Team Name: The Stellars

Project Title: Management Information System (MIS)

Software Design Document

Name(s): Aditya, Nandini, Aryan, Prakhar, Abhishek, Harsh

Date: September 17, 2025

TABLE OF CONTENTS

1. INTRODUCTION
   1.1 Purpose
   1.2 Scope
   1.3 Overview
   1.4 Reference Material
   1.5 Definitions and Acronyms
2. SYSTEM OVERVIEW
3. SYSTEM ARCHITECTURE
   3.1 Architectural Design
   3.2 Decomposition Description
   3.3 Design Rationale
4. DATA DESIGN
   4.1 Data Description
   4.2 Data Dictionary
5. COMPONENT DESIGN
6. HUMAN INTERFACE DESIGN
   6.1 Overview of User Interface
   6.2 Screen Images
   6.3 Screen Objects and Actions
7. REQUIREMENTS MATRIX
8. APPENDICES

1. INTRODUCTION

1.1 Purpose

This Software Design Document (SDD) describes the architecture and system design of the Management Information System (MIS). It translates the software requirements detailed in the SRS into a representation of software components, interfaces, and data necessary for the implementation phase. This document is the primary reference for the development team to write and implement the code. The intended audience includes the project developers, architects, and the project supervisor.

1.2 Scope

The scope of this project is to develop a centralized, real-time web platform to replace the current fragmented and manual processes for Wushu athlete management in India. The project's primary goal is to create a unified digital ecosystem that connects athletes, coaches, and administrators.

In-Scope Features:

- **User & Profile Management:** Secure registration, authentication, and profile management for all user roles.
- **Performance Tracking System:** A system for logging training data and competition results, with automated performance charting for athletes.
- **Online Payment Processing:** Functionality for athletes to pay admission fees online and for administrators to track these payments.
- **Role-Based Dashboards:** Customized dashboards for Athletes, Coaches, and Administrators to provide relevant, at-a-glance statistics and insights.

Out-of-Scope Features:

- Championship Management.
- A dedicated native mobile application (the system will be a mobile-responsive website).
- Integration with wearable IoT devices or sensors.

1.3 Overview

This document provides a detailed technical blueprint for the MIS project. It begins with an overview of the system, followed by a detailed description of the system architecture, including design rationale. It then covers the data design with a data dictionary, the internal logic of key components, the human interface design, and finally, a requirements traceability matrix linking design components back to the SRS.

1.4 Reference Material

This SDD is based on the information provided in the following source documents:

1. Business Requirements Document: Management Information System (MIS), Version 3.1
2. Functional Requirements Document: Management Information System (MIS), Version 1.1
3. Non-Functional Requirements Document: Management Information System (MIS), Version 1.1
4. Software Requirements Specification for Management Information System (MIS), Version 1.0

1.5 Definitions and Acronyms

- **API:** Application Programming Interface. A set of rules and protocols for building and interacting with software applications.
- **BRD:** Business Requirements Document.
- **FOSS:** Free and Open-Source Software. Software that is free to use and whose source code is publicly accessible.
- **FRD:** Functional Requirements Document.
- **MIS:** Management Information System. The official name of this project.
- **NFRD:** Non-Functional Requirements Document.
- **OTP:** One-Time Password. A temporary, secure code used for user verification.
- **RBAC:** Role-Based Access Control. A security paradigm that restricts system access based on a user's role.
- **SRS:** Software Requirements Specification.

2. SYSTEM OVERVIEW

The MIS is a new, self-contained web platform designed from the ground up to serve as the central information hub for a Wushu training center. Its core function is to digitize and streamline the entire lifecycle of an athlete, from registration and fee payment to daily performance tracking and long-term progress analysis. The system will provide distinct interfaces and functionalities for its three user classes:

Athletes, who manage their profiles and track their journey; **Coaches**, who log data for their assigned athletes; and **Administrators**, who oversee the training center's operations, manage user rosters, and confirm payments. By replacing inefficient paper-based methods, the MIS aims to improve data accuracy, provide valuable performance insights, and reduce administrative overhead.

