Лабораторная работа №4

Создание и подключение базы данных к веб-приложению. Разработка REST-методов взаимодействия с базой данных веб-приложения.

Задание:

1) Создать базу данных (БД) SQLite с таблицей, которая будет хранить данные, отправляемые с веб-формы вашего приложения. Таблица должна иметь не менее 4-х полей. Процедура создания таблицы должна быть сохранена в отдельный скрипт *.sql.

2) Подключить БД к приложению и реализовать CRUD-методы для работы с таблицей, т.е. методы должны использовать следующие типы SQL-запросов: SELECT (с фильтром и без), INSERT, UPDATE, DELETE.

3) Разработать REST API и привязать маршруты (routes) вашего веб-приложения к соответствующим CRUD-методам. При отправке данных с вашей веб-формы методом POST должны добавляться данные в таблицу, используйте отправку данных на основе подхода AJAX из ЛР №3.

4) Провести тестирование разработанного REST API с помощью любого REST-клиента.

Теоретический материал:

1) Понятие CRUD-методов, особенности БД SQLite (основные преимущества и ограничения).

2) Ознакомиться с протоколом HTTP (формат передачи данных, основные коды возвращаемых ошибок).

3) Ознакомиться с понятием REST API (наиболее часто используемые HTTP-методы, понятие URI, клиент-серверное взаимодействие).

Рекомендуемое программное обеспечение:

1) DBeaver CE – универсальный клиент для работы с БД:

https://dbeaver.io/files/dbeaver-ce-latest-x86_64-setup.exe

2) **Insomnia** или **Postman**- утилиты для тестирования REST API:

https://insomnia.rest/download

Методические рекомендации

1. Создание БД SQLite.

Если в разрабатываемом веб-приложении необходимо использовать БД, то реализация спроектированной структуры (реляционной) БД возможна следующими способами:

1) С помощью языка SQL.

2) С помощью технологии «объектно-реляционного отображения» - Object-Relational Mapping (ORM), которая используется в современных языках программирования, поддерживающих объектно-ориентированный подход (Python, Node.js, Java, C# и т.п.).

В рамках данного практического курса рассмотрим первый вариант реализации структуры БД.

БД SQLite является встраиваемой БД, часто применяемой при локальном тестировании приложений в процессе разработки, а также нередко используется в рабочих решениях в случае, когда основная масса запросов к БД – это запросы на чтение.

Для инициализации БД SQLite создайте пустой текстовый файл и измените его имя и расширение, например: **`appdb.sqlite`** . В менеджере БД DBeaver откройте выпадающее меню для нового подключения и выберите подключение к SQLite:

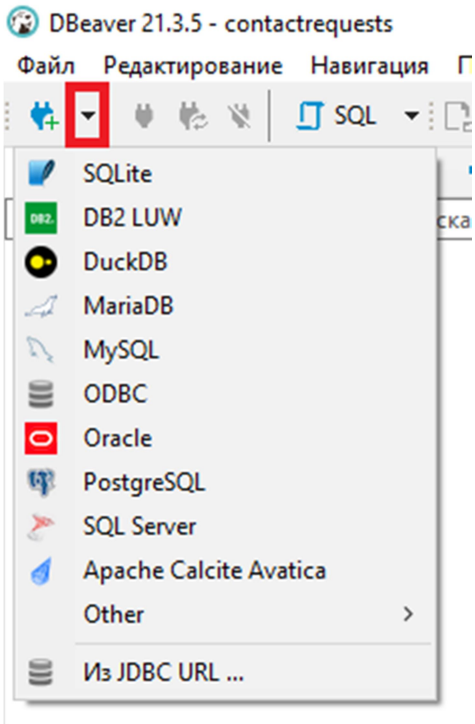


Рис. 1 – Выбор подключения в DBeaver

В окне подключения нажмите «Найти» и выберите созданный файл БД:

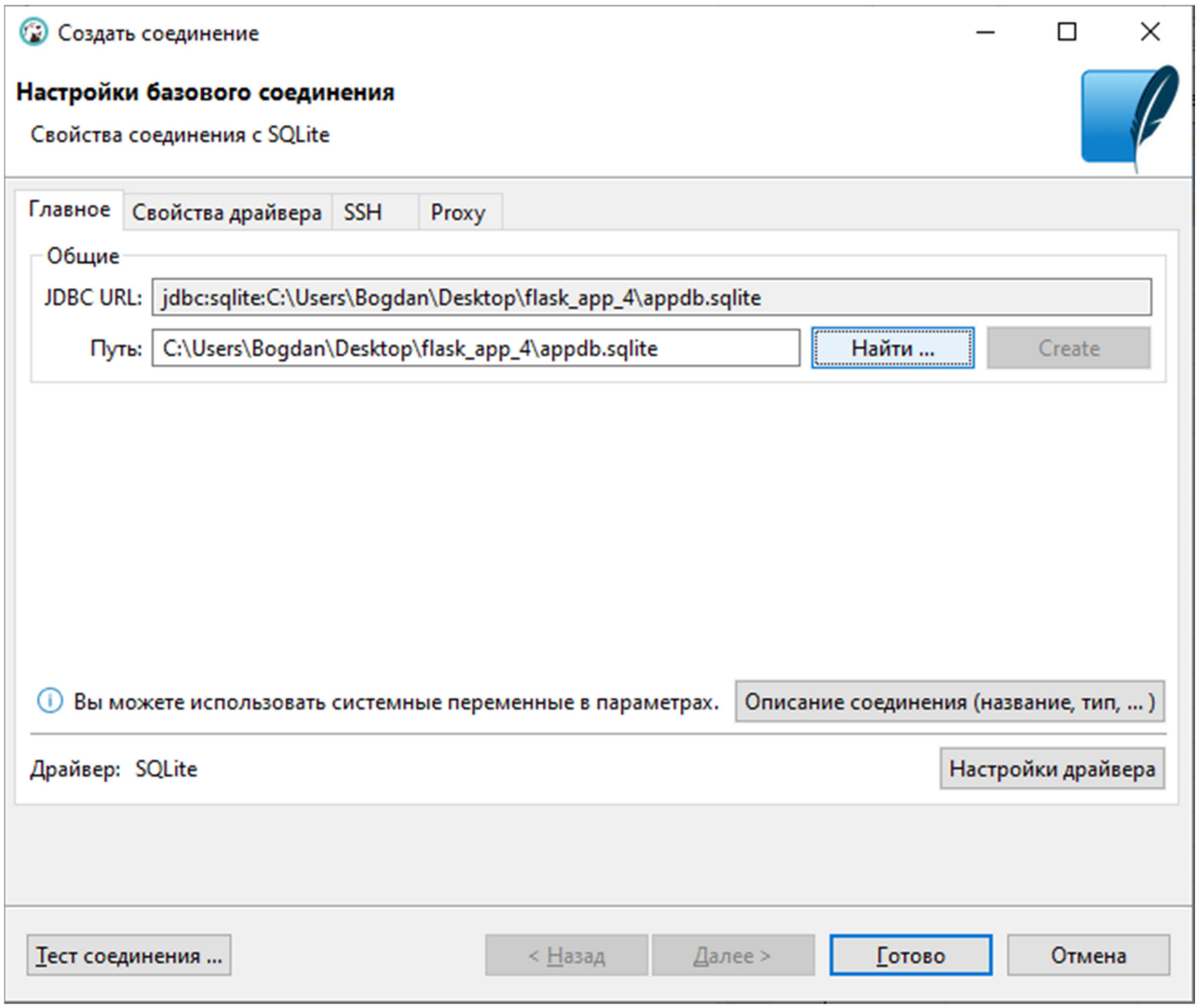


Рис. 2 – Настройка соединения с БД SQLite

После подтверждения клиент (возможно) автоматически предложить скачать драйвер для установления соединения. После установки соединения в списке соединений появится ваша БД. Выделите вашу БД и нажмите кнопку создания скрипта SQL на панели инструментов:

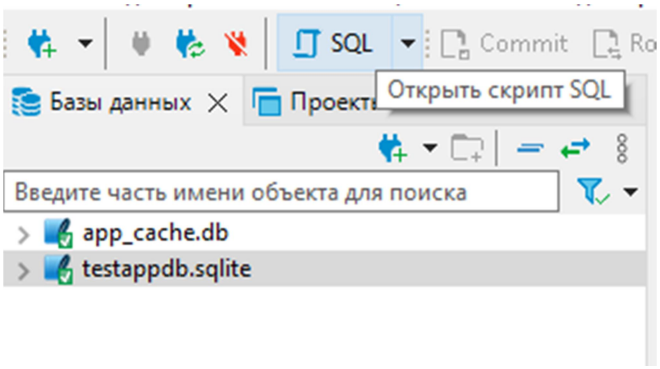


Рис. 3 – Запуск редактора SQL

В окне редактора SQL введите инструкцию для создания таблицы БД. Структура таблицы БД должна соответствовать веб-форме для отправки данных на одной из страниц вашего приложения. Например, на странице CONTACT реализована следующая веб-форма:

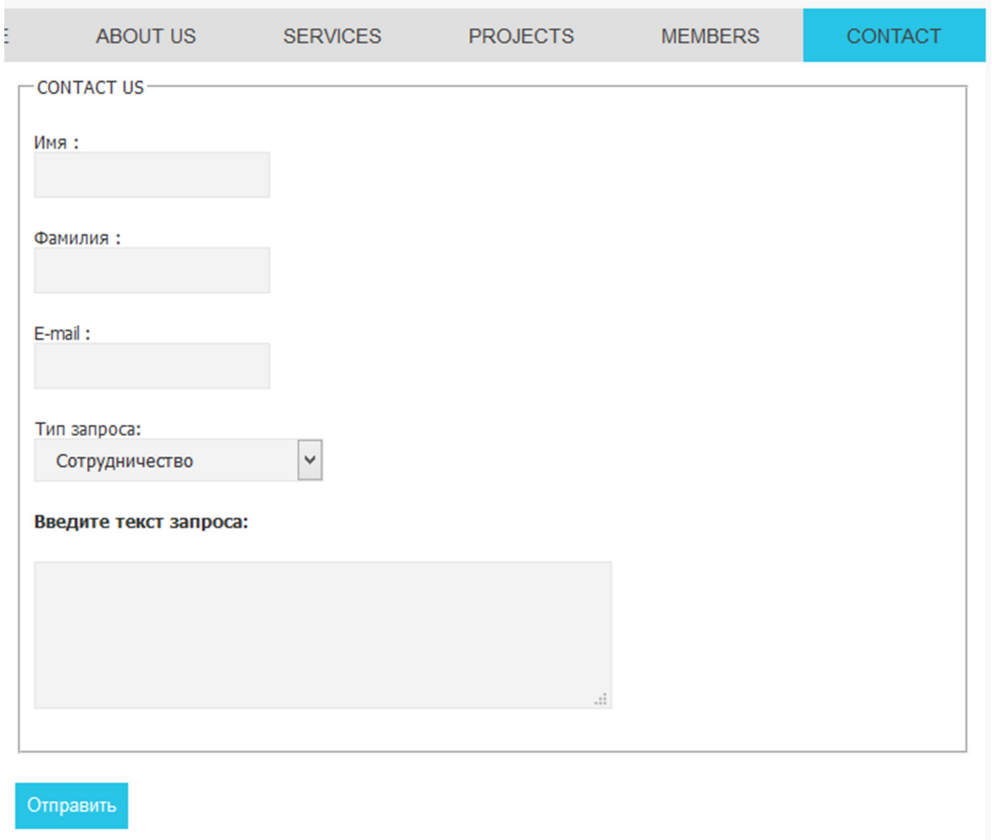


Рис. 4 – Форма создания нового запроса

Тогда инструкция для создания таблицы может иметь следующий вид:

```
create table contactrequests (
      id integer PRIMARY KEY autoincrement,
      firstname varchar(255) NOT NULL,
      lastname varchar(255),
      email varchar(255),
      reqtype varchar(255),
      reqtext varchar(255),
      cratedAt datetime,
      updatedAt datetime
 );
```

