

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бек-энд разработка

Отчет
Домашнее задание №4

Выполнил:
Горбатов Дмитрий Алексеевич
гр. К33402

Преподаватель:
Добряков Давид Ильич

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:

```
main > src > configs > TS db.ts > [e] syncModels
1  import { Sequelize } from "sequelize-typescript";
2  import Currency from "../models/currency";
3  import Basket from "../models/balance";
4  import History from "../models/history";
5
6  const sequelize = new Sequelize({
7    dialect: 'sqlite',
8    storage: './src/models/models.db',
9  });
10
11  const models = [Currency, Basket, History];
12  sequelize.addModels(models);
13
14  const syncModels = async () => {
15    try {
16      await Currency.sync();
17      console.log('Таблица Currency успешно синхронизирована');
18    } catch (err) {
19      console.error('Ошибка при синхронизации таблицы Currency:', err);
20    }
21  };
22
23  const testConnection = async () => {
24    try {
25      await sequelize.authenticate();
26      console.log('Соединение установлено успешно');
27    } catch (err) {
28      console.error('Ошибка подключения к базе данных:', err);
29    }
30  };
31
32  syncModels();
33  testConnection();
34
35  export default sequelize;
36
```

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

Balance:

```
main > src > controllers > balance > TS index.ts > BalanceController > get
1  import BalanceService from "../../service/balance";
2  import { Request, Response } from "express";
3
4  export default class BalanceController {
5      private balanceService: BalanceService;
6
7      constructor() {
8          this.balanceService = new BalanceService();
9      }
10
11     get = async (req: Request, res: Response) => {
12         try {
13             const result = await this.balanceService.getAll();
14             res.status(200).send(result);
15         } catch (err) {
16             const errorMessage = err instanceof Error ? err.message : "Unknown error";
17             res.status(400).send({ error: errorMessage });
18         }
19     };
20
21     getById = async (req: Request, res: Response) => {
22         try {
23             const result = await this.balanceService.getById(Number(req.params.id));
24             res.status(200).send(result);
25         } catch (err) {
26             const errorMessage = err instanceof Error ? err.message : "Unknown error";
27             res.status(400).send({ error: errorMessage });
28         }
29     };
30
31     getUserId = async (req: Request, res: Response) => {
32         try {
33             const result = await this.balanceService.getUserId(
34                 Number(req.params.id)
35             );
36             res.status(200).send(result);
37         } catch (err) {
38             const errorMessage = err instanceof Error ? err.message : "Unknown error";
39             res.status(400).send({ error: errorMessage });
40         }
41     };
42 }
```

```

31  getByUserId = async (req: Request, res: Response) => {
32      try {
33          const result = await this.balanceService.getByUserId(
34              Number(req.params.id)
35          );
36          res.status(200).send(result);
37      } catch (err) {
38          const errorMessage = err instanceof Error ? err.message : "Unknown error";
39          res.status(400).send({ error: errorMessage });
40      }
41  };
42
43  create = async (req: Request, res: Response) => {
44      try {
45          const userid = req.headers["user-id"];
46          console.log(userid, typeof userid);
47          const result = await this.balanceService.create(
48              Number(userid),
49              req.body.currencyId,
50              req.body
51          );
52          res.status(201).send({ result });
53      } catch (err) {
54          const errorMessage = err instanceof Error ? err.message : "Unknown error";
55          res.status(400).send({ error: errorMessage });
56      }
57  };
58
59  update = async (req: Request, res: Response) => {
60      try {
61          const result = await this.balanceService.update(
62              Number(req.params.id),
63              req.body.product
64          );
65          res.status(200).send(result);
66      } catch (err) {

