# Approach your first real project

1. **Choose a Manageable Project**:
   * Start with something small but meaningful. For instance, a simple API for tracking tasks or an app that takes user input and processes it (like a budget tracker).
   * This could even align with backend work you’re already doing, like building a CLI tool to manage user data or a mini CRUD (Create, Read, Update, Delete) application.
2. **Outline Your Project in Phases**:
   * **Phase 1**: Set up the environment and version control (get comfortable with git, creating branches, etc.).
   * **Phase 2**: Build core functionality (e.g., if you’re building a task manager, start with creating a task and listing tasks).
   * **Phase 3**: Add error handling, testing, and maybe even simple concurrency if you’re feeling bold (Go is great for this).
   * **Phase 4**: Deploy it on a platform like GitHub or deploy your API to a free service like Heroku or Fly.io.
3. **Use GitHub Copilot Strategically**:
   * GitHub Copilot is brilliant for writing repetitive code, suggesting functions, and providing examples of syntax you might not be familiar with. But, think of it as a guide, not a crutch.
   * When you see suggestions, take a moment to understand the code Copilot proposes, and don’t hesitate to rewrite parts to reinforce learning.
4. **Seek Guidance and Practice Debugging**:
   * Use online resources and ask questions, but try to debug and troubleshoot on your own first. This is one of the best ways to build confidence and deepen your understanding.
5. **Celebrate Small Wins**:
   * This sounds trivial but is super important. Each time you successfully run a function, fix a bug, or push a commit, you’re leveling up.

## First project: Personal Journal App

* **Frontend Phase**:
  + Build a journal where users can add daily entries, each with a title, date, and text.
  + Keep entries in local storage for now, so the data is persistent even if the page is refreshed.
* **Backend Phase**:
  + Set up a backend API to save, retrieve, update, and delete journal entries.
  + Add user authentication (login and registration), so each user can have a private journal.
* **Why It’s Great**: It starts with a simple front-end, but when you add the backend, you’ll explore CRUD operations and user sessions/authentication—core backend concepts.

## Phase 1: Set Up the Front-End Skeleton ⏳

1. **Project Initialization**: ✅
   * Create a new folder for your project and initialize a Git repository (git init). ✅
   * Set up your HTML, CSS, and JavaScript files (e.g., index.html, styles.css, script.js). ✅
   * If you’re using version control like GitHub, create a repository and push your initial setup. ✅
2. **HTML Structure**: ✅
   * Create a simple layout with sections for the **journal entry form** (where users will type entries) and **entry display** (where entries will be listed). ✅
   * Basic HTML structure:
     + A header with the journal title. ✅
     + An input area for title of the entry and a larger textarea for the entry text. ✅
     + A submit button to save the entry. ✅
3. **CSS Styling**: ⏳
   * Style your layout for readability:
     + Set a basic color scheme, use padding/margins for spacing, and add a nice font.
     + Use flexbox or grid for layout, making sure the entry form and entry list are well-separated.
4. **JavaScript Setup**:
   * Link your JavaScript file in index.html and add a console.log message to verify everything is connected properly.

## Phase 2: Adding Core Functionality

1. **Create and Display Journal Entries**:
   * Add JavaScript to capture data from the form inputs (title and text).
   * Create a function to display the entry below the form, dynamically updating the list of entries.
2. **Store Entries Locally**:
   * Use localStorage to save journal entries, so data persists after refreshing the page.
   * When the page loads, check for existing entries in localStorage and display them.
   * Save each entry as an object with properties like title, content, and date for future use.
3. **Basic Interactivity**:
   * Add a delete button next to each entry to remove it.
   * Add a simple confirmation prompt before deleting an entry (to prevent accidental deletions).
   * Update localStorage to reflect deleted entries.
4. **Testing**:
   * Test adding and deleting entries, refreshing the page, and confirming that entries persist correctly.
   * If any issues arise (like entries not appearing or disappearing), debug using console.log.

## Phase 3: Enhancing User Experience

1. **Date and Time Stamps**:
   * Automatically add a date and time to each entry when it’s created.
   * Display this timestamp next to each entry in the list.
2. **Improved Layout and Styling**:
   * Make the entry display area scrollable if there are many entries.
   * Add hover effects or subtle animations to buttons for a polished look.
3. **Editing Entries**:
   * Add an edit button next to each entry to allow users to update existing entries.
   * Pre-fill the form with the selected entry’s data when editing and save changes to localStorage.
4. **Mobile Responsiveness**:
   * Add media queries in CSS to ensure the layout is mobile-friendly.
   * Adjust text sizes, button sizes, and entry spacing for smaller screens.

