JS Level 2

open (js)
school

РАБОТА С MYSQL

Работа с MySQL

То, о чём мы должны поговорить - это работа с MySQL. Мы научились получать из него данные, но:

1. Не привязали это к своему веб-сервису
2. Сделали это не очень удобно

Давайте смотреть, как это можно реализовать.

Работа с MySQL

Итак, сервер с наших предыдущих лекций должен выглядеть вот так:

app.js > ...

```
1   'use strict';
2
3   const http = require('http');
4
5   const port = 9999;
6   const statusOk = 200;
7   const statusBadRequest = 400;
8   const statusNotFound = 404;
9
10  let nextId = 1;
11  const posts = [];
12
13  function sendResponse(response, {status = statusOk, headers = {}, body = null}) {…
19  }
20
21  function sendJSON(response, body) {…
28  }
29
30  const methods = new Map();
31  methods.set('/posts.get', function({response}) {…
34  methods.set('/posts.getById', function({response, searchParams}) {…
54  methods.set('/posts.post', function({response, searchParams}) {…
72  methods.set('/posts.edit', function({response, searchParams}) {…
100 methods.set('/posts.delete', function({response, searchParams}) {…
122 methods.set('/posts.restore', function({response, searchParams}) {…
149
150 const server = http.createServer(function(request, response) {…
168
169 server.listen(port);
```

Работа с MySQL

До этого мы работали с вами с сессиями, но в веб-приложении мы будем использовать новую сущность - client. Фактически, это тот, кто будет предоставлять нам сессии:

```
13  const client = mysqlx.getClient({
14    user: 'app',
15    password: 'pass',
16    host: '0.0.0.0',
17    port: 33060
18  });
```

Работа с MySQL

Помимо этого нам понадобится вспомогательные функция:

```
38  function map(columns) {
39    return row => row.reduce((res, value, i) => ({...res, [columns[i].getColumnLabel()]: value}), {});
40  }
```

map умеет вместо массивов (если вы помните, select нам возвращал массив массивов) делать красивые объекты, где названия свойств - это имена столбцов, а значения свойств - это значения столбцов.

Работа с MySQL

Давайте посмотрим, как она используются:

```
44  methods.set('/posts.get', async ({response, db}) => {
45    const table = await db.getTable('posts');
46    const result = await table.select(['id', 'content', 'likes', 'created'])
47      .orderBy('created DESC')
48      .execute();
49
50    const data = result.fetchAll();
51    result.getAffectedItemsCount()
52    const columns = result.getColumns();
53    const posts = data.map(map(columns));
54    sendJSON(response, posts);
55  });
```

Самая интересная строка - это 53. Фактически, вызов функции map(columns) возвращает нам функцию, которая каждую строку базы данных (а это массив) превращает в объект с использованием функции reduce. Вы можете использовать этот код "как есть", либо обратиться к лекциям JS01 или MDN.

Работа с MySQL

Но не всё так просто, если вы заметили, то функцию мы сделали асинхронной, а try-catch для возникающих rejection'ов Promise не написали. Почему?

Потому что мы слегка переделали нашу функцию, которая занимается приёмом запросов (см. следующий слайд):

```
183  const server = http.createServer(async (request, response) => {
184    const {pathname, searchParams} = new URL(request.url, `http://${request.headers.host}`);
185
186    const method = methods.get(pathname);
187    if (method === undefined) {
188      sendResponse(response, {status: statusNotFound});
189      return;
190    }
191
192    let session = null;
193    try {
194      session = await client.getSession();
195      const db = await session.getSchema(schema);
196
197      const params = {
198        request,
199        response,
200        pathname,
201        searchParams,
202        db,
203      };
204
205      await method(params);
206    } catch (e) {
207      sendResponse(response, {status: statusInternalServerError});
208    } finally {
209      if (session !== null) {
210        try {
211          await session.close();
212        } catch (e) {
213          console.log(e);
214        }
215      }
216    }
217  });
```

