Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВАЯ РАБОТА**

Разработка REST API приложения для управления данными зоопарка

по дисциплине «Базы данных»

Выполнил

студент гр.3530901/80203

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шрамков М. А.

(подпись)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мяснов А. В.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург   
2021

**Цели работы:**

Систематизация и углубление полученных знаний, самостоятельное изучение избранных вопросов и применение их на практике.

**Программа работы:**

1. Выбор способа реализации курсовой работы
2. Написание и согласование технического задания по курсовой работе с подробным описанием реализуемой функциональности
3. Реализация всей требуемой функциональности
4. Тестирование корректности работы
5. Демонстрация результатов преподавателю
6. Оформление отчета по курсовой работе

**Выбор способа реализации:**

В качестве курсовой работы было выбрано создание REST API web-приложения. Для написания backend-части было выбрано Node.js, а для frontend-части React.js.

**База данных:**

База данных была взята из предыдущих лабораторных работ, однако была добавлена таблица ZooUsers(id, login, password, access\_level) для создания пользователей и прав их доступа.

**Код на github:**

https://github.com/klunkela/rest\_api\_zoo\_backend

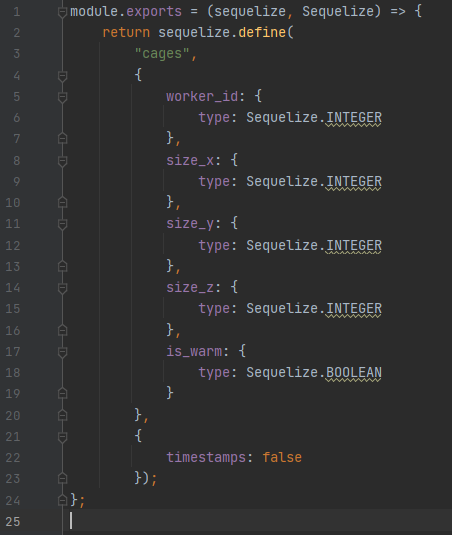
https://github.com/klunkela/rest\_api\_zoo\_frontend

**Ход работы:**

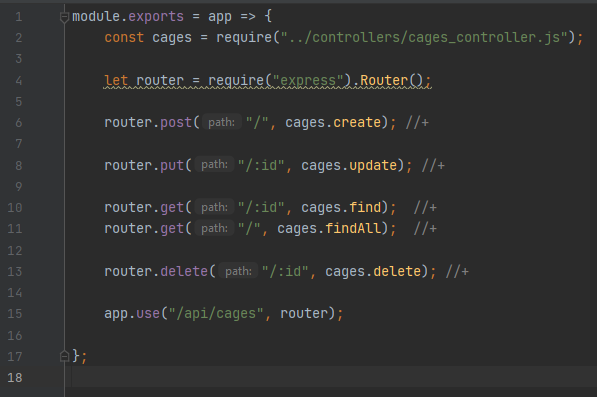
Приложение состоит из главного файла server.js, некоторых вспомогательных файлов, и трех основных директорий с файлами для работы с базами данных.

Первая – models, в которой находятся файлы с моделями таблиц БД. Для связи базы данных с объектами приложения использовалась ORM Sequelize.

Пример одной из моделей - cages:

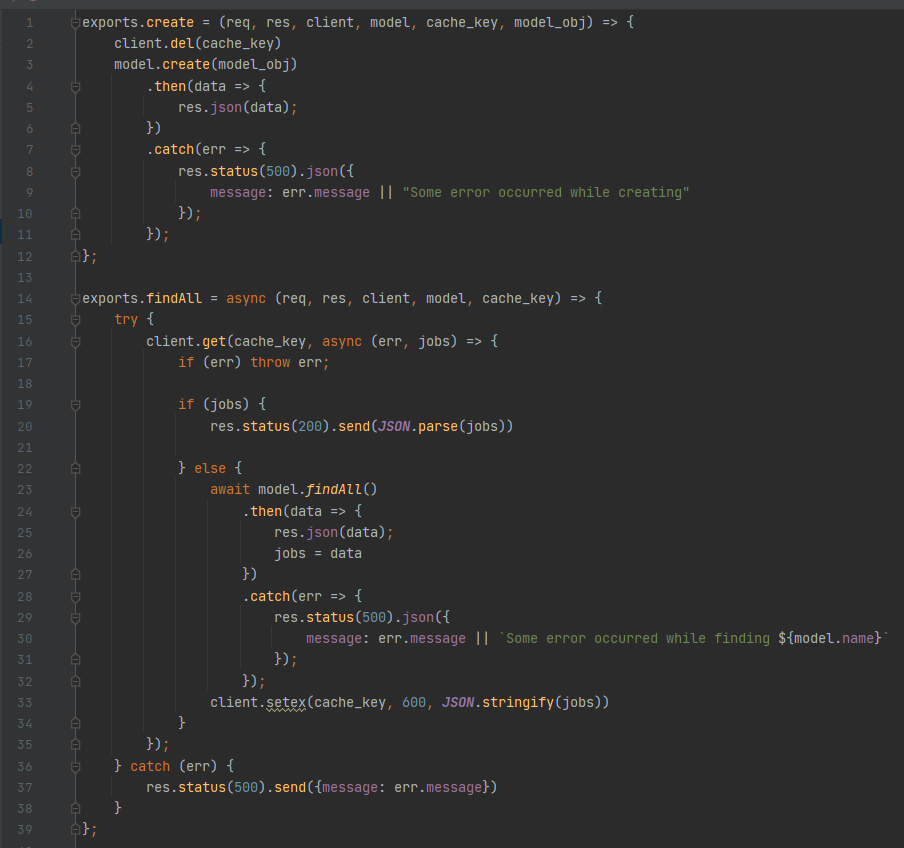
****

Следующей частью является routes, в которой указываются методы запросов, и вызов соответствующих функций у контроллера



