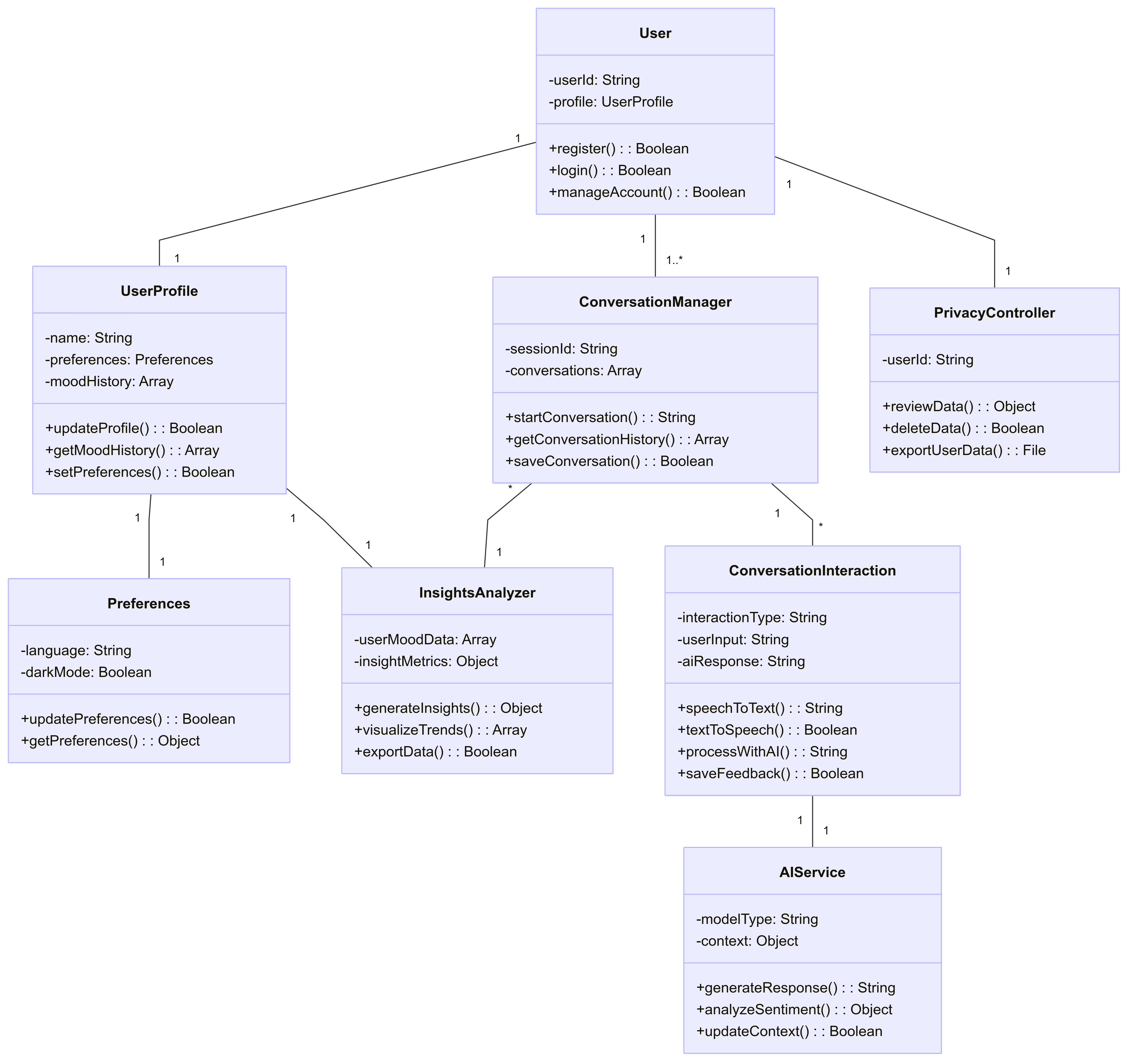
**Product Design**

|  |
| --- |
| **Team 49**  **Wave Diaries**  **Shivam Gupta, Aditya Gaur, Raunak Seksaria , Manit Roy** |

