Group06

VocaBoost
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
23/06/2025	1.0	Initial Software Architecture Document creation	All team members

Table of Contents

1. Introduction **5**
1.1 Purpose 5
1.2 Scope 5
2. Architectural Goals and Constraints **5**
3. Use-Case Model **7**
4. Logical View **8**
4.1 Component: View 9
4.1.1 Partials 9
4.1.2 Home Page 10
4.1.3 Account 11
4.1.4 Authentication 11
4.1.5 Statistic 12
4.1.6 Vocabulary Management 13
4.1.7 Learning Method 14
4.1.8 Classroom Management 15
4.1.9 Administration 17
4.2 Component: Route 19
4.3 Component: Middleware 20
4.3.1 Responsibilities 20
4.3.2 Class diagram 20
4.3.3 Key Middleware Classes 20
4.3.4 Detailed Method Descriptions 20
4.4 Component: Controller 26
4.4.1 Responsibilities 26
4.4.2 Class diagram 27
4.4.3 Key Controller Classes 27
4.4.4 Detailed Method Descriptions 27
4.5 Component: External Services 30
4.5.1 Google OAuth 31
4.5.2 Google Gemini service 31
4.5.3 Email Notification System 32
4.6 Component: Model 32
4.6.1 Services 32
4.6.2 Data Structures 35
4.6.3 Repositories 42
4.7 Component: Database 46
4.7.1 User management 47

4.7.2 Vocabulary Management 48
4.7.3 Revision and Progress Tracking 50
4.7.4 Classroom and Assignment 52
4.7.5 System Utilities and Notifications 53
5. Deployment 54
6. Implementation View 54

Software Architecture Document

1. Introduction

1.1 Purpose

This document describes the overall software architecture of the VocaBoost system, serving as the primary reference for the development team, lecturers, and teaching assistants. The architecture follows the Model-View-Controller (MVC) pattern enhanced with a layered architecture approach.

1.2 Scope

- Frontend – React 19 + Vite.
- Backend – Node.js 20 + Express 5 REST API.
- Database – Supabase (PostgreSQL).
- External services – Google OAuth 2.0, Google Gemini AI, SMTP (Gmail).
- Users – Guests, Learners, Teachers, Admins.

2. Architectural Goals and Constraints

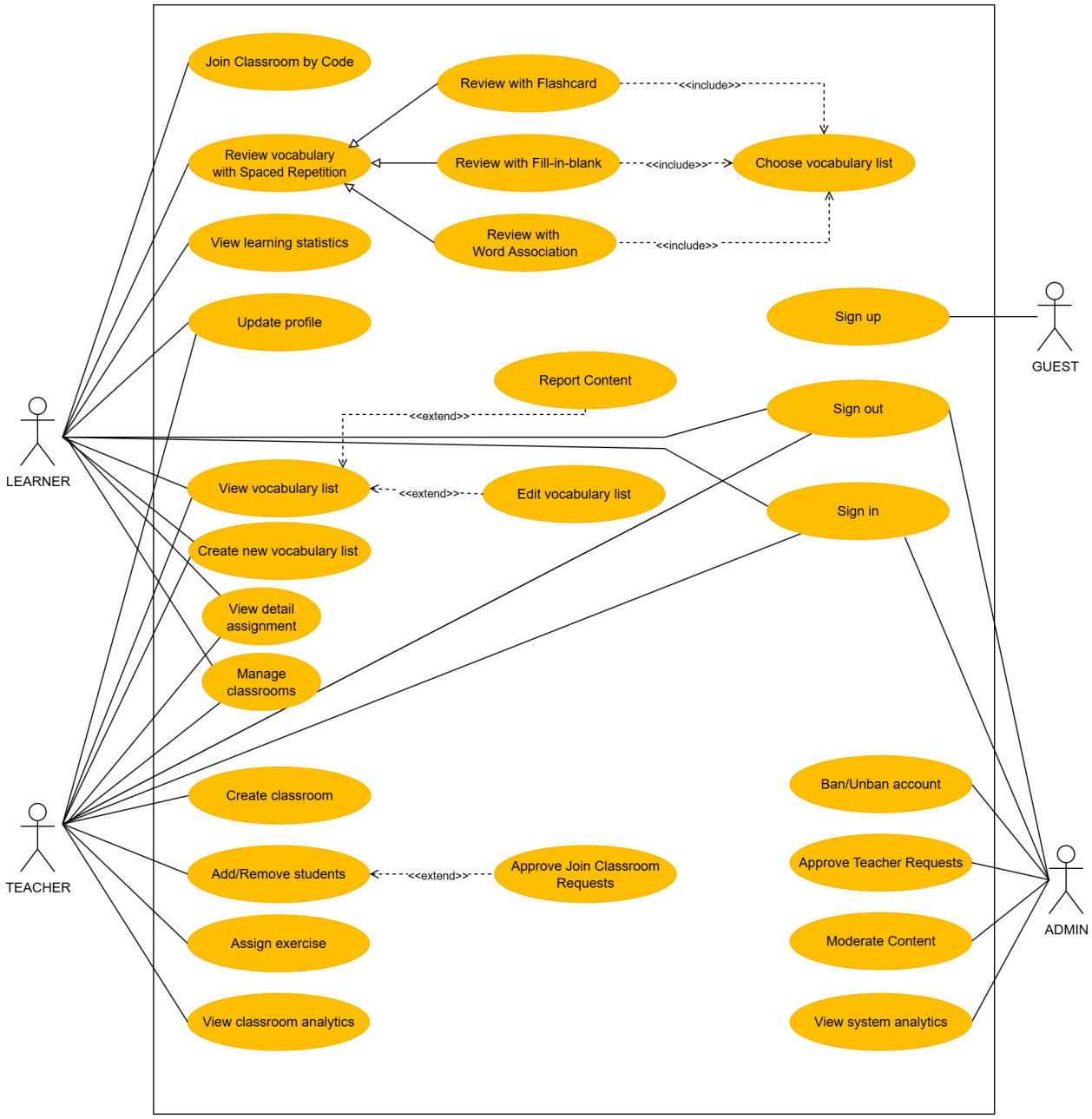
Architectural Goals

- Maintainability and Extensibility: The system must be designed with clear separation of concerns to facilitate future enhancements and modifications by team members with varying experience levels.
- User Experience Optimization: Architecture must support responsive, interactive learning experiences with minimal latency for vocabulary review sessions and real-time feedback.
- Scalability for Educational Use: System should handle concurrent users in classroom environments while maintaining performance for individual learning sessions.
- Data Integrity and Security: Robust data protection for user learning progress, personal information, and classroom data with proper authentication and authorization mechanisms.

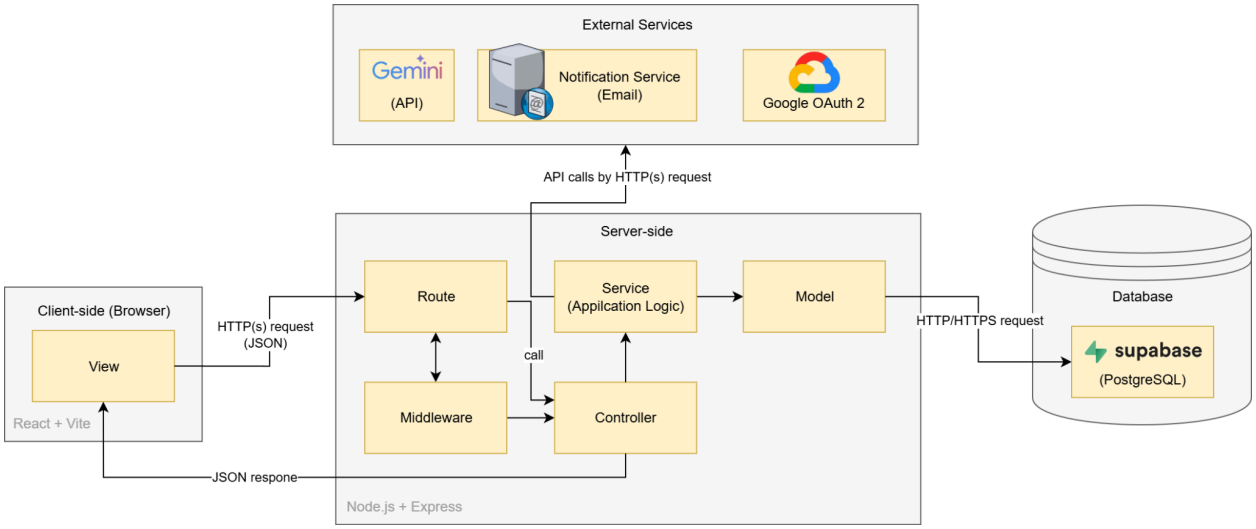
Technical Constraints

- Technology Stack Limitations: Must use React for frontend, Node.js with Express for backend, and Supabase (PostgreSQL) for database management within free tier limitations.
- Development Timeline: 12-week project duration requires architectural simplicity balanced with educational learning objectives.
- Team Experience Level: Architecture must accommodate team members new to web development while promoting best practices and learning opportunities.
- External Dependencies: Integration with Google Gemini AI API for example sentence generation and email services for notifications, both subject to free tier quotas.
- Deployment Constraints: Single-server deployment model without distributed system complexity, suitable for academic project scope.

3. Use-Case Model



4. Logical View



Architecture Overview

The system employs MVC architecture extended with Layered Architecture, comprising three main layers:

- **Presentation Layer:** View
- **Business Logic Layer:** Route → Middleware → Controller → Service
- **Data Access Layer:** Model → Database

Component Breakdown

Component	Responsibility	Public Interfaces
View (React)	SPA UI, client-side routing, state (React Query).	`/api/**` REST calls.
Route	Maps HTTP paths to controllers using Express routers.	`GET /lists/:id`, `POST /auth/login`, …
Middleware	Authentication, validation, rate-limiting, error handling.	`authValidator.register`, …
Controller	Orchestrates request flow; thin delegates.	`VocabularyController.createList()`
Service	Business logic, Spaced Repetition scheduling, AI proxy.	`SpacedRepetitionService`, `AIService`

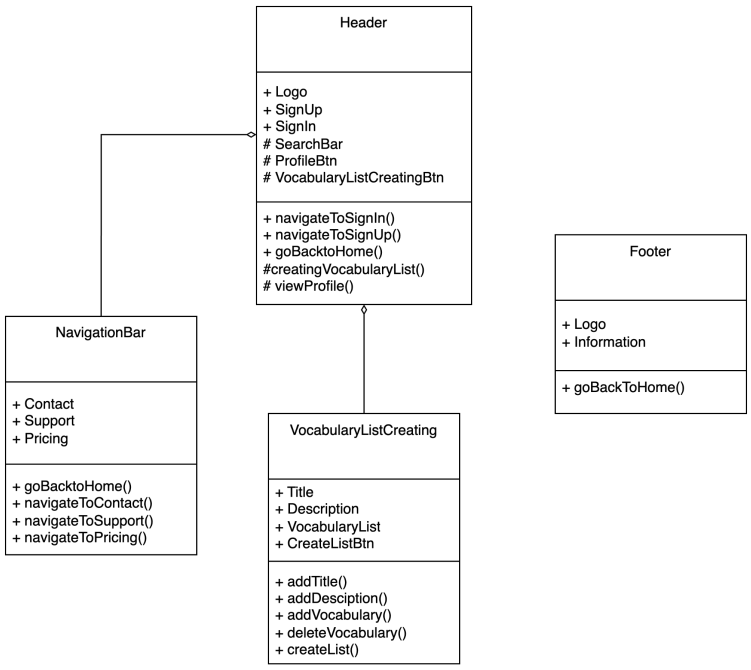
Model/Repository	Data access via Supabase JS.	`VocabularyRepo.findById(id)`
External Service	OAuth, Gemini AI, SMTP Email	API endpoints
Database	Normalised PostgreSQL schema (ERD in 4.7 Component: Database).	SQL interface

4.1 Component: View (Presentation Layer)

Responsibilities

The View component provides the complete user interface for VocaBoost, implemented as a React single-page application. It handles user interaction, presents data in accessible formats, and manages client-side state for optimal user experience.

