**Assess existing legacy software, and review the source code if available, appreciating that part or all of the system may be implemented using an obsolete programming language.**

I reviewed the source code to understand the implementation of the existing movie recommendation system. Checking all of dependencies which we used in the project I noticed that server import werkzeug, which may be outdated. This import could break under the latest Flask without adjustment. Our Flask server bundles authentication, API endpoints, and database access in one file - app.py. (Kloda, 2025a) It could be refactored into a package-based architecture with separate modules for authentication, API routes and config management.

We chose to use LLaMa 3.2 version, which is relatively new release from Meta. It allow interpreting user queries for movie recommendations. However, LLaMA 3.2 was released under a community license, which, while permissive, includes certain usage restrictions. (Olteanu, 2024)

I also assessed the API call implementations for efficiency Async. HTTP calls are handled properly using httpx.AsyncClient, which is great for non-blocking input output. Timeout is set globally, and the error catching implemented with try/except blocks to prevent outright crashes.

I also examined the current state of the database schema. (Kloda, 2025b) It became evident that the schema has not been updated to reflect recent data requirements. This outdated code could limit the data integrity, and compatibility with system.

Just Pick use OAuth2 for user authentication, implemented via the oauthlib.oauth2.WebApplicationClient. This allows the application to delegate authentication to trusted identity providers, which means we do not need to store or manage user passwords. However, when assessing legacy software it's critical to ensure that libraries like oauthlib, as well as other core dependencies, are kept up to date. In addition, proper error handling is essential when integrating with external services such as authentication providers or search engines.

As part of the evaluation, I also analyzed the project is using a Bitbucket pipeline. (Kloda, 2025c) This automation reflects good DevOps practices, however, upon reflection, in a future iteration I would choose to implement Docker to containerize the application and its services. This would further align the project with modern standards by improving deployment consistency.

# References

Kloda (2025). [online] github. Available at: https://github.com/Jkloda/movie\_recommendation\_system/blob/main/server/app.py [Accessed 10 May 2025].

Olteanu, A. (2024). *Llama 3.2 Guide: How It Works, Use Cases & More*. [online] Datacamp.com. Available at: <https://www.datacamp.com/blog/llama-3-2> [Accessed 15 Apr 2025].

Kloda (2025). [online] github. Available at: https://github.com/Jkloda/movie\_recommendation\_system/blob/main/data/mysql.sql [Accessed 15 Apr 2025].

Kloda (2025). [online] github. Available at: https://github.com/Jkloda/movie\_recommendation\_system/blob/main/bitbucket-pipelines.yml [Accessed 10 May 2025].