3. SYSTEM ARCHITECTURE

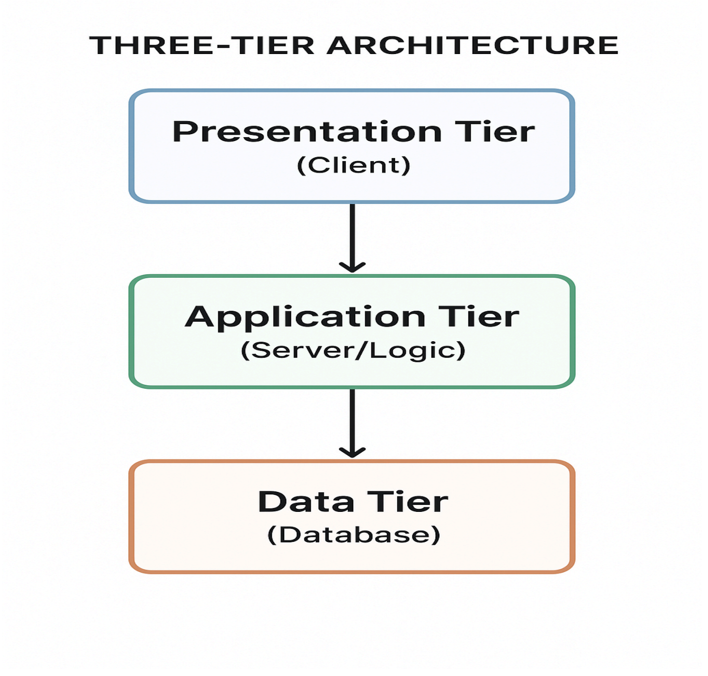
3.1 Architectural Design

The MIS will be built using a **Three-Tier Architecture**, a well-established software architecture pattern that organizes an application into three logical and physical computing tiers. This separation of concerns enhances scalability, maintainability, and security.

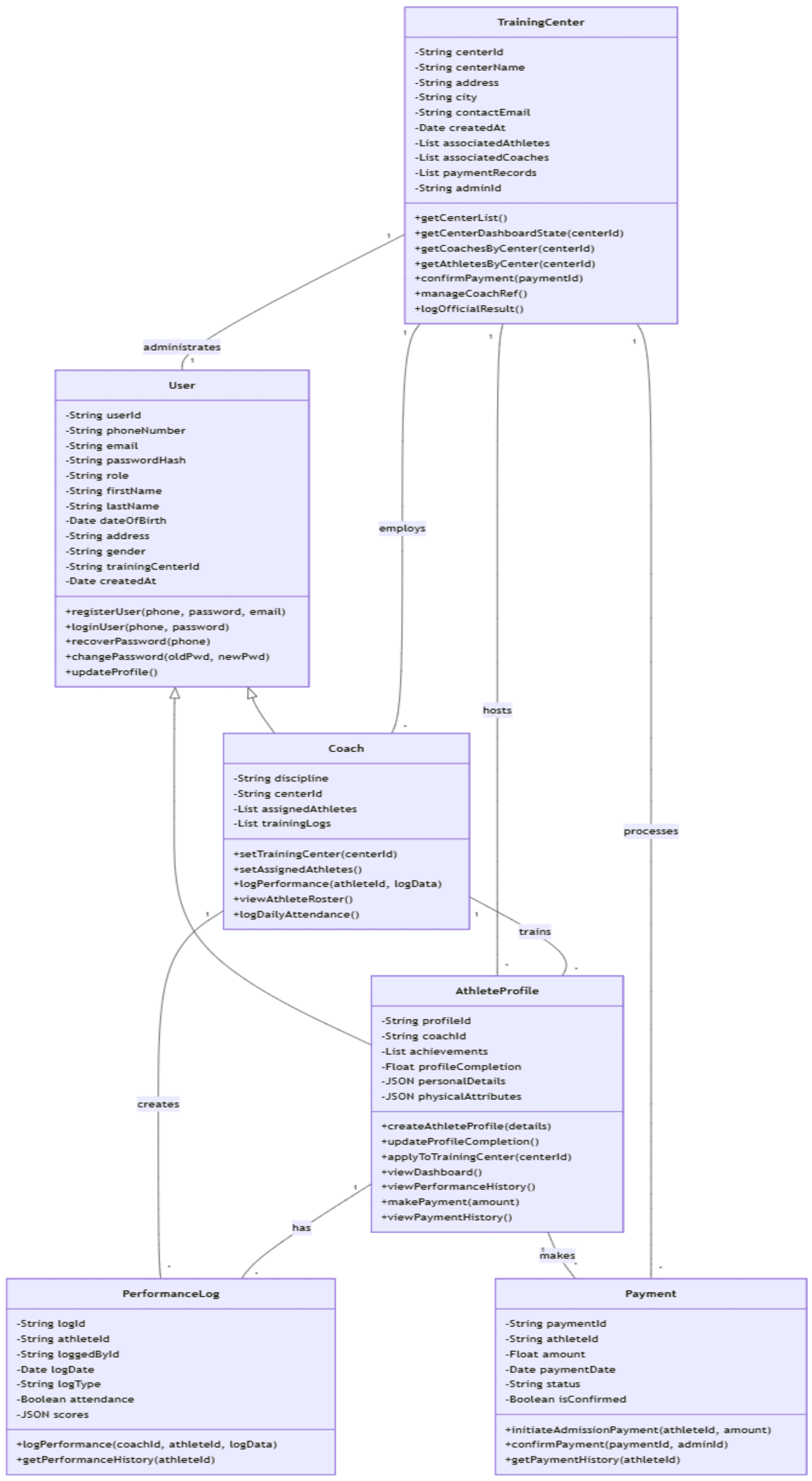
The three tiers are:

1. **Presentation Tier (Client):** The user-facing layer of the application, rendered in the user's web browser. It is responsible for displaying the user interface and capturing user input. It communicates exclusively with the Application Tier.
2. **Application Tier (Server/Logic):** The middle tier containing the business logic that processes requests from the Presentation Tier. It enforces business rules, performs calculations, and acts as a secure intermediary between the user and the database.
3. **Data Tier (Database):** The back-end tier where all application data is stored and managed. It is responsible for data persistence, retrieval, and integrity, and is only accessible via the Application Tier.

The following diagram illustrates this architecture:



3.2 Decomposition Description



3.3 Design Rationale

The provided UML Class Diagram employs a **Class Inheritance model** with an abstract User class and specialized subclasses (Athlete, Coach, Administrator). This specific design was deliberately chosen over a simpler single-table model to achieve several key architectural goals for the project.

The reasoning for this structure is as follows:

- **Adherence to Object-Oriented Principles:** This design is a direct and clean translation of object-oriented principles into a data model. The inheritance structure (Athlete is a User) provides a clear and intuitive mapping between the design and the source code, which is ideal for modern, component-based development.
- **High Data Integrity and Normalization:** This is the primary technical advantage. By separating role-specific attributes into their own classes (e.g., achievements and physicalAttributes exist only for Athlete), the design avoids irrelevant NULL values in a single, large user table. This ensures a higher degree of data integrity, as it's structurally impossible for a Coach to have an achievement record.
- **Extensibility and Future-Proofing:** The inheritance model is highly extensible. If a new user role, such as a "Judge" or "Parent," is needed in the future, a new class can be created to inherit from User without requiring any changes to the existing Athlete or Coach structures. This makes the system easier to scale and adapt to new requirements over time.
- **Clear Separation of Concerns:** The design logically separates common attributes and functions (like loginUser and passwordHash in the User class) from specialized ones (like logPerformance in the Coach class). This separation makes the overall system easier to understand, maintain, and test.

4. DATA DESIGN

4.1 Data Description

The information domain of the MIS will be transformed into a set of related data structures stored in a relational database. The database will serve as the central repository for all system entities. The conceptual data model consists of the following primary entities identified in the SRS:

- **Users:** A central entity representing any individual who can log in. It stores credentials and role information.
- **Profiles:** Stores personal information specific to each user role (e.g., physical attributes for an athlete). This is linked one-to-one with the Users entity.
- **Training Centers:** The organizational unit to which users belong.
- **Performance Logs:** Records of an athlete's training or competition results, logged by a coach or administrator.

- **Payments:** Records of all financial transactions, such as admission fees paid by athletes.

Key relationships include:

- a User belongs to one TrainingCenter.
- a Coach (User) is assigned many Athletes (Users).
- an Athlete has many PerformanceLogs and many Payments.

