## Лабораторная работа 4. Node.js. Реализация дополнительной функциональности

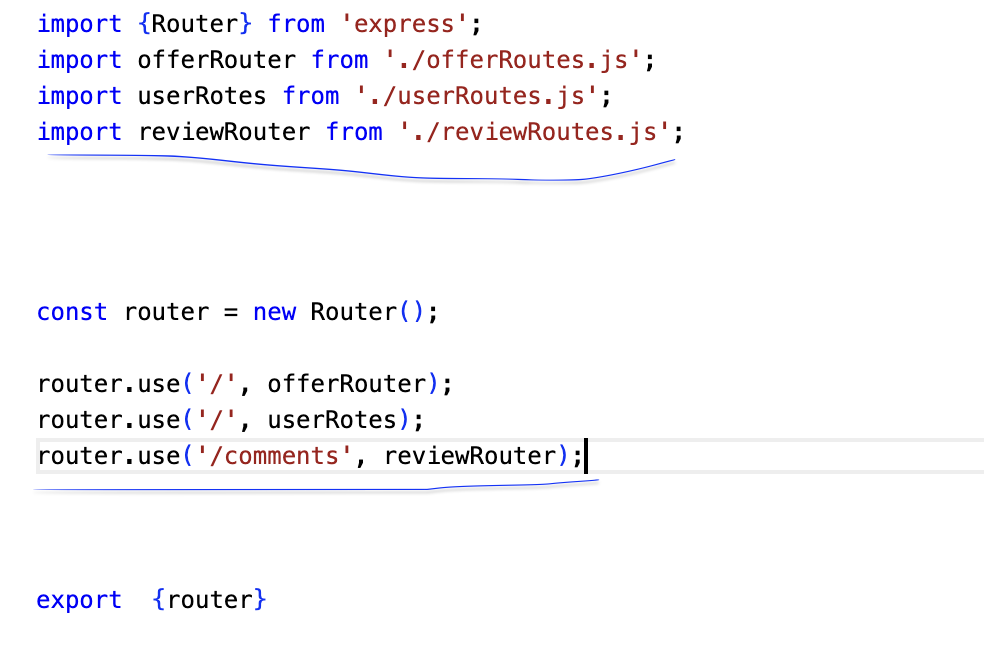
### Задание 1. Работа с отзывами

1. Согласно ТЗ необходимо реализовать 2 функции
   1. Добавление нового комментария
   2. Получение списка комментариев к предложению
2. Начнем с первой функции. Перейдем в controllers/[reviewController.js](http://reviewcontroller.js) и создадим функцию addReview. Но прежде импортируем класс Review из модели данных и класс ApiError для обработки ошибок.



Здесь переменные comment и rating приходят из тела POST-запроса, offerId - приходит как параметр, так как он был передан через URL, userID будет получен после того, как пользователь пройдет процедуру аутентификации на клиенте.   
Согласно модели данных, для формирования нового отзыва необходимы следующие данные: текст отзыва, рейтинг, дата публикации отзыва (по умолчанию это текущая дата), id-пользователя и id-предложения, для которого формируется отзыв.

! Не забудьте экспортировать функцию.

1. Сформируем маршрут для созданной функции. В файле routes/[reviewRoutes.js](http://reviewroutes.js)  
   
2. В файле routes/[index.js](http://index.js)  
   
3. Теперь перейдем к созданию второй функции - Получение списка комментариев к предложению. При возврате данных на клиента необходимо не только вернуть сами отзывы, но и данные о пользователях, которые эти отзывы оставили, а именно необходимо вернуть имя пользователя, аватар и данные о типе пользователя (pro/normal). Так как здесь фигурирует аватар, а это фотография пользователя, значит необходимо создать дополнительную функцию, которая будет формировать необходимый адрес для фотографии, и преобразовывать другие данные. В папке adapters создадим файл [reviewAdapter.js](http://reviewadapter.js).