# Design Model

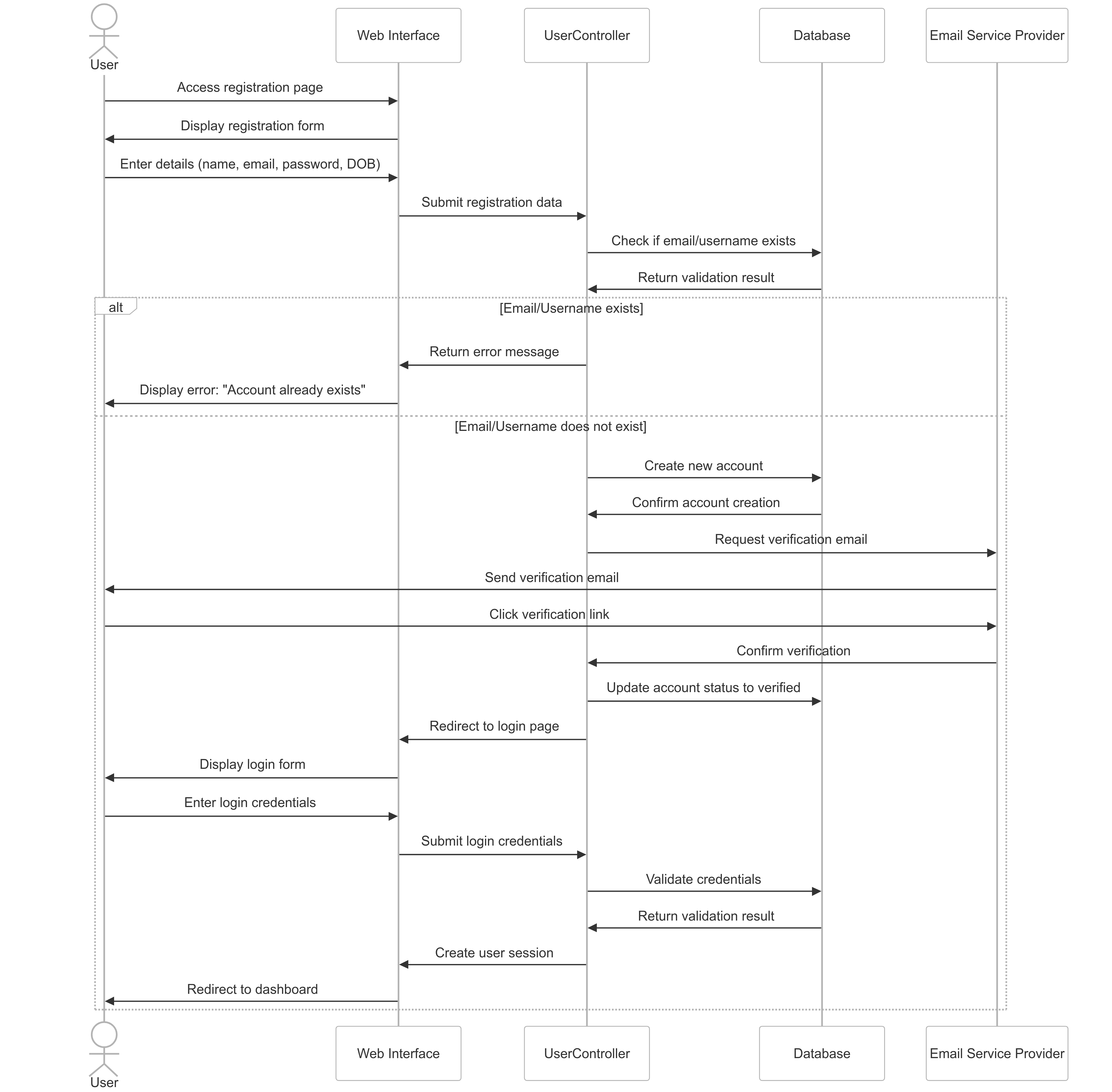


|  |  |
| --- | --- |
| **User** | **Class state:**  **userId**: Unique identifier for the user  **profile:** Reference to the user’s profile  **Class behavior:**  **register()**: Handles user registration process  l**ogin()**: Authenticates and logs in the user  **manageAccount()**: Allows users to update their account information |
| **UserProfile** | **Class state:**  **Name:** User’s name  **Preferences:** User’s app preferences  **MoodHistory:** Array of past mood entries  **Class behavior:**  **updateProfile():** Updates user profile information  **getMoodHistory():** Retrieves mood history for analysis  **setPreferences():** Updates user preferences |
| **Preferences** | **Class state:**  **language**: Preferred language for the user  **darkMode:** UI theme preference  **Class behavior:**  **updatePreferences():** Updates user preferences  **getPreferences():** Retrieves current preferences |
| **Conversation** | **Class state:**  **sessionId:** Unique identifier for conversation session  **conversationArray:** Array that holds messages  **conversationTime:** Timestamps of conversation  **Class behavior:**  **addMessage()**: Adds a new message to the conversation  **getConversationHistory():** Retrieves the entire conversation’s history |
| **Conversation Manager** | **Class state:**  **sessionId:** Unique identifier for conversation  **conversation:** Array of conversation objects  **Class behavior:**  **startConversation():** Initiates a new conversation  **getConversationHistory():** Retrieves past conversations  **saveConversation():** Persists conversation data |