Для выполнения скрипта, используйте кнопки слева от поля редактирования. После выполнения скрипта, выберите вашу БД в списке соединения и нажмите F5 для обновления данных. После обновления должна появиться возможность раскрыть структуру БД и просмотреть поля таблицы:

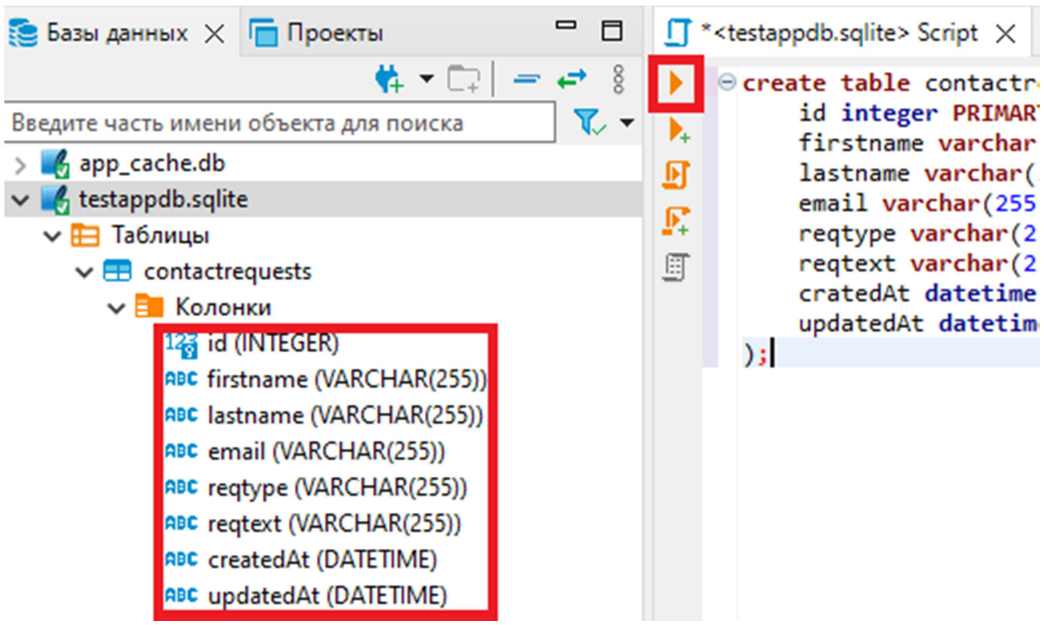


Рис. 5 – Выполнение SQL-скрипта и проверка структуры БД

2. Подключение БД в проекте веб-приложения.

Для подключения БД в **Node.js Express** можно воспользоваться популярным фреймворком для работы с БД **Sequelize**. Данный framework предоставляет универсальные методы (как на основе простых SQL-запросов, так и с помощью ORM) для работы со всеми популярными реляционными БД, в т.ч. MSSQL (подробнее см. документацию: **https://sequelize.org/master/manual**).

Для установки Sequlize, заходим в корневую папку вашего приложения и выполняем:

```
npm install --save sequelize
```

Для работы с той или иной БД понадобится установка «диалекта» (библиотеки для работы с БД). Например, для MySQL:

```
npm install --save mysql2
```

Для SQLite:

```
npm install --save sqlite3
```

Пример кода с реализацией подключения к БД:

```
const Sequelize = require('sequelize');   // подключаем пакет sequalize

// Подключение к БД SQLite
const dbcontext = new Sequelize({
  dialect: "sqlite",
```

```
  storage: "appdb.sqlite"
});

// Экспорт подключения к БД для использования в других модулях
module.exports = {
    dbcontext
}
```

3. Реализация CRUD-методов.

