САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №2: REST, RESTful, SOAP, GraphQL

Выполнил:

Жигалова Анастасия
K33392

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) Сначала нужно было создать модели

Модель скидки

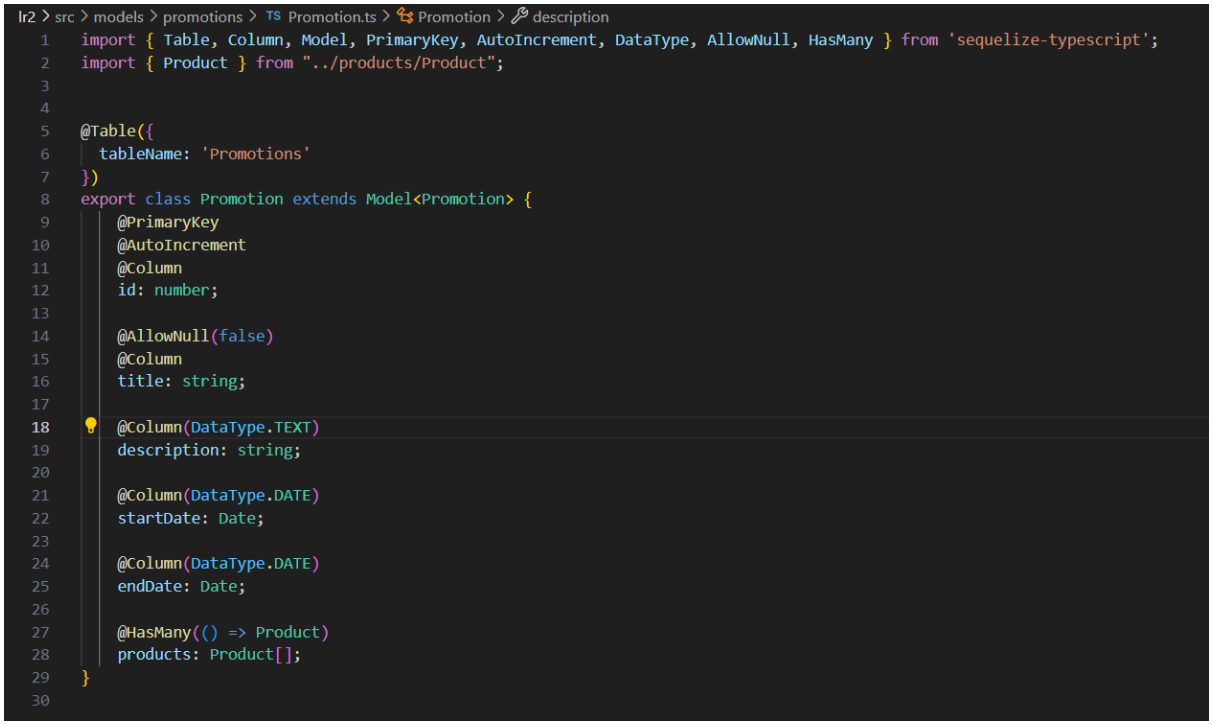
```
import { Table, Column, Model, PrimaryKey, AutoIncrement, DataType, AllowNull, HasMany } from 'sequelize-typescript';
import { Product } from "../products/Product";


@Table({
  tableName: 'Discounts'
})
export class Discount extends Model<Discount> {
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number;

    @AllowNull(false)
    @Column
    description: string;

    @AllowNull(false)
    @Column
    percentage: number;

    @Column(DataType.DATE)
    startDate: Date;

    @Column(DataType.DATE)
    endDate: Date;

    @HasMany(() => Product)
    products: Product[];
}
```

Модель пользователя (с прошлой лр)

