Part 1 - Planning:

Define the problem you are solving:

The a12n-server's admin UI currently requires manual troubleshooting and navigation through documentation for common configuration tasks, debugging, and log analysis. Admins sometimes struggle with understanding and resolving OAuth misconfigurations, searching through logs to diagnose authentication failures, and getting real-time insights into security incidents. The process of manually configuring clients and policies is inefficient and time-consuming. Furthermore, navigating complex authentication flows and debugging token-related issues can be challenging, especially for new administrators.

These challenges slow down system administration, increase response times, and reduce overall security effectiveness. As the system scales, the need for a more efficient, intelligent, and proactive support mechanism becomes critical. An AI-powered chatbot can address these issues by providing immediate assistance, reducing reliance on documentation, and offering real-time security insights, ultimately enhancing the efficiency and security of the admin experience.

Explain how the AI component enhances your project (e.g., automation, decision-making, prediction, personalization, etc.).

Firstly, the AI component should allow the administrator to request specific data (i.e. "Show me all users that received errors on Monday") by simply using words. The burden of actually producing a query (or manually inspecting the data) falls onto AI. This increases the accessibility of Admin UI, as the user does not need to be aware of the intricacies of how the data is stored, and is instead simply requiring them to type their request. It also increases the UI's versatility and personalization, as the user can now retrieve the data in a way the Admin UI does not necessarily support, which makes the system more adaptable to edge cases (like observing specific auth errors).

Secondly, the AI component should help the Admin troubleshoot various OAuth2 issues by examining the current configuration of the system. The admin should be able to request either general support or support for a specific issue, while the AI component should provide an analysis of the current configuration and introduce the necessary changes if the admin agrees. This enhances our project by automating a fairly menial task of troubleshooting configurations, which should in turn make the server more accessible.

Selection Criteria:
There are a few key points of criteria to consider when choosing the most suitable AI model/framework. Firstly, the AI solution should have maximum feasibility. This ensures that the given solution can integrate easily with our existing tech stack while having readily available API's with relatively low amount of overhead to setup. Secondly, it must also have high performance metrics. This is crucial with the application requiring a strong need for fast response times and high accuracy in understanding queries.

Security is also a highly important criteria when selecting the AI model, as the AI chatbot would need to comply with authentication and admin access policies, preventing unauthorized access to sensitive data. The model must also be customizable, allowing it to be adaptable regarding a12n-server configurations and admin-specific tasks. Lastly, cost efficiency must also be taken into consideration as the model must maintain operational costs at a sustainable level while maximizing benefits and ensuring a sustainable implementation.

Potential AI Models and Frameworks:

OpenAI ChatGPT: if hosting an LLM is daunting, ChatGPT API would be a good candidate for the model. Potential drawbacks include higher latencies and having to pay a fee, nonetheless, it is a very reasonable option.

DeepSeek R1: if hosting an LLM is feasible, then DeepSeek R1 is an adequate open-source option that would not require a fee. It might not be exactly up to commercial standards, but it being open-source would allow for me flexibility. One of the major drawbacks is that it is quite resource-intensive, so a separate machine for it would be most certainly required.

LangChain: LangChain is a framework specifically designed for integrating LLMs into software projects. It specifically allows one to chain together LLMs, APIs and databases, making it ideal for this AI component.

SQLDatabaseChain: SQLDatabaseChain, included in LangChain, was made to convert natural language queries into SQL queries dynamically. This makes it ideal for the natural language queries included in the AI component. One notable drawback is that this template is still in LangChain's experimental branch.

Datasets/Training required:
In terms of dataset training for the AI powered chatbot, not a considerable amount of effort is required. Initial prompt engineering and tailoring to a general model should provide sufficient domain knowledge of the application. Should further adjustments be required, the model can be

finetuned using datasets containing information on OAuth2 concepts, a12n-server, common admin tasks and FAQ's alongside system logs and error handling scenarios.

Ethical Considerations:
The third party AI model must not be trained on any user specific data in order to ensure privacy regulations are upheld and prevent passage of user data to third party applications. The datasets that are utilized to train the model must be kept constrained to general OAuth and a12n-server information alongside sample error handling scenarios.

Justification of Feature:
The main purpose of the a12n-server is to provide an authentication server which follows OAuth2 standards. It also will also be accompanied with a user-friendly Admin-UI developed by our project group which makes it easier to manage users, roles and privileges. The underlying goal of the entire project is to reduce the burden on developers by not having to develop their own authentication server and making it easier to manage users for their application. Having the AI powered chatbot complement the Admin-UI would ease pain points for the developer, hence lending to the overall project mission and goal.

Part 2 - Impact Analysis:

Product improvement before and after AI integration:

Before the AI integration, we have an Admin UI that is perfectly useful but quite rigid in the way it presents data. An experienced user can most certainly send queries to the server directly, but, firstly, we cannot expect our users to be experienced and, secondly, getting such experience would require the user to directly study the implementation of the server. This would be a hassle for a non-developer.

After integration, we have an Admin UI that allows the user to dynamically observe data without necessarily being familiar with the intricacies of the server. A user can more easily analyze issues with authorization and configuration without having to interact with the server itself. The developers are no longer burdened with having to create additional tools to resolve these specific issues.

Testing and Evaluation:

To measure the success of this integration, several testing and evaluation methods will be employed. Unit tests will be conducted to verify the chatbot's response accuracy for predefined admin queries. A/B testing will compare admin efficiency with and without chatbot support, measuring improvements in response time and accuracy. Admin feedback surveys will be

conducted to assess usability and effectiveness, ensuring that the chatbot meets the needs of administrators. Performance metrics, such as chatbot response time, accuracy, and admin interaction rates, will be monitored to ensure optimal operation.

Tradeoffs and Considerations:

There are several trade-offs to consider in this integration. API-based AI models introduce minor latency, with response times averaging around 500ms, though this is acceptable given the benefits of automation. Security is a crucial concern, requiring strict access control and monitoring to prevent unauthorized actions or confidential data leaks. The cost of API calls to OpenAI's GPT need to be optimized to minimize unnecessary queries while ensuring continued functionality. Another consideration is the long-term sustainability of relying on an external AI API versus developing a self-hosted alternative. Additionally, chatbot responses must maintain high accuracy ensuring that admins receive meaningful and relevant responses.

Using our own LLM integration would certainly allow for greater flexibility, however, available solutions would not only cut the development time but also ensure that this integration will be kept up to date with minimal effort from the developer. Moreover, tools like LangChain were likely made with domain-specific issues in mind, so the integration will likely be more stable and accurate when it comes to natural language. It would also make integrating several LLMs easier, if domain-specific LLMs are introduced down the road.