Software Requirements Specification

for

Loyalty Program Management System

Version 1.0 approved

Prepared by

Gourav Anirudh (IMT2023005)

Sathish Adithiyaa S V (IMT2023030)

Sahil Kolte (IMT2023066)

Sriram Srikanth (IMT2023115)

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements for the "Loyalty Program Management System" Version 1.0. This product is a web-based application designed to manage and track participant engagement during an event or a fest. Its purpose is to automate the process of awarding loyalty points for event attendance, provide a real-time ranking, and identify top participants for prizes.

1.2 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers:** To understand the system's requirements, features, and constraints for implementation.
- **Fest/Event Organizers:** To understand the scope, features, and business logic of the system.
- **Testers:** To develop test cases and verify that the system meets its specified requirements.
- **Admin Staff:** To understand their responsibilities and capabilities within the system.

1.3 Product Scope

The Loyalty Program Management System enables organizers to reward participants' engagement through a point-based system. Participants earn points on purchases, view rewards, and redeem them. Organizers can manage offers, track redemptions, and analyze user behavior.

2. Overall Description

2.1 Product Perspective

Loyalty Program Management System is a **new, stand-alone web system** consisting of:

- **Frontend:** TypeScript-based (React/Next.js) web app deployed on Vercel.
- **Backend:** Python API (FastAPI or Flask) provides logic and database connectivity.
- **Database:** NoSQL database (MongoDB).
- **Users:** Admin, participants, and volunteers.

2.2 Product Functions

- User registration & authentication
- Business registration and management
- Point allocation on purchases
- Reward catalog management
- Point redemption and transaction history
- Analytics dashboard for admins
- Notifications for offers and redemptions

2.3 User Classes and Characteristics

- **Participant:** Earns and redeems loyalty points.
- **Organizer (admin):** Manages offers, users, and redemptions.
- **System Admin:** Handles technical configuration and maintenance.

2.4 Operating Environment

- **Frontend:** Browser-based (Chrome, Firefox, Safari, Edge).
- **Backend:** Python 3.9+ environment.
- **Hosting:** On Vercel.

2.5 Design and Implementation Constraints

- Must be compatible with modern browsers.
- Built using TypeScript (frontend) and Python (backend).
- Deployment limited to platforms supporting Node.js 18+ and Python 3.9+.
- Must comply with data privacy regulations (e.g., GDPR).

2.6 User Documentation

- Web user guide (HTML help or Wiki page).
- API documentation via Swagger (FastAPI auto-docs).
- Developer setup guide (README.md).

2.7 Assumptions and Dependencies

- Users must have stable internet connectivity.

- Frontend and backend communicate over REST API.
- Users must have a specific organisational account if decided by event organisers to limit the audience to belong only to the organisation.

3. External Interface Requirements

3.1 User Interfaces

- **Login/Signup Page:** User authentication.
- **Dashboard:** Displays user points.
- **Admin Console:** Manage offers, users, and reward configurations.

3.2 Hardware Interfaces

- No dedicated hardware requirements; runs on any device with a web browser.

3.3 Software Interfaces

- **Frontend–Backend:** RESTful API using JSON.
- **Backend–Database:** mongo queries.

3.4 Communications Interfaces

- HTTPS for secure communication.
- REST API endpoints exposed at /api/....
- JSON is used as data interchange format.

4. System Features

4.1 User Authentication

Description: Secure login for participants, volunteers, and admins.

Priority: High

Functional Requirements:

- **REQ-1.1:** System shall allow users to log in with valid credentials.
- **REQ-1.2:** System shall differentiate users as Participants, Volunteers, or Admins based on their credentials.
- **REQ-1.3:** System shall maintain secure sessions for authenticated users.

4.2 Participant Features

Description: Participants can create or join teams, view team information, and track event performance.

