# Back-end Architecture
# + Data Model
# + Product Backlog

## Table of Contents

## 1. Data Model

The key adaptation for using a shared database with a tenant identifier involves adding a tenant_id column to all tables that store tenant-specific data. This ensures that data is logically partitioned and can be easily queried or manipulated on a per-tenant basis.

### 1.1. Tenant Identifier in Tables

Every table that holds tenant-specific data will include a tenant_id column and it should be indexed to optimize queries that filter data by the tenant.

### 1.2. Data Model Overview

The key entities in the system include:

1. Tenant (Organization).
2. Division_Type.
3. Division.
4. User.
5. Roles.
6. Permissions.
7. Roles_Permissions.
8. User_Roles.
9. Course.
10. Topic.
11. Content.
12. Enrollment.
13. Course_Progress.
14. Assessment.
15. Submission.
16. Calendar
17. Event
18. Event_Attendees
19. Notification.
20. Message.

## 1.3. Database Schema

### 1.3.1. Tenant (Organization) Table

| Column Name | Data Type | Constraints |
|---|---|---|
| tenant_id | UUID | Primary Key |
| name | VARCHAR | Not Null |
| subscription_status | subscription_status_enum | Not Null, Default: 'trial' |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.2. Division Type Table

| Column Name | Data Type | Constraints |
|---|---|---|
| division_type_id | UUID | Primary Key |
| name | VARCHAR | Not Null, Unique (e.g., College, Department, Class) |
| description | TEXT | Optional |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.3. Division Table

| Column Name | Data Type | Constraints |
|---|---|---|
| division_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| parent_division_id | UUID | Foreign Key (References Division) - Nullable for top-level divisions |
| division_type_id | UUID | Foreign Key (References Division Type) |
| name | VARCHAR | Not Null |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.4. User Table

| Column Name | Data Type | Constraints |
|---|---|---|
| user_id | UUID | Primary Key |

| | | |
|---|---|---|
| tenant_id | UUID | Foreign Key (References Tenant) |
| email | VARCHAR | Not Null, Unique |
| password_hash | VARCHAR | Not Null |
| first_name | VARCHAR | Not Null |
| last_name | VARCHAR | Not Null |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.5. Roles Table

| Column Name | Data Type | Constraints |
|---|---|---|
| role_id | UUID | Primary Key |
| name | VARCHAR | Not Null (e.g., Admin, Instructor) |
| description | TEXT | Optional description of the role |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.6. Permissions Table

| Column Name | Data Type | Constraints |
|---|---|---|
| permission_id | UUID | Primary Key |
| name | VARCHAR | Not Null (e.g., manage_users, manage_courses) |
| description | TEXT | Optional description of the permission |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.7. Role_Permissions Table

| Column Name | Data Type | Constraints |
|---|---|---|
| role_id | UUID | Foreign Key (References Role) |
| permission_ids | UUID[] | Foreign Key (References Permission) |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.8. User_Roles Table

| Column Name | Data Type | Constraints |
|---|---|---|
| user_id | UUID | Foreign Key (References User) |
| role_id | UUID | Foreign Key (References Role) |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.9. Course Table

| Column Name | Data Type | Constraints |
|---|---|---|
| course_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| title | VARCHAR | Not Null |
| description | TEXT | |
| instructor_ids | UUID[] | Foreign Key (References User) |
| topics | JSONB | Foreign Key (References User) |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.10. Topic Table

| Column Name | Data Type | Constraints |
|---|---|---|
| topic_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| course_id | UUID | Foreign Key (References Course) |
| title | VARCHAR | Not Null |
| content_ids | UUID[] | Foreign Key (References Content) |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.11. Content Table

| Column Name | Data Type | Constraints |
|---|---|---|
| content_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| type | VARCHAR | Not Null (e.g., video, pdf) |
| url | TEXT | Not Null |

| | | |
|---|---|---|
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.12. Enrollment Table

| Column Name | Data Type | Constraints |
|---|---|---|
| enrollment_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| user_id | UUID | Foreign Key (References User) |
| course_id | UUID | Foreign Key (References Course) |
| progress | DECIMAL | Default: 0.0 (from 0.0 to 1.0) |
| grade | DECIMAL | (from 0.0 to 1.0) |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.13. Course_Progress Table

