1. Requirement Analysis

Tool used: ChatGPT

Prompt:

Generate a comprehensive Software Requirements Specification (SRS) document for a Personal Task Manager Application. The application should be developed using Python, Streamlit, and SQLite3 for database management and user authentication. The SRS should include the following sections:

1.Introduction

* Purpose of the document
* Scope of the application
* Intended audience

2. Overall Description

* Product perspective
* Product functions
* User classes and characteristics

3. Functional Requirements

* User authentication (login, registration)
* Task management (add, edit, delete tasks)
* Task categorization (tags, priorities)
* Reminders and notifications
* User preferences (theme settings)

4. Non-Functional Requirements

* Performance requirements (response time, load handling)
* Security requirements (data encryption, user privacy)
* Usability requirements (user interface design principles)

5. User Interface Design

* Description of main screens (dashboard, task view)
* Navigation flow between screens

6. Data Management

* Database schema (tables for users, tasks, categories)
* Data storage and retrieval processes

7. External Interfaces

* APIs or third-party services used
* Integration points with other applications

8. Assumptions and Constraints

* Technology stack limitations
* User environment considerations

9. Dependencies

* Required software libraries or frameworks

Ensure that the document is well-structured, easy to read, and follows best practices in software documentation.

Output Screenshot:

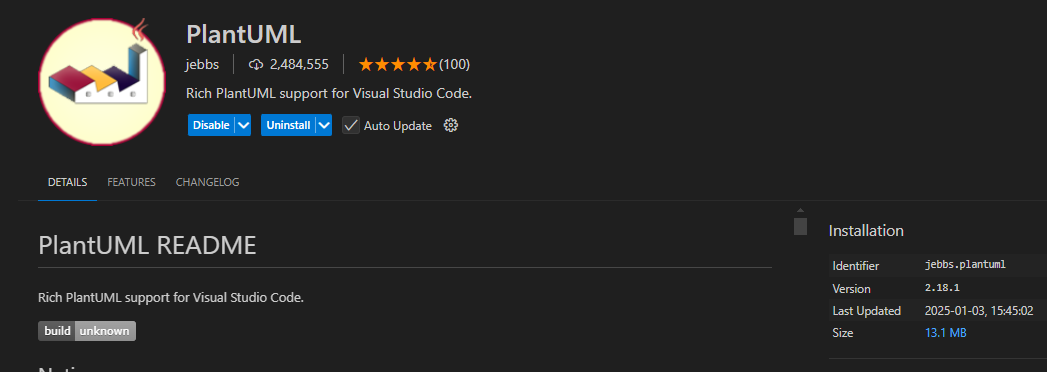


1. Design Phase:

Tool Used: PlantUML with ChatGPT/Any other AI

IDE: VSCode

Install PlantUML extensions in VSCode which can be accessed through sidebar or ctrl + shift + X.

Screenshot:  


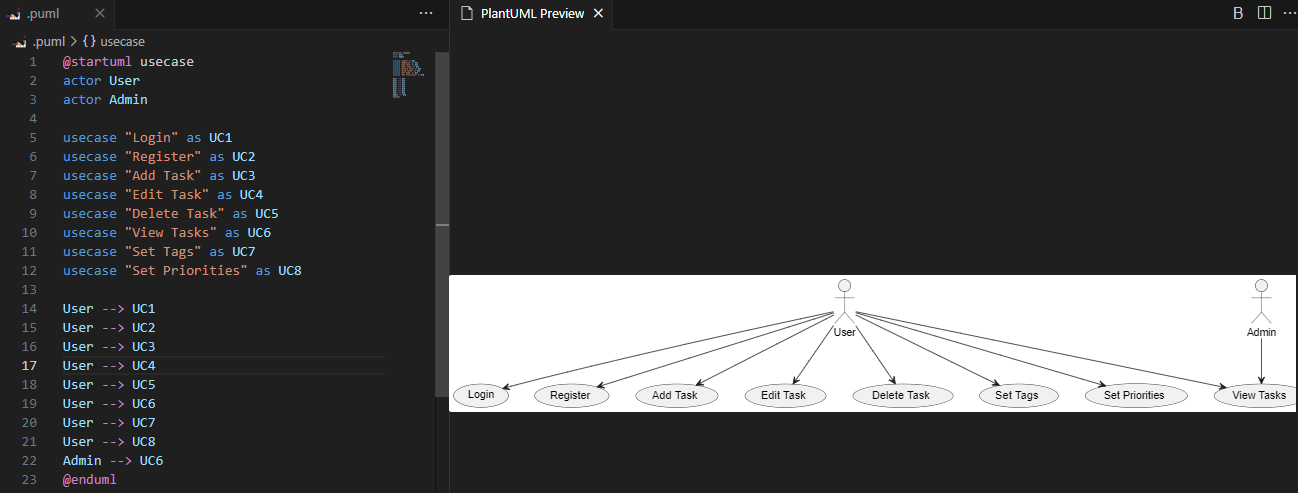
Initially create a .puml file in your working directory.

