

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 2

Выполнила:

Афанасьева Ирина Максимовна

Группа:

К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задание

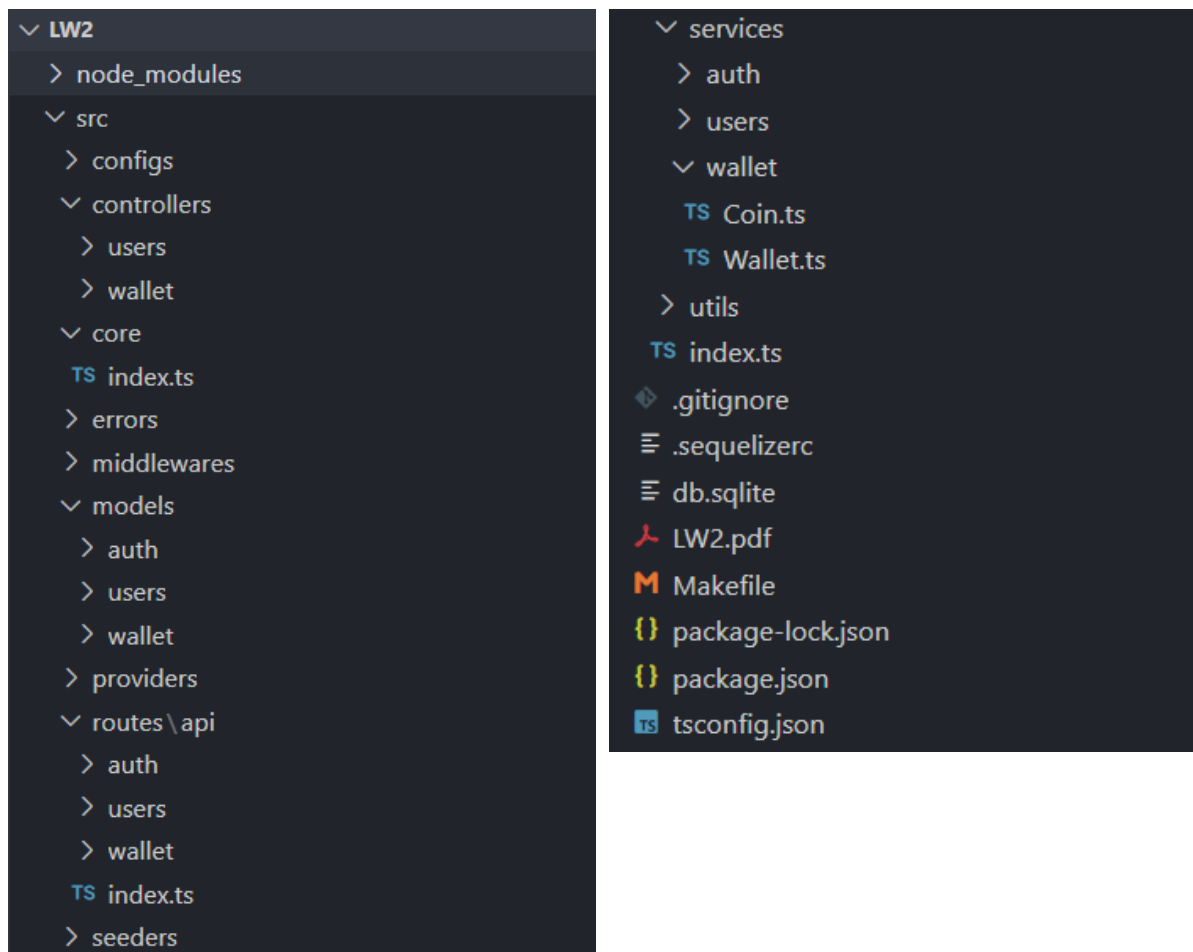
По выбранному варианту необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Ход работы

Выбранный вариант: 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

Структура приложения:



Модели:

RefreshToken.ts

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

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

export default RefreshToken
```

Coin.ts

```
@Table
class Coin extends Model {
  @PrimaryKey
  @AllowNull(false)
  @Column
  ticker: string

  @AllowNull(false)
  @Column
  name: string

  @BelongsToMany(() => Wallet, () => CoinWallet)
  wallets: Wallet[]
}

export default Coin
```

User.ts

```
@Table
class User extends Model {
  @AllowNull(false)
  @Column
  firstName: string

