САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Таначев Егор

Группа K33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate). Сделать платформу для администратора интернет-магазина компьютерной техники, которая будет включать:

- Вход;
- Регистрацию;
- Учет товара на складе;
- Графики по продажам товаров, по общей выручке предприятия;
- Управление сотрудниками.

Ход работы

Для начала работы необходимо продумать структуру данных. В нашей платформе будет пять моделей данных: информация о пользователе, токен для авторизации, текущий состав склада, поставки и реализация товара, как показано на Рисунке 1.

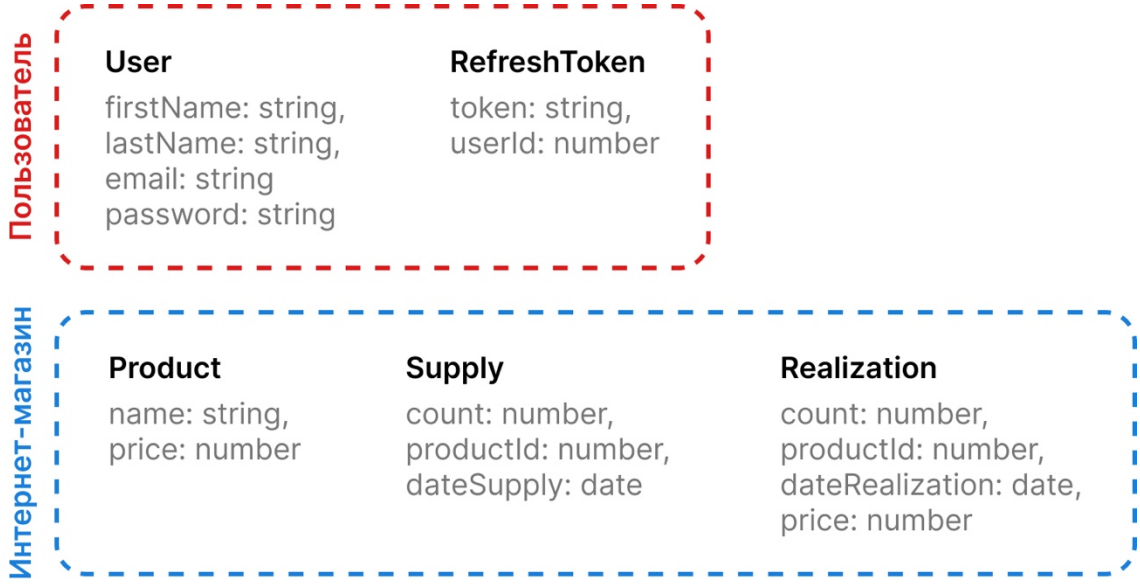


Рисунок 1 – Схема базы данных

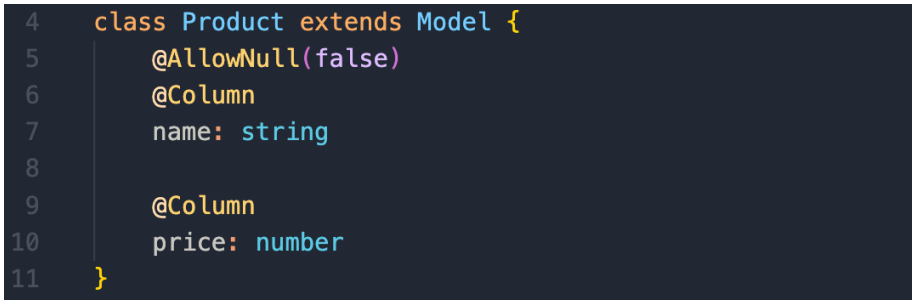
Далее рассмотрим, как реализованы модели данных в проекте, см. Рисунки 2-6.

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

    @AllowNull(false)
    @Column
    lastName: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance

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

Рисунок 2 – Модель пользователя

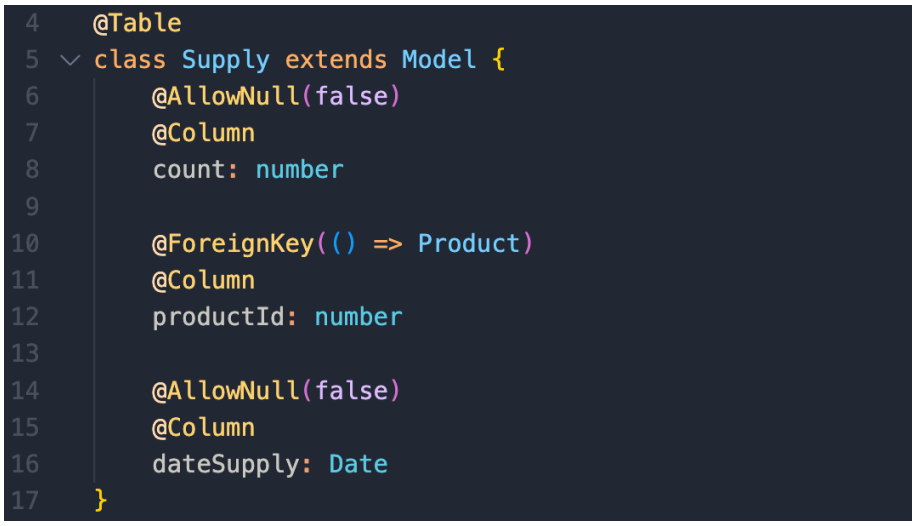
```
@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string

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

Рисунок 3 – Модель RefreshToken



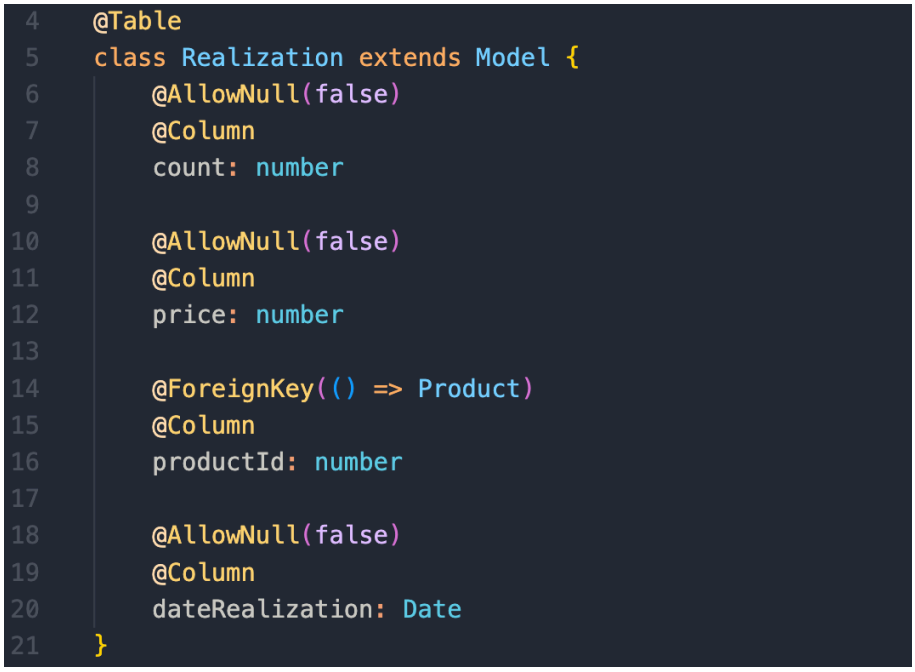
```
class Product extends Model {
    @AllowNull(false)
    @Column
    name: string

    @Column
    price: number
}
```

Рисунок 4 – Модель Product



```
@Table
class Supply extends Model {
    @AllowNull(false)
    @Column
    count: number

    @ForeignKey(() => Product)
    @Column
    productId: number

    @AllowNull(false)
    @Column
    dateSupply: Date
}
```

Рисунок 5 – Модель Supply



```
@Table
class Realization extends Model {
    @AllowNull(false)
    @Column
    count: number

    @AllowNull(false)
    @Column
    price: number

    @ForeignKey(() => Product)
    @Column
    productId: number

    @AllowNull(false)
    @Column
    dateRealization: Date
}
```

Рисунок 6 – Модель Realization

После этого необходимо было написать сервисы, как показано на Рисунках 7-9.

```
class ProductService {
    async getAll() : Promise<Product[]> {
        const products = await Product.findAll()

        return products
    }

    async getById(id: number) : Promise<Product> {
        const product = await Product.findByPk(id)

        if (product) {
          return product.toJSON()
        }

        throw new Error('Not found!')
    }

    async create(productData: Partial<ProductCreateRequest>): Promise<Product> {
        const product = await Product.create(productData)
        return product.toJSON()
    }

    async deleteById(id: number): Promise<boolean> {
        const deletedRows = await Product.destroy({ where: { id } })
        return deletedRows > 0;
    }

    async updateById(id: number, productData: Partial<ProductCreateRequest>): Promise<boolean> {
        const affectedCount = await Product.update(productData, { where: { id } })

        if (affectedCount[0] < 1) {
            throw new Error('Not found!')
        }
```

Рисунок 7 – Сервис для Product

```
class SupplyService {
    async getAll() : Promise<Supply[]> {
        const supplies = await Supply.findAll()

        return supplies
    }

    async getSuppliedAmount(params: SuppliedRequestParams): Promise<number> {
        const amount = await Supply.sum('count', {
            where: {
                productId: params.productId,
                dateSupply: {
                    [Op.gte]: params.dateFrom ?? new Date(0),
                    [Op.lte]: params.dateTo ?? new Date(),
                }
            }
        })

        return amount
    }

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

```
        if (supply) {
          return supply.toJSON()
        }

        throw new Error('Not found!')
    }

    async create(supplyData: Partial<SupplyCreateRequest>): Promise<Supply> {
        const supply = await Supply.create(supplyData)
        return supply.toJSON()
    }

    async deleteById(id: number): Promise<boolean> {
        const deletedRows = await Supply.destroy({ where: { id } })
        return deletedRows > 0;
    }

    async deleteByProductId(productId: number): Promise<boolean> {
        const deletedRows = await Supply.destroy({ where: { productId } })
        return deletedRows > 0;
    }

    async updateById(id: number, supplyData: Partial<SupplyCreateRequest>): Promise<boolean> {
        const affectedCount = await Supply.update(supplyData, { where: { id } })
        return affectedCount[0] > 0;
    }
}
```

