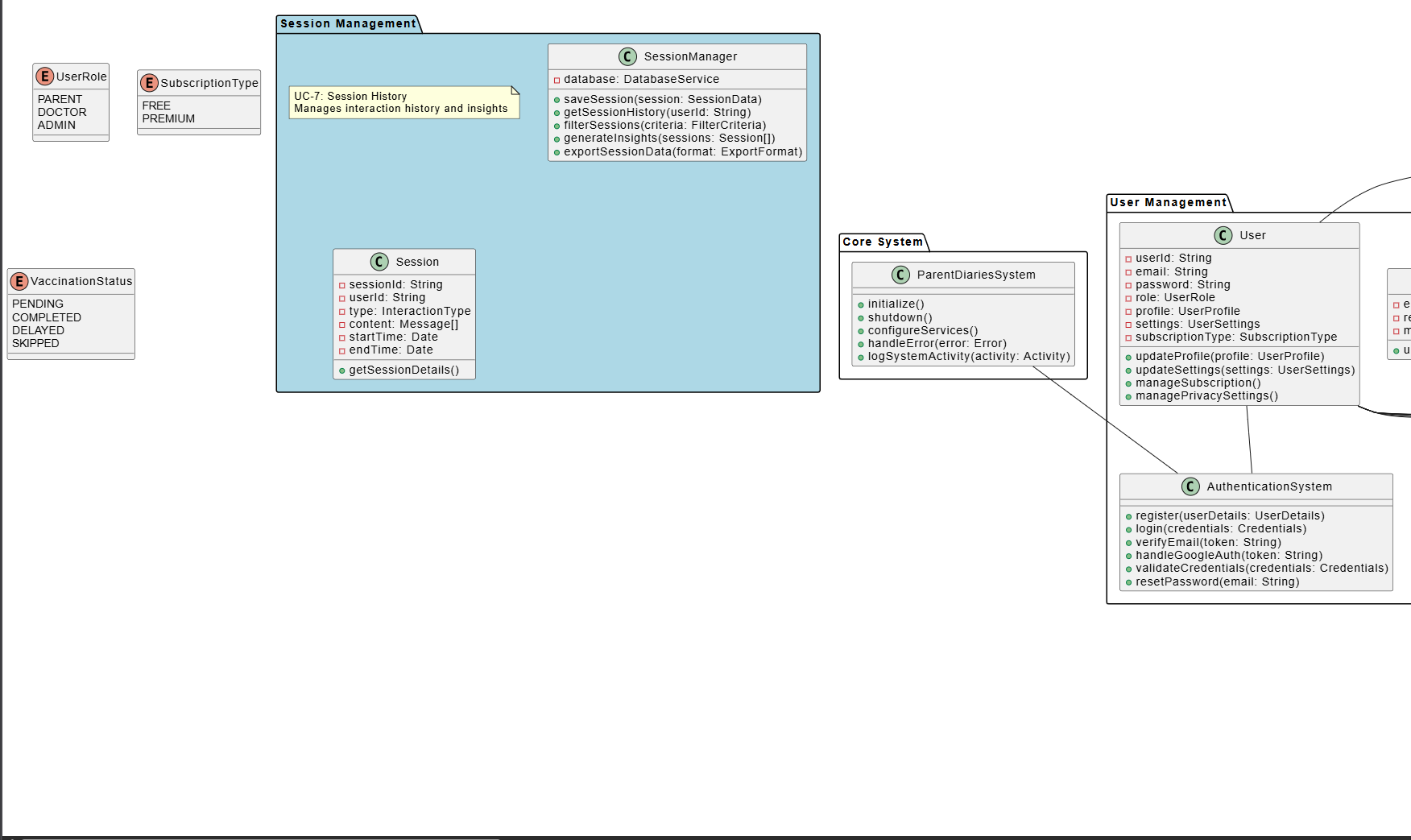
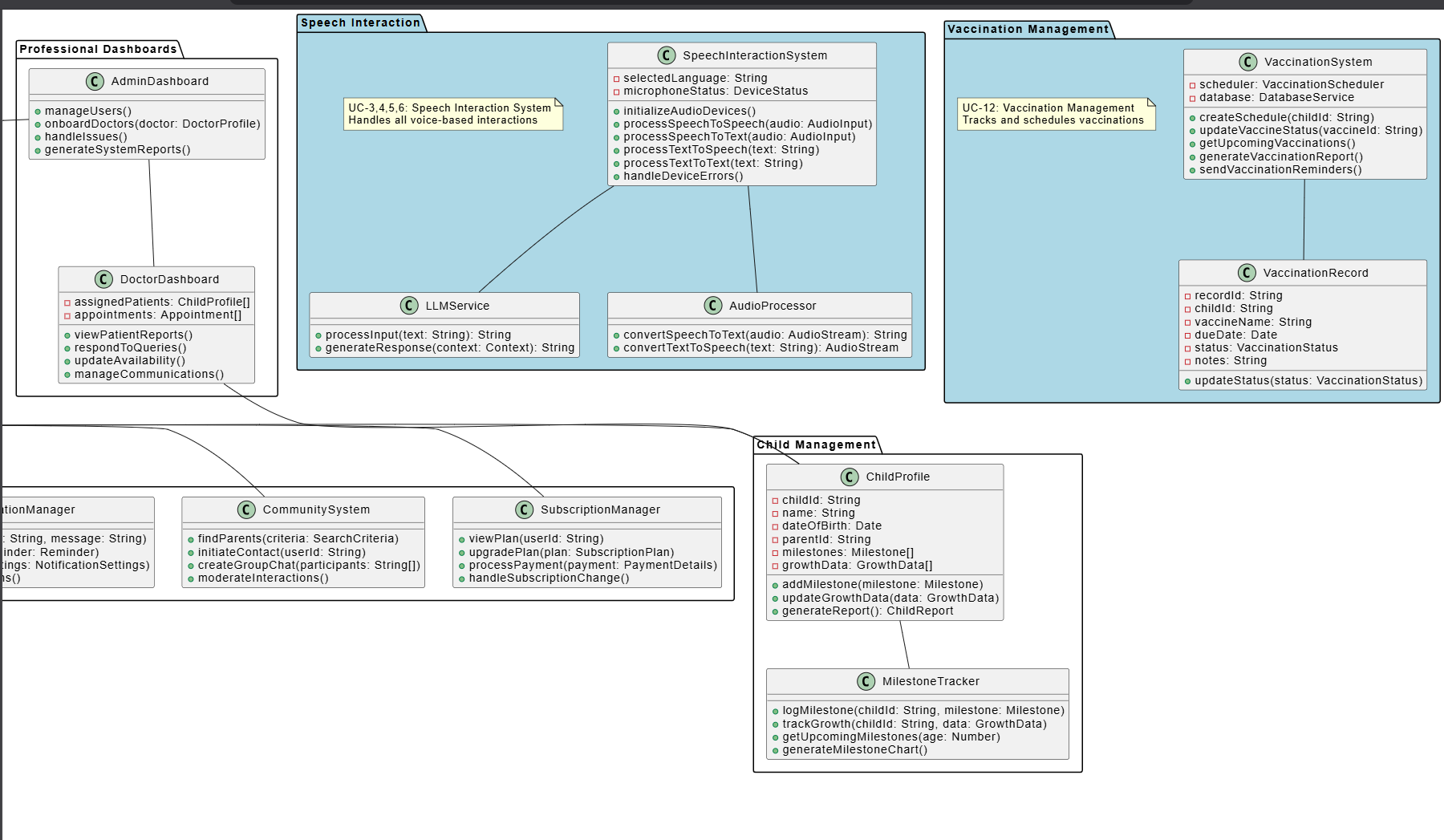
