Readmaster.ai - Technical Documentation

Table of Contents

1. System Overview
2. UML Diagrams
    - Database Entity Relationship Diagram (ERD)
    - System Architecture Diagram
    - User Role Activity Diagram
    - Assessment Process Sequence Diagram
    - Class Diagram - Core Domain Models
3. System Capabilities
    - Frontend Application
    - Backend Service (RESTful API)
4. Core Components & Design
    - Clean Architecture Structure
    - Design Patterns Implementation
    - Database Layer (SQLAlchemy)
    - Authentication & Authorization
    - API Endpoints with Request/Response Schemas
    - File Handling Strategy
5. Technical Requirements
    - Database Schema (PostgreSQL)
6. Performance Considerations
7. Integration Points
8. Error Handling and Logging Strategy
9. Deployment Architecture

1. System Overview

Readmaster.ai is a web-based reading assessment and development platform that uses artificial intelligence to analyze and help improve students' reading performance. The system consists of a frontend web application and a backend service that handles AI processing and database operations.

Key Features

- AI-powered reading fluency analysis
- Pronunciation assessment
- Reading comprehension evaluation
- Multi-language support

- Real-time progress tracking
- Role-based access control (Students, Parents, Teachers, Admins)

2. UML Diagrams

2.1. Database Entity Relationship Diagram (ERD)

```
erDiagram
  Users {
    UUID user_id PK
    VARCHAR email UK
    VARCHAR password_hash
    VARCHAR first_name
    VARCHAR last_name
    ENUM role
    TIMESTAMPTZ created_at
    TIMESTAMPTZ updated_at
    VARCHAR preferred_language
  }

  Classes {
    UUID class_id PK
    VARCHAR class_name
    VARCHAR grade_level
    UUID created_by_teacher_id FK
    TIMESTAMPTZ created_at
    TIMESTAMPTZ updated_at
  }

  Readings {
    UUID reading_id PK
    VARCHAR title
    TEXT content_text
    VARCHAR content_image_url
    VARCHAR age_category
    ENUM difficulty_level
    VARCHAR language
    VARCHAR genre
    UUID added_by_admin_id FK
    TIMESTAMPTZ created_at
    TIMESTAMPTZ updated_at
}

Assessments {
    UUID assessment_id PK
    UUID student_id FK
    UUID reading_id FK
    UUID assigned_by_teacher_id FK
    VARCHAR audio_file_url
    INTEGER audio_duration_seconds
    ENUM status
    TIMESTAMPTZ assessment_date
    TEXT ai_raw_speech_to_text
    TIMESTAMPTZ updated_at
}

AssessmentResults {
    UUID result_id PK
    UUID assessment_id FK
    JSONB analysis_data
    FLOAT comprehension_score
    TIMESTAMPTZ created_at
}

QuizQuestions {
    UUID question_id PK
    UUID reading_id FK
    TEXT question_text
    JSONB options
    VARCHAR correct_option_id
    VARCHAR language
    UUID added_by_admin_id FK
    TIMESTAMPTZ created_at
}

StudentQuizAnswers {
    UUID answer_id PK
    UUID assessment_id FK
    UUID question_id FK
    UUID student_id FK
    VARCHAR selected_option_id
    BOOLEAN is_correct
    TIMESTAMPTZ answered_at
}

Students_Classes {
    UUID student_id FK
    UUID class_id FK
    TIMESTAMPTZ joined_at
}

Parents_Students {
    UUID parent_id FK
    UUID student_id FK
    VARCHAR relationship_type
    TIMESTAMPTZ linked_at
}

Teachers_Classes {
    UUID teacher_id FK
    UUID class_id FK
    TIMESTAMPTZ assigned_at
}

ProgressTracking {
    UUID progress_id PK
    UUID student_id FK
    VARCHAR metric_type
    FLOAT value
    DATE period_start_date
    DATE period_end_date
    TIMESTAMPTZ last_calculated_at
}

Notifications {
    UUID notification_id PK
    UUID user_id FK
    ENUM type
    TEXT message
    UUID related_entity_id
    BOOLEAN is_read
    TIMESTAMPTZ created_at
  }

  Users ||--o{ Classes : "creates"
  Users ||--o{ Students_Classes : "enrolls"
  Users ||--o{ Parents_Students : "links"
  Users ||--o{ Teachers_Classes : "teaches"
  Users ||--o{ Assessments : "takes/assigns"
  Users ||--o{ Readings : "adds"
  Users ||--o{ QuizQuestions : "creates"
  Users ||--o{ StudentQuizAnswers : "answers"
  Users ||--o{ ProgressTracking : "tracks"
  Users ||--o{ Notifications : "receives"

  Classes ||--o{ Students_Classes : "contains"
  Classes ||--o{ Teachers_Classes : "assigned_to"

  Readings ||--o{ Assessments : "assessed_in"
  Readings ||--o{ QuizQuestions : "has"

  Assessments ||--|| AssessmentResults : "produces"
  Assessments ||--o{ StudentQuizAnswers : "includes"

  QuizQuestions ||--o{ StudentQuizAnswers : "answered_in"
```