| Column Name | Data Type | Constraints |
|---|---|---|
| course_progress_id | UUID | Primary Key |
| enrollment_id | UUID | Foreign Key (References Enrollment) |
| progress | JSONB | Default: '{}', Stores topic progress as JSON |
| completed_at | TIMESTAMP | Optional, Represents the completion time for the topic |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.14. Assessment Table

| Column Name | Data Type | Constraints |
|---|---|---|
| assessment_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| course_id | UUID | Foreign Key (References Course) |
| title | VARCHAR | Not Null |
| type | VARCHAR | Not Null (e.g., quiz, exam) |
| content | TEXT | Not Null |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

| | | |
|---|---|---|
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.15. Submission Table

| Column Name | Data Type | Constraints |
|---|---|---|
| submission_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| assessment_id | UUID | Foreign Key (References Assessment) |
| user_id | UUID | Foreign Key (References User) |
| score | DECIMAL | |
| submitted_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.16. Calendar Table

| Column Name | Data Type | Constraints |
|---|---|---|
| calendar_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| title | VARCHAR | Not Null |
| description | TEXT | |
| start_time | TIMESTAMP | Not Null |
| end_time | TIMESTAMP | Not Null |
| Location | VARCHAR | |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.17. Event Table

| Column Name | Data Type | Constraints |
|---|---|---|
| event_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| title | VARCHAR | Not Null |
| description | TEXT | |
| start_time | TIMESTAMP | Not Null |
| end_time | TIMESTAMP | Not Null |
| Location | VARCHAR | |

| | | |
|---|---|---|
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.18. Event_Attendees

| Column Name | Data Type | Constraints |
|---|---|---|
| event_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| user_id | UUID[] | Foreign Key (Reference User) |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.19. Notification Table

| Column Name | Constraints | Constraints |
|---|---|---|
| notification_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| user_id | UUID | Foreign Key (References User) |
| title | VARCHAR | Not Null |
| message | TEXT | |
| is_read | BOOLEAN | Default: FALSE |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

### 1.3.20. Message Table

| Column Name | Data Type | Constraints |
|---|---|---|
| message_id | UUID | Primary Key |
| tenant_id | UUID | Foreign Key (References Tenant) |
| sender_id | UUID | Foreign Key (References User) |
| receiver_id | UUID | Foreign Key (References User) |
| content | TEXT | Not Null |
| sent_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| is_read | BOOLEAN | Default: FALSE |
| created_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |
| updated_at | TIMESTAMP | Not Null, Default: CURRENT_TIMESTAMP |

## 1.4. Schema Explanation

Using the previously mentioned schema we can create a flexible and extensible division schema that accommodates the hierarchical structure typical in educational institutions like universities and schools. The schema will allow for the definition of various types of divisions (e.g., colleges, departments, classes), and will be designed to be adaptable, so new types of divisions can be added as needed.

### 1.4.1. Division Schema

#### 1.4.1.1. Overview

- **Organization (University/School)**: This is the top-level entity that represents the educational institution (e.g., a university or a school).
- **Division**: This represents any organizational unit within an institution, such as a college, department, or class. Divisions are hierarchical, meaning they can contain subdivisions (e.g., a department can be within a college).
- **Division Type**: This defines the nature of a division, such as a college, department, specialization, class, or any other custom type you might add in the future.

#### 1.4.1.2. Entities and Their Relationships:

- **Organization to Division**: An organization can have multiple top-level divisions (e.g., colleges or schools).
- **Division to Subdivision**: Divisions can have multiple subdivisions, creating a hierarchy (e.g., a college has departments, a department has specializations).

### 1.4.2. Roles, Permissions, Role-Permissions and User_Roles Schemas:

Let's break down the concepts of **roles**, **permissions**, and the overall **authorization workflow** within the context of Role-Based Access Control (RBAC) to clarify how they work together in a multi-tenant system.

