САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Дорофеева Арина

Группа к33401

Проверил:
Добряков Д. И.

Санкт-Петербург

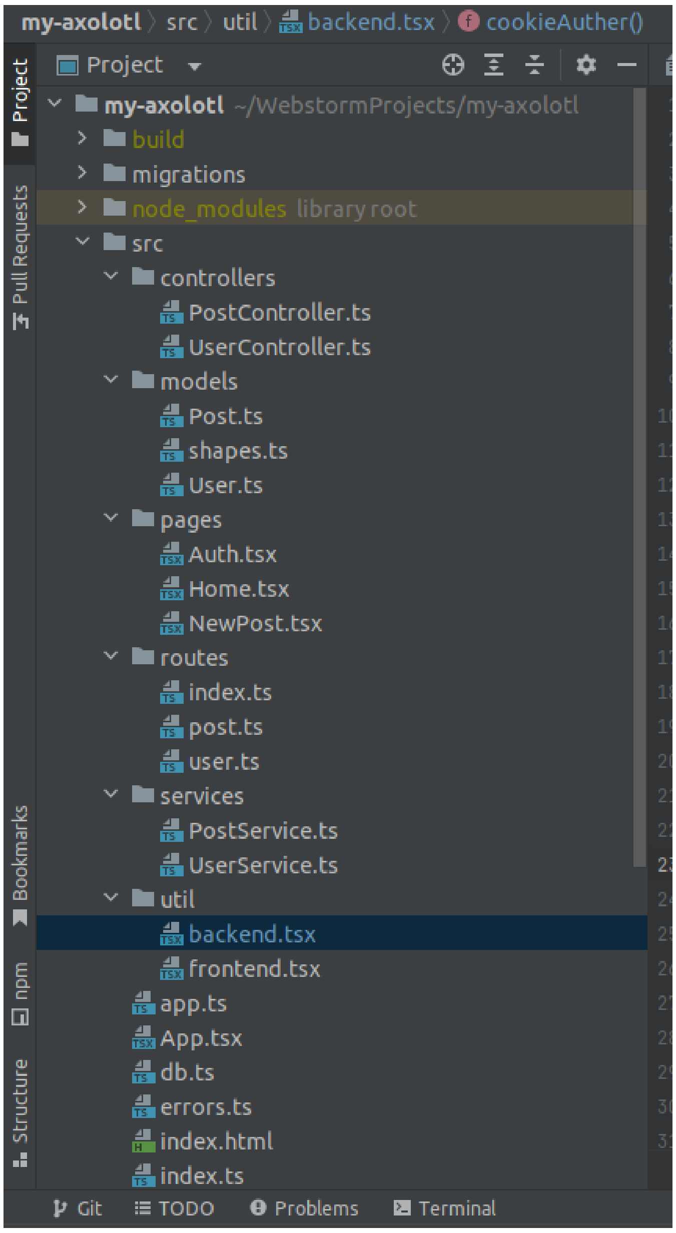
2022 г.

Задача

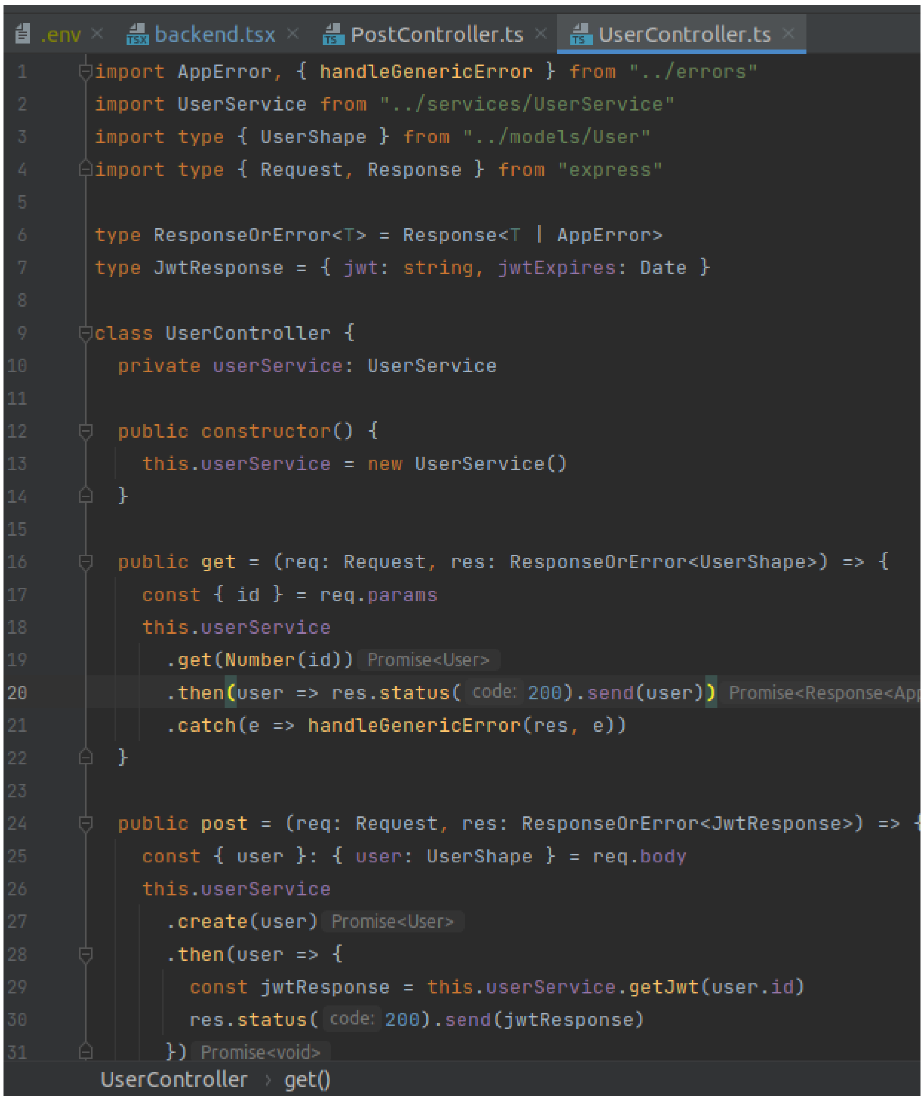
В рамках данной лабораторной работы Вам предложено выбрать один из нескольких вариантов. Выбранный вариант останется единым на весь курс и будет использоваться в последующих лабораторных работах.