2.2. System Architecture Diagram

```
graph TB
  subgraph "Client Layer"
    A[React Frontend<br/>TypeScript + i18n]
    B[Mobile Browser]
    C[Desktop Browser]
  end

  subgraph "CDN & Static Assets"
    D[CDN<br/>Static Files]
```

```
end

subgraph "API Gateway"
   E[Load Balancer<br/>HTTPS/TLS]
end

subgraph "Application Layer"
   F[Backend API<br/>RESTful Service]
   G[Authentication<br/>JWT Service]
   H[WebSocket Server<br/>Real-time Notifications]
end

subgraph "Processing Layer"
   I[AI Processing Service<br/>Async Workers]
   J[Queue System<br/>FastAPI BackgroundTasks]
end

subgraph "External Services"
   K[Google AI APIs<br/>Speech-to-Text<br/>Gemini Models]
end

subgraph "Data Layer"
   L[PostgreSQL<br/>Primary Database]
   M[Redis Cache<br/>Session & Data Cache]
   N[Cloud Storage<br/>Audio Files]
end

subgraph "Monitoring & Logging"
   O[Logging Service<br/>ELK Stack]
   P[Monitoring<br/>Metrics & Alerts]
end

A --> D
B --> E
C --> E
D --> E
E --> F
E --> G
E --> H
```

```
    F --> M
    F --> L
    F --> J
    F --> N

    G --> M
    G --> L

    H --> M

    J --> I
    I --> K
    I --> L
    I --> N

    F --> O
    I --> O
    F --> P
    I --> P
```

2.3. User Role Activity Diagram

```
graph TD
    A[User Login] --> B{Role Check}

    B -->|Student| C[Student Dashboard]
    B -->|Parent| D[Parent Dashboard]
    B -->|Teacher| E[Teacher Dashboard]
    B -->|Admin| F[Admin Dashboard]

    C --> C1[View Assignments]
    C --> C2[Take Reading Assessment]
    C --> C3[View Progress]

    C2 --> C21[Select Reading]
    C21 --> C22[Record Audio]
    C22 --> C23[Answer Quiz]
    C23 --> C24[Submit Assessment]
    C24 --> C25[View Results]

    D --> D1[View Children's Progress]
    D --> D2[View Assessment Results]
    D --> D3[Receive Notifications]

    E --> E1[Manage Classes]
    E --> E2[Assign Readings]
    E --> E3[Monitor Student Progress]
    E --> E4[View Reports]

    E1 --> E11[Create Class]
    E1 --> E12[Add Students]
    E2 --> E21[Select Reading Material]
    E2 --> E22[Assign to Student/Class]

    F --> F1[Manage Users]
    F --> F2[Manage Reading Materials]
    F --> F3[System Configuration]
    F --> F4[View System Analytics]

    F2 --> F21[Add New Reading]
    F2 --> F22[Create Quiz Questions]
    F2 --> F23[Manage Content Library]
```

2.4. Assessment Process Sequence Diagram

```
sequenceDiagram
    participant S as Student
    participant FE as Frontend
    participant API as Backend API
    participant DB as Database
    participant Q as Queue System
    participant AI as AI Service
    participant CS as Cloud Storage
    participant WS as WebSocket

    S->>FE: Select Reading
    FE->>API: GET /readings/{id}
```

```
API->>DB: Fetch reading content
DB-->>API: Reading data
API-->>FE: Reading content
FE-->>S: Display reading

S->>FE: Start Assessment
FE->>API: POST /assessments
API->>DB: Create assessment record
DB-->>API: Assessment ID
API-->>FE: Assessment created

S->>FE: Record Audio
FE->>FE: Audio recording
S->>FE: Submit Audio
FE->>API: POST /assessments/{id}/audio
API->>CS: Upload audio file
CS-->>API: File URL
API->>DB: Update assessment with audio URL
API->>Q: Queue AI processing job
API-->>FE: Audio uploaded

S->>FE: Answer Quiz Questions
FE->>API: POST /assessments/{id}/quiz-answers
API->>DB: Store quiz answers
API-->>FE: Quiz submitted

Q->>AI: Process audio analysis
AI->>CS: Download audio file
AI->>AI: Speech-to-text conversion
AI->>AI: Fluency analysis
AI->>AI: Pronunciation assessment
AI->>DB: Store analysis results
AI->>WS: Notify processing complete

WS->>FE: Real-time notification
FE->>API: GET /assessments/{id}
API->>DB: Fetch complete results
DB-->>API: Assessment results
API-->>FE: Complete analysis
```

```
    FE-->>S: Display results
```

