Project 1

Here's a detailed explanation of how I executed this IndexedDB project in the first person:

1. Project Setup

I started by setting up the environment and creating the necessary project files:

- HTML File (`index.html`): This file serves as the frontend interface for my project. It contains buttons that allow me to interact with the IndexedDB and trigger various tasks like initializing the database, adding tasks, reading completed tasks, and applying different optimization techniques. Additionally, I included a `<div>` with the ID `output` to display results on the page.

- JavaScript File (`main.js`): I wrote all the logic for interacting with IndexedDB in this file. This includes creating object stores, populating the database with 100,000 tasks, querying tasks, and performing performance optimizations, such as applying a read-only flag or indexing.

2. Initializing IndexedDB

I wrote the `initializeDatabase()` function in `main.js` to create an IndexedDB database called `TodoDB`. In this function, I set up the object store named `TodoList`, which stores each task with the properties `task`, `status`, and `dueDate`. Additionally, I created an index on the `status` field to optimize querying based on the task status.

3. Populating the Database with 100,000 Tasks

After initializing the database, I created a `populateTasks()` function to add 100,000 randomly generated tasks to the `TodoList` object store. The first 1000 tasks were assigned the status `"completed"`, while the remaining 99,000 were marked as `"in progress"`.

I ran this function by clicking the Populate Tasks button on the webpage, and it added all 100,000 tasks to the database.

4. Reading "Completed" Tasks and Measuring Time

To optimize querying, I created different functions to fetch all tasks with the status `"completed"`. First, I measured the time it took to query these tasks without any optimizations. This was done using the `getAll()` method on the index I had created for the `status` field.

I then ran the Read Completed Tasks button and observed the time taken to fetch the 1000 completed tasks.

5. Applying the Read-Only Flag and Measuring Performance

Next, I used the `readonly` transaction mode to optimize the query further. This ensures that the transaction is optimized for reading, and no writing operations can take place during this query. I wrote a function that repeated the query but with the read-only flag applied:

By clicking the Read with Read-Only Flag button, I compared the performance with and without the read-only transaction.

6. Reading Tasks Using Index

In another optimization, I created an index on the `status` field when I initialized the database. Using the index, I optimized the query to fetch only the tasks with the status `"completed"`. I wrote a function to test the speed of fetching tasks using the index:

I clicked the Read Using Index button to measure the performance improvement when using an index to retrieve the completed tasks.

7. Copying Completed Tasks to a New Object Store

As part of the performance optimization experiment, I also created a separate object store called `TodoListCompleted` and copied all completed tasks from the original `TodoList` object store into it. This allowed me to compare the performance of fetching tasks from a dedicated object store.

I clicked the Copy Completed Tasks to New Store button to execute this operation and then measured how fast the completed tasks could be read from the new store.

8. Viewing Results in the Browser

To view the results, I used two main methods:

1. Console Logs: I used `console.log()` statements to print the time taken for each operation to the browser's console.

2. HTML Output: I displayed the results directly on the webpage by appending messages to the `#output` div in the HTML file.

For example, I could display the times taken for each operation on the page using the following code:

Question 6

Here's a brief summary of lessons learned, challenges, and insights:

1. Database Initialization: Setting up IndexedDB and indexes was straightforward. The challenge was understanding the asynchronous nature of IndexedDB. Key insight: defining a schema early improves performance.

2. Populating 100,000 Tasks: IndexedDB handled the large dataset well, but adding so many tasks took time. Bulk transactions could be optimized for better performance.

3. Reading Completed Tasks (Without Optimizations): Fetching tasks from the large dataset was slow without optimizations. This highlighted the importance of performance tuning for large data queries.

