# Software Design Specification

**for**

# Notelee

**Version 2.0 approved**

**Prepared by Deja Hintzen, Josh Warren, Christian King**

**Kennesaw State University**

**Fall 2023**

**12/3/2023**

## Table of Contents



# 1. Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| Christian King | 9/24/23 | Document Revision | 1.0 |
| Christian King | 12/3/23 | Document Revision | 2.0 |

# 2. Introduction

The System Design Document (SDD) discusses the components of Notelee, a note-taking Google Chrome plugin, and its low and high-level system architecture. Using the

requirements document, the SDD provides an overview of the functional and nonfunctional software components of the system:

- Revision history.
- What was already assumed before and during the design process.
- The software and methods Notelee is dependent on to help with functionality.
- Constraints that restrict some design aspects of Notelee.
- Goals we want to achieve with creating and managing Notelee.
- Methods used for the frontend and backend development process.
- Design decisions that affect the overall structure of the system.
- Notelee's system architecture.
- Policies to follow when designing and developing Notelee.

#### 2.1.1. System Overview

Notelee is a Chrome extension designed to improve Chrome's functionality by enabling users to take notes through our user interface, either for/on a webpage or as generic notes. The desired functionalities include creating/editing/deleting notes, highlighting and making notes on any webpage, voice notes and drawings, grouping notes into folders, color customization, and font style adjustment. The core design strategy is to use HTML with stylesheets as the frontend and Javascript as the backend, together with a database system powered by Firebase.

# 3. Design Considerations

## *3.1. Assumptions and Dependencies*

This section lists any dependencies the software system has and explains any assumptions made during the design process. Assumptions include:

- Internet connectivity: It is assumed that the user has internet connectivity because the plugin will use various APIs to change the webpage as well as fetch and save notes to Firebase.
- Notelee was created primarily to be a Google Chrome plugin. Chrome must be installed and updated on a regular basis because it is incompatible with any other browser.
- Computer Knowledge: Users and developers must have basic computer knowledge, such as how to install and manage browser extensions, navigate web pages, and interact with online platforms.
- Javascript and HTML support: Javascript and HTML support are required for the plugin to work properly.

Dependencies include:

- Firebase access is required for the plugin to connect to and interact with the database in order to save and retrieve data from Notelee.
- Software Developers: Developers with experience in Javascript and web page construction are critical to Notelee's success because they will be in charge of connecting the plugin with Firebase and guaranteeing smooth functionality.
- Testing: All files and functions must go through testing to help developers find any problems or defects that were previously undetectable. This guarantees that the note-taking plugin is functional.

## *3.2. General Constraints*

- Compatibility with web browsers: Notelee is only designed to work with Google Chrome. Other browsers, such as Opera and Microsoft Edge, are not being examined, therefore the plugin may not function properly.
- Programming languages and web development: Other than javascript for logic coding and event handling for html5 user interface, no other programming language can be utilized. In order for data to be transported through Chrome, manifest.json is required.
- Performance: It is preferred that the plugin works fluidly and without latency as data and usage grow.
- Network: If there is no or slow internet access, Notelee must still work on creating notes and folders. If not, error handling is required.

## *3.3. Goals*

Our primary goal with Notelee is to provide a fully functional, error-free plugin by December. Its capabilities include the following:

- Using web page features such as buttons and text spaces, you may create, delete, and edit notes and folders.
- A simple, cute, and basic user interface.
- Interaction with firebase authentication and realtime database for logging in, registering, and editing notes/folders for the user’s data.
- Ability to delete accounts from Notelee and auto logging in.

Developmental coding goals are:

- Easily readable code with correct formatting.
- More than 90% of the code has a comment telling the reader what the line is used for.
- Passing all tests for all functions.

### *3.4. Development Methods*

Methods and approaches for Notelee system and software design include web development with object-oriented javascript, html, and css programming. Additionally, every plugin must include a manifest.json file that will transfer data into Chrome via our plugin.

Frontend:

- HTML for the user interface using our preferred IDE.
- CSS dynamic visuals using our preferred IDE.

Backend:

- Firebase is used to save user-transmitted data into Notelee.
- Logic coding, APIs, and interactions in Javascript utilizing our chosen IDE.
- Tools like Github are used to share code and track developed versions.

Furthermore, there will be unit tests for each Notelee version generated, with tests conducted using Chrome and continuous and regular team meetings.