Работа с MySQL

Ну и, конечно же, мы не забыли добавить новых констант:

```
6   const port = process.env.PORT || 9999;
7   const statusOk = 200;
8   const statusNoContent = 204;
9   const statusBadRequest = 400;
10  const statusNotFound = 404;
11  const statusInternalServerError = 500;
12  const schema = 'social';
```

500 - это код ошибки сервера

а social - это имя нашей базы данных

На этом локальная настройка работы с БД завершена. В качестве подсказок скажем вам, что при вставке записей можно использовать функцию getAutoIncrementValue (вернёт сгенерированный id), а при обновлении и удалении - getAffectedItemsCount (вернёт количество обновлённых/удалённых строк).

Работа с MySQL

```
126  methods.set('/posts.delete', async ({response, searchParams, db}) => {
127    if (!searchParams.has('id')) {
128      sendResponse(response, {status: statusBadRequest});
129      return;
130    }
131
132    const id = Number(searchParams.get('id'));
133    if (Number.isNaN(id)) {
134      sendResponse(response, {status: statusBadRequest});
135      return;
136    }
137
138    const table = await db.getTable('posts');
139    const result = await table.update()
140      .set('removed', true)
141      .where('id = :id')
142      .bind('id', id)
143      .execute();
144
145    const removed = result.getAffectedItemsCount();
146
147    if (removed === 0) {
148      sendResponse(response, {status: statusNotFound});
149      return;
150    }
151
152    sendResponse(response, {status: statusNoContent});
153  });
```

Работа с MySQL

Но при таком подходе мы получаем только количество удалённых записей, но не сами записи.

При этом возвращаем 404, если запись не найдена, либо 204 - значение константы statusNoContent, если мы удалили запись.

Что можно сделать? Можно перед удалением делать select и только потом update. С обновлением то же самое. Это вам придётся реализовать в ДЗ.

MySQL

Полный код приложения вы найдетё по адресу https://lms.openjs.io/08.zip.

ИТОГИ

Итоги

В этой лекции мы обсудили использование MySQL совместно с нашим небольшим веб-приложением. Эти знания позволят вам самостоятельно создавать бэкенд для своих фронтенд-приложений для разработки и тестирования.

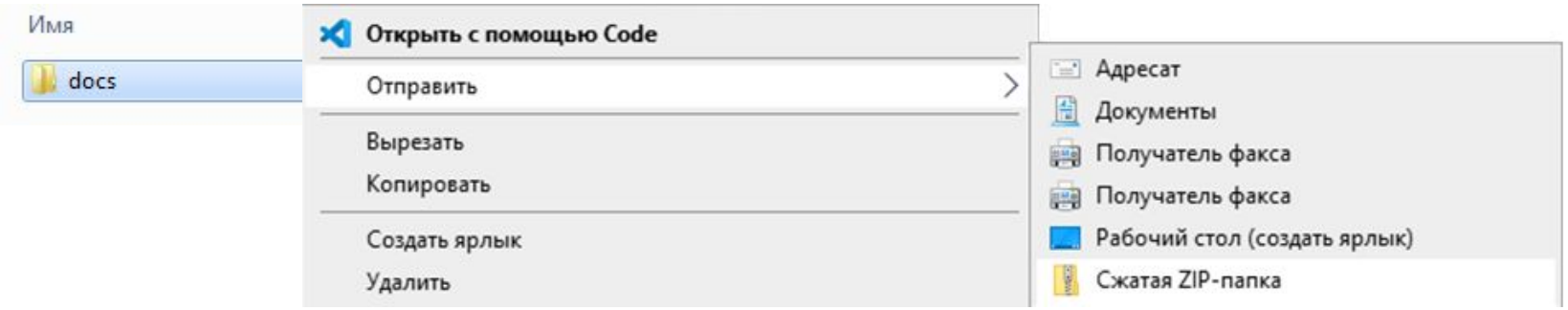
Конечно же, это не всё и в доп.руководстве вас ждут бонусные темы из всего, что мы заявляли в рамках курса - поэтому ждите! Сегодня мы опубликуем руководство по развёртыванию вашего сервиса на Heroku через Docker, а дальше вас ждут:

1. NPM - публикация библиотек
2. fetch, Promise, async/await в браузере (будем использовать наш же backend)
3. Jest - будем писать автотесты
4. Webpack - соберём готовое приложение и выложим на Github
5. REST API
6. Аудио и видео, загрузка файлов

ДОМАШНЕЕ ЗАДАНИЕ

Как сдавать ДЗ

Вам нужно запаковать в zip-архив ваш каталог с проектом (не содержимое каталога, а сам каталог) - выделяете его и выбираете Отправить –> Сжатая ZIP-папка:



Полученный архив загружаете в личном кабинете пользователя.

Важно: учитывается только последняя отправленная попытка.

Как сдавать ДЗ

Важно: каталог node_modules загружать не нужно!!! Скопируйте все ваши файлы без него, заархивируйте и отправьте.

Как сдавать ДЗ

Важно: используйте в параметрах подключения те данные, что указаны в лекции (иначе вы не сможете подключиться к той базе, что у бота).

posts

Что нужно сделать - это разработать схему базы данных (CREATE TABLE) и реализовать все методы из предыдущего ДЗ теперь уже подключаясь к базе данных из обработчиков.

CREATE TABLE разместите в файле schema.sql, а все запросы - в коде вашего сервера.

Задания

Далее выполните следующие задания:

1. /posts.get - запрос SELECT, который выбирает столбцы id, content, likes, created только для тех строк, у которых removed = FALSE.
2. /posts.getById - запрос, который выбирает поля id, content, likes, created для поста с переданным id при условии, что он не удалён.
3. /posts.post - запрос на создание поста со значениями:
   a. id (идентификатор) - автоматически
   b. content (текст поста) - присланный
   c. likes (количество лайков) - по умолчанию 0
   d. removed (удаленный или нет) - по умолчанию
   e. created (дата создания) - по умолчанию

   Запрос возвращает удалённый пост (те же поля, что в getById).

Задания

4. /posts.edit - запрос, который обновляет пост с переданным id, выставляя content равный переданному при условии, что пост не удалён. Запрос возвращает обновлённый пост (те же поля, что в getById).
5. /posts.delete - запрос, который выставляет removed в TRUE для поста с переданным id при условии, что он ещё не удалён. Запрос возвращает удалённый пост (те же поля, что в getById).
6. /posts.restore - запрос, который выставляет removed в FALSE для поста с переданным id при условии, что он уже удалён. Запрос возвращает удалённый пост (те же поля, что в getById).
7. /posts.like и /posts.dislike - запрос, который добавляет или удаляет один лайк посту, соответственно. Лайкать можно только не удалённые посты. Запрос возвращает удалённый пост (те же поля, что в getById).

Задания

И последнее задание - это публикация вашего сервера на Heroku. Про него вы прочитаете в доп.руководстве.

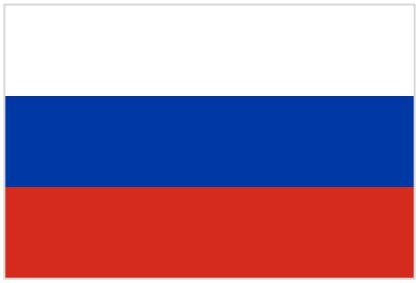
Спасибо за внимание

alif academy совместно с aims

2020г.

open (js)
school

Данный курс проводится в рамках инициативы #InnoResponseChallenge проекта ПРООН «Молодежь для бизнеса и инноваций» при финансовой поддержке Правительства Российской Федерации



При финансовой поддержке Российской Федерации

UNDP

Empowered lives.
Resilient nations.

open (js) school