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

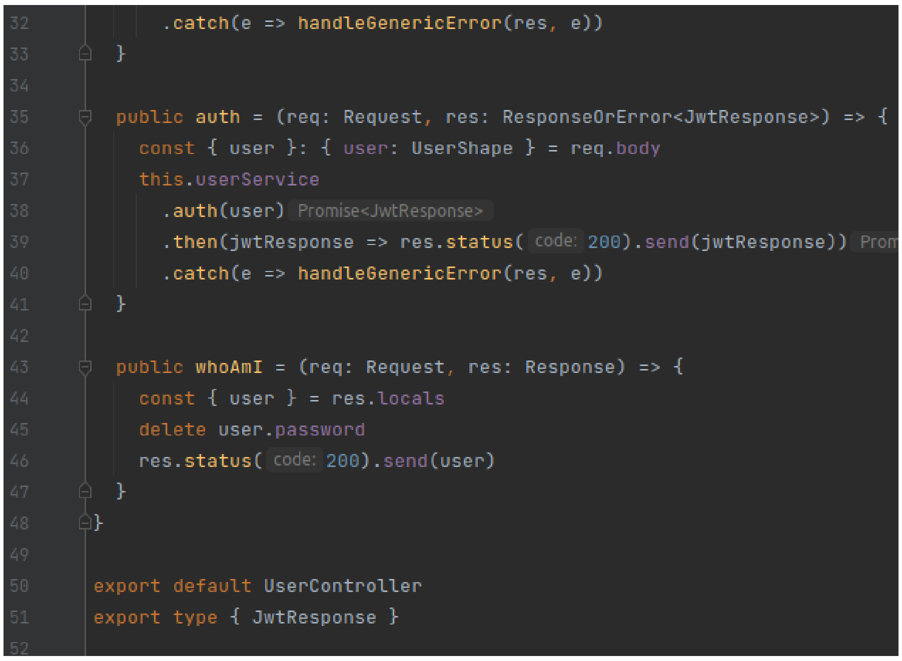
Ход работы


```
my-axolotl 〉 src 〉 util 〉 backend.tsx 〉 cookieAuther()
Project
my-axolotl ~/WebstormProjects/my-axolotl
  build
  migrations
  node_modules library root
  src
    controllers
      PostController.ts
      UserController.ts
    models
      Post.ts
      shapes.ts
      User.ts
    pages
      Auth.tsx
      Home.tsx
      NewPost.tsx
    routes
      index.ts
      post.ts
      user.ts
    services
      PostService.ts
      UserService.ts
    util
      backend.tsx
      frontend.tsx
    app.ts
    App.tsx
    db.ts
    errors.ts
    index.html
    index.ts
```

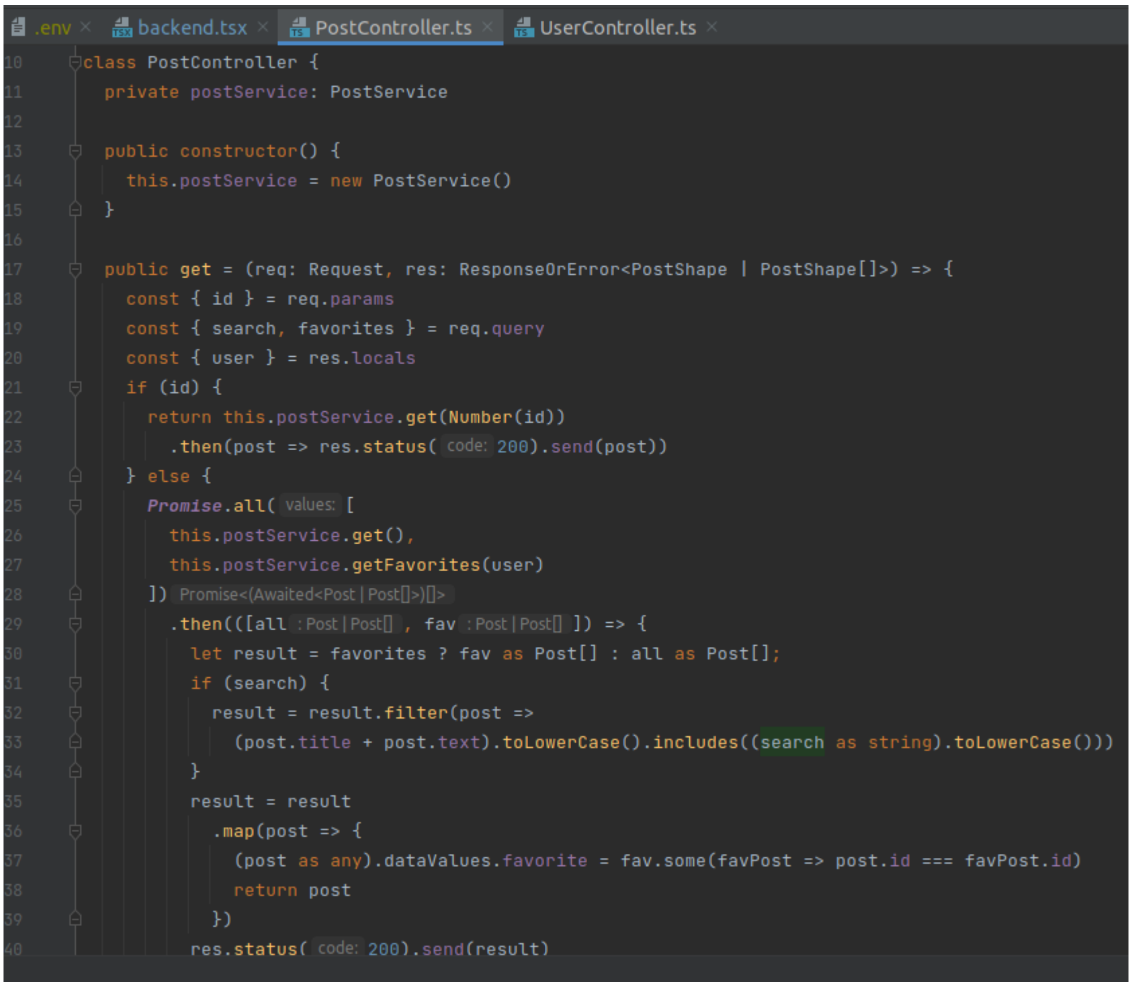
Контроллеры:



