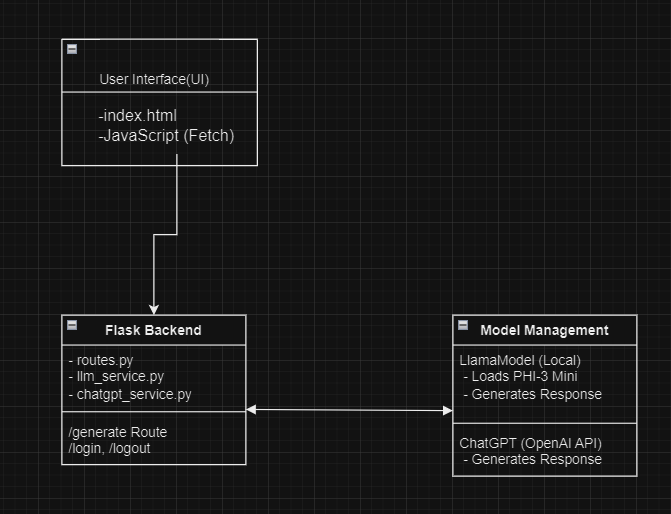
**1. Conception of the Software Solution**

The chatbot was designed using the Flask framework as a backend, a static front-end HTML page, and a local AI model (PHI-3 Mini) for text generation. Additionally, it supports dynamic switching between a local model and the OpenAI API (ChatGPT) based on user selection.

**1.1 Architecture Diagram**

The following diagram outlines the architecture of the chatbot application:



**Description**:

* **User Interface (UI)**: A web page (index.html) provides an interface for users to input prompts, choose between models, and receive responses. JavaScript (Fetch API) is used to send and receive data from the Flask backend.
* **Flask Backend**: Manages HTTP requests and responses, handling routing (/generate, /login, /logout). It contains the routes.py file to handle routes and the llm\_service.py to manage content generation.
* **Model Management**:
  + **Local Model**: Managed through LlamaModel, which uses the llama\_cpp library to load and interact with the local PHI-3 Mini model.
  + **ChatGPT API**: Uses the chatgpt\_service.py to send prompts to the OpenAI API if the user chooses this model.
* **Communication**: The backend handles requests via HTTP. The /generate route processes user prompts and returns generated content based on the selected model.

**1.2 Software Components**

1. **Frontend**:
   * index.html: Provides the user interface with input fields and options for model selection.
   * JavaScript: Handles form submission and communicates with the Flask backend using Fetch API.
2. **Backend (Flask)**:
   * **routes.py**: Defines routes (/generate, /login, /logout) and manages user sessions.
   * **llm\_service.py**: Manages content generation using the local PHI-3 Mini model.
   * **chatgpt\_service.py**: Sends prompts to the ChatGPT API when selected.
3. **Model Management**:
   * **LlamaModel Singleton**: Loads the local model once to optimize performance.
   * **Dynamic Model Selection**: Allows users to choose between using a local AI model or the ChatGPT API.

**2. Usage of a Tool to Organize Development**

For this project, GitHub Projects was used to manage tasks and organize the workflow. The project board was divided into columns representing different stages of development: "To Do," "In Progress," "Testing," and "Done."

**Sample GitHub Project Board:**

| **Column** | **Tasks** |
| --- | --- |
| **To Do** | - Set up Flask project - Implement login/logout functionality - Create form for prompt input |
| **In Progress** | - Integrate PHI-3 Mini with llama\_cpp - Develop llm\_service.py |
| **Testing** | - Test response generation - Test model switching between local and ChatGPT |
| **Done** | - Set up basic HTML interface - Add CSS for styling - Implement generate route |

**3. Feedback on Development Self-Organization (1 pt)**

Throughout the development of this chatbot, I learned the importance of careful planning and organizing tasks into smaller, manageable pieces. Using GitHub Projects to track tasks helped maintain focus on each part of the project. The division of labor into different modules (llm\_service, chatgpt\_service, and routes) improved the code's modularity and made testing more straightforward.

**Challenges**:

* The integration of the local AI model (PHI-3 Mini) required fine-tuning of parameters to achieve optimal results, which was time-consuming.
* Coordinating between different components (frontend, backend, and AI models) needed a structured approach to avoid conflicts and ensure smooth communication.

**Successes**:

* Using a Singleton pattern to handle model loading (LlamaModel) significantly improved performance.
* The decision to provide users with an option to select between local and API-based models enhanced the chatbot’s flexibility.

**Summary of Development Flow**

1. **Initial Setup**: Set up the Flask project, created basic routes, and designed the frontend in index.html.
2. **Backend Development**: Built the backend in Flask to handle user sessions and content generation.
3. **Model Integration**: Integrated the local AI model using llama\_cpp and implemented content generation logic.
4. **Testing and Tuning**: Tested the model outputs, adjusted parameters (e.g., max\_tokens, temperature), and optimized for a better user experience.
5. **Frontend Enhancement**: Added CSS styles, implemented JavaScript for interaction, and added functionality to select between different models.
6. **Final Testing and Review**: Conducted final tests to ensure seamless interaction between the frontend, backend, and models.

**Conclusion**

By breaking down the project into smaller tasks, organizing development using GitHub Projects, and providing feedback on the development process, this report illustrates the structure and approach taken in building the AI-powered writing assistant, "Inkspritation."

If you need diagrams, I can generate those based on the breakdown provided here. Let me know!