4.2 Data Dictionary

The following tables define the structure for the primary database entities.

Table: users

This table stores the core information for any individual who can log into the system. It manages credentials, roles, and links to a training center.

Column Name	**Data Type**	**Constraints**	**Description**
user_id	SERIAL	PRIMARY KEY	Unique identifier for the user.
phone_number	VARCHAR(15)	UNIQUE, NOT NULL	User's registered phone number, used for login and OTP.
email	VARCHAR(255)	UNIQUE	User's optional email address.
password_hash	VARCHAR(255)	NOT NULL	Securely hashed user password (using bcrypt/Argon2).
role	ENUM('Athlete', 'Coach', 'Admin')	NOT NULL	The role assigned to the user.
training_center_id	INTEGER	FOREIGN KEY	Links the user to their training center.
created_at	TIMESTAMP	DEFAULT NOW()	Timestamp of account creation.
gender	VARCHAR(50)		User's gender.

Table: athlete_profiles

This table contains detailed personal and athletic information specific to users with the '**Athlete**' role.

Column Name	Data Type	Constraints	Description
profile_id	SERIAL	PRIMARY KEY	Unique identifier for the profile.
user_id	INTEGER	FOREIGN KEY (users), UNIQUE	Links to the corresponding user account.
first_name	VARCHAR(100)	NOT NULL	Athlete's first name.
last_name	VARCHAR(100)	NOT NULL	Athlete's last name.
date_of_birth	DATE	NOT NULL	Athlete's date of birth.
coach_id	INTEGER	FOREIGN KEY (users)	The user_id of the coach assigned to this athlete.
achievements	TEXT		A summary of the athlete's achievements.
profile_completion	INTEGER	DEFAULT 0	Percentage of profile completion, tracked for athletes.
physical_attributes	JSONB		Flexible field for storing physical data (e.g., height, weight).

Table: Coach

Column Name	Data Type	Constraints	Description
profile_id	SERIAL	PRIMARY KEY	Unique identifier for the profile.

user_id	INTEGER	FOREIGN KEY (users), UNIQUE	Links to the corresponding user account.
discipline	VARCHAR(100)	NOT NULL	The coach's primary Wushu discipline (e.g., Taolu, Sanda).

Table: training_centers

This table stores information about the individual training centers that use the MIS. Every user in the system is associated with one of these centers.

Column Name	Data Type	Constraints	Description
center_id	SERIAL	PRIMARY KEY	Unique identifier for the training center.
center_name	VARCHAR(255)	NOT NULL, UNIQUE	The official name of the training center.
address	TEXT		The physical address of the center.
city	VARCHAR(100)		The city where the center is located.
contact_email	VARCHAR(255)		A primary contact email for the center.
created_at	TIMESTAMP	DEFAULT NOW()	Timestamp of when the center was added to the system.

The training_center_id foreign key in the users table references the training_centers table, establishing the relationship between a user and their designated center.

Table: performance_logs

This table stores records of an athlete's performance, whether from daily training or official competitions.

Column Name	Data Type	Constraints	Description
log_id	SERIAL	PRIMARY KEY	Unique identifier for the performance entry.
athlete_id	INTEGER	FOREIGN KEY (users)	The athlete for whom the data is being logged.
logged_by_id	INTEGER	FOREIGN KEY (users)	The coach or admin who logged the data.
log_date	DATE	NOT NULL	The date of the training or competition.
log_type	ENUM('Training', 'Official')	NOT NULL	Differentiates between practice and official results.
attendance	BOOLEAN		Daily attendance status logged by a coach.
scores	JSONB		Flexible field to store different score types (e.g., Taolu, Sanda).

Table: payments

This table tracks all financial transactions within the system, such as admission fees.

Column Name	Data Type	Constraints	Description
payment_id	SERIAL	PRIMARY KEY	Unique identifier for the transaction.

athlete_id	INTEGER	FOREIGN KEY (users)	The athlete who made the payment.
amount	DECIMAL(10, 2)	NOT NULL	The amount paid.
payment_date	TIMESTAMP	DEFAULT NOW()	Timestamp of the transaction.
status	ENUM('Pending', 'Completed', 'Failed')	NOT NULL	The status of the payment gateway transaction.
is_confirmed	BOOLEAN	DEFAULT FALSE	Flag set by an Administrator upon confirming payment receipt.