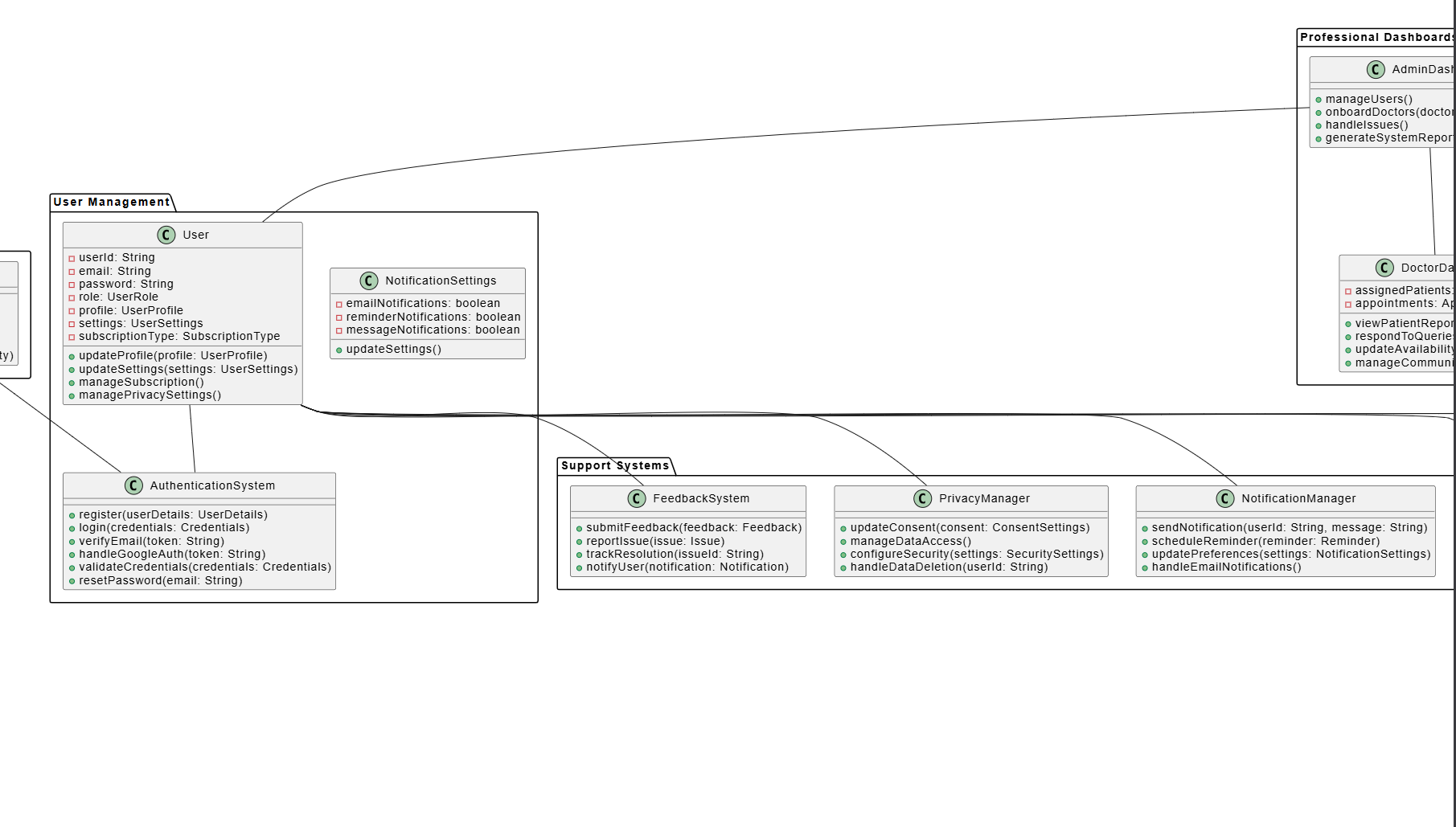
**Product Design**

