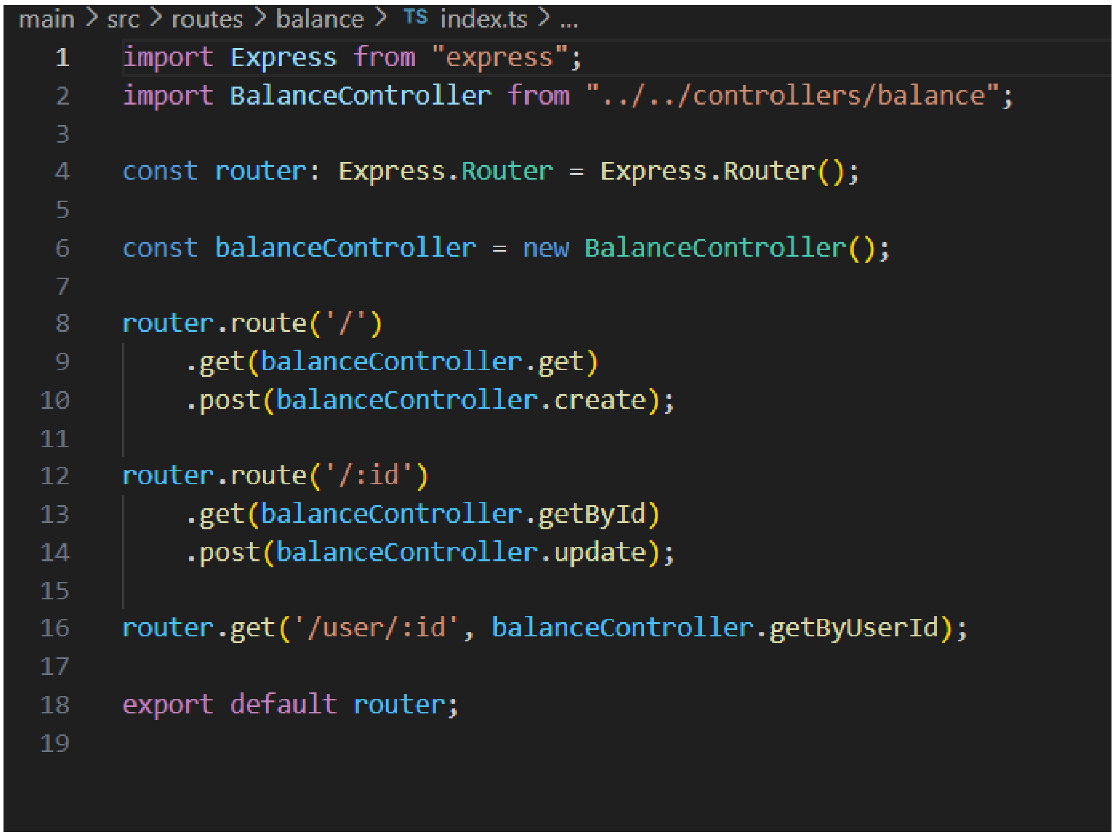


main > src > models > history > TS index.ts > ...

```typescript
import { Table, Model, PrimaryKey, AutoIncrement, Column, ForeignKey } from 'sequelize-typescript';
import Currency from '../currency';

export type HistoryAttributes = {
    id: number;
    idCurrency: number;
    nameCur: string;
    priceCur: number;
};


@Table
export default class History extends Model<HistoryAttributes> {
    @PrimaryKey
    @AutoIncrement
    @Column
    id!: number;

    @ForeignKey(() => Currency)
    @Column
    idCurrency!: number;

    @Column
    nameCur!: string;

    @Column
    priceCur!: number;
}

```

Следующая директория – routes.