#### 1.4.2.1. Roles

- **Definition**: A role is a collection of permissions that define what actions a user can perform within the system. Roles are typically named based on the responsibilities they represent, such as "Saas_Admin", "Orgnization_Admin", "Division_Admin", "Instructor", "Student", "Custom_Role1," etc.
- **Purpose**: Roles simplify the assignment of permissions to users by grouping related permissions together. Instead of assigning individual permissions to each user, you assign a role that encapsulates all the required permissions for that role.
- **Example**:
  - **Admin Role**: This might include permissions to manage users, manage courses, and access all areas of the application.
  - **Instructor Role**: This might include permissions to create and manage course content, view student progress, and grade assignments.

#### 1.4.2.2. Permissions

- **Definition**: A permission is a specific action or operation that a user can perform within the system, such as "create_user," "delete_course," or "view_reports."
- **Purpose**: Permissions define the granular level of control over what actions can be performed. They are the building blocks of roles.
- **Example**:
  - **manage_users**: Permission to create, update, and delete users.
  - **manage_courses**: Permission to create, update, and delete courses.
  - **view_reports**: Permission to access and view system reports.

#### 1.4.2.3. Role_Permissions

- **Definition**: The Role_Permissions table links roles with permissions, establishing what permissions are included in each role.
- **Purpose**: This relationship allows us to define which actions are available to users with specific roles.
- **Example**: If the "Admin" role is linked to the manage_users, manage_courses, and view_reports permissions, any user with the "Admin" role can perform these actions.

#### 1.4.2.4. User_Role

- **Definition:** The User_Role table maps users to their corresponding roles, linking users in the Users table with roles in the Role table.
- **Purpose:** This table allows the system to determine the access level of each user by associating them with specific roles. The roles, in turn, define the permissions and actions available to the user within the system.
- **Example:** If a user with user_id = ‘ID’ is mapped to the "Admin" role, this user will inherit all the permissions associated with the "Admin" role, such as manage_users, manage_courses, and view_reports.

## 2. Backend Architecture

The backend architecture will follow a **microservices** approach, allowing for scalability, maintainability, and independent deployment of different services. The architecture will also be **RESTful** with the possibility of incorporating **GraphQL** for more complex querying requirements.

### 2.1. Architecture Components

- **API Gateway**: Handles routing, authentication, and rate limiting.

- **Authentication Service**: Manages user registration, login, and token-based authentication (JWT).
- **User Service**: Manages user profiles, roles, and permissions.
- **Tenant Service**: Handles tenant creation, management, and subscription.
- **Course Service**: Manages course creation, enrollment, and content.
- **Content Service**: Handles uploading, managing, and serving educational content.
- **Assessment Service**: Manages assessments, submissions, and grading.
- **Notification Service**: Sends notifications to users and manages the notification log.
- **Messaging Service**: Handles internal user messaging within the platform.
- **Event Service:** Manages events, including scheduling and participation.
- **Calendar Service:** Manages, view calendar for specific user or division (e.g. class).

## 2.2. Backend Design (Microservices Architecture)

The backend will be based on a microservices architecture, with each service responsible for a distinct domain (e.g., authentication, user management, course management). This design allows services to be independently deployed, scaled, and maintained, which is crucial for a SaaS platform with multiple tenants.

### 2.2.1. Key Characteristics:

- **Loose Coupling**: Services communicate via RESTful APIs, reducing dependencies between services.
- **Scalability**: Individual services can be scaled horizontally to handle increased load.
- **Resilience**: Failures in one service do not impact the entire system. Services can be restarted or replaced independently.

### 2.2.2. API Gateway

The API Gateway will serve as the entry point for all client requests. It handles:

- **Authentication and Authorization**: Verifying user tokens and permissions.
- **Routing**: Directing requests to the appropriate microservices.
- **Rate Limiting and Throttling**: Protecting the backend from abuse or overload.

### 2.2.3. Authentication & Authorization

- **JWT (JSON Web Tokens)** will be used for authentication. Once a user logs in, a token is issued, which the client includes in the header of subsequent requests.

- **Role-Based Access Control (RBAC)** will enforce permissions based on user roles (e.g., Admin, Instructor, Student, custom roles). Each role has associated permissions that dictate access to certain API endpoints.