2.5. Class Diagram - Core Domain Models

```
classDiagram
    class User {
        +UUID userId
        +String email
        +String passwordHash
        +String firstName
        +String lastName
        +UserRole role
        +DateTime createdAt
        +DateTime updatedAt
        +String preferredLanguage
        +login()
        +updateProfile()
        +changePassword()
    }

    class Student {
        +List~Assessment~ assessments
        +List~Class~ classes
        +List~Parent~ parents
        +ProgressTracking progress
        +takeAssessment(Reading)
        +viewProgress()
        +submitQuizAnswers()
    }

    class Teacher {
        +List~Class~ classes
        +createClass()
        +assignReading(Student, Reading)
        +viewStudentProgress(Student)
        +manageStudents()
    }

    class Parent {
```

```
    +List~Student~ children
    +viewChildProgress(Student)
    +receiveNotifications()
}

class Admin {
    +manageUsers()
    +manageReadings()
    +viewSystemAnalytics()
}

class Reading {
    +UUID readingId
    +String title
    +String contentText
    +String contentImageUrl
    +String ageCategory
    +DifficultyLevel difficulty
    +String language
    +String genre
    +List~QuizQuestion~ questions
    +validateContent()
    +generateQuiz()
}

class Assessment {
    +UUID assessmentId
    +UUID studentId
    +UUID readingId
    +String audioFileUrl
    +Integer audioDuration
    +AssessmentStatus status
    +DateTime assessmentDate
    +String aiRawSpeechToText
    +AssessmentResult result
    +List~QuizAnswer~ quizAnswers
    +processAudio()
    +calculateScores()
}
```

```
class AssessmentResult {
   +UUID resultId
   +UUID assessmentId
   +Object analysisData
   +Float comprehensionScore
   +DateTime createdAt
   +generateReport()
   +calculateMetrics()
}

class QuizQuestion {
   +UUID questionId
   +UUID readingId
   +String questionText
   +Object options
   +String correctOptionId
   +String language
   +validateAnswer(String)
}

class Class {
   +UUID classId
   +String className
   +String gradeLevel
   +UUID createdByTeacherId
   +List~Student~ students
   +List~Teacher~ teachers
   +addStudent(Student)
   +removeStudent(Student)
   +assignTeacher(Teacher)
}

User <|-- Student
User <|-- Teacher
User <|-- Parent
User <|-- Admin

Student "1" -- "*" Assessment : takes
Student "*" -- "*" Class : enrolls
Student "*" -- "*" Parent : linked_to

Teacher "1" -- "*" Class : manages
Teacher "1" -- "*" Assessment : assigns

Reading "1" -- "*" Assessment : assessed_in
Reading "1" -- "*" QuizQuestion : has

Assessment "1" -- "1" AssessmentResult : produces
Assessment "1" -- "*" QuizAnswer : includes

QuizQuestion "1" -- "*" QuizAnswer : answered
```

3. System Capabilities

3.1. Frontend Application

- **Technology Stack:** React, TypeScript
- **Internationalization:** i18n library implementation

Key Features:

- **Student Panel:**
  - View assigned readings
  - Select new reading materials
  - View historical assessments and progress
  - Audio recording interface for reading assessment
  - Quiz system for comprehension testing
  - Detailed performance metrics and feedback
- **Parent Panel:**
  - Monitor connected students' progress
  - View assessment results and performance trends
  - Access detailed reading analytics
- **Teacher Panel:**
  - Manage classes and student enrollments
  - Assign readings to students
  - Track student progress and assessment results
  - Generate performance reports
- **Settings Management:**

    - Profile information management
    - Language preferences
    - Notification settings

3.2. Backend Service (RESTful API)

- **Technology Stack:**
    - Framework: FastAPI (Python)
    - Database ORM: SQLAlchemy with Alembic for migrations
    - Validation: Pydantic models
    - Authentication: JWT with refresh tokens
    - Testing: pytest with pytest-asyncio
    - Dependency Injection: FastAPI's built-in DI system
    - Caching: Redis
    - Background Tasks: FastAPI BackgroundTasks + Celery for complex workflows
- **Architecture Pattern:** Clean Architecture
- **Core Responsibilities:**
    - Database operations management
    - Audio file processing for reading analysis
    - AI model integration for:
        - Reading fluency analysis
        - Pronunciation assessment
        - Comprehension evaluation (based on original text, recorded audio, and quiz responses)
    - Progress tracking

