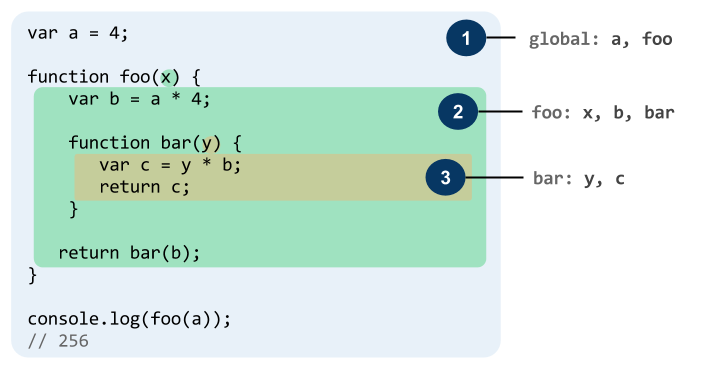
**[JS Module system](https://ui.dev/javascript-modules-iifes-commonjs-esmodules/)**:

Lets have two separate logics as separation of concern in two different files

1. User data storage and it’s getter function
2. Dom manipulation and data display

But we will get access to one variable in other with the hosted Environment global variable window to understand this shall we memorize

**1.scope,local scope,global scope,block scope,functional scope,lexical scope**

****

**2.immediately invoked function expressions**

**3.what is namespace and module**

**4.what is commonjs and ES6 or Ebove**

**5.how the modules organised before ES6 introduction to module system ie what is bundler (webpack)**

### **Modules in JavaScript**

Even though JavaScript never had built-in modules, the community has converged on a simple style of modules, which is supported by libraries in ES5 and earlier. This style has also been adopted by ES6:

* Each module is a piece of code that is executed once it is loaded.
* In that code, there may be declarations (variable declarations, function declarations, etc.).
  + By default, these declarations stay local to the module.
  + You can mark some of them as exports, then other modules can import them.
* A module can import things from other modules. It refers to those modules via *module specifiers*, strings that are either:
  + Relative paths ('../model/user'): these paths are interpreted relatively to the location of the importing module. The file extension .js can usually be omitted.
  + Absolute paths ('/lib/js/helpers'): point directly to the file of the module to be imported.
  + Names ('util'): What modules names refer to has to be configured.
* Modules are singletons. Even if a module is imported multiple times, only a single “instance” of it exists.

This approach to modules avoids global variables, the only things that are global are module specifiers.

#### 16.1.1 Multiple named exports

There can be multiple named exports:

*//------ lib.js ------*

**export** **const** sqrt = Math.sqrt;

**export** **function** square(x) {

**return** x \* x;

}

**export** **function** diag(x, y) {

**return** sqrt(square(x) + square(y));

}

*//------ main.js ------*

**import** { square, diag } from 'lib';

console.log(square(11)); *// 121*

console.log(diag(4, 3)); *// 5*

You can also import the complete module:

*//------ main.js ------*

**import** \* as lib from 'lib';

console.log(lib.square(11)); *// 121*

console.log(lib.diag(4, 3)); *// 5*

#### 16.1.2 Single default export

There can be a single default export. For example, a function:

*//------ myFunc.js ------*

**export** **default** **function** () { ··· } *// no semicolon!*

*//------ main1.js ------*

**import** myFunc from 'myFunc';

myFunc();

Or a class:

*//------ MyClass.js ------*

**export** **default** **class** { ··· } *// no semicolon!*

*//------ main2.js ------*

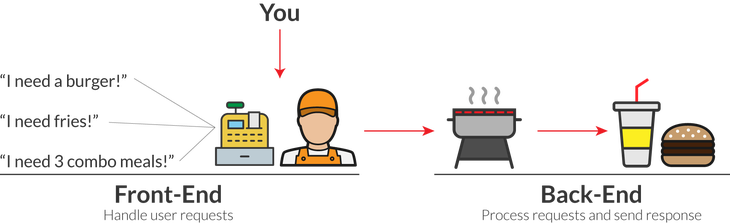
**import** MyClass from 'MyClass';