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

# Design Model

UML CLASS DIAGRAM   
Attached in the same folder (Too large an image to put on the doc) still putting images broken down here:  


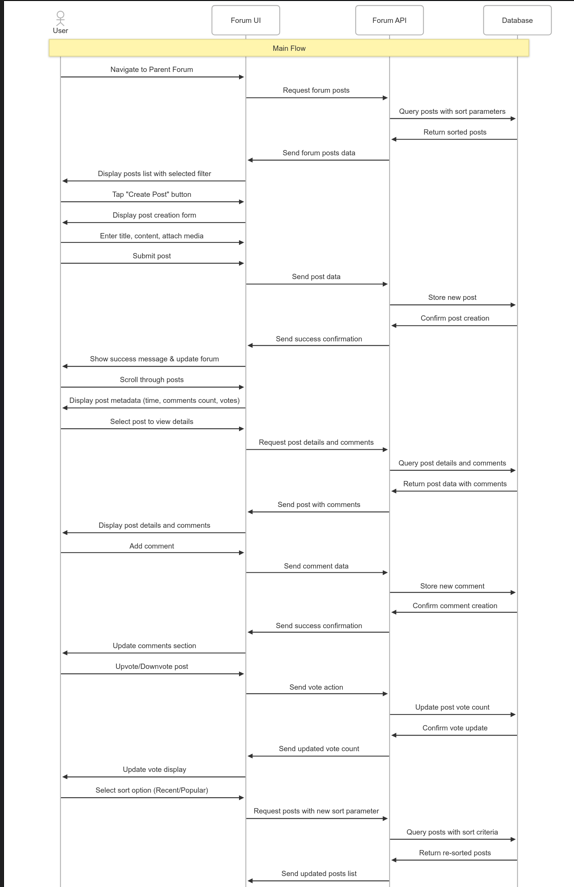


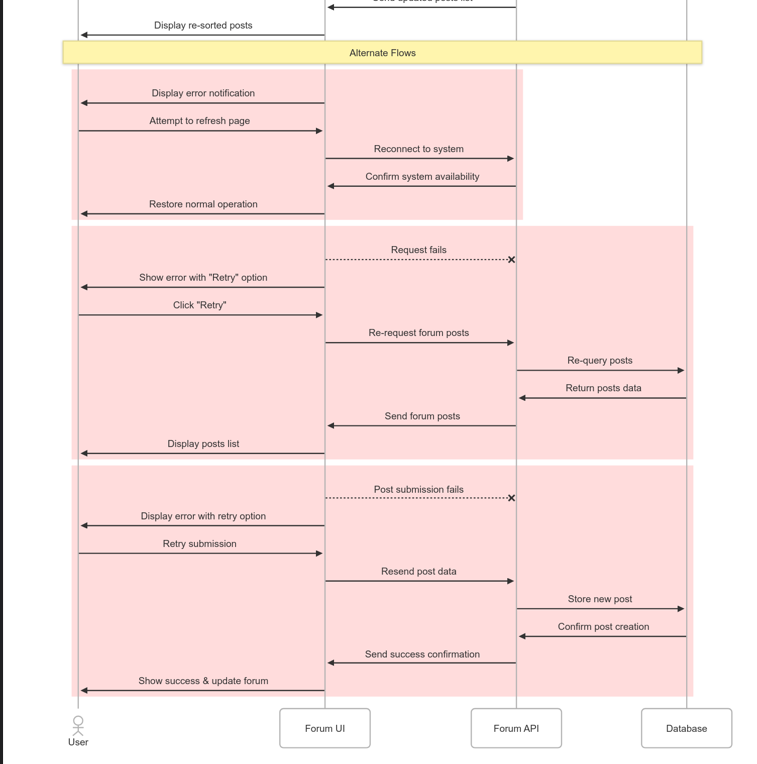
Explanation of various classes

|  |  |
| --- | --- |
| **Parent Diaries System** | **Class state:**   * No explicit attributes but manages overall system configuration   **Class behavior:**   * **initialize():** Starts up the system and initializes all components * **shutdown():** Properly closes all connections and terminates the system |
| **User** | **Class state:**   * **userName:** The unique identifier for a user * **email:** The user's email address for communication   **Class behavior:**   * **register():** Creates a new user account * **login():** Authenticates a user and grants access * **logout():** Ends a user session |
| **Subscription** | **Class state:**   * **subscriptionId:** Unique identifier for a subscription * **planType:** The type of subscription plan * **startDate:** When the subscription begins * **expiryDate:** When the subscription ends   **Class behavior:**   * **upgrade():** Changes to a higher-tier subscription * **cancel():** Terminates a subscription |
| **Vaccination Scheduler** | **Class state:**   * No explicit attributes but maintains vaccination records internally   **Class behavior:**   * **scheduleVaccination():** Creates a new vaccination appointment * **getUpcomingVaccinations():** Retrieves planned vaccinations |
| **PhysicalGrowthTracker** | **Class state:**   * No explicit attributes but stores growth measurements internally   **Class behavior:**   * **recordMeasurements():** Stores new height/weight measurements * **getGrowthHistory():** Retrieves historical growth data |
| **MilestoneTracker** | **Class state:**   * No explicit attributes but tracks developmental milestones internally   **Class behavior:**   * **recordMilestone():** Logs a new developmental achievement * **getMilestones():** Retrieves developmental milestone history |
| **NotificationSystem** | **Class state:**   * No explicit attributes but manages notification queue internally   **Class behavior:**   * **sendNotification():** Delivers immediate notifications * **scheduleNotification():** Sets up future notifications |
| **TextToTextInteraction** | **Class state:**   * No explicit attributes but maintains conversation context   **Class behavior:**   * **processTextInput():** Handles text input and returns text responses |
| **TextToSpeechInteraction** | **Class state:**   * No explicit attributes but maintains voice settings   **Class behavior:**   * **convertTextToSpeech():** Transforms text into spoken audio |