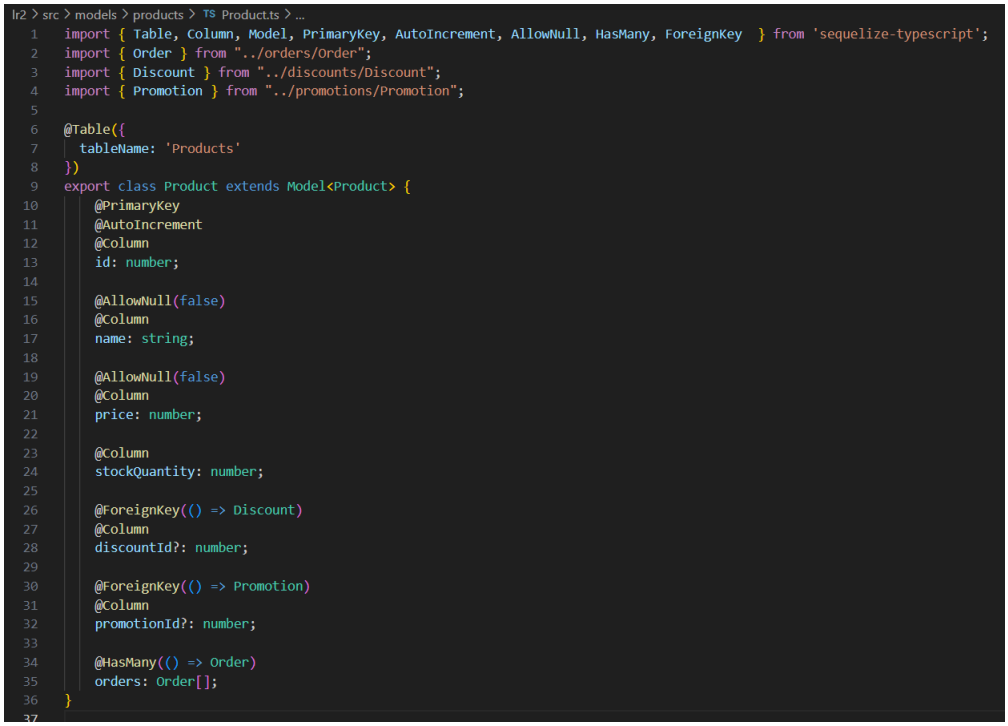
```
import { Table, Column, Model, Unique, AllowNull, HasMany } from 'sequelize-typescript';
import { Order } from "../orders/Order";

@Table({
  tableName: 'Users'
})
export class User extends Model<User> {
    @Column
    name: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

    @HasMany(() => Order)
    orders: Order[];
}
```