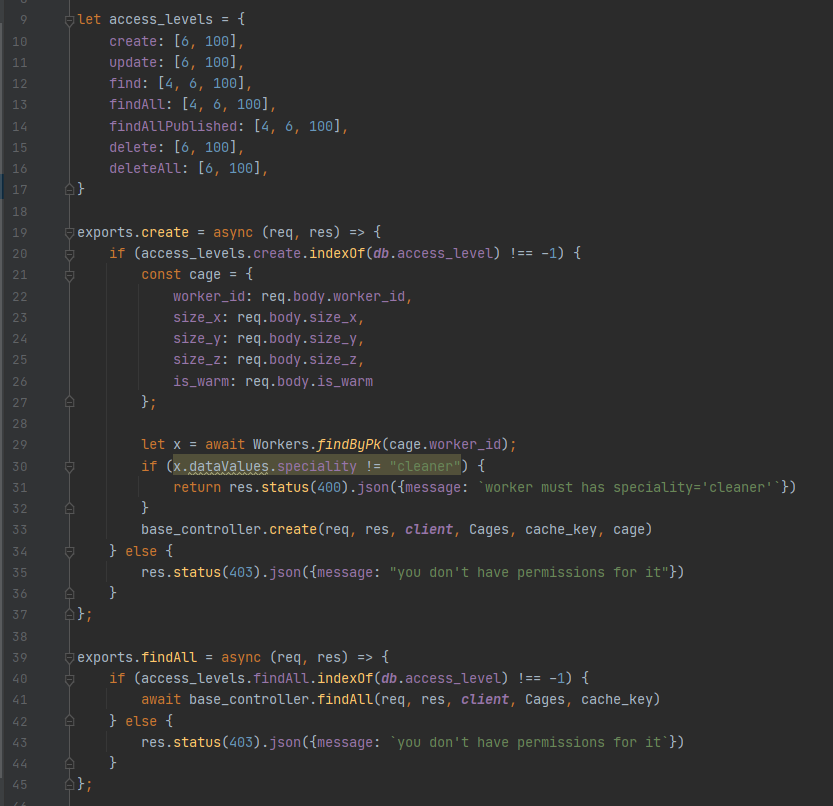
И сами контроллеры, в которых указаны уровни доступа для запросов, и сами запросы. Контроллеры в большинстве методов ссылаются на base\_controller, в котором с помощью ORM идут запросы к БД, обрабатываются ошибки, реализован кэш. Однако в некоторых ситуациях необходимо учитывать дополнительные условия, которые не обработаны в base\_controller’e, тогда не обращаясь к base\_controller’у, производится запрос.

Некоторые основные методы Base\_controllerа:

****

****

Cages\_controller, уровни доступа и некоторые методы:

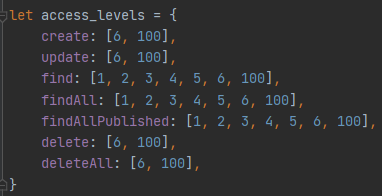
****

**Уровни доступа и авторизация:**

У пользователей реализовано 7 уровней доступа.

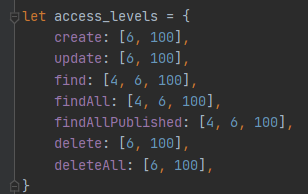
* Уровень 1 для пользователей;
* Уровни 2-6 для рабочих (doctor, cleaner, cook, cashier, manager)
* Уровень 100 для администратора.

Например уровни доступа для animals:



То есть смотреть список животных могут все, а добавлять/изменять/удалять только менеджер и администратор.

А таблицу cages смогут смотреть только менеджер, администратор и рабочие с ролью уборщика:



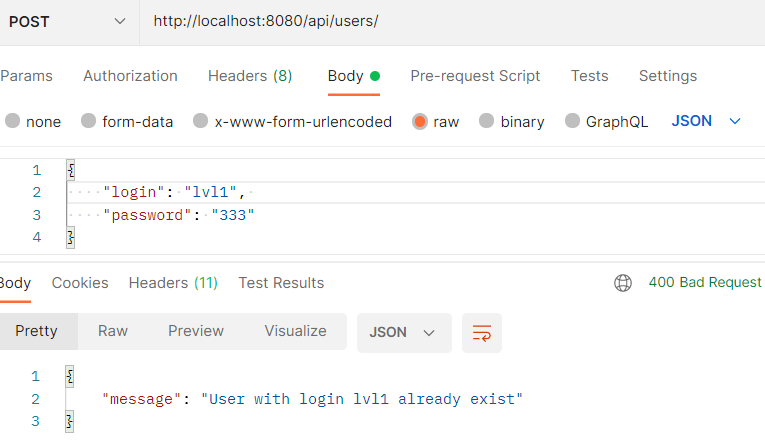
**Авторизация:**

При регистрации хэшируется пароль, а при входе на аккаунт – создается jwt token.