## 3. API Endpoints

The API endpoints will be designed following RESTful principles, with clear and consistent URL structures. Each endpoint will adhere to the RBAC model to ensure that only authorized users can access specific functionalities. The endpoints will be organized by service, with the following proposed structure:

### 3.1. Authentication Service

- **POST /auth/register**: Register a new user.
- **POST /auth/login**: Authenticate a user and return a JWT.
- **POST /auth/refresh-token**: Refresh the user's JWT.
- **POST /auth/logout**: Invalidate the user's session.

### 3.2. Tenant Service

- **GET /tenants**: List all tenants (SaaS Admin only).
- **GET /tenants/{id}**: Get a tenant's details.
- **POST /tenants**: Create a new tenant (SaaS Admin only).
- **PUT /tenants/{id}**: Update a tenant's details.
- **DELETE /tenants/{id}**: Soft delete a tenant.

### 3.3. User Service

- **GET /users**: List all users (Admin only).
- **GET /users/{id}**: Get a user's details.
- **POST /users**: Create a new user (Admin only).
- **PUT /users/{id}**: Update a user's details.
- **DELETE /users/{id}**: Soft delete a user.

### 3.4. Course Service

- **GET /courses**: List all courses within the tenant.
- **GET /courses/{id}**: Get details of a specific course.
- **POST /courses**: Create a new course.
- **PUT /courses/**: Update a course's details.
- **DELETE /courses/{id}**: Soft delete a course within the tenant.

### 3.5. Content Service

- **GET /contents**: List all contents within a tenant.
- **GET /contents/{id}**: Get details of specific content.
- **POST /contents**: Upload new content (e.g., video, PDF).
- **PUT /contents/{id}**: Update content details.
- **DELETE /contents/{id}**: Soft delete content.

### 3.6. Assessment Service

- **GET /assessments**: List all assessments within a tenant.
- **GET /assessments/{id}** Get details of a specific assessment.
- **POST /assessments**: Create a new assessment.
- **PUT /assessments/{id}**: Update an assessment's details.
- **DELETE /assessments/{id}**: Soft delete an assessment.
- **POST /assessments/submit**: Submit a user's assessment.

### 3.7. Enrollment Service

- **GET /enrollments**: List all enrollments within a tenant.
- **GET /enrollments/course/{id}**: List all enrollments within a course.
- **GET /enrollments/{id}**: Get details of a specific enrollment.
- **POST /enrollments**: Enroll a user in a course.
- **PUT /enrollments/{id}**: Update enrollment details, such as progress or grade.
- **DELETE /enrollments/{id}**: Unenroll a user from a course (soft delete).

### 3.8. Notification Service

- **GET /notifications**: List all notifications for the logged-in user.
- **GET /notifications/{id}**: Get details of a specific notification.
- **POST /notifications**: Send a new notification to a user or group.
- **PUT /notifications/{id} /read**: Mark a notification as read.
- **DELETE /notifications/{id}**: Soft delete a notification.

### 3.9. Messaging Service

- **GET /messages**: List all messages for the logged-in user.
- **GET /messages/{id}**: Get details of a specific message.

- **POST /messages**: Send a new message to another user.
- **PUT / messages/{id}**: edit message
- **PUT / messages/{id}/read**: Mark a message as read.
- **DELETE /messages/{id}**: Soft delete a message.

### 3.10. Event Service

- **GET /events**: List all events within a tenant.
- **GET /events/{id}**: Get details of a specific event.
- **POST /events**: Create a new event.
- **PUT /events/{id}**: Update an event's details.
- **DELETE /events/{id}**: Soft delete an event.
- **POST /events/{id}/join**: Register a user for an event.
- **GET /events/{id}/attendees**: Get a list of attendees for an event.

### 3.11. Calendar Service

- **GET /calendar**: get calendar events for logged-in user.
- **GET /calendar/{id}**: Get details of a specific calendar event.
- **POST /calendar**: Create an event in a calendar.
- **PUT /calendar/{id}**: Update a calendar event's details.
- **DELETE /calendar/{id}**: Soft delete a calendar event.
- **GET /calendar/division/{id}**: Get calendar events of a specific division (e.g. courses and exams for a class or a college).

