# Software Design Specifications

# for

# UNISYNC

Prepared by: Team UNISYNC

Kushala K - Database Developer
SE22UCSE148

Sai Manasa - Database Developer
SE22UCSE104

Pavan Koushik - Fronted Developer
SE22UCSE326

Abhinav Pulluri - Frontend Developer
SE22UCSE006

Rishi Tez Reddy Dhava - Backend Developer
SE22UCSE323

Samyukta Gade - Backend Developer
SE22UCSE096

Rishika Gaja - Integration and Testing
SE22UCSE308

Vaibhav Naidu - Integration and Testing
SE22UCSE282

**Document Information**

| | |
|---|---|
| **Title: UNISYNC Software Design Specifications Document (SDD)** | **Document Version No: Version 0.1** |
| **Project Manager: Team UNISYNC** | **Document Version Date: 9th April, 2025** |
| **Prepared By: Team UNISYNC** | **Preparation Date: 7th April, 2025** |

**Version History**

| Version No. | Version Date. | Revised by | Description | File Name |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |

## Table of Contents

# 1. Introduction

UNISYNC is a seamless calendar and event syncing web application designed for the professors and academic staff of Mahindra University. This document outlines the software design specifications, describing the architecture, modules, use case scenarios, data models, and quality of service elements necessary to implement and maintain the application.

## 1.1 Purpose

The purpose of this document is to define the software design for UNISYNC. It provides a blueprint to developers, designers, QA testers, and stakeholders involved in the application. The document outlines the technical design and structure to be followed and serves as a reference throughout the development lifecycle.

**Target Audience:**

- Frontend & Backend Developers
- QA Engineers
- Project Stakeholders
- System Architects

## 1.2 Scope

UNISYNC aims to streamline event scheduling, announcements, and academic collaboration among faculty and staff by providing a centralized calendar platform. It enables staff to create, share, and sync their calendars with department-wide or university-wide schedules, helping to avoid scheduling conflicts and improving productivity. Students have limited access; they can only view approved public events, ensuring transparency without compromising the privacy of internal staff schedules.

## 1.3 Definitions, Acronyms, and Abbreviations

**UNISYNC**: University Calendar Sharing and Syncing System

- MU - Mahindra University
- RBAC - Role-Based Access Control
- CRUD - Create, Read, Update, Delete
- API - Application Programming Interface
- FCM - Firebase Cloud Messaging
- OAuth 2.0 - Open Authorization 2.0
- SSL/TLS - Secure Sockets Layer / Transport Layer Security
- FERPA - Family Educational Rights and Privacy Act
- UML - Unified Modelling Language

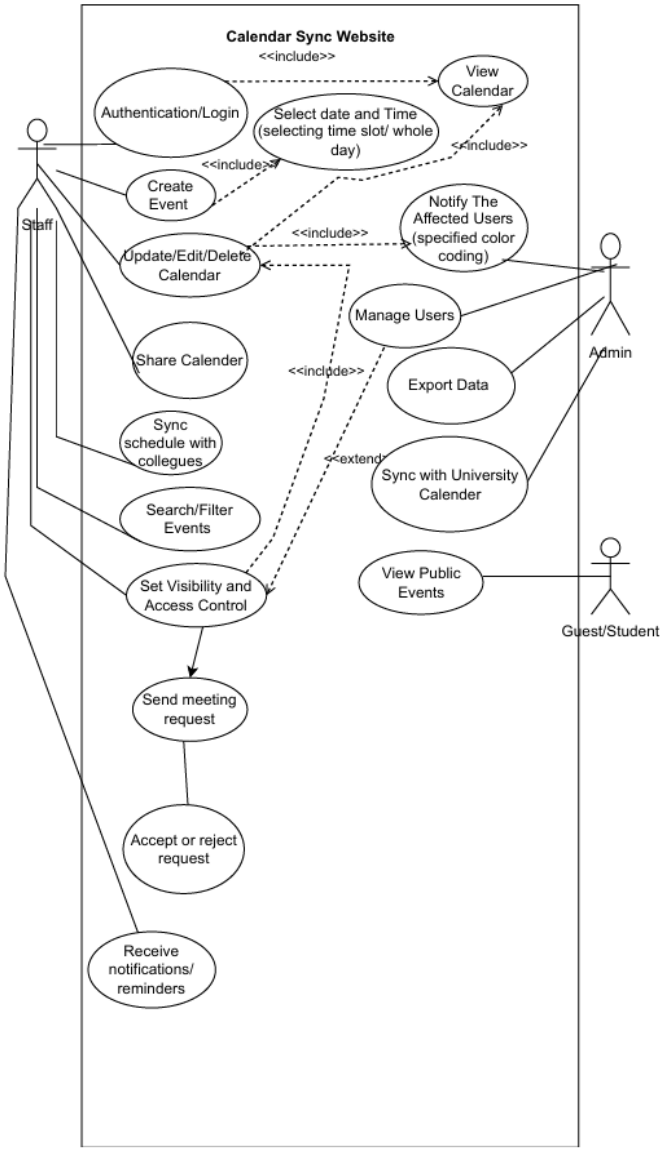
- HTML - Hypertext Markup Language
- CSS - Cascading Style Sheets
- UI - User Interface
- IT - Information Technology
- DB – Database
- CPU - Central Processing Unit
- RAM - Random Access Memory