const getBaseUrl = () => `${process.env.HOST}:${process.env.PORT || 5000}`;

const adaptReviewToClient = (review) => {

const baseUrl = getBaseUrl();

const prepareUrl = (url) =>

url && !url.startsWith('http') ? `${baseUrl}${url.startsWith('/') ? '' : '/'}${url}` : url;

return {

id: String(review.id),

comment: review.text,

rating: parseFloat(review.rating),

date: review.publishDate instanceof Date

? review.publishDate.toISOString()

: new Date(review.publishDate).toISOString(),

user: {

name: review.author?.username || 'Unknown',

avatarUrl: prepareUrl(review.author?.avatar || ''),

isPro: review.author?.userType === 'pro'

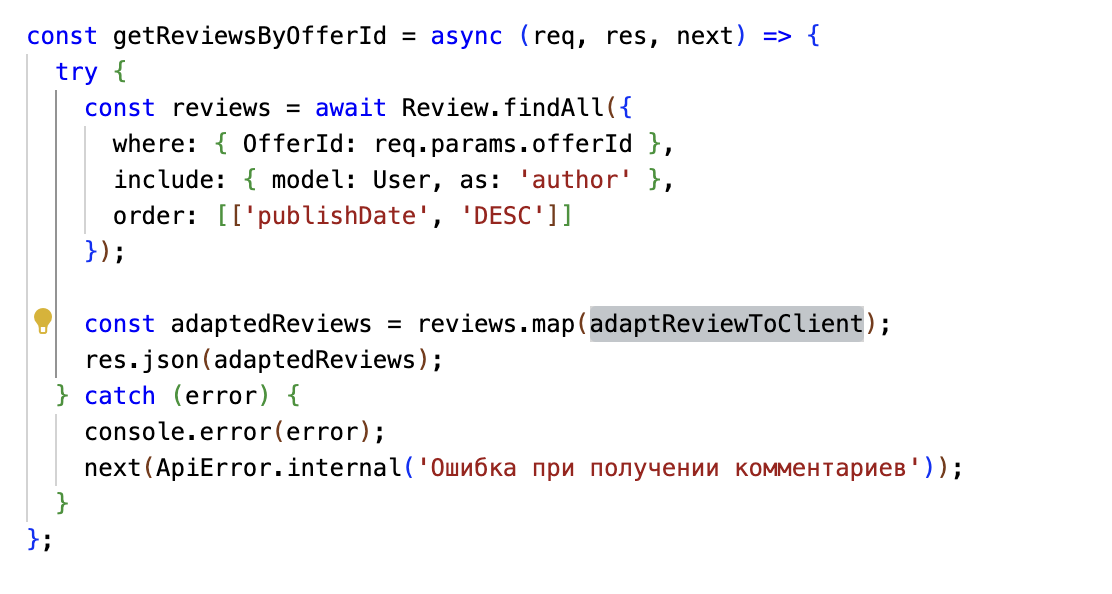
}

};

};

export { adaptReviewToClient };

1. Теперь в [reviewController.js](http://reviewcontroller.js) создадим функцию



Функцию adaptReviewToClient - предварительно импортируйте. А функцию getReviewsByOfferId - экспортируйте.

1. Создайте маршрут для функции в [reviewRoutes.js](http://reviewroutes.js)

router.get('/:offerId', getReviewsByOfferId);