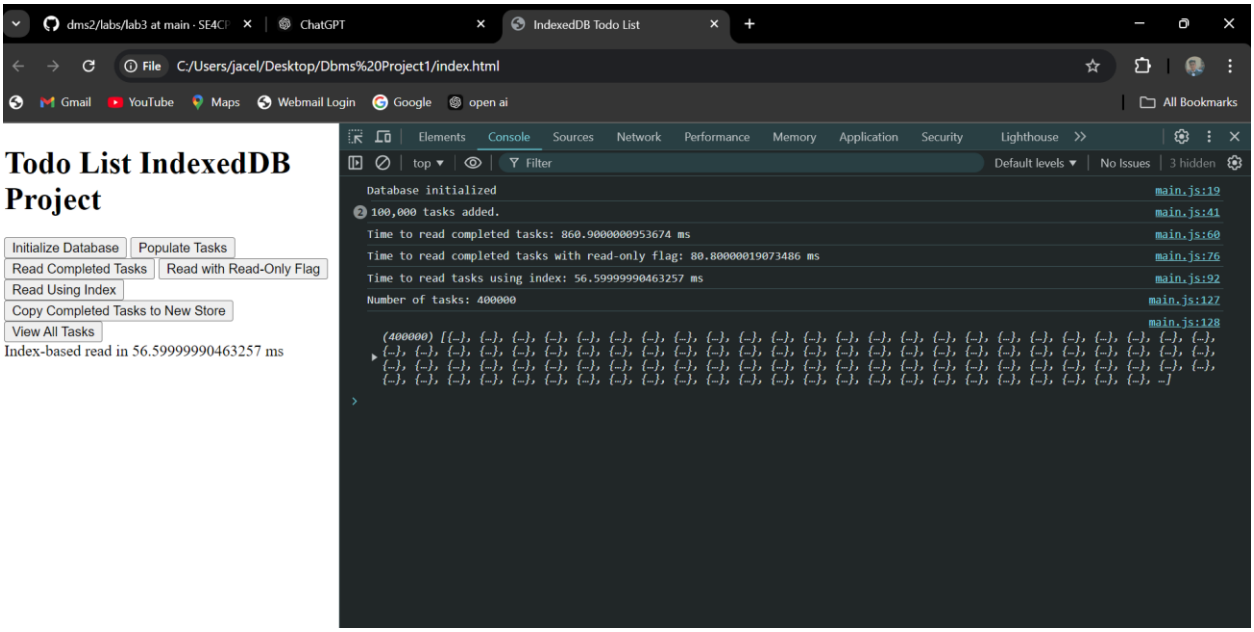
4. Read-Only Flag: Applying a read-only flag improved query speed. The challenge was understanding transaction modes, and the insight was that read-only transactions are faster.

5. Using Index: Using an index on the `status` field greatly improved performance. Indexes are crucial for optimizing queries on large datasets.

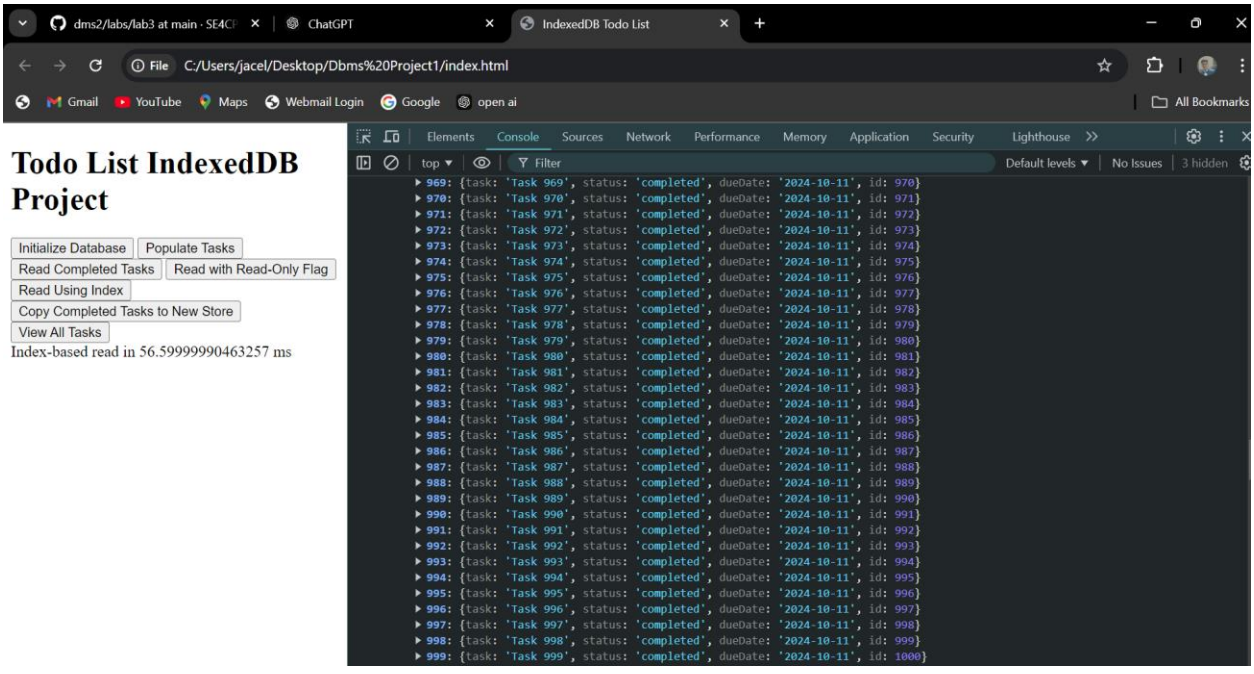
6. Copying to New Store: Creating a dedicated object store for completed tasks improved performance. Bulk copying was slow, but separating frequently queried data improved efficiency.