  @AllowNull(false)
  @Column
  lastName: string

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

  @HasMany(() => Wallet)
  wallets: [Wallet]

  @AllowNull(false)
  @Column
  password: string

  @Column
  salt: string

  @BeforeCreate
  static setPasswordInitial(instance: User) {
    instance.salt = generateSalt()
    const {password} = instance
    instance.password = encodePassword(password, instance.salt)
  }

  @BeforeUpdate
  static setPasswordUpdate(instance: User) {
    const {password} = instance
    if (instance.changed('password')) {
      instance.password = encodePassword(password, instance.salt)
    }
  }
}
```

CoinWallet.ts

```
@Table
class CoinWallet extends Model {
  @AllowNull(false)
  @Column
  amount: number

  @ForeignKey(() => Coin)
  @Column
  coinId: number

  @ForeignKey(() => Wallet)
  @Column
  walletId: number
}
```

Wallet.ts

```
@Table
class Wallet extends Model {
  @AllowNull(false)
  @Column
  name: string

  @Default(0)
  @Column({
    type: DataType.FLOAT,
    set(this, val): void {
      console.log('Balance not allowed to be set')
    }
  })
  balance: number

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

  @BelongsTo(() => User)
  user: User

  @HasMany(() => CoinWallet)
  coinWallets: CoinWallet[]

  @BelongsToMany(() => Coin, () => CoinWallet)
  coins: Coin[]
}
```

Новые контроллеры:

Coin.ts

```
class CoinController {
  private coinService: CoinService

  constructor() {
    this.coinService = new CoinService()
  }

  get = async (request: any, response: any) => {
    const {ticker} = request.params
    try {
      response.send(await this.coinService.getByTicker(ticker))
    } catch (e: any) {
      response.status(404).send({'detail': e.message})
    }
  }
}
```

```

getAll = async (request: any, response: any) => {
  const time = request.query.time
  let seconds = null
  if (time) {
    seconds = TIME_INTERVAL_MAPPING[time]
  }
  if (seconds) {
    response.send(await this.coinService.getAllFilter(new Date(Date.now() - seconds * 1000)))
  } else {
    response.send(await this.coinService.getAll())
  }
}

price = async (request: any, response: any) => {
  const {ticker} = request.params
  try {
    response.send(await this.coinService.getPrice(ticker))
  } catch (e: any) {
    response.status(404).send({'detail': e.message})
  }
}

```

Wallet.ts

```

class WalletController {
  private walletService: WalletService

  constructor() {
    this.walletService = new WalletService()
  }

  private wrapperFunc = async (
    request: any,
    response: any,
    func: (id: number, walletData: any) => Promise<Wallet | APIError>
  ) => {
    const {body, user} = request
    const {id} = request.params
    if (user && await this.walletService.checkUser(id, user.id)) {
      try {
        const wallet = await func(id, body)
        response.status(200).send(wallet)
      } catch (error: any) {
        response.status(400).send({'detail': error.message})
      }
    } else {
      response.status(403).send({'detail': 'Unauthorized'})
    }
  }
}

```

```

get = async (request: any, response: any) => {
  const {user} = request
  const {id} = request.params
  if (user && await this.walletService.checkUser(id, user.id)) {
    try {
      const wallet: Wallet | APIError = await this.walletService.getById(id)
      response.status(200).send(wallet)
    } catch (error: any) {
      response.status(404).send({'detail': error.message})
    }
  } else {
    response.status(403).send({'detail': 'Unauthorized'})
  }
}

getAll = async (request: any, response: any) => {
  const {user} = request
  if (user) {
    response.send(await this.walletService.getByUserId(user.id))
  } else {
    response.status(403).send({'detail': 'Unauthorized'})
  }
}

```

```

post = async (request: any, response: any) => {
  const {body, user} = request
  if (user) {
    try {
      const wallet: Wallet | APIError = await this.walletService.create(body, user)
      response.status(201).send(wallet)
    } catch (error: any) {
      response.status(400).send({'detail': error.message})
    }
  } else {
    response.status(401).send({'detail': 'Not authenticated'})
  }
}

increaseBalance = async (request: any, response: any) => {
  return this.wrapperFunc(request, response, this.walletService.increaseBalance)
}

decreaseBalance = async (request: any, response: any) => {
  return this.wrapperFunc(request, response, this.walletService.decreaseBalance)
}

buyCoin = async (request: any, response: any) => {
  return this.wrapperFunc(request, response, this.walletService.buyCoin)
}

sellCoin = async (request: any, response: any) => {
  return this.wrapperFunc(request, response, this.walletService.sellCoin)
}

sellAllCoin = async (request: any, response: any) => {
  return this.wrapperFunc(request, response, this.walletService.sellAllCoin)
}

```

Новые сервисы:

Coin.ts

