САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 2

Выполнил:

Таякин Даниил

Группа К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

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

Выбран вариант сервиса для работы с магазином одежды. Требуемый функционал: регистрация, авторизация, создание профиля, работа с товарами, просмотр количества единиц товара, управление скидками и акциями, работа с базой клиентов.

Ход работы

1. Опишем модели, которые будут использованы в приложении для магазина одежды.

   Покупатель:

```
@Table({
    timestamps: true,
    paranoid: true,
})
export class Customer extends Model {
    @Unique
    @PrimaryKey
    @AutoIncrement
    @Column
    declare id: number

    @Column
    declare firstName: string

    @Column
    declare lastName: string

    @Column
    declare bio: string
}
```

   Продукты:

```
@Table({
    timestamps: true,
    paranoid: true,
})
export class Product extends Model {
    @Unique
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number

    @Column
    name: string

    @Column
    description: string

    @Column
    price: number

    @Column
    quantity: number

    @Column
    imageUrl: string
}
```

Скидки:

```
import { Product } from './Product.js'

@Table({
    timestamps: true,
    paranoid: true,
})
export class Sale extends Model {
    @Unique
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number

    @Column
    name: string

    @Column
    percent: number

    @Column
    startsAt: Date

    @Column
    endsAt: Date

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

    @BelongsTo(() => Product)
    product: Product
}
```

Пользователь:

```
@Table({
    timestamps: true,
    paranoid: true,
})
export class User extends Model {
    @Unique
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number

    @Column
    email: string

    @Column
    passwordHash: string

    @Column
    firstName: string

    @Column
    lastName: string

    @Default(false)
    @Column
    isAdmin: Boolean
}
```

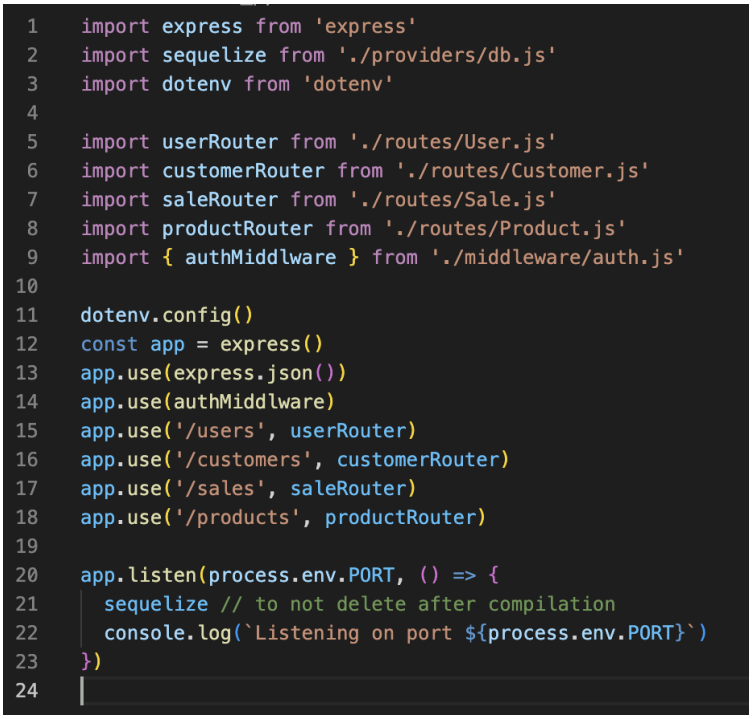
2. Для работы с базой данных создадим базовый класс сервиса, в котором будут определены функции, необходимые для получения, удаления и создания новой сущности.

```
export interface IService<T extends Model> {
    create(data: object): Promise<T>
    getById(id: number): Promise<T | null>
    updateById(id: number, data: object): Promise<[affectedCount: number]>
    deleteById(id: number): Promise<number>
}

export class BaseService<T extends Model> implements IService<T> {
    protected model: ModelCtor<T>

    constructor(model: ModelCtor<T>) {
        this.model = model
    }

    create = async (data: any): Promise<T> => {
        return (await this.model.create(data)) as T
    }

    getById = async (id: number): Promise<T | null> => {
        return (await this.model.findByPk(id)) as T | null
    }

    updateById = async (id: any, data: any): Promise<[affectedCount: number]> => {
        return await this.model.update(data, { where: { id: id } })
    }

    deleteById = async (id: any): Promise<number> => {
        return await this.model.destroy({ where: { id: id } })
    }
}
```

3. Создадим базовый класс контроллера, который будет хранить в себе сервис и выполнять CRUD операции при помощи него.

```ts
export class BaseController<T extends Model> {
    protected service: IService<T>

    get = async (req: Request, res: Response) => {
        try {
            const data = await this.service.getById(+req.params.pk)
            if (!data) {
                res.status(404).json({ error: 'Resource not found' })
                return
            }
            res.status(200).json(data)
        } catch (error) {
            console.error('Error:', error)
            res.status(500).json({ error: 'Internal Server Error' })
        }
    }

    post = async (req: Request, res: Response) => {
        try {
            res.status(201).send(await this.service.create(req.body))
        } catch (error) {
            console.error('Error:', error)
            res.status(500).json({ error: 'Internal Server Error' })
        }
    }

    put = async (req: Request, res: Response) => {
        try {
          const updatedData = await this.service.updateById(
            +req.params.pk,
            req.body
          )
          res.status(200).json(updatedData)
        } catch (error) {
          console.error('Error:', error)
          res.status(500).json({ error: 'Internal Server Error' })
        }
      }

    delete = async (req: Request, res: Response) => {
        try {
            const deletedCount = await this.service.deleteById(+req.params.pk)
            if (deletedCount === 0) {
                res.status(404).json({ error: 'Resource not found' })
                return
            }
            res.status(204).send()
        } catch (error) {
            console.error('Error:', error)
            res.status(500).json({ error: 'Internal Server Error' })
        }
    }
}
```

4. В src/index.ts файле импортируем все роутеры и подключаем их к необходимому префиксу.



```
import express from 'express'
import sequelize from './providers/db.js'
import dotenv from 'dotenv'

import userRouter from './routes/User.js'
import customerRouter from './routes/Customer.js'
import saleRouter from './routes/Sale.js'
import productRouter from './routes/Product.js'
import { authMiddlware } from './middleware/auth.js'

dotenv.config()
const app = express()
app.use(express.json())
app.use(authMiddlware)
app.use('/users', userRouter)
app.use('/customers', customerRouter)
app.use('/sales', saleRouter)
app.use('/products', productRouter)

app.listen(process.env.PORT, () => {
  sequelize // to not delete after compilation
  console.log(`Listening on port ${process.env.PORT}`)
})

```

Вывод

В данной лабораторной работе удалось создать сервис для магазина одежды на основе boilerplate, написанного в рамках первой лабораторной работы.