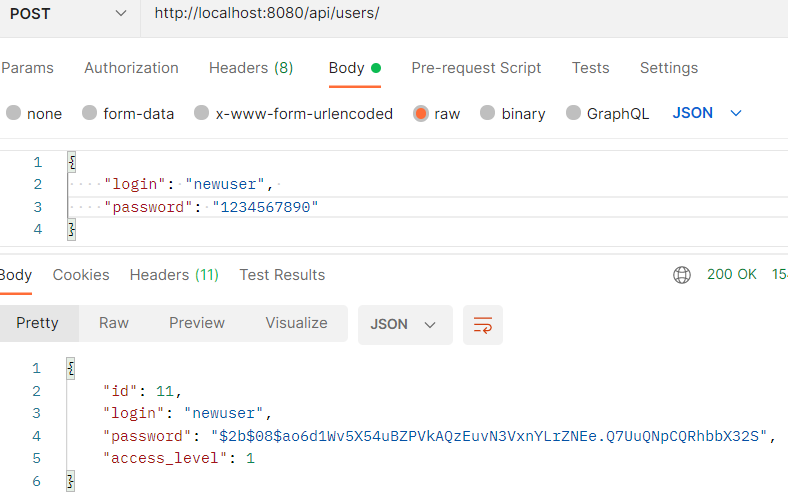
**Результаты работы:**

Рассмотрим результаты запросов в Postman.

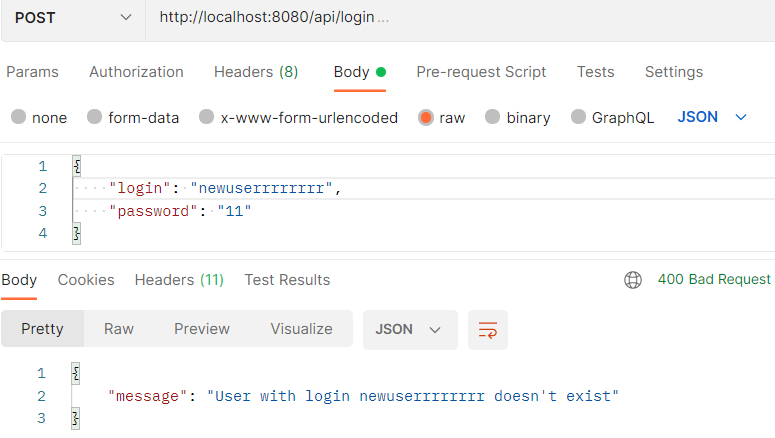
Попытка создать пользователя с уже занятым логином.



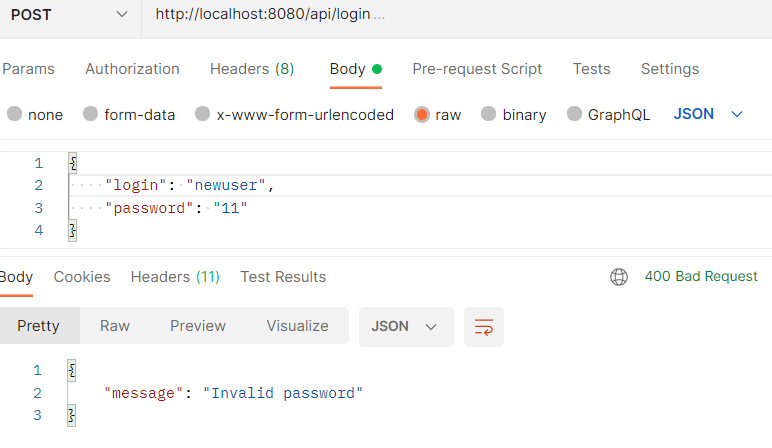
Успешная попытка создать пользователя



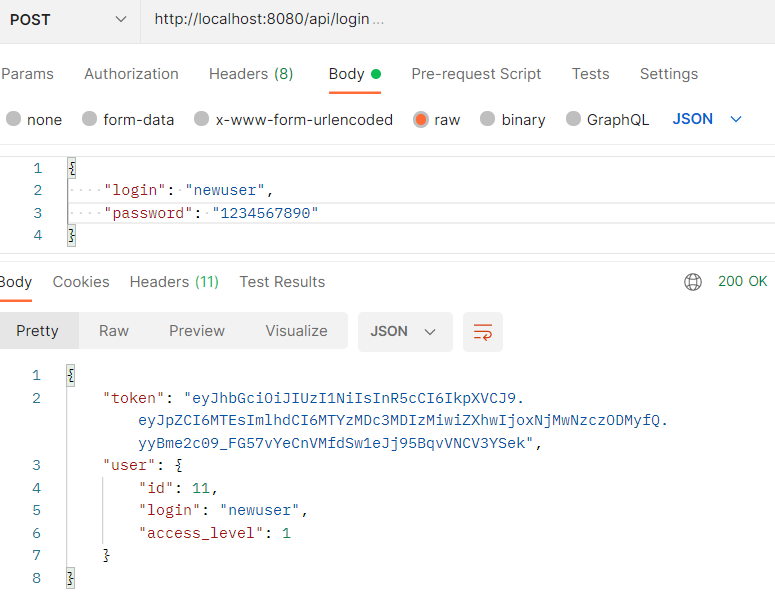
Попытка войти с несуществующим логином:



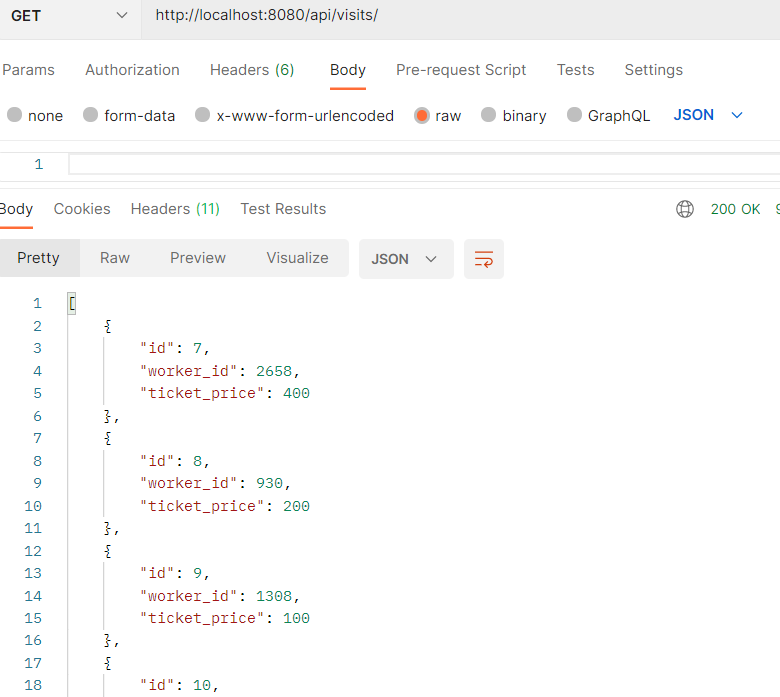
Попытка войти с неправильным паролем:



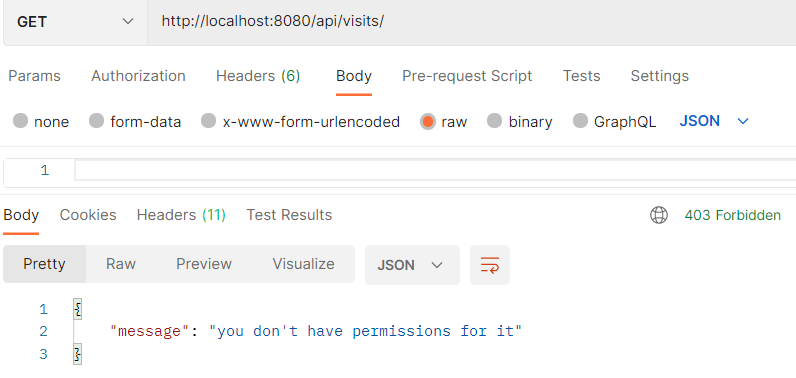
При правильных введенных данных успешно входим в аккаунт:



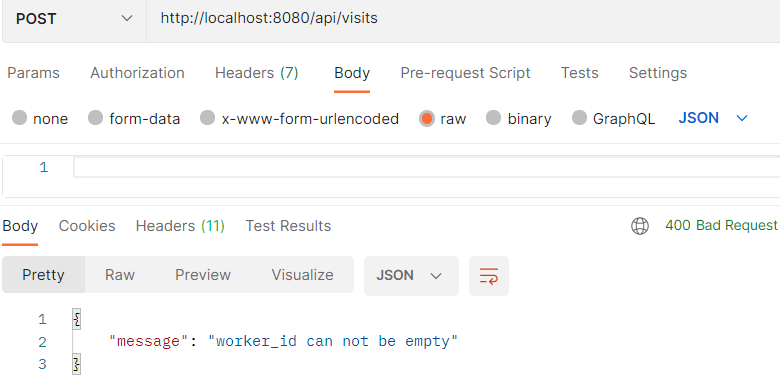
Зайдем на аккаунт администратора и получим все посещения

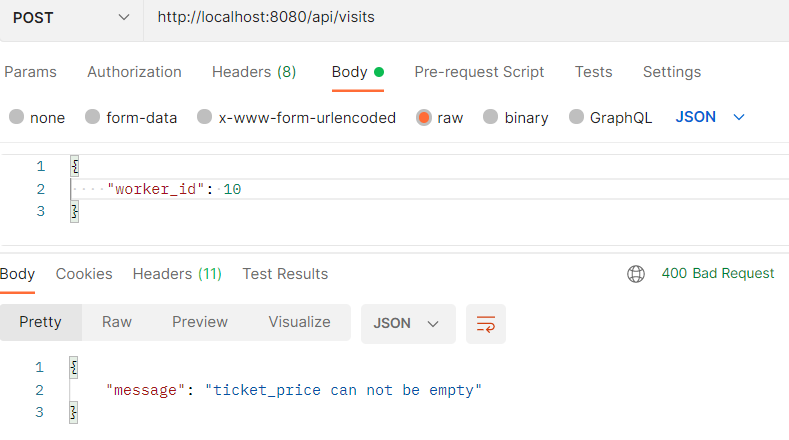


Зайдем на аккаунт пользователя и попробуем сделать то же самое, получим ошибку 403 и сообщение о том, что нет необходимых прав доступа

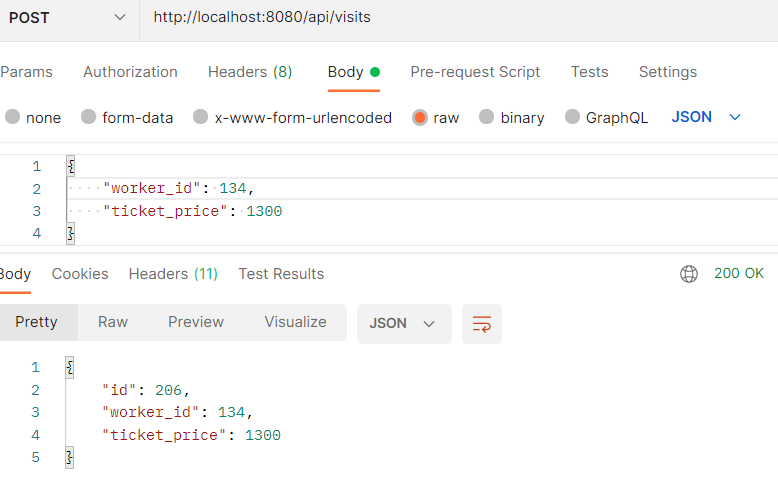


Зайдем на аккаунт администратора и попробуем создать посещение. Видим, что ошибки обрабатываются.