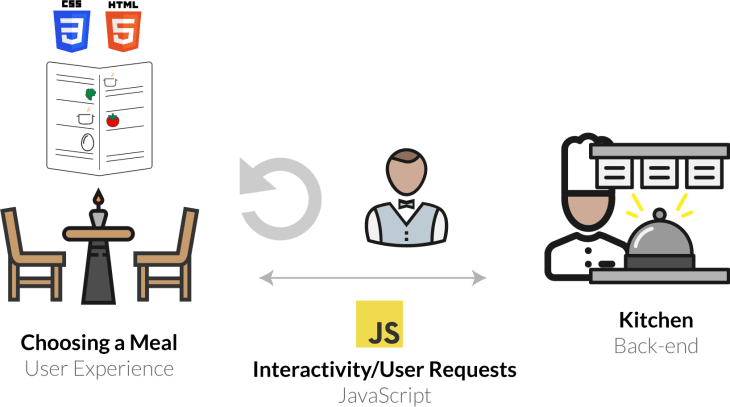
**const** inst = **new** MyClass();

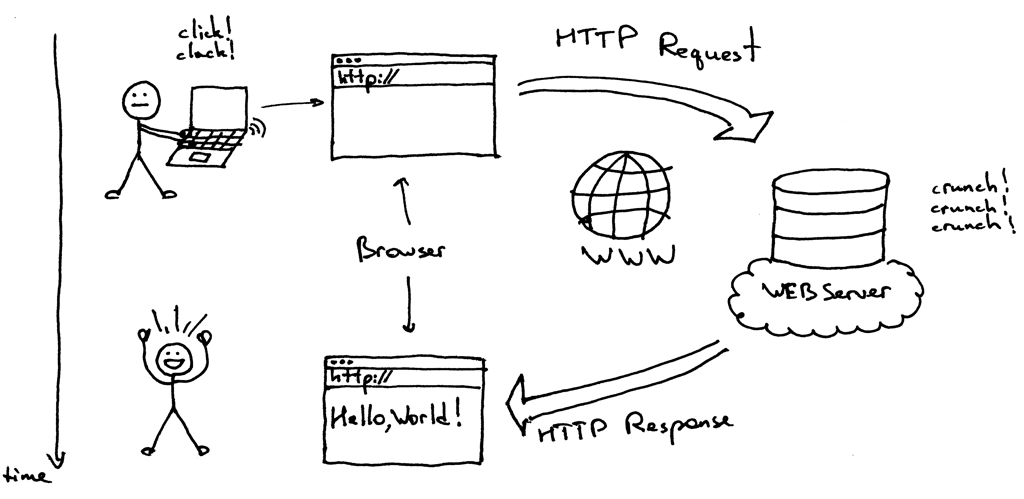
Note that there is no semicolon at the end if you default-export a function or a class (which are anonymous declarations).

#### 6.1.3 Browsers: scripts versus modules

|  | **Scripts** | **Modules** |
| --- | --- | --- |
| HTML element | <script> | <script type="module"> |
| Default mode | non-strict | strict |
| Top-level variables are | global | local to module |
| Value of this at top level | window | undefined |
| Executed | synchronously | asynchronously |
| Declarative imports (import statement) | no | yes |
| Programmatic imports (Promise-based API) | yes | yes |
| File extension | .js | .js |

[**how client communicates with server**](https://medium.com/free-code-camp/ajax-basics-explained-by-working-at-a-fast-food-restaurant-88d95f5fcb7a)****