| SpeechToTextInteraction | **Class state:**   * No explicit attributes but manages speech recognition settings   **Class behavior:**   * **convertSpeechToText():** Transcribes spoken audio to text |
| SpeechToSpeechInteraction | **Class state:**   * No explicit attributes but maintains voice interaction context   **Class behavior:**   * **processSpeechInput():** Processes spoken input and returns spoken output |
| SessionHistory | **Class state:**   * No explicit attributes but stores interaction records   **Class behavior:**   * **recordSession():** Saves a new interaction session * **getHistory():** Retrieves past interaction sessions |
| Feedback | **Class state:**   * No explicit attributes but stores user feedback data   **Class behavior:**   * **submitFeedback():** Collects and stores user opinions and ratings |
| PrivacyManager | **Class state:**   * No explicit attributes but maintains privacy configuration   **Class behavior:**   * **updatePrivacySettings():** Changes privacy configurations * **getPrivacySettings():** Retrieves current privacy settings |
| EmailService | **Class state:**   * No explicit attributes but maintains email service configuration   **Class behavior:**   * **sendEmail():** Delivers emails to specified recipients |
| GoogleAccountService | **Class state:**   * No explicit attributes but maintains OAuth connections   **Class behavior:**   * **authenticateWithGoogle():** Handles Google sign-in authentication |
| TextToSpeechService | **Class state:**   * No explicit attributes but maintains voice synthesis settings   **Class behavior:**   * **convertText():** Transforms text into spoken audio via external service |
| SpeechRecognitionService | **Class state:**   * No explicit attributes but maintains speech recognition configurations   **Class behavior:**   * **recognizeSpeech():** Transcribes audio to text via external service |
| LLM Service | **Class state:**   * No explicit attributes but maintains language model configurations   **Class behavior:**   * **generateResponse():** Creates AI-generated text responses |
| DataAnalytics | **Class state:**   * No explicit attributes but maintains analytics datasets   **Class behavior:**   * **analyzeUserData():** Processes user data to generate insights and reports |
| PaediatricianService | **Class state:**   * No explicit attributes but maintains doctor directory   **Class behavior:**   * **findPaediatricians():** Locates nearby pediatricians * **bookAppointment():** Schedules doctor visits |
| PaymentGateway | **Class state:**   * No explicit attributes but maintains payment processor connections   **Class behavior:**   * **processPayment():** Handles financial transactions |
| MultilingualSupport | **Class state:**   * No explicit attributes but maintains language configurations   **Class behavior:**   * **translate():** Converts text between languages * **getSupportedLanguages():** Lists available languages |

