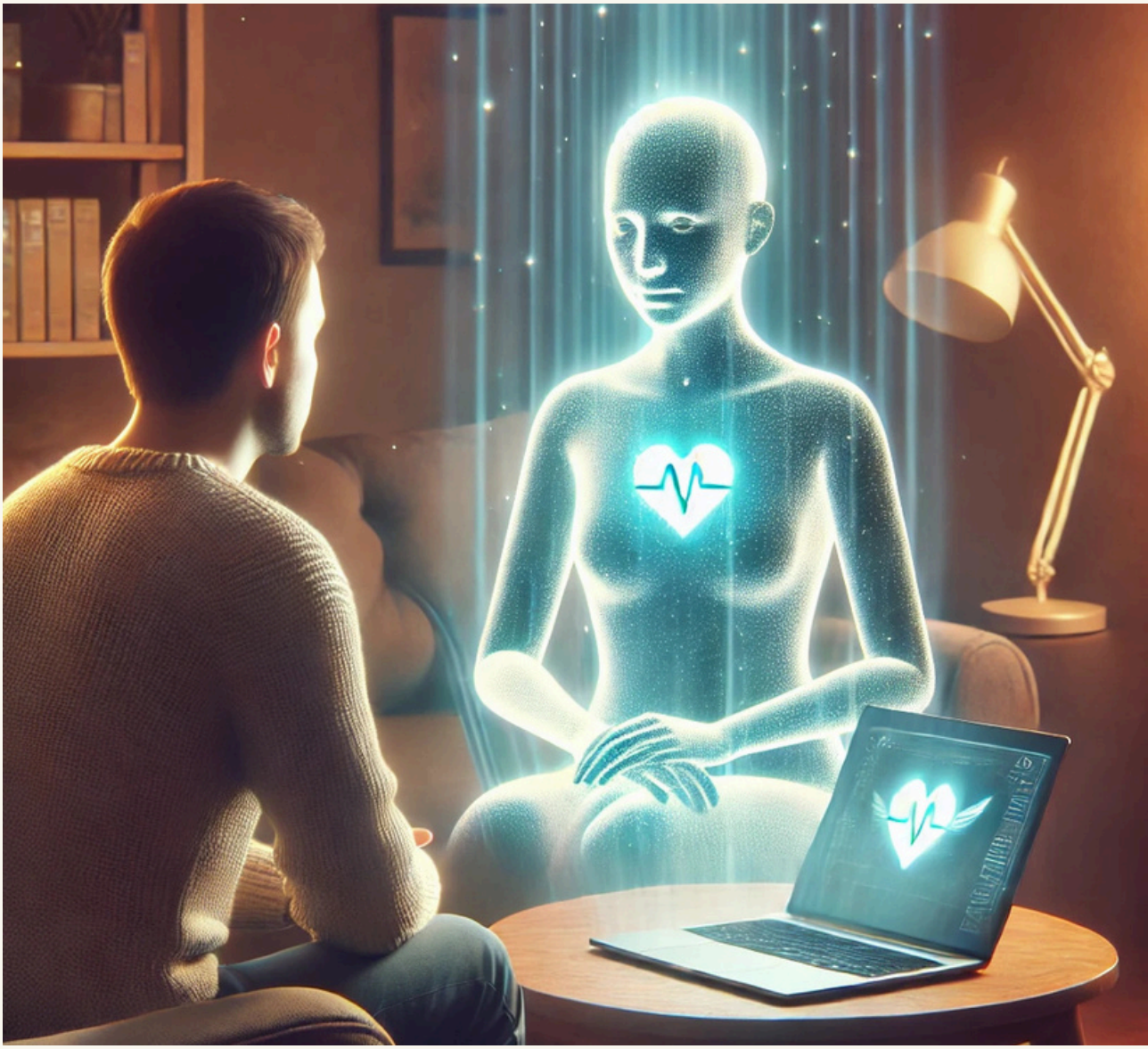
INTERNATIONAL UNIVERSITY OF RABAT

Project Report

AI-Powered Mental Health Advisor



Team Members: Ranya Laamari | Chiguer Mohamed | Nada Aznak| Safouane Bouskif
Course Name: Artificial intelligence | Nosql
Date of Submission: 15/01/2025

Table of Contents

I. Abstract

This project presents an AI-powered mental health advisor that combines advanced natural language processing, real-time transcription, and an engaging avatar interface. Using tools like **FastAPI, Whisper, and the D-ID API,** the system provides empathetic and informative responses to user queries. The avatar enhances the experience by delivering responses through expressive video interactions. This report outlines the app's development, technical implementation, and challenges addressed, showcasing its potential as a novel solution for mental health support.