4.1.1 Partials

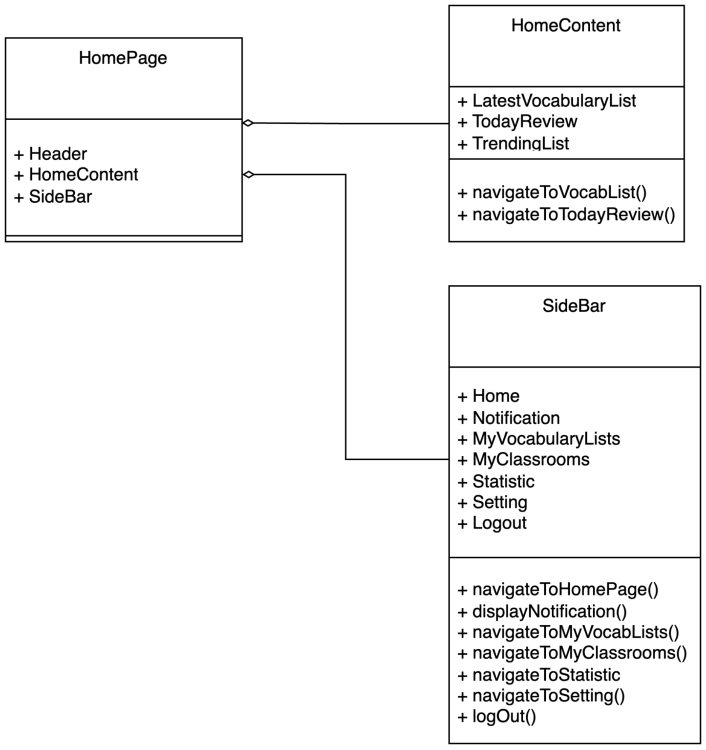


Description: small interface sections that are separated into their own files to be reused across multiple pages in a web application.

Classes:

- **Header** displays the top section of the website, including: Logo, Sign Up button, Sign In button Search bar, Profile and Vocabulary List Creation button. Functions such as accessing profiles or creating vocabulary lists are only available to users logged in the system.
- **VocabularyListCreating** navigates to the Vocabulary List Creating page which allows users to create personal vocabulary lists. The flow is input title, description, and vocabulary items, add, delete words, and finalize the vocabulary list
- **NavigationBar** is a top navigation panel showing links to: Contact, Support and Pricing page helps users navigate to support and service-related pages.
- **Footer** displays the bottom section of the website provides extra information and support links.

4.1.2 Home Page

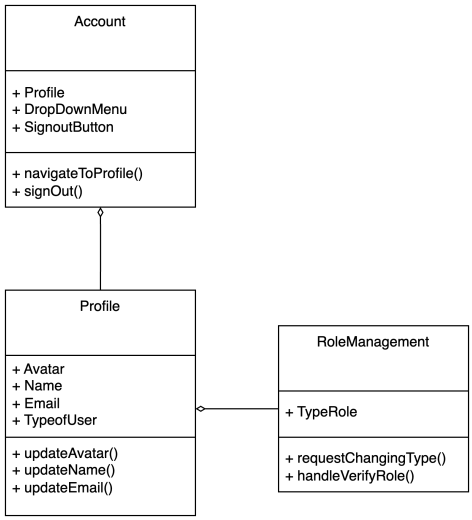


Description: represent the home screen of the website.

Classes:

- **HomePage** serves as the layout container for the main user interface, composes and displays two subcomponents HomeContent and Sidebar.
- **HomeContent** manages the main display content area of the home screen, which presents core content, which may include data views, dynamic elements, or interactive content like tables, charts, or forms.
- **SideBar** represents the navigation panel of the home interface which provides menu items to navigate to different parts of the application.

4.1.3 Account

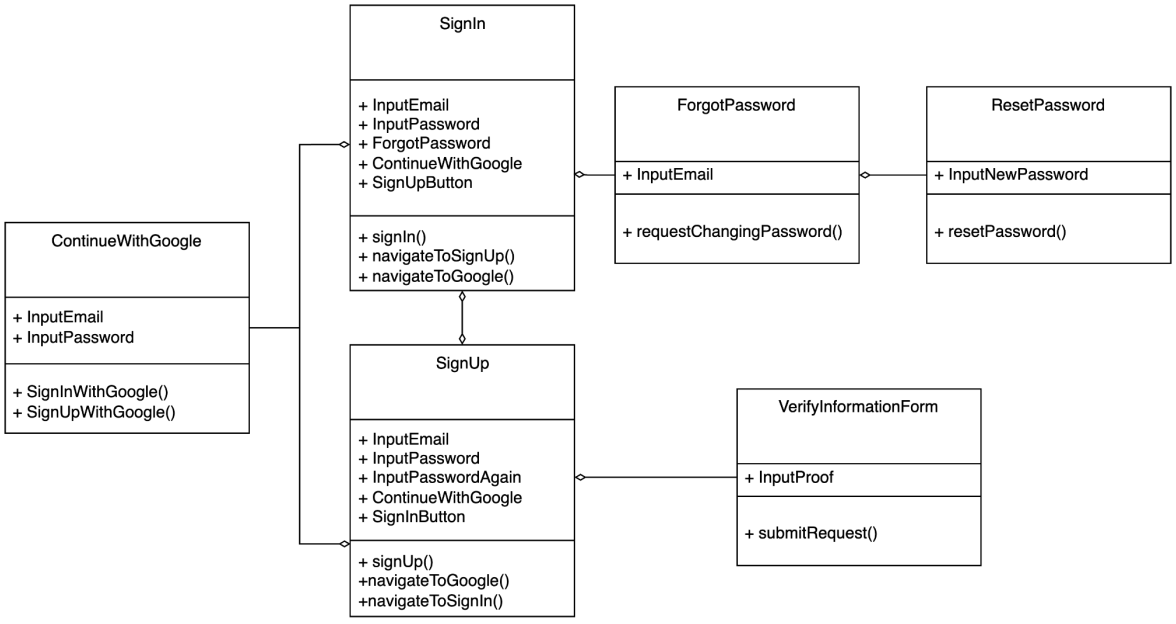


Description: handling user authentication, authorization, and profile personalization.

Classes:

- **Account** representing a user account entity in the system, manages basic information such as username, gmail etc.
- **RoleManagement** displays the role of users, supporting them to change their role.
- **Profile manages** the user's personal information and preferences, stores profile attributes like full name, contact details, bio, avatar, settings, etc

4.1.4 Authentication

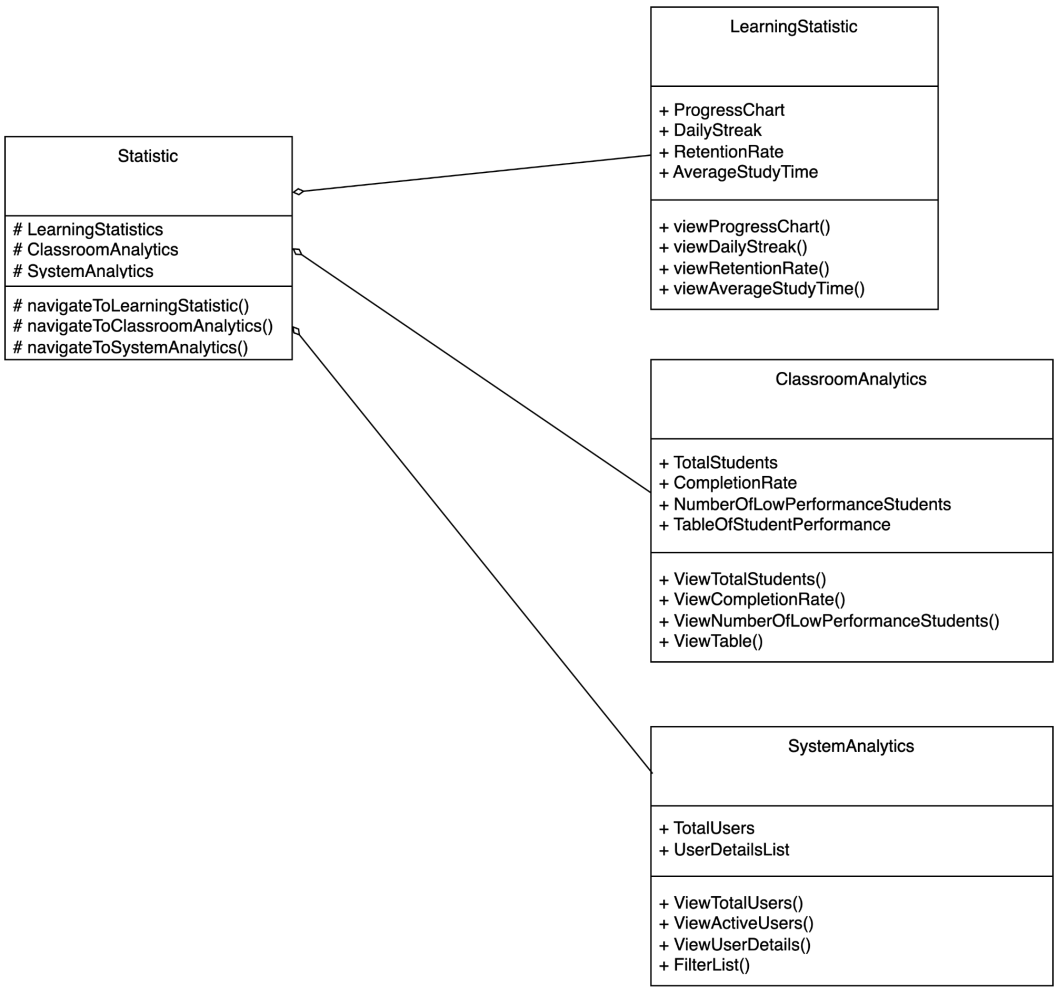


Description: responsible for user access and identity establishment in the application.