## 4. User Stories

### 4.1. SaaS Administrator

#### 4.1.1. Manage Role

- **User Story**: As a SaaS Administrator, I want to manage roles (Create, Edit and delete) to control access level based on different responsibility, only assigns high level roles for each tenant, so that tenants can access and manage their tenant hierarchy.

- **Acceptance Criteria**
  1. **Create Role Record**
     - SaaS administrator can “Create” new role record.
     - System requires mandatory fields (Name).

- System validates role name is unique.
- System validates mandatory fields and displays an error message if misses or incorrect format occurs.
- When submitted, system adds new record to roles.
- System prompts a message “Rols added successfully”.
- Role became available for assignment in system.
- System redirects to roles list.

2. **View Role List**
    - SaaS administrator can view roles list.
    - List columns are (Name, Description, Actions)
    - Each list record contains actions (Edit, Details, Add Permissions, delete).

3. **Edit Role Record**
    - SaaS administrator can “Edit” an existing record in roles list.
    - System displays role’s existing information in editable form.
    - SaaS administrator can change fields (Name, description).
    - System validates mandatory fields and displays an error if misses or incorrect format occur.
    - When submitted successfully, system saves changes and updates role’s record.
    - System prompts message “Role updated successfully”.
    - Existing users with the role, automatically inherits updated information and permissions.
    - System redirects to roles list.

4. **Details Role Record**
    - SaaS administrator can select a role from list to view its “Details”.
    - System displays existing information in locked fields.

5. **Delete Role Record**
    - SaaS administrator can “Delete” role from roles list.
    - System prompts a message “Are you sure you want to delete this role?”.
    - When confirmed, then system deletes role record.
    - System reloads new role list.

6. **Add permissions**
    - SaaS administrator can “add permissions” to existing roles in role list.
    - System displays all permissions in a role permissions form.
    - SaaS can select all permissions related to the role.
    - SaaS can deselect permissions assigned to the role.
    - When submitted, system saves all selected permissions to that role.
    - System prompts a message “permissions added to the role successfully”.

- System reloads roles list.

### 4.1.2. Manage Permission

- **User Story:** As a **SaaS Administrator,** I want to manage permissions (Create, Edit, Delete), so that I can control which actions and features are available to different roles int he system.

- **Acceptance Criteria**
  1. **View Permission List**
     - SaaS administrator can view permissions list.
     - List columns are (Name, Permission Name)
     - Each list record contains actions (Edit, Details, Delete).

  2. **Create Permission Record**
     - SaaS administrator can "Create" a new permission.
     - System requires mandatory fields (Name, URL).
     - System validates permission name uniqueness and displays error message if permission name already exists.
     - System validates mandatory fields and display error message if misses or incorrect format occurs.
     - When submitted successfully, system added new record to permissions.
     - System prompts message "Permissions added successfully".
     - System redirects to Permissions List.

  3. **Edit Permission Record**
     - SaaS administrator can "Edit" an existing permission from list.
     - System displays permission's existing information in editable form.
     - SaaS can change mandatory fields (Name, URL).
     - System validates permission name uniqueness and displays error message if permission name already exists.
     - System validates mandatory fields and display error message if misses or incorrect format occurs.
     - When submitted successfully, system saves changes and update record.
     - System prompts message "Permissions updated successfully".
     - All roles inherit this permission become updated.
     - System redirects to Permissions List.

  4. **Details Permission Details**
     - SaaS administrator can select an existing permission to view its "Details".
     - System displays existing information in locked fields.

  5. **Delete Permission Record**
     - SaaS administrator can "Delete" permission from list.

    - System prompts a message "Are you sure you want to delete this permission?".
    - When confirmed, then system deletes permission record.
    - All roles inherit this permission become disabled.
    - System reloads new role list.

#### 4.1.3. **Manage Tenant**

- **User Story:** As a **SaaS Administrator**, I want to manage tenant record (Create, Edit, Delete) so that I can onboard new clients.
- **Description:**
  The SaaS Administrator needs the ability to onboard new tenant organizations, which involves setting up their accounts, assigning initial settings, and ensuring they have access to all necessary features of the LMS. This feature will also allow for the management of existing tenants, including updating their information, monitoring their usage, and addressing any issues related to their accounts.

