Debate Bot Aurelius: Formal Documentation

Overview

Debate Bot Aurelius is an interactive web application designed to simulate, evaluate, and enhance the experience of debating using advanced AI features. The platform allows users to participate in AI-facilitated debates, receive personalized performance feedback, view full transcripts, and engage with various AI personalities. The application emphasizes usability, real-time voice interaction, and robust evaluation capabilities.

Table of Contents

1. Features
2. Exploring the Application
3. Technology Stack
4. Workflow and Development Process
5. Running a Local Demo

Features Overview

The core functionality of Debate Bot Aurelius includes the following modules and capabilities:

1. Debate Session Management

Users can initiate and manage debate sessions with full control over configuration settings. The session flow is supported by custom React hooks and specialized components, ensuring a smooth experience from start to finish.

- **Key Files and Components:**
    - `useDebateSession.ts`: Handles session state and logic.
    - `DebateSetup.tsx`: Interface for configuring and launching debates.
    - `DebateInterface.tsx`: Main platform for live debating.
    - `debate.ts`: Contains essential data structures and debate logic.

2. Voice Input and Output

Debate Bot Aurelius features real-time voice capabilities. Users can communicate through speech and receive synthesized voice responses. Additionally, the system supports customization of voice parameters such as pitch, speed, and volume, and offers multiple voice options (e.g., Microsoft Zira, David).

- **Voice Input Components:**
    - `VoiceInput.tsx`: Captures and transcribes speech.
    - `voiceService.ts`: Manages voice processing functions.
- **Voice Output Components:**
    - `VoiceOutput.tsx`: Delivers AI-generated voice responses.
- **Testing & Debugging:**
    - `VoiceTest.tsx`: Utility for testing voice functionality.
    - `DEBUG_VOICE.md`, `VOICE_FEATURES.md`: Documentation for troubleshooting voice features.

3. AI Personalities

The platform provides several AI personalities, each with unique debating styles. These personalities allow users to experience dynamic and diverse debate scenarios.

- **Key File:**
    - `aiPersonalities.ts`: Defines and manages AI personas.

4. Personalized Feedback

At the end of each debate session, users receive detailed feedback generated by the Gemini AI model. This includes insights on performance, strengths, areas for improvement, and personalized suggestions.

- **Key Component:**
    - `PersonalizedFeedback.tsx`: Displays structured feedback.

5. Debate History

Debate Bot Aurelius stores a comprehensive history of user sessions, allowing users to track their progress and review past debates.

- **Key Files:**
    - `DebateHistory.tsx`: Interface for viewing historical sessions.
    - `useDebateHistory.ts`: Logic for retrieving and storing session data.

6. Evaluation Dashboard

A dedicated dashboard provides performance metrics such as debate scores and participation analytics.

- **Key Component:**
    - `EvaluationDashboard.tsx`: Displays evaluative data and performance statistics.

7. Transcript Viewer

Users have access to complete transcripts of their debates. The viewer supports both on-screen reading and transcript downloads for future reference.

- **Key Components:**
    - `TranscriptViewer.tsx`: Viewer interface.
    - `transcriptUtils.ts`: Supporting utilities.

8. Case Preparation

Prior to engaging in debates, users can structure their arguments and organize evidence using a dedicated preparation workspace.

- **Key Component:**
    - `CasePreparation.tsx`: Tool for building and outlining cases.

9. Motion Selector

Users can either select debate motions from a predefined list or opt for randomized topics to streamline the setup process.

- **Key Files:**
    - `MotionSelector.tsx`: Interface for topic selection.
    - `motions.ts`: Repository of debate motions.

10. API Key Setup

To access AI services, users must input their API key through a secure configuration interface.

- **Key Component:**
    - `ApiKeySetup.tsx`: Interface for API key management.

11. Typing Effect

Bot responses are rendered with a simulated typing effect, adding realism to the chat experience.

- **Key Files:**
    - `TypingMessage.tsx`: Renders messages with animation.
    - `useTypingEffect.ts`: Implements typing behavior.

12. Local Storage and Persistence

User preferences, debate data, and session states are saved in the browser's local storage to ensure continuity across sessions.