Classes:

- **SignIn** handles user login via email and password which collects credentials and sends authentication requests to the backend and provides feedback on login success/failure.
- **SignUp** manages new user registration by collecting user data for account creation (email, password) and includes validation checks (duplicate email, weak password, etc.).
- **VerifyInformationForm** support users who are teachers request Teacher roles to system administrators.
- **ForgotPassword** support user enters email account and sends verification/reset link to this email.
- **ResetPassword** verifies reset token, accepts and stores new password.
- **ContinueWithGoogle** enables Google OAuth login and initiates Google login flow and handles OAuth callback.

4.1.5 Statistic

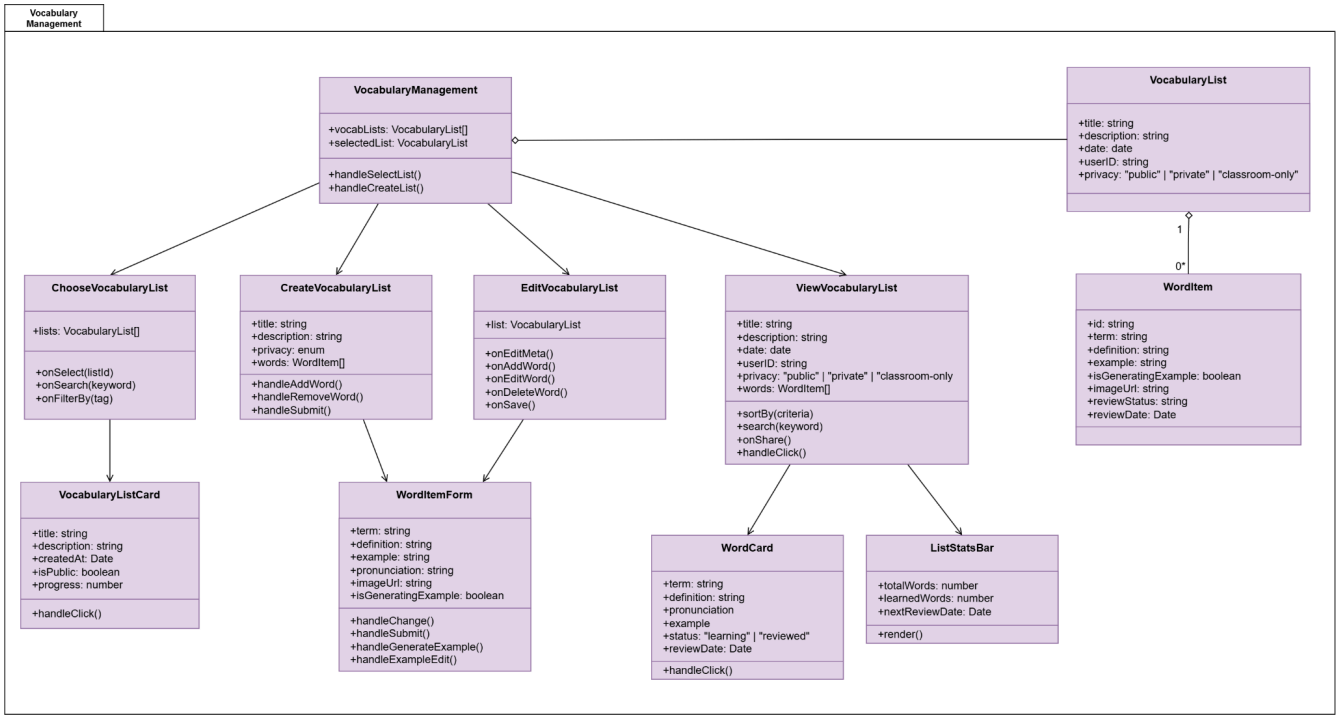


Description: responsible for user access and identity establishment in the website.

Classes:

- **Statistic** acts as the main controller for different types of analytics in the system which serves as the centralized interface for retrieving statistical data by role of users.
- **LearningStatistic** provides statistics related to learning activities such as progress charts, daily streak etc for users who are in a student role.
- **ClassroomAnalytics** supports users who are in a teacher role to manage their classroom by displaying reports including total students, completion rate, etc.
- **SystemAnalytics** supports admin by providing the number of users, the variety of active users and users detailed information.

4.1.6 Vocabulary Management

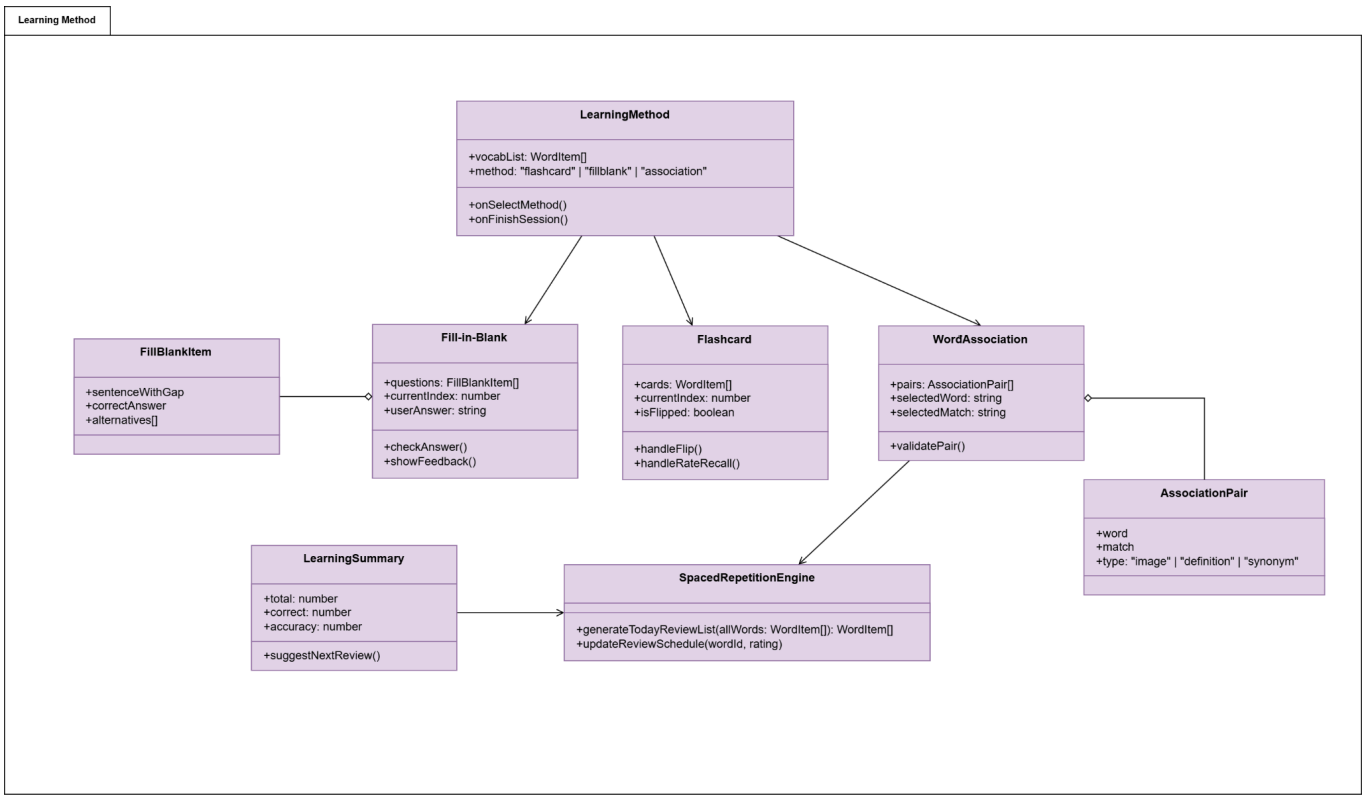


Description: responsible for enabling users to create, edit, view, and organize vocabulary lists within the VocaBoost system. It empowers users to manage their personal vocabulary collections, track learning progress, and leverage AI-generated example sentences to enhance learning effectiveness.
Role: Learner, Teacher
Classes:

- **VocabularyManagement**: The root class managing global state and navigating between different subviews like creation, selection, and editing of lists.
- **ChooseVocabularyList**: Displays available vocabulary lists, allowing users to search, filter, and select a specific list.

- **CreateVocabularyList**: Provides a form interface for creating a new list, adding words, setting description and privacy options
- **EditVocabularyList**: Allows editing of an existing list's metadata and vocabulary items.
- **ViewVocabularyList**: A read-only interface for viewing a list's content, supporting sorting, search, and sharing
- **VocabularyListCard**: Compact card layout to display basic information of each list (e.g., title, progress, visibility).
- **WordItemForm**: A form for adding or editing individual word items, including pronunciation, example, and AI-assisted generation.
- **WordCard**: Flashcard-like display of vocabulary items used during learning or review sessions.
- **ListStatsBar**: Displays summary statistics of the selected list including total words, learned words, and next review date.
- **VocabularyList** & **WordItem**: Data structures representing vocabulary collections and individual vocabulary entries, respectively.

4.1.7 Learning Method



Description:

The **Learning Method** component defines and handles the interactive learning experiences within the VocaBoost platform. It presents different learning modes—**Flashcard**, **Fill-in-the-Blank**, and **Word Association**—that help users memorize vocabulary more effectively using spaced repetition principles.