| **ConversationInteraction** | **Class state:**  **interactionType:** Type of interaction (speech-to-speech, text-to-text, etc.)  **userInput:** User’s message input  **aiResponse:** Response from the AI model  **Class behavior:**  **speechToText():** Converts user speech to text  **textToSpeech():** Converts AI response to speech  **processWithAI():** Sends input to AI service for processing  **saveFeedback():** Records user feedback about interaction |
| **AIService** | **Class state:**  **modelType**: Type of AI model being used (Groq, Gemini, GPT, etc.)  **context:** Contextual information for personalized responses  **Class behavior:**  **generateResponse():** Creates therapeutic responses based on user input  **analyzeSentiment():** Performs sentiment analysis on user input  **updateContext():** Updates conversation context for better personalization |
| **InsightsAnalyzer** | **Class state:**  **userMoodData:** Collection of user mood data  **insightMetrics:** Metrics derived from mood data  **Class behavior:**  **generateInsights()**: Analyzes mood data to generate insights  **visualizeTrends():** Creates visualizations of mood changes over time  **exportData():** Exports analyzed data for sharing with therapists |
| **PrivacyController** | **Class state:**  **userId**: Reference to user whose data is being managed  **Class behavior:**  **reviewData():** Allows reviewer to see their stored data  **userConsents():** Oversee user’s data consents  **exportUserData():** Exports user data upon user request  **deleteUserData():** Deletes user data as required  **manageConsents()**: Manages user data consent statuses |

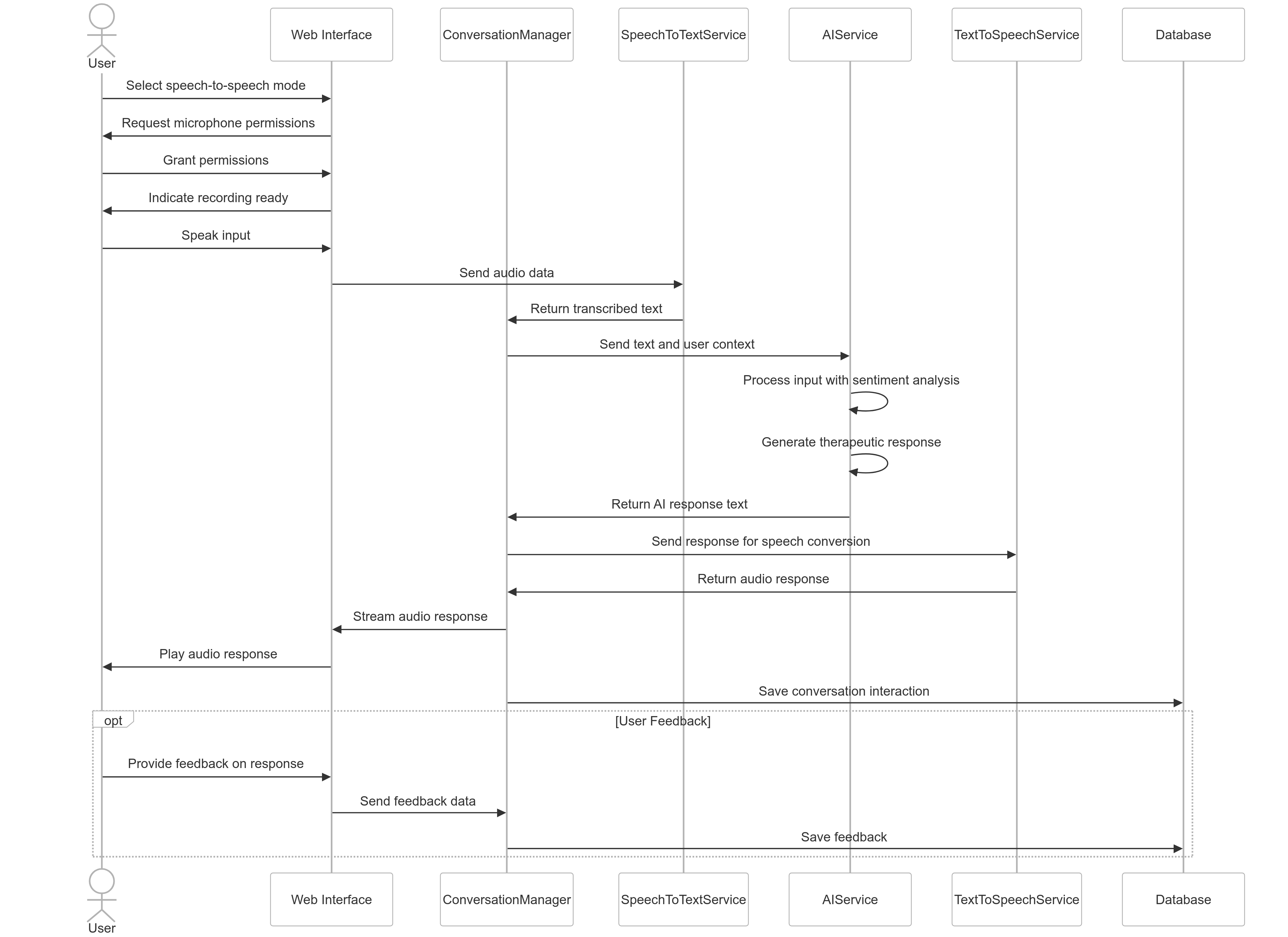
.