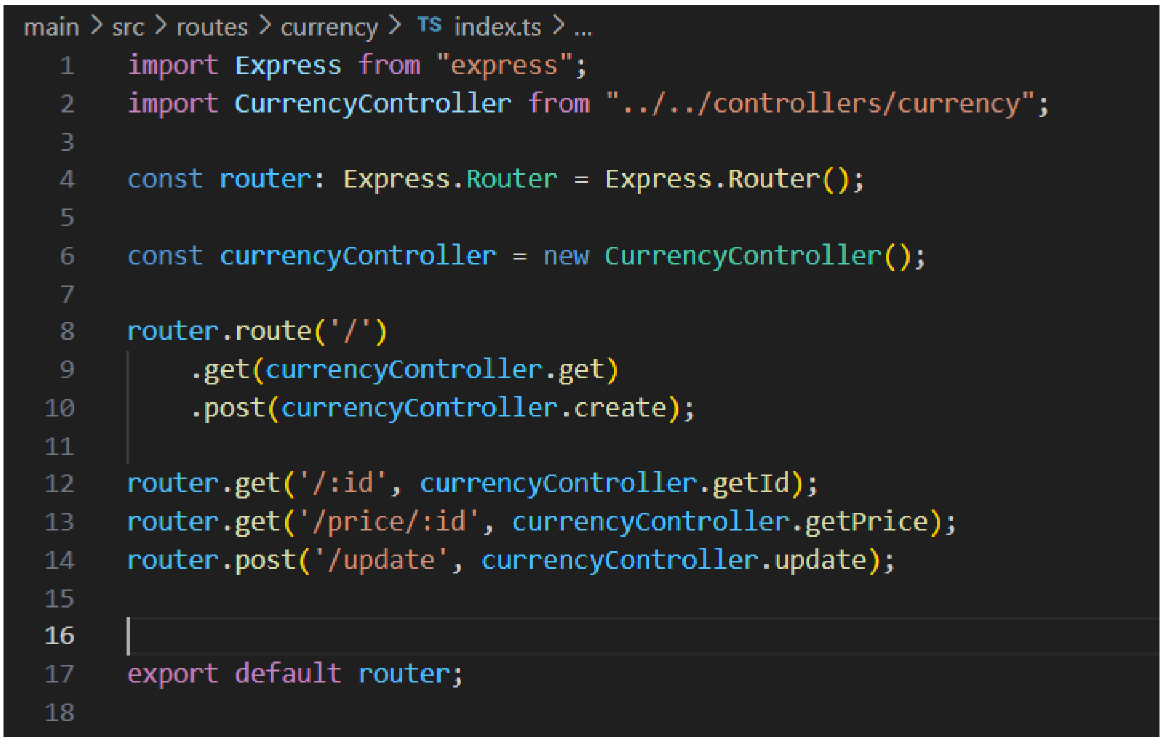
Balance:



main > src > routes > balance > TS index.ts > ...

```typescript
import Express from "express";
import BalanceController from "../../controllers/balance";

const router: Express.Router = Express.Router();

const balanceController = new BalanceController();

router.route('/')
    .get(balanceController.get)
    .post(balanceController.create);

router.route('/:id')
    .get(balanceController.getById)
    .post(balanceController.update);

router.get('/user/:id', balanceController.getByUserId);

export default router;

```

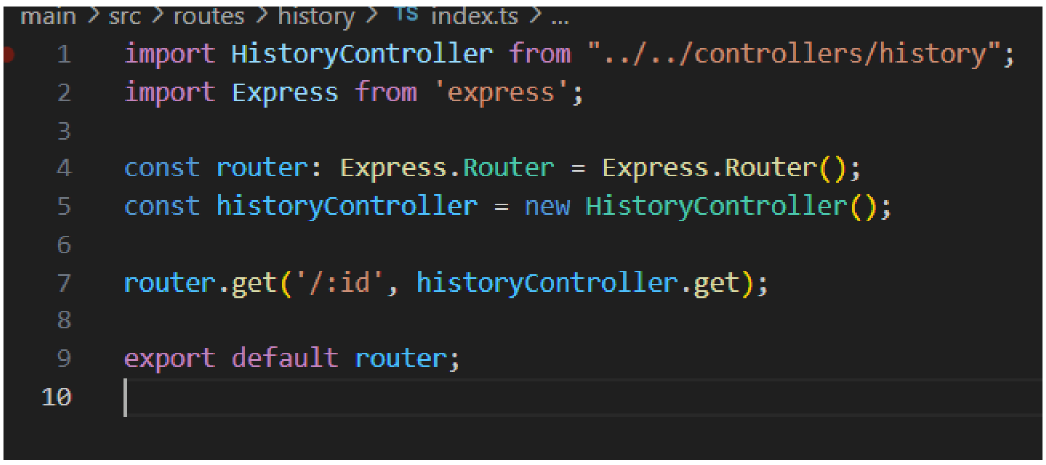
Currency:



main > src > routes > currency > TS index.ts > ...

```typescript
import Express from "express";
import CurrencyController from "../../controllers/currency";

const router: Express.Router = Express.Router();

const currencyController = new CurrencyController();

router.route('/')
    .get(currencyController.get)
    .post(currencyController.create);

router.get('/:id', currencyController.getId);
router.get('/price/:id', currencyController.getPrice);
router.post('/update', currencyController.update);


export default router;