```

Currency:

```
38 ✓ create = async (req: Request, res: Response) => {
39   try {
40     const product = await this.currencyService.createCurrency(req.body);
41     res.status(201).send(product);
42   } catch (err) {
43     const errorMessage = err instanceof Error ? err.message : 'Unknown error';
44     console.log(errorMessage);
45     res.status(400).send({ error: errorMessage });
46   }
47 };
48
49 ✓ getPrice = async (req: Request, res: Response) => {
50   try {
51     const id: number = Number(req.params.id);
52     const result = await this.currencyService.getById(id);
53     if (result === null) {
54       res.status(404).send('Product not found');
55       return;
56     }
57     res.status(200).send({ price: result.price });
58   } catch (err) {
59     const errorMessage = err instanceof Error ? err.message : 'Unknown error';
60     console.log(errorMessage);
61     res.status(400).send({ error: errorMessage });
62   }
63 };
64
65 ✓ update = async (req: Request, res: Response) => {
66   try {
67     const id: number = Number(req.body.id);
68     const result = await this.currencyService.updatePrice(id, req.body.price);
69     res.status(200).send(`Updated successfully: \n ${result}`);
70   } catch (err) {
71     const errorMessage = err instanceof Error ? err.message : 'Unknown error';
72     res.status(400).send({ error: errorMessage });
73   }
}
```

```
main > src > controllers > currency > TS index.ts > CurrencyController > get > errorMessage
1  import { Request, Response } from "express";
2  import CurrencyService from "../../service/currency";
3
4  export default class CurrencyController {
5      private currencyService: CurrencyService;
6
7      constructor() {
8          this.currencyService = new CurrencyService();
9      }
10
11     get = async (req: Request, res: Response) => {
12         try {
13             const product = await this.currencyService.getAll();
14             res.status(200).send(product);
15         } catch (err) {
16             const errorMessage = err instanceof Error ? err.message : 'Unknown error';
17             console.log(errorMessage);
18             res.status(400).send({ error: errorMessage });
19         }
20     };
21
22     getId = async (req: Request, res: Response) => {
23         try {
24             const id: number = Number(req.params.id);
25             const result = await this.currencyService.getById(id);
26             if (result === null) {
27                 res.status(404).send('Product not found');
28                 return;
29             }
30             res.status(200).send(result);
31         } catch (err) {
32             const errorMessage = err instanceof Error ? err.message : 'Unknown error';
33             console.log(errorMessage);
34             res.status(400).send({ error: errorMessage });
35         }
36     };
37
```

History:

```
main > src > controllers > history > TS index.ts > ...
1  import HistoryService from "../../service/history";
2  import { Request, Response } from 'express';
3
4  class HistoryController {
5      private historyService: HistoryService;
6
7      constructor() {
8          this.historyService = new HistoryService();
9      }
10
11     get = async (req: Request, res: Response) => {
12         try {
13             const result = await this.historyService.getById(Number(req.params.id));
14             if (result === null) {
15                 res.status(404).send('History record not found');
16                 return;
17             }
18             res.status(200).send(result);
19         } catch (err) {
20             const errorMessage = err instanceof Error ? err.message : 'Unknown error';
21             console.log(errorMessage);
22             res.status(400).send({ error: errorMessage });
23         }
24     };
25 }
26
27 export default HistoryController;
28
```

Следующая папка – модели.

Balance:

```
main > src > models > balance > TS index.ts > Balance > userName
1  import { Table, Column, AutoIncrement, PrimaryKey, ForeignKey, Model } from "sequelize-typescript";
2  import Currency from "../currency";
3  import { Optional } from "sequelize";
4
5  export type BalanceAttributes = {
6      id: number;
7      userId: number;
8      userName: string;
9      currency: string;
10     currencyId: number;
11     count: number;
12 };
13
14 export type BalanceCreationAttributes = Optional<BalanceAttributes, 'id'>;
15
16 @Table
17 export class Balance extends Model<BalanceAttributes, BalanceCreationAttributes> {
18
19     @PrimaryKey
20     @AutoIncrement
21     @Column
22     id: number;
23
24     @Column
25     userId: number;
26
27     @Column
28     userName: string;
29
30     @ForeignKey(() => Currency)
31     @Column
32     currencyId: number;
33
34     @Column
35     currency: string;
36
37     @Column
38     count: number;
39 }
40
41 export default Balance;
42
```