- **Key File:**
    - `useLocalStorage.ts`: Manages persistent storage.

13. Debate Results

Summary results, including scores and verdicts, are displayed at the conclusion of each debate.

- **Key Component:**
    - `DebateResults.tsx`: Presents final outcomes.

14. Testing and Debugging Tools

The application includes extensive testing tools and documentation, particularly for voice-related features, to ensure high performance and ease of troubleshooting.

- **Resources:**
    - `VoiceTest.tsx`: Voice testing utility.
    - `DEBUG_VOICE.md`, `VOICE_FEATURES.md`: Advanced guides for developers.

15. Utilities and Configuration

A collection of configuration files supports styling, build processes, and general utilities.

- **Key Files:**
    - `config.json`, `api.ts`, `tailwind.config.js`, `vite.config.ts`, `index.css`, `index.html`

16. Documentation and Guides

Markdown files located in the project directory offer setup instructions, feature documentation, demo walkthroughs, and troubleshooting tips.

Exploring the Application

To use all available features, follow these steps:

1. **Debate Setup**
   Launch the application and access the debate setup screen. Choose or randomize a motion and configure the participant roles.

2. **Case Preparation**
   Organize your arguments, evidence, and rebuttals before starting.

3. **Start the Debate**
   Use the `DebateInterface.tsx` to initiate and conduct the debate session.

4. **Voice Interaction**
   Utilize speech input and listen to AI-generated responses. Testing tools are available for ensuring voice features work correctly.

5. **Select AI Personalities**
   Customize your experience by choosing from a variety of AI personas.

6. **Receive Feedback**
   Upon completion, view detailed feedback based on AI analysis.

7. **Use the Evaluation Dashboard**
   Analyze performance metrics and trends over time.

8. **Review Debate History**
   Browse past sessions and transcripts for learning and progress tracking.

9. **View Transcripts**
   Access full-text transcripts and download them if needed.

10. **Configure API Key**
    Set up the AI API key securely in the dedicated setup section.

11. **Explore Additional Utilities**
    Utilize the typing effect feature, and consult the markdown files for advanced functionality and debugging.

Technology Stack

- **Frontend**: React with TypeScript
- **Build Tool**: Vite
- **Styling**: Tailwind CSS
- **AI Integration**: Gemini AI API (or similar)
- **Voice Technology**: Web Speech API
- **State Management**: React hooks and browser local storage
- **Testing**: Custom test tools and documentation

Workflow and Development Process

1. Planning & Design

- Defined essential features such as voice interaction, debate management, and feedback generation.
- Designed user flows and component structure.

2. Core Implementation

- Developed the debate interface and session logic.
- Integrated AI and voice functionality.

3. Feature Expansion

- Added history tracking, motion selection, case preparation, and transcript viewing.

4. Testing & Debugging

- Built testing tools and detailed documentation for voice-related issues.
- Refined the user experience through iterative testing.

5. Finalization and Documentation

- Created comprehensive markdown guides.
- Ensured feature accessibility and maintainability.

Learnings and Challenges

- **Voice Integration**: Required handling permissions and browser compatibility.
- **AI Feedback**: Involved continuous tuning for meaningful results.
- **State Persistence**: Robust local storage was essential for smooth sessions.
- **Testing**: Extensive testing was needed across devices and environments.

Running a Demo Locally

Prerequisites

- **Node.js** (version 14 or higher)
- **npm** (included with Node.js)

Steps

Clone the Repository

```
git clone <repo-url>
cd Debate_Bot2_main
```

1.

Install Dependencies

bash
CopyEdit

```
npm install
```

2.
3. **Configure API Key**
    - Open the application and go to the API Key Setup section.
    - Enter the AI API key as required.

Start the Application

bash
CopyEdit

```
npm run dev
```

4.
    - Access the application at `http://localhost:5173` or the URL specified in the terminal.
5. **Explore the Features**
    - Follow the steps under "Exploring the Application" for a guided tour.

Additional Notes

- Refer to `DEBUG_VOICE.md` and `VOICE_FEATURES.md` for advanced voice debugging.
- For full feature exploration and demo usage, consult `DEMO.md` and `SUBMISSION.md`.

- This application is intended primarily for educational and demonstration purposes.