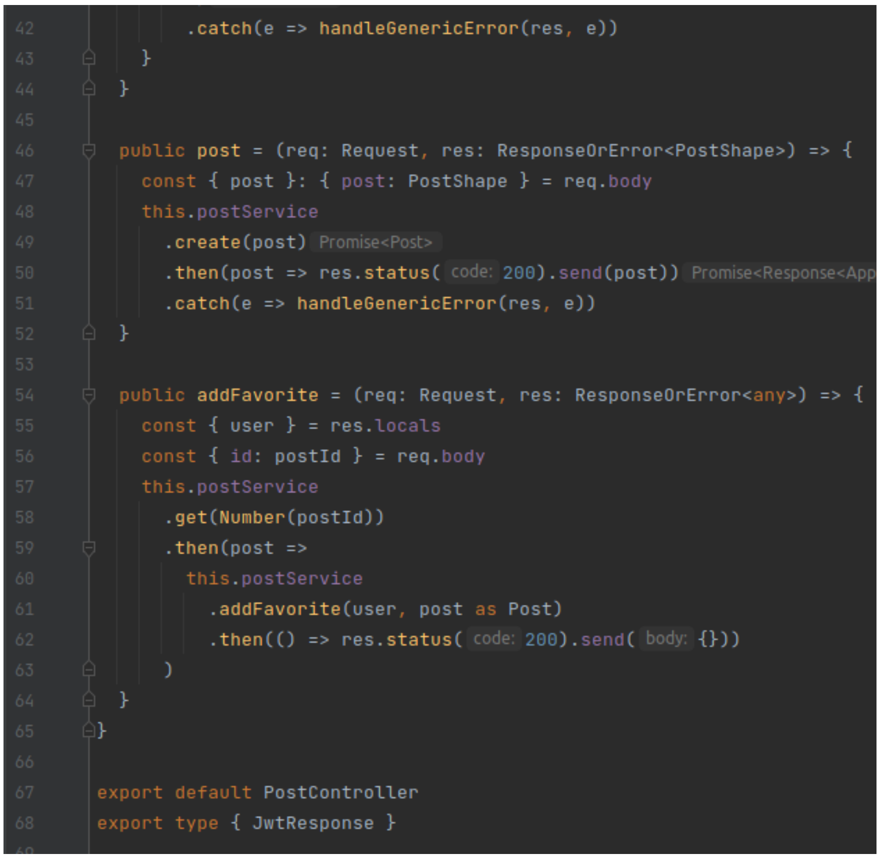
```typescript
import AppError, { handleGenericError } from "../errors"
import UserService from "../services/UserService"
import type { UserShape } from "../models/User"
import type { Request, Response } from "express"

type ResponseOrError<T> = Response<T | AppError>
type JwtResponse = { jwt: string, jwtExpires: Date }

class UserController {
  private userService: UserService

  public constructor() {
    this.userService = new UserService()
  }

  public get = (req: Request, res: ResponseOrError<UserShape>) => {
    const { id } = req.params
    this.userService
      .get(Number(id))
      .then(user => res.status(200).send(user))
      .catch(e => handleGenericError(res, e))
  }

  public post = (req: Request, res: ResponseOrError<JwtResponse>) => {
    const { user }: { user: UserShape } = req.body
    this.userService
      .create(user)
      .then(user => {
        const jwtResponse = this.userService.getJwt(user.id)
        res.status(200).send(jwtResponse)
      })
```



```typescript
      .catch(e => handleGenericError(res, e))
  }

  public auth = (req: Request, res: ResponseOrError<JwtResponse>) => {
    const { user }: { user: UserShape } = req.body
    this.userService
      .auth(user)
      .then(jwtResponse => res.status(200).send(jwtResponse))
      .catch(e => handleGenericError(res, e))
  }

  public whoAmI = (req: Request, res: Response) => {
    const { user } = res.locals
    delete user.password
    res.status(200).send(user)
  }
}

export default UserController
export type { JwtResponse }
```

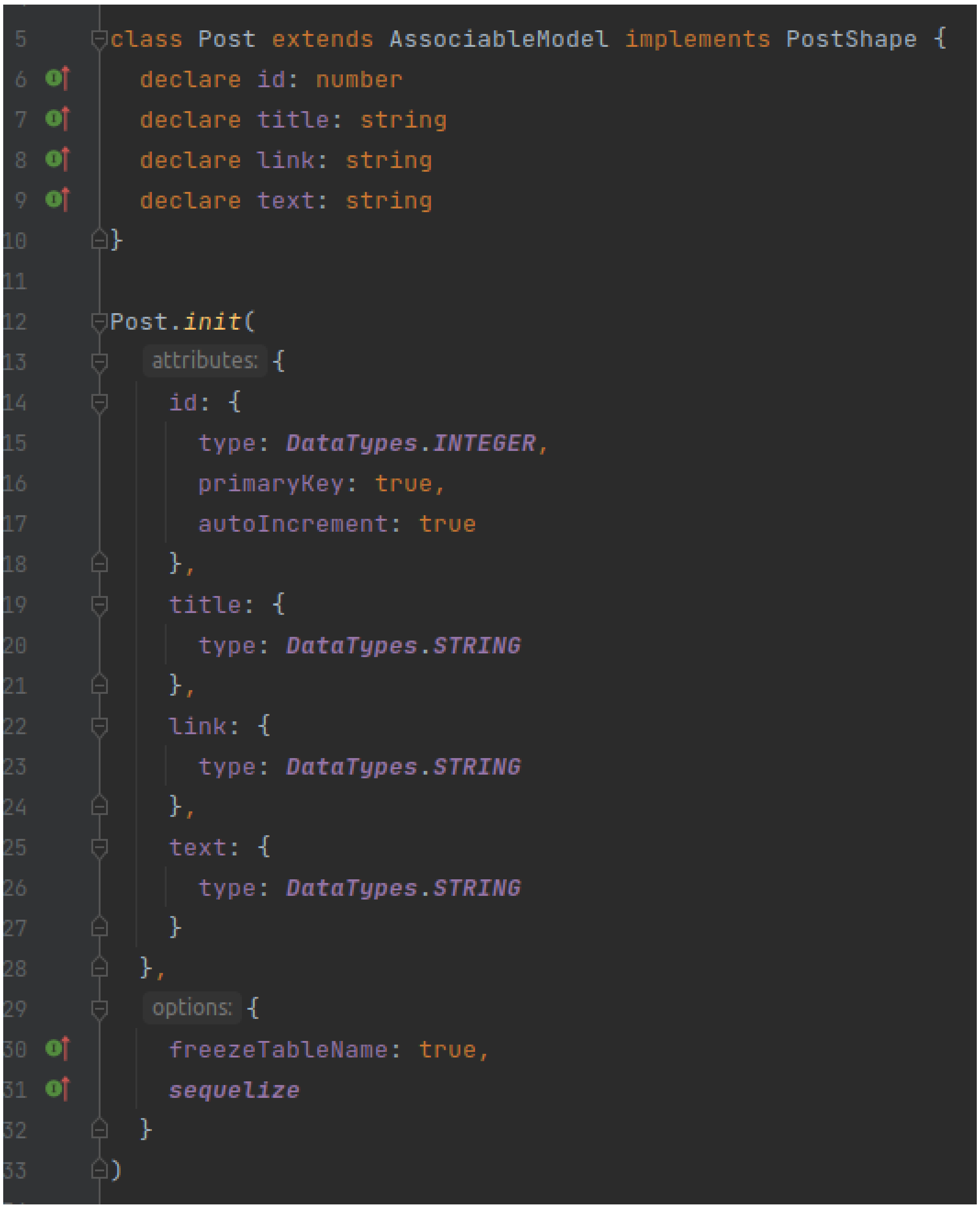


```
class PostController {
  private postService: PostService

  public constructor() {
    this.postService = new PostService()
  }

  public get = (req: Request, res: ResponseOrError<PostShape | PostShape[]>) => {
    const { id } = req.params
    const { search, favorites } = req.query
    const { user } = res.locals
    if (id) {
      return this.postService.get(Number(id))
        .then(post => res.status( code: 200).send(post))
    } else {
      Promise.all( values: [
        this.postService.get(),
        this.postService.getFavorites(user)
      ]) Promise<(Awaited<Post | Post[]>)[]>
        .then(([all :Post | Post[] , fav :Post | Post[] ]) => {
          let result = favorites ? fav as Post[] : all as Post[];
          if (search) {
            result = result.filter(post =>
              (post.title + post.text).toLowerCase().includes((search as string).toLowerCase()))
          }
          result = result
            .map(post => {
              (post as any).dataValues.favorite = fav.some(favPost => post.id === favPost.id)
              return post
            })
          res.status( code: 200).send(result)
```