Query an GenAI(ChatGPT here) to generate PlantUML code for the given SRS document.

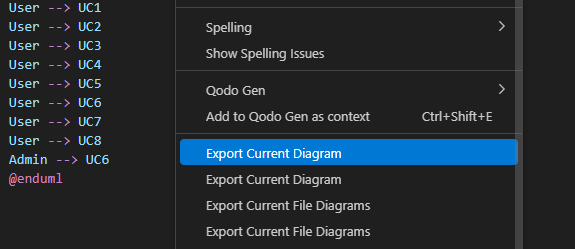
Paste the code in .puml file.

You can use alt + D to preview the diagram.

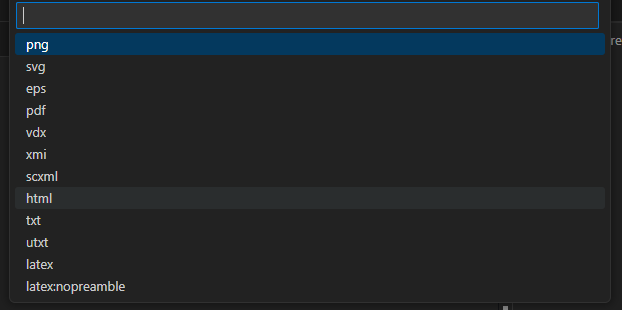
Screenshot:



To export the diagram: Right Click on the code and enter “Export Current Diagram”



You can export the diagram in various formats as shown below:



After exporting the diagram, it creates a new directory called “out”. You can view all your UML diagrams in this directory.

Do the same for all the UML Diagrams and document it.

1. Coding:

Tools Used: TabNine

Check the UML Diagrams to understand what functions/code to be needed.

As we can see in the use case diagram, User needs to Register, Login, Add Task, Edit Task, Delete Task, View Tasks, Set Tags, Set priorites.

So first the user needs to register. For this we are setting a database initially.

Here I am using sqlite3 library in python for database management.

By using TabNine, we can finish the code within seconds.

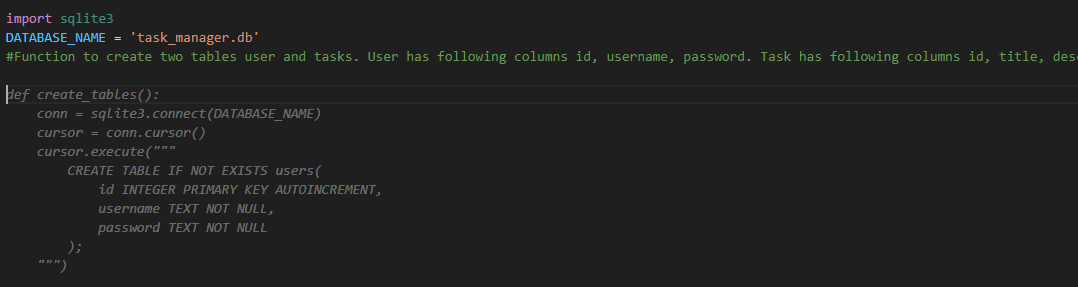
Just write the function name, it autocompletes the code.

Example:

Here I just wrote a comment on what I want to do.

Just to create a table with name “user” and the attributes.

After writing def only it generated code directly. To insert the code just press “Tab”



In the same way wrote the code to create tables, execute query and fetch query.