****



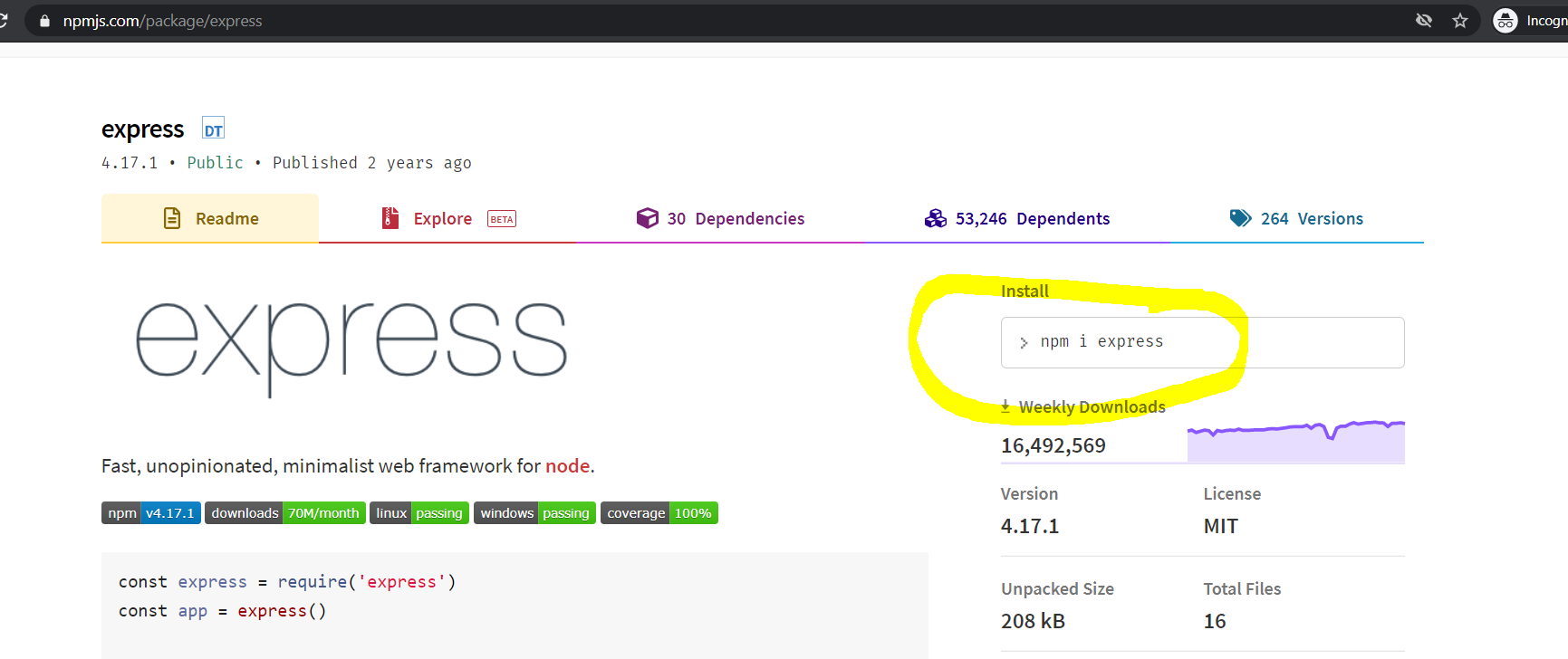
| **HTTP Verb** | **CRUD** | **Entire Collection (e.g. /customers)** | **Specific Item (e.g. /customers/{id})** |
| --- | --- | --- | --- |
| POST | Create | 201 (Created), 'Location' header with link to /customers/{id} containing new ID. | 404 (Not Found), 409 (Conflict) if resource already exists.. |
| GET | Read | 200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single customer. 404 (Not Found), if ID not found or invalid. |
| PUT | Update/Replace | 405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| PATCH | Update/Modify | 405 (Method Not Allowed), unless you want to modify the collection itself. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable. | 200 (OK). 404 (Not Found), if ID not found or invalid. |

* **1xx — Informational**: The server has not fully completed the request, it is still thinking and is in a transitional phase
* **2xx — Successful:** The server has successfully completed the request
* **3xx — Redirects:**This block is for redirections, it means you requested an address but you were sent somewhere else
* **4xx — Client Errors:** There is some error from your side
* **5xx — Server Errors:** There is some error on the server-side.

[**NPM:How to install and update our required Node.js Packages:**](https://www.npmjs.com/)

**--create package.json :** npm init -y or yarn init -y

**Install or update:** npm install (package name)

**Install any package:** npm i express (or) yarn add express

**Why do we need Database or data stores**

* Whether the data will be accessed and edited by one or multiple people simultaneously.
* The number of people and the volume of data you need to share with others, simultaneously.
* The amount of data to be stored and processed by the database software.
* The complexity of the data e.g. whether data is images, text, sound, videos, or various other formats.

**What is sql and why it is need ??**

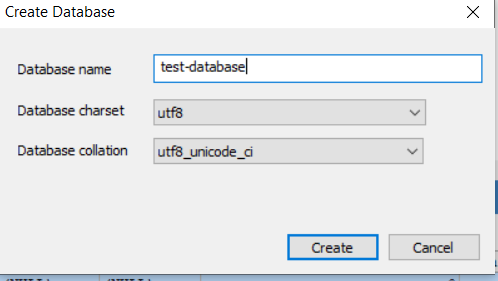
* It helps users to access data in the RDBMS system.
* It helps you to describe the data.
* It allows you to define the data in a database and manipulate that specific data.
* With the help of SQL commands in DBMS, you can create and drop databases and tables.
* SQL offers you to use the function in a database, create a view, and stored procedure.
* You can set permissions on tables, procedures, and views.