# Sequence Diagram(s)*.*

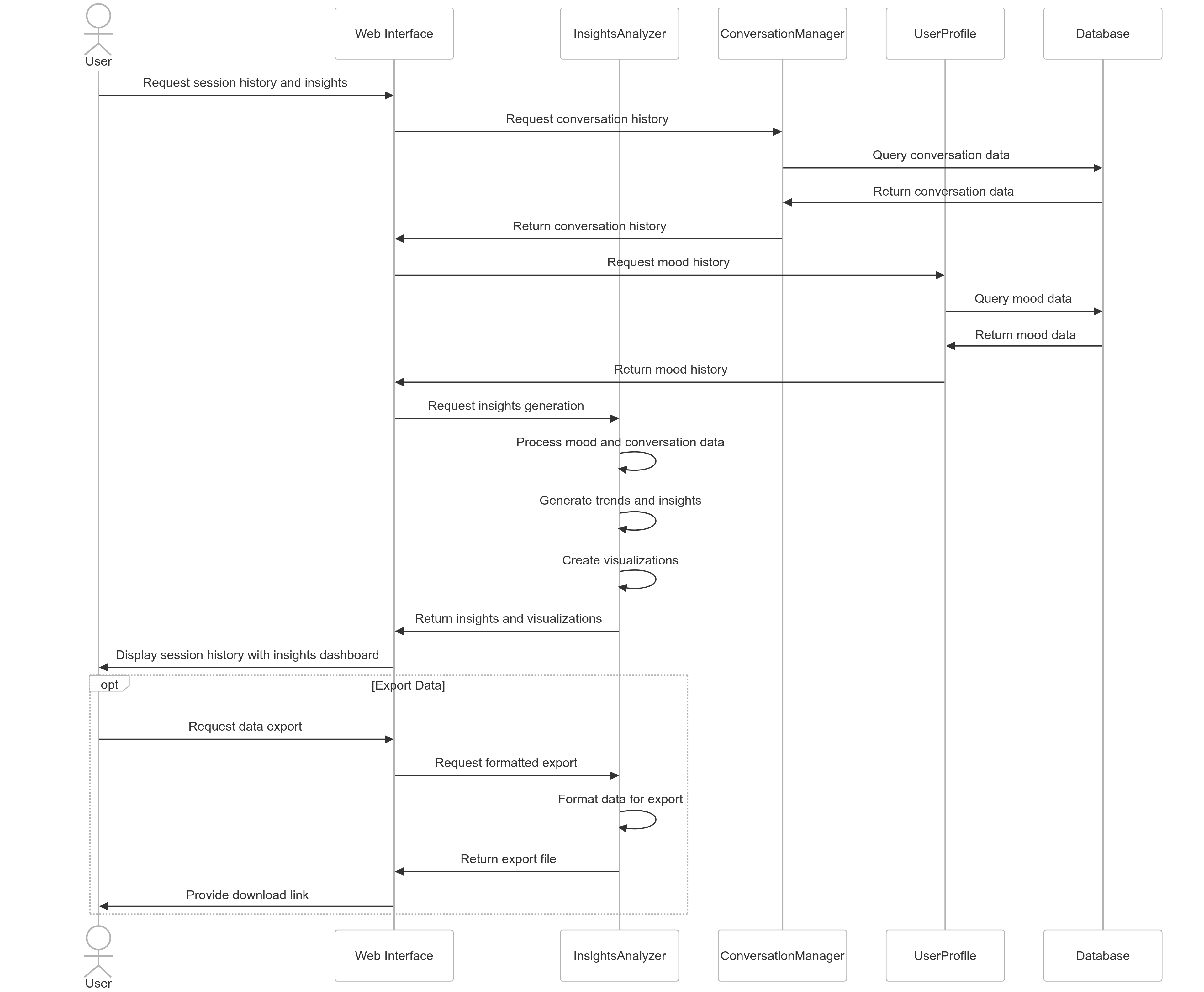
1. U**ser Registration and Login Sequence Diagram**