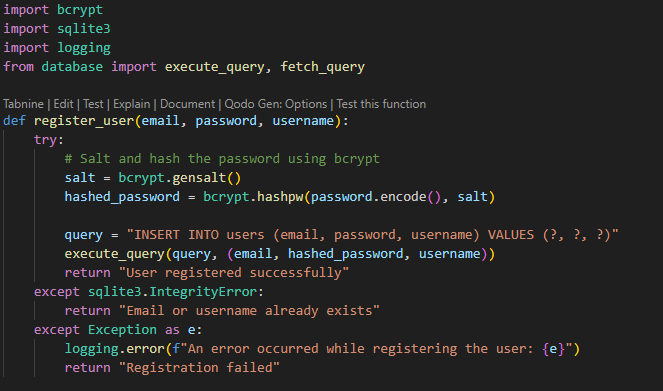
Moving on to User Authentication.

Create a user\_auth.py file.

First, users need to register and login.

So creating register function.

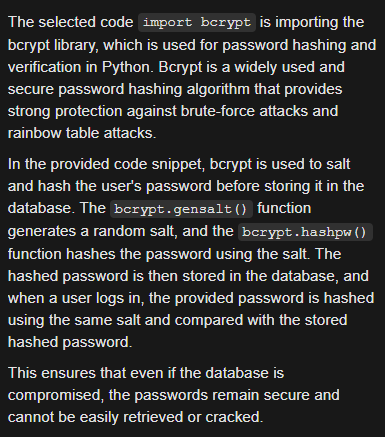
Just add a comment or start defining the function it will give the code as shown in below.



Let’s say you do not know bcrypt module and why it is used.

Just right click, go to TabNine and then click “Explain This Code”. It creates a chat interface at left panel.

Then select the code and press continue



It gives the definition of the library used and it’s functionality.

In the same way, finished coding for the login\_user() function also.

Monving on to task management.

Create a new file task.py. We need to add task, delete task, update task , set priorities and tags.

In the above manner just comment about the function it generates the code.   
Always check the code for any errors or any redundancies.

Moving on to UI:

Here we are using streamlit to build UI.

First install streamlit by using “pip install streamlit”

Start by importing all the modules which we did from start.

**Features**

**Authentication**

The application supports user registration and login. The authentication state is managed using st.session\_state.

**Task Management**

* **Add Tasks:** Users can create tasks with a title, description, due date, priority, and status.
* **Edit Tasks:** Users can modify existing tasks.
* **Delete Tasks:** Users can remove tasks from the list.
* **Task Status Updates:** Users can change the status of tasks (e.g., Pending, To Do, In Progress, Completed).
* **Task Filtering:** Tasks can be filtered by status using the sidebar.

**Streamlit Components Used**

**Page Setup**

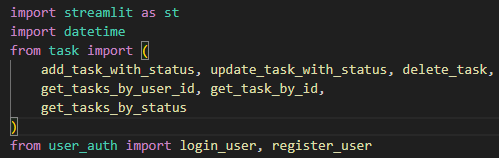


**UI Components**

* st.title(): Adds a title to the page.
* st.header(): Adds a section header.
* st.text\_input(): Provides text input fields.
* st.text\_area(): Creates a multi-line text input box.
* st.date\_input(): Allows date selection.
* st.selectbox(): Provides a dropdown menu for selecting options.
* st.button(): Adds clickable buttons.
* st.sidebar: Allows adding UI elements to the sidebar.

**Code Explanation**

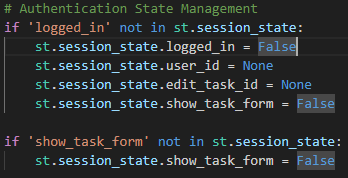
**1. Imports**



* streamlit: Provides the UI framework.
* datetime: Handles date and time operations.
* task and user\_auth modules: Handle backend operations such as task management and user authentication.