## 4. Architectural Strategies

Design decisions that affect the system include:

- Creating a Google Chrome extension. This implies we must adhere to the programming requirements for Chrome plugins of manifest version 3, such as only using specific programming languages as well as how we should code in Java script, such as no inline event handlers, resulting in us to code differently than normal. Chrome also requires us to utilize only HTML and CSS for the plugin's user interface.
- Using HTML, we can create a popup window for Notelee which enables the user to interact with it by including various buttons and text areas.
- Using CSS, we can visually style Notelee by changing the color, margins, font styles, alignments, popup size, and more.
- Using a database affects the system by requiring developers to code in ways to add and retrieve information from the database.
- Creating the user in the realtime database using the user’s id and email.

## 5. System Architecture

Using a manifest.json file, the developers can load the Notelee folder in Chrome and add it as an extension. The frontend then enables a popup with which the user can engage in a variety of ways, all utilizing HTML and CSS through the user's browser. Our user interface allows the user to create, edit, and delete items, while our backend database, which uses Firebase, continuously saves the changes made. Additionally, the backend is running javascript while using event listeners and handlers. Using the web server, the plugin can utilize Chrome

Extension APIs to read and write to the google chrome storage. Our frontend user interface and our backend logic coding can communicate with each other through HTTP requests.

# 6. Policies and Tactics

- Coding tasks are evenly distributed among the three of us while we collaborate on Github. It is important to us that the code is easy to understand with documentation and comments throughout the code.
- Notelee should not directly plagiarize from already existing note taking applications. We should document where our ideas come from.
- Notelee should be tested after every new version to find bugs. This requires the software test plan STP document where we will use a table to describe the function, what it should do, and whether it passed or failed the test.
- As a team, we must communicate with each other with ideas, questions, or concerns.

# 7. Detailed System Design

## *7.1. background.js*

**Classification**: Javascript file for Notelee.
**Responsibilities**: To check the chrome storage for userdata with an active status. If found, load their folders. If not, Notelee stays on popup.html.
**Functions**:

- **window.onload:** Gets the userdata from chrome storage. As a result, saves the user data as userData. If userData exists and the status is “active”, saves the user’s id and removes the welcome menu to add the folder navigation and the manage account button. Then, calls loadFolderEventListener("login", userid) after adding a loading screen.

## *7.2. popup.html*

**Classification**: HTML file for Notelee.
**Responsibilities**: Serves as the default popup display when opening Notelee. Displays a login and register button with a welcome message.

## *7.3. accountManagement.js*

**Classification**: Javascript file for Notelee.
**Responsibilities**: To allow users to interact with the login and register buttons in popup.html. Allows users to login, register, and send a password reset email. Displays the tutorial popup. Adds a manage account button along with event listeners to sign off and delete their account.
**Functions**:

- **replaceAccountMenuWithLogin():** removes the welcome menu from popup.html and adds a login menu with 2 input boxes, a “login” button and a “forgot password” button. Adds event listeners to the login button and the forgot password button.
- **replaceAccountMenuwithRegister():** removes the welcome menu from popup.html and adds a register menu with 3 input boxes for email, password, and confirm password. Also adds a “register” button

- **sendPasswordReset():** removes the login menu and adds an input field for an email with a “reset password” button. Adds event listener to the reset button when clicked to get the text entered in the input field and use firebase function sendPasswordResetEmail(auth, email) to send a password reset email which will update their password in firebase.

- **tryLogin():** gets the email and password entered. If the fields are not empty, use firebase function signInWithEmailAndPassword(auth, email, password) to see if they are listed as an authenticated user. If they are, then find their reference in the realtime database using their user id. If it exists, save their user data. If the user has verified their email, update their chrome storage to save their email, id, and status as “active”. Then, remove the login menu, add the folder navigation bar, add the manage account menu, and call loadFolderEventListener("login", userId) to load the folder buttons and load folder 1.

- **register():** calls replaceAccountMenuwithRegister(). Gets the text entered into the input fields. If the password value is less than 6 characters, then alert the user’s computer. If the password field does not match the confirmed password field, then alert the user’s computer. Otherwise, call firebase function createUserWithEmailAndPassword(auth, email, password) to create them as an authenticated user. Save the user’s credentials and call firebase function sendEmailVerification(user) to send a verification email, then alert the user that it has been sent. Create userData as the user’s id, email, and 4 empty folders. Create the chromeUserDat as their email, id, and status as empty. Set the chrome storage data and the realtime database reference, then reload the plugin.

- **addManageAccountButton():** Exported function that adds a transparent person icon to the right of the question mark icon in the top bar.

