Лабораторная работа 04

ПСКП

**Задание 01**

1. Разработайте серверное приложение **04-01**, которое принимает HTTP-запросы **GET**, **POST**, **PUT**, **DELETE** на [**http://localhost:5000/api/db**](http://localhost:5000/api/db).
2. Для имитации таблицы базы данных (БД) используйте JS-массив, а для имитации операций с «БД» соответствующие операции с массивом.
3. Назначение запросов.

|  |  |
| --- | --- |
| **GET** | Получить все строки таблицы «БД». В теле ответа возвращается массив строк в json-формате. |
| **POST** | Добавить новую строку в таблицу «БД»; строка передается серверу в json-формате.  В теле ответа возвращается измененная строка в json-формате. |
| **PUT** | Изменить существующую строку таблицы «БД»; строка передается серверу в json-формате. |
| **DELETE** | Удалить существующую строку (по id) в таблице «БД»; серверу передается id (в query-строке). В теле ответа возвращается удаленная строка в json-формате.  Пример: [**http://localhost:5000/api/db?id=25**](http://localhost:5000/api/db?id=25) |

1. Для работы с «БД» в приложении должен быть разработан специальный объект **DB**, который должен располагаться в отдельном модуле.
2. Объект **DB** должен уметь генерировать и ***асинхронно*** обрабатывать следующие события:

|  |  |
| --- | --- |
| **GET** | Получить все строки таблицы «БД». |
| **POST** | Добавить новую строку в таблицу «БД». |
| **PUT** | Изменить существующую строку таблицы «БД». |
| **DELETE** | Удалить существующую строку в таблице «БД». |

1. Для выполнения операций с «БД» объект **DB** должен обеспечивать следующие функции:

|  |  |
| --- | --- |
| **select** | Получить все строки таблицы «БД». |
| **insert** | Добавить новую строку в таблицу «БД». |
| **update** | Изменить существующую строку таблицы «БД». |
| **delete** | Удалить существующую строку в таблице «БД». |

1. Строка таблицы содержит:

- идентификатор (**id**);

- имя (**name**);

- дата рождения (**bday**).

1. Проверьте работоспособность приложения с помощью **POSTMAN**.

**Задание 02**

1. Разработайте серверное приложение **04-02**, использовав код приложения **04-01**.
2. Дополнительно к функциональности, унаследованной от **04-01,** приложение **04-02** должно отвечать на GET-запросы к [**http://localhost:5000/**](http://localhost:5000/)**.**
3. В ответ на GET-запрос к [**http://localhost:5000/**](http://localhost:5000/)сервер пересылает клиенту HTML-страницу, которая должна располагаться в той же директории, что и js-файл приложения**.**
4. HTML-страница должна обеспечивать ввод необходимых данных, отправку всех типов запросов к серверу и обработку полученных результатов с применением функции **fetch**.
5. Проверьте работоспособность приложения с помощью браузера.

**Задание 03.**Ответьте на следующие вопросы.

1. Дайте пояснение понятию «событие программного объекта».
2. Какой встроенный механизм используется в **Node.js** для генерации и обработки событий. Поясните принцип его работы.

14. JS-класс, предоставляющий функциональность для обработки событий в Node.js.

Производный от Event Emitter объект приобретает функциональность, позволяющую генерировать и прослушивать события Для генерации событий предназначена функция emit(), а для прослушивания – функция on()

Методы EventEmitter’а • emit (eventName[, ...args]) – генерация события с именем eventName. Синхронно вызывает каждый слушатель (в порядке их регистрации) и передает им аргументы args. • on (eventName, listener) – добавляет слушателя в конец массива слушателей на событие с именем eventName. Один и тот же слушатель может быть вызван для события несколько раз (зависит от того, сколько раз вызван .on(). • once (eventName, listener) – вызов слушателя только один раз. • removeAllListeners ([eventName]) – удаляет либо всех слушателей для всех событий объекта, либо всех слушателей для указанного события eventName. • removeListener (eventName, listener) – удале

Событие программного объекта процесс перехода объекта из одного состояние в другое. При этом, об этом переходе могут быть извещены другие объекты. У события есть издатель (или генератор) события и могут быть подписчики (или обработчики) события.

15. Таймер = механизм, позволяющий генерировать событие или выполнить некоторое действие через заданный промежуток времени.

setTimeout(), setInterval() • реализация находится в библиотеке libuv; • setTimeout и setInterval отноятся к макротаскам; • коллбэки

setTimeout() выполняется только один раз через некоторый промежуток времени

setInterval() выполняется регулярно через некоторый промежуток времени

clearTimeout (timeoutObj) останавливает таймер, созданный с помощью setTimeout()

В качестве параметра передается объект ранее созданного с помощью setTimeout() таймера, который необходимо отменить.

clearInterval() останавливает таймер, созданный посредством setInterv

В качестве параметра передается объект ранее созданного с помощью setTimeout() таймера, который необходимо отменит

Процесс Node.js работает до тех пор, пока есть события, требующие обработки