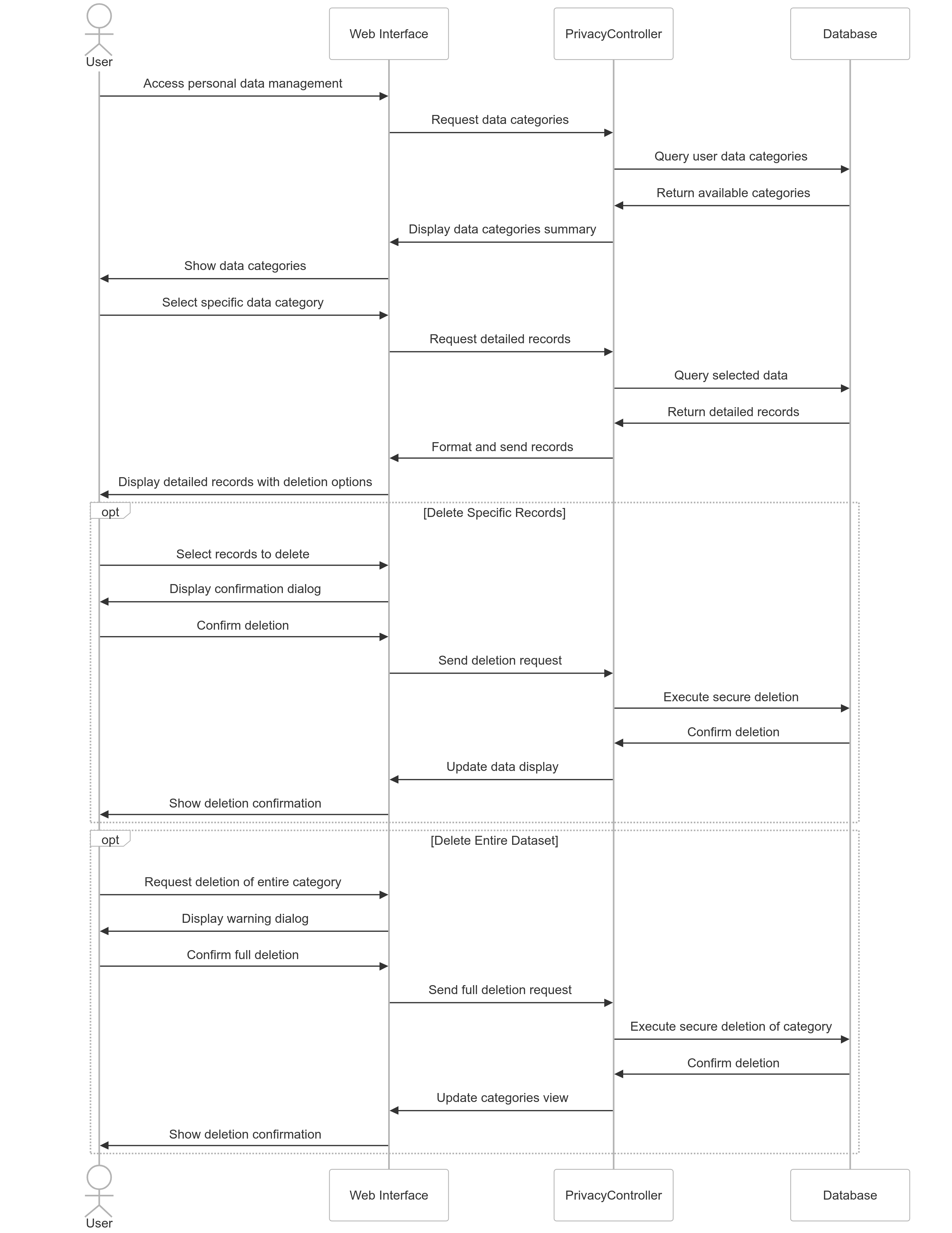
1. Speech-to-Speech Conversation Sequence Diagram



