

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 2

Выполнил:

Трофимов Андрей

Группа К33402

Проверил
: Добряков Д.
И.

Санкт-Петербург, 2024 г

Задача

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

Сервис 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. Для начала опишем модели, которые будут присутствовать в нашем проекте.

а) У каждого пользователя будут поля для имени, фамилии, хеша пароля, почты и флаг, который отвечает за то, является ли пользователь админом

```
@Table
export class User extends Model {
  @Unique
  @PrimaryKey
  @AutoIncrement
  @Column
  declare id: number

  @Column
  declare firstName: string

  @Column
  declare lastName: string

  @Column
  declare passwordHash: string

  @Column
  declare email: string

  @Default(false)
  @Column
  declare isAdmin: boolean
}
```

б) Далее опишем модель товара, у которого будет название, количество, цена и ссылка на фотографию для показа

```
@Table
export class Product extends Model {
  @Unique
  @PrimaryKey
  @AutoIncrement
  @Column
  declare id: number

  @Column
  declare name: string

  @Column
  declare quantity: number

  @Column
  declare price: number

  @Column
  declare imageUrl: string
}
```

в) Более того, пользователям будут доступны скидки. Для этого создадим модель, которая будет хранить в себе название скидки, дату начала и окончания акции, процент скидки и связь с продуктом

```
@Table
export class Sale extends Model {
  @Unique
  @PrimaryKey
  @AutoIncrement
  @Column
  declare id: number

  @Column
  declare title: string

  @Column
  declare startsAt: Date

  @Column
  declare percentage: number

  @Column
  declare endsAt: Date | null

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

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

д) Для работы с базой клиентов создадим сущность покупателя, которая будет включать себя его имя, фамилию и дополнительную информацию

```
@Table
export class Customer extends Model {
  @Unique
  @PrimaryKey
  @AutoIncrement
  @Column
  declare id: number

  @Column
  declare firstName: string

  @Column
  declare lastName: string

  @Column
  declare info: string
}
```

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

```
import { Model, ModelCtor } from 'sequelize-typescript'

export interface IService<T extends Model> {
  create(data: object): Promise<T>
  findByPk(pk: number): Promise<T | null>
  updateByPk(pk: number, data: object): Promise<[affectedCount: number]>
  deleteByPk(pk: 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
  }

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

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

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

На примере сервиса для получения продуктов покажем, как данный класс может расширяться. В данном случае мы определили функцию для получения списка всех продуктов, а также в функции `findByPk` добавили параметр `include`, который позволяет автоматически получать продукт по полю `foreignKey`

```
import { BaseService } from '../base/index.js'
import { Sale } from '../../models/sale.js'
import { Product } from '../../models/product.js'

export class SalesService extends BaseService<Sale> {
  findByPk = async (pk: number): Promise<Sale | null> => {
    return (await this.model.findByPk(pk, { include: Product })) as Sale | null
  }
  list = async (): Promise<Sale[]> => {
    return await this.model.findAll({ include: Product })
  }
}
```


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

```

import { IService } from '../services/base/index.js'
import { Model } from 'sequelize-typescript'
import { Request, Response } from 'express'

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

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

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

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

```

На примере контроллера для продуктов покажем, как базовый класс может быть расширен. В данном примере видно, что в конструкторе класса происходит инициализация сервиса и в класс добавлен новый метод для получения списка всех продуктов.

```
import { Request, Response } from 'express'
import { Product } from '../models/product.js'
import { ProductsService } from '../services/products/index.js'
import { BaseController } from '../base/index.js'

export class ProductsController extends BaseController<Product> {
  protected service: ProductsService

  constructor() {
    super()
    this.service = new ProductsService(Product)
  }

  list = async (req: Request, res: Response) => {
    try {
      res.status(200).send(await this.service.list())
    } catch (error) {
      console.error(error)
      res.status(500).send('Internal Server Error')
    }
  }
}
```

4. Для того, чтобы данный контроллер начал обрабатывать запросы, создадим `express.Router` и укажем методы для обработки.

```
const router = Router()
const controller = new CustomersController()

router.get('/:pk', controller.get)
router.post('/', controller.post)
router.put('/:pk', controller.put)
router.delete('/:pk', controller.get)

export default router
```

5. В файле `src/index.ts` импортируем все роутеры и подключаем их к необходимому префиксу. Данный вариант удобен тем, что при открытии данного файла мы сразу можем увидеть все пути, которые определены в приложении

```
dotenv.config();
const app = express();
app.use(express.json());
app.use(authMiddleware);
app.use('/products', productsRouter);
app.use('/sales', salesRouter);
app.use('/customers', customersRouter);
app.listen(process.env.PORT, () => {
  sequelize;
  console.log(`Listening on port ${process.env.PORT}`);
});
```

Вывод

В данной лабораторной работе удалось создать сервис для интернет-магазина на основе boilerplate, написанного в рамках прошлой лабораторной работы.