Создайте папку **/controllers** в корне вашего проекта. В данной папке будут располагаться файлы с кодом функций, которые **реализуют CRUD методы** для работы с вашей таблицей: создание (create), чтение (read), модификация (update), удаление (delete). Хорошей практикой считается создание под каждую модель отдельного контроллера с соответствующим названием. Например, если у нас есть таблица «books», то создаем для нее контроллер controllers/bookController.js. В рамках данной лабораторной работы используется только одна таблица, поэтому можно использовать общий контроллер (**см. пример приложения**) **/models/mainController.js**.

Пример функции, реализующей чтение записи из БД по идентификатору:

```
// Показать запрос по id (primary key).
exports.get_contact_req_by_id = function(req, res) {
    dbcontext.query(
        'SELECT * FROM contactrequests WHERE id = :id',
        {
            replacements: { id: req.params.id },
            type: dbcontext.QueryTypes.SELECT
        }
    )
    .then(data => {
      res.json(data[0]);
    })
    .catch(err => {
      res.status(500).json({ message: err.message });
    });
};
```

Данная функция выполняет поиск в БД по основному ключу (primary key) записи. Выполнение SQL-запроса происходит с помощью вызова метода query(…). Метод принимает 2 основных аргумента:

- **replacements** – объект в формате «ключ: значение», с помощью которого в SQL-выражение будет подставлено значение

соответствующего ключа (например, вместо «:id» будет подставлено значение req.param.id – параметр, получаемый из запроса.)

- **type** – тип SQL-выражения (SELECT, INSERT и т.п.)

Результатом выполнения query будет массив (data), содержащий 2 значения:

data[0] – непосредственно данные, полученный из БД;

data[1] – метаданные (например, количество полученных/измененных строк и т.п.).

Полученные данные конвертируются в json-формат, используя метод `.then(data => {res.json(data[0]);}` , и возвращаются в качестве результата.

Если в процессе поиска записи возникла ошибка, то перехватываем ее с помощью инструкции **`catch`** и в ответ направляем текст ошибки в json-формате **`({ message: err.message })`** с HTTP-кодом 500 (Internal Server Error).

4. Реализация REST API.

REST (Representational State Transfer) – это модель взаимодействия клиент-серверного приложения в сети по протоколу **HTTP**.

API (Application Programming Interface – программный интерфейс приложения) – описание классов, процедур, функций и методов взаимодействия между приложениями. Проще говоря, это «язык общения» между приложениями.

Данная модель взаимодействия позволяет осуществлять вызов удаленных процедур (методов) web-приложения для взаимодействия с ресурсами данного приложения. Вызов данных методов осуществляется с помощью структурированных (унифицированных) адресов, которые обозначаются аббревиатурой **URI (Uniform Resource Identifier)**.

Web-приложение, использующее для предоставления своих ресурсов REST-модель взаимодействия, называется **RESTful веб-сервис**. Такое приложение использует специальные методы протокола HTTP и соответствующие структурированные веб-адреса.

В **табл. 1** представлены основные методы HTTP для взаимодействия с RESTful веб-сервисом и URI для предоставления ресурсов веб-приложения из примера.

Таблица №1 – Описание REST API веб-приложения из примера

Метод HTTP	Действие	Пример URI
GET	Получить информацию о всех запросах	http://127.0.0.1:3000/api/contactrequest
GET	Получить информацию о запросе по id	http://127.0.0.1:3000/api/contactrequest/123 (информация о запросе №123)
GET	Получить информацию о всех запросах, созданных определённым автором	http://127.0.0.1:3000/api/contactrequest/author/Tom (информация о всех запросах, созданных автором с именем Tom)
POST	Создать новый запрос	http://127.0.0.1:3000/api/contactrequest (создать новый запрос из json-данных переданных с запросом)
PUT	Обновить запрос	http://127.0.0.1:3000/api/contactrequest/123 (обновить запрос №123 json-данными переданными с запросом)
DELETE	Удалить запрос	http://127.0.0.1:3000/api/contactrequest/123 (удалить запрос №123)

Таким образом, данную REST-модель взаимодействия можно использовать для вызова соответствующих CRUD-методов вашей модели через HTTP-протокол.