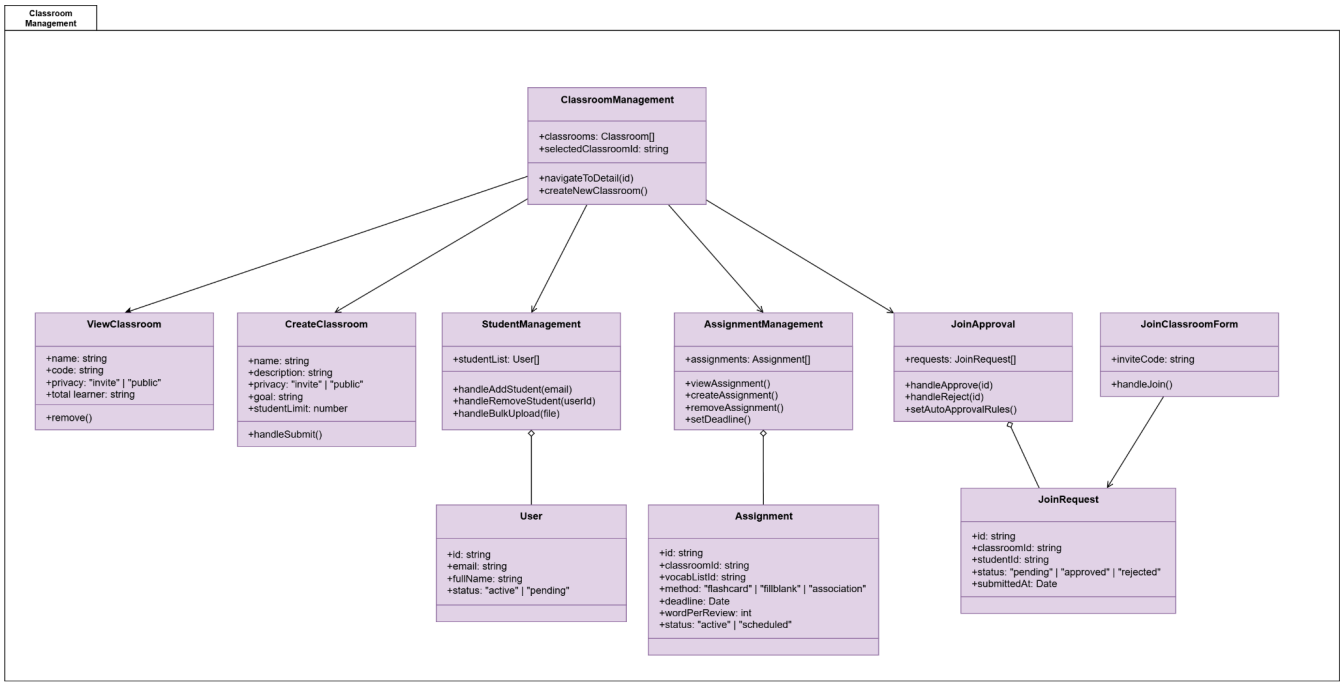
This component manages the learning session flow, checks user answers and updates review schedules accordingly. It integrates directly with the Spaced Repetition engine and generates a learning summary at the end of the session.

Role: Learner

Class:

- **LearningMethod**: Orchestrates the entire learning session, including method selection and finalization.
- **Flashcard**: Implements the flip-card interaction to review and rate recall quality.
- **Fill-in-Blank**: Presents exercises where users type the correct word into a sentence.
- **WordAssociation**: Displays matching tasks such as word-image, word-definition, or synonym associations.
- **LearningSummary**: Shows post-session analytics (total attempts, accuracy) and suggests review timing.
- **SpacedRepetitionEngine**: Calculates which words should be reviewed today and updates review schedules based on user performance.
- **Support classes** like *FillBlankItem* and *AssociationPair* model the data structures used in each exercise type

4.1.8 Classroom Management



Description:

The **Classroom Management** component provides a comprehensive interface for teachers to manage their virtual classrooms within the VocaBoost platform. It enables the full lifecycle of classroom

operations—from creating and configuring a class, managing student enrollment, assigning vocabulary-based tasks, to moderating join requests.

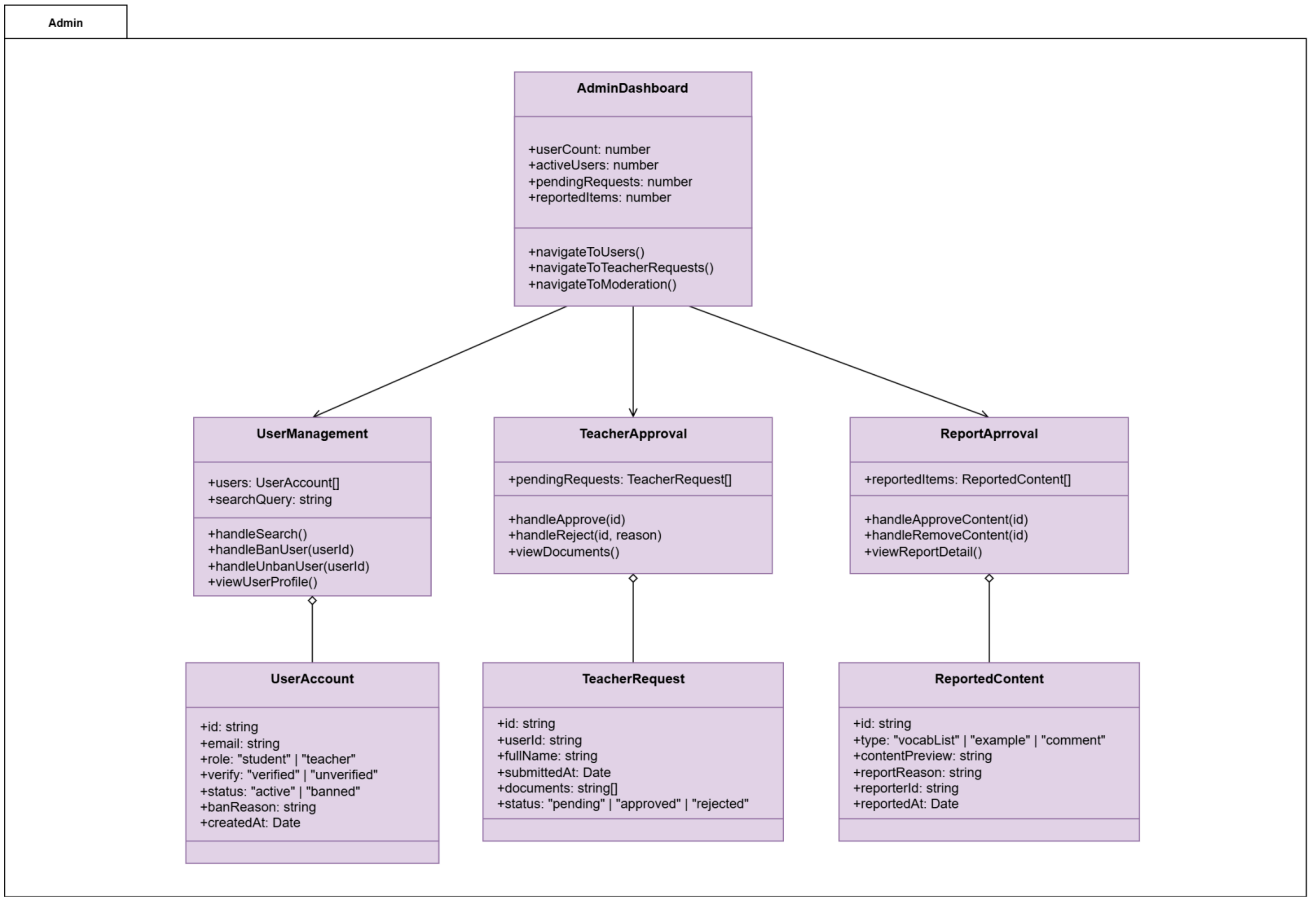
Role: Teacher

Class:

The component is composed of multiple interconnected classes that serve specific responsibilities:

- **ClassroomManagement**: Acts as the root container handling the list of classrooms and navigating to specific classroom details.
- **ViewClassroom:** Present classroom information including classroom name, classroom code, privacy and total learners of that classroom
- **CreateClassroom**: Presents a form to create a new classroom with name, privacy settings, and limits.
- **StudentManagement**: Allows the teacher to add, remove, or bulk-import students via email or file upload.
- **AssignmentManagement**: Enables assignment creation, selection of vocabulary lists, and deadline management.
- **JoinApproval**: Handles student join requests, including manual approval/rejection and auto-approval rules.
- **JoinClassroomForm**: Simple form interface for students to join a classroom using an invite code.
- **Supporting data structures** like *User*, *Assignment*, and *JoinRequest* define the entities manipulated by the above classes.

4.1.9 Administration



Description:

The **Admin Interface** provides administrative users with tools to monitor and moderate the VocaBoost platform. It enables high-level management of user accounts, teacher verification requests, and user-reported content. This component ensures that the system remains secure, trustworthy, and compliant with platform policies.

The main functions of the Admin Interface include:

- Displaying global platform statistics (e.g., total users, active users, pending teacher requests, reported content).
- Managing user accounts including banning/unbanning users.
- Reviewing and approving or rejecting teacher verification requests.
- Handling reported content such as inappropriate vocabulary lists or user-generated content.

Role: Admin

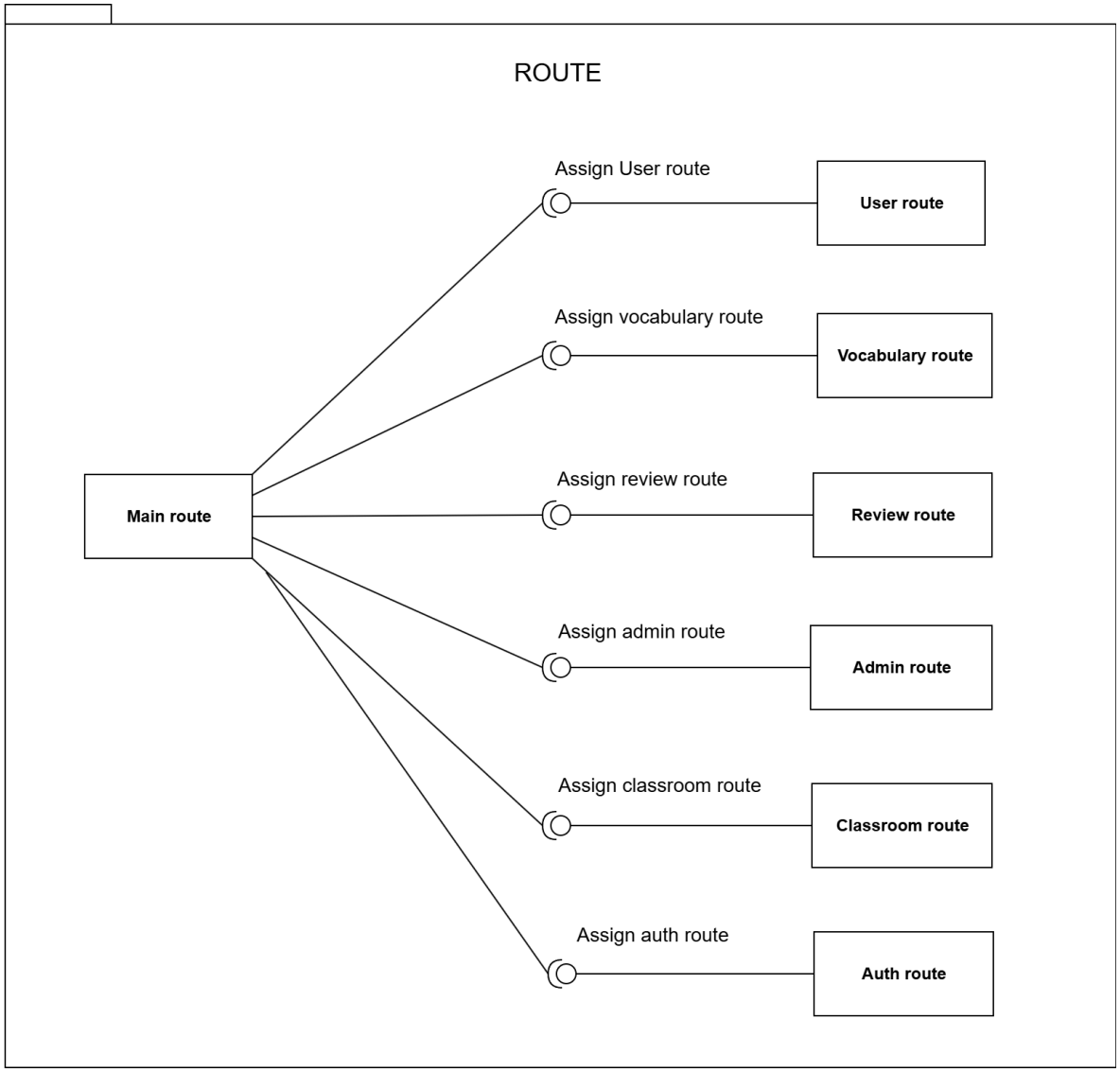
Class:

- **AdminDashboard**: Overview interface displaying system-level stats and navigation to moderation tools.
- **UserManagement**: Searchable table of user accounts with ban/unban and profile viewing actions.
- **TeacherApproval**: Handles processing of teacher verification requests submitted by users.
- **ReportApproval**: Moderates content flagged by users for potential violations.
- **Supporting data models** like *UserAccount*, *TeacherRequest*, and *ReportedContent* provide the structures needed for moderation workflows.

User Experience Features

All interface components implement responsive design principles, accessibility standards (ARIA labels, keyboard navigation), and loading states for optimal user experience. The interface supports real-time updates for collaborative features and maintains state persistence across browser sessions.

4.2 Component: Route (Business Logic Layer)



- The number of routes is equal to the number of controllers because routes format URLs from client requests for controllers to process. There are 6 controllers AuthController, VocabularyController, AdminController, ClassroomController, ReviewController, UserController, so there will be 6 corresponding routes. There will be a Main route responsible for categorizing requests and assigning them to one of the 6 routes appropriately.
- For example, if a learner wants to add a vocabulary list, this request will go to the Main route, and from there, the Main route will assign the Vocabulary route to process the request. Then, the VocabularyController will access the VocabularyRepository to process data as requested. When done, the controller returns data, through the routes, to the client.

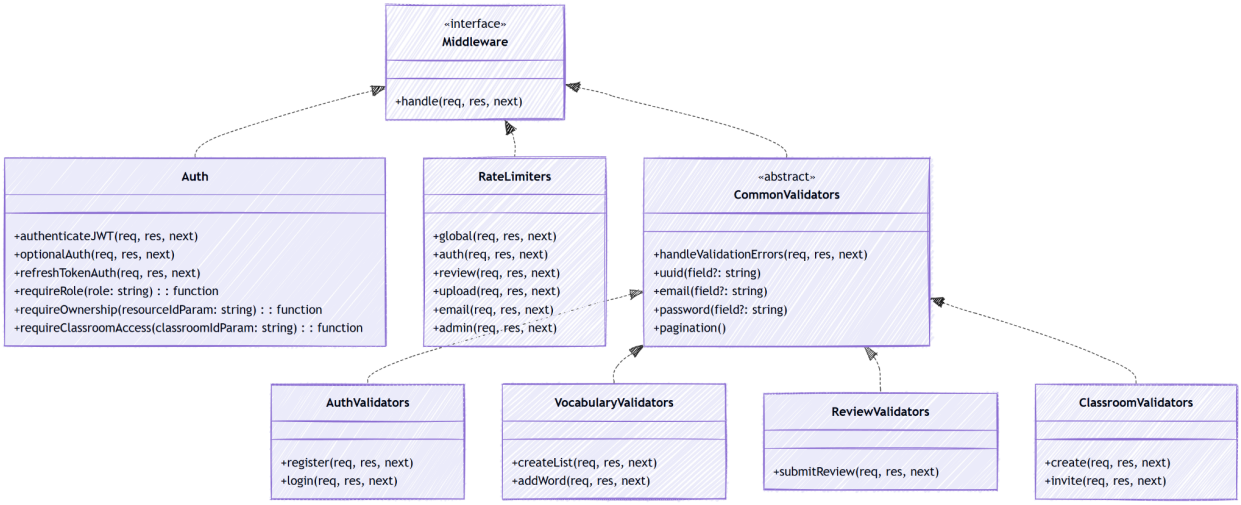
4.3 Component: Middleware (Business Logic Layer)

4.3.1 Responsibilities

Middleware is the component responsible for intercepting all requests and responses to handle cross-cutting concerns before they reach Controllers.

- **Authentication & Authorization** (JWT, Google OAuth, role-based)
- **Validation** (input checking, error collection)
- **Rate-limiting** (global, per API group)
- **Error handling** (catch exceptions, return standardised responses)
- **Request tracing** (attach request ID)
- **Performance monitoring** (measure processing time)

4.3.2 Class diagram



4.3.3 Key Middleware Classes

- **Middleware** (interface): defines a common handle(req, res, next) method..
- **Auth**: responsible for authentication and authorization.
- **RateLimiters**: define rate-limiting rules for different contexts (global, auth, review, etc.).
- **CommonValidators** (abstract): provides a shared handler and basic validation rules (UUID, email, password, pagination).
- **XValidators** (Auth/Vocabulary/Review/Classroom): extend CommonValidators, adding specific schemas/logic for each route group.

4.3.4 Detailed Method Descriptions

a) Class: Auth

1. authenticateJWT(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Authenticate the JWT token from the Authorization header using the Passport JWT strategy. If the token is valid, attach the user object to req.user. If not, return a 401 Unauthorized.
- **Error Handling**:
    - 500: Authentication error (when a server error occurs)
    - 401: Unauthorized (invalid token or user does not exist)
- **Implementation**: Use passport.authenticate('jwt', { session: false })

2. optionalAuth(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Optional JWT authentication. If a valid token is present, assign req.user; if not, assign null and allow the request to proceed.
- **Use Case**: Used for public endpoints that support personalization when authentication is available (e.g., viewing vocabulary lists).

3. refreshTokenAuth(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Authenticate the refresh token from the request body, verify it using JWT_REFRESH_SECRET, extract the userId, and assign it to req.userId.
- **Error Response**: 401 với message "Refresh token required" hoặc "Invalid refresh token"

4. requireRole(...roles)

- **Parameters**: ...roles - list of allowed roles (string[])
- **Returns**: Middleware function
- **Description**: A factory function that creates middleware to check user roles. The request is allowed only if req.user.role is included in the list of allowed roles.
- **Error Responses**:
    - 401: { error: 'Authentication required' } - when req.user is missing
    - 403: { error: 'Forbidden', message: 'Insufficient permissions' } - when the user's role lacks sufficient permissions

5. requireOwnership(resourceKey = 'userId')

- **Parameters**: resourceKey - the name of the parameter that contains the resource owner ID (default: 'userId')
- **Returns**: Middleware function
- **Description**: Checks resource ownership. Admins are allowed to access all resources. Other users can only access their own resources.
- **Logic**:

    - If the user is an admin → allow access
    - Otherwise, compare req.user.id with req.params[resourceKey] or req.body[resourceKey]
- **Error**: 403 with the message: "You can only access your own resources"

6. requireClassroomAccess(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Checks classroom access permissions based on classroomId from request parameters.
- **Access Rules**:
    - Admin: Allowed to access all classrooms
    - Teacher: Allowed to access classrooms where they are the teacher
    - Learner: Allowed to access classrooms where they are an active learner
- **Process Flow**:
    - Validate classroomId from req.params
    - Call Classroom.checkUserAccess() to verify access rights
    - Attach access info to req.classroomAccess for use in controllers
- **Error Responses**:
    - 400: { success: false, error: 'Classroom ID is required' } - Missing classroomId
    - 404: { success: false, error: 'Classroom not found' } - Classroom does not exist
    - 403: { success: false, error: 'Access denied', message: 'You do not have permission to access this classroom' } - Access denied
    - 500: { success: false, error: 'Access check failed', message: 'Unable to verify classroom access' } - Internal error during access check

b) Class: RateLimiters

1. global

- **Type**: Rate limiter instance
- **Configuration**:
    - Window: 15 minutes (900,000ms)
    - Max requests: 1000
    - Message: "Too many requests from this IP"
- **Key Generation**: Use userId (if authenticated) or IP address

2. auth

- **Type**: Rate limiter instance
- **Configuration**:
    - Window: 15 minutes
    - Max requests: 5
    - Message: "Too many authentication attempts"

- **Use Case**: Protects login/register endpoints from brute-force attacks

3. review

- **Type**: Rate limiter instance
- **Configuration**:
  - Window: 1 minute (60,000ms)
  - Max requests: 30
  - Message: "Please slow down your learning pace"
- **Use Case**: Limits the rate of review submissions

4. upload

- **Type**: Rate limiter instance
- **Configuration**:
  - Window: 1 minute
  - Max requests: 5
  - Message: "Too many file uploads"

5. email

- **Type**: Rate limiter instance
- **Configuration**:
  - Window: 1 hour (3,600,000ms)
  - Max requests: 10
  - Message: "Email limit exceeded"

6. admin

- **Type**: Rate limiter instance
- **Configuration**:
  - Window: 1 minute
  - Max requests: 100
  - Message: "Admin action rate limit"

c) Class: CommonValidators

1. handleValidationErrors(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Checks the validation results from express-validator. If there are any errors, returns a 400 response with a detailed list of validation errors.
- **Response Format**:
  {

```
  "success": false,
  "error": "Validation failed",
  "details": [
    {
      "field": "fieldName",
      "message": "Error message",
      "value": "Invalid value"
    }
  ]
}
```

2. uuid(field = 'id')

- **Parameters**: field - name of the parameter to validate (default: 'id')
- **Returns**: express-validator chain for the param
- **Description**: Validate UUID v4 format
- **Error Message**: "Invalid ID format"

3. email(field = 'email')

- **Parameters**: field - name of the field to validate (default: 'email')
- **Returns**: express-validator chain for the body field
- **Description**: Validates and normalizes email (converts to lowercase, removes dots)
- **Error Message**: "Invalid email format"

4. password(field = 'password')

- **Parameters**: field - name of the field to validate (default: 'password')
- **Returns**: express-validator chain for the body field
- **Validation Rules**:
  - Length: 8-128 characters
  - Pattern: Must include at least one uppercase letter, one lowercase letter, and one number
- **Error Messages**:
  - "Password must be 8-128 characters"
  - "Password must contain uppercase, lowercase and number"