5. Component Design

This section provides a detailed breakdown of the system's major software components. Each component's responsibilities, primary functions, and internal logic are described below. It is organized by component, with each function detailed for clarity.

5.1 User & Authentication Component

Purpose: This component handles the user lifecycle, authentication, security, and management of core user data stored in the users table.

- **5.1.1 Function: registerUser()**
  - **Purpose**: To handle the initial registration of a new user, including OTP verification.
  - **Associated Requirements**: FR-GEN-001
  - **Inputs**:
    - phoneNumber: string
    - password: string
    - email (optional): string
  - **Processing Logic**:
    1. Validate input formats and check if the phoneNumber already exists in the users table.
    2. Generate a 6-digit OTP and send it to the user's phone number via an SMS/E-mail gateway.
    3. Temporarily store the user's details and the OTP for subsequent verification.
  - **Outputs**: A success or failure message indicating whether the OTP was sent.

- **5.1.2 Function: loginUser()**
  - **Purpose**: To authenticate a user with their phone number and password and establish a secure session.
  - **Associated Requirements**: FR-GEN-002 , NFR-SEC-02
  - **Inputs**:
    - phoneNumber: string
    - password: string
  - **Processing Logic**:
    1. Retrieve the user_record from the users table where the phone_number matches.
    2. If no user is found, return an "User not found" error.
    3. Securely compare the provided password with the password_hash stored in the user_record.
    4. If the password is valid, generate a session token containing the user_id and role.
    5. Return a success status along with the session token.
  - **Outputs**: A success or failure status. On success, a session token is also returned.

- **5.1.3 Function: changePassword()**
  - **Purpose**: To allow a logged-in user to change their password.
  - **Associated Requirements**: Standard security best practice.
  - **Inputs**:
    - userId: integer
    - oldPassword: string
    - newPassword: string
  - **Processing Logic**:
    1. Retrieve the user's current password_hash from the users table.
    2. Verify that the oldPassword matches the stored hash.
    3. If it matches, securely hash the newPassword and update the password_hash in the database.
  - **Outputs**: A success or failure confirmation message.

- **5.1.4 Function: getCoachesByCenter()**
  - **Purpose**: To retrieve a list of all coaches associated with a specific training center for an administrator's view.
  - **Associated Requirements**: FR-TCA-002
  - **Inputs**:
    - centerId: integer
  - **Processing Logic**:
    1. Query the users table.
    2. Select all records where training_center_id matches the input centerId and the role is 'Coach'.

  - **Outputs**: An array of user objects representing the coaches.

- **5.1.5 Function: recoverPassword()**
  - **Purpose**: To handle the workflow for a user who has forgotten their password and cannot log in.
  - **Associated Requirements**: FR-GEN-002.
  - **Inputs**:
    - phoneNumber: string
  - **Processing Logic**:
    1. Find the user in the users table based on the provided phoneNumber.
    2. If no user is found, return an error message.
    3. Generate a secure, temporary token or a 6-digit One-Time Password (OTP).
    4. Store this token/OTP in a temporary cache or a dedicated database table, linking it to the user's ID with a short expiry time (e.g., 10 minutes).
    5. Send the token/OTP to the user's registered phoneNumber via the SMS gateway.
  - **Outputs**: A success message indicating that a recovery code has been sent. This response does not include the code itself.

- **5.1.6 Function: setTrainingCenter()**
  - **Purpose**: To handle the one-time action of a new coach selecting their primary training center upon their first login.
  - **Associated Requirements**: FR-COA-001
  - **Inputs**:
    - coachId: integer
    - centerId: integer
  - **Processing Logic**:
    1. Verify that the user with coachId has the 'Coach' role.
    2. Update the record in the users table for the given coachId, setting the training_center_id to the provided centerId.
  - **Outputs**: A success or failure confirmation message.

5.1.7 Function: updateProfile()

- **Purpose:** To allow any authenticated user to update their own profile information stored across the `users` and role-specific profile tables.
- **Associated Requirements:** FR-GEN-003
- **Inputs:**
  - userId: integer
  - profileData: object (e.g., { firstName: 'NewName', address: 'New Address' })
- **Processing Logic:**
  - Verify that the session userId matches the userId being updated to ensure users can only edit their own profile.