### 1.4 References

The following documents and resources were referenced during the preparation of this SDD:

- IEEE Std 830-1998: Recommended Practice for Software Requirements Specifications
- Mahindra University IT Policies and Guidelines
- Software Engineering: A Practitioner's Approach – Roger S. Pressman
- Web development best practices and calendar synchronization techniques
- User Interface Design Principles and Guidelines

# 2. Use Case View




## 2.1 Use Case

### 1. Authentication/Login

**Description:**
Allows users (staff/admin/students) to log in securely using Google OAuth.

**Steps:**

1. User clicks on "Sign in with Google"
2. OAuth flow redirects to Google login

3. On successful login, user is authenticated
4. Session/token is created and stored

**2. View Calendar**

**Description:**
Allows users to see their calendar with events, meetings, and schedules.

**Steps:**

1. User navigates to the dashboard
2. The calendar view is rendered (daily/weekly/monthly)
3. Events are color-coded and displayed with metadata
4. User can click to view event details

**3. Select Date and Time**

**Description:**
Facilitates selection of time slots for scheduling or editing events.

**Steps:**

1. User opens event creation/edit modal
2. Chooses date via date-picker
3. Selects time range or marks as all-day
4. Confirms selection for event timing

**4. Create Event**

**Description:**
Lets users schedule new events or meetings.

**Steps:**

1. Click “+ Create Event”
2. Fill in title, description, time
3. Select attendees and access permissions
4. Save event (API call to backend)
5. Trigger notifications if needed

**5. Update/Edit/Delete Calendar**

**Description:**
Allows staff to modify or delete existing events.

**Steps:**

1. Click on an existing event
2. Choose “Edit” or “Delete”
3. Make changes in form or confirm deletion
4. Updated info synced to backend and re-rendered

**6. Notify the Affected Users (Colour Coding)**

**Description:**
Alerts relevant users (e.g., attendees) when event changes happen.

**Steps:**

1. Event is created or modified
2. System checks attendees
3. Sends notifications (email/in-app)
4. Affected users see color-coded changes on their calendar

**7. Manage Users *(Admin Only)***

**Description:**
Allows admin to manage user roles, access, and logs.

**Steps:**

1. Admin navigates to user management panel
2. Views all registered users
3. Edits roles or permissions
4. May remove inactive or unauthorized users

**8. Export Data *(Admin)***

**Description:**
Enables data export for analytics or reporting.

**Steps:**

1. Admin selects export format (CSV, JSON)
2. Chooses data scope (events, logs, usage stats)
3. Clicks “Export”
4. File is generated and downloaded

**9. Sync with University Calendar *(Admin – Optional Extension)***

**Description:**
Fetches official university calendar events to merge with UNISYNC.

**Steps:**

1. Admin connects to university API or uploads iCal feed
2. Events fetched and parsed
3. Synced into internal DB with appropriate labels
4. Displayed on user calendars

**10. Share Calendar**

**Description:**
Allows users to share their calendar with others (e.g., colleagues, students).

**Steps:**

1. Navigate to calendar sharing
2. Input email or user ID
3. Choose access level (view/edit/share)
4. Confirm to initiate sharing

**11. Sync Schedule with Colleagues**

**Description:**
Helps view overlap of schedules among peers to plan meetings/events.

**Steps:**

- Select users to compare
- View combined calendar overlay
- Identify common free slots
- Create joint event if needed

# 3. Design Overview

## 3.1 Design Goals and Constraints

**Goals**: Provide a reliable, user-friendly, secure, and scalable calendar system.
**Constraints**: Integration with third-party APIs (Google Calendar), web-based frontend, secure authentication