- **showTutorial():** Exported function that adds mouseover functions to the question mark icon to change the opacity when and when not hovered by the mouse. Adds a click event listener that creates a popup window with a tutorial html inside.

- **manageAccountEventListener(userId):** Exported function. Saves the email of the current user. Gives mouseover event listeners to the manage account

button that increases and decreases its opacity. Adds click event listener that removes all possible displays and adds a Manage Account menu that uses the email to show the current user of Notelee, along with a log off, delete account, and back to folders button.

Adds mouseover and click event listeners to the back button that changes the border color and goes back to viewing folders when clicked.

Adds mouseover and click event listeners to the logout button that changes the border color and signs the user off of Notelee when clicked by setting the user's data in chrome storage as "inactive."

Adds mouseover and click event listeners to the delete account button that changes the border color and adds a new display when clicked that shows a warning message with a yes and a no button. Creates a click event listener to both buttons. Clicking no goes back to viewing folders. Clicking yes gets the current authenticated user and deletes their information, along with removing the user's reference in the realtime database using the userid. Reloads the plugin.

## *7.4.folders.js*

**Classification**: Javascript file for Notelee.
**Responsibilities**: To let users read and write to their folders by clicking on the folder navigation buttons. Loads notes onto the display.
**Functions**:

- **loadFolderEventListener(string, userId)**: Exported function. Make the background color of the document white if it was changed. Using a for loop, gives each folder button a mouseover and mouseout function that changes the border color between pink and black. Additionally, the for loop adds a click event listener to load the notes for the folder when clicked. If the string passed into the function is "login", automatically load the folder notes for 1. Then, add event listeners to the manage account button and the tutorial button from accountManagement.js.

- **loadFolderScreen(folderNum, userid)**: Removes the account management menu if it's there. If there is a list of notes, remove it. Add a no notes display (createNoNotesPage()), a create note button with its event listener (createBottomBar(folderNum), createNoteEventListener(folderNum, userid)), load the folder buttons (loadFolderEventListener("alreadyLogged", userid)), and remove any displays that are not needed.

- **makeNoteList(folderNum, notes, userId)**:Gets the folder's reference in the database using the userId and the folderNum. If it exists, save the folder data. For each note inside of array notes (from loadFolderNotes(folderID, userId)), make the note in notes with a note in the database and saves the id of the note into the array noteIDFB. Then, separates the title and text of each note and pushes them into their respective arrays.

If there are no titles, that means the array was empty. Call loadFolderScreen(folderNum, userId). Otherwise, call function appendNoteList().

- **AppendNoteList()**: Removes any displays. Using a for loop, creates a table of note buttons and delete buttons along with a unique html id for each, then calls buttonColor(noteButton, i), createBottomBar(folderNum) and createNoteEventListener(folderNum, userId). Adds a click event listener to the note buttons using function loadNote(folderNum, id, userId) (can read this function in notes.js) and a click event listener to the delete buttons using function deleteNote(id, userId, folderNum).

- **loadFolderNotes(folderID, userId)**: Exported function. Call folderColor(folderID). Gets the database and a reference for the current folder in the database using the userId and the folderID. Creates an empty array called notes. If the folder exists, save the folder's data. For every note in the folder data, add the note into the notes array. Then, call makeNoteList(folderID, notes, userId) to make the folder's notes.

  If the folder had no data, then call loadFolderScreen(folderID, userId) to add the no notes display.

- **folderColor(folderID)**: Using a for loop, gets the html id for the folder navigation buttons and makes the background color blue. Get the selected button using folderID and make the background color pink.

- **buttonColor(noteButton, index)**: creates an array of 4 colors. Gives the note button in a note list a background color from the array.

- **removeNoteCreationScreen()**: exported function that removes the title input box, text area, and save button from the note taking display if it is there.
- addFolderNav(): exported function that adds a table of 4 folder buttons onto the document if it is not already there.

- **createNoteEventListener(folderNum, userId)**: adds a mouseover event listener to the create note buttons that changes the border color. Creates a click event listener function to the create note buttons that will call newNote(folderNum, userId).

- **createBottomBar(folderNum)**: Creates a div element that has a create note button. Gives the button a unique id using the folderNum. Adds the button to the document and removes the previous button.

- **createNoNotesPage()**: Creates a table that says "No Notes." adds it to the document and removes it if it's already there.