5. pagination

- **Type**: Array of validators
- **Validates**:
  - page: optional, integer min 1, auto convert to int
  - limit: optional, integer 1-100, auto convert to int
  - sort: optional, accepts only 'asc' or 'desc'
  - sortBy: optional, string length 1-50

d) Class: AuthValidators

1. register

- **Type**: Middleware array
- **Includes**:
  - commonValidators.email() - validate email format
  - commonValidators.password() - validate password strength
  - body('role') - optional; only accepts 'learner' or 'teacher'
  - handleValidationErrors - handles validation errors

2. login

- **Type**: Middleware array
- **Includes**:
  - commonValidators.email() - validate email format
  - body('password').notEmpty() - password must not be empty
  - handleValidationErrors - handles validation errors

e) Class: VocabularyValidators

1. createList

- **Type**: Middleware array
- **Validates**:
  - title: trim, length 1-200 chars
  - description: optional, trim, max 1000 chars
  - language: required, must be one of ['en', 'vi', 'fr', 'de', 'ja', 'ko']
  - difficulty: optional, must be one of ['beginner', 'intermediate', 'advanced']
  - isPublic: optional, boolean
- **Includes**: handleValidationErrors

2. addWord

- **Type**: Middleware array
- **Validates**:
  - word: trim, length 1-100 chars
  - translation: trim, length 1-200 chars
  - definition: optional, trim, max 500 chars
  - example: optional, trim, max 300 chars
  - pronunciation: optional, trim, max 100 chars
- **Includes**: handleValidationErrors

f) Class: ReviewValidators

1. submitReview

- **Type**: Middleware array
- **Validates**:
  - vocabularyId: must be in UUID format (using isUUID())
  - performance: integer between 0 and 3
  - responseTime: optional, integer 0-300000 (milliseconds)
  - isNew: optional, boolean
- **Includes**: handleValidationErrors

g) Class: ClassroomValidators

1. create

- **Type**: Middleware array
- **Validates**:
  - name: trim, length 1-100 chars
  - description: optional, trim, max 500 chars
  - gradeLevel: optional, integer 1-12
  - maxLearners: optional, integer 1-100
- **Includes**: handleValidationErrors

2. invite

- **Type**: Middleware array
- **Validates**:
  - classroomId: UUID format trong params
  - emails: array with 1 to 50 items
  - emails.*: each item must be a valid email
- **Includes**: handleValidationErrors

4.4 Component: Controller (Business Logic Layer)

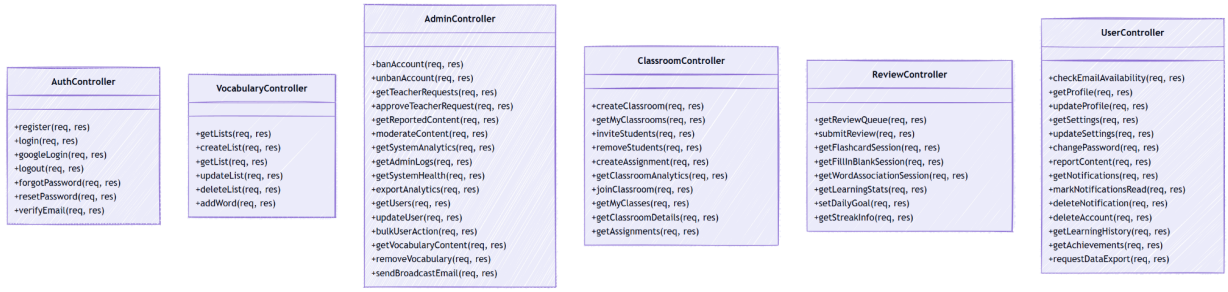
4.4.1 Responsibilities

Controllers are responsible for receiving requests from the API layer, validating permissions, invoking the corresponding Service, and returning responses to the client. Each controller maps to a specific group of primary endpoints:

- **AuthController**: Handles registration, login, OAuth, password changes, email verification, etc.
- **VocabularyController**: Manages vocabulary lists, adding words, updating, and deleting.
- **AdminController**: Covers system administration features, content moderation, analytics, and broadcasting messages.

- **ClassroomController**: Manages classroom creation/invitation/deletion, assignments, and classroom analytics.
- **ReviewController**: Handles the review flow (flashcards, fill-in-the-blank, word suggestions) and learning statistics.
- **UserController**: Manages user profiles, settings, notifications, and data export requests.

4.4.2 Class diagram



4.4.3 Key Controller Classes

- **AdminController**: Manages system administration features, including user management, teacher request approvals, content violation report handling, and system-wide data analytics.
- **AuthController**: Handles user authentication and account-related operations such as registration, login, email verification, and password management.
- **ClassroomController**: Supports classroom management activities like creating new classes, adding or removing students, assigning homework, and monitoring learner performance.
- **ReviewController**: Coordinates vocabulary review activities using spaced repetition methods and other formats such as flashcards, fill-in-the-blank, and word matching.
- **UserController**: Manages user profiles, settings, notifications, and personalized requests such as updating user information and exporting personal data.
- **VocabularyController**: Manages vocabulary lists, supports creating, editing, and deleting vocabulary sets, and offers supplementary features like AI-generated example sentences.

4.4.4 Detailed Method Descriptions

a) Class: AuthController

1. register(req, res)
    - **Description:** Registers a new user account.
    - **Input Parameters:**
        1. req: Contains email, password, and role (default is "learner")
        2. res: Used to send the response
    - **Return Value:** Registered user information and a JWT token
    - **Execution Flow:**

1. Check if the email already exists
2. Create a new user and send a verification email
3. Generate a JWT token and return user information

2. login(req, res)
    - **Description:** Logs the user into the system.
    - **Input Parameters:**
        1. req: Contains email and password
        2. res: Used to send the response
    - **Return Value:** User information and a JWT token
    - **Execution Flow:**
        1. Authenticate the email and password
        2. Check the account status
        3. Update the last login timestamp and return a JWT token

b) Class: UserController

1. getProfile(req, res)
    - **Description**: Retrieves the user's profile information.
    - **Input Parameters:**
        1. req: Contains user information
        2. res: Used to send the response
    - **Return Value:** The user's detailed profile
    - **Execution Flow:**
        1. Query the user's profile data from the database
2. updateProfile(req, res)
    - **Description**: Updates the user's personal profile.
    - **Input Parameters:**
        1. req: Contains the data to update (e.g., full name, avatar)
        2. res: Used to send the response
    - **Return Value:** The updated user profile
    - **Execution Flow:**
        1. Validate the update data
        2. Update the profile in the database

c) Class: VocabularyController

createList(req, res)

- **Description**: Creates a new vocabulary list.
- **Input Parameters:**
    1. req: Contains vocabulary list data (name, description, privacy setting, vocabulary items)
    2. res: Used to send the response
- **Return Value:** Information about the newly created vocabulary list

- **Execution Flow:**
    1. Create the vocabulary list
    2. Add vocabulary items to the list, optionally use AI to generate example sentences

d) Class: ReviewController

getReviewQueue(req, res)

- **Description**: Retrieves the vocabulary items that need to be reviewed based on spaced repetition.
- **Input Parameters:**
    1. req: Specifies the limit on the number of vocabulary items to return
    2. res: Used to send the response
- **Return Value:** A list of vocabulary items due for review and new items
- **Execution Flow:**
    1. Check the cache for review items
    2. Query vocabulary items needing review
    3. If there are not enough, fetch additional new words
    4. Cache the result

e) Class: ClassroomController

1. createClassroom(req, res)
    - **Description**: Creates a new classroom
    - **Input Parameters:**
        1. req: Contains classroom details (name, description, subject, grade level, learner limit)
        2. res: Used to send the response
    - **Return Value:** Information about the newly created classroom
    - **Execution Flow:**
        1. Generate a unique classroom code
        2. Create the classroom with the provided details
2. inviteLearners(req, res)
    - **Description**: Sends invitations to learners to join a classroom
    - **Input Parameters:**
        1. req: Contains a list of learner emails, classroom code, and optional invitation message
        2. res: Used to send the response
    - **Return Value:** Notification about the invitation status
    - **Execution Flow:**
        1. Verify classroom ownership
        2. Generate learner invite codes and send email invitations

f) Class: AdminController

1. banAccount(req, res)
    - **Description**: Bans a user account for violating rules
    - **Input Parameters:**
        1. req: Contains the user ID, reason for the ban, and optional duration
        2. res: Used to send the response
    - **Return Value:** Success response along with information about the banned user
    - **Execution Flow:**
        1. Verify admin privileges and check the user's role
        2. Update the account status to "suspended"
        3. Log the action
        4. Send a notification email to the banned user
2. unbanAccount(req, res)
    - **Description**: Lifts the ban on a user account.
    - **Input Parameters:**
        1. req: Contains the user ID
        2. res: Used to send the response
    - **Return Value:** Success response with information about the reactivated account
    - **Execution Flow:**
        1. Update the account status to "active"
        2. Log the action
3. getTeacherRequests(req, res)
    - **Description**: Retrieves a list of teacher registration requests.
    - **Input Parameters:**
        1. req: Contains the desired request status (e.g., pending, approved, rejected)
        2. res: Response containing the returned data
    - **Return Value:** List of teacher requests
    - **Execution Flow:**
        1. Query the request list based on the provided status

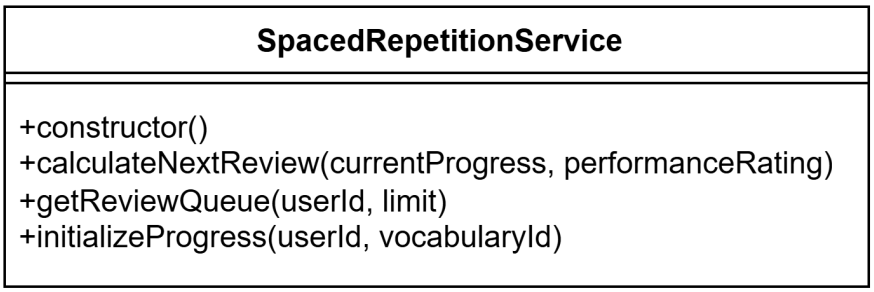
4.5 Component: Service (Business Logic Layer)

4.5.1 Responsibilities:

Responsible for containing core business logic, separated from controllers and data access layers.