1. Session History and Insights Sequence Diagram



1. User Privacy and Data Control Sequence Diagram



# Design Rationale

**Design Rationale for Mood Tracking Application**

Based on the class diagram and sequence diagrams provided, I'll develop a comprehensive design rationale document that captures the thought process, design decisions, alternatives considered, and tradeoffs made during the design of this mood tracking application.

**1. User Management and Authentication**

**Chosen Design**

The application uses a separate User and UserProfile class structure with email verification for account creation.

**Alternatives Considered**

1. **Single User Class Approach**: Combining User and UserProfile into a single class.
   * Pros: Simpler data model, fewer database relationships
   * Cons: Less flexibility for profile customization, harder to separate authentication from preferencesd
2. **Social Authentication Option**: Using OAuth with third-party providers (Google, Facebook)
   * Pros: Faster onboarding, reduced friction, no need for email verification
   * Cons: Dependency on external services, potential privacy concerns, more complex implementation

**Rationale for Final Decision**

We separated User from UserProfile to create a cleaner separation of concerns. The User class handles authentication and account management, while UserProfile focuses on user-specific data and preferences. This approach:

1. Makes it easier to extend profile features without modifying authentication logic
2. Provides better modularity for maintenance and future enhancements
3. Allows for more granular privacy controls
4. Provides a foundation for potential multi-profile support in the future

Email verification was chosen despite the additional friction to ensure user authenticity and reduce spam accounts, which is important for a mental health/mood tracking application.

**2. Conversation Management and AI Integration**

**Chosen Design**

A centralized ConversationManager interacting with specialized services (STT, TTS, AI) for processing user interactions.

**Alternatives Considered**

1. **Monolithic AI Service**: Having a single AIService handle all aspects of the conversation.
   * Pros: Simplified implementation, fewer moving parts
   * Cons: Less flexibility, harder to swap out individual components, harder to test
2. **Client-Side Processing**: Moving speech-to-text processing to the client side.
   * Pros: Reduced server load, potentially faster response times
   * Cons: Inconsistent performance across devices, increased app size, security concerns
3. **Third-Party Conversation API**: Outsourcing conversation management to a specialized service.
   * Pros: Faster development, potentially better AI capabilities
   * Cons: Dependency on external service, less control over data, potential costs

**Rationale for Final Decision**

We chose a service-oriented architecture with specialized components because:

1. It provides modularity that allows services to be improved or replaced independently
2. The separation of speech-to-text and text-to-speech from AI processing allows for specialized optimization of each function
3. It enables easier testing of individual components
4. The centralized ConversationManager provides a consistent interface for the UI while abstracting the complexity of multiple services
5. It creates a foundation for potentially adding other input/output modalities in the future (text, images, etc.)

**3. Mood Tracking and Insights Analysis**

**Chosen Design**

Using InsightsAnalyzer as a central component to process mood data and generate insights, with data stored in the user profile.