    - Based on the user's role, update the corresponding records in the users table and the athlete_profiles or coach_profiles table.
    - Validate all incoming profileData to ensure it meets format and type constraints.
- **Outputs:** A success or failure confirmation message.

5.2 Athlete Profile Component

Purpose: This component manages all detailed information specific to users with the 'Athlete' role. It primarily interacts with the athlete_profiles table.

- **5.2.1 Function: createAthleteProfile()**
    - **Purpose**: To create the initial profile record for a newly registered athlete.
    - **Associated Requirements**: FR-ATH-001

    - **Inputs**:
        - userId: integer
        - firstName: string
        - lastName: string
        - dateOfBirth: date
    - **Processing Logic**:
        1. Create a new row in the athlete_profiles table.
        2. Link the new profile to the user by setting the user_id foreign key.
        3. Populate the record with the provided personal details.
    - **Outputs**: The ID of the newly created profile.

- **5.2.2 Function: updateProfileCompletion()**
    - **Purpose**: To calculate and update the profile completion percentage displayed on an athlete's dashboard.
    - **Associated Requirements**: FR-ATH-001
    - **Inputs**:
        - userId: integer
    - **Processing Logic**:
        1. Retrieve the athlete's profile from the athlete_profiles table using the userId.
        2. Calculate a completion percentage based on how many of the key fields are filled.
        3. Update the profile_completion column for that athlete's record in the database.
    - **Outputs**: A confirmation that the update was successful.

- **5.2.3 Function: assignCoach()**

- **Purpose**: To allow an administrator to assign a specific coach to an athlete.
- **Associated Requirements**: Implied by BR-02 and the coach_id field in athlete_profiles.
- **Inputs**:
  - adminId: integer
  - athleteId: integer
  - coachId: integer
- **Processing Logic**:
  1. Verify the adminId has the 'Admin' role.
  2. Update the athlete_profiles record for the athleteId, setting the coach_id to the provided coachId.
- **Outputs**: A success or failure message.

- **5.2.4 Function: getAssignedAthletes()**
  - **Purpose**: To retrieve the roster of athletes assigned to a specific coach.
  - **Associated Requirements**: FR-COA-002
  - **Inputs**:
    - coachId: integer
  - **Processing Logic**:
    1. Query the athlete_profiles table.
    2. Select all records where coach_id matches the input coachId.
  - **Outputs**: An array of athlete profile objects.

- **5.2.5 Function: getAthletesByCenter()**
  - **Purpose**: To retrieve a list of all athletes registered at a specific training center for an administrator's view.
  - **Associated Requirements**: FR-TCA-003
  - **Inputs**:
    - centerId: integer
  - **Processing Logic**:
    1. Query the users table and join with athlete_profiles.
    2. Select all records where training_center_id matches the input centerId and the role is 'Athlete'.
  - **Outputs**: An array of objects containing combined user and athlete profile data.

- **5.2.6 Function: applyToTrainingCenter()**
  - **Purpose**: To formally link an athlete's account to a specific training center after they have registered and completed their initial profile.
  - **Associated Requirements**: FR-ATH-002
  - **Inputs**:

        - athleteId: integer
        - centerId: integer
    - **Processing Logic**:
        1. Verify that both the athleteId and centerId are valid and exist.
        2. Update the record in the users table for the given athleteId, setting the training_center_id to the provided centerId.
        3. This action may trigger other processes, such as initiating the admission payment or sending a notification to the center's administrator.
    - **Outputs**: A success or failure confirmation message.

5.3 Training Center Component

Purpose: Manages the organizational entities within the MIS. It interacts with the training_centers table.

- **5.3.1 Function: getCenterList()**
    - **Purpose**: To retrieve a list of all available training centers for selection during user onboarding.
    - **Associated Requirements**: FR-ATH-002 ,FR-COA-001
    - **Inputs**: None.
    - **Processing Logic**:
        1. Execute a query to select all records from the training_centers table.
    - **Outputs**: An array of training center objects.