Добавила заказы

Модель акций



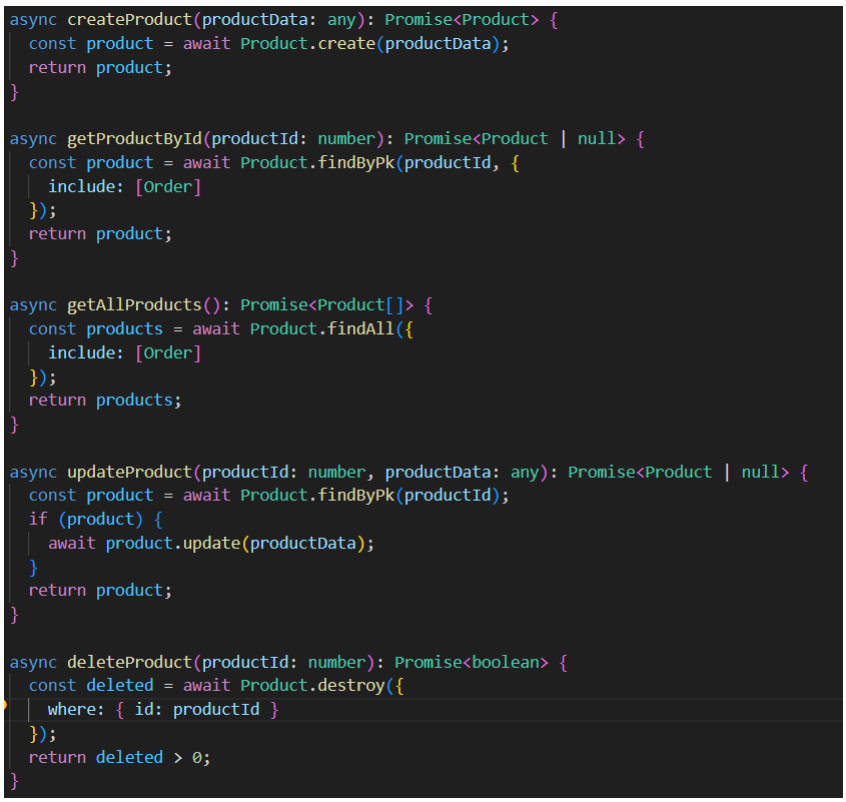
Модель товара



Модель заказа

```
import { Table, Column, Model, PrimaryKey, AutoIncrement, ForeignKey, DataType, AllowNull } from 'sequelize-typescript';
import { User } from "../users/User";
import { Product } from "../products/Product";

@Table({
  tableName: 'Orders'
})
export class Order extends Model<Order> {
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number;

    @ForeignKey(() => User)
    @AllowNull(false)
    @Column
    userId: number;

    @ForeignKey(() => Product)
    @AllowNull(false)
    @Column
    productId: number;

    @AllowNull(false)
    @Column
    quantity: number;

    @Column(DataType.DATE)
    orderDate: Date;
}
```

2) Следующий шаг – создание сервисов для всех моделей, где пропишем методы для регистрации, аутентификации (для пользователей), получение товаров, получение определенных товаров, изменение, удаление, редактирование товаров и т.д



```
async createProduct(productData: any): Promise<Product> {
  const product = await Product.create(productData);
  return product;
}

async getProductById(productId: number): Promise<Product | null> {
  const product = await Product.findByPk(productId, {
    include: [Order]
  });
  return product;
}

async getAllProducts(): Promise<Product[]> {
  const products = await Product.findAll({
    include: [Order]
  });
  return products;
}

async updateProduct(productId: number, productData: any): Promise<Product | null> {
  const product = await Product.findByPk(productId);
  if (product) {
    await product.update(productData);
  }
  return product;
}

async deleteProduct(productId: number): Promise<boolean> {
  const deleted = await Product.destroy({
    where: { id: productId }
  });
  return deleted > 0;
}
```

```
export class UserService {
  async register(email: string, password: string) {
    const hashedPassword = await bcrypt.hash(password, 8);
    const user = await User.create({ email, password: hashedPassword });
    return user;
  }

  async login(email: string, password: string) {
    const user = await User.findOne({ where: { email } });
    if (!user) {
      throw new Error('User not found');
    }

    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      throw new Error('Invalid password');
    }

    const token = jwt.sign({ id: user.id }, secretKey);
    return token;
  }
```

3) Создаем контроллеры для всех моделей. Пример – контроллеры для заказов и акций.

```
export class OrdersController {
    private orderService: OrderService;

    constructor() {
        this.orderService = new OrderService();
    }

    async createOrder(req: Request, res: Response) {
        try {
            const { userId, productId, quantity } = req.body;
            const order = await this.orderService.createOrder(userId, productId, quantity);
            if (order) {
                res.status(201).json(order);
            } else {
                res.status(400).json({ message: 'Unable to create order. Product may be out of stock or not found.' });
            }
        } catch (error) {
            if ( error instanceof Error){
                res.status(500).json({ message: error.message });
            }
        }
    }

    async getOrderById(req: Request, res: Response) {
        try {
            const { orderId } = req.params;
            const order = await this.orderService.getOrderById(Number(orderId));
            if (order) {
                res.status(200).json(order);
            } else {
                res.status(404).json({ message: 'Order not found' });
            }
        } catch (error) {
            if ( error instanceof Error){
                res.status(500).json({ message: error.message });
            }
        }
```

```
class PromotionController {
    async createPromotion(req: Request, res: Response) {
        try {
            const { title, description, startDate, endDate } = req.body;
            const promotion = await PromotionService.createPromotion(title, description, new Date(startDate), new Date(endDate));
            res.status(201).json(promotion);
        } catch (error) {
            if ( error instanceof Error){
                res.status(500).json({ message: error.message });
            }
        }
    }

    async addProductToPromotion(req: Request, res: Response) {
        try {
            const { promotionId, productId } = req.params;
            await PromotionService.addProductToPromotion(Number(promotionId), Number(productId));
            res.status(200).json({ message: 'Product added to promotion successfully' });
        } catch (error) {
            if ( error instanceof Error){
                res.status(500).json({ message: error.message });
            }
        }
    }

    async removeProductFromPromotion(req: Request, res: Response) {
        try {
            const { productId } = req.params;
            await PromotionService.removeProductFromPromotion(Number(productId));
            res.status(200).json({ message: 'Product removed from promotion successfully' });
        } catch (error) {
            if ( error instanceof Error){
                res.status(500).json({ message: error.message });
            }
```