```
class CoinService {
  async getByTicker(ticker: string): Promise<Coin | APIError> {
    const coin = await Coin.findByPk(ticker)
    if (coin) {
      return coin.toJSON()
    }
    throw new APIError('Coin not found')
  }

  async getAll(): Promise<Coin[]> {
    return await Coin.findAll()
  }

  async getAllFilter(from: Date): Promise<Coin[]> {
    return await Coin.findAll({
      where: {
        createdAt: {[Op.gte]: from}
      }
    })
  }

  async getPrice(ticker: string): Promise<{ price: number } | APIError> {
    // Get only to check
    const coin = await Coin.findByPk(ticker)
    if (coin) {
      return {price: await getCurrentPrice(ticker)}
    }
    throw new APIError('Coin not found')
  }
}
```

Wallet.ts

```
class WalletService {
  async getById(id: number): Promise<Wallet | APIError> {
    const wallet = await Wallet.scope('nested').findByPk(id)
    if (wallet) {
      return wallet.toJSON()
    }
    throw new APIError('Wallet not found')
  }

  async getByUserId(userId: number): Promise<Wallet[]> {
    return await Wallet.findAll({where: {userId: userId}})
  }

  async checkUser(id: number, userId: number): Promise<boolean> {
    const wallet = await Wallet.findByPk(id)
    if (wallet) {
      return wallet.userId == userId
    }
    return false
  }
}
```



```

async create(walletData: any, user: User): Promise<Wallet | APIError> {
  try {
    const wallet = await Wallet.create({...walletData, userId: user.id})
    return wallet.toJSON()
  } catch (e: any) {
    const errors = e.errors.map((error: any) => error.message)
    throw new APIError(errors)
  }
}

async increaseBalance(id: number, data: any): Promise<Wallet | APIError> {
  const {amount} = data
  const wallet = await Wallet.scope('nested').findByPk(id)
  if (wallet) {
    if (amount <= 0) {
      throw new APIError(`Got non-positive amount: ${amount}`)
    }
    console.log(amount)
    wallet.setDataValue('balance', wallet.balance + amount)
    await wallet.save()
    await wallet.reload()
    return wallet.toJSON()
  }
  throw new APIError('Wallet not found')
}

```

```

async decreaseBalance(id: number, data: any): Promise<Wallet | APIError> {
  const {amount} = data
  const wallet = await Wallet.scope('nested').findByPk(id)
  if (wallet) {
    if (amount <= 0) {
      throw new APIError(`Got non-positive amount: ${amount}`)
    }
    if (amount > wallet.getDataValue('balance')) {
      throw new APIError('Not enough money')
    }
    wallet.setDataValue('balance', wallet.balance - amount)
    await wallet.save()
    await wallet.reload()
    return wallet.toJSON()
  }
  throw new APIError('Wallet not found')
}

```

```

async buyCoin(id: number, data: any): Promise<Wallet | APIError> {
  const {amount, ticker} = data
  if (amount <= 0) {
    throw new APIError(`Got non-positive amount: ${amount}`)
  }
  const wallet = await Wallet.scope('nested').findByPk(id)
  const coin = await Coin.findByPk(ticker)
  if (wallet && coin) {
    let coinWallet = await CoinWallet.findOne({where: {coinId: ticker, walletId: id}})
    if (!coinWallet) {
      coinWallet = await CoinWallet.create({amount: 0, coinId: ticker, walletId: id})
    }
    let price: number
    try {
      price = await getCurrentPrice(ticker)
    } catch (e: any) {
      throw new APIError(e.message)
    }
    if (price * amount > wallet.getDataValue('balance')) {
      throw new APIError('Not enough money')
    }
    wallet.setDataValue('balance', wallet.getDataValue('balance') - price * amount)
    coinWallet.amount += amount
    await coinWallet.save()
    await wallet.save()
    await wallet.reload()
    return wallet.toJSON()
  } else if (!wallet) {
    throw new APIError('Wallet not found')
  } else {
    throw new APIError(`Unknown coin: ${ticker}`)
  }
}

```