```

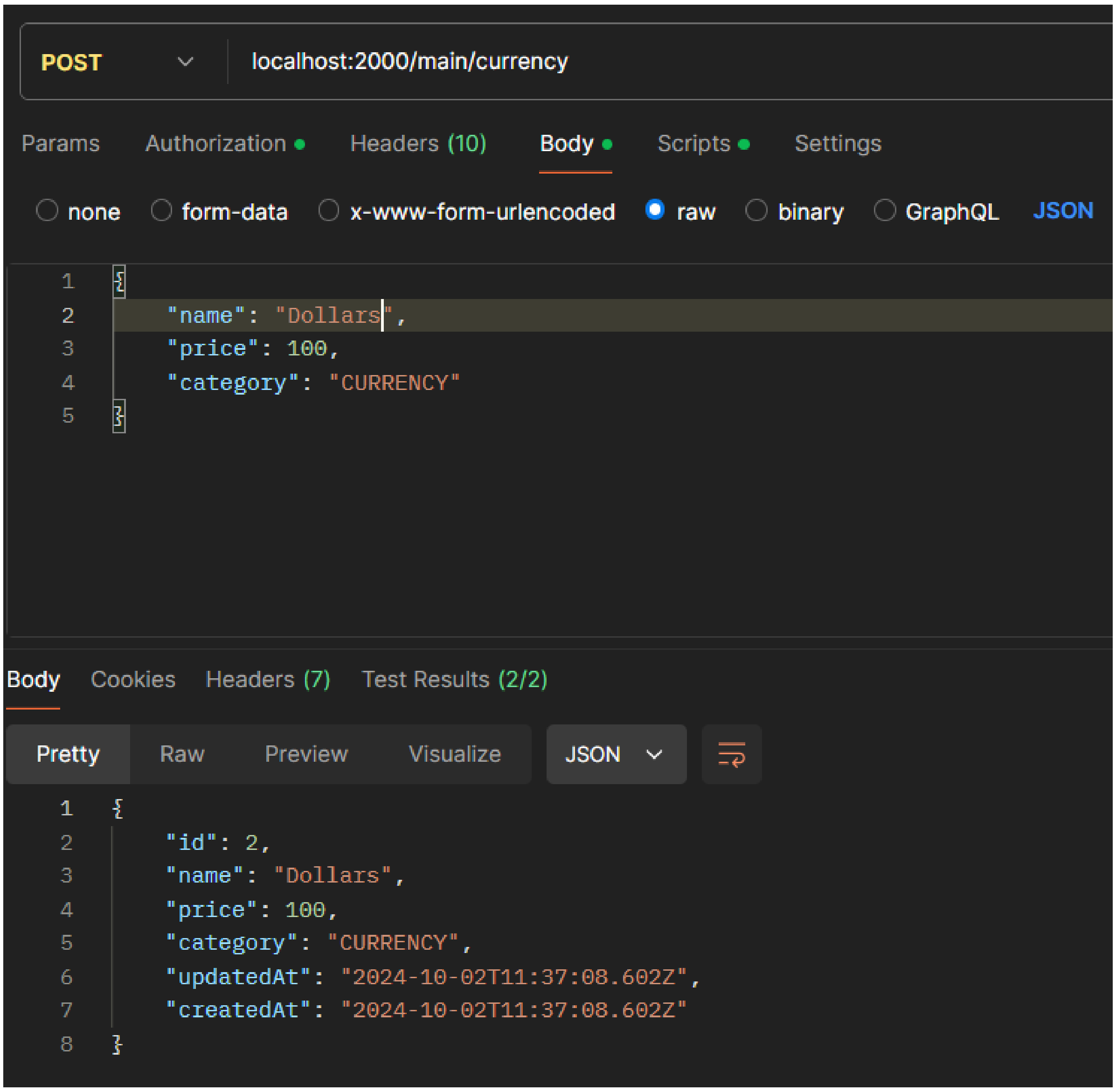
History:



main > src > routes > history > TS index.ts > ...

```typescript
import HistoryController from "../../controllers/history";
import Express from 'express';

const router: Express.Router = Express.Router();
const historyController = new HistoryController();

router.get('/:id', historyController.get);

export default router;

