# **Ievolve Event Management System - Software Design Document (SDD)**

**Version: 1.2**

**Date: August 6, 2025**

**Author: Harini**

### 1. Introduction

#### 1.1 Purpose

This Software Design Document (SDD) provides a detailed technical blueprint for the development of the Ievolve Event Management System. It outlines the high-level system architecture, component-level design, data structures, and security mechanisms. This document translates the functional and non-functional requirements specified in the SRS into a concrete design, serving as the primary guide for the development team.

#### 1.2 Scope

The scope of this design is strictly limited to **Phase 1: CM Trophy Accommodation Management**, as defined in the Software Requirements Specification (SRS v1.0) and the Scope of Work (SOW v1.0). The design covers the functionalities required for Ievolve Admins to manage hotel inventory and participant data, and for Team Coaches to manage their team's accommodation check-in and check-out processes. Any functionalities listed as "out-of-scope" in the SOW, such as catering or travel management, are not covered in this design.

#### 1.3 References

* **Software Requirements Specification (SRS) for Ievolve Event Management System, v1.0**
* **Scope of Work (SOW) for Ievolve Event Management System, v1.0**
* **Database Design Document (DDD) for Ievolve Event Management System, v1.0**

#### 1.4 Definitions and Acronyms

* **API:** Application Programming Interface
* **DDD:** Database Design Document
* **ERD:** Entity-Relationship Diagram
* **ORM:** Object-Relational Mapper
* **OTP:** One-Time Password
* **RBAC:** Role-Based Access Control
* **REST:** Representational State Transfer
* **SOW:** Scope of Work
* **SRS:** Software Requirements Specification
* **UI:** User Interface

#### 1.5 Version History

| **Version** | **Date** | **Author** | **Reviewer** | **Customer Reviewers** | **Description of Changes** |
| --- | --- | --- | --- | --- | --- |
| 1.0 | August 6, 2025 | Harini | Gunasekaran | Diwakar, AV Prashanth | Initial Draft of SDD. |

### 2. System Architecture

#### 2.1 Architectural Style

The Ievolve Event Management System will be built using a **Monorepo with Three-Tier Architecture**. This pattern organizes the frontend, backend, and shared code within a single repository, which enhances code sharing, simplifies dependency management, and streamlines the development workflow. The tiers are:

1. **Presentation Tier (Client):** A responsive web interface built with React, responsible for rendering the UI and capturing user input.
2. **Logic Tier (Application Server):** A back-end server built with Node.js/Express.js that contains the core business logic, processes data, and handles all application workflows. It exposes a **RESTful API** with types shared from the monorepo for communication with the Presentation Tier.
3. **Data Tier (Database Server):** A **PostgreSQL** database responsible for the persistent storage and retrieval of all application data, accessed via the Drizzle ORM.

#### 2.2 System Overview Diagram

The following diagram illustrates the high-level interaction between the system's components.

#### 2.3 Technology Stack

* **Front-end (Presentation Tier):**
  + **Framework:** React.js with TypeScript
  + **UI Library:** Shadcn/ui (built on Radix UI)
  + **Styling:** Tailwind CSS with CSS custom properties
  + **Routing:** Wouter
  + **Data Fetching & State Management:** TanStack Query
  + **Form Handling:** React Hook Form with Zod for validation
  + **Build Tool:** Vite
  + **Icons:** Lucide React
* **Back-end (Logic Tier):**
  + **Runtime:** Node.js with Express.js
  + **Language:** TypeScript (ES Modules)
  + **ORM:** Drizzle ORM with Drizzle Kit for migrations
  + **Authentication:** Express sessions with a PostgreSQL session store
  + **Password Hashing:** bcrypt.js
  + **Development/Execution:** tsx
* **Database (Data Tier):**
  + **Database:** PostgreSQL
* **Deployment:**
  + **Hosting:** (**Yet to be decided**)

### 3. Component Design

The Logic Tier is composed of several interconnected modules within a monorepo structure, each responsible for a specific set of functionalities.

