DEPLOYMENT METHOD

<u>FASTAPI and Streamlit for Web Application Development</u>

FASTAPI: Backend API
FASTAPI is a modern, high-performance web framework for building RESTful APIs using Python. It is built on top of Starlette for web components and Pydantic for data validation. FASTAPI simplifies the creation of RESTful services by automatically generating interactive API documentation (Swagger UI or ReDoc). It leverages Python's type hints for input validation and supports asynchronous programming, allowing it to handle many concurrent requests efficiently, making it a popular choice for scalable, production-grade APIs.

In this web application, FASTAPI acts as the backend service that exposes API endpoints for tasks such as data processing, model inference, and CRUD operations. It serves as the logic layer that performs heavy computations, interacts with the saved model files, and returns JSON responses that the frontend can consume.

Trade-offs and Alternative Backend Options
While FASTAPI was chosen for its performance and ease of use, other backend frameworks were considered.

Framework	Pros	Cons
Django REST Framework (DRF)	Full-fledged web framework, built-in ORM	Heavier than needed for lightweight API services
FLASK	Simple, flexible, easy to learn	Not async by default, requires extra setup for concurrency

FASTAPI was selected because it is asynchronous, Python-native, and provides built-in validation via Pydantic. FLASK was considered but lacks native async support, making FASTAPI a better option for handling concurrent requests efficiently.

Streamlit: Frontend Interface
Streamlit is an open-source Python library for building interactive, data-driven web applications without needing to learn HTML, CCS, or JavaScript. It generates user interfaces dynamically from Python code, allowing the creation of custom dashboards, data visualizations, and machine learning apps with just a few lines of code. In this web application, Streamlit serves as the interactive frontend, where users input data, visualize results, and trigger API requests to the FASTAPI backend. It communicates with the backend via HTTP requests (using Python's requests library) and updates the UI based on the responses received.

Trade-offs and Alternative Backend Options
Although Streamlit was chosen for its simplicity, alternative frontend solutions were considered.

Framework	Pros	Cons
Dash (by Plotly)	Better for interactive dashboards, Python-friendly	Requires more manual component handling
React (with Flask/FASTAPI backend)	Highly customizable, ideal for production-grade apps	Requires JavaScript knowledge, more development time
Shiny (R/Python)	Strong for data visualization	Not optimized for ML-specific apps

Streamlit was chosen because it allows for rapid development of interactive dashboards and integrates seamlessly with Python scripts.While Dash was considered, it requires more effort to handle UI elements manually. A React-based frontend was also an option, however, since this application is primarily data-driven, Streamlit's simplicity made it the best fit.

Planned architecture

FASTAPI will run as a separate service (or microservice) dedicated to processing requests and executing backend logic. Meanwhile, Streamlit will serve as the client-side application with which users interact. Communication between the two will occur via HTTP. Specifically, the Streamlit app will make HTTP requests (using POST or GET) to the FASTAPI endpoints. For example, when a user clicks the "Predict" button in the Streamlit app, it will send a JSON payload with the user inputs to https://my-backend-url/predict. FASTAPI will then process this request, perform the necessary prediction, and return a JSON response. Finally, Streamlit will receive this response and update the UI accordingly.

Data flow

- User Input: Users will fill in input fields in the Streamlit app.
- Request Submission: The Streamlit app will use the requests library to send a POST request with the input data.
- Backend Processing: FASTAPI will receive the data, validate it using Pydantic models, run the prediction using the pre-trained ML model, and return the result as JSON.
- UI Update: The Streamlit app will parse the JSON response and display the prediction.

Deployment strategy

FASTAPI will be deployed on Render as a backend service, running as a standalone API. Streamlit will be deployed on Streamlit Community Cloud, offering a publicly accessible UI for interacting with the model. Communication between the two applications will be handled using public URLs, allowing seamless integration.

Trade-offs and alternative deployment options

Deployment option	Pros	Cons
AWS Lambda (Serverless API)	Scales on demand	Cold start latency, complexity in managing dependencies
Docker on AWS/GCP	Full control, highly scalable	Requires DevOps expertise

Render for FASTAPI was chosen because it provides an easy-to-deploy managed service, avoiding the need for manual server management. AWS Lambda was considered, however, not chosen because it introduces cold start issues for infrequent requests. Streamlit Community Cloud was chosen because it's free and simple to deploy, making it ideal for an interactive ML prototype.

Error handling and responsiveness

I will leverage FASTAPI's robust input validation and error handling to ensure that the backend returns meaningful error messages if something goes wrong. The Streamlit app will display these errors in the UI and provide clear feedback to users.

Benefits of combining FASTAPI and Streamlit

- Separation of Concerns: Backend logic and data processing will be isolated from the user interface, which will simplify development and maintenance.
- Scalability and Performance: FASTAPI's asynchronous capabilities will help manage high loads, while Streamlit will offer a responsive UI for predictions.
- Rapid Iteration: Building and iterating on both the backend and frontend will be streamlined since the entire stack is in Python.
- Enhanced User Experience: Users will enjoy an interactive, real-time experience in the browser, while heavy computations are handled by FASTAPI in the background.

Future scalability considerations

If higher concurrency is required, FASTAPI will be deployed on Unicorn and Gunicorn behind a load balancer. Streamlit will be replaced with a React frontend for a more optimized UI experience.

Conclusion

By using FASTAPI and Streamlit together, I will create a web application where FASTAPI manages the backend API logic and Streamlit provides a dynamic, interactive frontend. They will communicate via HTTP, allowing each to leverage its strengths—FASTAPI for high-performance backend services and Streamlit for engaging, data-driven user interfaces. This architecture is ideal for deploying my machine learning model, as it enables users to interact with the model and visualize prediction outputs in real time. Trade-offs were carefully considered to ensure optimal performance, ease of development, and future scalability.