Technologies Table:

Component	Technology Used	Purpose
Backend	FastAPI	Server-side logic, API development
AI Models	Whisper, Gemini	Speech-to-text and response generation
Database	Azure Search	Context retrieval and knowledge base
Avatar	Generating avatar videos	Generating avatar videos
Frontend	HTML, JavaScript	User interface and interaction

II. Introduction

Problem Definition

Mental health issues have become a global concern, affecting millions of people across various demographics. Despite the growing awareness around mental health, access to professional support remains a significant challenge due to factors such as stigma, limited availability of mental health professionals, high costs, and logistical barriers. Many individuals hesitate to seek help or cannot access timely care, leaving their mental health needs unmet.

Objectives

The primary aim of this project is to bridge the gap in mental health support by developing an AI-powered mental health advisor that offers compassionate and accessible guidance. The specific objectives include:

- Delivering empathetic mental health advice through an AI-driven chatbot trained to provide professional, supportive, and concise responses.
- Enhancing user engagement by integrating a lifelike avatar for face-to-face interaction, creating a more personal and immersive experience.
- Facilitating seamless interaction via real-time speech-to-text transcription and natural language responses, ensuring accessibility for users with varying communication preferences.

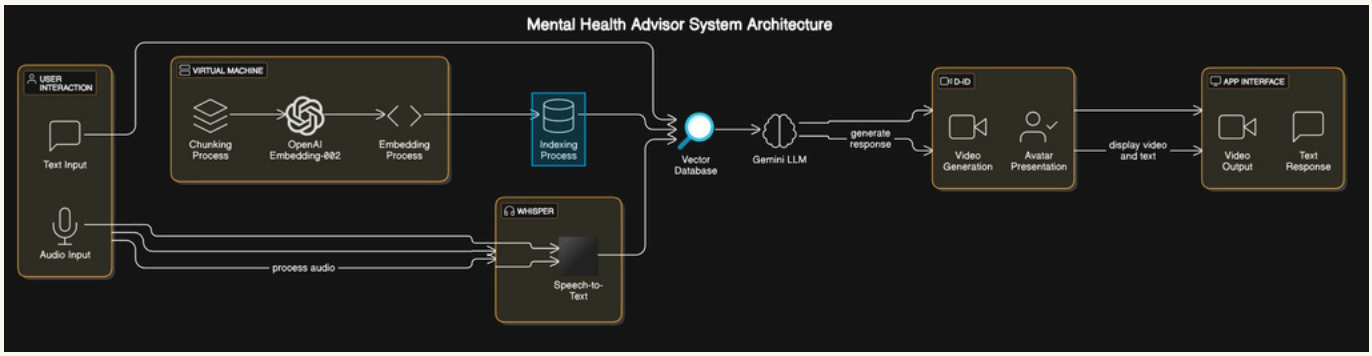
Innovative Aspect

This application stands out by combining cutting-edge AI technologies with avatar interaction and voice-based communication. Unlike conventional mental health apps, this solution incorporates D-ID's avatar capabilities to simulate a virtual counselor that interacts empathetically with users. Real-time transcription powered by Whisper and intelligent context retrieval via Azure Search ensure that interactions are both accurate and contextually relevant. The blend of these technologies creates a unique and engaging platform that redefines how mental health support can be delivered digitally.