- **Acceptance Criteria:**

### 4.2. Create Tenant Record

- SaaS Administrator can open a "Tenant Form".
- The form requires mandatory fields (Tenant Name, Location details)
- The system validates mandatory fields and displays an error message if missing or incorrect format occurred.
- When submitted successfully, tenant status is immediately active and date is current date, then record is added to the system
- An email is sent to the primary contact of new the tenant, when account is created or updated, containing necessary credentials to access their system.
- A success message prompted "Tenant Record is added successfully".
- System redirects to tenant list.

1. **View Tenant List**
    - SaaS Administrator can view a list of Tenant
    - The list columns contain (Tenant Name, Status, Location and actions).
    - Each List's record has different actions (Edit, Details, Delete).

2. **Edit Tenant Record**
    - SaaS Administration can "Edit" an existing tenant from the tenant list.
    - The system displays tenant's existing information in editable form (Tenant Name, Location Information, Status).
    - CreatedAt field is a locked field.

- SaaS Administrator can change any non-locked fields.
- Tenant status can be (deactivate, Suspend, activate).
- System validates required fields, displays an error message if missing or incorrect format occurred.
- When submitted form successfully, system saves changes and update tenant's record.
- System prompts a message "Tenant record updated successfully".
- System redirects to Tenant list.

3. **Delete Tenant Record**
   - SaaS Admins can "delete" an existing tenant's record from tenant list
   - System prompts a confirmation message "Are you sure you want to delete this tenant?"
   - Only if confirmed, the tenant record is permanently deleted from tenant list.
   - System prompts a message "Tenant removed successfully"
   - System reloads new tenant list.

4. **View Detail Tenant Record**
   - SaaS Administrator can select a tenant from tenant list to view its "details".
   - System displays all existing tenant's information in locked field

### 4.2.1. Manage User Account

- **User Story:** As a SaaS administrator, I want to manage user account (Create, Edit and Delete), So that I can assign high level roles, then users will be able to access and manage their tenant.

- **Acceptance Criteria:**

1. **Create User Account Record**
   - SaaS administrator can "create" a new user account.
   - System requires mandatory fields (Tenant, FirstName, LastName, Email, Password).
   - System displays default option "Organization Administrator" for mandatory field role.
   - System validates mandatory fields and displays an error message if incorrect format or misses a mandatory field.
   - When submitted successfully, then record is added to the system.
   - System prompts a message "User Account Added Successfully".

2. **View User Account List**
   - SaaS Administrator can "view" users Accounts list.
   - A list contains columns (Full Name, Tenant, Role, Actions).
   - Each list record contains actions (Edit, Details, Delete).

3. **Edit User Account Record**
   - SaaS Administrator can "Edit" an existing record from user account list.
   - System displays an existing user account's information in an editable form.
   - System requires mandatory fields (Tenant, FirstName, LastName, Email, password).

- System validates mandatory fields and displays an error message if missing or incorrect format occur.
- When submitted successfully, a message prompts “User Account Updated Successfully”.
- System redirects to User Account List.

4. **Details User Account Record**
   - SaaS administrator can select user account to view its “Details”.
   - System displays existing user account information in locked fields.

5. **Delete User Account Record**
   - SaaS administrator can “Delete” an existing record from user account list.
   - System prompts message “Are you sure you want to permanently delete this record?”.
   - Once confirmed, then system delete the record from system.
   - System prompts a message “Tenant removed successfully”.
   - System reloads new user account list

### 4.2.2. Manage Subscription

- **User Story**: As a **SaaS Administrator**, I want to manage tenants' subscription (create, edit, delete), so that I can control which tenant have active access to LMS system according to their plan and billing cycle.

- **Acceptance Criteria:**
  1. **Create Subscription Record**
     - SaaS administrator can create a “subscription form”.
     - System requires mandatory fields (Tenant, plan-type, billing cycle, start-date)
     - System validates that tenant does not have an already active subscription.
     - System calculates end-Date, based on billing cycle and start-date, then displays result in a locked field.
     - System calculates cost based on plan’s price, then displays result in a locked field.
     - System validates all mandatory fields and displays an error message if misses a mandatory field.
     - When submitted successfully, system sets status to active and added a new record.
     - System prompts a success message “Tenant’s Subscription added successfully”.
     - System redirects to subscription list.
     - System takes immediate effect based on plan-type.

