CSS Micro project

What is JavaScript

JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Where HTML and CSS are languages that give structure and style to web pages, JavaScript gives web pages interactive elements that engage a user.

What is CSS (client-side scripting) language

Client-side scripting languages create the scripts that run on the client side (i.e., your browser). These are sent from the server by server-side scripts. Some good examples are JavaScript, jQuery, CSS etc.

Client-Side Scripting refers to the output which is requested to the server by the end-users. The majority of this page is written in HTML. With client-side scripting, JavaScript is the primary language used. It is the most widely used language in this area, and it works with all programs.

Project Title:

Student Application (CRUD Operation) with Login page

What is the working of this application?

This project is a client side programmed using JavaScript with fully styled with CSS cascading style code, when your first execute this program on live server it open the login page first only authorize user can login means, only users can enter who signed up before, once the user logged in then they redirect to student page

The user can create, read update and delete (CRUD) the student information into the database.

What is database?

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

What is SQL Database?

Structured Query Language

SQL stands for Structured Query Language. SQL lets you access and manipulate databases. SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

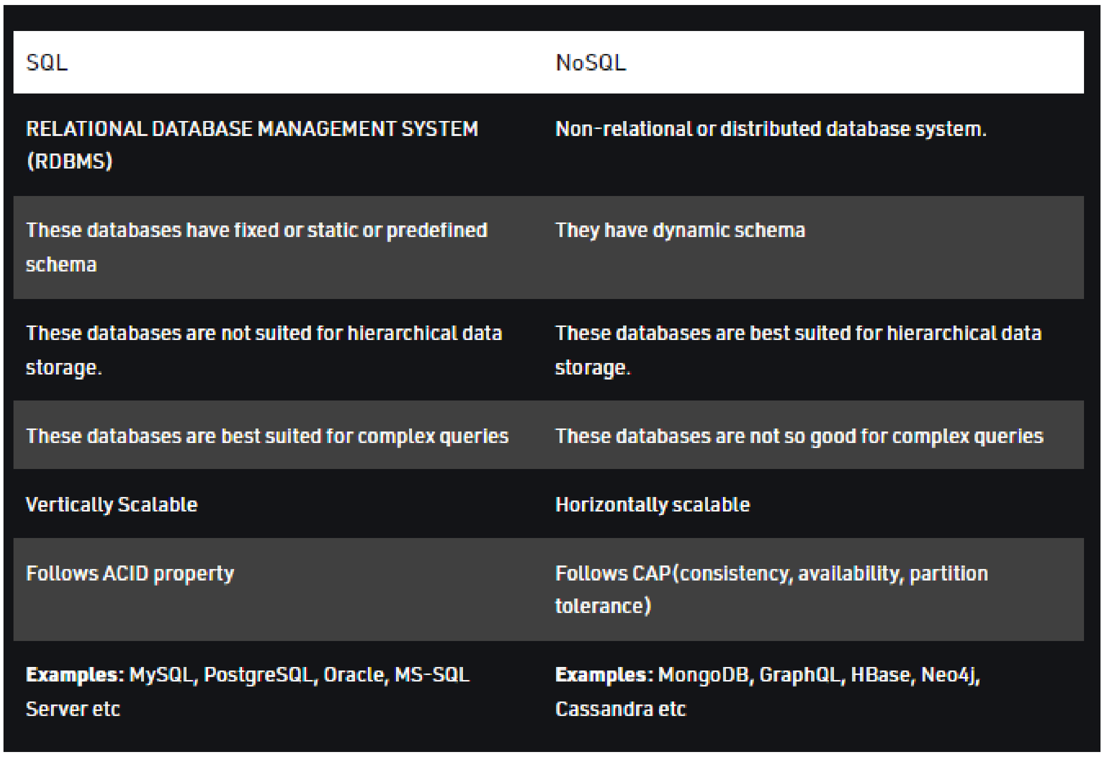
Structured Query Language (SQL) refers to a standard programming language utilized to extract, organize, manage, and manipulate data stored in relational databases. SQL is thereby referred to as a database language that can execute activities on databases that consist of tables made up of rows and columns

SQL offers two main advantages over older read–write APIs such as ISAM or VSAM. Firstly, it introduced the concept of accessing many records with one single command. Secondly, it eliminates the need to specify how to reach a record, e.g., with or without an index.