4. Core Components & Design

4.1. Clean Architecture Structure

The backend will follow the principles of Clean Architecture to ensure separation of concerns, testability, and maintainability.

```
src/
├── domain/              # Enterprise Business Rules (No frameworks)
│   ├── entities/          # Core domain objects (e.g., User, Assessment)
│   ├── value_objects/       # Value objects (e.g., Email, Role, Permissions)
│   ├── repositories/       # Abstract repository interfaces
│   └── services/          # Domain-specific services
├── application/          # Application Business Rules
│   ├── use_cases/         # Implements specific application actions
│   ├── interfaces/        # Application-level interfaces (e.g., AI service)
│   └── dto/              # Data Transfer Objects for use cases
├── infrastructure/         # Frameworks & Drivers (External dependencies)
│   ├── database/          # SQLAlchemy models, repositories implementation
│   ├── external_services/  # API clients for external services (e.g., Google AI)
│   ├── file_storage/       # Cloud storage service implementation
│   └── ai/               # AI service implementation details
├── presentation/          # Interface Adapters
│   ├── api/              # FastAPI routers and endpoints
│   ├── schemas/          # Pydantic request/response models
│   └── dependencies/      # FastAPI dependency injection setup
└── shared/               # Shared utilities
    ├── exceptions/       # Custom application exceptions
    ├── constants/        # Application-wide constants
    └── utils/            # General utility functions
```

4.2. Design Patterns Implementation

- **Repository Pattern:** Abstract the data layer, allowing domain and application layers to remain independent of the database technology.

```
# domain/repositories/user_repository.py
from abc import ABC, abstractmethod
from typing import Optional
from domain.entities.user import User

class UserRepository(ABC):
    @abstractmethod
    async def get_by_id(self, user_id: str) -> Optional[User]: pass

    @abstractmethod
    async def get_by_email(self, email: str) -> Optional[User]: pass

    @abstractmethod
    async def create(self, user: User) -> User: pass
```

- **Service Layer Pattern (Use Cases):** Encapsulate application-specific business logic. Each use case orchestrates the flow of data between entities and repositories.

```
# application/use_cases/create_assessment_use_case.py
class CreateAssessmentUseCase:
    def __init__(self, assessment_repo: AssessmentRepository):
        self.assessment_repo = assessment_repo

    async def execute(self, request_dto: CreateAssessmentDTO) -> Assessment:
        # Business logic to create an assessment
        pass
```

- **Factory Pattern:** Decouple the creation of complex objects, such as the AI service client.
  ```
  # infrastructure/ai/ai_service_factory.py
  class AIServiceFactory:
      @staticmethod
      def create_service() -> AIAnalysisInterface:
          return GeminiAnalysisService(api_key=settings.GEMINI_API_KEY)
  ```
- **Observer Pattern:** For handling notifications. A central service will notify multiple observers (e.g., email service, WebSocket service) when an event occurs, like an assessment being completed.
  ```
  # domain/services/notification_service.py
  class NotificationService:
      def __init__(self):
          self._observers = []

      def subscribe(self, observer): self._observers.append(observer)

      async def notify(self, event, data):
          for observer in self._observers:
              await observer.update(event, data)
  ```
- **Dependency Injection:** FastAPI's built-in dependency injection system will be used extensively to manage dependencies like database sessions and repositories.

4.3. Database Layer (SQLAlchemy)

- **Database Configuration:**
  ```
  # infrastructure/database/config.py
  from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
  from sqlalchemy.orm import sessionmaker
  ```

```
DATABASE_URL = "postgresql+asyncpg://user:password@localhost/readmaster"
engine = create_async_engine(DATABASE_URL, echo=True)
AsyncSessionLocal = sessionmaker(engine, class_=AsyncSession,
expire_on_commit=False)

async def get_db():
    async with AsyncSessionLocal() as session:
        yield session
```

- **Entity Models & Migrations:** SQLAlchemy will be used for ORM, with declarative_base for models. Alembic will manage database schema migrations.
  ```
  # infrastructure/database/models.py
  # ... (SQLAlchemy models corresponding to the ERD) ...
  ```

