Project Documentation: Todo App (with Spaces + Realtime + AI)

1. Signup/Login Page Progress (Completed)

Feature Summary

- Signup and login pages built using SvelteKit.
- Authentication handled using JWT stored in HTTP-only cookies for security.
- Backend logic includes user creation, password hashing (bcrypt), and token generation.
- Basic form validation implemented on the frontend.

🔍 2. References and Helpful Docs

📚 **SvelteKit and Form Handling**

- SvelteKit Forms: https://kit.svelte.dev/docs/form-actions
- SvelteKit Routing: https://kit.svelte.dev/docs/routing
- `+page.server.js` usage (for backend logic): https://kit.svelte.dev/docs/load

3. Bugs and Challenges Encountered

Issue	Description	Solution
JWT not setting properly	Tokens weren't stored after login due to using regular cookies	Used `setCookie` in `handle` hook with `httpOnly: true, path: '/'`

2. Group Creation & Role-Based Access Control

Feature Summary:

- **Users can create named groups ("Spaces").**
- **Every user in a group is assigned a role, which governs permissions.**
- **Supported roles and their capabilities:**

Role	View Todos	Edit Todos	Add/Remove Users	Edit Group / Delete Group
Member	✅	❌	❌	❌
Elder	✅	✅	❌	❌
Co-Lead	✅	✅	✅	❌
Admin	✅	✅	✅	✅

- **Role data is embedded with the group-user mapping, e.g., `{ userId, role: 'elder' }`.**
- **Backend restricts access at the service/controller level based on roles.**
- **Frontend dynamically hides buttons/actions based on current user's role.**

Challenges Faced

Issue	Description
Frontend-backend integration	Getting real-time group permission UI synced with backend checks was messy at first.
JWT user desync	After login, `locals.user` was not immediately available in form submissions or page loads.
Cookie path issues	Login tokens weren't applying correctly because cookie path wasn't set.

Race conditions with load functions	**On slow networks, sometimes roles weren't fetched before UI rendered.(for example i delete one from user1 and user2 still able to see that in his laptop and try to delete same because he has not got information that the to do is already deleted but not got the updated on ui) -solve using socket.io**

---

3. Group Invite System

Feature Summary:

- **Group Admins & Co-Leads can invite users by email or username.**
- **Invite is stored as a pending notification in the backend.**
- **Users get real-time invites via Socket.IO (`invite:received`).**
- **Accepting invites moves the user to the group with a member role by default.**
- **Declining the invite removes it from the notification store.**

Challenges Faced

Issue	Description
Socket room mismatch	**Users weren't subscribed to their own room on reconnect → missed invite notifications.**
API and Socket race conditions	**Accepting invites via API didn't always sync with the updated socket state.**
Frontend fetch not reflecting invite updates	**After accepting invite, dashboard still showed user as "not part of group" until manual refresh.**
Access control race condition	**Sometimes invites were sent to users who already belonged to the group.**

4. Real-Time Collaboration & Keyboard Shortcuts

Feature Summary

We added real-time collaboration using Socket.IO, allowing users in the same group to see updates to todos instantly — whether it's creation, edit, or status changes. This also includes online status syncing and reconnection handling.
Alongside, we implemented keyboard shortcuts to improve productivity — enabling quick task creation and navigation via custom key combinations.

Key Components

⚙ Server-Side

- **Standalone Socket.IO server for cross-client communication**
- **Group-based room architecture (`group:<id>`, `user:<id>`)**
- **Disconnection cleanup + online presence tracking**

Client-Side

- **Svelte store (`stores/socket.js`) for managing socket state**
- **Realtime toast notifications + event-driven UI sync**
- **Platform-aware keyboard shortcut bindings**

Challenges Faced

Issue	Description	Solution
1. High Latency in Updates	Noticed ~1s delay in UI sync between clients	Prioritized WebSocket transport, reduced payloads, used `volatile.emit()` for non-critical
2. Delivery Reliability	Needed balance between speed (updates) and guarantee (creates/deletes)	Used `volatile.emit` for updates, standard `emit` for creates/deletes, fallback retry logic

3. Connection Cleanup	Disconnecting sockets weren't updating online status in group	Tracked users per group, removed from room on disconnect, broadcasted via `user:disconnected`
5. Browser Shortcut Conflicts	`Cmd+N` triggered browser new window	Switched to less-used modifiers for example:-(`Alt+Shift+N`)
6. Cross-Platform Variations	Shortcut keys varied on macOS vs Windows	Used `navigator.platform` check to assign keys dynamically
7. App Focus Detection	Shortcuts fired even when user wasn't active in tab	Used `document.visibilityState` to scope listeners only when app is focused