5. Связывание CRUD-операций с REST API веб-приложения.

Базовые принципы подключения маршрутов в веб-приложении Node.js были изложены в методических рекомендациях к лабораторной работе №3. Для подключения маршрутов к вашим CRUD-методам, созданных в контроллере, создайте одноименный файл в папке **/routes**.

Рассмотрим код роутера **/routes/contactrequest.js** из примера:

```
router.get('/', mainController.get_contact_req_all);
```

Обрабатывает **GET**-запрос по адресу:

http://127.0.0.1:3000/api/contactrequest

и вызывается метод контроллера **get_contact_req_all(req, resp)**, который возвращает все записи из БД.

```
router.get('/:id', mainController.get_contact_req_by_id);
```

Обрабатывает **GET**-запрос по адресу:

http://127.0.0.1:3000/api/contactrequest/1

Последняя часть адреса (**/:id**) читается в функции контроллера с помощью инструкции **req.params.id**.

```
router.get('/author/:firstname', mainController.get_contact_req_by_firstname);
```

Обрабатывает **GET**-запрос по адресу:

http://127.0.0.1:3000/api/contactrequest/author/Tom

В данном примере в качестве параметра запроса (**req**) в функцию контроллера будет передано значение «Tom» (**req.params.firstname** == “Tom”).

Также дополнительные параметры можно передавать с помощью следующей строки запроса:

http://127.0.0.1:3000/api/contactrequest/author?firstname=Tom

Тогда в функции контроллера параметр firstname можно получить, используя инструкцию **req.query.firstname**.

```
router.post('/', mainController.create_contact_req);
```

Обрабатывает **POST**-запрос по адресу:

http://127.0.0.1:3000/api/contactrequest

Здесь функция контроллера принимает json-данные, читает их, используя инструкцию **req.body.<наименование_поля>**, по прочитанным json-данным создает объект модели БД и записывает в соответствующую таблицу.

Аналогично работает обработка маршрутов:

```
router.put('/:id', mainController.update_contact_req_by_id);
router.delete('/:id', mainController.delete_contact_req_by_id);
```

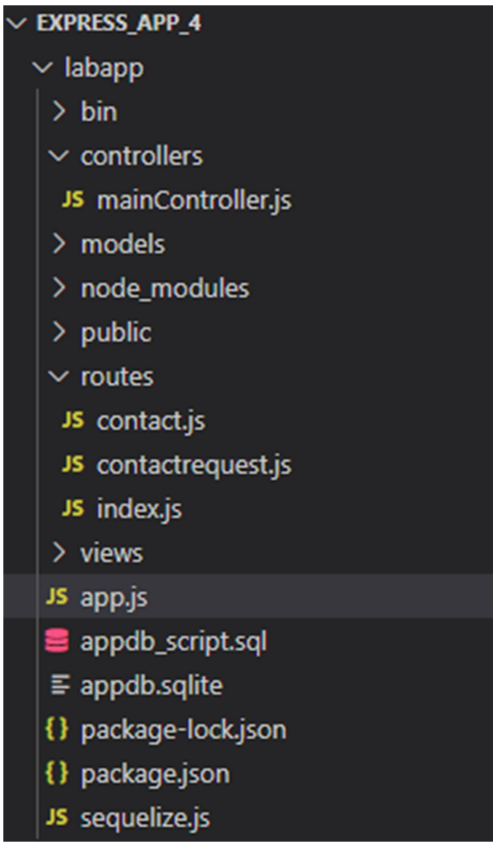
Для соответствующих HTTP-методов PUT и DELETE.

