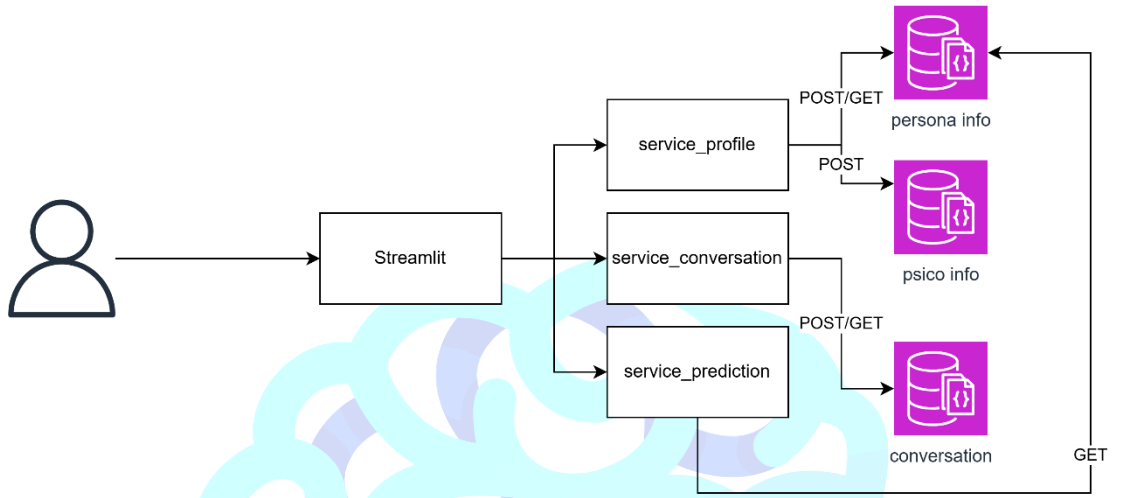
Proposed Serving Architecture

MVP

To demonstrate value and functionality, we implemented the following architecture:

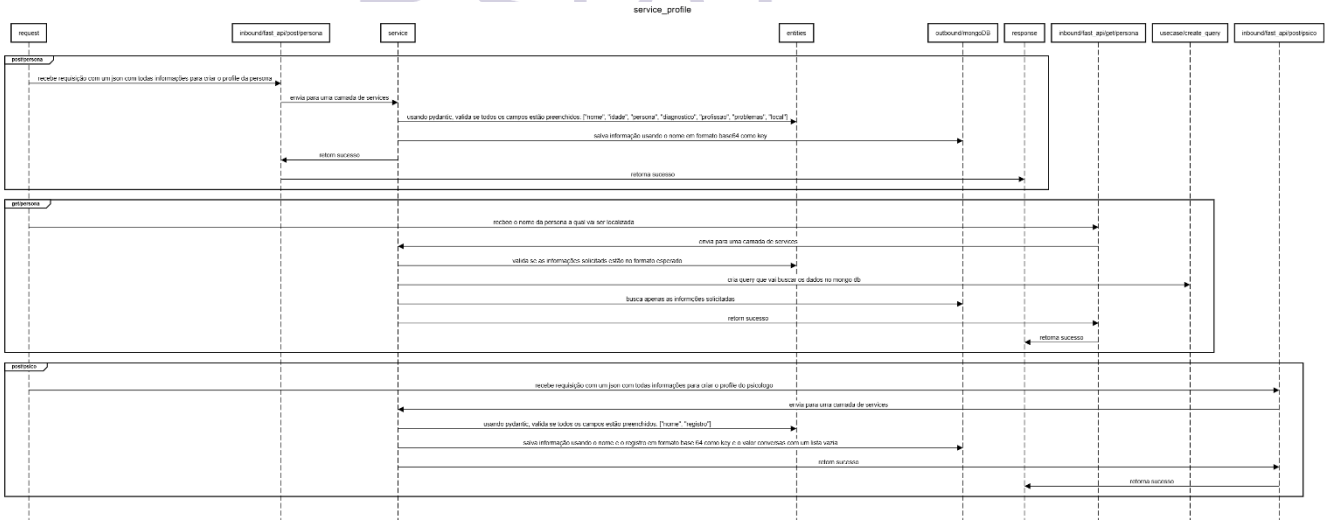


The Streamlit platform serves as the user's visualization front-end. Following the single-responsibility principle, it is only responsible for displaying and receiving information, without managing or storing data. Currently, there is no validation to check if a user is registered in the system or what type of access they have.