**2. Authentication State Management**

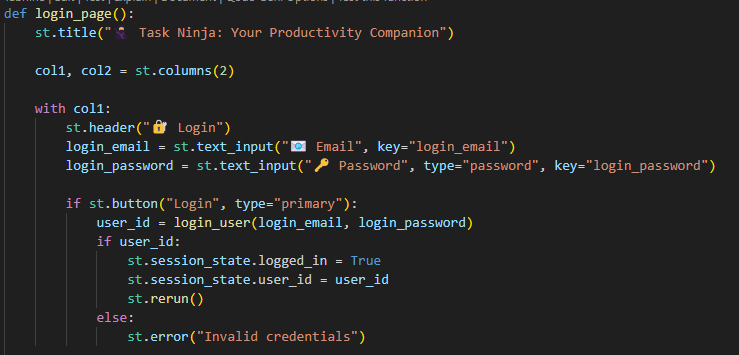
Session states (st.session\_state) are used to manage the user's login status and UI interactions.



1. **Login and Registration**

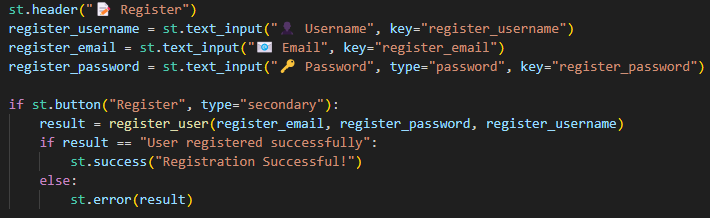
**Login Page**

Allows users to log in or register for an account:



**Register Page**

Enables new users to sign up:



1. **Task Dashboard**

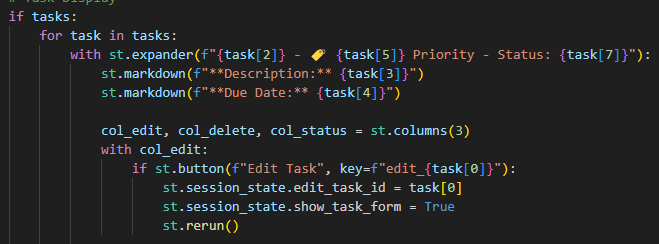
**Task Filtering**

Tasks are filtered by status using the sidebar:



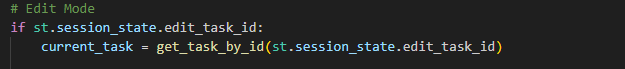
**Task Display**

Tasks are displayed using expandable sections:



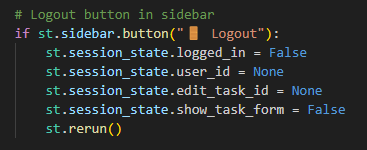
**Add/Edit Task Form**

The form dynamically switches between Add and Edit modes based on user interaction:

****

1. **Logout**

Users can log out using the sidebar button:

****

**Application Flow**

1. Login or Register: Users are presented with a login and registration page.
2. Task Dashboard: Logged-in users can view, filter, and manage their tasks.
3. Add/Edit Task: Users can add new tasks or edit existing ones.
4. Logout: Users can log out to return to the authentication page.

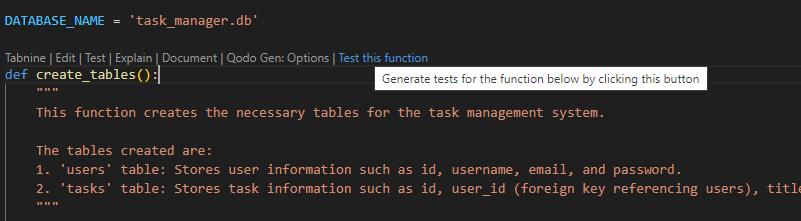
Testing:

Tools Used: Qodo Gen

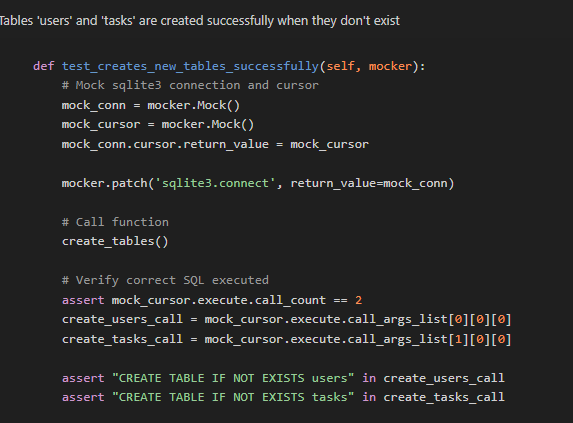
Install the extension in VSCode.