**Here are five types of widely used SQL queries.**

* **Data Definition Language (DDL)**: define the database structure or schema (CREATE,DROP,ALTER,TRUNCATE)
* **Data Manipulation Language (DML):** modify the database instance by inserting, modifying, and deleting its data(INSERT,UPDATE,DELETE)
* **Data Control Language(DCL):** rights & permissions(GRANT and REVOKE)
* **Transaction Control Language(TCL):** deal with the transaction (Commit,Rollback,SAVEPOINT)
* **Data Query Language (DQL):** fetch the data(SELECT)

**Create Data Base:**

CREATE DATABASE `test-databse`CHARACTER SET utf8 COLLATE utf8\_unicode\_ci;



**Create Table**:

CREATE TABLE `test-databse`.`test-table`(

`id` INT NOT NULL AUTO\_INCREMENT,

`userName` VARCHAR(120) NOT NULL,

`password` VARCHAR(25) NOT NULL,

`totalAttendedClassDuration` FLOAT NOT NULL DEFAULT 0,

`totalClassDuration` FLOAT NOT NULL,

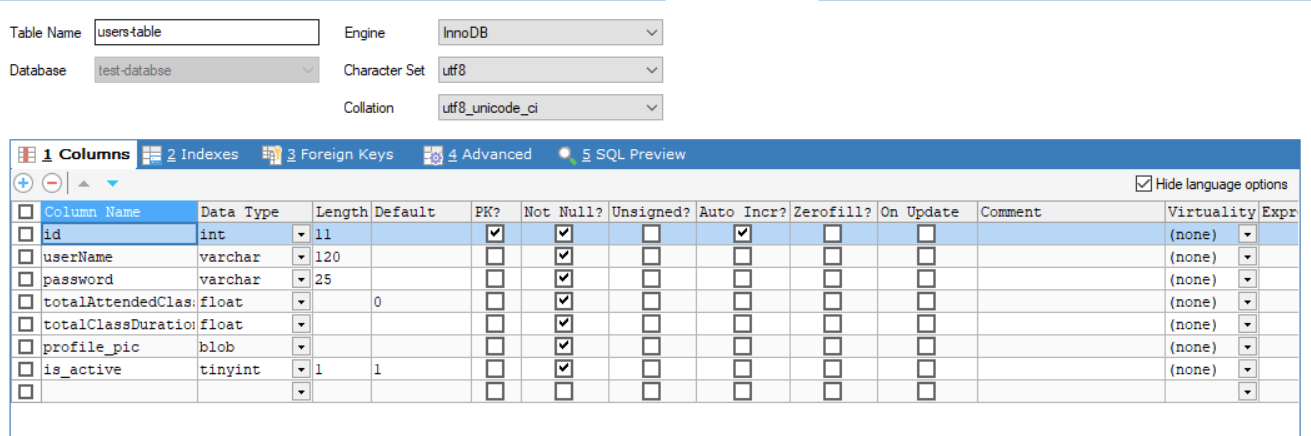
`profile\_pic` BLOB NOT NULL,

`is\_active` BOOLEAN NOT NULL DEFAULT TRUE,

PRIMARY KEY (`id`),

UNIQUE INDEX `userNameUnique` (`userName`)

) ENGINE=INNODB CHARSET=utf8 COLLATE=utf8\_unicode\_ci;