Priority: High

Functional Requirements:

- **REQ-2.1:** System shall allow a participant to create a new team.
- **REQ-2.2:** System shall allow a participant to join an existing team using a team code or request approval.
- **REQ-2.3:** System shall allow a participant to leave a team.
- **REQ-2.4:** System shall allow a participant to view team details.
- **REQ-2.5:** System shall display a leaderboard ranking teams or participants based on points or attendance.

4.3 Volunteer Features

Description: Volunteers can authenticate themselves for an event and mark attendance for participants by scanning QR codes.

Priority: Medium

Functional Requirements:

- **REQ-3.1:** System shall allow volunteers to authenticate for an event using a secret code.
- **REQ-3.2:** System shall allow volunteers to scan participant QR codes to mark attendance.
- **REQ-3.3:** System shall ensure that the scan attendance process includes authentication validation.

4.4 Admin Features

Description: Admins manage volunteers and events within the system.

Priority: High

Functional Requirements:

- **REQ-4.1:** System shall allow admins to **add new volunteers**.
- **REQ-4.2:** System shall allow admins to **remove existing volunteers**.
- **REQ-4.3:** System shall allow admins to **create, read, update, and delete (CRUD) events**.
- **REQ-4.4:** System shall allow admins to view attendance and participation statistics for each event.

4.5 System Constraints

- Only authenticated users can access their respective functionalities.
- Volunteers must successfully authenticate using the event secret code before scanning participant QR codes.
- Admin functions (CRUD events, add/remove volunteers) are restricted to admin-level accounts.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- Response time < 2 seconds for 90% of API requests.
- Supports up to 1000 concurrent users.

5.2 Safety Requirements

- Regular backups of database.
- Validation of all API inputs to prevent crashes.

5.3 Security Requirements

- HTTPS is enforced across all connections.
- JWT-based authentication.
- Passwords hashed using standard algorithms.

5.4 Software Quality Attributes

- **Usability:** Responsive UI, mobile-friendly.
- **Maintainability:** Modular TypeScript and Python codebase.
- **Reliability:** Auto-restart and error logging.

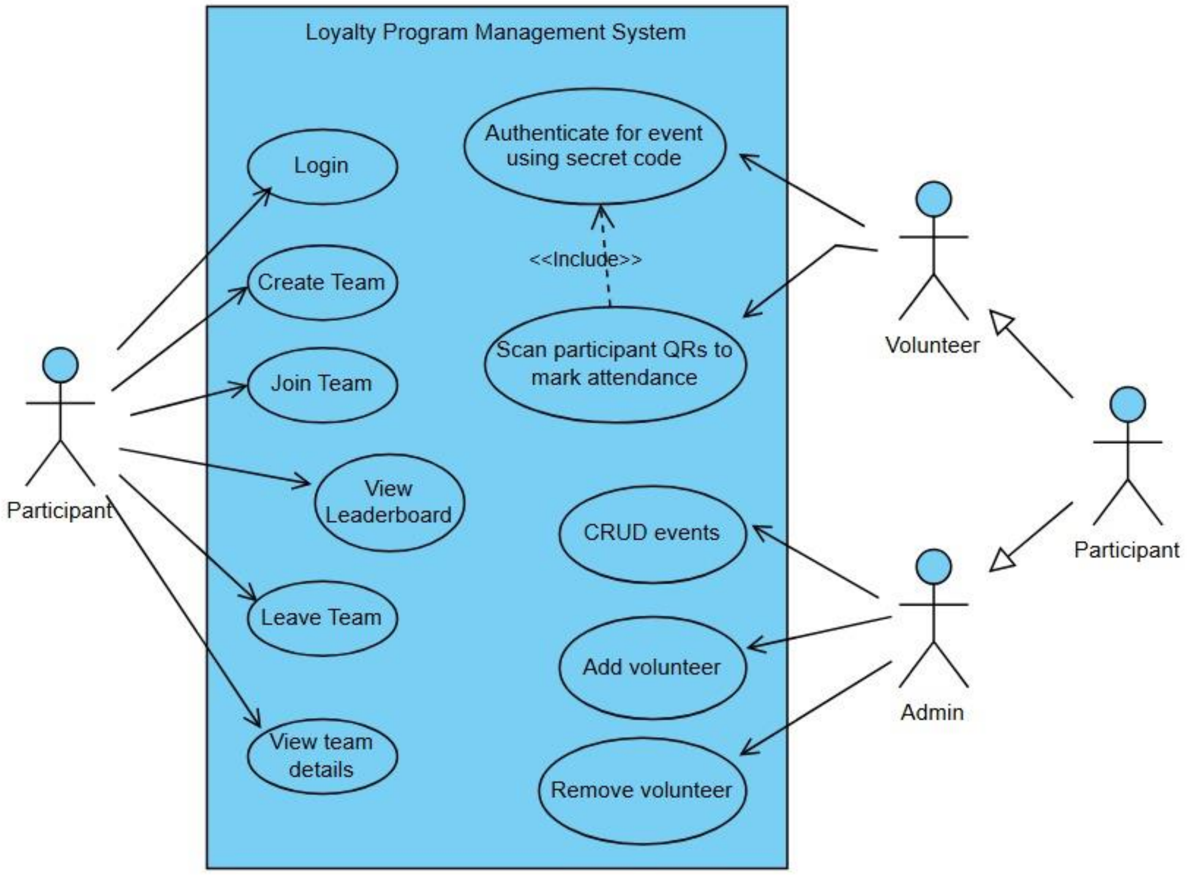
- **Scalability:** Deployable to cloud platforms.