**Alternatives Considered**

1. **Real-time Analysis Only**: Not storing historical data and only providing in-the-moment insights.
   * Pros: Reduced data storage, privacy by design, simpler implementation
   * Cons: Limited trend analysis, less personalized over time, less user value
2. **Distributed Analytics**: Moving analytics processing to a separate microservice.
   * Pros: Better scalability, dedicated resources for computation-heavy processing
   * Cons: Increased system complexity, potential data synchronization issues
3. **Fixed Insight Categories**: Pre-defined insight types rather than flexible data-driven insights.
   * Pros: Easier to implement, consistent user experience
   * Cons: Less personalization, inability to discover unexpected patterns

**Rationale for Final Decision**

We chose to implement a centralized InsightsAnalyzer that processes both conversation and mood data because:

1. It enables comprehensive analysis across different data sources
2. It provides a centralized point for applying machine learning and analytics algorithms
3. Storing mood history in the user profile facilitates personalization over time
4. The design allows for both real-time feedback and longitudinal analysis
5. It strikes a balance between complexity and analytical power

**4. Privacy and Data Management**

**Chosen Design**

A dedicated PrivacyController class with fine-grained control over user data visibility and deletion.

**Alternatives Considered**

1. **Automatic Data Expiration**: Setting automatic data expiration periods.
   * Pros: Reduced privacy risk, automatic compliance with data retention policies
   * Cons: Potential loss of valuable historical data, less user control
2. **Anonymized-Only Storage**: Storing only anonymized data for insights.
   * Pros: Enhanced privacy, reduced regulatory concerns
   * Cons: Limited personalization, reduced feature set
3. **Integrated Privacy Controls**: Embedding privacy functions in other classes.
   * Pros: Simpler architecture, fewer components
   * Cons: Scattered responsibility, harder to ensure complete privacy implementation

**Rationale for Final Decision**

We implemented a dedicated PrivacyController because:

1. Privacy is a core concern for a mood tracking application handling sensitive data
2. Centralized privacy controls ensure consistent implementation of privacy policies
3. The design allows for granular control over different categories of user data
4. It simplifies compliance with regulations like GDPR that require comprehensive data management capabilities
5. It creates clear ownership of privacy features within the system architecture

**5. General Architecture Considerations**

**Chosen Design**

A modular, object-oriented architecture with clear separation of concerns.

**Alternatives Considered**

1. **Event-Driven Architecture**: Using a publish-subscribe model for system interactions.
   * Pros: Better decoupling, more scalable, easier to extend
   * Cons: More complex to understand, harder to debug, potential performance overhead
2. **Microservices**: Breaking the application into independently deployable services.
   * Pros: Better scalability, independent development and deployment
   * Cons: Increased operational complexity, potential performance issues from service communication
3. **Serverless Architecture**: Using function-as-a-service for key components.
   * Pros: Reduced operational overhead, better scalability, potential cost savings
   * Cons: Cold start issues, vendor lock-in, more complex local development

**Rationale for Final Decision**

We chose a modular object-oriented architecture because:

1. It provides a good balance between simplicity and separation of concerns
2. It's more appropriate for the initial scale of the application
3. The development team has strong expertise in this approach
4. It provides a foundation that could evolve toward microservices if needed
5. It simplifies testing and maintenance while still allowing for component evolution

**6. UI Interaction Model**

**Chosen Design**

A web interface with both text and speech interaction capabilities.

**Alternatives Considered**

1. **Native Mobile App**: Developing platform-specific mobile applications.
   * Pros: Better performance, deeper OS integration, better offline capabilities
   * Cons: Higher development cost, platform fragmentation, harder to update
2. **Text-Only Interface**: Focusing solely on text interaction.
   * Pros: Simpler implementation, wider device compatibility
   * Cons: Less accessible, more limited interaction model
3. **Voice-First Design**: Optimizing primarily for voice interaction.
   * Pros: More natural interaction, potentially more accessible
   * Cons: Privacy concerns in public settings, recognition challenges, more complex implementation

**Rationale for Final Decision**

We chose a web interface with multimodal interaction because:

1. It provides the widest accessibility across devices without platform-specific development
2. The combination of text and speech allows users to choose their preferred interaction mode
3. A web-based approach enables faster iterations and updates
4. It allows for progressive enhancement based on device capabilities
5. It creates a foundation that could be wrapped in a native container later if needed

This design rationale document captures the key decisions made during the design process and the reasoning behind them, providing valuable context for future development and evolution of the mood tracking application.