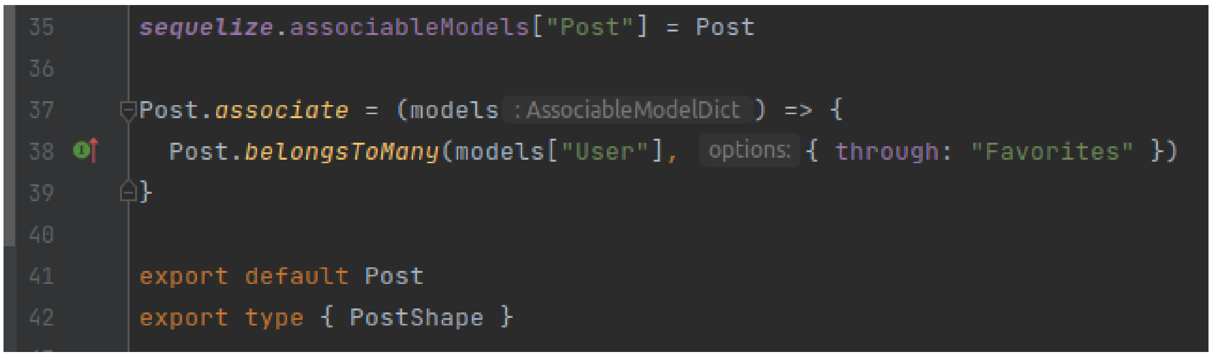


```
        .catch(e => handleGenericError(res, e))
    }
  }

  public post = (req: Request, res: ResponseOrError<PostShape>) => {
    const { post }: { post: PostShape } = req.body
    this.postService
      .create(post) Promise<Post>
      .then(post => res.status( code: 200).send(post)) Promise<Response<App
      .catch(e => handleGenericError(res, e))
  }

  public addFavorite = (req: Request, res: ResponseOrError<any>) => {
    const { user } = res.locals
    const { id: postId } = req.body
    this.postService
      .get(Number(postId))
      .then(post =>
        this.postService
          .addFavorite(user, post as Post)
          .then(() => res.status( code: 200).send( body: {}))
      )
  }
}

export default PostController
export type { JwtResponse }
```

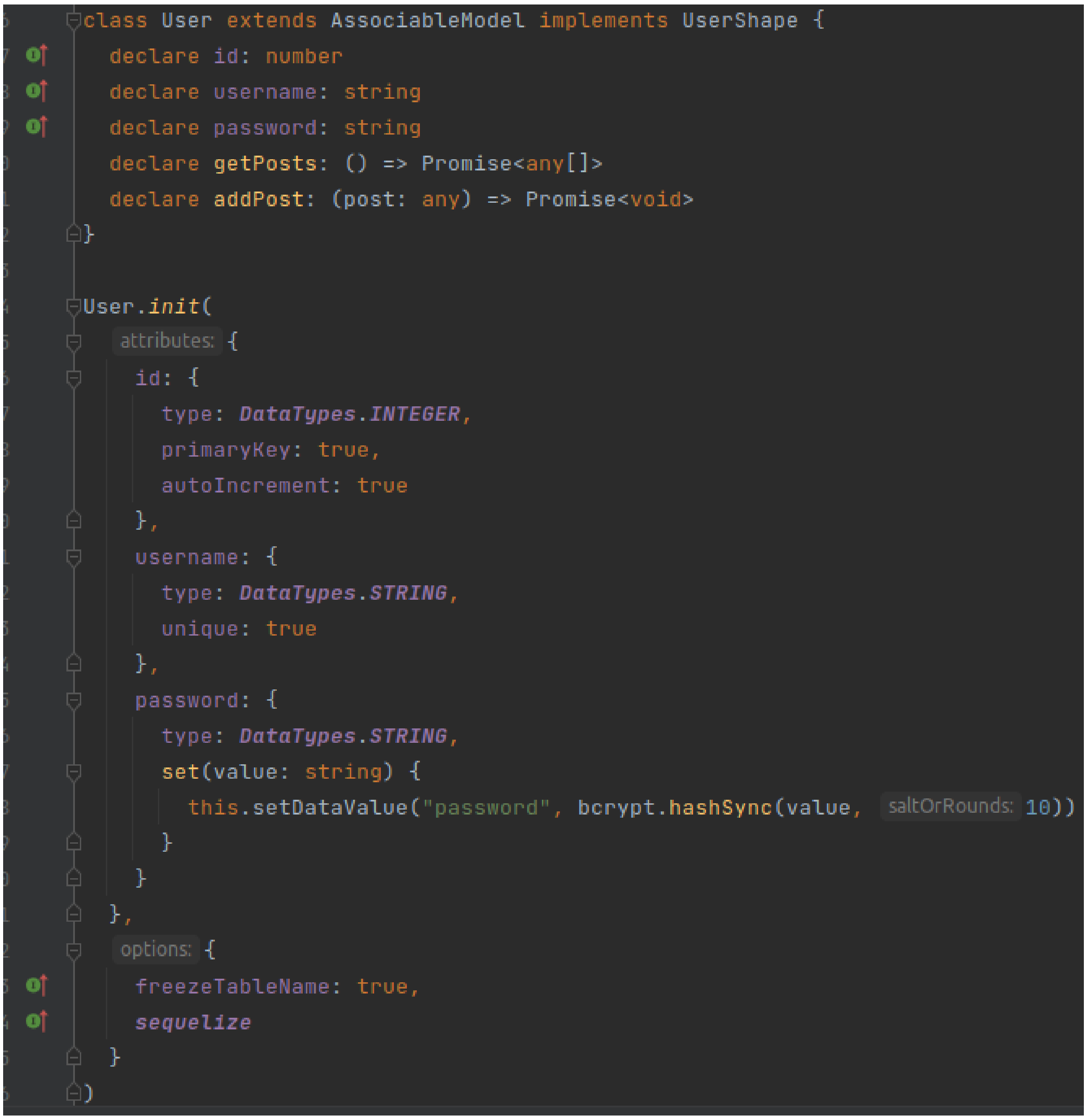
Модели:



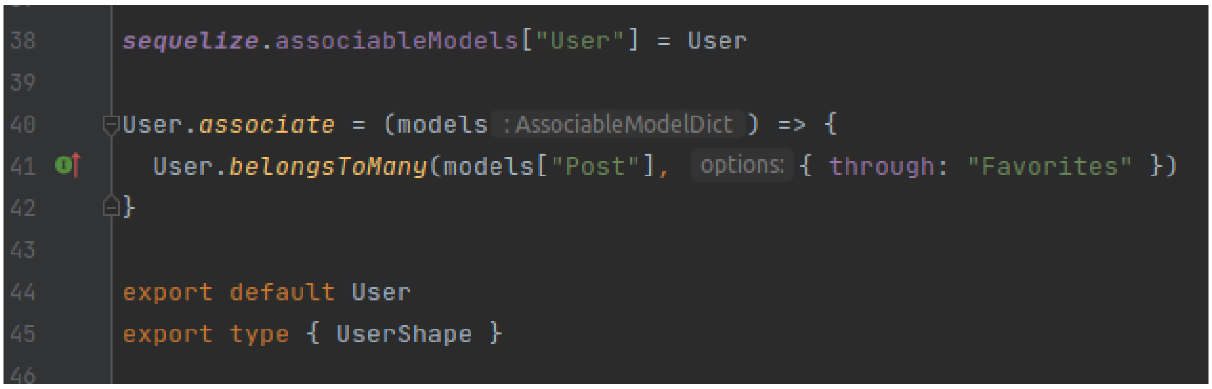
```
class Post extends AssociableModel implements PostShape {
  declare id: number
  declare title: string
  declare link: string
  declare text: string
}

Post.init(
  attributes: {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    title: {
      type: DataTypes.STRING
    },
    link: {
      type: DataTypes.STRING
    },
    text: {
      type: DataTypes.STRING
    }
  },
  options: {
    freezeTableName: true,
    sequelize
  }
)
```



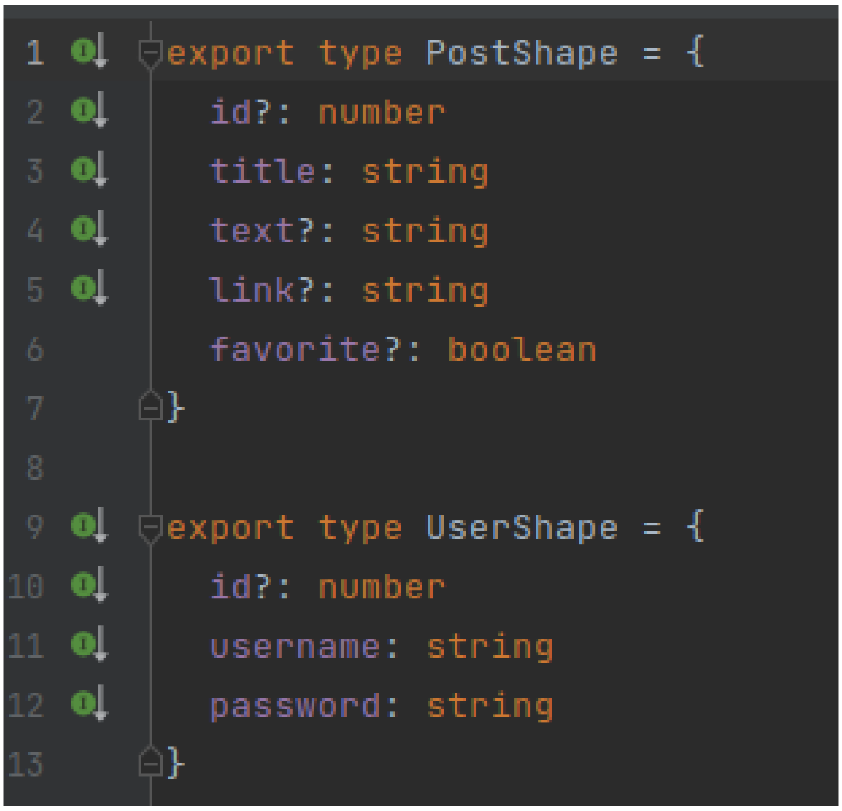
```
sequelize.associableModels["Post"] = Post

Post.associate = (models : AssociableModelDict ) => {
  Post.belongsToMany(models["User"],  options: { through: "Favorites" })
}

export default Post
export type { PostShape }
```



```
class User extends AssociableModel implements UserShape {
  declare id: number
  declare username: string
  declare password: string
  declare getPosts: () => Promise<any[]>
  declare addPost: (post: any) => Promise<void>
}

User.init(
  attributes: {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    username: {
      type: DataTypes.STRING,
      unique: true
    },
    password: {
      type: DataTypes.STRING,
      set(value: string) {
        this.setDataValue("password", bcrypt.hashSync(value,  saltOrRounds: 10))
      }
    }
  },
  options: {
    freezeTableName: true,
    sequelize
  }
)
```