#### 3.1 User Management & Authentication Component

* **Responsibilities:** Manages user identity, session-based authentication, and authorization.
  + Handles Ievolve Admin login via email and password.
  + Handles Team Coach login via registered mobile number and OTP.
  + Creates and manages user sessions stored in the PostgreSQL session store.
  + Enforces Role-Based Access Control (RBAC) to ensure users can only access authorized resources.
* **Interfaces (API Endpoints):**
  + POST /api/auth/admin/login: Authenticates an Admin and creates a session.
  + POST /api/auth/coach/request-otp: Sends an OTP to a coach's mobile number.
  + POST /api/auth/coach/verify-otp: Verifies the OTP and authenticates a coach, creating a session.
* **Dependencies:** users table (DDD 3.1), External SMS Gateway, Express sessions, PostgreSQL session store.

#### 3.2 Data Import & Validation Component

* **Responsibilities:** Processes and validates bulk data uploads from Ievolve Admins.
  + Parses PSV (Pipe Separated Files) files for Hotel Inventory, Coach/Official Data, and Player Data.
  + Enforces all validation rules specified in the SRS (FR-1.6, FR-1.8, FR-1.9), including unique HotelID/date ranges, minimum stay validation, and foreign key checks against hotel inventory.
  + Creates new records in the hotels and participants tables upon successful validation.
  + Returns a detailed error report to the Admin UI if validation fails.
  + Creates user accounts for new Team Coaches from the uploaded data without creating duplicates.
* **Interfaces (API Endpoints):**
  + POST /api/admin/upload/hotel-inventory: Uploads the hotel inventory sheet.
  + POST /api/admin/upload/coaches-officials: Uploads the coach and official data sheet.
  + POST /api/admin/upload/players: Uploads the player data sheet.
* **Dependencies:** hotels table (DDD 3.2), participants table (DDD 3.3).

#### 3.3 Accommodation & Booking Management Component

* **Responsibilities:** Manages all core logic related to accommodation lifecycle events.
  + Facilitates check-in and early check-out initiated by Team Coaches. Check In should send notification to the Travel POC added in the Bulk Upload Sheet.
  + Allows Admins to manage transit bookings, perform early checkouts for teams, and handle hotel reassignments.
  + Updates the checkin\_status, checkin\_time, and checkout\_time fields in the participants table.
  + Logs reassignment details in the reassignments table.
* **Interfaces (API Endpoints):**
  + PUT /api/coach/players/{participantId}/check-in: Coach initiates player check-in.
  + PUT /api/coach/players/{participantId}/check-out: Coach initiates player early check-out.
  + GET /api/coach/team: Fetches the list of players managed by the logged-in coach.
  + PUT /api/admin/teams/early-checkout: Admin initiates early check-out for a team.
  + POST /api/admin/reassign: Admin performs a hotel reassignment.
* **Dependencies:** participants table (DDD 3.3), reassignments table (DDD 3.5), audit\_log table (DDD 3.6).

#### 3.4 Reporting & Dashboard Component

* **Responsibilities:** Aggregates and computes data for dashboards and reports.
  + Calculates estimated daily room vacancies based on the logic in DDD 5.1.
  + Provides filtered and paginated views of all participants and booking data.
  + Generates CSV exports of filtered data for Admins.
* **Interfaces (API Endpoints):**
  + GET /api/admin/dashboard/stats: Fetches summary statistics for the Admin dashboard.
  + GET /api/admin/dashboard/bookings: Fetches a filterable list of all bookings.
  + GET /api/admin/dashboard/bookings/export: Exports the filtered booking data to CSV.
  + GET /api/coach/dashboard: Fetches dashboard data for the logged-in coach.
* **Dependencies:** participants, hotels, and bookings tables.