5.5 Business Rules

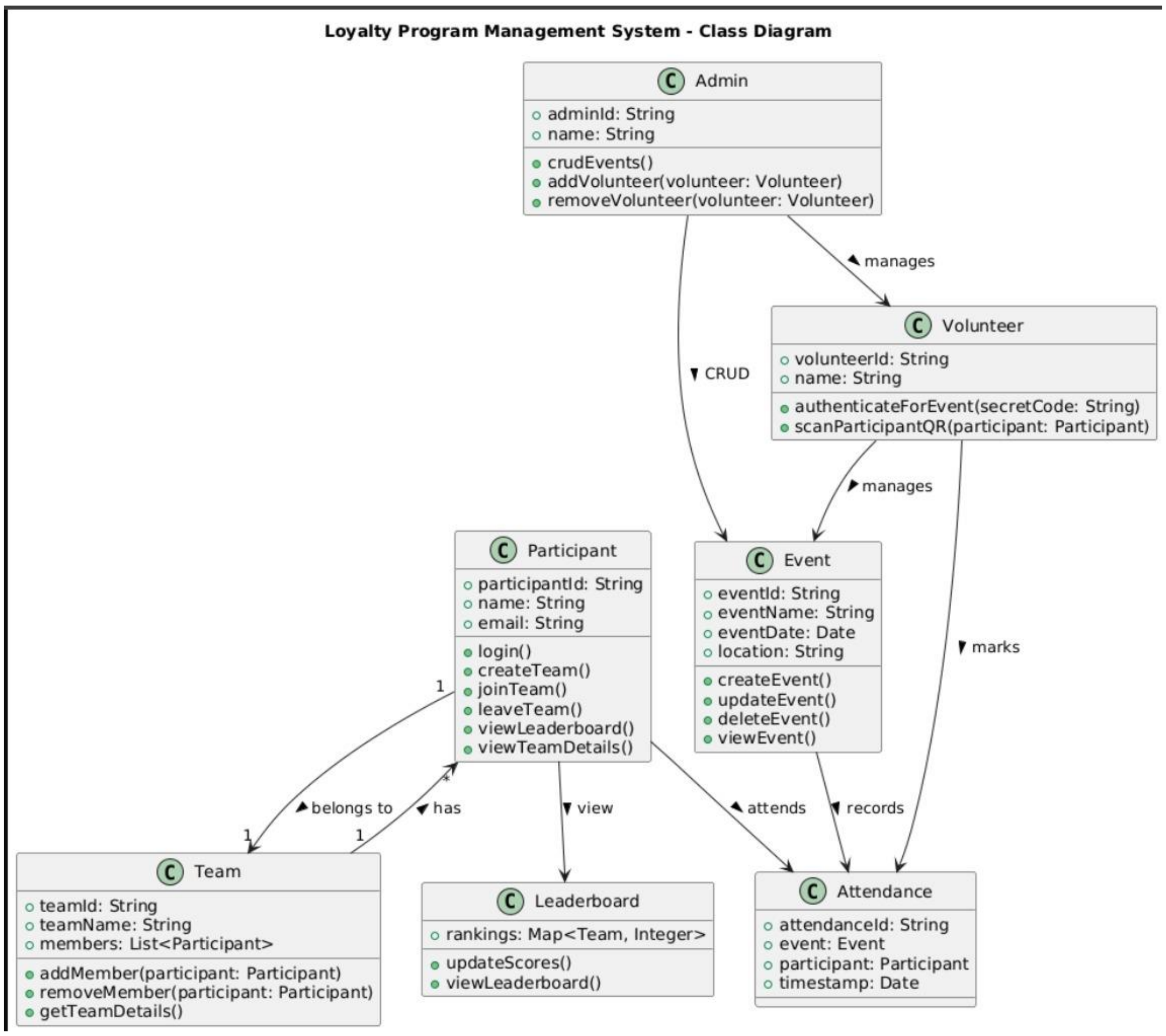
- Each customer account is unique per email.
- Points cannot be transferred between accounts.
- Admin approval required for manual adjustments.

