Teamwork Reflection

Yadong (Hugo) Hu

My work: Customized AI Model & Use of API & Chat Frontend

1. Reflection on My Work

1.1. Positive Attributes:

- Initiative in Code Implementation: I took the initiative to implement the initialization of the AI chat, ensuring a seamless connection between the user and the AI model. The *initialize_ai_chat()* function demonstrates my ability to set up the chat environment and handle the initial interaction with the AI model. At first I was thinking about using the synchronized way, but then I realized I cannot block the whole user interface while waiting for new message coming, so I changed it into asynchronized way.
- Efficient Message Handling: To use the Character.ai model, I have to find the API, but it does not have the official one, so I found a community version of the API. The *send_message()* function showcases my efficiency in handling user input. I successfully implemented the functionality to send and receive messages between the user and the AI model. The use of asynchronous programming with the *PyAsyncCAI* lib highlights my commitment to efficient and responsive code.
- User Interface Design: I contributed to the design and functionality of the user interface. The HTML templates, including the use of Jinja templating, demonstrate my skills in creating an interactive and visually appealing chat interface. The integration of JavaScript for dynamic message loading and display adds to the overall user experience.

1.2. Areas for Improvement:

- Code Documentation: While the code functionality is evident, there is room for improvement in terms of code documentation since I thought only me would use this part of code. But adding comments and docstrings could enhance the readability of the code, making it easier for future maintenance.
- Error Handling: The code lacks comprehensive error handling in some sections. Incorporating robust error-handling mechanisms, especially in asynchronous operations (there's no time-out mechanism, and the web keeps waiting if no new message coming), would contribute to the overall reliability and stability.
- Consistency in Code Style: Although the code is generally well-structured, there are instances where the coding style could be more consistent. Ensuring uniformity in variable naming, indentation, and other style conventions would improve code maintainability.

2. Evaluation on Team's Work

	Frontend UI **(Eric Yin)**	**CDK & Load Test** **(Yilin Yang)**	**AWS Resources** **(Zhihan Xu)**
P1	Adapted a well-designed UI framework to make the login and register page very beautiful	Very well-integrated with CDK toolkit to use code deploying all resources	One Lambda with one feature to avoid unnecessary cost for invoking it
P2	Implemented a dark / light mode toggle feature	Did quantitative analysis on latency	All lambda handles error cases correctly
P3	Had error messages to tell user the exact error, and has different response error page (404, 500, etc.)	Figured out how to setup permissions for each resource and the AWS credential	Exported all names to environment, so other resource could use it without knowing the hard-coded names
I1	The login UI style is not strictly aligned with our chat view's UI, and there is no dark mode for the chat interface	The Docker file lacks error prompts or logs, the user may be confused when running the docker image	These three Lambda functions have similar features, and are better to put in just one Lambda function
I2	The lambda function's name is hard-coded	The load test was done in a quite small scale	The password field in DynamoDB is not hashed before store
I3	JavaScript files are placed in html file. It's better to separate them	There's no continuous deployment of the CDK package	The use of "Scan" method to do a "Query" job is not efficient in DDB

Note: P1 - Positive point 1, I1 - Improvement point 1.

3. Team Discussion Outcome

Overall, we made a great job. We implemented the frontend and the microservice. The frontend is well designed, and the APIs are integrated seamlessly. The microservice is managed by AWS CDK, and three Lambda functions perform individual tasks. This whole system works well on a single-session case, however, when facing a high demand of visit, our latency is high because we do not have a load balancer, and the host CPU is powerless; and there might even be some bugs without properly handled. Our frontend does not correctly manage different users' sessions and local storage (messages). And in our login lambda, the scan method in DynamoDB will result in a very high response time when we have larger user data.