---

Implementation Highlights

- Users join private (`user:<id>`) and group (`group:<id>`) socket rooms
- Events like `todo:created`, `todo:updated`, `invite:received` broadcast to those rooms
- Online presence tracked per group and sent to frontend in real-time
- Keyboard shortcuts scoped to active app sessions only and mapped per platform

---

📈 Performance After Optimization

Metric	Before	After
Update Latency	~1000ms	~200ms
Memory Footprint	High	Moderate
Socket Bandwidth	Heavy	Optimized
Connection Recovery	Spotty	Resilient

Challenges and Errors faced during the Gemini 2.0 Flash Model integration with the Todos;

- The API Key for the model was not working properly despite of using the pre-enabled API key for the model from https://aistudio.google.com/apikey
- The env.js not recognizing the GEMINI_API_KEY from the environment variables
- Getting the “Exhausted Resources” error from the Model due to too many API Calls - HTTP Status code 429
- Additional Parsing needed for Gemini Responses due to variability in responses sent by the Model
- Handling the partial input from user to generate the whole To-do through the model
- Invalid API keys problem due to credentials issue in the GCP.

How I Solved those errors:

- Went on to see 2-3 Youtube videos for the integration of the Gemini model to the Application. Used postman to check the working of the model that I was going to integrate using its API key and code block

```
curl
"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=GEMINI_API_KEY" \
-H 'Content-Type: application/json' \
-X POST \
-d '{
  "contents": [{
    "parts":[{"text": "Explain how AI works"}]
    }]
   }'
```

After I found out that the model was working using Postman, But I was giving Errors when I used it for To-do Generation from the Application, I figured out that there is surely some error in the integration of Model in the Backend

So For that I first made an array of mock responses that will be sent to the user no matter what the to do is in order to check where the error was. And I found out that the mock responses were working, so this made it 100% sure that there is some error in the configuration or integration of the model with the backend,

After that I started from the first step and went on to check for every possible error that could’ve led to this And I found out that the env.js was not recognizing the Gemin’s Api key from the .env file

So I kept it in such a way that it will either use the API key from the .env file or it will use the key that is present on the RHS of the OR operation and that is the same key that is present in the .env and after this it worked.

```
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import { dirname, resolve } from 'path';
import { readFileSync } from 'fs';

// Load .env file
dotenv.config();

const __dirname = dirname(fileURLToPath(import.meta.url));
const envPath = resolve(__dirname, '../../.env');

try {
   const envFile = readFileSync(envPath, 'utf-8');
   const envVars = dotenv.parse(envFile);

   // Set environment variables from .env file
   Object.entries(envVars).forEach(([key, value]) => {
       if (!process.env[key]) {
           process.env[key] = value;
       }
   });
} catch (error) {
   console.warn('No .env file found or error reading it');
}

// Export environment variables - ensure defaults are set
correctly
export const MONGO_URI = process.env.MONGO_URI ||
'mongodb://localhost:27017/todo-app';
export const JWT_SECRET = process.env.JWT_SECRET ||
'fallback_secret_key_do_not_use_in_production';
export const GEMINI_API_KEY = process.env.GEMINI_API_KEY ||
'AIzaSyC4a4kwcr5swbJeUa_U0aCdHnLfzOXuPrM';
```

- For handling exhaustion of resources due to too many API calls to the models, I implemented debouncing in JS after getting some help from the Claude 3.7 sonnet model and this article - https://dev.to/aneeqakhan/throttling-and-debouncing-explained-1ocb
- The debouncing works in the following way:

Each keystroke by a user calls the handleInput() function in the TodoInput.Svelte component. This handleInput() function implements the 500ms debounce patterns.
A new timeout interval is created after a keystroke that indicates that after no typing for 500ms a fetchSuggestions() request will be sent to get AI suggestions
Whenever a new keystroke is registered the previous timeout is cleared and the new timeout is set - So this prevents sending API calls while typing as the older timeouts are getting cleared and new timeouts are being set
And After 500ms of no typing, fetchSuggestion() function checks if the input length is greater than 2 characters and if it is then it send a request for AI suggestions generation.

```
function handleInput(): void {
 if (debounceTimeout) {
   clearTimeout(debounceTimeout);
 }
 debounceTimeout = window.setTimeout(() => {
   fetchSuggestions();
 }, 500); // 500ms debounce
}
```

- For partial To-dos, I"m explicitly telling the model to the work with partial to in the Generalized Prompt for every to do:

```
   const prompt = `
   You are a helpful todo list assistant. Based on the following partial
todo item, suggest 3 clear, concise, and specific ways to complete it.
   Partial todo: "${userInput}"
   `;
```

