САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Акулов Даниил

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Необходимо реализовать RESTful API, используя ранее написанный boilerplate.

Ход работы

Была использована архитектура, разработанная в ЛР1, включающая: слой моделей (для работы с данными), слой сервисов (для бизнес-логики), слой роутов (эндпоинты)

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.

На основе этой архитектуры реализованы REST-эндпоинты для всех сущностей:

```
router.use('/auth', authRouter)
router.use('/user', userRouter)
router.use('/blog-post', blogPostRouter)
router.use('/exercise-type', exerciseTypeRouter)
router.use('/session-exercise', sessionExerciseRouter)
router.use('/workout-plan', workoutPlanRouter)
router.use('/workout-session', workoutSessionRouter)
```

Пример набора поддерживающих операций:

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

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

export default router
```

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());
    }
}
```

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

В ходе выполнения лабораторной работы было реализовано RESTful API на базе ранее разработанного boilerplate-приложения. API соответствует принципам REST и включает все необходимые CRUD-операции для ключевых сущностей. Проект легко расширяется и масштабируется благодаря использованию DI и принципов чистой архитектуры.