- Responsive, accessible design
- Modular, scalable code architecture
- Secure user data handling
- PostgreSQL for structured data
- Minimal downtime with CI/CD pipelines

## 3.2 Design Assumptions

- All users will access the system through institutional credentials.
- Staff members have internet access and use modern browsers.
- All users are part of the university system
- Users have authenticated access via institution ID
- Primary users are professors and administrative staff

## 3.3 Significant Design Packages

- **User Management**
- **Auth Module** – login, registration, session management
- **Notification Module** – Managing delivery of alerts

- **Role & Permissions Module** – RBAC handling
- **Task Module** – Task creation, updates, reminders
- **Calendar Management** - CRUD operations on events
- **Event Sharing Module**
- **API Integration** (Google Calendar)
- **Notifications Module**

## 3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

| External Application and Interface Name | Module Using the Interface | Functionality / Description |
|---|---|---|
| Google Calendar API | Calendar Sync Module | Used to fetch, create, and update calendar events from the user's Google Calendar for seamless two-way integration. |
| Microsoft Outlook Calendar API | Calendar Sync Module | Supports event syncing and scheduling from Outlook calendars into UNISYNC'S internal calendar module. |
| University LDAP Authentication System | Auth Module | Used for authenticating faculty, students, and admins via existing university credentials. |
| Notification Service (e.g., Firebase) | Notification Module | Used to send push/email notifications to users regarding upcoming events, task deadlines, and shared invites. |

## 3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

| Interface Name | Module Implementing the Interface | Functionality / Description |
|---|---|---|
| UNISYNC Event API | Events Module | Provides RESTful endpoints for creating, reading, updating, and deleting events within calendars. |
| UNISYNC Task API | Task Management Module | Exposes CRUD APIs to external tools for managing user tasks and to-dos. |
| UNISYNC Calendar API | Calendar Management Module | Enables external applications to retrieve calendar structure and schedules for integrated display. |
| UNISYNC User Access API | Authentication and User Module | Allows university systems to validate user roles and fetch access-level permissions. |

# 4 Logical View

## 4.1 Design Model

- **AuthModule** – Handles login and RBAC
- **EventManager** – Create/edit/delete events
- **CalendarSync** – Handles integration with external calendars
- **Notifier** – Sends notifications and alerts

## 4.2 Use Case Realization

### 1. Use Case: Authentication/Login

**Overview:** Users (faculty/admin) authenticate using Google OAuth to access the platform.

**High-Level Sequence:**

1. User clicks "Sign in with Google" on the frontend (React.js)
2. Google OAuth window opens
3. Google returns auth token
4. Frontend sends token to backend */auth/google*
5. Backend verifies token using Passport.js
6. User data is stored/retrieved from *users* table (MySQL)
7. JWT is issued and returned to client

**Lower-Level Flow (within Backend):**

- *AuthController -> AuthService -> GoogleStrategy -> UserService -> UserRepository*

### 2. Use Case: View Calendar

**Overview:** User accesses dashboard and views personalized calendar.

**High-Level Sequence:**

1. Frontend loads dashboard after login
2. Sends GET /calendar/events with user token
3. Backend authenticates JWT
4. EventService fetches all events the user can view (based on roles/permissions)
5. Events returned and rendered in FullCalendar UI

**Lower-Level Flow:**

CalendarController -> AuthMiddleware -> EventService -> EventRepository + PermissionService

### 3. Use Case: Create Event

**Overview:** User creates a new event by selecting time, entering details, and saving.

**High-Level Sequence:**

1. User opens event creation form
2. Fills title, date/time, invitees, access settings
3. Clicks "Save"
4. POST request sent to /calendar/events
5. Backend validates data, creates event, saves attendees and permissions
6. Sends notifications to invitees
7. Event is returned and calendar updates in UI

**Lower-Level Flow:**

- CalendarController -> EventService
    - EventRepository
    - AttendeeService
    - PermissionService
    - NotificationService

### 4. Use Case: Notify Affected Users

**Overview:** When an event is created/edited/deleted, affected users are notified.

**High-Level Sequence:**

1. EventService creates/updates event
2. Calls NotificationService with affected user_ids
3. Notifications are created in DB
4. Notification jobs pushed to queue (e.g., BullMQ)
5. Worker sends email/push via Firebase/SendGrid