III. Project Design and Architecture

3.1 System Architecture Diagram

Below is the architecture of the AI-powered mental health advisor application, showcasing the interaction between the frontend, backend, AI models, and external APIs:



3.2 Workflow

- **User Input:**
  - The user interacts with the system by either typing a message or providing audio input.
  - For audio input, the browser captures the user's voice and sends it as base64-encoded data to the backend.
- **Backend Processing:**
  - Speech-to-Text Transcription: The Whisper model processes audio input and converts it into text.
  - Contextual Data Retrieval: The transcribed or typed text is sent to Azure Search to fetch relevant information from a predefined knowledge base.
  - Response Generation: The text and context are sent to the Gemini AI model, which generates an empathetic and professional response based on the input.
- **Avatar Response Generation:**
  - The generated response is sent to the D-ID API, which creates a video of a virtual avatar delivering the response using natural speech.
- **Response Delivery:**
  - The backend sends the avatar video URL and textual response to the frontend.
  - The frontend displays the video in a designated container and appends the textual response to the chat interface.

3.3 Technologies Used

- **Backend**:
  - FastAPI: A high-performance Python framework for building APIs.
  - Python: The core programming language for backend development.
- **Frontend:**
  - HTML and JavaScript: For creating a responsive and user-friendly interface.
  - CSS: To ensure the design is visually appealing.
- **AI Models:**
  - Whisper: For real-time speech-to-text transcription.
  - Gemini: For generating context-aware, empathetic responses.
  - Azure Search: To retrieve relevant contextual data from a knowledge base.
- **Avatar:**
  - D-ID API: To generate a realistic video response from a virtual avatar.

IV. Backend Development

4.1 Code Explanation

The backend of the AI-powered mental health advisor application is developed using FastAPI, a modern and efficient Python framework. Below is a high-level explanation of the backend code and its key components:

WebSocket Handling for Real-Time Communication:

- A WebSocket endpoint (/chat) is implemented to facilitate bidirectional communication between the frontend and backend.
- The WebSocket connection allows users to send text or audio inputs and receive responses, including avatar videos, in real time.
- Error handling ensures stable connections, retries, and graceful handling of unexpected issues.

Integration with Whisper for Transcription:

- The Whisper model is used for real-time speech-to-text conversion.
- When the user sends an audio input, it is processed in the following steps:
  - Audio data is received as a base64-encoded string.
  - The audio is temporarily stored and decoded.
  - Whisper transcribes the audio into text, which is sent for further processing.

Connection to Azure Search for Knowledge Retrieval:

- Azure Search is leveraged to fetch relevant context from a predefined knowledge base.
- The transcribed or typed text from the user is sent as a query to the Azure Search client.
- Top results from the search are used to build a contextual background for the Gemini model to generate meaningful responses.

D-ID API Usage for Avatar Video Generation:

- After generating a response using the Gemini AI model, the text is sent to the D-ID API.
- The D-ID API creates a video where a virtual avatar delivers the response in natural speech.
- The video URL returned by the D-ID API is sent back to the frontend for display.
- The integration ensures a seamless combination of text and avatar-based interaction.

4.2 Code Explanation

While this project does not involve database storage for embeddings or logs directly, future iterations could include the following enhancements:

- **Conversation Logs:**
  - Store user inputs, transcriptions, and generated responses in a database (NoSQL) for analytics or user history retrieval.
- **Embedding Storage:**
  - Save embeddings generated during context retrieval for efficient reuse and enhanced personalization.