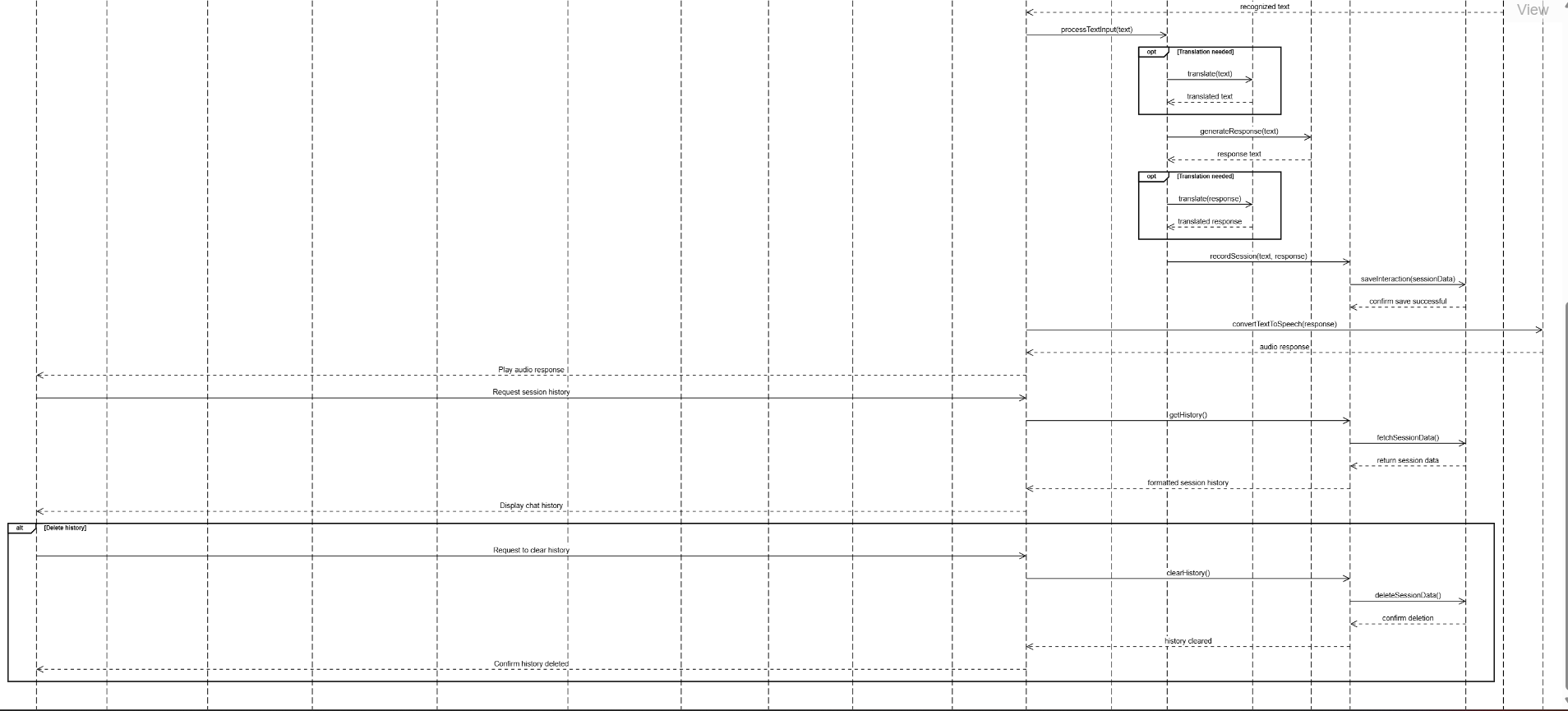