2. **View Subscription List**
    - SaaS administrator can view subscription list (Tenant, plan-type, Status, Actions).
    - Each list record has actions (Edit, Details, Delete).

3. **Edit Subscription Record**
    - SaaS administrator can "Edit" an existing tenant's subscription record.
    - System displays tenant's subscription existing information in editable form.
    - System requires mandatory fields (Tenant, plan-type, billing cycle, start-date, status).
    - SaaS administrator can change tenant, status, Plan-type, Bill cycle and Start date.
    - System calculates end date and cost and displays results in locked fields.
    - Status field can be (Trail, Active, cancel).
    - System validates and can displays an error message if a mandatory field is missing.
    - When submitted successfully, system saves changes and update record.
    - System prompts a message "Tenant's subscription updated successfully".
    - System redirects to Subscription list.

4. **Detail Subscription Record**
    - SaaS administrator can select tenant's subscription details from list.
    - System displays subscription existing information in locked fields.

5. **Delete Subscription Record**
    - SaaS administrator can "Delete" a tenant's subscriptions from the list.
    - System displays a prompt "Are you sure want to delete this tenant's subscription?"
    - Once confirmed, system prompts "Tenant's name subscription deleted successfully".
    - System reloads new subscription list.

## 4.3. Organization Administrator

- **User Story:** As an **Organization Administrator**, I want to create and manage users within my organization so that I can control access to the LMS.
- **Description:** The Organization Administrator needs to be able to manage all users within their tenant, including creating new user accounts, updating user information, and assigning roles and permissions. This functionality is crucial for maintaining control over who can access different parts of the LMS and what they can do within the system.
- **Acceptance Criteria:**
    - The Organization Administrator can create a new user by filling out a form with details such as name, email, and role.
    - The Organization Administrator can update user information, including roles and permissions.

  - The Organization Administrator can deactivate or delete a user account, preventing further access to the LMS.
  - An email invitation is sent to the new user with login instructions when their account is created.
  - The Organization Administrator can view a list of all users within their organization, with options to filter and search.

### 4.4. Instructor User Stories

- **User Story:** As an **Instructor**, I want to create and upload course content so that students can access learning materials.
- **Description:** The Instructor needs the ability to create courses and upload various types of content, including videos, PDFs, and quizzes. This feature allows instructors to structure their courses and provide the necessary learning materials for their students. The content management system should be user-friendly and support different types of media.
- **Acceptance Criteria:**
  - The Instructor can create a new course by providing a course title, description, and structure.
  - The Instructor can upload different types of content, such as videos, PDFs, and quizzes, to the course.
  - The Instructor can organize content into modules or sections within the course.
  - The content is immediately accessible to enrolled students once published.
  - The Instructor can update or delete course content at any time.

### 4.5. Student User Stories

#### 4.5.1. User Story 1

- **User Story:** As a **Student**, I want to browse and enroll in courses so that I can access learning materials.
- **Description:** The Student needs to be able to search for available courses, view details, and enroll in those that interest them. Once enrolled, students should be able to access all the learning materials, track their progress, and complete assessments. The enrollment process should be straightforward, allowing students to start learning quickly.
- **Acceptance Criteria:**
  - The Student can browse a list of available courses, with options to filter and search.
  - The Student can view detailed information about each course, including content, instructor details, and course duration.
  - The Student can enroll in a course with a single click.

  - Once enrolled, the Student gains immediate access to the course content.
  - The Student can track their progress through the course, including viewing completed and remaining modules.

**4.5.2. User Story 2**

- **User Story:** As a **Student**, I want to view my class schedules, course deadlines, and events on a calendar so that I can manage my time effectively.
- **Description:** The Student needs an easy way to keep track of all their academic activities within the LMS. A calendar view will allow students to see upcoming classes, assessment dates, and any events they are registered for. This helps in planning and avoiding missed deadlines or classes.
- **Acceptance Criteria:**
  - The Student can browse a list of available courses, with options to filter and search.
  - The Student can access a calendar view showing all scheduled classes, assessments, and events.
  - The calendar allows filtering by course or type of activity (e.g., only show assessments).
  - The Student receives notifications for upcoming activities based on their calendar.
  - The Student can sync their LMS calendar with external calendars (e.g., Google Calendar).
  - The Student can view details of each event, including time, location, and description.