**With foreign key:**

CREATE TABLE `test-databse`.`daily-attended-classes`(

`id` INT NOT NULL AUTO\_INCREMENT,

`userId` INT NOT NULL,

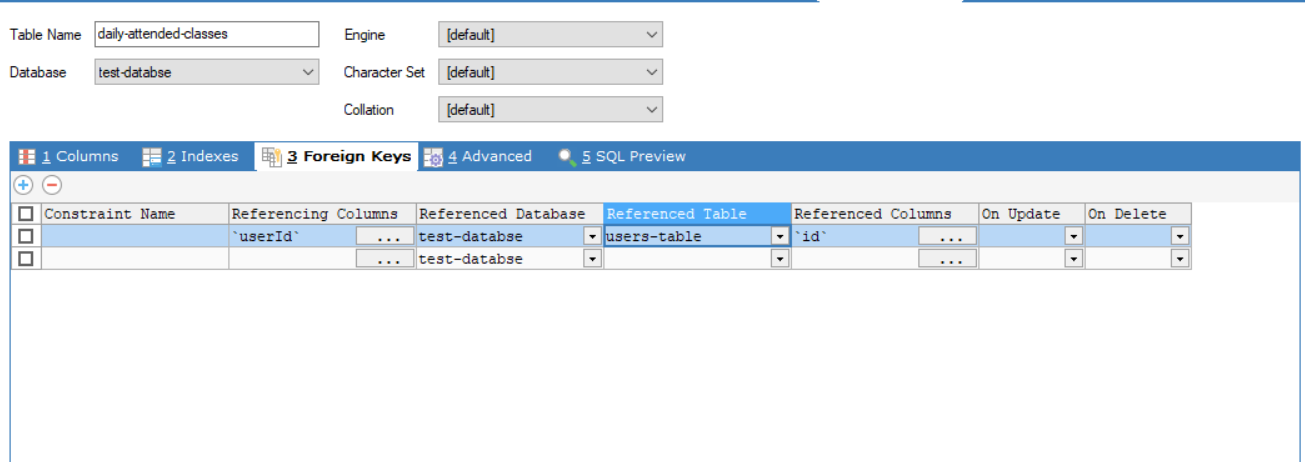
`date` DATETIME NOT NULL,

`is\_attended` BOOLEAN NOT NULL DEFAULT TRUE,

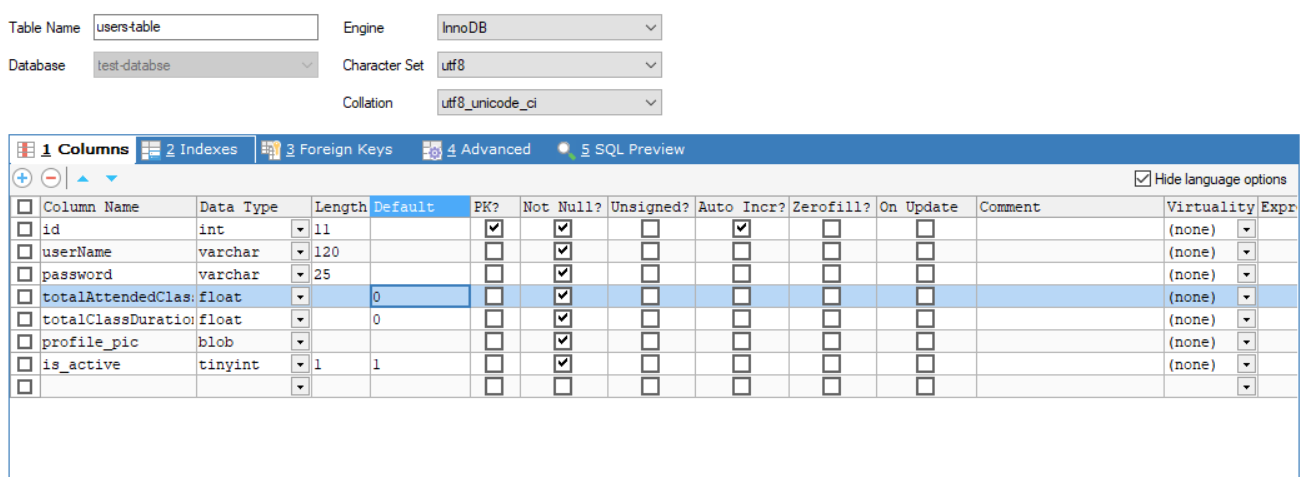
PRIMARY KEY (`id`, `userId`, `is\_attended`),

FOREIGN KEY (`userId`) REFERENCES `test-databse`.`users-table`(`id`)

);



**Alter Table:**



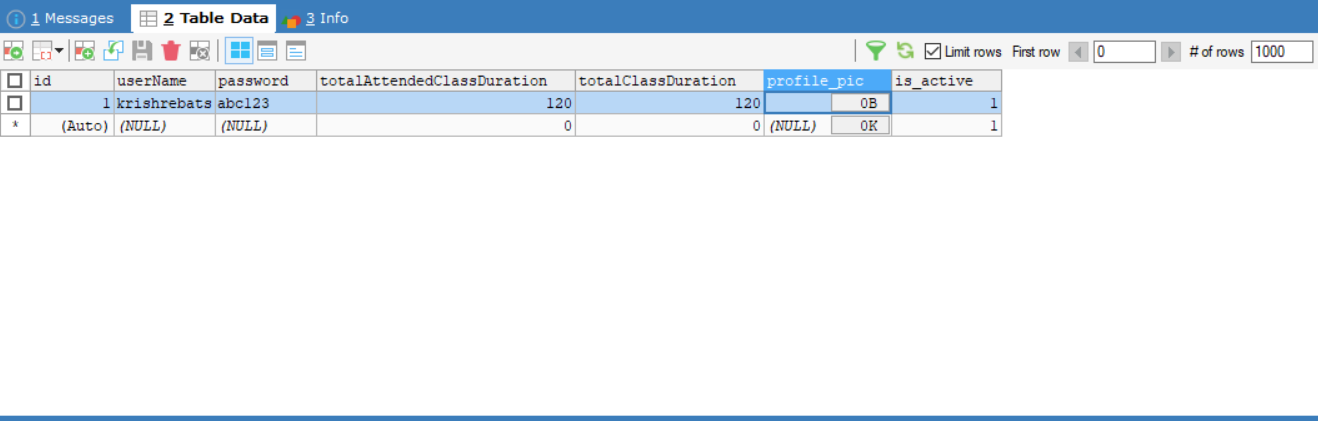
ALTER TABLE `test-databse`.`users-table`