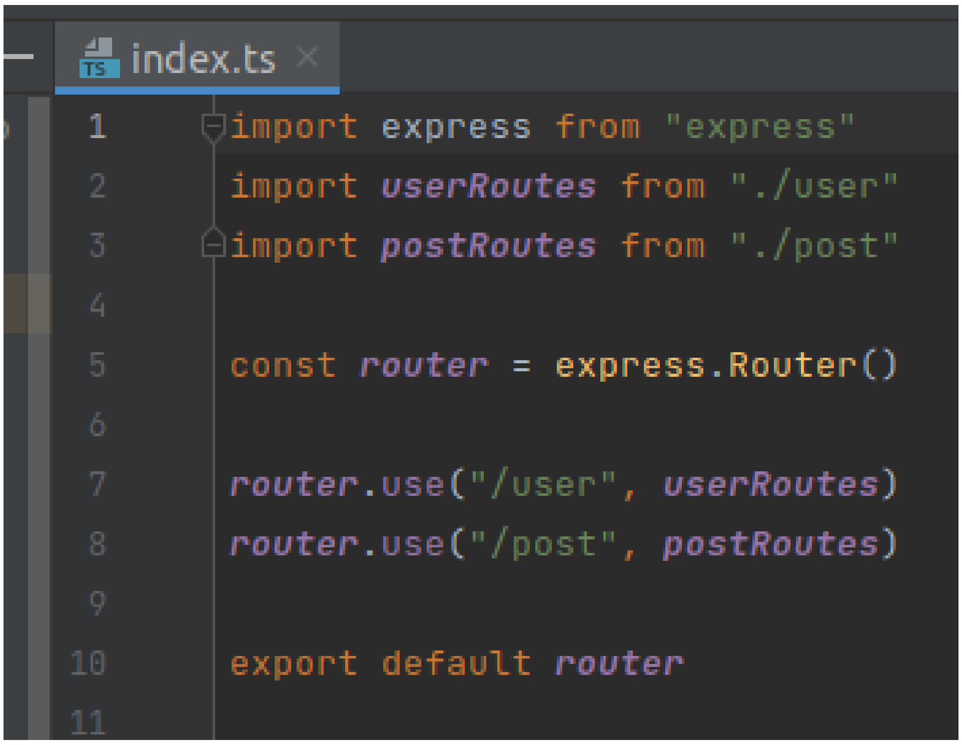


```
sequelize.associableModels["User"] = User

User.associate = (models : AssociableModelDict ) => {
  User.belongsToMany(models["Post"],  options: { through: "Favorites" })
}

export default User
export type { UserShape }
```



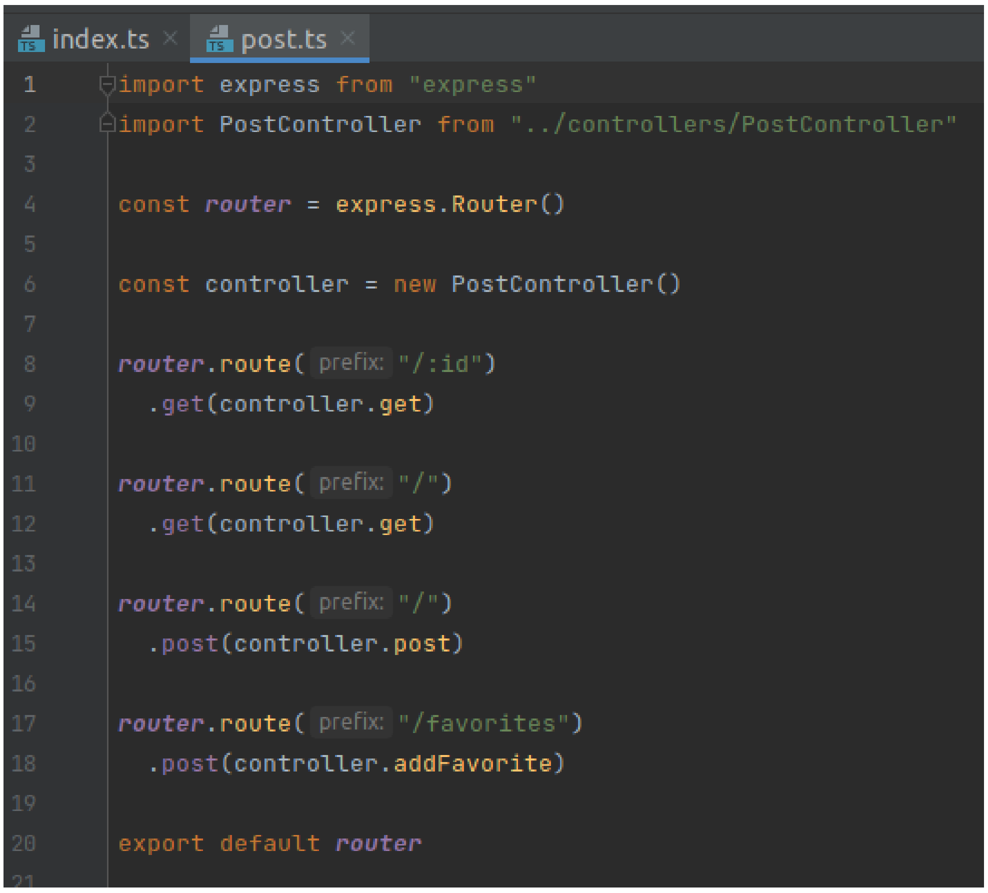
```
export type PostShape = {
   id?: number
   title: string
   text?: string
   link?: string
   favorite?: boolean
}

export type UserShape = {
   id?: number
   username: string
   password: string
}
```

Роуты:

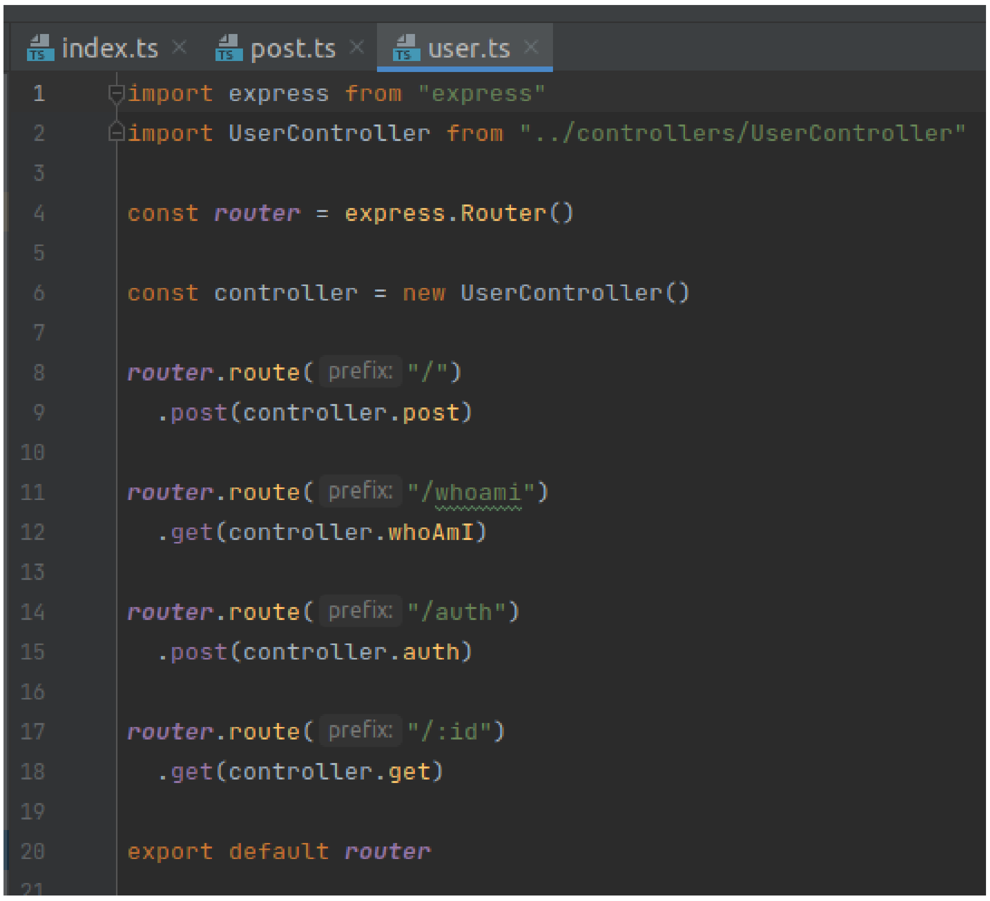


index.ts

```
import express from "express"
import userRoutes from "./user"
import postRoutes from "./post"

const router = express.Router()

router.use("/user", userRoutes)
router.use("/post", postRoutes)

export default router
```

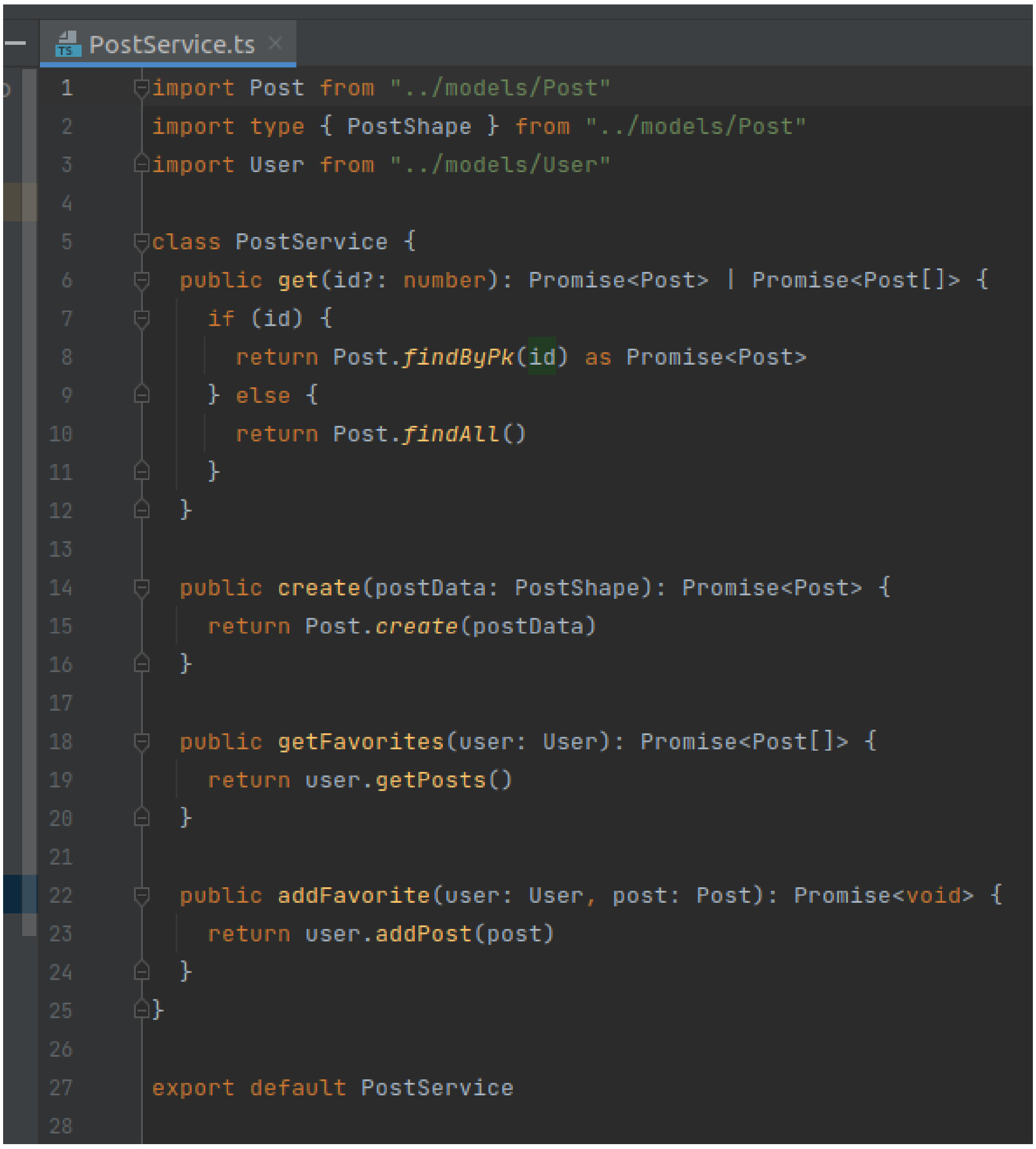

index.ts × post.ts ×

```
import express from "express"
import PostController from "../controllers/PostController"

const router = express.Router()

const controller = new PostController()

router.route( prefix: "/:id")
  .get(controller.get)

router.route( prefix: "/")
  .get(controller.get)

router.route( prefix: "/")
  .post(controller.post)

router.route( prefix: "/favorites")
  .post(controller.addFavorite)

export default router
```