Open Qodo Gen by using ctrl + shift + P and enable it.

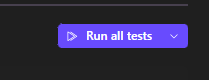
Open your code file. You will see extension above your functions. Click “Test this function”

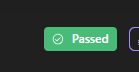


It generates tests for the function.



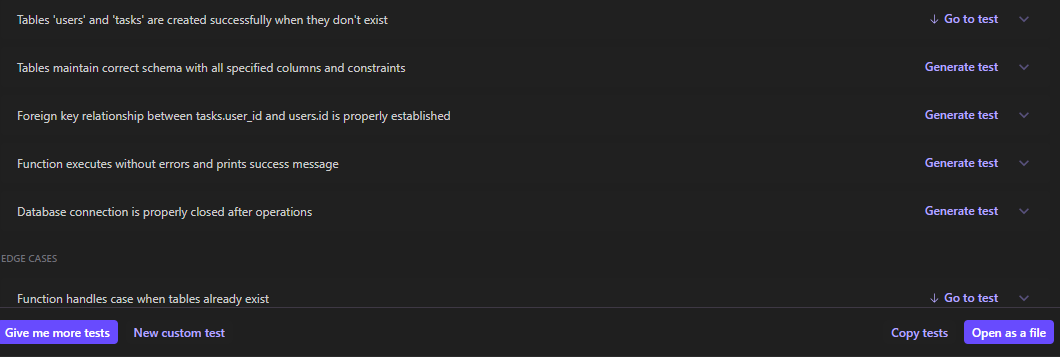
You can either click Run All Tests option at the top right side or copy the code and run it in a separate file by using “pytest test.py”. For the latter part, you need to install pytest by using “pip install pytest”



You will be getting a message as passed 

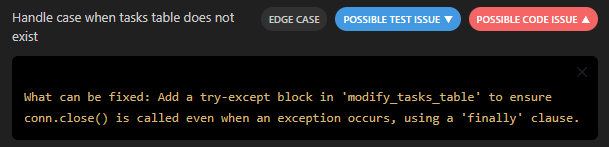
If there is an error, it shows an error message to fix issues in the code.

You can generate more tests, or you can also write custom tests also.

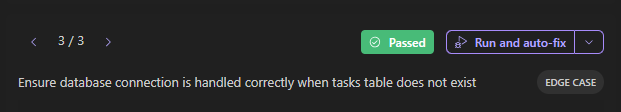


In case of a failed test case, It displays as “FAILED”, It gives POSSIBLE TEST ISSUE, POSSIBLE CODE ISSUE

When you click POSSIBLE CODE ISSUE, it gives suggestions on what to fix in the code.



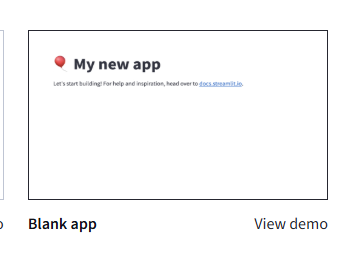
After fixing the code, run the tests again until it shows a PASSED message



Do this for all and fix the tests.

After signing in, You will be seeing a template.

Select Blank App



To do this, at first you need to push your code into your GitHub repository.

Either use VSCode or CMD or GitHub for it

Example:

Using Git in Terminal

Pushing your project to GitHub involves a few steps. Here's a guide to get it done:

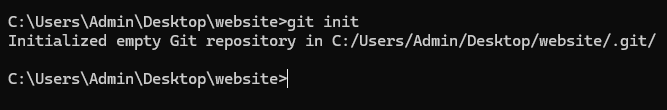
**Step 1: Initialize Git in Your Project Directory**

1. Open your terminal and navigate to your project directory:

cd /path/to/your/project

1. Initialize the Git repository (if you haven't already):

git init



**Step 2: Add and Commit Your Files**

1. Stage all files for the first commit:
2. git add .
3. Commit the changes:
4. git commit -m "Initial commit"

**Step 3: Create a Repository on GitHub**

1. Go to [GitHub](https://github.com/) and log in to your account.
2. Click the "+" icon in the top-right corner and select **"New Repository"**.
3. Fill in the repository details (e.g., name, description) and click **"Create Repository"**.