Подключение созданных модулей обработки маршрутов из папки /routes осуществляется в основном файле веб-приложения **app.js**. Маршруты подключаются ДО инициализации самого приложения express:

```
// Подключение обработчиков маршрутов
var indexRouter = require('./routes/index');
var contactRouter = require('./routes/contact');
var contactrequestRouter = require('./routes/contactrequest');

var app = express();
```

Итоговая структура проекта представлена ниже:



6. Тестирование REST API. Примеры, тестирования API с помощью Insomnia представлены на рисунках ниже:

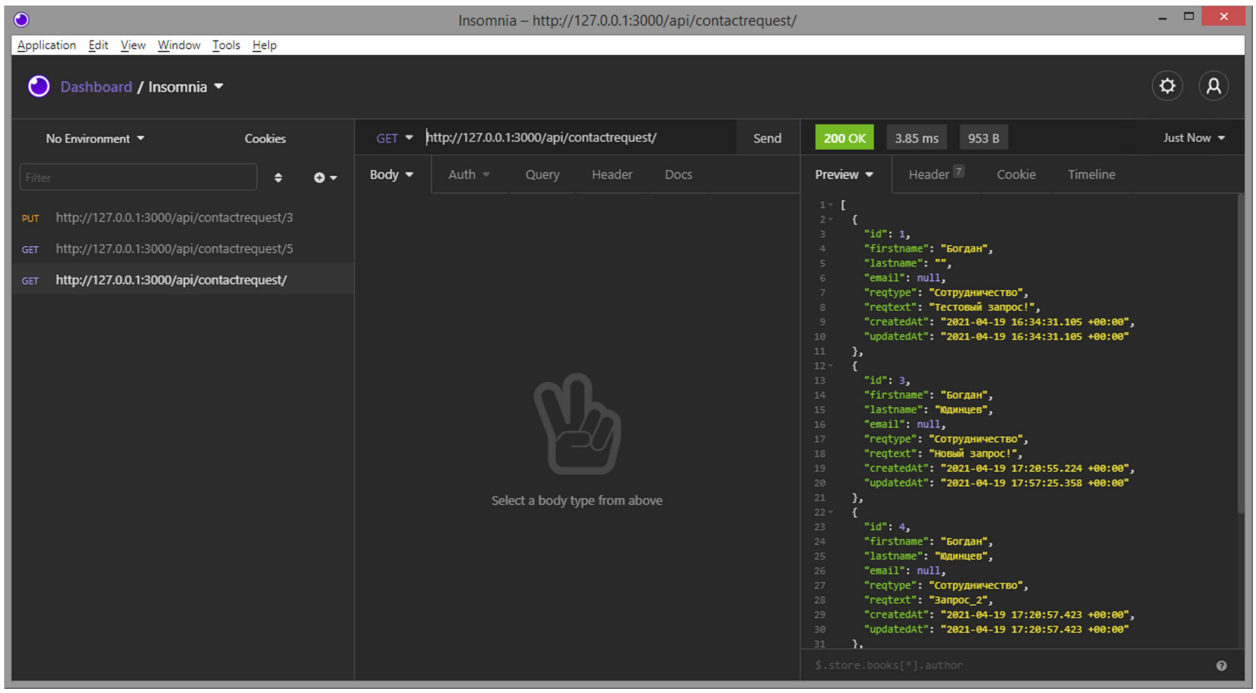


Рис. 6 – Тестирование GET-метода

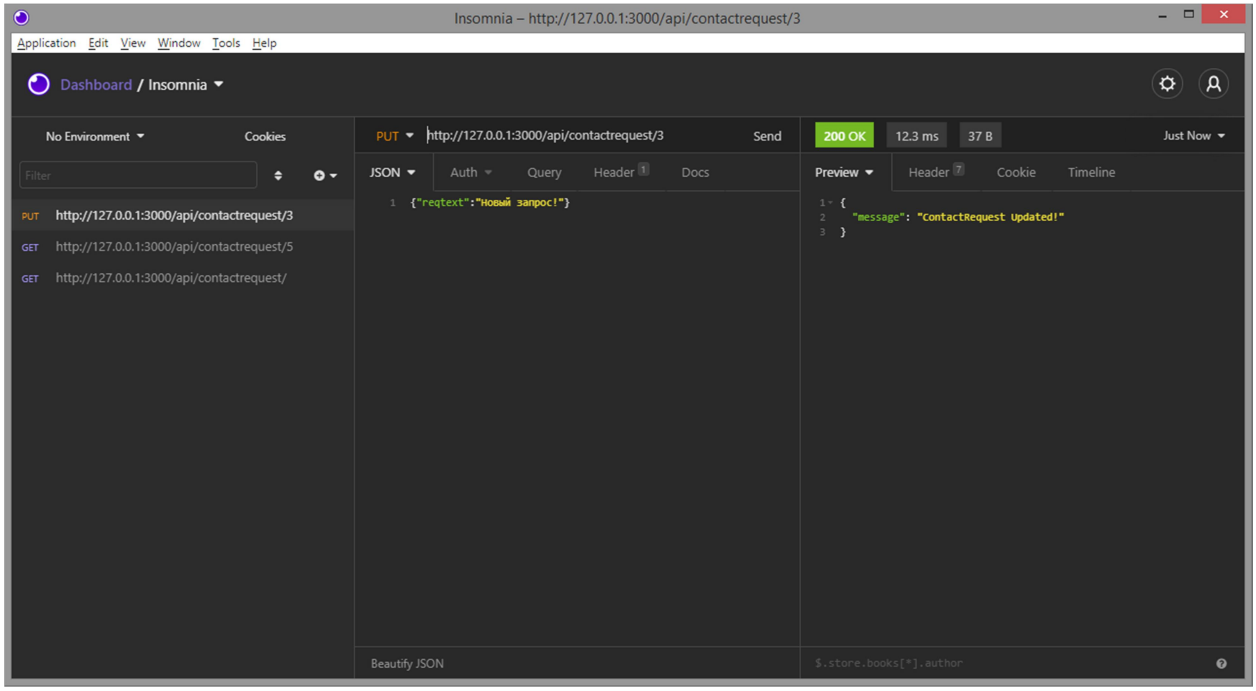


Рис. 7 – Тестирование PUT-метода

Дополнительная литература:

Работа с маршрутами в приложении Node.js:

https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/routes

Tutorial по созданию простого приложения Node.js с использованием MySQL и Sequelize:

https://www.codementor.io/@mirko0/how-to-use-sequelize-with-node-and-express-i24l67cuz

Асинхронная реализация запросов к БД с помощью Sequelize (с использованием инструкций async и await):

https://sequelize.org/master/manual/model-instances.html

Результатом выполнения задания являются файлы с кодом веб-приложения и отчет, содержащий следующую информацию:

1) Конспект теоретического материала по темам: основы протокола HTTP, основные HTTP-методы, HTTP-коды.
2) Таблица (аналогичная таблице №1 в методических рекомендациях) с описанием API для вашей модели.
3) Скриншоты, содержащие результаты проверки вашего API в Insomnia или любом другом REST-клиенте.
4) Работающее веб-приложение, реализованное по заданию.

Требования к оформлению отчета:

Способ выполнения текста должен быть единым для всей работы. **Шрифт** –

Times New Roman, кегль 14, **межстрочный интервал** – 1,5, **размеры полей**: левое – 30 мм; правое – 10 мм, верхнее – 20 мм; нижнее – 20 мм. Сокращения слов в тексте допускаются только общепринятые.

Абзацный отступ (1,25) должен быть одинаковым во всей работе. **Нумерация страниц** основного текста должна быть сквозной. Номер страницы на титульном листе не указывается. Сам номер располагается внизу по центру страницы или справа.

Разработано: Юдинцев Б.С.