Currency:

```
main > src > models > currency > TS index.ts > ...
1  import { Table, Column, Model, Unique, AllowNull, DataType, AutoIncrement, PrimaryKey } from 'sequelize-typescript';
2  import { Optional } from "sequelize";
3
4  export type CurrencyAttributes = {
5      id: number;
6      name: string;
7      price: number;
8      category: CategoryName;
9      latestPrice: number;
10 };
11
12 export enum CategoryName {
13     TOKEN = "TOKEN",
14     STABLECOIN = "STABLECOIN",
15     CURRENCY = "CURRENCY"
16 }
17
18 export type CurrencyCreationAttributes = Optional<CurrencyAttributes, 'id'>;
19
20 @Table
21 export class Currency extends Model<CurrencyAttributes, CurrencyCreationAttributes> {
22     @PrimaryKey
23     @AutoIncrement
24     @Column
25     id: number;
26
27     @Unique
28     @Column
29     name: string;
30
31     @Column
32     price: number;
33
34     @Column({
35         type: DataType.ENUM(...Object.values(CategoryName)),
36         defaultValue: CategoryName.TOKEN,
37     })
```

```
38     category: CategoryName;
39 }
40
41 export default Currency;
42
```

History:

```
main > src > models > history > TS index.ts > ...
1  import { Table, Model, PrimaryKey, AutoIncrement, Column, ForeignKey } from 'sequelize-typescript';
2  import Currency from '../currency';
3
4  export type HistoryAttributes = {
5      id: number;
6      idCurrency: number;
7      nameCur: string;
8      priceCur: number;
9  };
10
11
12  @Table
13  export default class History extends Model<HistoryAttributes> {
14      @PrimaryKey
15      @AutoIncrement
16      @Column
17      id!: number;
18
19      @ForeignKey(() => Currency)
20      @Column
21      idCurrency!: number;
22
23      @Column
24      nameCur!: string;
25
26      @Column
27      priceCur!: number;
28  }
29
```

Следующая директория – routes.

Balance:

```
main > src > routes > balance > TS index.ts > ...
1  import Express from "express";
2  import BalanceController from "../../controllers/balance";
3
4  const router: Express.Router = Express.Router();
5
6  const balanceController = new BalanceController();
7
8  router.route('/')
9      .get(balanceController.get)
10     .post(balanceController.create);
11
12  router.route('/:id')
13     .get(balanceController.getById)
14     .post(balanceController.update);
15
16  router.get('/user/:id', balanceController.getByUserId);
17
18  export default router;
19
```

Currency:

```
main > src > routes > currency > TS index.ts > ...
1  import Express from "express";
2  import CurrencyController from "../../controllers/currency";
3
4  const router: Express.Router = Express.Router();
5
6  const currencyController = new CurrencyController();
7
8  router.route('/')
9    .get(currencyController.get)
10   .post(currencyController.create);
11
12  router.get('/:id', currencyController.getId);
13  router.get('/price/:id', currencyController.getPrice);
14  router.post('/update', currencyController.update);
15
16  |
17  export default router;
18
```

History:

```
main > src > routes > history > TS index.ts > ...
1  import HistoryController from "../../controllers/history";
2  import Express from 'express';
3
4  const router: Express.Router = Express.Router();
5  const historyController = new HistoryController();
6
7  router.get('/:id', historyController.get);
8
9  export default router;
10 |
```

Работа микросервиса:

The screenshot displays a REST client interface with a POST request to `localhost:2000/main/currency`. The request body is a JSON object with the following fields: `"name": "Dollars", "price": 100, "category": "CURRENCY"`. The response is also in JSON format, showing the server's reply with an `id`, the same request data, and timestamps for `updatedAt` and `createdAt`.