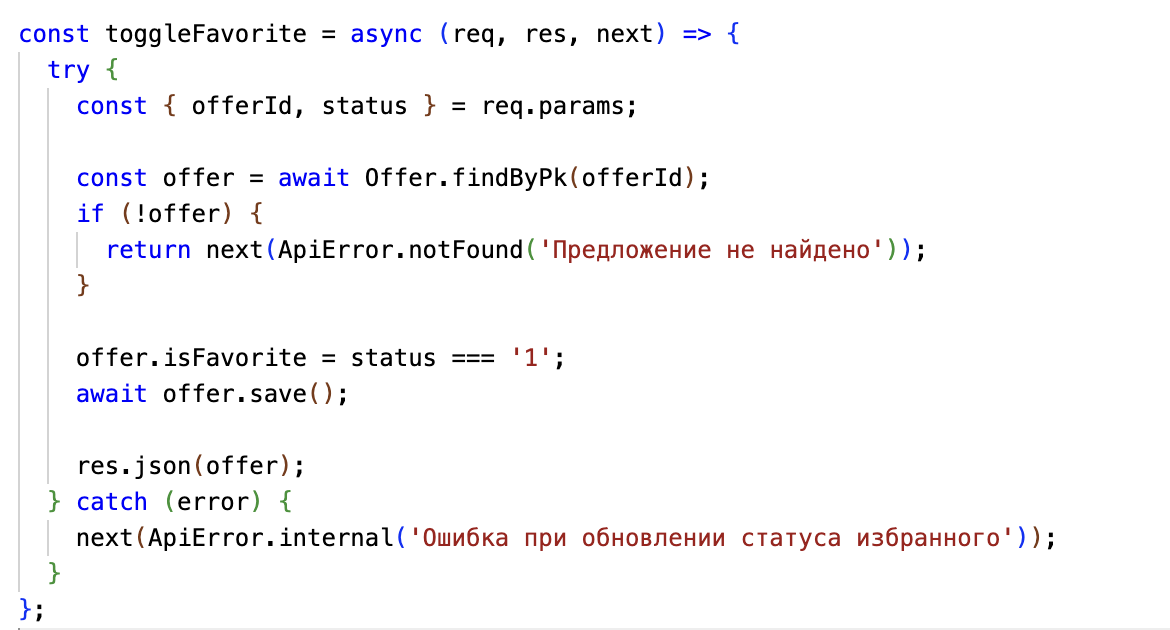
1. Протестируйте функцию getReviewsByOfferId в Postman для проверки работоспособности.

### Задание 2. Работа с предложениями по аренде

Согласно ТЗ осталось реализовать 2 функции:

- Получение списка предложений в избранном  
- Добавление / удаление предложения в/из избранное

У нас в БД нет отдельной таблицы для хранения избранных офферов каждого пользователя, поэтому для функции Получения списка избранных предложений будем использовать просто офферы, у которых isFavorite = true.

1. Создайте самостоятельно функцию getFavoriteOffers
2. Создайте маршрут для получения эти офферов router.get('/favorite', getFavoriteOffers).
3. Создадим функцию добавления/удаления предложения в/из избранное. В файле [offerControllers.js](http://offercontrollers.js) добавим функцию toggleFavorite:  
   
4. Создадим маршрут:

router.post('/favorite/:offerId/:status', toggleFavorite);

### Задание 3. Работа с пользователями. JWT-token

Согласно ТЗ осталось реализовать следующие функции:

- Авторизация пользователя  
- Выход пользователя  
- Проверка статуса аутентификации

Также необходимо учесть тот факт, что при авторизации пользователя будет генерироваться JWT-токен.

JWT (JSON Web Token) — это способ аутентификации без сессий. После входа клиент получает токен и отправляет его в каждом запросе. Сервер проверяет подпись токена и извлекает из него [user.id](http://user.id).

jsonwebtoken.sign(payload, secret) - создание токена

jsonwebtoken.verify(token, secret) - проверка токена

Authorization: Bearer <token> - передача токена в заголовке.

1. Установим пакет jsonwebtoken
2. Создадим функцию авторизации пользователя - login.
   1. Импортируем в [userController.js](http://usercontroller.js)

import jwt from 'jsonwebtoken';

* 1. Там же создаем функцию

const login = async (req, res, next) => {

try {

const { email, password } = req.body;

const user = await User.findOne({ where: { email } });

if (!user) return next(ApiError.badRequest('Пользователь не найден'));

const isValid = await bcrypt.compare(password, user.password);

if (!isValid) return next(ApiError.badRequest('Неверный пароль'));

const token = jwt.sign({ id: user.id }, process.env.JWT\_SECRET, { expiresIn: '24h' });

res.json({ token });

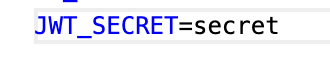
} catch (error) {

next(ApiError.internal('Ошибка авторизации'));

}

};

В переменных окружения создайте JWT\_SECRET и присвойте любое значение.