CHANGE `totalClassDuration` `totalClassDuration` FLOAT DEFAULT 0 NOT NULL;

**INSERT:**

INSERT INTO `test-databse`.`daily-attended-classes` (`userId`, `date`,` classDuration`,` attendedDuration`) VALUES ('1', '2021-05-04 18:21:05',120,100);

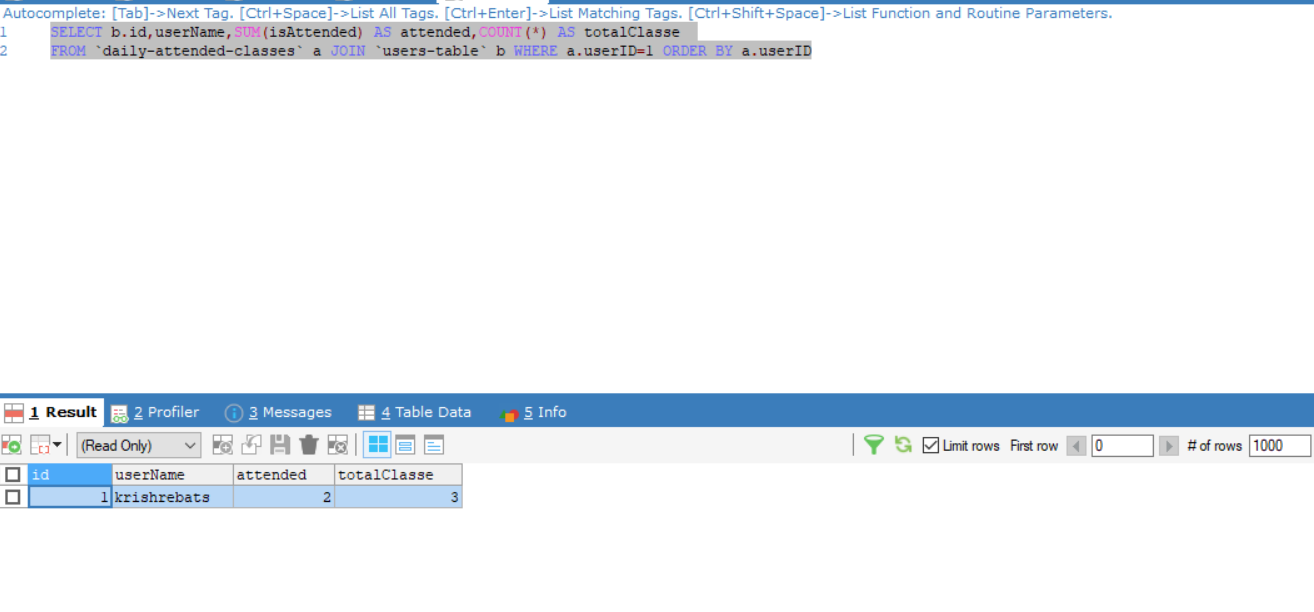
INSERT INTO `test-databse`.`users-table` (`userName`, `password`, `totalAttendedClassDuration`, `totalClassDuration`, `profile\_pic`) VALUES ('krishrebats', 'abc123', '120', '120', '');



**JOINS,ORDER BY,COUNT,SUM,WHERE:**

SELECT b.id,userName,SUM(isAttended) AS attended,COUNT(\*) AS totalClasse

FROM `daily-attended-classes` a JOIN `users-table` b WHERE a.userID=1 ORDER BY a.userID



[**Types of Frontend and Backend Technologies and their adavantages and disadavantages**](https://www.geeksforgeeks.org/frontend-vs-backend/)