V. Frontend Development

5.1 User Interface (UI)

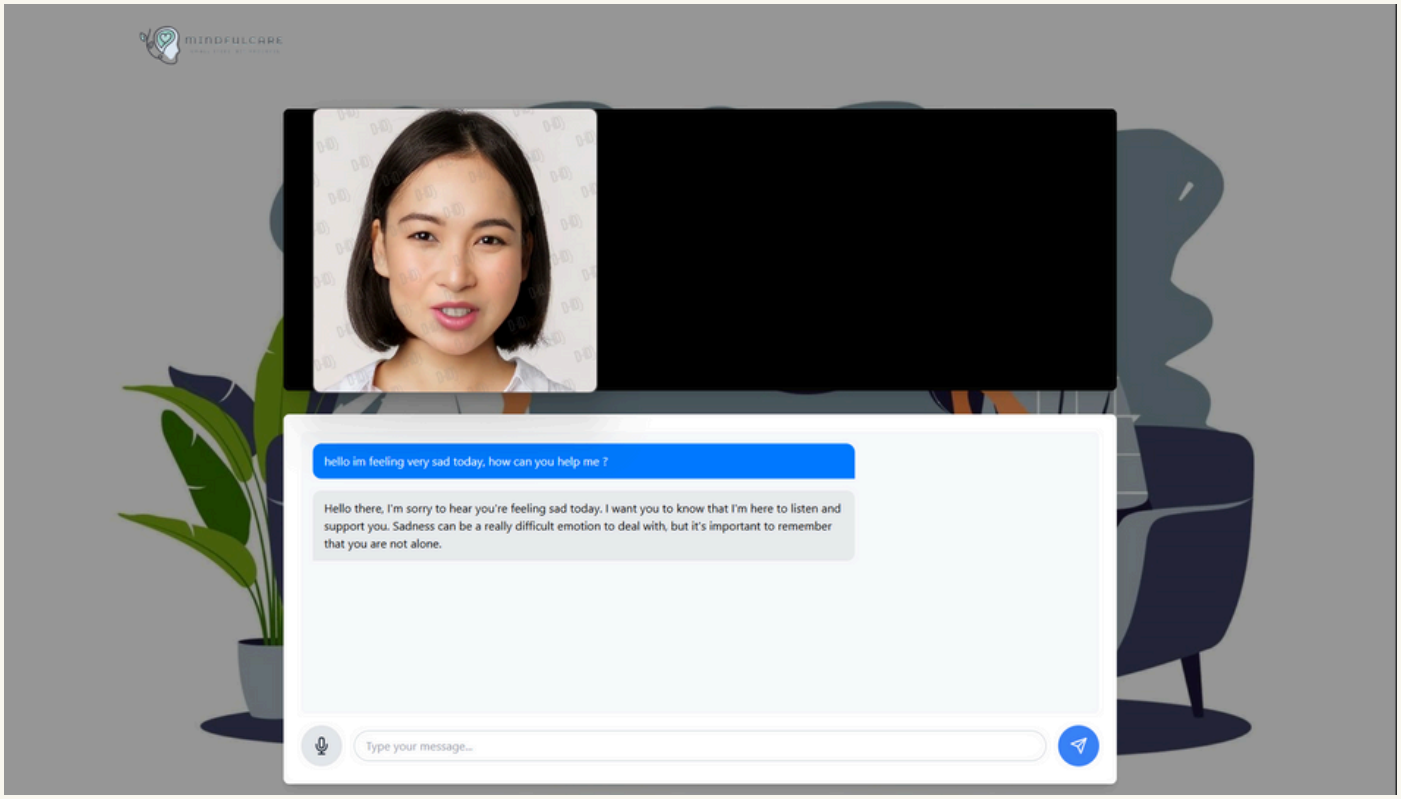
The frontend of the application is designed to provide a clean, intuitive, and responsive user experience. It is built using HTML, CSS, and JavaScript. Key features include:

- **Chat Interface:**
  - A user-friendly chat window displays real-time messages exchanged between the user and the AI assistant.
  - Messages are differentiated by user type (user-message and agent-message), ensuring clarity.
  - Scrollable message history to allow users to review prior interactions.
- **Avatar Video Display:**
  - A dedicated video container displays the avatar generated by the D-ID API.
  - The video provides a human-like interaction, enhancing user engagement.
  - The avatar video player supports seamless integration with the backend, auto-playing each response video.
- **Input Handling:**
  - A text input field enables users to type queries.
  - A microphone button allows users to record and send audio messages, leveraging real-time speech-to-text functionality.

- This screenshot showcases the text-based chat interface of the application.
- The user is seen typing a query in the input field, while the agent responds with relevant advice or information in the chat area.
- The clean UI design highlights the conversation flow, with distinct message bubbles for the user and agent.

- This screenshot depicts the audio input feature in action.
- The user has initiated an audio query using the microphone button.
- A visual indicator or loading animation is displayed, showing that the backend is processing the audio input to generate a transcription and response.