- **Repository Implementation:**
  ```
  # infrastructure/database/repositories/user_repository_impl.py
  from sqlalchemy.ext.asyncio import AsyncSession
  from sqlalchemy import select
  from domain.entities.user import User
  # ...
  class UserRepositoryImpl(UserRepository):
      def __init__(self, session: AsyncSession):
          self.session = session

      async def get_by_email(self, email: str) -> Optional[User]:
          stmt = select(UserModel).where(UserModel.email == email)
          # ... execution and conversion logic ...
  ```

4.4. Authentication & Authorization

- **JWT Implementation:**
  - **Access Token:** Short-lived (e.g., 15-30 minutes), used for authenticating API requests.
  - **Refresh Token:** Long-lived (e.g., 7 days), stored securely on the client (e.g., HttpOnly cookie). Used to obtain a new access token without requiring the user to log in again. The refresh token's lifetime is extended upon use, effectively creating a sliding session.
  - **Password Security:** Passwords will be hashed using bcrypt via the passlib library.

- **Role-based Permission Matrix:**
```
# domain/value_objects/permissions.py
from enum import Enum
class Permission(Enum):
    CREATE_USER = "create_user"
    CREATE_READING = "create_reading"
    ASSIGN_ASSESSMENT = "assign_assessment"
    # ... more permissions

ROLE_PERMISSIONS = {
    "student": {Permission.VIEW_OWN_PROGRESS},
    "teacher": {Permission.ASSIGN_ASSESSMENT,
Permission.VIEW_STUDENT_PROGRESS},
    "admin": {p for p in Permission}
}
```

4.5. API Endpoints with Request/Response Schemas

- **API Versioning:** URL-based versioning (/api/v1/...) will be used for clarity.
- **Pydantic Schemas:** All API endpoints will use Pydantic for request/response validation.
```
# presentation/schemas/user_schemas.py
class UserCreateRequest(BaseModel):
    email: EmailStr
    password: str
    # ...
class UserResponse(BaseModel):
    user_id: UUID
    email: EmailStr
    class Config: from_attributes = True
```
- **Pagination:** A standardized pagination model will be used for all list endpoints.
```
# presentation/schemas/pagination.py
class PaginatedResponse(BaseModel, Generic[T]):
    items: List[T]
    total: int
    page: int
    size: int
```

4.6. File Handling Strategy

- **Upload Strategy:** Client will request a **pre-signed URL** from the backend to upload audio files directly to a cloud storage bucket (e.g., GCS, S3). This avoids proxying large files through the backend service.
- **File Processing:**
  - **Validation:** Audio format (mp3, wav, m4a), duration (e.g., max 10 minutes), and size will be validated.
  - **Standardization:** All uploaded audio will be converted to a consistent, lossless format like FLAC for reliable AI processing.
  - **Metadata:** Duration, sample rate, and channels will be extracted and stored.
- Storage Organization: Files will be organized logically in the storage bucket.
  readmaster-audio/{environment}/{year}/{month}/{day}/{assessment_id}.flac

5. Technical Requirements

5.1. Database Schema (PostgreSQL)

- **Users Table:**
  ```
  CREATE TYPE user_role_enum AS ENUM ('student', 'parent', 'teacher', 'admin');
  CREATE TABLE Users ( ... );
  ```

- **Assessments Table:**
  ```
  CREATE TYPE assessment_status_enum AS ENUM ('pending_audio', 'processing',
  'completed', 'error');
  CREATE TABLE Assessments ( ... );
  CREATE INDEX idx_assessment_student_date ON Assessments (student_id,
  assessment_date);
  CREATE INDEX idx_assessment_status ON Assessments (status);
  ```

  ... *(All other CREATE TABLE statements as previously defined, with added indexes for foreign keys and frequently filtered columns)*

6. Performance Considerations

- **Asynchronous Operations:** The entire backend will be asynchronous, leveraging FastAPI's async/await support for all I/O-bound operations (database, external APIs).
- **Background Jobs:** AI analysis, which is time-consuming, will be executed as a background task using Celery with RabbitMQ/Redis as a message broker. This ensures API requests return immediately. If analysis fails, the task can be retried, and the assessment status will be updated to 'error'.

- **Caching Strategy:** Redis will be used for caching:
  - **User Sessions/Permissions:** To reduce database lookups on authenticated requests. (TTL: 30 mins)
  - **Reading Materials:** Content of Readings and QuizQuestions tables. (TTL: 24 hours, with cache invalidation on update).
- **Database Optimization:**
  - **Indexing:** All foreign keys and columns frequently used in WHERE clauses, JOINs, or ORDER BY clauses will be indexed.
  - **Connection Pooling:** SQLAlchemy's connection pool will be configured to manage database connections efficiently.