Сначала находим пользователя по указанному email, затем сравниваем пароль, предварительно зашифрованный. Если проверки прошли, то создаем токен, который будет передан клиенту.

1. Создадим маршрут для созданной функции

router.post('/login', login);

1. Теперь у клиента после авторизации есть сформированный токен для того, чтобы можно было посещать закрытые для обычных пользователей страницы приложения. Для проверки доступа к данным страницам необходимо создать дополнительную функцию authenticateToken. Обычно эта проверка токена производится в middleware.
2. в папке middleware создадим файл [authMiddleware.js](http://authmiddleware.js), в котором опишем функцию authenticateToken

import jwt from 'jsonwebtoken';

import ApiError from '../error/ApiError.js';

import {User} from '../models/user.js';

const authenticateToken = async (req, res, next) => {

try {

const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith('Bearer ')) {

return next(ApiError.unauthorized('Нет токена'));

}

const token = authHeader.split(' ')[1];

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const user = await User.findByPk(decoded.id);

if (!user) {

return next(ApiError.unauthorized('Пользователь не найден'));

}

// Можно сохранить пользователя в req.user для использования дальше

req.user = user;

next();

} catch (error) {

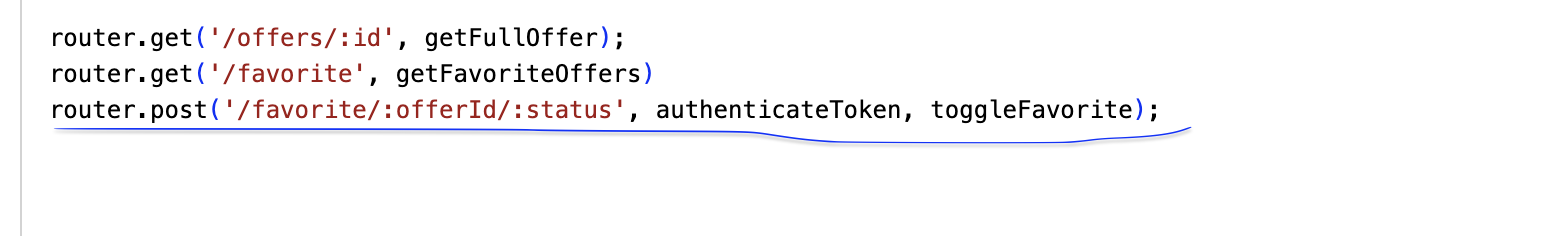
next(ApiError.unauthorized('Недействительный токен'));

}

};

export { authenticateToken };

1. теперь эту функцию можно использовать в тех маршрутах, где действия пользователя подразумевают авторизацию. Такая проверка необходима при добавлении/удалении оффера в/из избранное и при добавлении отзыва. Перейдем в [offerRoutes.js](http://offerroutes.js) и добавим проверку.



и в [reviewRoutes.js](http://reviewroutes.js)



5. Перейдем к созданию функции Проверка статуса аутентификации. В [userController.js](http://usercontroller.js) создаем функцию checkAuth

const checkAuth = (req, res) => {

const user = req.user;

const token = jwt.sign(

{

id: user.id,

email: user.email,

username: user.username,

userType: user.userType,

avatar: user.avatar

},

process.env.JWT\_SECRET,

{ expiresIn: '24h' }

);

return res.json({

id: user.id,

email: user.email,

username: user.username,

avatar: user.avatar,

isPro: user.userType === 'pro',

token

});

};

6. Определим для функции новый маршрут

router.get('/login', authenticateToken, checkAuth);

7. Осталось реализовать последнюю функцию - Выход пользователя. Так как токен не храниться на сервере, значит его нельзя удалить. Удаление токена будет производится на клиенте, но для этого необходимо сформировать запрос на соответствующую ошибку, после которой будет производится удаление. В [userControllers.js](http://userroutes.js) создадим функцию logout



8. Создадим соответствующий маршрут в userRoutes

router.delete('/logout', logout);

### Задание 4. Проверка работоспособности

1. Проверьте работоспособность всех реализованных маршрутов.
2. Внесите данные в БД, где это необходимо.