#### 3.5 Notification Component

* **Responsibilities:** Manages all outgoing user notifications.
  + Integrates with a third-party SMS gateway to send OTPs for coach login.
  + Sends automated SMS notifications to Team Coaches and Players upon an Admin-initiated early checkout (FR-3.2).
* **Interfaces (Internal):**
  + NotificationService.sendOtp(mobileNumber, otp)
  + NotificationService.sendEarlyCheckoutAlert(mobileNumber, message)
* **Dependencies:** External SMS Gateway API, participants table.

#### 3.6 Audit Component

* **Responsibilities:** Logs all significant administrative actions for accountability and traceability.
  + Creates an audit trail for data uploads, record edits/deletions, early checkouts, and reassignments, as detailed in DDD 3.7.
* **Interfaces (Internal):**
  + AuditService.logAction(userId, actionType, targetEntity, targetId, details)
* **Dependencies:** audit\_log table (DDD 3.6).

### 4. Data Design

The data storage layer will be implemented exactly as specified in the **Ievolve Event Management System - Database Design Document (DDD), v1.0**.

#### 4.1 Database Schema

The database schema, including all tables (users, hotels, participants, bookings, reassignments, audit\_log), columns, data types, constraints, and relationships, will adhere to the definitions in the DDD. All database interactions will be managed through the **Drizzle ORM**.

#### 4.2 Entity-Relationship Diagram (ERD)

The following ERD visually represents the relationships between the primary data entities as defined in the DDD.

### 5. User Interface (UI) Design - Wireframes

The following high-level wireframes illustrate the conceptual layout and user flow for the key screens of the application.

#### 5.1 Admin Portal

* **Dashboard:**
* **Participant Management & Data Upload:**

#### 5.2 Coach Portal (Mobile View)

* **Login & Team View:**

### 6. Security Design

* **Authentication:**
  + **Session-Based Authentication:** The system will use Express sessions. Upon successful login, a session ID is created and stored in a secure, server-side PostgreSQL session store. A cookie containing this session ID is sent to the client. Subsequent requests are authenticated by validating this session cookie against the server-side store.
  + **Password Hashing:** Passwords will be securely hashed using **bcrypt.js**.
* **Authorization (RBAC):**
  + Access to API endpoints will be protected based on the user's role, which is stored in their session data. A middleware will check the user's session and role on every request to a protected route.
  + Coaches will be restricted to accessing data related only to their assigned players. This will be enforced by filtering database queries by the coach\_id from their validated session.
* **Data Security:**
  + **In Transit:** All communication between the client and server will be encrypted using **TLS/SSL**.
  + **At Rest:** Sensitive data in the database will be protected using measures outlined in the DDD.
  + **SQL Injection:** The use of the **Drizzle ORM** will prevent SQL injection vulnerabilities by parameterizing all queries.
* **API Security:**
  + All authenticated endpoints will require a valid session cookie.
  + The system will implement measures like rate limiting to prevent brute-force attacks.

### 7. Non-Functional Requirements

* **Performance:** The system will be optimized to handle bulk data uploads of up to 10,000 records in under 60 seconds. Dashboard queries and data filtering will render results in under 3 seconds.
* **Scalability:** The stateless nature of the Logic Tier and the chosen database solution will allow for scaling to handle increased load during peak event times.
* **Reliability:** The application will leverage the reliability of the Replit hosting platform and the chosen PostgreSQL provider. Automated backups of the PostgreSQL database will be configured.
* **Usability:** The UI will be designed to be intuitive and self-explanatory, minimizing the need for extensive user training.

### 8. Deployment and Maintenance

* **Deployment:** The application will be deployed and hosted directly on the **Customer Decided** platform, leveraging its integrated environment for development, testing, and production.
* **Monitoring:** Application performance and server health will be monitored using Replit's built-in tools. A logging service will aggregate logs from all application components for easy troubleshooting.
* **Maintenance:** Regular maintenance will include database backups, security patching, and dependency updates.