

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнила:

Олейникова Полина

Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

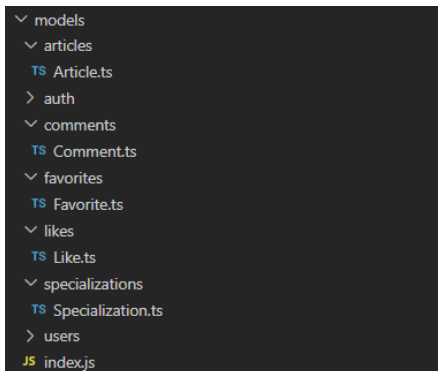
Задача

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

Вариант: сервис для публикации статей с разделением прав.

Ход работы

Создаем новые модели:



На примере Статьи:

```
TS Article.ts X
1 import { Table, Column, ForeignKey, Model, PrimaryKey, DataType, AutoIncrement, HasMany, BelongsTo } from 'sequelize-typescript'
2 import User from '../users/User'
3 import Comment from '../comments/Comment'
4 import Like from '../likes/Like'
5 import Specialization from '../specializations/Specialization'
6 import Favorite from '../favorites/Favorite';
7
8 export enum StatusType {
9   ACCEPTED = "ACCEPTED",
10  REJECTED = "REJECTED",
11  NOT_CONSIDER = "NOT_CONSIDER",
12 };
13
14 export enum LevelType {
15   SIMPLE = "SIMPLE",
16   MEDIUM = "MEDIUM",
17   HARD = "HARD",
18 };
19
20 export type ArticleAttributes = ArticleAttributesOptionally & ArticleCreationAttributes
21
22 export type ArticleCreationAttributes = {
23   title: string;
24   content: string;
25   tags: string;
26   level: LevelType;
27   status?: StatusType;
28   userId: number;
29   specializationId: number;
30 };
31
32 type ArticleAttributesOptionally = {
33   id: number;
34   specialization: Specialization;
35   user: User;
36   comments: Comment[];
37   likes: Like[];
38   favorites: Favorite[];
39 };
40
41 export type ArticleUpdateAttributes = Pick<ArticleCreationAttributes, 'content' | 'title' | 'tags' | 'level' | 'specializationId'>;
42
```

```

@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:

```

18 ArticleService X
1  import { Op } from 'sequelize';
2  import Article, { ArticleCreationAttributes, ArticleUpdateAttributes, LevelType, StatusType } from '../models/articles/Article';
3  import User, { RoleType, UserAttributes } from '../models/users/User';
4  import Specialization from '../models/specializations/Specialization';
5  import Favorite from '../models/favorites/Favorite';
6  import Like from '../models/likes/Like';
7  import Comment from '../models/comments/Comment';
8
9  type FilterArticles = {
10    level?: LevelType;
11    specializationId?: string;
12    title?: string;
13    offset?: number;
14    limit?: number;
15  }
16
17  class ArticleService {
18    async getById(id: number): Promise<Article> {
19      try {
20        const article = await Article.findByPk(id, { include: [Comment, Specialization, User.scope('withoutPassword'), Like, Favorite] });
21        if (!article) {
22          throw new Error('Not Found');
23        }
24        return article;
25      } catch (error) {
26        throw error;
27      }
28    }
29
30    async getAll(filters: FilterArticles): Promise<any> {
31      try {
32        let whereCondition: any = {};
33
34        if (filters.title) {
35          whereCondition.title = { [Op.like]: `%${filters.title}%` };
36        }
37        if (filters.level) {
38          whereCondition.level = filters.level;
39        }
40        if (filters.specializationId) {
41          whereCondition.specializationId = filters.specializationId;
42        }
43      }
44    }

```

...

```

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 [updatedRowCount, 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 (updatedRowCount === 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 [updatedRowCount, updatedArticle] = await Article.update(
      { status: articleData.status },
      {
        where: { id: articleId, status: StatusType.NOT_CONSIDER },
        returning: true,
      });
    if (updatedRowCount === 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 deletedRowCount = await Article.destroy({ where: { id: articleId, ...whereCondition } });
    if (deletedRowCount === 0) {
      throw new Error('Article not found');
    }
    return deletedRowCount;
  } catch (error) {
    throw error;
  }
}

```

Создаем контроллеры:

На примере Статьи:

```
TS ArticleControllers X
1 import ArticleService from '../../services/articles/ArticleService'
2
3 export default class ArticleController {
4   private articleService: ArticleService
5
6   constructor() {
7     this.articleService = new ArticleService()
8   }
9
10  get = async (request: any, response: any) => {
11    try {
12      const article = await this.articleService.getById(request.params.id)
13      response.status(200).send(article)
14    } catch (error: any) {
15      response.status(400).send({ "error": error.toString() })
16    }
17  }
18
19  getAll = async (request: any, response: any) => {
20    try {
21      const articles = await this.articleService.getAll(request.query)
22      response.status(200).send(articles)
23    } catch (error: any) {
24      response.status(400).send({ "error": error.toString() })
25    }
26  }
27
28  create = async (request: any, response: any) => {
29    try {
30      const article = await this.articleService.create(request.user, request.body)
31      response.status(200).send(article)
32    } catch (error: any) {
33      response.status(400).send({ "error": error.toString() })
34    }
35  }
36
37  update = async (request: any, response: any) => {
38    try {
39      const article = await this.articleService.update(request.user, request.params.id, request.body)
40      response.status(200).send(article)
41    } catch (error: any) {
42      response.status(400).send({ "error": error.toString() })
43    }
44  }
45}
```

Создаем маршруты:

```
TS index.ts X
1 import express from "express"
2 import userRoutes from "./users/UserRoutes"
3 import authRoutes from "./auth/AuthRoutes"
4 import ArticleRoutes from "./articles/ArticleRoutes"
5 import CommentRoutes from "./comments/CommentRoutes"
6 import FavoriteRoutes from "./favorites/FavoriteRoutes"
7 import SpecializationRoutes from "./specializations/SpecializationRoutes"
8 import { auth } from "../../middlewares/auth"
9
10 const router: express.Router = express.Router()
11
12 router.use('/users', auth, userRoutes)
13 router.use('/articles', auth, ArticleRoutes)
14 router.use('/comments', auth, CommentRoutes)
15 router.use('/favorites', auth, FavoriteRoutes)
16 router.use('/specializations', auth, SpecializationRoutes)
17 router.use('/', authRoutes)
18
19 export default router
```

На примере Статьи:

```
TS ArticleRoutes.ts X
1  import express from "express"
2  import ArticleController from "../../controllers/articles/ArticleController"
3  import CommentController from "../../controllers/comments/CommentController"
4  import LikeController from "../../controllers/likes/LikeController"
5  import FavoriteController from "../../controllers/favorites/FavoriteController"
6
7
8  const router: express.Router = express.Router()
9
10 const controller: ArticleController = new ArticleController()
11 const commentController: CommentController = new CommentController()
12 const likeController: LikeController = new LikeController()
13 const favoriteController: FavoriteController = new FavoriteController()
14
15
16 router.route('/')
17   .get(controller.getAll)
18   .post(controller.create)
19
20 router.route('/:id')
21   .get(controller.get)
22   .patch(controller.update)
23   .delete(controller.delete)
24
```

Вид в БД:

Tables (8)

Articles

Comments

Favorites

Likes

RefreshTokens

SequelizeMeta

Specializations

Users

Data OutputMessagesNotifications

	id	title	content	tags	level	status	userid	specializationid	createdAt
	[PK] integer	character varying (255)	character varying (255)	character varying (255)	'enum_Articles_level'	'enum_Articles_status'	integer	integer	timestamp with time zone
1	1	string	string	string, string2	SIMPLE	NOT_CONSIDER	2	1	2024-04-24 22:38:02.173+01

Пример запроса:

```
GET {{base_url}}/articles/id Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
Body Cookies Headers (8) Test Results 200 OK 25 ms 1.2 KB Save as example
Pretty Raw Preview Visualize JSON
1 {
2   "id": 1,
3   "title": "string",
4   "content": "string",
5   "tags": "string, string2",
6   "level": "SIMPLE",
7   "status": "NOT_CONSIDER",
8   "userId": 2,
9   "specializationId": 1,
10  "createdAt": "2024-04-24T19:38:02.173Z",
11  "updatedAt": "2024-04-24T19:38:02.173Z",
12  "comments": [
13    {
14      "id": 5,
15      "content": "Comment Прикладная информатика",
16      "userId": 2,
17      "articleId": 1,
18      "createdAt": "2024-04-24T19:38:45.721Z",
19      "updatedAt": "2024-04-24T19:38:45.721Z"
```

Вывод

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