**Front End Development:** The part of a website that the user interacts with directly is termed the front end. It is also referred to as the ‘client side’ of the application. It includes everything that users experience directly: text colors and styles, images, graphs and tables, buttons, colors, and navigation menu. HTML, CSS, and JavaScript are the languages used for Front End development. The structure, design, behavior, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers. Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e. it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

**Languages:**

1. **HTML - Skelton**
2. **CSS - Presentation**
3. **JavaScript – Interaction**

**Front End Frameworks and Libraries:**

1. **jQuery**
2. **AngularJS**
3. **React.js**
4. **AngularJS**
5. **VueJs**
6. **Flutter**

**Backend Development:** Backend is the server-side of the website. It stores and arranges data, and also makes sure everything on the client-side of the website works fine. It is the part of the website that you cannot see and interact with. It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application. Activities, like writing APIs, creating libraries, and working with system components without user interfaces or even systems of scientific programming, are also included in the backend.

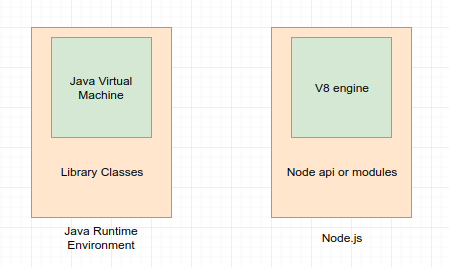
**Languages:**

1. **PHP**
2. **Java**
3. **JavaScript(Node)**
4. **Python**
5. **C#**
6. **Ruby**
7. **GO**

**Back End Frameworks and Libraries:**

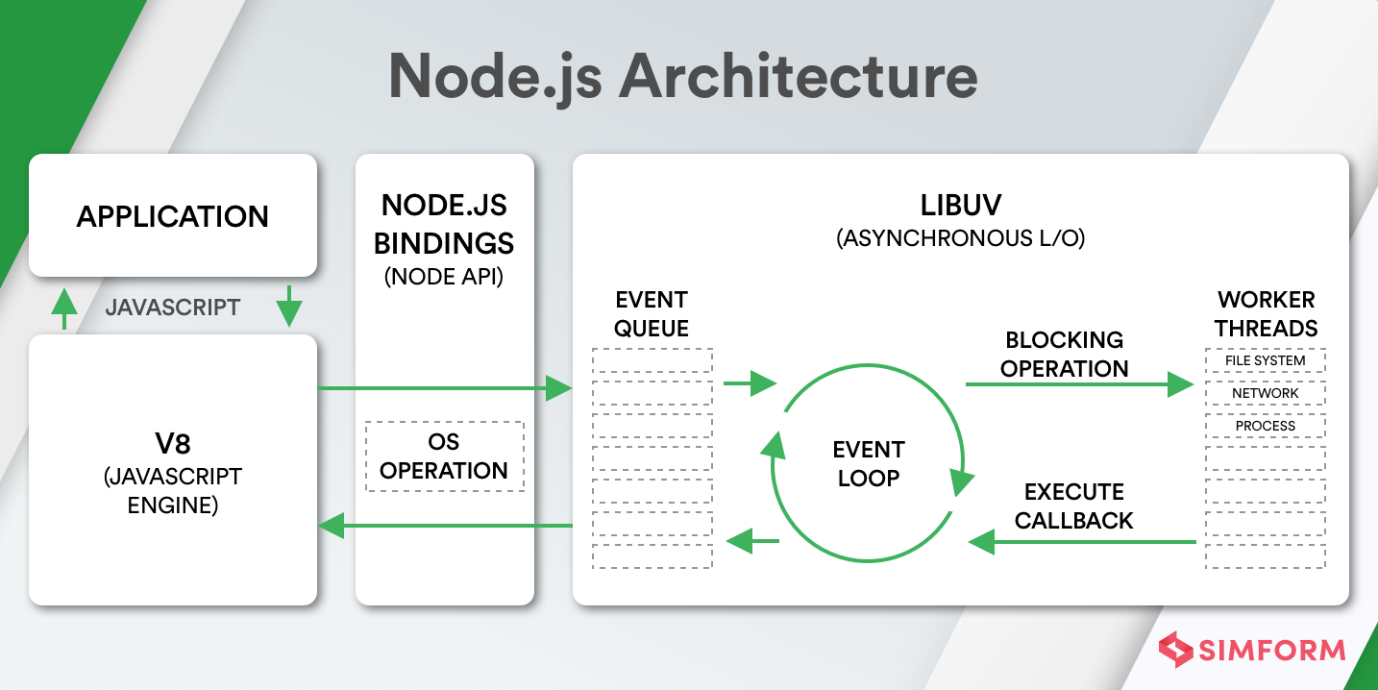
1. Express
2. Django
3. Rails
4. Laravel
5. Spring