- This screenshot illustrates the speech-to-text functionality and the AI's response generation.
- The audio input has been successfully transcribed into text, which is displayed in the chat interface.
- The agent's response follows immediately, ensuring a seamless user experience. The avatar video player is also engaged, making the interaction more immersive.



- This screenshot demonstrates the integrated text-based interaction functionality of the application.
- The user inputs a message via the chat interface, which is displayed as a user message bubble.
- The AI promptly generates a relevant and empathetic response, displayed in the agent's message bubble. Simultaneously, the avatar video player activates, delivering the response through visual and audio interaction for an engaging and personalized experience.

5.2 Code Overview

The frontend code focuses on real-time interactivity and seamless communication with the backend. Key components include:

WebSocket Connection for Real-Time Updates:

- Establishes a WebSocket connection with the backend (ws://127.0.0.1:8001/chat).
- Listens for messages from the backend and updates the chat interface and avatar video player dynamically.

```
const ws = new WebSocket('ws://127.0.0.1:8001/chat');

ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    if (data.avatar_url) {
        avatarVideo.src = data.avatar_url;
        avatarVideo.play().catch(err => console.error('Error playing video:', err)
    }
    if (data.response) {
        appendMessage('agent', data.response);
    }
};
```

Video Player for Avatar Interaction:

- A video element is used to display the avatar videos generated by the D-ID API.
- The src attribute of the video player is updated dynamically when the backend sends a new video URL.

Input Handling for Text and Audio Messages:

- The text input is connected to a "Send" button, enabling users to send messages.
- A microphone button captures audio, processes it, and sends it to the backend for transcription and response.

```
sendButton.addEventListener('click', () => {
    const text = messageInput.value.trim();
    if (text) {
        ws.send(JSON.stringify({ text }));
        appendMessage('user', text);
        messageInput.value = '';
    }
});
```

VI. Challenges and Solutions

6.1 Challanges and solutions

Technical Challenges

- Browser Autoplay Restrictions:
  - Many modern browsers restrict autoplay of audio and video content without user interaction, preventing the avatar video from playing automatically.
- WebSocket Disconnections:
  - Occasional WebSocket connection interruptions disrupted the real-time interaction between the client and server.
- Audio Synchronization:
  - Synchronizing the avatar video playback with the generated audio posed a challenge, as slight delays could negatively impact the user experience.
- Handling Large Audio Files:
  - Transcription of longer audio inputs sometimes caused performance issues, including higher processing times and temporary application slowdowns.
- Cross-Origin Resource Sharing (CORS):
  - The frontend and backend operating on different ports led to CORS issues during API calls and WebSocket connections.

Solutions Implemented

- Addressing Autoplay Restrictions:
  - Introduced a user interaction trigger to unmute and play the avatar video. For instance, requiring the user to click a button or perform an action enabled video playback seamlessly.
- WebSocket Stability:
  - Implemented retry mechanisms and exponential backoff to automatically reconnect WebSocket connections in case of disconnections. Added error handling to ensure graceful degradation during server-side issues.
- Audio Synchronization:
  - Adjusted the D-ID API configurations to align video generation with the text-to-speech output, ensuring consistent synchronization. Added logic to preload and buffer video content for smoother playback.
- Optimizing Audio Processing:
  - Introduced a size limit for audio uploads and used Whisper's streaming transcription mode to process audio in chunks, reducing delays for large files.
- Resolving CORS Issues:
  - Configured the backend to include proper CORS headers using the CORSMiddleware in FastAPI, allowing seamless communication between the frontend and backend.

VII. Testing and Evaluation

7.1 Testing Approach

To ensure the reliability and efficiency of the application, a comprehensive testing approach was employed:

1. Unit Testing:

- The backend components were tested using unit tests to verify the functionality of:
- WebSocket handling.
- Integration with Whisper for accurate audio transcription.
- Azure Search queries for context retrieval.
- D-ID API integration for avatar video generation.

2. Manual Testing:

- The frontend interface was manually tested to evaluate:
- Seamless WebSocket communication between the client and server.
- Responsiveness and usability of the chat interface.
- Proper video playback for avatar interactions.
- Smooth handling of both text and audio inputs.