**Lower-Level Flow:**

- NotificationService -> NotificationRepository -> NotificationQueue

### 5. Use Case: Share Calendar

**Overview:** A user shares their calendar with another user with view/edit rights.

**High-Level Sequence:**

1. User opens share settings from profile or calendar view
2. Enters recipient's email and permissions
3. Clicks "Share"
4. POST /calendar/share is called
5. Backend creates entry in shared_calendars table
6. Shared calendar appears on recipient’s dashboard

**Lower-Level Flow:**

- CalendarController -> SharingService -> SharedCalendarRepository

### 6. Use Case: View Public Events

**Overview:** Guest or student user views public university events.

**High-Level Sequence:**

1. Guest navigates to public events page
2. Frontend requests GET /public/events
3. Backend queries events where visibility = 'public'
4. Events are returned and rendered

**Lower-Level Flow:**

- PublicEventController -> EventRepository

### 7. Use Case: Audit Logs for Admin Tracking

**Overview:** Admin views system logs showing who performed which critical actions.

**High-Level Sequence:**

1. Admin accesses audit log view
2. Frontend sends GET /logs/audit
3. Backend authenticates request and role
4. Queries audit_logs table for actions
5. Returns paginated result

**Lower-Level Flow:**

- AdminController -> AuditLogService -> AuditLogRepository

# 5 Data View

This section describes the persistent data storage perspective of the UniSync system, including both the conceptual domain model and its corresponding physical data model. Since UniSync deals with events, users, tasks, and syncing calendars, it contains several persistent entities that are crucial to its functionality.

## 5.1 Domain Model

| Table Name | Purpose | Key Fields |
|---|---|---|
| **users** | Stores authenticated user details | `user_id` (PK), `name`, `email`, `role`, `institutional_id`, `department_id` (FK), `profile_picture`, `created_at`, `last_login` |
| **departments** | Stores academic department information | `department_id` (PK), `name`, `abbreviation`, |

| | | |
|---|---|---|
| | | faculty_incharge (FK to users.user_id) |
| **events** | Stores calendar event details | event_id (PK), title, description, start_time, end_time, created_by, location, visibility, event_type, is_recurring, recurrence_pattern, created_at |
| **event_attendees** | Stores event participants | event_id (FK), user_id (FK), status, is_organizer<br>**PK:** Composite (event_id, user_id) |
| **permissions** | Manages role-based access to events | permission_id (PK), user_id (FK), event_id (FK), access_level |
| **notifications** | Stores in-app/email/push notifications | notification_id (PK), user_id (FK), type, message, is_read, created_at |
| **tasks** | Stores to-do items and reminders | task_id (PK), user_id (FK), title, description, due_date, priority, status, is_recurring, recurrence_pattern, created_at |
| **audit_logs** | Logs user actions for transparency | log_id (PK), user_id (FK), action, target_id, timestamp, ip_address |
| **recurring_events** | Manages recurrence patterns for events | recurring_id (PK), event_id (FK), frequency, interval, end_date |
| **course_assignments** | Links professors to specific courses (used in profile views) | assignment_id (PK), professor_id (FK), course_code, course_title, semester |
| **availability_slots** | Defines available time slots for faculty | slot_id (PK), faculty_id (FK), start_time, end_time, is_booked |
| **shared_calendars** | Manages calendar sharing between users | calendar_id (PK), owner_id (FK), shared_with_id (FK), access_level, created_at |

## 5.2 Data Model (persistent data view)

### 5.2.1 Data Dictionary

#### Table : Users

| Field Name | Data Type | Constraints | Description |
|---|---|---|---|
| user_id | UUID / INT | Primary Key, Not Null | Unique identifier for each user |
| name | VARCHAR(100) | Not Null | Full name of the user |
| email | VARCHAR(100) | Unique, Not Null | User's email address |
| password_hash | TEXT | Not Null | Hashed password for authentication |
| role | ENUM | Not Null | Role: 'student', 'faculty', 'admin' |
| created_at | TIMESTAMP | Default: CURRENT_TIME | Account creation date |
| updated_at | TIMESTAMP | On update CURRENT_TIME | Last update time |

#### Table : Calendars

| Field Name | Data Type | Constraints | Description |
|---|---|---|---|
| calendar_id | UUID / INT | Primary Key | Unique calendar ID |
| user_id | UUID / INT | Foreign Key (Users) | Owner of the calendar |
| name | VARCHAR(100) | Not Null | Calendar title |
| type | ENUM | Not Null | 'university', 'personal', 'synced' |
| color_code | VARCHAR(10) | | Color tag for UI |
| is_shared | BOOLEAN | Default: FALSE | True if shared with other users |
| created_at | TIMESTAMP | | Calendar creation timestamp |