```

async sellCoin(id: number, data: any): Promise<Wallet | APIError> {
  const {amount, ticker} = data
  if (amount <= 0) {
    throw new APIError(`Got non-positive amount: ${amount}`)
  }
  const wallet = await Wallet.scope('nested').findByPk(id)
  const coin = await Coin.findByPk(ticker)
  if (wallet && coin) {
    const coinWallet = await CoinWallet.findOne({where: {coinId: ticker, walletId: id}})
    if (coinWallet) {
      if (amount > coinWallet.amount) {
        throw new APIError(`You don't have ${amount} ${coin.name}`)
      }
      try {
        const price = await getCurrentPrice(ticker)
        wallet.setDataValue('balance', wallet.getDataValue('balance') + price * amount)
      } catch (e: any) {
        throw new APIError(e.message)
      }
      if (amount == coinWallet.amount) {
        await coinWallet.destroy()
      } else {
        coinWallet.amount -= amount
        await coinWallet.save()
      }
      await wallet.save()
      await wallet.reload()
      return wallet.toJSON()
    } else {
      throw new APIError(`You don't have any ${coin.name}`)
    }
  } else if (!wallet) {
    throw new APIError('Wallet not found')
  } else {
    throw new APIError(`Unknown coin: ${ticker}`)
  }
}

```

```

async sellAllCoin(id: number, data: any): Promise<Wallet | APIError> {
  const {ticker} = data
  const wallet = await Wallet.scope('nested').findByPk(id)
  const coin = await Coin.findByPk(ticker)
  if (wallet && coin) {
    const coinWallet = await CoinWallet.findOne({where: {coinId: ticker, walletId: id}})
    if (coinWallet) {
      const amount = coinWallet.amount
      try {
        const price = await getCurrentPrice(ticker)
        wallet.setDataValue('balance', wallet.getDataValue('balance') + price * amount)
      } catch (e: any) {
        throw new APIError(e.message)
      }
      await coinWallet.destroy()
      await wallet.save()
      await wallet.reload()
      return wallet.toJSON()
    } else {
      throw new APIError(`You don't have any ${coin.name}`)
    }
  } else if (!wallet) {
    throw new APIError('Wallet not found')
  } else {
    throw new APIError(`Unknown coin: ${ticker}`)
  }
}

```

Пример работы API:

Создание пользователя:

The screenshot shows a REST client interface with a POST request to `http://localhost:5000/api/users`. The request body is a JSON object: `{ "firstName": "Irina", "lastName": "Afanasieva", "email": "multifructina@gmail.ru", "password": "5snemELX" }`. The response is also shown in JSON format: `{ "id": 2, "firstName": "Irina", "lastName": "Afanasieva", "email": "multifructina@gmail.ru" }`.

```
POST http://localhost:5000/api/users

{ "firstName": "Irina", "lastName": "Afanasieva", "email": "multifructina@gmail.ru", "password": "5snemELX" }
```

```
{ "id": 2, "firstName": "Irina", "lastName": "Afanasieva", "email": "multifructina@gmail.ru" }
```

Вход:

The screenshot shows a REST client interface with a POST request to `http://localhost:5000/api/auth/login`. The request body is a JSON object: `{ "email": "multifructina@gmail.ru", "password": "5snemELX" }`. The response is a JSON object containing access and refresh tokens: `{ "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiYW90IjoxNjg2MzM3MTM2fQ.LAYzakleUt4Va98smVh5jn15faHjmc0q0sw8tP1BPrg", "refreshToken": "4799fba7-b9c3-48e6-9e31-606585a5369e" }`. The status is 200 OK, and the response size is 455 B.