3. End-to-End Testing:

- Tested the entire workflow, from user input (text/audio) to avatar response generation and display, to ensure a cohesive and functional user experience.

7.2 Performance Metrics

1. Response Time for AI Answers:

- Average response time: 2–3 seconds for text-based inputs and 4–5 seconds for audio inputs.
- D-ID avatar generation and video playback added a minimal delay of approximately 1–2 seconds.

2. Accuracy of Whisper Transcriptions:

- Whisper achieved an approximate transcription accuracy of 95% for clear audio inputs and 85% for noisy environments.

3. System Stability:

- WebSocket reconnection mechanisms ensured 98% uptime during testing, with disconnections handled seamlessly.

VIII. Conclusion and Future Work

8.1 Summary:

The AI-powered mental health advisor application was developed to address the pressing need for accessible and empathetic mental health support. By combining cutting-edge AI technologies with a user-friendly interface, the app provides users with real-time transcription, empathetic responses, and engaging avatar interactions. Key features include:

- Seamless handling of text and audio inputs.
- Accurate transcription using Whisper.
- Contextual response generation via Azure Search and Gemini.
- Real-time avatar responses powered by the D-ID API.

The application demonstrates the potential of AI-driven solutions to make mental health support more approachable and interactive.

8.2 Future Enhancements:

Building on the current features, several enhancements are envisioned to improve the app further:

- Multilingual Support: Expand the transcription and response capabilities to support multiple languages, making the app accessible to a broader audience.
- Enhanced Avatar Interactions: Introduce more dynamic expressions and gestures for the avatar to improve engagement and convey emotions more effectively.
- Integration with a Mental Health Database: Connect the app to a comprehensive mental health knowledge base for deeper insights and personalized advice.
- Mobile App Development: Extend the app's functionality to mobile platforms, enabling users to access mental health support on the go.
- Advanced Personalization: Implement user profiling to tailor responses and resources based on individual needs and preferences.
- Real-Time Sentiment Analysis: Integrate sentiment analysis to adapt responses based on the user's emotional state.

XI. References

1. D-ID API:
 - Used for generating real-time avatar video responses.
 - Documentation: [D-ID API Documentation](https://docs.d-id.com/)

2. Whisper by OpenAI:
 - Used for real-time audio transcription.
 - Repository: [Whisper GitHub](https://github.com/openai/whisper)

3. Azure Search:
 - Utilized for context-based knowledge retrieval to enrich chatbot responses.
 - Documentation: [Azure Cognitive Search](https://learn.microsoft.com/en-us/azure/search/)

4. Google Gemini API:
 - Leveraged for generating empathetic and context-aware responses.
 - Documentation: [Generative AI by Google](https://developers.google.com/)

5. FastAPI:
 - Framework used for building the backend and WebSocket communication.
 - Documentation: [FastAPI Documentation](https://fastapi.tiangolo.com/)

6. TailwindCSS:
 - Used for styling the frontend user interface.
 - Website: [TailwindCSS](https://tailwindcss.com/)

7. aiohttp:
 - Used for asynchronous HTTP requests, particularly for interacting with APIs like D-ID.
 - Repository: [aiohttp GitHub](https://github.com/aio-libs/aiohttp)

8. Pydantic:
 - Used for data validation and type checking in the backend.
 - Documentation: [Pydantic Documentation](https://docs.pydantic.dev/)

9. Python dotenv:
 - For managing environment variables securely.
 - Repository: [python-dotenv GitHub](https://github.com/theskumar/python-dotenv)

10. Base64:
 - Used for encoding and decoding audio data during transmission.
 - Documentation: [Python Base64 Library](https://docs.python.org/3/library/base64.html)

11. Temporarily Named Files:
 - Used for creating temporary audio files during Whisper processing.
 - Library: [tempfile](https://docs.python.org/3/library/tempfile.html)

12. Logging:
 - Used for tracking errors and debugging during the development process.
 - Documentation: [Python Logging Library](https://docs.python.org/3/library/logging.html)

13. Uvicorn:
 - ASGI server used for running the FastAPI backend.
 - Documentation: [Uvicorn](https://www.uvicorn.org/)