The next layer comprises three microservices, also aligned with the principle of minimal responsibility. The first is the service profile, available at this repository: https://github.com/mpatusco/llama_service_profile_psiAI.

Its responsibility includes managing and controlling user data (both created profiles and individuals evaluating the profiles).

The following sequence diagram shows how it operates and the available routes:



Examples of route usage:

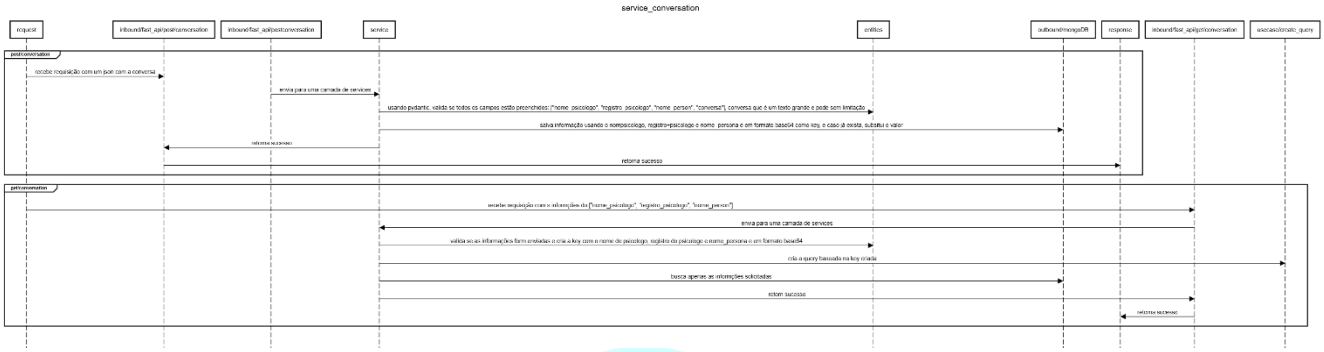
curl -X GET "http://localhost:8000/api/v1/persona/?nome=M"

curl -X POST http://localhost:8000/api/v1/persona/ -H "Content-Type: application/json" -d "{\"nome\":\"M\",\"idade\":30,\"persona\":\"Paciente\",\"diagnostico\":\"Ansiedade\",\"profissao\":\"Engenheiro\",\"problemas\":\"Estresse\",\"local\":\"São Paulo\"}"

Further text description (visit the site https://sequencediagram.org/ and paste it to generate the same image)

```
title service_profile

fast_api que executa ações sobre profile de perfis e dos psicologos

group post/persona
request -> inbound/fast_api/post/persona: recebe requisição com um json com todas informações para criar o profile da persona
inbound/fast_api/post/persona -> service: envia para uma camada de services
service -> entities: usando pydantic, valida se todos os campos estão preenchidos: ["nome", "idade", "persona", "diagnostico", "profissao", "problemas", "local"]
service -> outbound/mongoDB: salva informação usando o nome em formato base64 como key
service -> inbound/fast_api/post/persona: retorn sucesso
inbound/fast_api/post/persona -> response: retorna sucesso
end

group get/persona
request -> inbound/fast_api/get/persona: recbee o nome da persona a qual vai ser localizada
inbound/fast_api/get/persona -> service: envia para uma camada de services
service -> entities: valida se as informações solicitads estão no formato esperado
service -> usecase/create_query: cria query que vai buscar os dados no mongo db
service -> outbound/mongoDB: busca apenas as informções solicitadas
service -> inbound/fast_api/get/persona: retorn sucesso
inbound/fast_api/get/persona -> response: retorna sucesso
end

group post/psico
request -> inbound/fast_api/post/psico: recebe requisição com um json com todas informações para criar o profile do psicologo
inbound/fast_api/post/psico -> service: envia para uma camada de services
service -> entities: usando pydantic, valida se todos os campos estão preenchidos: ["nome", "registro"]
service -> outbound/mongoDB: salva informação usando o nome e o registro em formato base 64 como key e o valor conversas com um lista vazia
service -> inbound/fast_api/post/psico: retorn sucesso
inbound/fast_api/post/psico -> response: retorna sucesso
end
```

