

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа 2

Выполнил:

Рыбалко Олег

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).

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

```
11  @Table
12  export class User extends Model {
13      @Unique
14      @PrimaryKey
15      @AutoIncrement
16      @Column
17      declare id: number
18
19      @Column
20      declare firstName: string
21
22      @Column
23      declare lastName: string
24
25      @Column
26      declare passwordHash: string
27
28      @Column
29      declare email: string
30
31      @Default(false)
32      @Column
33      declare isAdmin: boolean
34  }
```

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

```
1 import { Model, ModelCtor } from 'sequelize-typescript'
2
3 export interface IService<T extends Model> {
4   create(data: object): Promise<T>
5   findByPk(pk: number): Promise<T | null>
6   updateByPk(pk: number, data: object): Promise<[affectedCount: number]>
7   deleteByPk(pk: number): Promise<number>
8 }
9
10 export class BaseService<T extends Model> implements IService<T> {
11   protected model: ModelCtor<T>
12
13   constructor(model: ModelCtor<T>) {
14     this.model = model
15   }
16
17   create = async (data: any): Promise<T> => {
18     return (await this.model.create(data)) as T
19   }
20
21   findByPk = async (pk: number): Promise<T | null> => {
22     return (await this.model.findByPk(pk)) as T | null
23   }
24
25   updateByPk = async (pk: any, data: any): Promise<[affectedCount: number]> => {
26     return await this.model.update(data, { where: { id: pk } })
27   }
28
29   deleteByPk = async (pk: any): Promise<number> => {
30     return await this.model.destroy({ where: { id: pk } })
31   }
32 }
```

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

```
5 export class BaseController<T extends Model> {
6   protected service: IService<T>
7
8   get = async (req: Request, res: Response) => {
9     try {
10      const data = await this.service.findByPk(+req.params.pk)
11      if (!data) {
12        res.status(404).json({ error: 'Resource not found' })
13        return
14      }
15      res.status(200).json(data)
16    } catch (error) {
17      console.error('Error:', error)
18      res.status(500).json({ error: 'Internal Server Error' })
19    }
20  }
21
22  post = async (req: Request, res: Response) => {
23    try {
24      res.status(201).send(await this.service.create(req.body))
25    } catch (error) {
26      console.error('Error:', error)
27      res.status(500).json({ error: 'Internal Server Error' })
28    }
29  }
30
31  put = async (req: Request, res: Response) => {
32    try {
33      const updatedData = await this.service.updateByPk(
34        +req.params.pk,
35        req.body
36      )
37      res.status(200).json(updatedData)
38    } catch (error) {
39      console.error('Error:', error)
40      res.status(500).json({ error: 'Internal Server Error' })
41    }
42  }
43
44  delete = async (req: Request, res: Response) => {
45    try {
46      const deletedCount = await this.service.deleteByPk(+req.params.pk)
47      if (deletedCount === 0) {
48        res.status(404).json({ error: 'Resource not found' })
49        return
50      }
51      res.status(204).send()
52    } catch (error) {
53      console.error('Error:', error)
54      res.status(500).json({ error: 'Internal Server Error' })
55    }
56  }
57 }
```

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.

```
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.

```
4   const router = Router()
5   const controller = new ProductsController()
6
7   router.get('/:pk', controller.get)
8   router.get('/', controller.list)
9   router.post('/', controller.post)
10  router.put('/:pk', controller.put)
11  router.delete('/:pk', controller.get)
12
13  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

```
9  dotenv.config()
10 const app = express()
11 app.use(express.json())
12 app.use('/users', usersRouter)
13 app.use('/products', productsRouter)
14 app.use('/sales', salesRouter)
15 app.use('/customers', customersRouter)
16
17 app.listen(process.env.PORT, () => {
18   // sequelize // to not delete after compilation
19   console.log(`Listening on port ${process.env.PORT}`)
20 })
21
```

## **Вывод**

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