Overall, indexing and transaction handling were key to optimizing performance.

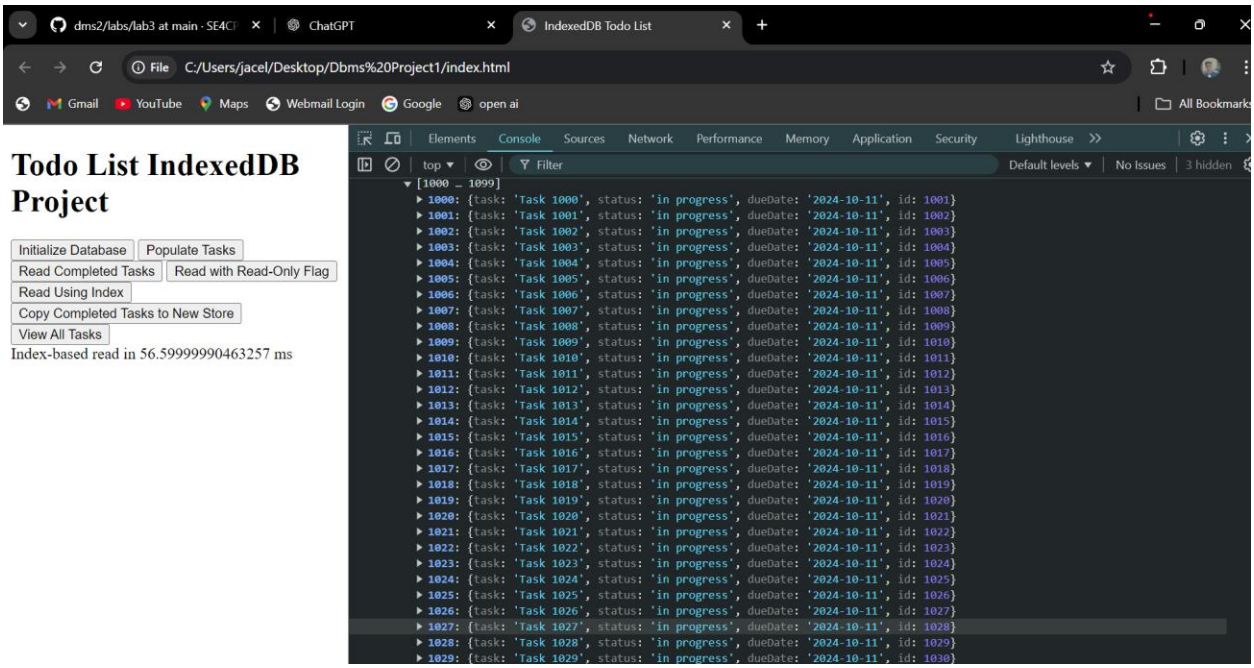
Screenshot of the executed tasks



Screen shot of complete task status from 1-1000



Screen shot of task with status in -progress


dms2/labs/lab3 at main · SE4CP
ChatGPT
IndexedDB Todo List
File C:/Users/jacel/Desktop/Dbms%20Project1/index.html
Gmail
YouTube
Maps
Webmail Login
Google
open ai
All Bookmarks
Todo List IndexedDB Project
Initialize Database
Populate Tasks
Read Completed Tasks
Read with Read-Only Flag
Read Using Index
Copy Completed Tasks to New Store
View All Tasks
Index-based read in 56.59999990463257 ms
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
top
Filter
Default levels
No Issues
3 hidden
[1000 … 1099]
1000: {task: 'Task 1000', status: 'in progress', dueDate: '2024-10-11', id: 1001}
1001: {task: 'Task 1001', status: 'in progress', dueDate: '2024-10-11', id: 1002}
1002: {task: 'Task 1002', status: 'in progress', dueDate: '2024-10-11', id: 1003}
1003: {task: 'Task 1003', status: 'in progress', dueDate: '2024-10-11', id: 1004}
1004: {task: 'Task 1004', status: 'in progress', dueDate: '2024-10-11', id: 1005}
1005: {task: 'Task 1005', status: 'in progress', dueDate: '2024-10-11', id: 1006}
1006: {task: 'Task 1006', status: 'in progress', dueDate: '2024-10-11', id: 1007}
1007: {task: 'Task 1007', status: 'in progress', dueDate: '2024-10-11', id: 1008}
1008: {task: 'Task 1008', status: 'in progress', dueDate: '2024-10-11', id: 1009}
1009: {task: 'Task 1009', status: 'in progress', dueDate: '2024-10-11', id: 1010}
