**Core Functionalities**

**1. Create, Read, Update, and Delete (CRUD) Notes**

* **Create Notes**:
  + A form with fields for the note's title, content, and optional tags.
  + Validate inputs (e.g., title and content are required).
  + Save the note to the database or local storage.
* **Read Notes**:
  + Display a list or grid view of notes with:
    - Title
    - Preview of the content
    - Tags
    - Timestamp
  + Allow users to click on a note to view full details.
* **Update Notes**:
  + Provide an option to edit an existing note.
  + Prepopulate the form with the current details of the note.
  + Save changes back to the database or storage.
* **Delete Notes**:
  + Allow users to delete a note with a confirmation prompt.
  + Remove the note from the database or storage.

**2. Archive Notes**

* Provide an "Archive" button for each note.
* Move archived notes to a separate list or section.
* Archived notes should not appear in the default notes list.
* Allow users to unarchive notes and move them back to the main list.

**3. View All Notes**

* Display all active notes in a grid or list view.
* Include pagination or infinite scrolling for large datasets.
* Allow sorting (e.g., by date created, last updated, or alphabetical order).

**4. View All Archived Notes**

* Separate section or page for archived notes.
* Show archived notes similar to active notes.
* Allow sorting and unarchiving from this view.

**5. View Notes with Specific Tags**

* Display a list of all tags as clickable buttons or a dropdown.
* Clicking a tag filters and displays notes with that tag.
* Combine multiple tags to display notes with any or all selected tags.

**6. Search Notes by Title, Tag, and Content**

* Add a search bar with the ability to:
  + Search by title (partial or full match).
  + Search by tags (single or multiple).
  + Search by content (keywords or phrases).
* Dynamically display search results as the user types.
* Highlight the matching terms in the search results.

**7. Select Their Color Theme**

* Provide a color theme selector (light, dark, and custom themes).
* Persist the selected theme across sessions using:
  + Local storage or cookies for client-side.
  + Database storage for server-side (user-specific themes).
* Apply the selected theme globally to the app.

**8. Select Their Font Theme**

* Provide a dropdown or settings menu with font options (e.g., serif, sans-serif, monospace).
* Preview the selected font style in real-time.
* Save the font preference across sessions.

**9. Receive Validation Messages for Required Fields**

* Validate required fields (e.g., title and content) on form submission.
* Show real-time error messages if:
  + Title or content is empty.
  + Invalid characters or inputs are detected.
* Use accessible validation (e.g., ARIA roles for screen readers).

**10. Navigate Using Only the Keyboard**

* Implement **keyboard accessibility**:
  + Use tabindex to allow tab navigation.
  + Use Enter or Space to trigger buttons and actions.
  + Add shortcut keys (e.g., Ctrl + N for a new note, Ctrl + F for search).
* Use **focus states**:
  + Clearly highlight the currently focused element.

**11. Responsive Layout**

* Design the interface to adapt to different screen sizes:
  + Mobile: Stacked layout, minimal buttons, and collapsible menus.
  + Tablet: Grid layout with larger buttons and more visible details.
  + Desktop: Full layout with sidebars and detailed views.
* Test the layout on various screen sizes (e.g., using browser dev tools).

**12. Hover and Focus States**

* Add visible states for interactive elements:
  + Buttons
  + Links
  + Input fields
* Use CSS to indicate hover (:hover) and focus (:focus) states.

**Bonus Functionalities**

**13. Save Details to a Database**

* Use a **full-stack approach**:
  + Frontend: HTML, CSS, JavaScript (React, Vue, etc.).
  + Backend: Node.js, Django, or other backend frameworks.
  + Database: MongoDB, MySQL, or PostgreSQL.
* Create a REST API or GraphQL endpoint to handle CRUD operations.

**14. User Authentication**

* **Sign Up**:
  + Allow users to create an account with:
    - Email
    - Password
  + Hash passwords using a library like **bcrypt**.
* **Login**:
  + Authenticate users with email and password.
  + Use **JWT (JSON Web Tokens)** or **sessions** for authentication.
* **Change Password**:
  + Provide a form to update the password securely.
  + Validate the old password before allowing the update.

**15. Password Reset**

* Add a "Forgot Password" link on the login page.
* Send a password reset email with a secure token using tools like:
  + **NodeMailer** or **SendGrid**.
* Validate the token on the reset password page.
* Allow the user to set a new password.

**Development Checklist**

1. **Frontend**:
   * Build reusable components for forms, modals, buttons, and lists.
   * Use a CSS framework like Tailwind or Bootstrap for quick styling.
   * Ensure responsiveness and accessibility.
2. **Backend**:
   * Create secure APIs for notes and user management.
   * Use middleware for authentication and validation.
3. **Database**:
   * Design tables or collections for:
     + Users (ID, email, hashed password, preferences)
     + Notes (ID, user ID, title, content, tags, archived status, timestamps)
4. **Testing**:
   * Test each functionality manually and with automated tests.
   * Check for edge cases like empty inputs, invalid data, or network failures.
5. **Deployment**:
   * Deploy the app on platforms like **Netlify** (frontend) and **Heroku** or **Vercel** (backend).
   * Use **MongoDB Atlas** or a hosted database for storage.

Bottom of Form