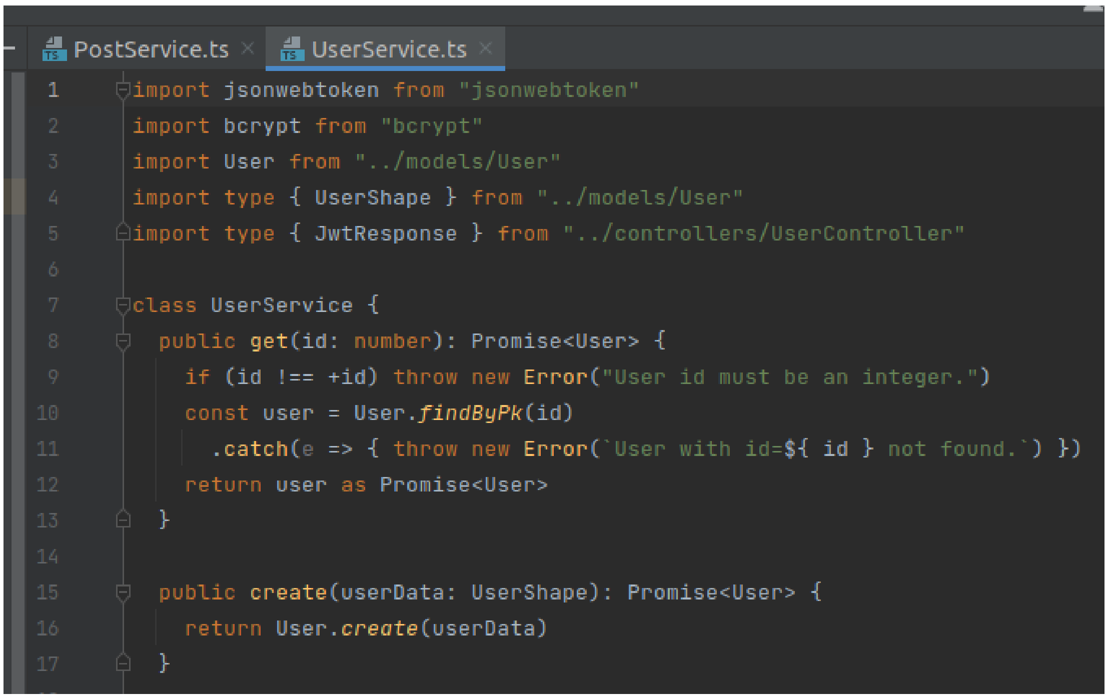

index.ts × post.ts × user.ts ×

```
import express from "express"
import UserController from "../controllers/UserController"

const router = express.Router()

const controller = new UserController()

router.route( prefix: "/")
  .post(controller.post)

router.route( prefix: "/whoami")
  .get(controller.whoAmI)

router.route( prefix: "/auth")
  .post(controller.auth)

router.route( prefix: "/:id")
  .get(controller.get)

export default router
```

Сервисы:

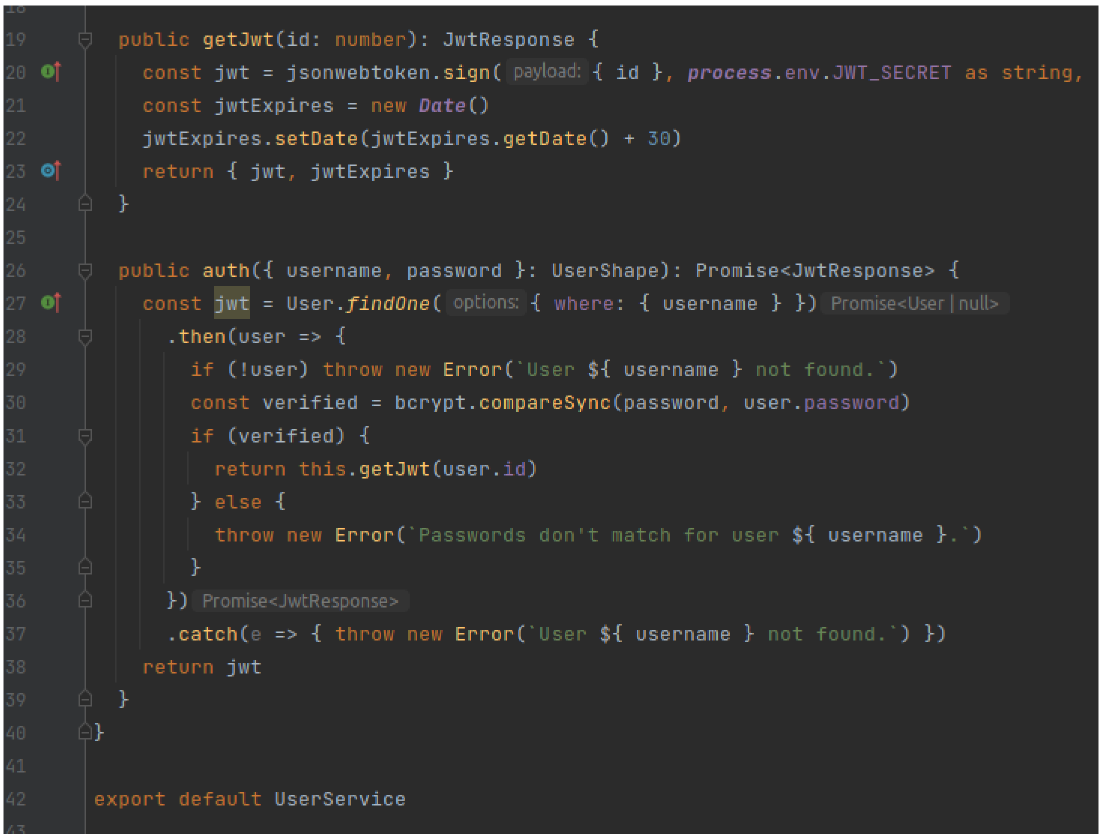


```typescript
import Post from "../models/Post"
import type { PostShape } from "../models/Post"
import User from "../models/User"

class PostService {
  public get(id?: number): Promise<Post> | Promise<Post[]> {
    if (id) {
      return Post.findByPk(id) as Promise<Post>
    } else {
      return Post.findAll()
    }
  }

  public create(postData: PostShape): Promise<Post> {
    return Post.create(postData)
  }

  public getFavorites(user: User): Promise<Post[]> {
    return user.getPosts()
  }

  public addFavorite(user: User, post: Post): Promise<void> {
    return user.addPost(post)
  }
}

export default PostService

```



PostService.ts UserService.ts

```
import jsonwebtoken from "jsonwebtoken"
import bcrypt from "bcrypt"
import User from "../models/User"
import type { UserShape } from "../models/User"
import type { JwtResponse } from "../controllers/UserController"

class UserService {
  public get(id: number): Promise<User> {
    if (id !== +id) throw new Error("User id must be an integer.")
    const user = User.findByPk(id)
      .catch(e => { throw new Error(`User with id=${ id } not found.`) })
    return user as Promise<User>
  }

  public create(userData: UserShape): Promise<User> {
    return User.create(userData)
  }
```



```
  public getJwt(id: number): JwtResponse {
    const jwt = jsonwebtoken.sign( payload: { id }, process.env.JWT_SECRET as string,
    const jwtExpires = new Date()
    jwtExpires.setDate(jwtExpires.getDate() + 30)
    return { jwt, jwtExpires }
  }

  public auth({ username, password }: UserShape): Promise<JwtResponse> {
    const jwt = User.findOne( options: { where: { username } }) Promise<User | null>
      .then(user => {
        if (!user) throw new Error(`User ${ username } not found.`)
        const verified = bcrypt.compareSync(password, user.password)
        if (verified) {
          return this.getJwt(user.id)
        } else {
          throw new Error(`Passwords don't match for user ${ username }.`)
        }
      }) Promise<JwtResponse>
      .catch(e => { throw new Error(`User ${ username } not found.`) })
    return jwt
  }
}

export default UserService
```

Вывод

В ходе работы я реализовала RESTful API для приложения с аксолотлями средствами express + typescript (используя ранее написанный boilerplate).