IndexedDB

▼ What is IndexedDB?

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.

IndexedDB is a transactional database system, like an SQL-based Relational Database Management System (RDBMS). However, unlike SQL-based RDBMSes, which use fixed-column tables, IndexedDB is a JavaScript-based object-oriented database. (source MDN)

▼ How is it used?

Important IndexedDB APIs are:

1. `indexedDB.open`: Opens a connection to a database. If the database does not exist, it will be created
2. `IDBDatabase`: Represents a connection to a database. Allows you to create, delete, and modify object stores and indexes, as well as perform transactions. Important methods are `createObjectStore(name, options)` , `deleteObjectStore(name)` , `transaction(storeNames, mode)`
3. `IDBObjectStore`: Represents an object store in the database. Provides methods to add, retrieve, and delete records.
4. `IDBTransaction`: Represents a transaction on the database. Provides a way to execute multiple operations as a single unit.

For more Interfaces, visit this link

Dexie

Dexie is a wrapper library for IndexedDB that simplifies its usage and improves the developer experience.

Following is the explanation of the Todo List App discussed in the lecture

```
// Creates a DB instance named todoDB
const db = new Dexie('todoDB');

// Creates an object store named todos with an auto-increme
db.version(1).stores({ todos: '++id,task' });


// get the elements to target their values nad display data
const todoForm = document.getElementById('todoForm');
const todoInput = document.getElementById('todoInput');
const todoList = document.getElementById('todoList');

// Adds a new record to the todos object store with the tas
function addTodo() {
  db.todos.add({ task: todoInput.value }).then(displayTodos
  todoInput.value = '';
}

//Retrieves all records from the todos object store as an a
function displayTodos() {
  db.todos.toArray().then(todos => {
    while (todoList.firstChild) {
      todoList.removeChild(todoList.firstChild);
    }

    todos.forEach(todo => {
      const listItem = document.createElement('li');
      listItem.textContent = todo.task;
      todoList.appendChild(listItem);
    });
  });
}

// Event listener to the form element to add the todos to D
```

```
todoForm.addEventListener('submit', function (event) {
  event.preventDefault();
  addTodo();
});

// Display the initial todos which may be in the indexedDB
displayTodos();
```

▼ Size Limit?

more than 100MB of data is available. It is used for large datasets.

▼ Important Performance points

Asynchronous: Operations performed using IndexedDB are done asynchronously, so as not to block applications.

▼ Data Persistence

Data stored in `IndexedDB` persists indefinitely (across browser sessions), similar to `localStorage`, until it is explicitly deleted by the user or via JavaScript.

▼ Data Structure

The data stored in IndexedDB is in the key-value format. The value here can be any complex data structure.

IndexedDB lets you store and retrieve objects that are indexed with a **key**; any objects supported by the structured clone algorithm can be stored. You need to specify the database schema, open a connection to your database, and then retrieve and update data within a series of **transactions**. (source: MDN)

This API uses indexes to enable high-performance searches of this data.

▼ Security

Accessibility: Data in `IndexedDB` can be accessed by any script running on the same origin. It is protected by the same-origin policy but can be vulnerable to XSS attacks.

No Encryption: Data is not encrypted by default. Sensitive data should be encrypted before storing.

▼ When to use?

1. Storing large amounts of data that need to be queried efficiently.
2. Applications requiring offline support with complex data models.
3. Storing binary data such as images, files, and blobs.
4. Complex transactions that require ACID (Atomicity, Consistency, Isolation, Durability) properties.

▼ When not to use?

1. Storing small amounts of simple data (use `localStorage` or `sessionStorage` instead).
2. Applications where synchronous data access is required (IndexedDB is asynchronous).
3. If the data does not need to persist beyond a session and is not complex (use `sessionStorage`).
4. If the data is to be secure or is sensitive in nature