САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Акулов Даниил

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Нужно написать свой boilerplate на express + TypeORM + typescript. Должно быть явное разделение на модели, контроллеры и роуты. Должна быть реализована базовая пользовательская модель, jwt авторизация и конфигурация через переменные среды

Ход работы

Авторизация реализована с использованием библиотек jsonwebtoken и bcrypt.

Во время регистрации и логина пользователю выдается JWT, которые он далее использует при взаимодействии с API. Пароль пользователей хешируется, может быть применено несколько раундов хеш-функции в зависимости от конфигурации проекта.

Функция для генерации jwt-токена:

```
const generateAccessToken = (id: number, email: string) => {
   const payload = { id, email }
     return jwt.sign(payload, SETTINGS.JWT_SECRET_KEY, {expiresIn:
"30d"} )
}
```

Метод для обработки авторизации:

```
login: RequestHandler = async (req, res, next) => {
   try {
       const {email, password} = req.body
       const user = await userRepo.findOne({where: {email}})
       if (!user) {
           return
next(ApiError.badRequest(errorMessages.userNotFound))
       }
       const validPassword = bcrypt.compareSync(password,
user.password)
       if (!validPassword) {
           return
next(ApiError.badRequest(errorMessages.wrongPassword))
       }
       const token = generateAccessToken(user.id, user.email)
       return res.json({token, user})
```

```
    }  catch (e) {
        next(ApiError.internal())
    }
}
```

Выбранной СУБД стала PostgreSQL, так как является самой популярной и доступной на текущий момент. Для подключения к БД используется TypeORM.

Реализации модели User:

```
@Entity("users")
export class User {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({type: "varchar", length: 256, unique: true})
    email: string;

    @Column({type: "varchar", length: 256})
    password: string;

    @Column({type: "varchar", length: 256})
    avatarUrl: string;

    @Column({type: 'varchar', length: 256})
    name: string;

    @CreateDateColumn()
    createdAt: Date;
}
```

Листинг части контроллера для модели User:

```
const userRepo = dataSource.getRepository(User);

class UserController {
    getOne: RequestHandler = async (req, res, next) => {
    try {
```

```
        const user = await userRepo.findOne({ where: { id:
+req.params.id } });
        if (!user) {
            return
next(ApiError.badRequest(errorMessages.userNotFound));
        }
        return res.json({user});
    } catch (e) {
        next(ApiError.internal());
    }
}

update: RequestHandler = async (req, res, next) => {
    try {
        const user = await userRepo.findOne({ where: { id:
+req.params.id } });
        if (!user) {
            return
next(ApiError.badRequest(errorMessages.userNotFound));
        }
        user.name = req.body.name || user.name;
        user.email = req.body.email || user.email;
        user.avatarUrl = req.body.avatarUrl || user.avatarUrl;
        await userRepo.save(user);
        return res.json({user});
    } catch (e) {
        next(ApiError.internal());
    }
}
```

Реализованы API-эндпоинты для регистрации, авторизации и получения информации о пользователе по id. Листинг роутера для модели User:

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

router.get('/get-all', userController.getAll)
router.get('/get-one/:id', userController.getOne)
router.put('/', checkAuth, userController.update)
router.delete('/', checkAuth, userController.delete)

export default router;
```

Листинг index.ts файла:

```
const app = express();

app.use(express.json());
app.use('/api', router);
app.use(errorHandler);

dataSource
   .initialize()
   .then(() => {
       console.log('Data Source has been initialized!');
           app.listen(SETTINGS.API_PORT, () => console.log(`Server
started on http://localhost:${SETTINGS.API_PORT}`));
   })
   .catch((err) => {
         console.error('Error during Data Source initialization:',
err);
   });
```

Вывод

Создал удобный boilerplate для переиспользования при создании новых проектов на Node.JS с TypeORM с различными конфигурациями.