1010: {task: 'Task 1010', status: 'in progress', dueDate: '2024-10-11', id: 1011}
1011: {task: 'Task 1011', status: 'in progress', dueDate: '2024-10-11', id: 1012}
1012: {task: 'Task 1012', status: 'in progress', dueDate: '2024-10-11', id: 1013}
1013: {task: 'Task 1013', status: 'in progress', dueDate: '2024-10-11', id: 1014}
1014: {task: 'Task 1014', status: 'in progress', dueDate: '2024-10-11', id: 1015}
1015: {task: 'Task 1015', status: 'in progress', dueDate: '2024-10-11', id: 1016}
1016: {task: 'Task 1016', status: 'in progress', dueDate: '2024-10-11', id: 1017}
1017: {task: 'Task 1017', status: 'in progress', dueDate: '2024-10-11', id: 1018}
1018: {task: 'Task 1018', status: 'in progress', dueDate: '2024-10-11', id: 1019}
1019: {task: 'Task 1019', status: 'in progress', dueDate: '2024-10-11', id: 1020}
1020: {task: 'Task 1020', status: 'in progress', dueDate: '2024-10-11', id: 1021}
1021: {task: 'Task 1021', status: 'in progress', dueDate: '2024-10-11', id: 1022}
1022: {task: 'Task 1022', status: 'in progress', dueDate: '2024-10-11', id: 1023}
1023: {task: 'Task 1023', status: 'in progress', dueDate: '2024-10-11', id: 1024}
1024: {task: 'Task 1024', status: 'in progress', dueDate: '2024-10-11', id: 1025}
1025: {task: 'Task 1025', status: 'in progress', dueDate: '2024-10-11', id: 1026}
1026: {task: 'Task 1026', status: 'in progress', dueDate: '2024-10-11', id: 1027}
1027: {task: 'Task 1027', status: 'in progress', dueDate: '2024-10-11', id: 1028}
1028: {task: 'Task 1028', status: 'in progress', dueDate: '2024-10-11', id: 1029}
1029: {task: 'Task 1029', status: 'in progress', dueDate: '2024-10-11', id: 1030}