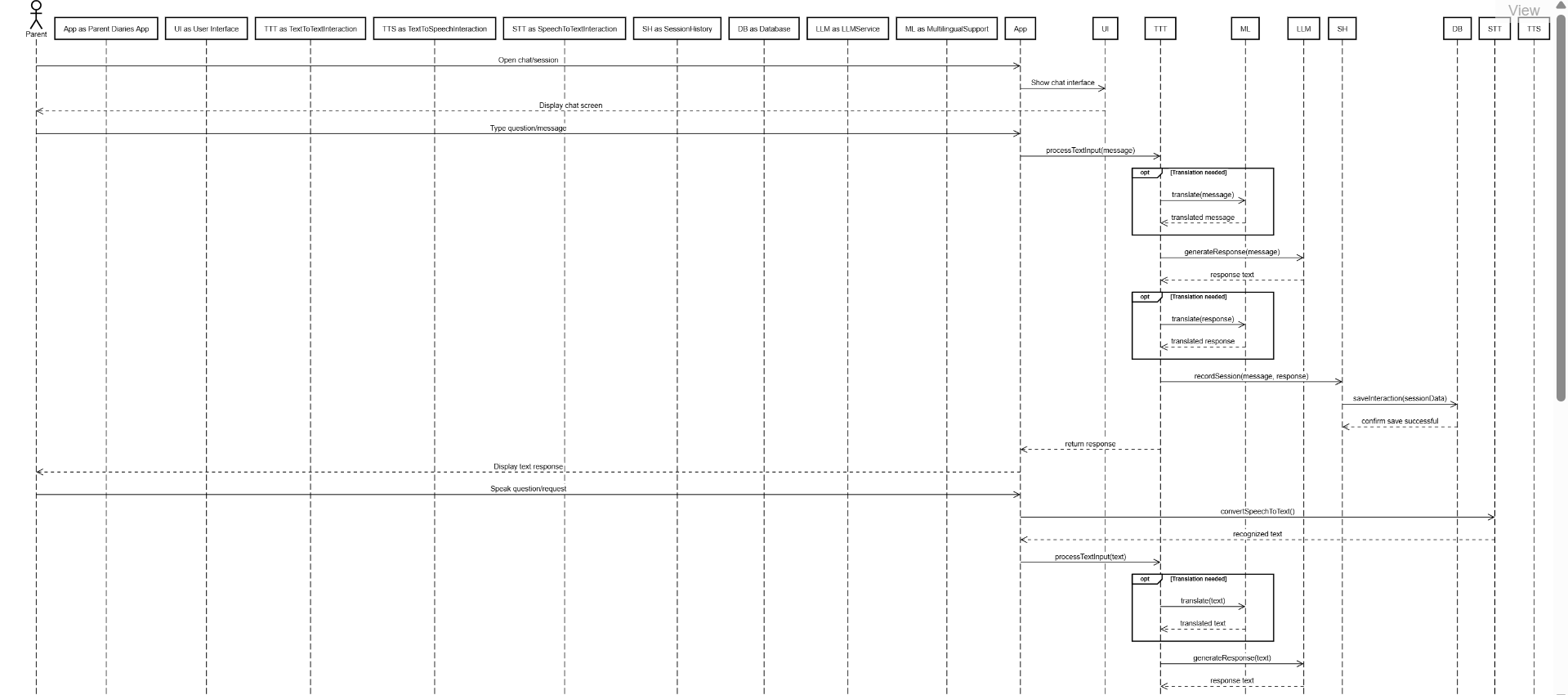
# Sequence Diagram(s)