What is NOSQL Database?

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.

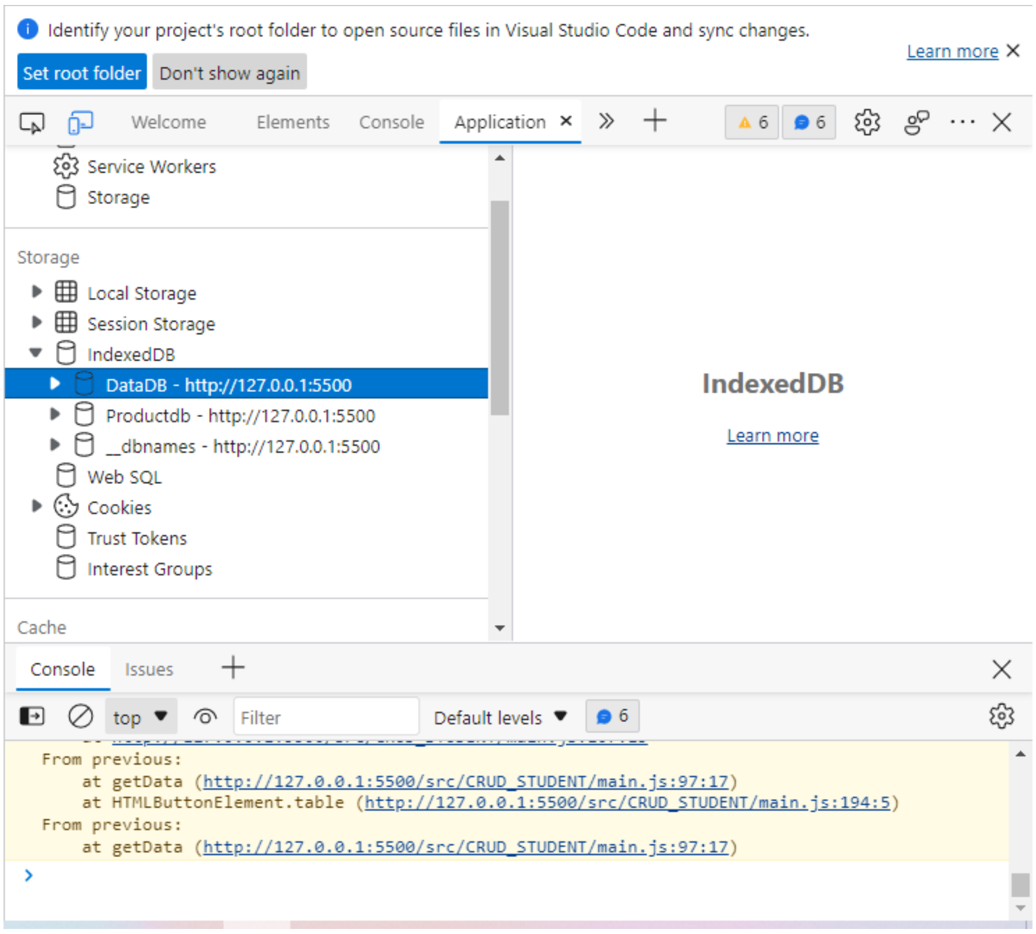
NoSQL databases store data in documents rather than relational tables. Accordingly, we classify them as “not only SQL” and subdivide them by a variety of flexible data models. Types of NoSQL databases include pure document databases, key-value stores, wide-column databases, and graph databases.



SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP(consistency, availability, partition tolerance)
Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server etc	**Examples:** MongoDB, GraphQL, HBase, Neo4j, Cassandra etc

We use Dexie.js in our project

Dexie.js is a wrapper library for indexedDB - the standard database in the browser



Identify your project's root folder to open source files in Visual Studio Code and sync changes.

```
From previous:
    at getData (http://127.0.0.1:5500/src/CRUD_STUDENT/main.js:97:17)
    at HTMLButtonElement.table (http://127.0.0.1:5500/src/CRUD_STUDENT/main.js:194:5)
From previous:
    at getData (http://127.0.0.1:5500/src/CRUD_STUDENT/main.js:97:17)
>
```

What is Indexed DB?

The Indexed Database API is a JavaScript application programming interface provided by web browsers for managing a NoSQL database of JSON objects. It is a standard maintained by the World Wide Web Consortium. As an alternative to the Web storage standard, IndexedDB can provide more storage capacity.

Key concepts and usage. IndexedDB is a transactional database system, like an SQL-based RDBMS. However, unlike SQL-based RDBMSes, which use fixed-column tables, IndexedDB is a JavaScript-based object-oriented database.

IndexedDB is a way for you to persistently store data inside a user's browser. Because it lets you create web applications with rich query abilities regardless of network availability, your applications can work both online and offline

IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data. While Web Storage is useful for storing smaller amounts of data, it is less useful for storing larger amounts of structured data. IndexedDB provides a solution. This is the main landing page for MDN's IndexedDB coverage — here we provide links to the full API reference and usage guides, browser support details, and some explanation of key concepts.

Dexie.js is a wrapper library for indexedDB - the standard database in the browser

Dexie solves three main issues with the native IndexedDB API:

1. Ambiguous error handling
2. Poor queries
3. Code complexity

Dexie provides a neat database API with a well thought-through API design, robust error handling, extendibility, change tracking awareness and extended Key Range support (case insensitive search, set matches and OR operations).

Indexed DB vs local Storage

On the surface the two technologies may seem directly comparable, however if you spend some time with them you'll soon realize they are not. They were designed to achieve a similar goal, client side storage, but they approach the task at hand from significantly different perspectives and work best with different amounts of data.

localStorage, or more accurately Web Storage, was designed for smaller amounts of data. It's essentially a strings only key - value storage, with a simplistic synchronous API. That last part is key. Although there's nothing in the specification that prohibits an asynchronous Web Storage, currently all implementations are synchronous (i.e. blocking requests). Even if you didn't mind using a naive key - value storage for larger amounts of data, your clients will mind waiting forever for your application to load.

indexedDB, on the other hand, was designed to work with significantly larger amounts of data. First, in theory, it provides both a synchronous and an asynchronous API. In practice, however, all current implementations are asynchronous, and requests will not block the user interface from loading. Additionally, indexedDB, as the name reveals, provides indexes. You can run rudimentary queries on your database and fetch records by looking up theirs keys in specific key ranges. indexedDB also supports transactions, and provides simple types (e.g. Date).

At this point, indexedDB might seem the superior solution for every situation ever. However, there's a penalty for all its features: Compared to Web Storage, its API is quite complicated. indexedDB assumes a general familiarity with database concepts, whereas with Web Storage you can jump right in. If you have ever worked with cookies, you won't have an issue working with Web Storage. Also, in general you'll need to write more code in indexedDB to achieve exactly the same result as in Web Storage (and more code = more bugs). Furthermore, emulating Web Storage for browsers that don't support it is relatively straightforward. With indexedDB, the task wouldn't be worth its time. Lastly, before you dive into indexedDB, you should first take a look at the Quota API.

At the end of the day, it's completely up to you if you use Web Storage or indexedDB, or both, in your application. A good use case for Web Storage would be to store simple session data, for example a user's name, and save you some requests to your actual database. indexedDB's additional features, on the other hand, could help you store all the data you need for your application to work offline.