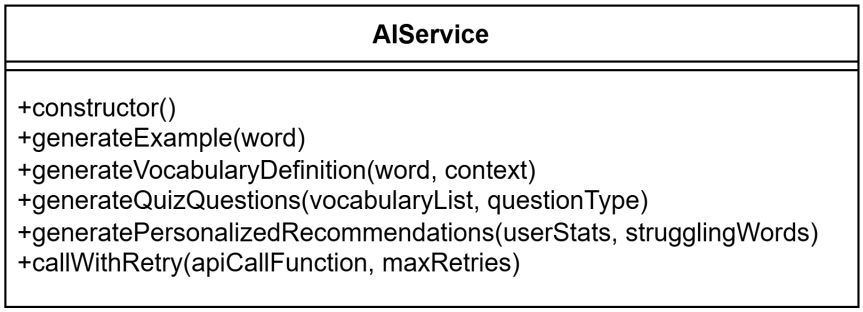
4.5.2 Key Service Classes:

a. SpacedRepetitionService



- **Description**: Manages the core logic of the Spaced Repetition System (SRS) using the SM-2 algorithm. This service is the brain behind scheduling vocabulary reviews to optimize long-term memory retention.
- **Methods**:
    - **constructor()**: Initializes settings for the SM-2 algorithm, such as default easiness factors and initial intervals.
    - **calculateNextReview(currentProgress, performanceRating)**: Calculates the next review date for a vocabulary item based on the user's performance (a rating from 0-5) and the current state of their learning progress (interval, repetitions).
    - **getReviewQueue(userId, limit)**: Fetches a list of vocabulary items that are due for review for a specific user. It queries the database via the repository and may use caching for performance. It prioritizes items by their due date and also includes new words.
    - **initializeProgress(userId, vocabularyId)**: Creates the initial learning progress record for a user and a new word, setting default values.

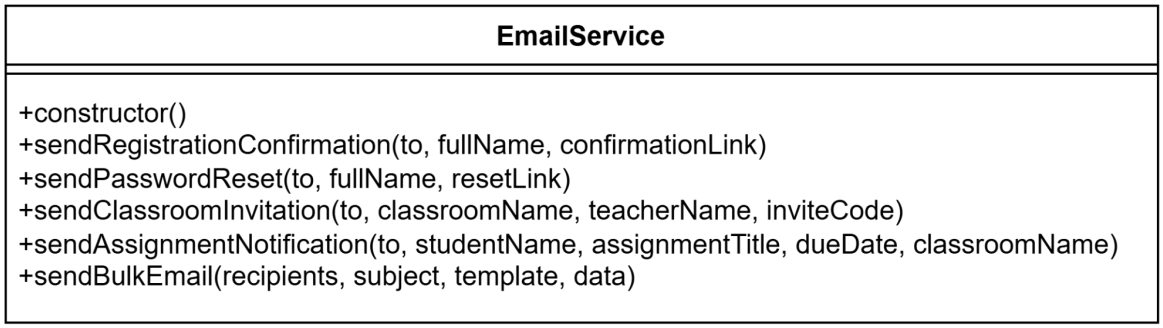
b. AIService



- **Description**: Acts as a gateway to an external AI provider. It abstracts the complexities of making API calls, handling retries, and formatting prompts.
- **Methods**:
    - **constructor()**: Sets up the HTTP client, loading API keys and base URLs from environment variables.

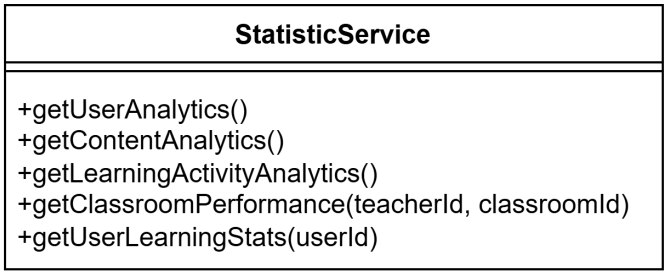
- **generateExample(word)**: Sends a request to the AI to generate a simple example sentence for a given word.
- **generateVocabularyDefinition(word, context)**: Requests a definition from the AI, potentially providing context for more accurate results.
- **generateQuizQuestions(vocabularyList, questionType)**: Generates multiple-choice or fill-in-the-blank questions for a given list of vocabulary.
- **generatePersonalizedRecommendations(userStats, strugglingWords)**: Creates a personalized list of recommended vocabulary for a user to learn next.
- **callWithRetry(apiCallFunction, maxRetries)**: A wrapper to execute any AI API call with a retry/backoff mechanism for handling rate limits or transient server errors.

c. EmailService



- **Description**: Manages all aspects of sending transactional emails. It handles template rendering and interacts with an SMTP provider.
- **Methods**:
  - **constructor()**: Initializes the email transporter with SMTP configuration and pre-compiles email templates from the filesystem.
  - **sendRegistrationConfirmation(to, fullName, confirmationLink)**: Sends an email with a verification link to a new user.
  - **sendPasswordReset(to, fullName, resetLink)**: Sends an email containing a password reset link.
  - **sendClassroomInvitation(to, classroomName, teacherName, inviteCode)**: Sends an invitation email to a student to join a classroom.
  - **sendAssignmentNotification(to, studentName, assignmentTitle, dueDate, classroomName):** Notifies a student about a new or upcoming assignment.
  - **sendBulkEmail(recipients, subject, template, data)**: Sends a templated email to a list of recipients.

d. StatisticService



- **Description**: A specialized service responsible for aggregating raw data from various repositories to generate meaningful statistics and analytics. It acts as the central hub for all data analysis tasks, providing insights for users, teachers, and administrators.
- **Methods**:
    - **getUserAnalytics()**: Retrieves key metrics about user growth and distribution across the platform.
    - **getContentAnalytics()**: Retrieves key metrics about the content created on the platform.
    - **getLearningActivityAnalytics()**: Retrieves metrics about the overall learning engagement across the platform.
    - **getClassroomPerformance(teacherId, classroomId)**: Retrieves performance-related metrics for a specific classroom.
    - **getUserLearningStats(userId)**: Retrieves a comprehensive summary of a single user's learning journey.

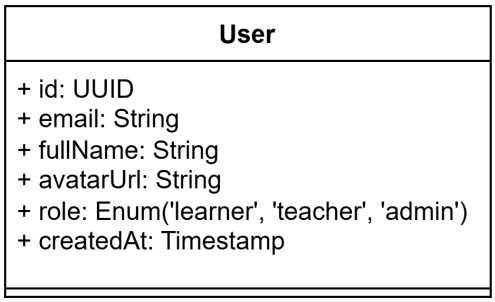
4.6 Component: Model/Repository (Data Access Layer)

- **Data Structures (Schema)**: Simple interfaces or classes that define the structure of data objects. They serve as a data "contract" throughout the system.
- **Repositories**: Classes containing methods to perform CRUD (Create, Read, Update, Delete) operations and handle complex business logic related to the data entities.

4.6.1 Data Structures

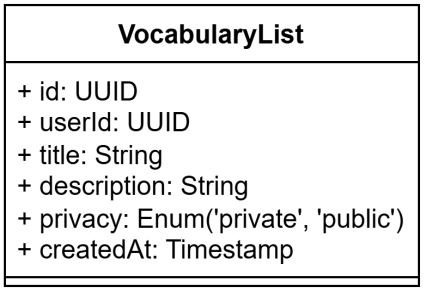
This section defines the "shape" of the core data objects within the VocaBoost system.

a. User



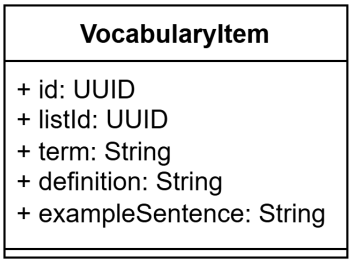
- **Description:** Represents a user account in the system.
- **Attributes:**
  - **id (UUID)**: The primary key, a unique identifier for the user.
  - **email (String)**: The user's email address, used for login.
  - **fullName (String)**: The full name of the user.
  - **avatarUrl (String, nullable)**: A URL to the user's profile picture.
  - **role (Enum: 'learner', 'teacher', 'admin')**: The user's role within the system.
  - **createdAt (Timestamp)**: The timestamp when the account was created.

b. VocabularyList



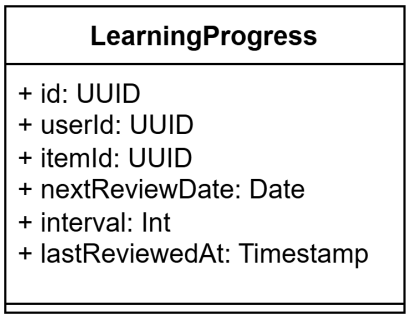
- **Description:** Represents a collection of vocabulary words created by a user.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the list.
  - **userId (UUID)**: The ID of the user who owns the list.
  - **title (String)**: The title of the list.
  - **description (String, nullable)**: A detailed description of the list.
  - **privacy (Enum: 'private', 'public')**: The privacy setting for the list.
  - **createdAt (Timestamp)**: The timestamp when the list was created.

c. VocabularyItem



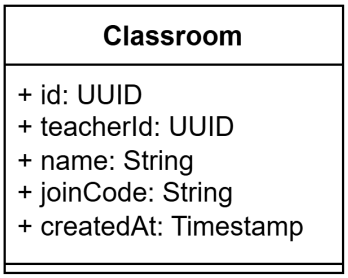
- **Description:** Represents a single vocabulary word within a VocabularyList.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the item.
  - **listId (UUID)**: The ID of the VocabularyList this item belongs to.
  - **term (String)**: The vocabulary word (e.g., "Sustainable").
  - **definition (String)**: The definition of the term.
  - **exampleSentence (String, nullable)**: An example sentence using the term.

d. LearningProgress



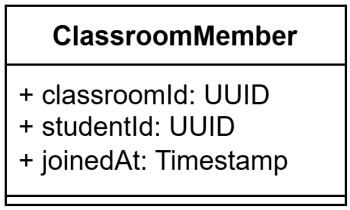
- **Description:** Tracks a User's learning progress for a specific VocabularyItem. This is the core entity for the Spaced Repetition feature.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the progress record.
  - **userId (UUID)**: The ID of the learner.
  - **itemId (UUID)**: The ID of the VocabularyItem being learned.
  - **nextReviewDate (Date)**: The date when the item should be reviewed again.
  - **interval (Integer)**: The number of days until the next review.
  - **lastReviewedAt (Timestamp)**: The timestamp of the last review.

e. Classroom



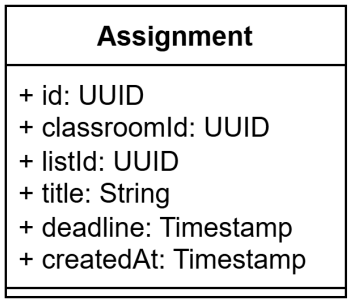
- **Description:** Represents a virtual classroom created by a teacher to manage students and assignments.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the classroom.
  - **teacherId (UUID)**: The ID of the teacher who created the classroom.
  - **name (String)**: The name of the classroom.
  - **joinCode (String)**: A code for students to join the classroom.
  - **createdAt (Timestamp)**: The timestamp when the classroom was created.

