САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнила:

Олейникова Полина

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

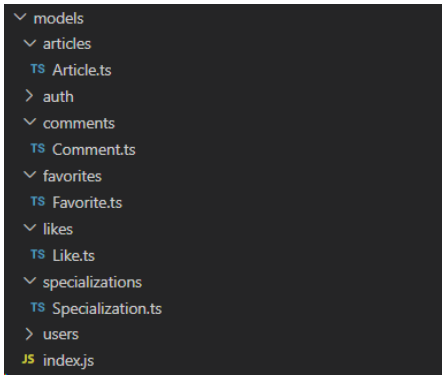
Задача

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

Вариант: сервис для публикации статей с разделением прав.

Ход работы

Создаем новые модели:



На примере Статьи:

Article.ts

```ts
import { Table, Column, ForeignKey, Model, PrimaryKey, DataType, AutoIncrement, HasMany, BelongsTo } from 'sequelize-typescript'
import User from '../users/User'
import Comment from '../comments/Comment'
import Like from '../likes/Like'
import Specialization from '../specializations/Specialization'
import Favorite from '../favorites/Favorite';

export enum StatusType {
    ACCEPTED = "ACCEPTED",
    REJECTED = "REJECTED",
    NOT_CONSIDER = "NOT_CONSIDER",
};

export enum LevelType {
    SIMPLE = "SIMPLE",
    MEDIUM = "MEDIUM",
    HARD = "HARD",
};

export type ArticleAttributes = ArticleAttributesOptionally & ArticleCreationAttributes

export type ArticleCreationAttributes = {
    title: string,
    content: string;
    tags: string;
    level: LevelType;
    status?: StatusType;
    userId: number;
    specializationId: number;
};

type ArticleAttributesOptionally = {
    id: number;
    specialization: Specialization;
    user: User;
    comments: Comment[];
    likes: Like[];
    favorites: Favorite[];
};

export type ArticleUpdateAttributes = Pick<ArticleCreationAttributes, 'content' | 'title' | 'tags' | 'level' | 'specializationId'>;
```

```ts
@Table
export class Article extends Model<ArticleAttributes, ArticleCreationAttributes> {
    @PrimaryKey
    @AutoIncrement
    @Column
    id: number;

    @Column
    title: string;

    @Column
    content: string

    @Column
    tags: string

    @Column({
        type: DataType.ENUM(...Object.values(LevelType)),
    })
    level: LevelType

    @Column({
        type: DataType.ENUM(...Object.values(StatusType)),
        defaultValue: StatusType.NOT_CONSIDER,
    })
    status!: StatusType;

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

    @ForeignKey(() => Specialization)
    @Column
    specializationId: number

    @BelongsTo(() => Specialization)
    specialization: Specialization;

    @BelongsTo(() => User)
    user: User;

    @HasMany(() => Comment)
    comments: Comment[];

    @HasMany(() => Like)
    likes: Like[];
```

Создаем сервисы:

На примере Статьи:

ArticleService.ts

```ts
import { Op } from 'sequelize';
import Article, { ArticleCreationAttributes, ArticleUpdateAttributes, LevelType, StatusType } from '../../models/articles/Article'
import User, { RoleType, UserAttributes } from '../../models/users/User';
import Specialization from '../../models/specializations/Specialization';
import Favorite from '../../models/favorites/Favorite';
import Like from '../../models/likes/Like';
import Comment from '../../models/comments/Comment';

type FilterArticles = {
    level?: LevelType,
    specializationId?: string;
    title?: string,
    offset?: number,
    limit?: number
}

class ArticleService {
    async getById(id: number): Promise<Article> {
        try {
            const article = await Article.findByPk(id, { include: [Comment, Specialization, User.scope('withoutPassword'), Like, Favorite] })
            if (!article) {
                throw new Error('Not Found');
            }
            return article
        }
        catch (error) {
            throw error;
        }
    }

    async getAll(filters: FilterArticles): Promise<any> {
        try {
            let whereCondition: any = {};

            if (filters.title) {
                whereCondition.title = { [Op.like]: `%${filters.title}%` };
            }
            if (filters.level) {
                whereCondition.level = filters.level
            }
            if (filters.specializationId) {
                whereCondition.specializationId = filters.specializationId
            }
```

...

```
async create(user: UserAttributes, articleData: Omit<ArticleCreationAttributes, 'userId'>): Promise<Article> {
    try {
        const article = await Article.create({ ...articleData, userId: user.id })
        return article
    } catch (error) {
        throw error;
    }
}

async update(user: UserAttributes, articleId: string, articleData: ArticleUpdateAttributes): Promise<Article> {
    try {
        let whereCondition = {}
        if (user.role === RoleType.USER) {
            whereCondition = {
                userId: user.id, status: StatusType.NOT_CONSIDER
            }
        }
        if (user.role === RoleType.MODERATOR) {
            whereCondition = {
                userId: user.id
            }
        }
        const [updatedRowsCount, updatedArticle] = await Article.update(
            {
                title: articleData.content,
                content: articleData.content,
                tags: articleData.tags,
                level: articleData.level,
                specializationId: articleData.specializationId,
            },
            {
                where: { id: articleId, ...whereCondition },
                returning: true,
            });

        if (updatedRowsCount === 0) {
            throw new Error('Article not found');
        }
        return updatedArticle[0];
    } catch (error) {
        throw error;
    }
}
```

```
async updateStatus(user: UserAttributes, articleId: string, articleData: { status: StatusType }): Promise<Article> {
    try {
        if (user.role === 'USER') {
            throw new Error('Not enough rights');
        }
        const [updatedRowsCount, updatedArticle] = await Article.update(
            { status: articleData.status },
            {
                where: { id: articleId, status: StatusType.NOT_CONSIDER },
                returning: true,
            });

        if (updatedRowsCount === 0) {
            throw new Error('Article not found');
        }
        return updatedArticle[0];
    } catch (error) {
        throw error;
    }
}

async delete(user: UserAttributes, articleId: string,): Promise<number> {
    try {
        let whereCondition = {}
        if (user.role !== RoleType.ADMIN) {
            whereCondition = {
                userId: user.id
            }
        }
        const deletedRowsCount = await Article.destroy({ where: { id: articleId, ...whereCondition } });
        if (deletedRowsCount === 0) {
            throw new Error('Article not found');
        }
        return deletedRowsCount;
    } catch (error) {
        throw error;
    }
}
```