```

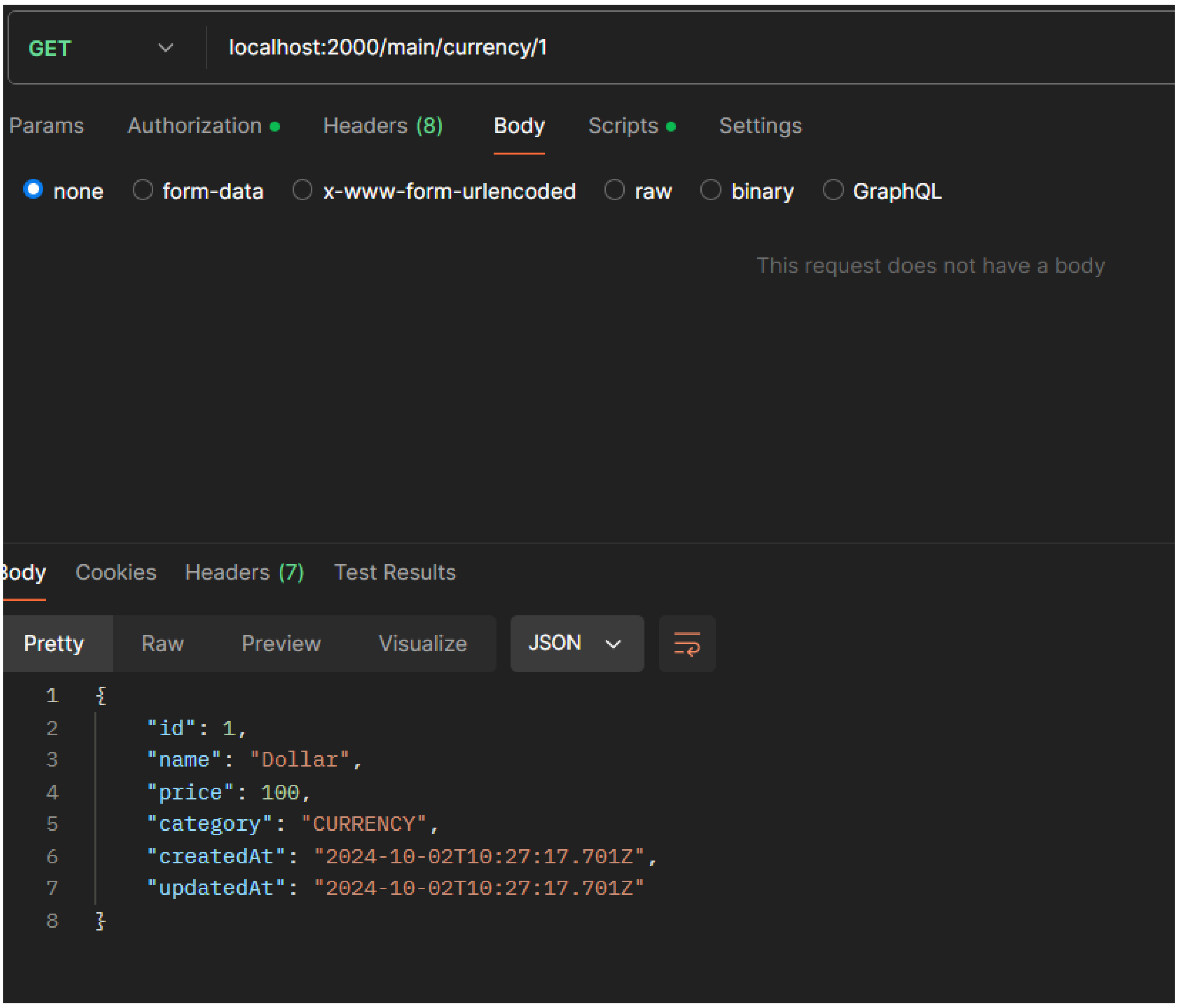
Работа микросервиса:



POST localhost:2000/main/currency

```
{
    "name": "Dollars",
    "price": 100,
    "category": "CURRENCY"
}
```

```
{
    "id": 2,
    "name": "Dollars",
    "price": 100,
    "category": "CURRENCY",
    "updatedAt": "2024-10-02T11:37:08.602Z",
    "createdAt": "2024-10-02T11:37:08.602Z"
}
```


GET localhost:2000/main/currency/1

Params | Authorization | Headers (8) | Body | Scripts | Settings

[✓ none] form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body | Cookies | Headers (7) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
{
    "id": 1,
    "name": "Dollar",
    "price": 100,
    "category": "CURRENCY",
    "createdAt": "2024-10-02T10:27:17.701Z",
    "updatedAt": "2024-10-02T10:27:17.701Z"
}
```

Вывод

В данной лабораторной работе я реализовал микросервис, который позволяет создавать валюту, смотреть историю её изменений и создавать баланс у пользователей с этой валютой .