Another microservice, **service_conversation**, manages everything related to conversations. It's available at: https://github.com/mpatusco/llama_service_conversation_psiAI.



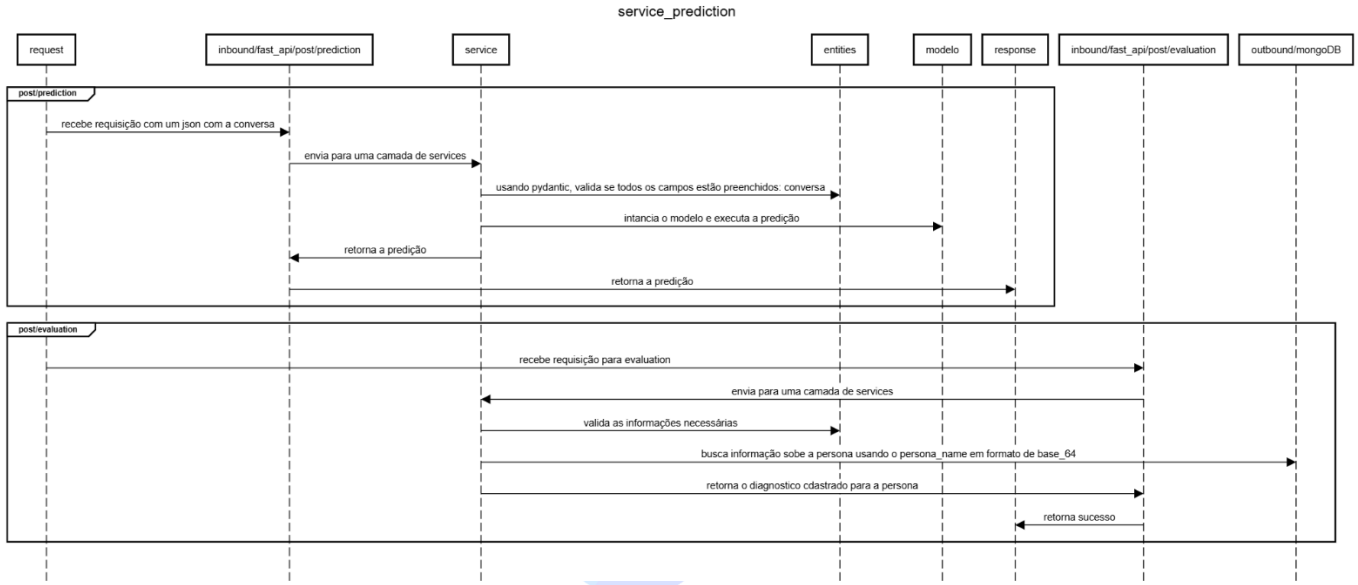

```
title service_conversation

fast_api que execut ções necessárias sobre o processo de conversaçõ

group post/conversation
request -> inbound/fast_api/post/conversation: recebe requisição com um json com a conversa
inbound/fast_api/postconversation -> service: envia para uma camada de services
service -> entities: usando pydantic, valida se todos os campos estão preenchidos: ["nome_psicologo", "registro_psicologo", "nome_person", "conversa"], conversa que é um texto grande e pode sem limitação
service -> outbound/mongoDB: salva informação usando o nompsicologo, registro+psicologo e nome_persona e em formato base64 como key, e caso já exista, subsitui o valor
service -> inbound/fast_api/post/conversation: retorna sucesso
inbound/fast_api/post/conversation -> response: retorna sucesso
end

group get/conversation
request -> inbound/fast_api/get/conversation: recebe requisição com s informções do ["nome_psicologo", "registro_psicologo", "nome_person"]
inbound/fast_api/get/conversation -> service: envia para uma camada de services
service -> entities: valida se as informações form enviadas e cria a key com o nome do psicologo, registro do psicologo e nome_persona e em formato base64
service -> usecase/create_query: cria a query baseada na key criada
service -> outbound/mongoDB: busca apenas as informções solicitadas
service -> inbound/fast_api/get/conversation: retorn sucesso
inbound/fast_api/get/conversation -> response: retorna sucesso
end
```