1. **Parent Discussion Forum**

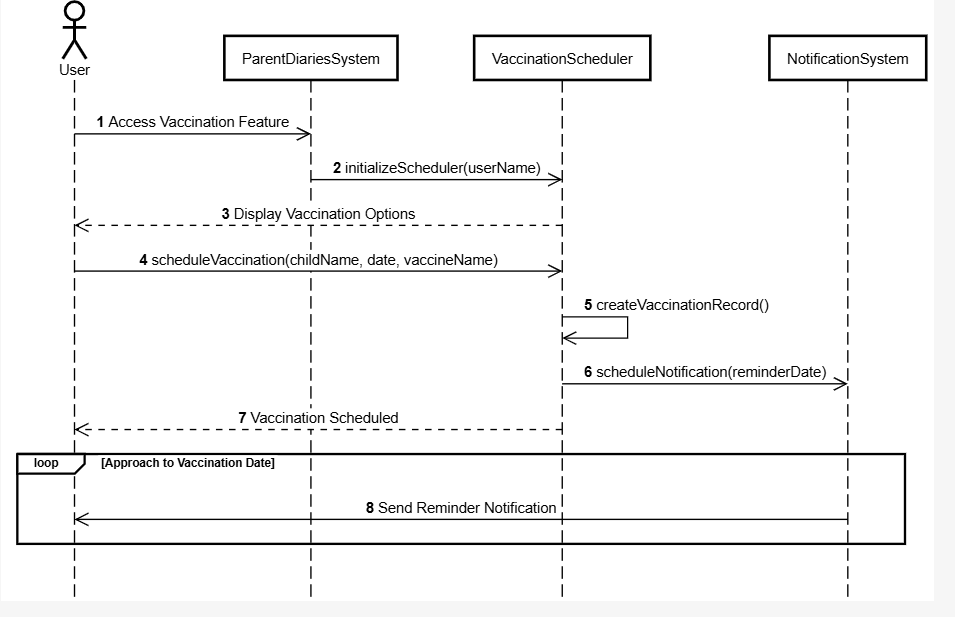




1. **Session History Sequence Diagram**



1. Vaccination Scheduling and Charting Sequence Diagram



# Design Rationale

**Design Rationale for Parenting Application**

## **1. User Registration System**

**Chosen Design**  
 Multi-step registration process with email verification, separate User and AccountManager classes, and optional Google account integration.

**Alternatives Considered**

1. **Single-Step Registration**: Simple form with minimal information collection.
   1. Pros: Reduced friction, higher completion rates, faster onboarding
   2. Cons: Less initial user data, potential for low-quality accounts, limited personalization
2. **Mandatory Social Authentication**: Requiring registration through Google or other social platforms.
   1. Pros: Simplified authentication, reduced password management, verified identities
   2. Cons: Excludes users without social accounts, privacy concerns, dependency on third-party services
3. **Progressive Registration**: Basic registration first, with incremental profile completion.
   1. Pros: Lower initial barrier, better engagement metrics, adaptable to user interest
   2. Cons: Incomplete user profiles, complex state management, potential abandonment

**Rationale for Final Decision**

We implemented a multi-step registration with email verification because:

1. Parenting applications require trustworthy accounts due to sensitive child data
2. Email verification reduces spam accounts and enables critical communication
3. The separate AccountManager class provides specialized account lifecycle management
4. Google account integration offers convenience while maintaining direct registration options
5. This approach balances security concerns with reasonable user experience

## **2. Privacy and Security Architecture**

**Chosen Design**  
Dedicated PrivacyManager class with user-controlled settings, coupled with a comprehensive security approach including data encryption and access controls.

**Alternatives Considered**

1. **Privacy Settings in User Profile**: Embedding privacy controls in User Profile.
   1. Pros: Simplified data model, reduced complexity, direct user association
   2. Cons: Limited granularity, harder to audit, mixed responsibility
2. **Global Privacy Settings**: System-wide privacy settings applied to all users.
   1. Pros: Consistent implementation, easier maintenance, simplified interface
   2. Cons: Limited personalization, potential regulatory issues across regions, reduced user control
3. **Third-Party Privacy Management**: Integration with specialized privacy management services.
   1. Pros: Expert implementation, compliance guarantees, offloaded maintenance
   2. Cons: Increased costs, external dependencies, potential integration challenges

**Rationale for Final Decision**

We implemented a dedicated PrivacyManager class because:

1. Child-related data requires specialized, granular privacy controls
2. Separating privacy logic allows for focused security audits and compliance checks
3. Different regions have varying privacy regulations affecting parenting applications
4. Users need transparent control over sensitive information sharing
5. The dedicated class provides a foundation for future privacy enhancements and compliance features

The system also implements comprehensive security measures including:

* End-to-end encryption for all health-related data
* Role-based access controls for data sharing
* Regular security audits and penetration testing
* Secure data storage with minimal retention periods
* Transparent data usage policies with user consent management

## **3. Core Features Architecture**

**Chosen Design**  
Modular feature classes (VaccinationScheduler, PhysicalGrowthTracker, MilestoneTracker) connected to the central User entity with a shared NotificationSystem.