## Table : Events

| Field Name | Data Type | Constraints | Description |
|---|---|---|---|
| event_id | UUID / INT | Primary Key | Unique identifier for the event |
| calendar_id | UUID / INT | Foreign Key (Calendars) | Reference to parent calendar |
| title | VARCHAR(200) | Not Null | Event title |
| description | TEXT | | Event details |
| start_time | DATETIME | Not Null | Event start time |
| end_time | DATETIME | Not Null | Event end time |
| location | VARCHAR(255) | | Physical or virtual location |
| created_by | UUID / INT | Foreign Key (Users) | User who created the event |
| is_recurring | BOOLEAN | Default: FALSE | Whether the event repeats |
| recurrence_rule | TEXT | Nullable | Recurrence details (e.g., RRULE format) |

## Table : Tasks & To-Dos

| Field Name | Data Type | Constraints | Description |
|---|---|---|---|
| task_id | UUID / INT | Primary Key | Unique task ID |
| user_id | UUID / INT | Foreign Key (Users) | Owner of the task |
| title | VARCHAR(150) | Not Null | Task title |
| description | TEXT | | Additional notes |
| due_date | DATETIME | Nullable | Task due date/time |
| priority | ENUM | | 'low', 'medium', 'high' |
| status | ENUM | Default: ‘pending’ | 'pending', 'in-progress', 'completed' |
| created_at | TIMESTAMP | | Date task was created |

**Table : Notifications**

| Field Name | Data Type | Constraints | Description |
|---|---|---|---|
| notification_id | UUID / INT | Primary Key | Unique ID for notification |
| user_id | UUID / INT | Foreign Key (Users) | Receiver of the notification |
| message | TEXT | Not Null | Notification message |
| type | ENUM | | 'reminder', 'invitation', 'announcement' |
| is_read | BOOLEAN | Default: FALSE | Whether the user has read it |
| created_at | TIMESTAMP | | Notification creation time |

**Table : Sync Accounts**

| Field Name | Data Type | Constraints | Description |
|---|---|---|---|
| sync_id | UUID / INT | Primary Key | Unique ID for the sync configuration |
| user_id | UUID / INT | Foreign Key (Users) | Owner of the sync account |
| provider | ENUM | Not Null | 'Google', 'Outlook', etc. |
| access_token | TEXT | Encrypted | Access token for syncing |
| refresh_token | TEXT | Encrypted | Refresh token for auto-renewal |
| last_synced_at | TIMESTAMP | | Last sync timestamp |

# 6 Exception Handling

This section outlines the structured approach taken by UNISYNC to handle exceptions, ensure system stability, and provide traceability for debugging and resolution.

| Exception Name | Description |
|---|---|
| AuthError | Raised when user authentication fails due to invalid or missing credentials. |
| PermissionDenied | Triggered when a user attempts to access or modify data beyond their role. |
| ValidationError | Raised when input data does not meet the required schema or constraints. |

| Exception Name | Description |
|---|---|
| ResourceNotFound | Thrown when a requested entity (event/task/user) does not exist in the system. |
| DatabaseError | Triggered during any failure in database connection or query execution. |
| ExternalAPIError | Raised when communication with an external API (e.g., Google Calendar) fails. |
| RateLimitExceeded | Triggered when a user or IP exceeds the maximum number of API calls. |

All exceptions are logged using a centralized logging mechanism.

- **Logging Framework:** Winston or Morgan (Node.js) / Log4j (Java) / Logger (Python).
- **Log Format:**
  - Timestamp
  - Error type
  - Stack trace
  - Request context (user ID, endpoint, method)
- **Log Levels:**
  - INFO for handled flows
  - WARN for unexpected but recoverable conditions
  - ERROR for unhandled or critical failures
- **Storage:**
  - Local file system for development
  - Centralized logging system (e.g., Logstash, Elasticsearch, CloudWatch) for production

**Exception Handling Mechanisms**

1. **Try-Catch Blocks:**
   - Used around critical code (DB operations, API integrations, file operations).
2. **Global Error Handler:**
   - A middleware or service (e.g., errorHandler.js in Express or @ControllerAdvice in Spring) captures unhandled exceptions and formats responses.
3. **Client Feedback:**
   - User-friendly messages are sent to the frontend.
   - Technical messages and stack traces are hidden from users to avoid security leaks.
4. **Fallback Procedures:**
   - Retry logic for transient errors (like sync API calls).
   - Graceful degradation (e.g., load cached calendar if sync fails).
   - Auto-logout and redirect to login for expired sessions.