- And for parsing of Gemini Responses, I took the help of Claude and Understood the Gemini's Response format and added constraints according to it

```
const suggestions = text
  .split('\n')         // Splits the response into lines
  .map(line => line.trim())  // Removes extra whitespace
  .filter(line => line.length > 0 && !line.startsWith('-'))  // Keeps
only non-empty lines & removes bullet points
  .slice(0, 3);        // Takes only the first 3 suggestions
```

Figured out the overall configuration and integration of Gemini to the Application though the below resources:

https://youtu.be/osKzYw6xdig?si=xAbqeGYDVseUJxbG
https://youtu.be/Xflxonmx0_0?si=KM4VDOfrUYDFys5c
https://ai.google.dev/gemini-api/docs

Challenges and Errors faced configuration and integration of TypeSense with the Todo Application:

- Issues in the Cloud TypeSense due to Free version of the service
- TypeSense search failures due to connectivity issues
- Error status codes breaking the UI when Typesense failed
- Queries containing colons (e.g., "Work:1") were causing Typesense search to fail

How I Solved those errors:

- After getting too many connectivity issues from the free version of TypeSense Cloud. I configured it locally using Docker. Installed the TypeSense locally allowing the application to use the local .env configurations

```
// Configurable connection parameters with fallbacks
const typesenseClient = new Typesense.Client({
  nodes: [
    {
      host: process.env.TYPESENSE_HOST || 'localhost',
      port: parseInt(process.env.TYPESENSE_PORT || '8108'),
      protocol: process.env.TYPESENSE_PROTOCOL || 'http'
    }
  ],
  apiKey: process.env.TYPESENSE_API_KEY || 'xyz',
  connectionTimeoutSeconds: 2  // Short timeout prevents hanging
});
```

- And even after all these what if the TypeSense still failed, so in order to prevent the breaking of the Application due to TypeSense issues. I implemented the Fallback search through MongoDB that would work inplace of TypeSense if it failed which will keep the app constantly running.

```
export async function searchTodos(userId, query) {
  try {
    // If Typesense is not available, fallback to basic search
    if (!typesenseAvailable) {
      console.log('Typesense not available, using fallback search');
      return fallbackSearchTodos(userId, query.trim());
    }

    // Normal Typesense search...
  } catch (error) {
    console.error('Error searching todos in Typesense:', error);
    // On error, use the fallback search
```

```
    return fallbackSearchTodos(userId, query.trim());
  }
```

- In order to prevent the Error codes from breaking the UI of the system, returned 200 status code even when the search infrastructure failed but included a signal **fallback: true** to indicate the currently the fallback mode is being executed
  Implemented the frontend gracefully for this

```
// Always return 200 with fallback flag instead of error status
return res.status(200).json({
  message: 'Using fallback search due to unhandled error',
  error: error.message || 'Unknown error',
  fallback: true
});
```

```
{#if usingFallback}
  <div class="mt-1 text-xs text-gray-500">
    Using basic search mode (Typesense not available)
  </div>
{/if}
```

- And for the Queries with the Colon or a Number Format Causing Search Failures, I made a strict check for queries having those characters and stripped the colons and trailing content before sending it to the TypeSense
  Also implemented Logging in the Terminal to keep track of any issues during Search or query transmission

```
// Check if query has a colon and number format (like "Work:1")
if (rawQuery && rawQuery.includes(':')) {
    // Remove the colon and anything after it
    rawQuery = rawQuery.split(':')[0];
    console.log('Fixed query after removing colon:', rawQuery);
}
```

Resources which I used for TypeSense:
https://typesense.org/docs/
https://youtu.be/Dnovj-DR_tY?si=unhG5c7xgaX9kNZ8

With the below logging checks for TypeSense:

```
Search request received:
URL: /todos?query=C
Query params: {}
Request params: [Object: null prototype] {}
Request path: /api/search/todos?query=C
Extracted query from URL manually: C
Express search controller received query: C
Searching Typesense with user ID: 6823802470f5efd2fa61741c and query: C
Typesense search service: Searching for "C" for user ID:
6823802470f5efd2fa61741c
Typesense search parameters:
{"q":"C","query_by":"title","filter_by":"userId:6823802470f5efd2fa61741c
","sort_by":"created_at:desc","per_page":100,"highlight_full_fields":"ti
tle"}
Typesense initialized successfully
Typesense found 4 matching todos
Search found 4 results
```

- Also found out that in Linux, the TypeSense stops itself after powering of the PCs and the same goes for MongoDB, so had to save commands like systemctl start mongod and systemctl start typesense to start them everytime after a restart.