```
POST http://localhost:5000/api/auth/login

{ "email": "multifructina@gmail.ru", "password": "5snemELX" }
```

```
{ "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiYW90IjoxNjg2MzM3MTM2fQ.LAYzakleUt4Va98smVh5jn15faHjmc0q0sw8tP1BPrg", "refreshToken": "4799fba7-b9c3-48e6-9e31-606585a5369e" }
```

Status: 200 OK Time: 171 ms Size: 455 B Save as Example

Получение профиля:

GET ▼ http://127.0.0.1:5000/api/users/me Send ▼

Params **Authorization** ● Headers (7) Body Pre-request Script Tests Settings Cookies

Type Bearer Token ▼ Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#) ↗

body Cookies Headers (8) Test Results ⊕ Status: 200 OK Time: 18 ms Size: 352 B Save as Example ⋮

Pretty Raw Preview Visualize JSON ▼ ⋮

```
1 {
2   "id": 1,
3   "firstName": "Irina",
4   "lastName": "Afanasieva",
5   "email": "multifructina@gmail.ru"
6 }
```

Создание портфеля:

POST ▼ http://localhost:5000/api/wallets Send ▼

Params **Authorization** ● Headers (9) **Body** ● Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON ▼ Beautify

```
1 {
2   "name": "my_first_wallet"
3 }
```

body Cookies Headers (8) Test Results ⊕ Status: 201 Created Time: 152 ms Size: 407 B Save as Example ⋮

Pretty Raw Preview Visualize JSON ▼ ⋮

```
1 {
2   "balance": 0,
3   "id": 3,
4   "name": "my_first_wallet",
5   "userId": 2,
6   "updatedAt": "2023-06-09T19:06:42.514Z",
7   "createdAt": "2023-06-09T19:06:42.514Z"
8 }
```

Добавление денег в портфель:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/api/wallets/3/add_money
- Body:**

```
{ "amount": 1000000 }
```
- Response:**

```
{ "id": 3, "name": "my_first_wallet", "balance": 1000000, "userId": 2, "coinWallets": [] }
```
- Status:** 200 OK
- Time:** 207 ms
- Size:** 346 B

Покупка криптовалюты:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/api/wallets/3/buy_coin
- Body:**

```
{ "ticker": "btc", "amount": "5" }
```
- Response:**

```
{ "id": 3, "name": "my_first_wallet", "balance": 915835, "userId": 2, "coinWallets": [ { "amount": 5, "walletId": 3, "coinId": "btc" } ] }
```
- Status:** 200 OK
- Time:** 945 ms
- Size:** 386 B

Продажа криптовалюты:

POST http://localhost:5000/api/wallets/3/sell_coin

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 [{"ticker": "btc", "amount": 3}]
```

body Cookies Headers (8) Test Results Status: 200 OK Time: 552 ms Size: 386 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "name": "my_first_wallet",
4   "balance": 966334,
5   "userId": 2,
6   "coinWallets": [
7     {
8       "amount": 2,
9       "walletId": 3,
10      "coinId": "btc"
11    }
12  ]
13 }
```

Вывод денег с биржи:

POST http://localhost:5000/api/wallets/3/withdraw_money

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 [{"amount": 960000}]
```

body Cookies Headers (8) Test Results Status: 200 OK Time: 207 ms Size: 384 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "name": "my_first_wallet",
4   "balance": 6334,
5   "userId": 2,
6   "coinWallets": [
7     {
8       "amount": 2,
9       "walletId": 3,
10      "coinId": "btc"
11    }
12  ]
13 }
```

Список доступных криптовалют:

GET

http://localhost:5000/api/coins

Send

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettingsCookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

BodyCookiesHeaders (8)Test Results

Status: 200 OKTime: 11 msSize: 788 BSave as Example

PrettyRawPreviewVisualizeJSON

```
1 {
2   {
3     "ticker": "btc",
4     "name": "Bitcoin",
5     "createdAt": "2012-01-01T00:00:00.000Z"
6   },
7   {
8     "ticker": "eth",
9     "name": "Ethereum",
10    "createdAt": "2018-03-05T00:00:00.000Z"
11  },
12  {
13    "ticker": "doge",
14    "name": "Dogecoin",
15    "createdAt": "2013-01-01T00:00:00.000Z"
16  }
17 }
```

Цена криптовалюты в данный момент (используется API api.cryptowat.ch и данные биржи FTX):

GET

http://localhost:5000/api/coins/btc/price

Send

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettingsCookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

BodyCookiesHeaders (8)Test Results

Status: 200 OKTime: 306 msSize: 281 BSave as Example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "price": 16833
3 }
```

Вывод

В ходе работы был реализован RESTful API для сайта криптобиржи с использованием фреймворка Express, ORM Sequelize, Axios и другими библиотеками.