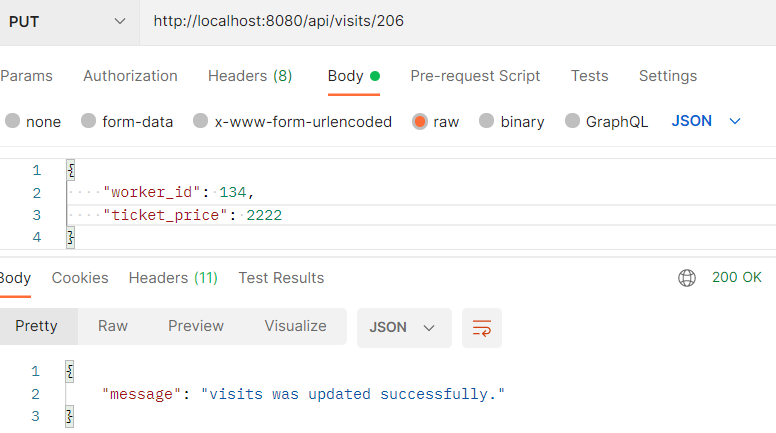




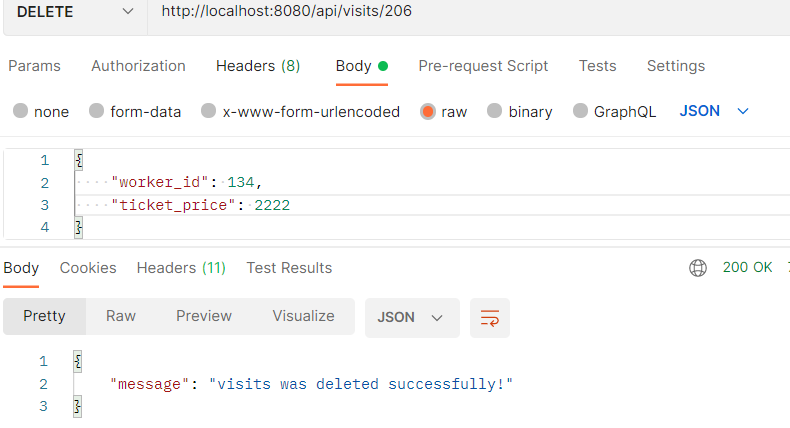
При правильно введенных данных – посещение успешно создается.

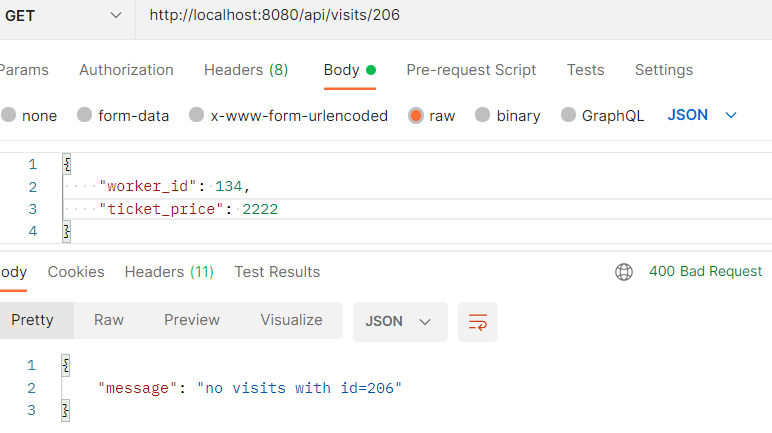


Изменим данные:



Удалим и попробуем получить уже несуществующее посещение:





Как видно, всё работает правильно.

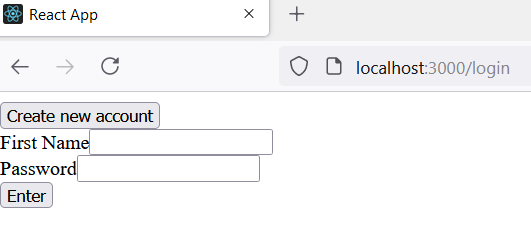
**Front-end**

Front-end часть приложения была реализована не полностью, однако основные элементы были сделаны для обучения и понимания работы. Сделана регистрация и авторизация на сайте, и обработка 2 таблиц. Код для всех таблиц почти что одинаков и его много, поэтому я реализовал только работу с animals и visits. Они имеют разные уровни доступа, так что для всех остальных таблиц создать страницы web-приложения будет несложно.

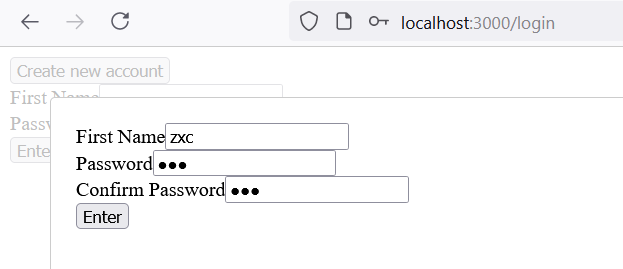
В качестве используемых технологий использовался react и redux.