Создаем контроллеры:

На примере Статьи:

ArticleController.ts

```ts
import ArticleService from '../../services/articles/ArticleService'

export default class ArticleController {
    private articleService: ArticleService

    constructor() {
        this.articleService = new ArticleService()
    }

    get = async (request: any, response: any) => {
        try {
            const article = await this.articleService.getById(request.params.id)
            response.status(200).send(article)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const articles = await this.articleService.getAll(request.query)
            response.status(200).send(articles)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }

    create = async (request: any, response: any) => {
        try {
            const article = await this.articleService.create(request.user, request.body)
            response.status(200).send(article)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }

    update = async (request: any, response: any) => {
        try {
            const article = await this.articleService.update(request.user, request.params.id, request.body)
            response.status(200).send(article)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }
```

Создаем маршруты:

index.ts

```ts
import express from "express"
import userRoutes from "./users/UserRoutes"
import authRoutes from "./auth/AuthRouters"
import ArticleRoutes from "./articles/ArticleRoutes"
import CommentRoutes from "./comments/CommentRoutes"
import FavoriteRoutes from "./favorites/FavoriteRoutes"
import SpecializationRoutes from "./specializations/SpecializationRoutes"
import { auth } from "../../middlewares/auth"

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

router.use('/users', auth, userRoutes)
router.use('/articles', auth, ArticleRoutes)
router.use('/comments', auth, CommentRoutes)
router.use('/favorites', auth, FavoriteRoutes)
router.use('/specializations', auth, SpecializationRoutes)
router.use('/', authRoutes)

export default router
```

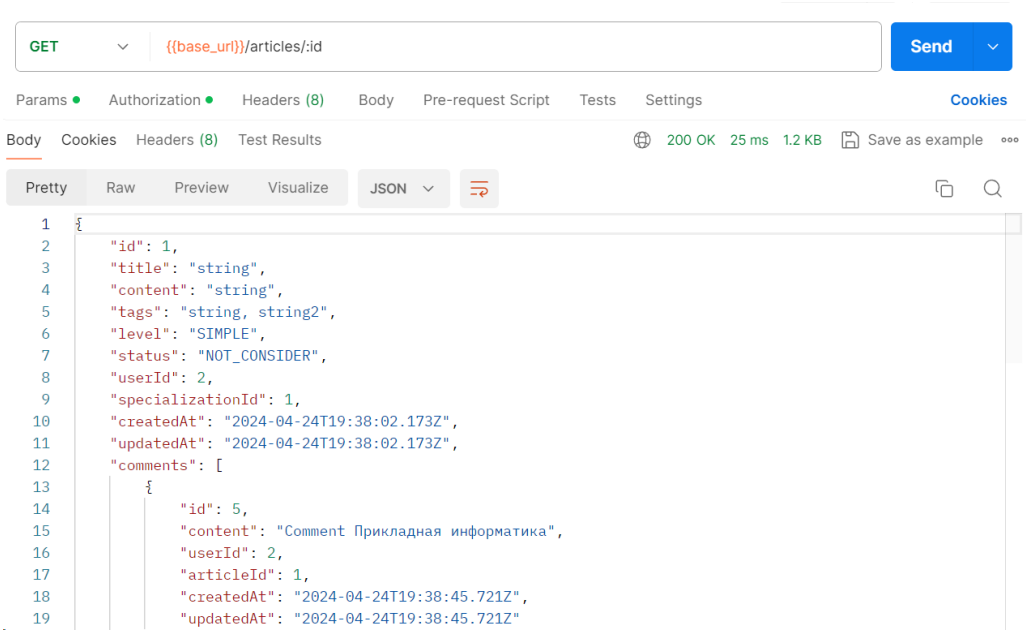
На примере Статьи:

TS ArticleRoutes.ts

```ts
import express from "express"
import ArticleController from "../../../controllers/articles/ArticleController"
import CommentController from "../../../controllers/comments/CommentController"
import LikeController from "../../../controllers/likes/LikeController"
import FavoriteController from "../../../controllers/favorites/FavoriteController"


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

const controller: ArticleController = new ArticleController()
const commentController: CommentController = new CommentController()
const likeController: LikeController = new LikeController()
const favoriteController: FavoriteController = new FavoriteController()


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

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

```

Вид в БД:

Пример запроса:



Вывод

В ходе данной работы был написан свой RESTful API средствами express + typescript (используя ранее написанный boilerplate).