Рисунок 8 – Сервис для Supply

```
class RealizationService {
    async getAll() : Promise<Realization[]> {
        const realizations = await Realization.findAll()

        return realizations
    }

    async getSoldRevenue(params: Partial<SoldRequestParams>): Promise<number> {
        let revenue;

        if (params.productId) {
            revenue = await Realization.sum('price', {
                where: {
                    productId: params.productId,
                    dateRealization: {
                        [Op.gte]: params.dateFrom ?? new Date(0),
                        [Op.lte]: params.dateTo ?? new Date(),
                    }
                }
            })
        } else {
            revenue = await Realization.sum('price', {
                where: {
                    dateRealization: {
                        [Op.gte]: params.dateFrom ?? new Date(0),
                        [Op.lte]: params.dateTo ?? new Date(),
                    }
                }
            })
        }
```

```
52      async getById(id: number) : Promise<Realization> {
53          const realization = await Realization.findByPk(id)
54
55          if (realization) {
56            return realization.toJSON()
57          }
58
59          throw new Error('Not found!')
60      }
61
62      async create(supplyData: Partial<RealizationCreateRequest>): Promise<Realization> {
63          const realization = await Realization.create(supplyData)
64          return realization.toJSON()
65      }
66
67      async deleteById(id: number): Promise<boolean> {
68          const deletedRows = await Realization.destroy({ where: { id } })
69          return deletedRows > 0;
70      }
71
72      async deleteByProductId(productId: number): Promise<boolean> {
73          const deletedRows = await Realization.destroy({ where: { productId } })
74          return deletedRows > 0;
75      }
76
77      async updateById(id: number, realizationData: Partial<RealizationCreateRequest>): Promise<boo
78          const affectedCount = await Realization.update(realizationData, { where: { id } })
79          return affectedCount[0] > 0;
80      }
```

Рисунок 9 – Сервис для Realization

На Рисунках 10-14 отображены роуты в нашем проекте.

```
1   import express from "express"
2   import userRoutes from "./users/User"
3   import productRoutes from "./products/Product"
4   import supplyRoutes from "./products/Supply"
5   import realizationRoutes from "./products/Realization"
6
7   const router: express.Router = express.Router()
8
9   router.use('/users', userRoutes)
10  router.use('/products', productRoutes)
11  router.use('/supplies', supplyRoutes)
12  router.use('/realizations', realizationRoutes)
13
14  export default router
```

Рисунок 10 – Общий роут

```
import express from "express"
import UserController from "../../../controllers/users/User"
import passport from "../../../middlewares/passport"

const router: express.Router = express.Router()

const controller: UserController = new UserController()

router.route('/')
    .post(controller.post)

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

router.route('/email/:email')
    .get(controller.getByEmail)

router.route('/all')
    .get(controller.getAll)

router.route('/login')
    .post(controller.auth)

router.route('/profile')
    .get(passport.authenticate('jwt', { session: false }), controller.me)

router.route('/refresh')
    .post(controller.refreshToken)


export default router
```

Рисунок 11 – Роут пользователя

```
import express from "express"
import ProductController from "../../../controllers/products/Product"

const router: express.Router = express.Router()

const controller: ProductController = new ProductController()

router.route('/')
    .get(controller.get)
    .post(controller.post)

router.route('/:id')
    .get(controller.get)
    .delete(controller.delete)

export default router
```

Рисунок 12 – Роут Product

```
import express from "express"
import SupplyController from "../../../controllers/products/Supply"

const router: express.Router = express.Router()

const controller: SupplyController = new SupplyController()

router.route('/amount')
    .get(controller.getSuppliedProductAmount)

router.route('/supply')
    .get(controller.get)
    .post(controller.post)

router.route('/supply/:id')
    .get(controller.get)
    .delete(controller.delete)

export default router
```

Рисунок 13 – Роут Supply

```
import express from "express"
import RealizationController from "../../../controllers/products/Realization"

const router: express.Router = express.Router()

const controller: RealizationController = new RealizationController()

router.route('/')
    .post(controller.addRealization)

router.route('/amount')
    .get(controller.getSoldAmount)

router.route('/revenue')
    .get(controller.getSoldRevenue)

router.route('/stat/amount')
    .get(controller.getAmountStat)

router.route('/stat/revenue')
    .get(controller.getRevenueStat)

router.route('/realization')
    .get(controller.get)
    .post(controller.post)

router.route('/realization/:id')
    .get(controller.get)
    .delete(controller.delete)
```

Рисунок 14 – Роут Realization

Вывод

В результате работы было успешно реализовано RESTful API с использованием express и typescript, используя предварительно написанный boilerplate. Цель работы заключалась в создании платформы для администратора интернет-магазина, включающей функциональность входа, регистрации, учета товара на складе, графиков по продажам товаров и общей выручке предприятия, а также управления сотрудниками.