## *7.5. notes.js*

**Classification**: Javascript file for Notelee.
**Responsibilities**: To let users read and write to their folders by saving notes, deleting notes, or editing notes.
**Functions**:

- newNote(folderNum, userID): Remove all unneeded displays and calls addNewNoteTakingHTML(). Then, gets the save note button and adds mouseover and mouseout functions that change the background color of the button, and adds a click function that calls saveNote(folderNum, userID) when clicked.
- **loadNote(folderNum, noteID, userId)**: Removes unneeded displays and calls addNewNoteTakingHTML() to add the note taking page to the document. Adds mouseover and mouseout event listeners to the save button that changes the background color from light green to dark green. Gets the reference of the folder from the database and states that if the folder exists, save the folder data and add all of the notes into an array. For all of the id's of the notes in the folder in firebase, match the text content of the note to the note in Firebase. If found, save the notes ID and leave the loop. Then, separate the content into the title and text and put them into their respective input fields in the note-taking HTML. Lastly, it adds an event listener to the save note button that states to call saveReloadNote(folderNum, userId, matchingNoteId) if the title input field is not empty.

- **saveReloadNote(folderNum, userId, matchingNoteId)**: Gets the title and text of the input fields and combines them into one note. Then, get the reference to the folder in the database. Get the note's reference from the folder using matchingNoteId, and set the note with id matchingNoteId to be the text of the new note. The original content will be replaced with the new content in the database. Then, add the folder navigation bar (addFolderNav()), then load the notes for the current folder and reload the folder button's event listeners (loadFolderNotes(folderNum, userId) and loadFolderEventListener("alreadyLogged", userId)).

- **saveNote(folderNum, userId)**: Gets the text entered into the title and text input fields. If they are empty, alert the user. Otherwise, combine the title and text into one note, get the user's reference for the folder, and push in the full note in the user's folder. Then, add the folder navigation bar (addFolderNav()), then load the notes for the current folder and reload the folder button's event listeners (loadFolderNotes(folderNum, userId) and loadFolderEventListener("alreadyLogged", userId)).

- **deleteNote(deleteNoteButtonID, userID, folderNum)**: Takes the deleteNoteButtonID and removes the word "delete" to get the note's id in firebase. If a reference to the note exists in firebase, add a delete confirmation text, a text area that shows the content of the note, and a yes and no button after removing unnecessary displays. Creates event listeners to both yes and no

buttons. When yes is clicked, remove the note's reference in the database and call loadFolderNotes(folderNum, userID) before removing the delete confirmation elements. Then, call loadFolderEventListener("alreadyLogged", userID). When no is clicked, call loadFolderNotes(folderNum, userID) and loadFolderEventListener("alreadyLogged", userID) to reload the folders.

- **addNewNoteTakingHTML()**: Changes the background color of the document to a light pink and removes the folder navigation bars. Creates and adds a title table and a note content table. The title table is an input field with text "Title:" while the note content table is a text area and a save button beneath.

### *7.6. firebase-main.js*

**Classification**: Javascript file for Notelee that defines our firebase database.
**Responsibilities**: To initialize our firebase configurations, including our databaseURL, api key, authentication domain, storage bucket, messaging sender id, and measurement id. Defines our app and database to use in the above javascript files.

### *7.7.firebase-database.js, firebase-app.js, firebase-auth.js*

**Classification**: Javascript files that Notelee imports a variety of functions from for database reading and writing.
**Responsibilities**: Exports various functions that allow Notelee to communicate with firebase and the realtime database. Exported functions include:

- From firebase-auth: getAuth, deleteUser, signInWithEmailAndPassword, sendPasswordResetEmail, createUserWithEmailAndPassword, sendEmailVerification
- From firebase-database: etDatabase, ref, set, get, remove, child, push
- From firebase-main: app

## Appendix A: Glossary

1. **Backend**: Server-side logic of an application that a user cannot see.
2. CSS: Cascading stylesheet. Used to design HTML documents.
3. **Event listener**: Javascript function that runs a piece of code when an event happens.
4. **Extension**: software added to a different piece of software to enhance its features.
5. **Firebase**: Database service from Google.
6. **Frontend**: The user interface of an application.
7. **Functionality**: Operations done by a software system.
8. **HTML**: Most used markup language to create web pages.
9. **Javascript**: Programming language used for websites.
10. **Mouseout event listener**: Defines what happens if the mouse hovers over a specific HTML element.

11. **Mouseover event listener**: Defines what happens if the mouse stops hovering over a specific HTML element.
12. **Notelee**: Note Taking plugin for Chrome created by Christian King, Deja Hintzen and Josh Warren.
13. **Plugin**: software added to a different piece of software to enhance its features.