# 7 Configurable Parameters

This section outlines configurable parameters used across the UNISYNC application. These settings provide flexibility for deployment, tuning, and runtime adjustments without modifying core code. Each parameter can be defined via environment variables or configuration files. Parameters marked as *dynamic* can be changed during runtime without restarting the application.

| Configuration Parameter Name | Definition and Usage | Dynamic? |
|---|---|---|
| NOTIFICATION_DELAY | Time delay (in minutes) before sending a reminder for scheduled events and tasks. | Yes |
| MAX_EVENT_LENGTH | Defines the maximum allowed duration (in minutes) for any event to ensure consistency. | No |
| MAX_LOGIN_ATTEMPTS | Maximum allowed consecutive failed login attempts before the user is temporarily locked. | No |
| LOCKOUT_DURATION | Time period (in minutes) for which a user is locked out after exceeding login attempts. s | Yes |
| SENTRY_DSN | Data Source Name used for connecting the app to Sentry error tracking service. | Yes |
| ENABLE_ANALYTICS | Boolean flag to enable/disable Google Analytics tracking across the application. | Yes |
| DEFAULT_TIMEZONE | Sets the default time zone used for displaying and creating events. | No |
| EMAIL_PROVIDER_API_KEY | API key for external email provider (e.g., SendGrid) used for sending notifications. | No |
| TOKEN_EXPIRY | Sets the session token expiration time (in hours) for JWT-based authentication. | Yes |

# 8 Quality of Service

## 8.1 Availability

To ensure that the application is consistently accessible to users (faculty, students, and admins), the following design choices have been made:

1. **Stateless Backend:** The Express.js server is stateless, which makes it horizontally scalable using load balancers.
2. **Deployment Strategy:** Can be deployed using a **containerized approach (Docker)** with **orchestration support (e.g., Kubernetes)** for high availability and auto-scaling.
3. Failover & Recovery: The use of persistent cloud databases (e.g., MySQL on AWS RDS or Google Cloud SQL) ensures data redundancy, automated backups, and rapid recovery in case of failures.
4. **Graceful Degradation:** In case of non-critical service disruptions (e.g., notification system), the core calendar functionalities will continue to function.

## 8.2 Security and Authorization

Security is critical, especially with sensitive user and institutional data involved. UniSync implements a multi-layered security model:

- **OAuth 2.0 with Google Authentication:** Users must log in using institutional Gmail accounts for secure identity verification.

- **Role-Based Access Control (RBAC):** Ensures that actions like editing, deleting, or exporting calendar data are restricted based on user roles (staff, admin, student/guest).
    1. **Admin (us)**: Can manage the platform, logs, and users.
    2. **Staff**: Full access to their calendars and shared ones.
    3. **Students**: Read-only access to public/department events.
    4. OAuth2 for secure third-party integration.
    5. HTTPS encryption throughout.

## 8.3 Load and Performance Implications

UNISYNC implements robust security measures to ensure safe and authenticated access. It uses **JWT-based authentication** for secure session handling and **role-based access control (RBAC)** to manage user permissions effectively. Passwords are securely hashed using **bcrypt**, protecting user credentials against potential breaches. Additionally, an **admin interface** is provided for managing roles, monitoring access, and controlling authorization policies within the system.

## 8.4 Monitoring and Control

UNISYNC includes basic background processes and monitoring features to ensure the application runs smoothly and issues are identified early.

| Process | Purpose |
|---|---|
| Sync Handler | Regularly syncs events with external calendars like Google Calendar. |
| Notification Sender | Sends reminders or alerts to users about upcoming events. |
| Health Checker | Checks if services like database and APIs are running correctly. |

### Monitoring Metrics:

The system provides a few important values that help track its status:

| Metric | What it shows | When we act |
|---|---|---|
| Sync time | Time it takes to complete calendar sync | If it exceeds 5 seconds consistently |
| API response time | Time taken for the system to reply to users | If it slows down a lot |
| Failed Sync Count | Number of times event syncing fails | If it's growing without reducing |
| Notification Queue Size | Number of unsent messages | If many fail in a short time |

### Control Features:

- **Restart Sync/Notification Services** from the admin panel if they stop working.
- **Logs** are generated for syncs, errors, and login attempts.
- **Threshold Alerts** (e.g., for high sync failure) notify developers if something needs attention.