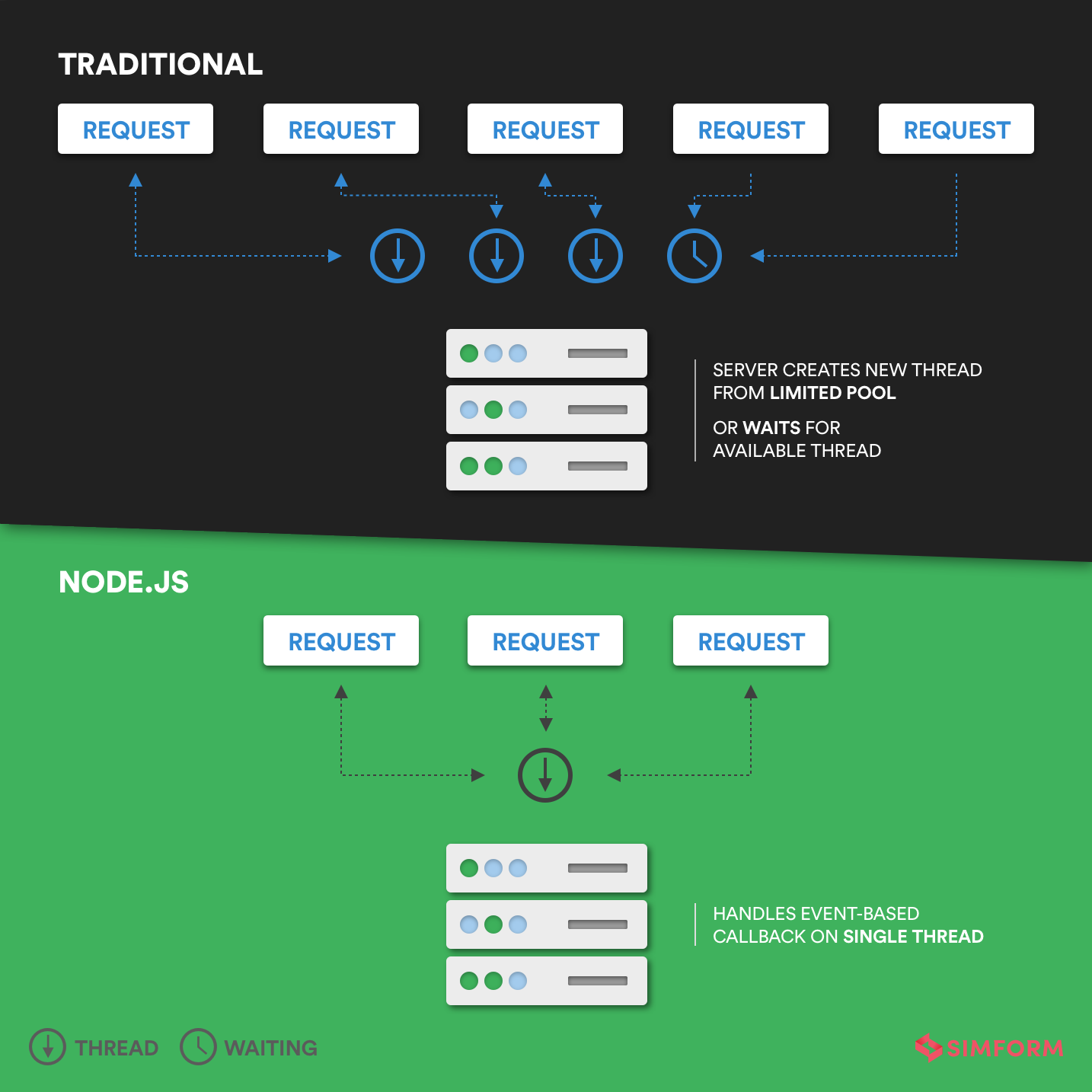
[What is Node.js:](https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/)



1. Node.js® is a JavaScript runtime built on Chrome’s V8 JavaScript engine.
2. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.
3. Node.js’ package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

[Why we use it](https://www.simform.com/what-is-node-js/)







[Download and install](https://nodejs.org/en/download/)