## Phase 4: Moving to Backend (Setting Up an API)

1. **Set Up a Basic Server**:
   * Create a simple backend server using Node.js and Express.
   * Define routes for CRUD operations on journal entries (GET, POST, PUT, DELETE).
2. **Database Integration**:
   * Use a database like MongoDB or PostgreSQL to store entries.
   * Define a schema/model for journal entries with fields like title, content, date, and userID (for Phase 5).
3. **Connect Frontend with Backend**:
   * Replace localStorage with API calls to fetch, add, edit, and delete entries.
   * Update the front-end JavaScript to handle API responses and error cases.
4. **Testing the API**:
   * Test your API using Postman or Insomnia to ensure all routes are working as expected.
   * Test the frontend to verify that entries are saved in the database and persist across devices.

## Phase 5: Adding User Authentication and Personalization

1. **User Registration and Login**:
   * Create routes for user registration and login in the backend.
   * Hash passwords securely (e.g., using bcrypt) and store users in the database.
2. **Authentication in the Frontend**:
   * Add a simple login form and integrate it with the backend.
   * Store a user session/token to manage login status (e.g., with JWT).
3. **Link Journal Entries to Users**:
   * Modify the API to ensure each entry is associated with a specific user.
   * Display only the logged-in user’s entries on the front end.
4. **Testing Authentication and Personalization**:
   * Test creating multiple accounts and verify that users can only access their own entries.

## Phase 6: Advanced Features (Optional Enhancements)

1. **Rich Text Editing**:
   * Add support for rich text editing, allowing users to format their entries with bold, italics, etc.
2. **Search and Filter**:
   * Add a search feature to allow users to filter entries by keyword or date.
3. **Dark Mode Toggle**:
   * Add a toggle button to switch between light and dark themes for user preference.

## Phase 7: Deployment

1. **Deploy Backend and Frontend**:
   * Host your backend on a platform like Heroku or Fly.io.
   * Deploy your frontend on GitHub Pages, Netlify, or Vercel.
   * Ensure the app functions well on deployment, and troubleshoot any issues that may arise.
2. **Final Testing and Debugging**:
   * Test thoroughly to ensure that everything works smoothly in a production environment.

**1. Create the GitHub Repository**

1. **Go to GitHub**:
   * Visit [github.com](https://github.com/) and log into your account.
2. **Create a New Repository**:
   * Click on **New** (next to your profile icon) or go to [this link](https://github.com/new).
   * Name your repository **Reflekt**.
   * Add a description if you like, something like “Personal Journal App built with HTML, CSS, and JavaScript.”
   * Set the repository to **public** if you want others to view it, or **private** if it’s for personal use only.
   * **Do not** initialize with a README, .gitignore, or license since you already have a README.md locally.
3. **Create the Repository**:
   * Click **Create repository**.

**2. Connect Your Local Project to GitHub**

In your VS Code terminal, navigate to the **Reflekt** project folder if you’re not already there.

Run these commands to connect your local project with the new GitHub repository:

bash

Copy code

# Initialize Git if you haven't already done so

git init

# Add all files to staging area

git add .

# Commit your changes with a descriptive message

git commit -m "Initial commit - set up project structure"

# Link your local repo to the GitHub repository

git remote add origin https://github.com/Julinhio/Reflekt.git

# Push your code to GitHub

git push -u origin main

**3. Enable GitHub Pages**

1. **Go to the Repository Settings**:
   * Navigate to the **Settings** tab in your newly created Reflekt repository on GitHub.
2. **Set Up GitHub Pages**:
   * Scroll down to the **Pages** section (usually near the bottom).
   * Under **Source**, select the branch main and set the folder to / (root).
   * Click **Save**.
3. **Wait for Deployment**:
   * GitHub will now deploy your project. After a moment, you’ll see a link to your site (something like https://Julinhio.github.io/Reflekt). This is your live GitHub Pages URL!

**4. Update the README with the GitHub Pages Link**

* Add your live site URL to the README.md under a section called **Live Demo** or **GitHub Pages**.
* Example:

## Live Demo

You can view the project live at [Reflekt on GitHub Pages](https://Julinhio.github.io/Reflekt).