- **5.3.2 Function: getCenterDashboardStats()**
    - **Purpose**: To aggregate and retrieve key statistics for a Training Center Administrator's dashboard.
    - **Associated Requirements**: FR-TCA-001
    - **Inputs**:
        - centerId: integer
    - **Processing Logic**:
        1. Count all users in the users table where training_center_id matches and role is 'Athlete'.
        2. Count all users in the users table where training_center_id matches and role is 'Coach'.
        3. Assemble the counts into a data object.
    - **Outputs**: A data object with statistics (e.g., { total_athletes: 52, total_coaches: 4 }).

5.4 Performance Logging Component

Purpose: Handles the recording and retrieval of athlete performance data. This component interacts primarily with the performance_logs table.

- **5.4.1 Function: logPerformance()**
  - **Purpose**: To allow an authorized user (Coach) to log training data for one of their assigned athletes.
  - **Associated Requirements**: FR-COA-003 , BR-02
  - **Inputs**:
    - coachId: integer
    - athleteId: integer
    - logData: object
  - **Processing Logic**:
    1. Perform an authorization check by querying athlete_profiles to confirm the athleteId is assigned to the coachId.
    2. If authorized, insert a new record into the performance_logs table.
  - **Outputs**: A success or failure message.

- **5.4.2 Function: getPerformanceHistory()**
  - **Purpose**: To retrieve all historical performance logs for a specific athlete for display on their dashboard.
  - **Associated Requirements**: FR-ATH-004
  - **Inputs**:
    - athleteId: integer
  - **Processing Logic**:
    1. Query the performance_logs table for all records where athlete_id matches.
    2. Order the results by log_date.
  - **Outputs**: An array of performance log objects.

5.5 Payment Management Component

Purpose: Manages all financial transactions. It interacts with the payments table.

- **5.5.1 Function: initiateAdmissionPayment()**
  - **Purpose**: To create a new payment record when an athlete begins the admission payment process.
  - **Associated Requirements**: FR-ATH-007
  - **Inputs**:
    - athleteId: integer
    - amount: decimal
  - **Processing Logic**:
    1. Insert a new record into the payments table with the athleteId, amount, and a 'Pending' status.
  - **Outputs**: The ID of the newly created payment transaction.

- **5.5.2 Function: confirmPayment()**
    - **Purpose**: To allow a user with the 'Admin' role to confirm the receipt of a payment.
    - **Associated Requirements**: FR-TCA-004
    - **Inputs**:
        - paymentId: integer
        - adminId: integer
    - **Processing Logic**:
        1. Verify the adminId has the 'Admin' role.
        2. Update the specified record in the payments table, setting is_confirmed to TRUE and status to 'Completed'.
    - **Outputs**: A success or failure message.

- **5.5.3 Function: getPaymentHistory()**
    - **Purpose**: To retrieve a full history of an athlete's transactions.
    - **Associated Requirements**: FR-ATH-008
    - **Inputs**:
        - athleteId: integer
    - **Processing Logic**:
        1. Query the payments table for all records where athlete_id matches the input.
    - **Outputs**: An array of payment objects.

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The MIS user interface will be clean, modern, and intuitive, designed to be fully responsive for optimal use on desktops, tablets, and mobile phones. The design will emphasize ease of use, ensuring that users with basic web literacy can complete key tasks with minimal training (NFR-QUAL-05). A consistent navigation structure, layout, and design language will be used across the entire application to provide a predictable and user-friendly experience. The system will provide immediate and clear feedback for all user actions, such as success messages on form submission or descriptive error messages for invalid input (UI-03).

6.2 Screen Images(to be done)

The following are conceptual mockups of key system screens.

Login & Registration Screen: A simple and clean interface for new user registration and existing user login.

Athlete Dashboard: The landing page for athletes, featuring a profile completion progress bar, a prominent performance history chart, and a list of upcoming events.

Coach's Athlete Roster: A page for coaches to view a list of all athletes assigned to them, with options to select an athlete to log daily performance or attendance data.

Administrator's Payment Confirmation Page: A tabular view for administrators to see all incoming payments, with functionality to search, filter, and confirm transactions.

6.3 Screen Objects and Actions(Explain what each button, field, or menu does) —-> to be done

This section describes the objects and actions for a key screen.