Request:

```
POST localhost:2000/main/currency
```

Params Authorization Headers (10) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

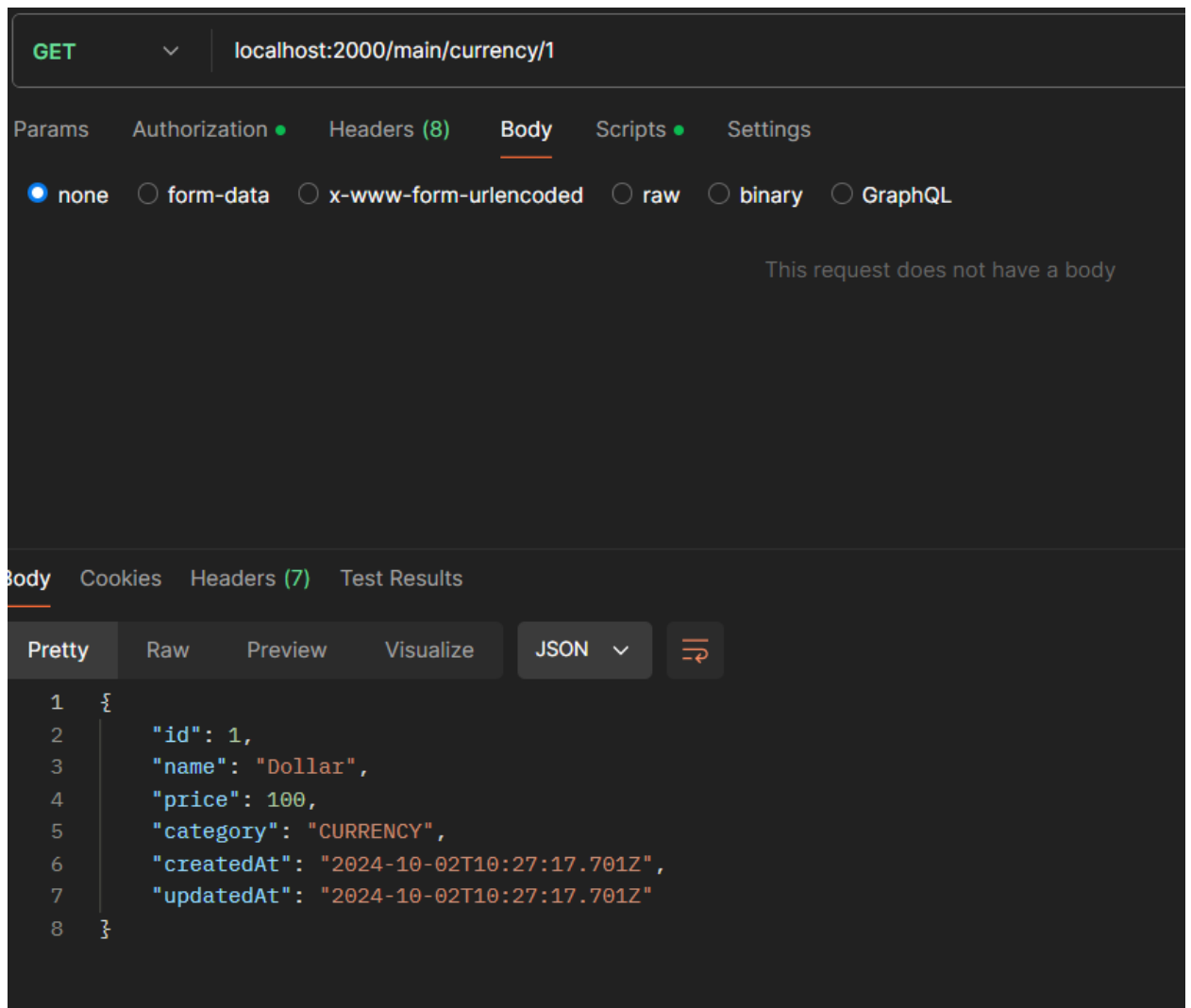
```
1 {
2   "name": "Dollars",
3   "price": 100,
4   "category": "CURRENCY"
5 }
```

Response:

Body Cookies Headers (7) Test Results (2/2)

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "name": "Dollars",
4   "price": 100,
5   "category": "CURRENCY",
6   "updatedAt": "2024-10-02T11:37:08.602Z",
7   "createdAt": "2024-10-02T11:37:08.602Z"
8 }
```



Вывод

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