6. Analysis Models

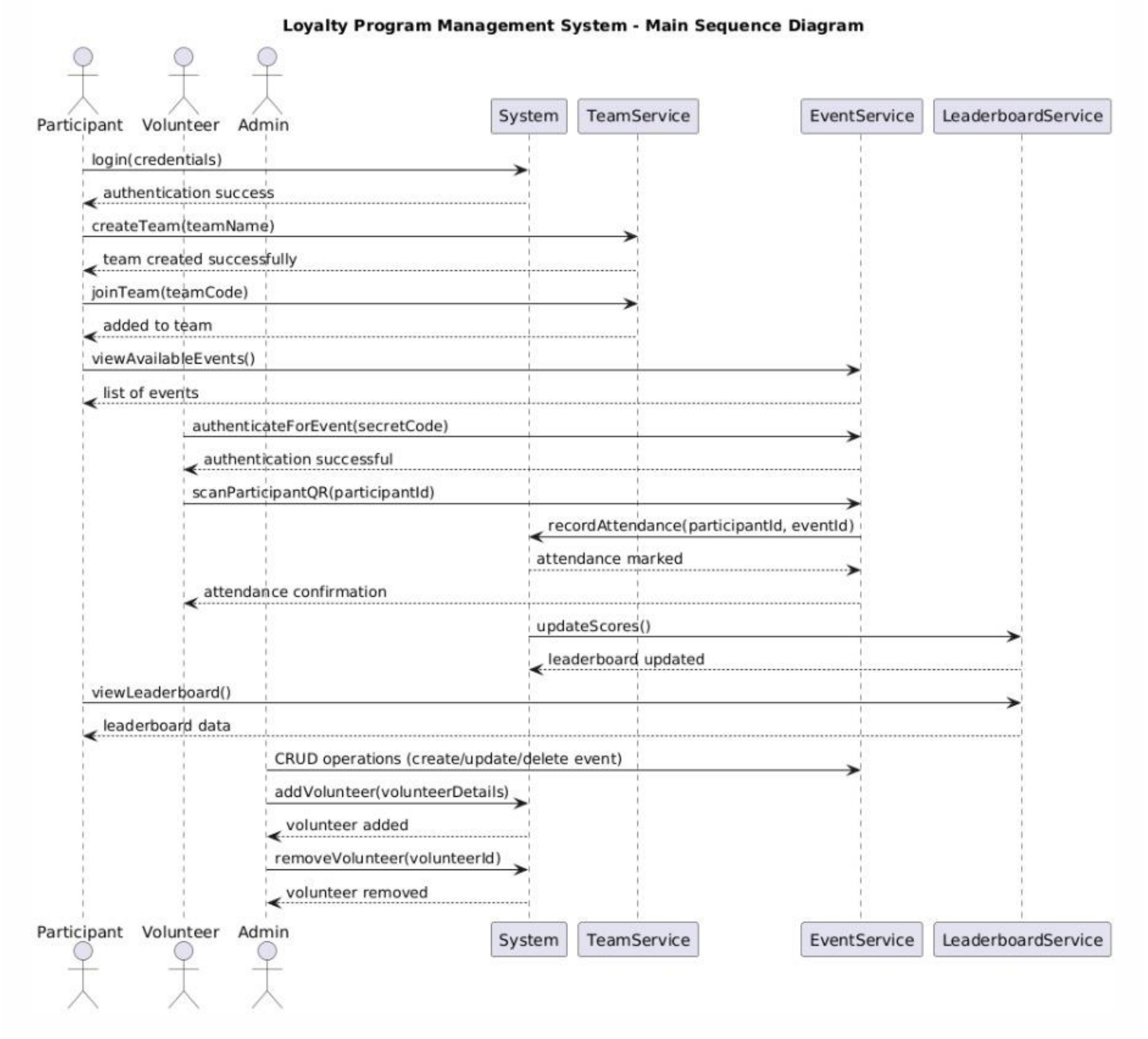
Use Case Diagram:

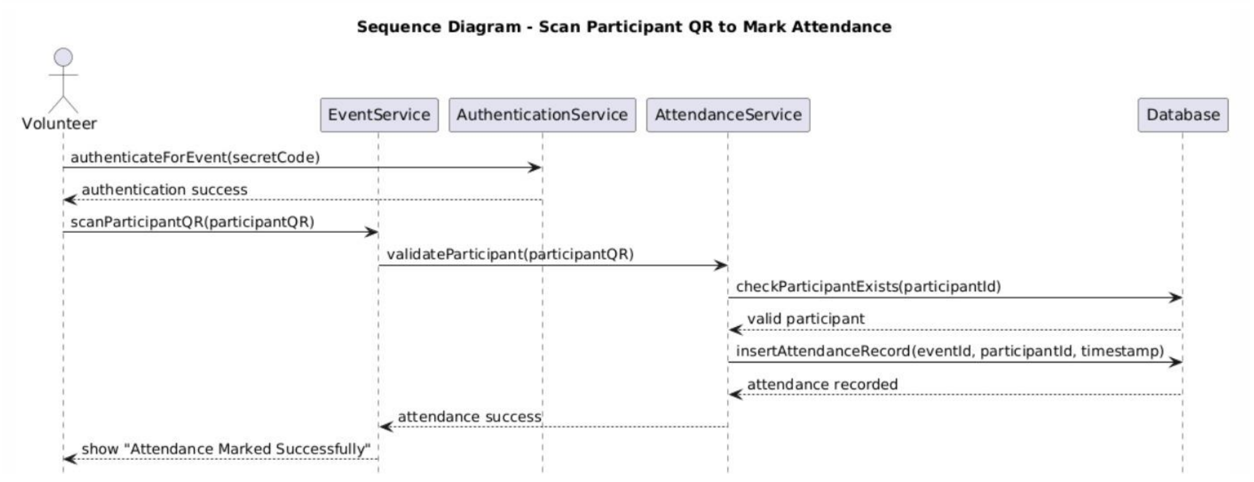


Class Diagram:



Sequence Diagram:




Sequence Diagram - Scan Participant QR to Mark Attendance
Volunteer
EventService
AuthenticationService
AttendanceService
Database
authenticateForEvent(secretCode)
authentication success
scanParticipantQR(participantQR)
validateParticipant(participantQR)
checkParticipantExists(participantId)
valid participant
insertAttendanceRecord(eventId, participantId, timestamp)
attendance recorded
attendance success
show "Attendance Marked Successfully"