Semi-automatic development blog system.

We are going to create a Python application, that is capable of installing itself into its own virtual environment using UV.

The goal of the application is to enable developers to quickly create and summarise commits they made in a specific repo/branch on GitHub.

There should be an interactive GUI, where the developer can set up the application to produce the entries, and the resulting dev-blog entry should be saved with a timestamp, with comments in the file indicating exactly which commits were used to create the blog entry. The blog entry should be in English, be concise, worded professionally in the first person, but be interesting whenever possible.

Parts of the GUI and their function:

Github repository:

Login button: The user should log into Github with their user, and the application should inherit their credentials. Note: The application SHOULD ONLY HAVE READ ONLY PRIVILIGES to any repository, except the Dev-blog Repo. There should be a green checkmark if the user is logged in and credentials are available, and a red X to idicate that the login needs to be done.

Select repository button.

When the user completes login, the application should use the credentials of the user to scan, what repositories the user has access privilages on Github, and populate a dropdown list for the user with the names of these repositories for the user to choose from. If the list is more than 10 items long, the user should be able to search among them to find the right one. The user should select one repo to work with. Only one repo at a time.

Browse commits tab:

Once a repo has been selected, the software should scan for, and list the last, yet unprocessed commits in a scrollable (searchable) list. There should be an checkbox included at the front of the commit. If it is checked, it should be included in the dev-blog generataion, and the software should remember this state for future use – so the user can easily filter for items that have already been used for blog entry generation. There should also be filters for a date range and displaying only the last X number of commits.

Actually, there should be two checkboxes for each entry: process commit message, and process code comments. These should indicate if the application should use either/or or both of these to create its blog entries.

In a tab to the right of this scrollable list, there should be a preview pane (scrollable, maximizable, searchable for specific words) that shows the contents of the selected commit.

AI section:

Select model button/settings button.

The user should be able to select their favorite AI model/API to process the commits and create a dev-blog entry based on them. Make the following choices available, and generate the appropriate code / settings to use these APIs in the code.

* Google Gemini (enter API code, test to see it works)
* ChatGPT (enter API code, test to see it works)
* Local Ollama

There should be an indication of what model is active, with a red X if there is a problem with the setting (not working or not defined), and a green checkmark to indicate that the setting is working.

Generate Blog Entry button, with a prompt field.

There should be an editable text section showing the current prompt that will be sent to the LLM to process the GitHub entries selected previously. (If not GitHub entries are selected, this section should be “greyed out” and be inactive.)

The default prompt should be the following:

“Write a concise informative but interesting development blog entry to each commit message (if you think it warrants a blog entry) if you think it is interesting enough to post about. Keep the wording professional, in the first person. Sign with the comitter's name.” If the user edits this, a save / reset to default button should become active.

Once the generation is complete, it should be saved as an .md file to a subdirectory called “Generated\_entries”. The filename should be a concatenation of the repo name, and(the current time and date). This file should be displayed in an editor window, where the following functions should be available:

* The user should be able to search, find-replace and edit within the file, freely.
* The user should be able to save the file, if they change anything.
* The user should be able to accept/reject the generated text.
* The user should be able to “regenerate” the text, causing the LLM generation to run again and the newly generated file should be displayed. For the regeneration, the user should be able to choose a different LLM that they already have configured within the application.
* The user should be able to return to the previous configuration screen upon accepting or rejecting the blog entry.