**Alternatives Considered**

1. **Feature-as-Service Approach**: Implementing each feature as an independent microservice.
   1. Pros: Maximum isolation, independent scaling, specialized teams per feature
   2. Cons: More complex integration, increased operational overhead, potential consistency issues
2. **Monolithic Feature Manager**: Single class managing all parenting tracking features.
   1. Pros: Simplified implementation, centralized business logic, easier data sharing
   2. Cons: Limited maintainability, difficult to extend, harder to test individual components
3. **Event-Based Feature Communication**: Features communicating through an event bus rather than direct relationships.
   1. Pros: Lower coupling, easier to add new features, more resilient architecture
   2. Cons: More complex to implement, potential for missed events, harder to debug

**Rationale for Final Decision**

We chose the modular feature class approach with direct user relationships because:

1. It balances modularity with implementation simplicity, appropriate for a mobile application
2. Each feature has distinct tracking needs but benefits from shared user context
3. The centralized NotificationSystem provides consistent alerting while maintaining feature independence
4. Direct relationships make data flow more transparent and easier to reason about
5. This approach supports incremental development of features without major architectural changes

## **4. AI Interaction Models**

**Chosen Design**  
 Multi-modal interaction system with specialized classes for different communication patterns (TextToText, TextToSpeech, SpeechToText, SpeechToSpeech) supported by external AI services.

**Alternatives Considered**

1. **Single Interaction Controller**: Unified interaction class handling all input/output modalities.
   1. Pros: Simplified interface, consistent handling, easier integration
   2. Cons: Complex internal logic, difficult to extend, potential performance bottlenecks
2. **Client-Side Voice Processing**: Moving speech processing to the mobile device.
   1. Pros: Works offline, reduced server costs, faster response for simple queries
   2. Cons: Inconsistent quality across devices, larger app footprint, limited AI capabilities
3. **Third-Party Conversational Platform**: Using a complete solution like Dialogflow or Lex.
   1. Pros: Faster time-to-market, specialized conversation design tools, built-in analytics
   2. Cons: Less control over experience, potential vendor lock-in, recurring costs

**Rationale for Final Decision**

We implemented separate interaction classes with external service integration because:

1. Parents need flexibility in how they interact while caring for children (hands-free voice or quiet text)
2. Different interaction modalities have distinct technical requirements and optimizations
3. The separate classes allow for specialized error handling and recovery strategies
4. External AI services provide best-in-class capabilities without rebuilding specialized ML models
5. This approach allows us to focus development efforts on parenting-specific features rather than general AI capabilities

## **5. Subscription and Monetization Model**

**Chosen Design**  
Tiered subscription model with free basic features and premium subscription unlocking advanced functionality, integrated with external PaymentGateway.

**Alternatives Considered**

1. **One-Time Purchase**: Charging a single up-front fee for the application.
   1. Pros: Simpler revenue model, no recurring billing complexity, clear value proposition
   2. Cons: Limited ongoing revenue, higher initial barrier to adoption, difficult sustainability model
2. **Feature-Based Microtransactions**: Individual payment for specific premium features.
   1. Pros: Lower entry barrier, customized user experience, potentially higher revenue
   2. Cons: User friction, complex implementation, potential for negative user perception
3. **Ad-Supported Model**: Free application with revenue from advertisements.
   1. Pros: No cost barrier, potential broad adoption, passive revenue generation
   2. Cons: Privacy concerns with child-related data, potential regulatory issues, degraded user experience

**Rationale for Final Decision**

We implemented the tiered subscription model because:

1. It provides sustainable ongoing revenue to support continuous development and AI service costs
2. The free tier allows users to experience core value before committing financially
3. Premium features can justify the subscription cost through enhanced AI capabilities
4. Parents are accustomed to subscription models for ongoing services
5. This model avoids advertising-related privacy concerns in a sensitive parenting application

**6. UI Interaction Model**

**Chosen Design**

A **native mobile application** with both **text and speech interaction** capabilities.

**Alternatives Considered**

**Web Interface:** A responsive web app accessible via browsers on any device.

* *Pros:* Cross-platform compatibility, faster updates, no need for installation
* *Cons:* Limited offline capabilities, less seamless integration with mobile hardware

**Text-Only Interface:** Focusing solely on text-based inputs.

* *Pros:* Simpler to implement, wider accessibility across diverse devices
* *Cons:* Less natural interaction, potentially less inclusive for users with accessibility needs

**Voice-First Design:** Optimized primarily for voice interaction.

* *Pros:* Natural and hands-free interaction, more accessible for users with disabilities
* *Cons:* Privacy concerns in public spaces, speech recognition accuracy issues, more complex implementation

**Rationale for Final Decision**

We chose to develop a native mobile app with multimodal (text and speech) interaction because:

* Mobile apps offer **better performance and deeper OS integration**, enabling smoother interactions
* They provide **offline functionality**, which is valuable for a mood tracking app used throughout the day
* A native app can leverage **device features** like notifications, sensors, and local storage more effectively
* Text and speech modes ensure **inclusive and flexible** interaction, letting users choose what’s comfortable
* The app can be continuously improved with **platform-specific enhancements and user feedback**

This design decision ensures an engaging, responsive, and accessible experience, forming a strong foundation for future feature development and potential cross-platform expansion.