**Step 4: Link Your Local Repository to GitHub**

1. Copy the URL of the repository you just created (e.g., https://github.com/username/repository.git).
2. Link your local repository to the GitHub repository:
3. git remote add origin https://github.com/username/repository.git

**Step 5: Push Your Code to GitHub**

1. Push your code to the repository (replace main with your branch name if it’s different):
2. git branch -M main
3. git push -u origin main

**Step 6: Verify**

1. Go to your GitHub repository URL.
2. Refresh the page, and you should see your project files.

Deployment:

Tools Used: Streamlit Cloud

Go to the [Streamlit Cloud Community](https://streamlit.io/cloud)

Sign into the cloud by using Google Account or GitHub

Documentation:

Tools Used: Doxygen, TabNine

First install Doxygen from this [website](https://www.doxygen.nl/manual/install.html)/ VSCode Extension

Use TabNine for Document for each function you had used until now.

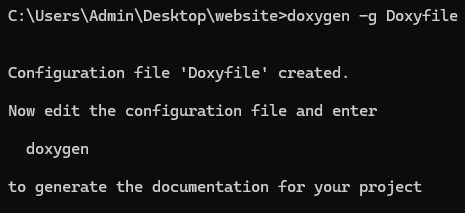
Change the parameters, return type, raise or any features into Doxygen syntax.

**Create a Doxygen Configuration File**

In the root directory of your project (where your Python files are located), create a Doxygen configuration file (Doxyfile). To generate the configuration file, run:

doxygen -g Doxyfile

Output:



Edit the configuration file:

**1. Project Information:**

* **PROJECT\_NAME**: Set the project name to "Task Management App".
* **PROJECT\_BRIEF**: Added a brief description for the project as "A Streamlit app for task management".
* **PROJECT\_LOGO**: Left empty, but you can add a path to your logo if needed.

**2. Input Settings:**

* **INPUT**: Specifies the directory where your Python files are located for documentation (e.g., ./path/to/your/python/files).
* **FILE\_PATTERNS**: Restricts documentation generation to only Python files (\*.py).
* **RECURSIVE**: Set to YES so that Doxygen will search subdirectories for Python files as well.

**3. Output Settings:**

* **OUTPUT\_DIRECTORY**: Set to docs, which is the folder where the generated documentation will be stored.
* **GENERATE\_LATEX**: Set to NO, meaning no LaTeX output will be generated, which is useful if you're not generating PDFs.
* **HTML\_OUTPUT**: Specifies the folder (html) within the docs folder to store the HTML documentation files.
* **HTML\_COLORSTYLE**: Set to AUTO, which automatically determines the color scheme for the HTML output.

**4. Documentation Style:**

* **EXTRACT\_ALL**: Set to YES, ensuring that Doxygen will extract documentation from all functions, methods, and members (even those without explicit comments).
* **EXTRACT\_PRIVATE**: Set to YES to include private members of classes in the generated documentation.
* **EXTRACT\_STATIC**: Set to YES to include static members of classes in the generated documentation.
* **EXTRACT\_LOCAL\_METHODS**: Set to YES to extract and document local methods as well.
* **SHOW\_NAMESPACES**: Set to YES in case namespaces are used in your Python code (useful for organizing your project).
* **TOC\_INCLUDE\_HEADINGS**: Set to YES, so the Table of Contents includes section headings for better navigation.

**5. Class and Function Documentation:**

* **INLINE\_INHERITED\_MEMBERS**: Set to YES to show inherited members of classes within the documentation.
* **BRIEF\_MEMBER\_DESC**: Set to YES to provide brief descriptions for methods and variables within the documentation.
* **FULL\_MEMBER\_DESC**: Set to YES for full descriptions of methods and variables, ensuring complete documentation for class members.

**6. Miscellaneous:**

* **ENABLE\_PREPROCESSING**: Set to YES to preprocess your Python code before documentation generation (this might not be very relevant for Python).
* **SHOW\_USED\_FILES**: Set to YES to display all the files used within the documentation, showing the relationships between them.
* **USE\_MDFILE\_AS\_MAINPAGE**: Left empty for now, but you can specify a markdown file to be used as the main entry point of the documentation if desired.

After the changes, save the file.

Run Doxygen with the command:

doxygen Doxyfile