Request exmples:

curl -X POST http://localhost:8001/api/v1/conversation/ -H "Content-Type: application/json" -d "{\"nome_psicologo\":\"Silva\",\"registro_psicologo\":\"12345\",\"nome_person\":\"J\",\"conversa\":\"Sessão inicial sobre ansiedade e estresse no trabalho.\"}"

curl -X GET "http://localhost:8001/api/v1/conversation/?nome_psicologo=Silva&registro_psicologo=12345&nome_person=J"

Lastly, the **prediction service** is responsible for predictions and their evaluations. It's accessible at https://github.com/mpatusco/llama_service_prediction_psiAI.




```
title service_prediction

fast_api que executa predições e a avaliação do que o psicologo identificou como transtorno na paciente

group post/prediction
request -> inbound/fast_api/post/prediction: recebe requisição com um json com a conversa
inbound/fast_api/post/prediction -> service: envia para uma camada de services
service -> entities: usando pydantic, valida se todos os campos estão preenchidos: conversa
service -> modelo: intancia o modelo e executa a predição
service -> inbound/fast_api/post/prediction: retorna a predição
inbound/fast_api/post/prediction -> response: retorna a predição
end

group post/evaluation
request -> inbound/fast_api/post/evaluation: recebe requisição para evaluation
inbound/fast_api/post/evaluation -> service: envia para uma camada de services
service -> entities: valida as informações necessárias
```

service -> outbound/mongoDB: busca informação sobe a persona usando o persona_name em formato de base_64
service -> inbound/fast_api/post/evaluation: retorna o diagnostico cdastrado para a persona
inbound/fast_api/post/evaluation -> response: retorna sucesso
end

Exemplos de requisições:

curl -X POST http://localhost:8002/api/v1/prediction/ -H "Content-Type: application/json" -d "{\"conversa\":\"Paciente relatou sintomas de ansiedade e dificuldade para dormir.\"}"

curl -X POST http://localhost:8002/api/v1/evaluation/ -H "Content-Type: application/json" -d "{\"diagnostico\":\"Ansiedade Generalizada\",\"persona_name\":\"João\"}"

To run local we need to have mongoDB installed and start all 4 applications locally, each on a different port.

Pay attention to the backend microservices that need to be on the same ports described in the streamlit file, otherwise errors of not finding the data will occur.

With all the services available, just use streamlit and carry out the tests.

Architecture motivation

As we think about use by companies and education centers, this type of service needs to be made available in such a way that it works on any cloud or local server. In addition, we already use good programming and architectural practices, based on SOLID concepts, clean architecture and AWS well architected.

Although the first version does not yet include unit tests and interfaces, it is possible to recognize patterns such as single responsibility within functions and modules, separation into layers of inbound, outbound and services. In addition to being ready to be deployed in serveless services that optimize costs, and ensures that only one service is responsible for adding, updating and deleting resources from a database, increasing the confidence of the architecture.

Next steps

In just 24 hours, local executions were possible, but did not meet all the necessary quality requirements.

We need to run services via container, thus ensuring replicability in each environment.

It is also necessary to develop unit tests and integrated tests to increase code quality and ensure reliability.

Insert error handling and logs at important points for error handling and better understanding.

Next steps architecture

To truly allow it to be replicated across different infrastructures, we need to implement the interfaces, ensuring that in any environment in which it is implemented, code is added to make external connections without having to change any other part of the code, thus ensuring not only replicability but also maintainability.