## 5. Product Backlog

The main sysem features are:

### 5.1. [High] Tenant Management

- Create and manage tenant organizations.
- Suspend or deactivate tenant accounts.
- Configure global settings for all tenants.

### 5.2. [High] User Authentication and Role Management

- Develop user registration and login.
- Implement JWT-based authentication.
- Implement RBAC for managing access control.

### 5.3. [High] Course Management

- Create and manage courses.
- Upload course content.
- Schedule classes and assessments.

**5.4. [High] Calendar and Scheduling Management**

Implement a unified Calendar entity for classes, courses, assessments, and events.

Allow students to view their schedules in a calendar format.

Enable instructors and administrators to add, edit, and manage events on the calendar.

Integrate the calendar with the notification system for reminders.

Implement synchronization with external calendar applications (e.g., Google Calendar).

**5.5. [High] Notification System**

- Develop a system for sending notifications to users.
- Implement read/unread status tracking.

**5.6. [High] Assessment and Grading**

- Develop assessment creation and management.
- Implement grading and feedback mechanisms.

**5.7. [Medium] Messaging System**

- Implement internal messaging between users.
- Develop message history and search functionality.

**5.8. [Medium] Event Management**

- Implement event creation and scheduling.
- Develop event registration for users.

**5.9. [Medium] Reporting and Analytics**

- Develop reporting tools for instructors and admins.
- Implement dashboards for tracking student progress.

**5.10. [Low] Content Recommendations**

- Develop algorithms for recommending courses to students.

**5.11. [Low] Third-Party Integrations**

- Integrate with external tools (e.g., Google Drive, Zoom).
- Implement API for third-party developers.

## 6. System Features Divided into Sprints

### 6.1. Sprint 1: Core Functionality and Authentication

- **Features:**
  - Implement tenant management (create, manage, deactivate tenants).
  - Implement user registration, login, and JWT-based authentication.
  - Implement basic user and role management with RBAC.
  - Secure API endpoints based on roles and permissions.
- **Success Criteria:**
  - Users can register and log in to the system.
  - JWT tokens are issued upon login and are required for subsequent requests.
  - Students can browse available courses with filtering options.
  - The system correctly assigns roles to users upon registration.

### 6.2. Sprint 2: Course and Content Management

- **Features:**
  - Implement course creation and management.
  - Allow instructors to upload and manage course content.
  - Implement basic course enrollment for students.
  - Ensure content is accessible and downloadable for students.
- **Success Criteria:**
  - Admins and Instructor can create and manage courses.
  - Instructor can upload course materials.
  - Students can enroll in and access courses

### 6.3. Sprint 3: Assessment and Grading

- **Features:**
  - Assessment Creation (Quizzes, Exams)
  - Grading System for Assessments
  - Student Feedback Mechanism
- **Success Criteria:**
  - Instructors can create assessments and assign them to specific courses.
  - Grading is accurate and reflects in the student's course progress.
  - Students receive feedback on their assessments from instructors.

### 6.4. Sprint 4: Notifications and Messaging

- **Features:**
  - Notification System for Course Updates and Assessments

    - Messaging System for Student-Instructor Communication
    - Read/Unread Status for Messages and Notifications
- **Success Criteria:**
    - Notifications are sent to users for course updates and assessments.
    - The messaging system is functional, allowing communication between students and instructors.
    - Users can see the read/unread status for both messages and notifications.

### 6.5. Sprint 5: Event Management, Calendar Management and Reporting

- **Features:**
    - Event Creation and Scheduling for Courses
    - Introduce Calendar entity and allow instructors to schedule classes on the course calendar.
    - Event Registration for Students
    - Reporting Tools for Instructors and Admins
- **Success Criteria:**
    - Instructors can create events and schedule them within the course.
    - Instructors can schedule classes on the calendar, and students can view their schedules.
    - Students can register for events and see them in their course schedule.
    - Instructors and admins can generate reports on course performance and student progress.