**Результат**:

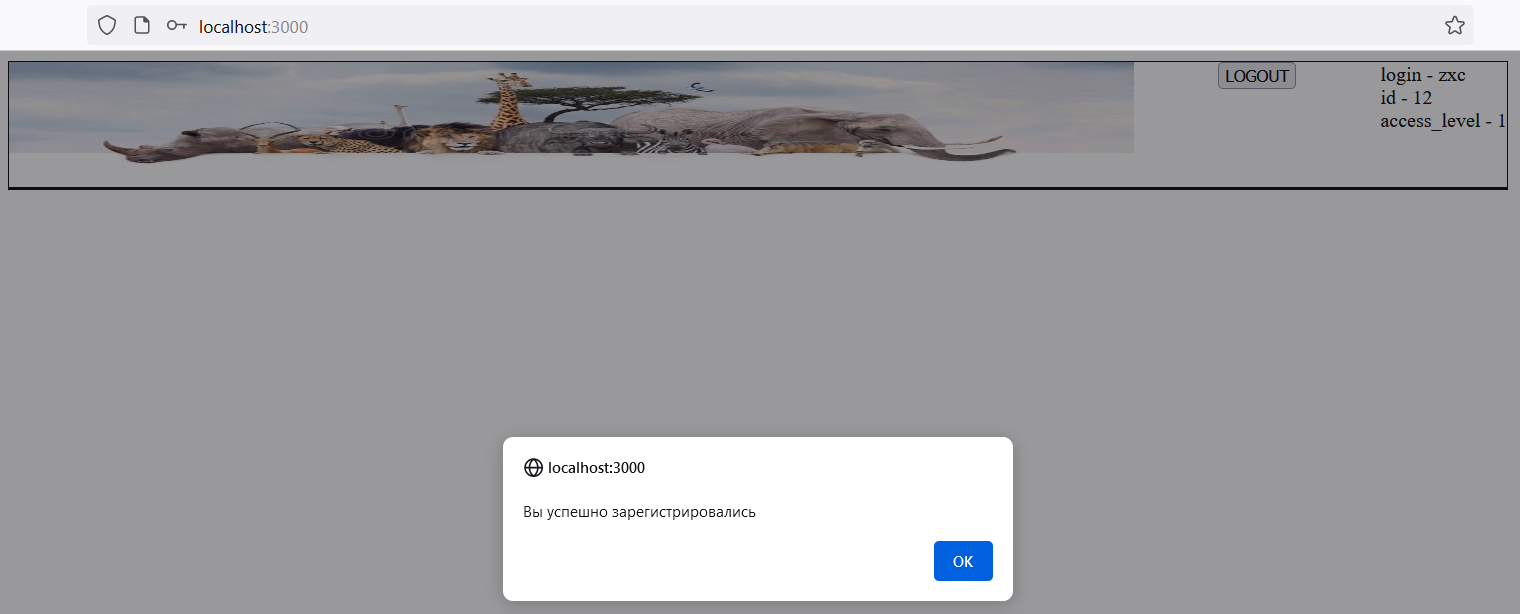
При входе на сайт – попадаем на страницу авторизации



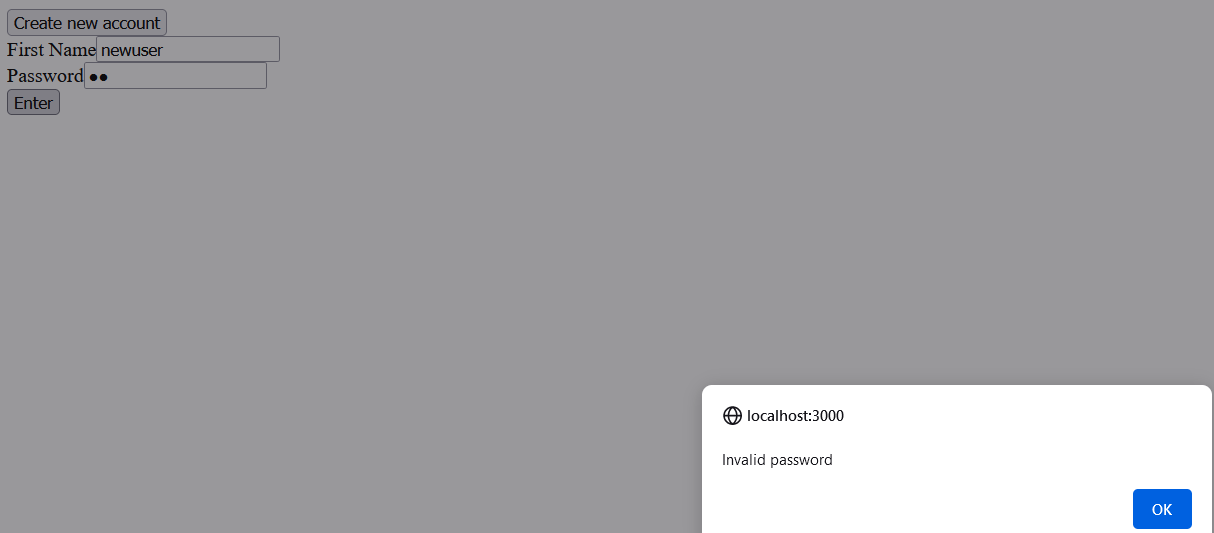
При нажатии на кнопку создания аккаунта – открывается модальное окно:



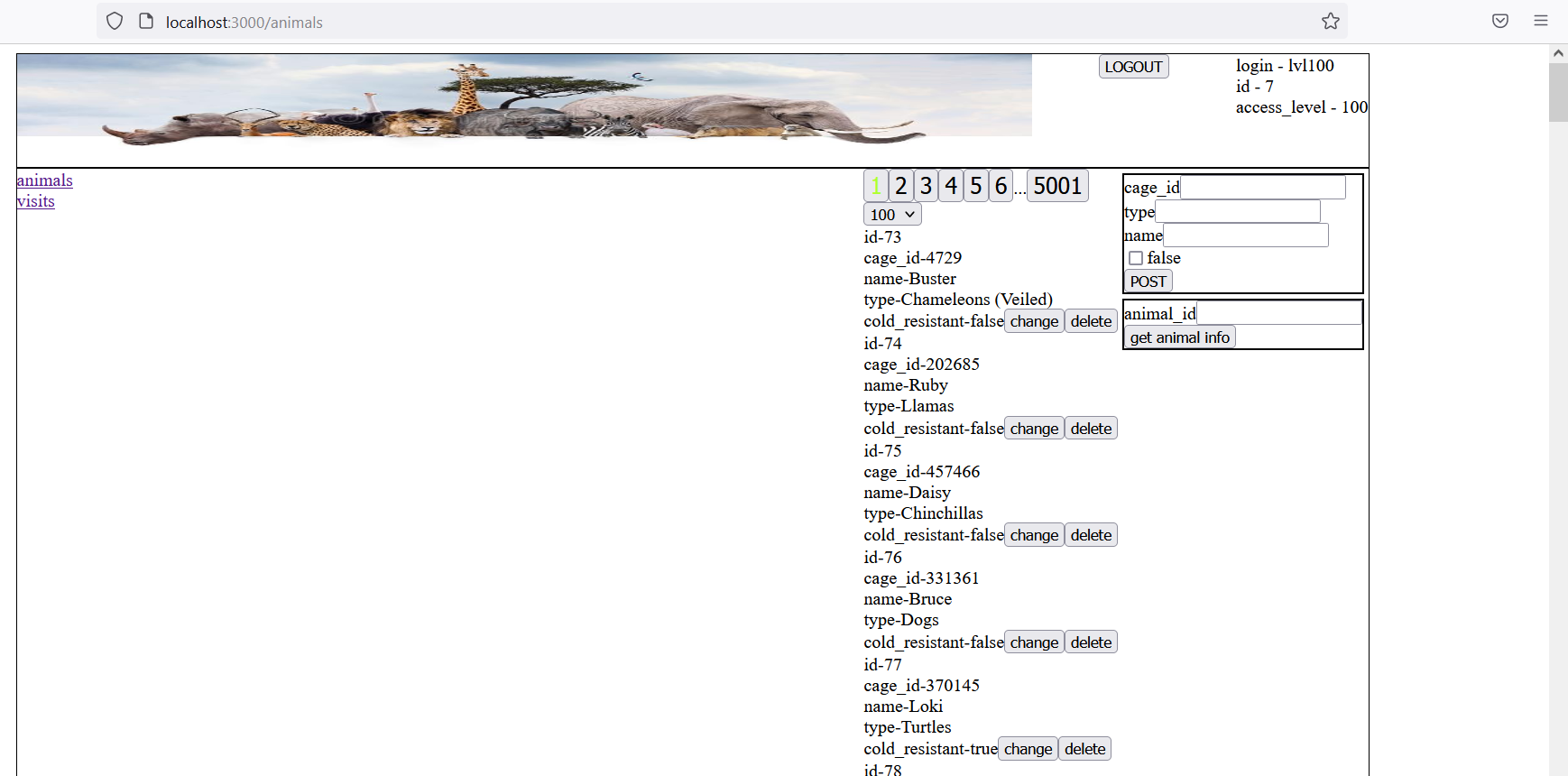
Успешно зарегистрировались (вход в аккаунт происходит сразу):



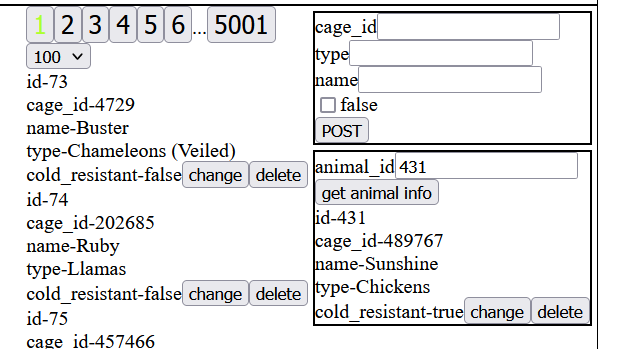
Ошибки при попытке входа – также обрабатываются:

Зайдем от имени администратора на страницу animals:



Попробуем получить данные животного с id=431



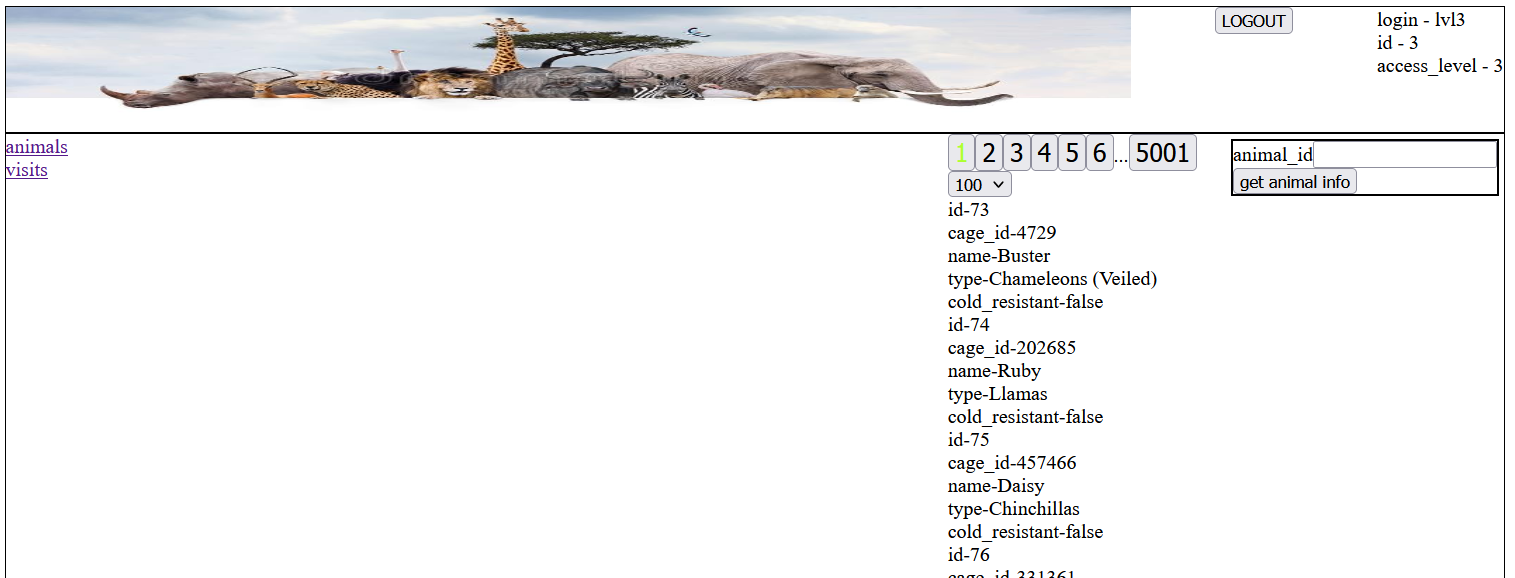
Добавление, изменение и удаление также работают, однако скриншотами это тяжело продемонстрировать.

Зайдем от имени пользователя с правами 1 (обычный пользователь):

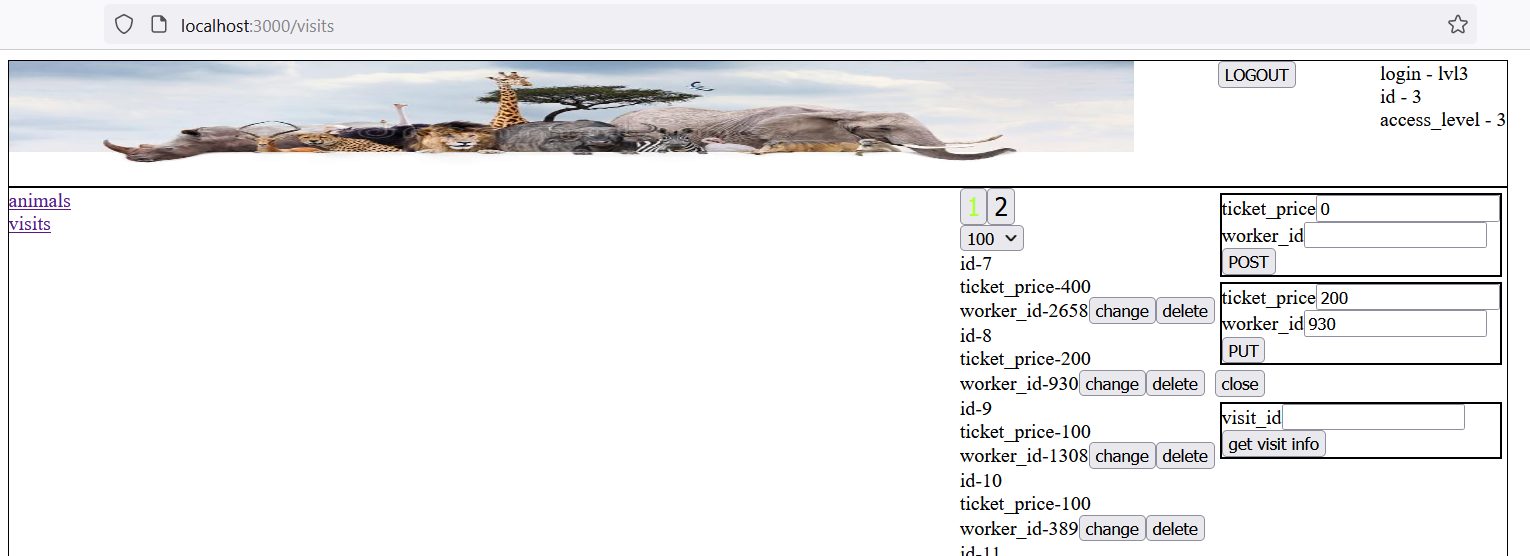


Видим, что он может только посмотреть животных, а страница посещений ему не показывается (при попытке перейти на /visits через url – перекидывает на главную страницу).

Зайдем на аккаунт пользователя 3 (отвечает за посещения). Страница животных, он может только ее посмотреть:



Страница посещений, на ней у него полные права:



**Кэширование**

Кэширование было реализовано с помощью Redis.

При получении данных – они записываются в кэш, при изменении, добавлении, удалении – кэш очищается.

Точные вычисления улучшения производительности не проводились (тем более при написании front-end части – я понял, что лучше отправлять данные «порциями» и немного переделал код), но с помощью Postman можно оценить, что время ожидания результата get-запросов большого объема данных с использованием кэша – уменьшилось примерно в 4 раза.

**Вывод**

Я написал своё первое rest api приложение, ознакомился со многими технологиями, работой с авторизацией, кэшированием.