САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №2

Выполнила:
Самчук Анита
К34402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

В рамках данной лабораторной работы необходимо реализовать RESTful API, используя написанный ранее boilerplate. Сервис, подобно PasteBin, позволяет пользователям размещать посты с кодом, а также оставлять комментарии под этими постами.

Ход работы

Мы определили три основные модели данных: User (пользователь), Post (пост с кодом) и Comment (комментарий к посту).

У юзера добавились новые поля, которые связывают его с оставленными им постами и комметариями

```
@HasMany( associatedClassGetter: () => Post,  options: {onDelete: 'CASCADE'})
posts!: Post[]

@HasMany( associatedClassGetter: () => Comment,  options: {onDelete: 'CASCADE'})
comments!: Comment[]
```

Модель поста

```
@Table
class Post extends Model {
    @Default( value: "No title")
    @Column
    title!: string

    @AllowNull( allowNull: false)
    @Column
    body!: string

    @Default(Language.none)
    @Column
    language!: Language

    @UpdatedAt
    updatedOn!: Date;

    @CreatedAt
    creationDate!: Date

    @ForeignKey( relatedClassGetter: () => User)
    @Column
    userId!: number;

    @BelongsTo( associatedClassGetter: () => User,  options: {onDelete: 'CASCADE'})
    user!: User;

    @HasMany( associatedClassGetter: () => Comment,  options: {onDelete: 'CASCADE'})
    comments!: Comment[];
}
```

Модель комментария

```
@Table
class Comment extends Model {
    @AllowNull( allowNull: false)
    @Column
    body!: string

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

    @BelongsTo( associatedClassGetter: () => User, options: {onDelete: 'CASCADE'})
    user?: User

    @ForeignKey( relatedClassGetter: () => Post)
    @Column
    postId?: number

    @BelongsTo( associatedClassGetter: () => Post, options: {onDelete: 'CASCADE'})
    post?: Post
}
```

Для каждой модели были созданы соответствующие роуты и контроллеры, которые обрабатывают HTTP-запросы.

Основные маршруты

```
const router : Router  = Router()

router.use('/users', userRouter)
router.use('/posts', postRouter)
```

У пользователя появился маршрут возвращающий его посты

```
// Посты авторизированного пользователя
userRouter.get( path: '/account/posts',
    passport.authenticate( strategy: 'jwt',  options: {session: false}),
    controller.getPosts)
```

Маршруты поста, где его создание, редактирование и удаление доступно только для авторизированного пользователя

```
export const postRouter : Router  = Router()
const controller: PostController = new PostController()

postRouter.use("/comments", commentRouter)
postRouter.get( path: "/", controller.getPosts)
postRouter.get( path: "/filter", controller.getFiltered)
postRouter.get( path: "/:postId/comments", controller.getComments)
postRouter.get( path: "/:id", controller.getPostById)
postRouter.post( path: "/create", passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.createPost)
postRouter.patch( path: "/update/:id", passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.updatePost)
postRouter.delete( path: "/delete/:id", passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.deletePost)
```

Маршруты комментариев

```
export const commentRouter : Router  = Router()
const controller: CommentController = new CommentController()

commentRouter.get( path: "/:postId", controller.getComments)
commentRouter.get( path: "/:postId/author/:userId", controller.getFiltered)
commentRouter.get( path: "/:postId/:commentId", controller.getCommentById)
commentRouter.post( path: "/:postId/create", passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.createComment)
commentRouter.delete( path: "/:postId/delete/:commentId", passport.authenticate( strategy: 'jwt',  options: {session: false}), controller.deleteComment)
```

Для работы с данными моделей были реализованы сервисы, вот пример части методов

Вывод постов пользователя (UserService)

```
    async getPosts(userId: number): Promise<Post[]> {
        const user : User | null  = await userRepository.findByPk(userId)

        if (user) {
            // @ts-ignore
            const posts = await user.getPosts()
            console.log(posts)
            if (posts) return posts
        }
        throw new UserError('User not found')
    }
```

Создание и обновление поста (PostService)

```
    async create(postData: Partial<Post>, userId: number): Promise<Post> {
        try {
            const user : User | null  = await userRepository.findByPk(userId)
            if (user) {
                // @ts-ignore
                const post = user.createPost(postData)
                if (post) return post
                throw new PostError('Could not create post')
            }
            throw new PostError('User not found')
        } catch (error: any) {
            throw new PostError(error.message)
        }
    }

    1 usage  Anita Samchuk
    async updatePost(id: number, newData: any, userId: number): Promise<Post> {
        try {
            const post : Post | null  = await postRepository.findByPk(id)
            if (post?.userId !== userId) throw new PostError('Could not edit someone else\'s post')
            if (post) {
                Object.assign(post, newData)
                return await post.save()
            }
            throw new PostError(`Post with id ${id} not found`)
        } catch (error: any) {
            throw new PostError(error.message)
        }
    }
```

Удаление комментария (CommentService)

```
async deleteComment(postId: number, commentId: number, userId: number) : Promise<void>  {
    try {
        const comment : Comment | null  = await commentRepository.findOne({
            where: {
                'postId': postId,
                'id': commentId
            }
        })

        if (comment) {
            if (comment?.userId !== userId) throw new Error("Could not delete someone else\'s comments")
            return await comment.destroy()
        }
        throw new Error("Comment not found")
    } catch (error: any) {
        throw new CommentError(error)
    }
}
```

Вывод

В результате выполнения лабораторной работы мы создали RESTful API, реализующее функциональность похожую на сервис PasteBin. Использование заранее разработанного boilerplate значительно упростило процесс разработки и обеспечило структурированность кода.