4) Создаем роуты

```
const router = express.Router();

const userController = new UserController();

router.post('/register', (req, res) => {
  userController.register(req, res);
});

router.post('/login', (req, res) => {
  userController.login(req, res);
});

router.get('/users', (req, res) => {
    userController.getAllUsersWithOrders(req, res);
});

export default router;
```

```
const router = express.Router();

const promotionController = new PromotionController();

router.post('/', (req, res) => {
  promotionController.createPromotion(req, res);
});

router.post('/:promotionId/products/:productId', (req, res) => {
  promotionController.addProductToPromotion(req, res);
});

router.delete('/products/:productId', (req, res) => {
  promotionController.removeProductFromPromotion(req, res);
});

router.get('/:promotionId/products', (req, res) => {
  promotionController.getProductsByPromotion(req, res);
});

router.get('/:promotionId', (req, res) => {
  promotionController.getPromotionById(req, res);
});
```

5) В главном файле index.ts прописываем пути к каждой модели (роутам)

```
dotenv.config();

const app = express();
const PORT = process.env.PORT || 8080;

app.use(express.json());

app.use('/discounts', authenticateToken, discountRoute);
app.use('/orders', authenticateToken, ordersRoutes);
app.use('/products', productsRoutes);
app.use('/promotions', authenticateToken, promotionRoutes);
app.use('/users', userRoute);


app.listen(PORT, () => {
  Sequelize
  console.log(`Server is running on port ${PORT}`);
});
```

6) Спрячем данные о бд

```
const sequelize = new Sequelize({
  database: process.env.DB_NAME || 'some_db',
  dialect: process.env.DB_DIALECT as 'mysql' | 'postgres' | 'sqlite' | 'mariadb' | 'mssql' || 'sqlite',
  username: process.env.DB_USERNAME || 'root',
  password: process.env.DB_PASSWORD || '',
  storage: process.env.DB_STORAGE || 'db.sqlite',
  logging: console.log,
  })


  const models = [User, Discount, Order, Product, Promotion]


  sequelize.addModels(models)
```

config.ts

lr2 > src > config > TS config.ts > default

```
import dotenv from 'dotenv';
dotenv.config();

export default {
  port: process.env.PORT || 8080,
  db: {
    database: process.env.DB_NAME || 'some_db',
    dialect: process.env.DB_DIALECT || 'sqlite',
    username: process.env.DB_USERNAME || 'root',
    password: process.env.DB_PASSWORD || '',
    storage: process.env.DB_STORAGE || 'db.sqlite',
  },
  secretKey: process.env.SECRET_KEY,
};

```

.env

lr2 > .env

```
SECRET_KEY=secret_key
PORT=8080
DB_NAME=some_db
DB_DIALECT=sqlite
DB_USERNAME=root
DB_PASSWORD=
DB_STORAGE=db.sqlite

```

7) Создадим middleware/ authenticateToken.ts, где сделаем аутентификацию по токену

```
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';
import process from 'process';
import { UserController } from '../controllers/users/userController';

dotenv.config();

const secretKey = process.env.SECRET_KEY;

const authenticateToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];
if (authHeader === undefined) return res.sendStatus(401);
if (!authHeader.startsWith('Bearer')) return res.sendStatus(401);

    const token = authHeader.split(' ')[1];



    jwt.verify(token, process.env.SECRET_KEY, (err, user) => {
      if (err) {
        console.error(err);
        return res.status(403).send("Access Denied: Invalid Token");
      }
      req.user = user;
      next();
    });

```

Вывод

В лабораторной работе был реализован RESTful API средствами express + typescript (используя ранее написанный boilerplate). Вариант – магазин одежды.