Screen: Athlete Dashboard

Screen Objects:

Navigation Bar: Contains links to "Dashboard," "My Profile," and a "Logout" button.

Profile Completion Widget: A visual progress bar with a percentage text (e.g., "Your profile is 75% complete") and a link to "Complete Profile."

Performance History Chart: An interactive line graph. The X-axis represents time, and the Y-axis represents scores. A dropdown filter allows switching between different metrics.

Data Entry Button: A button labeled "Log Today's Training".

Associated Actions:

- Clicking the "Complete Profile" link navigates the user to the profile editing page (FR-GEN-004).

- Hovering over a data point on the Performance History Chart displays a tooltip with the exact date and score for that entry.
- Using the chart's filter dropdown re-renders the graph to display the selected performance metric (FR-ATH-002).

- Clicking the "Logout" button terminates the user's session and redirects them to the login screen.

7. REQUIREMENTS MATRIX

This matrix provides a cross-reference tracing system design components back to the requirements specified in the SRS.

Requirement ID	Requirement Description	Design Component(s) Satisfying Requirement
FR-GEN-001	User registration with mandatory phone number and OTP verification.	The User & Authentication Component (registerUser() function) uses the users table and integrates with an SMS Gateway.
FR-GEN-002	Secure user login and password recovery.	The User & Authentication Component (loginUser(), recoverPassword() functions) queries the users table's password_hash column.
FR-ATH-001	Athlete profile completion is required and tracked by a percentage.	The Athlete Profile Component (createAthleteProfile(), updateProfileCompletion() functions) updates the athlete_profiles table, which is displayed on the Athlete Dashboard.
FR-ATH-002	Athletes select a training center from a list.	The Training Center Component (getCenterList() function) provides the list, and the Athlete Profile Component (applyToTrainingCenter() function) handles the selection.
FR-ATH-004	Athletes can view their own performance data and progress charts.	The Performance Logging Component (getPerformanceHistory() function) retrieves data from the performance_logs table for the Presentation Tier to display on the

		Athlete Dashboard.
FR-ATH-008	An athlete can view a full history of their transactions.	The Payment Management Component (getPaymentHistory() function) queries the payments table for all of the athlete's records.
FR-COA-001	On first login, coaches must select their Training Center.	The Training Center Component (getCenterList() function) provides the options, and the User & Authentication Component (setTrainingCenter() function) saves the selection.
FR-COA-002	Coaches can view a list of all athletes registered under them.	The Athlete Profile Component (getAssignedAthletes() function) retrieves the roster of athletes assigned to a specific coach.
FR-COA-003	Coaches can log daily attendance and performance data for their assigned athletes.	The Performance Logging Component's logPerformance() function writes to the performance_logs table after performing an authorization check.
FR-TCA-001	The admin dashboard displays center statistics, including total coach and athlete counts.	The Training Center Component's getCenterDashboardStats() function queries the users and training_centers tables for display on the

		Administrator Dashboard.
FR-TCA-002	Admins can view a list of all coaches associated with their center.	The User & Authentication Component's getCoachesByCenter() function retrieves all users with the 'Coach' role for a given center.
FR-TCA-003	Admins can view a complete roster of all athletes.	The Athlete Profile Component's getAthletesByCenter() function retrieves all users with the 'Athlete' role for a given center.
FR-TCA-004	Admin can view and confirm payments received from athletes.	The Payment Management Component (confirmPayment() function) updates the payments table, with results shown on the Administrator's Payment Page.
NFR-PERF-01	All user-facing pages shall achieve a load time of under 3 seconds.	This is satisfied by the overall Three-Tier Architecture, including an optimized Presentation Tier, efficient queries in the Application Tier, and database indexing in the Data Tier.
NFR-SEC-02	All user passwords shall be securely stored using a strong, one-way hashing algorithm.	The User & Authentication Component handles this, storing the hashed value in the users table's password_hash column.

NFR-SEC-03	The system shall implement strict Role-Based Access Control (RBAC).	RBAC is enforced in the Application Tier's middleware and in the business logic of functions like logPerformance() and assignCoach(), using the role column from the users table.
NFR-QUAL-04	The application shall be containerized (e.g., using Docker) for easy deployment.	This is satisfied by the System Architecture's choice to use Docker for containerization and deployment.