f. ClassroomMember



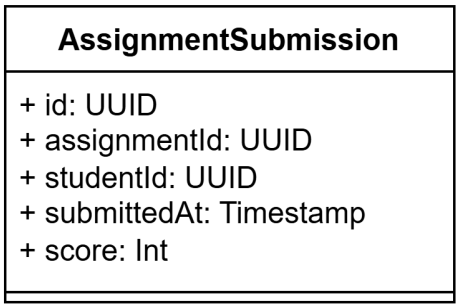
- **Description:** A join table representing a User's membership in a Classroom.
- **Attributes:**
  - **classroomId (UUID)**: The ID of the classroom.
  - **studentId (UUID)**: The ID of the student member.
  - **joinedAt (Timestamp)**: The timestamp when the student joined.

g. Assignment



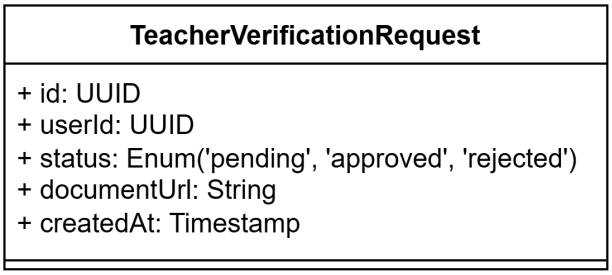
- **Description:** Represents a vocabulary exercise assigned by a teacher to a Classroom.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the assignment.
  - **classroomId (UUID)**: The ID of the classroom the assignment is for.
  - **listId (UUID)**: The ID of the VocabularyList used for the assignment.
  - **title (String)**: The title of the assignment.
  - **deadline (Timestamp)**: The due date for the assignment.
  - **createdAt (Timestamp)**: The timestamp when the assignment was created.

h. AssignmentSubmission



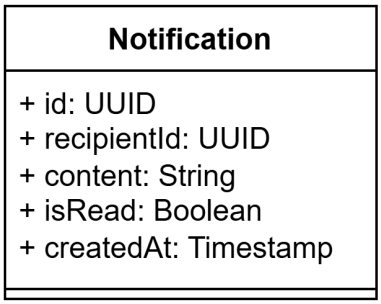
- **Description:** Records a student's submission for an Assignment.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the submission.
  - **assignmentId (UUID)**: The ID of the submitted assignment.
  - **studentId (UUID)**: The ID of the student who submitted.
  - **submittedAt (Timestamp)**: The timestamp of the submission.
  - **score (Integer, nullable)**: The score received for the submission.

i. TeacherVerificationRequest



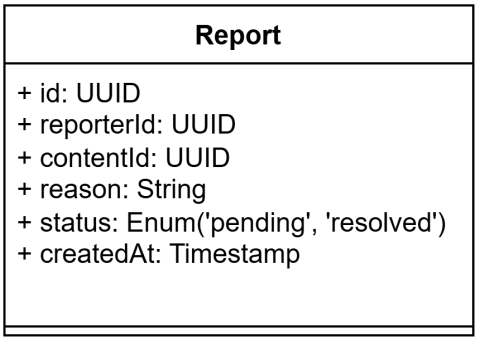
- **Description:** Stores a request from a user to be upgraded to the teacher role.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the request.
  - **userId (UUID)**: The ID of the user making the request.
  - **status (Enum: 'pending', 'approved', 'rejected')**: The status of the request.
  - **documentUrl (String)**: A URL to the credential document.
  - **createdAt (Timestamp)**: The timestamp when the request was made.

j. Notification



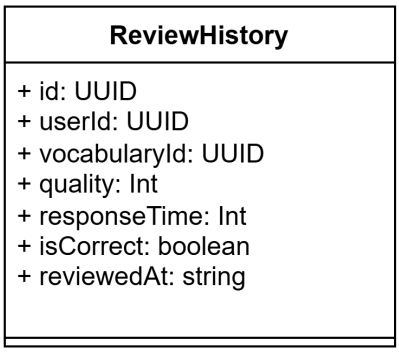
- **Description:** Represents a notification sent to a user.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the notification.
  - **recipientId (UUID)**: The ID of the user receiving the notification.
  - **content (String)**: The content of the notification.
  - **isRead (Boolean)**: The read status of the notification.
  - **createdAt (Timestamp)**: The timestamp when the notification was created.

k. Report



- **Description:** Records a user's report of inappropriate content.
- **Attributes:**
  - **id (UUID)**: A unique identifier for the report.
  - **reporterId (UUID)**: The ID of the user who made the report.
  - **contentId (UUID)**: The ID of the reported content (VocabularyList or VocabularyItem).
  - **contentType (String)**: The type of reported content ('list' or 'item').
  - **reason (String)**: The reason for the report.
  - **status (Enum: 'pending', 'resolved')**: The processing status of the report.
  - **createdAt (Timestamp)**: The timestamp when the report was created.

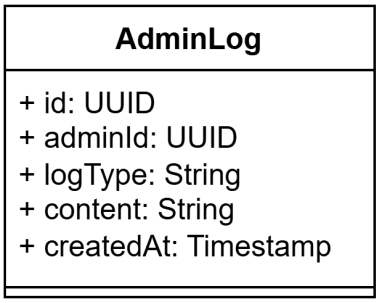
l. ReviewHistory



- **Description**: Represents a log of a single review event. This data is crucial for tracking a user's detailed interaction with a vocabulary item and for future analytical purposes.
- **Attributes**:
  - **id (UUID)**: The unique identifier for the review log entry.
  - **userId (UUID)**: The ID of the user who performed the review.
  - **vocabularyId (UUID)**: The ID of the vocabulary item that was reviewed.
  - **quality (Int)**: A score (typically 0-5) representing the user's recall quality during the review.

- **responseTime (Int)**: The time taken to respond, measured in milliseconds.
- **isCorrect (Boolean)**: A flag indicating whether the user's answer was correct.
- **reviewedAt (String)**: The timestamp indicating when the review occurred.

m. AdminLog

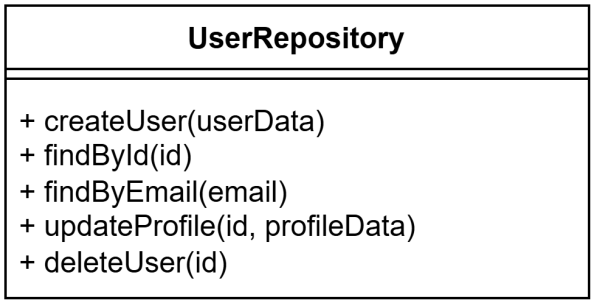


- **Description:** Represents a log entry that records a significant action performed by an administrator.
- **Attributes:**
  - **id (UUID)**: The unique identifier for the log entry.
  - **adminId(UUID)**: The ID of the administrator who performed the action.
  - **logType (String)**: A category or type for the logged action (e.g., "UserBan", "ContentModeration", "TeacherRequest").
  - **content (String)**: Description of the action that was performed.
  - **createdAt (Timestamp)**: The exact time the action occurred and the log entry was created.

4.6.2 Repositories

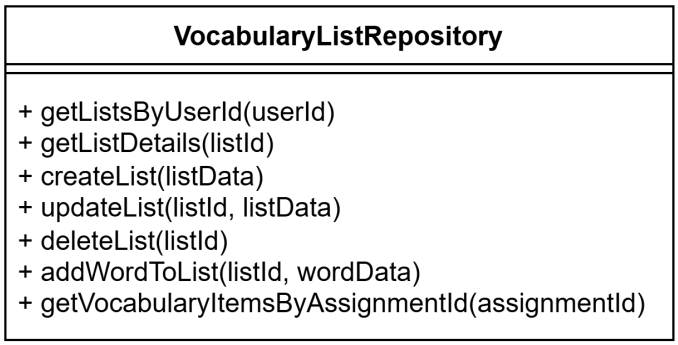
This section describes the classes responsible for interacting with the database. Each method corresponds to a specific logical operation.

a. Class: UserRepository



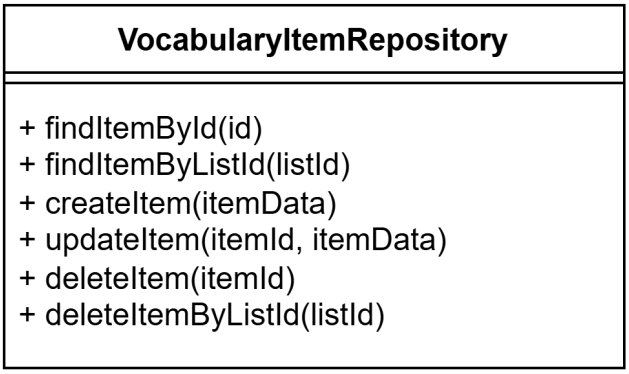
- **Description:** Manages data operations related to user entities.
- **Methods:**
    - **createUser(userData)**: Creates a new user. Returns a structured object representing the new user.
    - **findById(id)**: Retrieves a structured object with user information by their ID.
    - **findByEmail(email)**: Retrieves a structured object with user information by their email.
    - **updateProfile(id, profileData)**: Updates a user's fullName and avatarUrl. Returns the updated user object.
    - **deleteUser**: Deletes a user account from the database.

b. Class: VocabularyListRepository



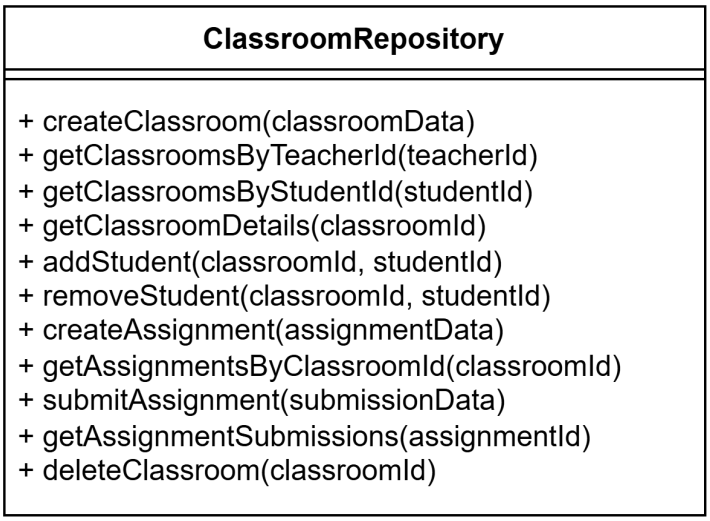
- **Description:** Manages data operations for VocabularyList entities.
- **Methods:**
    - **getListsByUserId(userId)**:Retrieves a collection of all vocabulary list objects for a specific user.
    - **getListDetails(listId)**: Retrieves a single vocabulary list object, including all its associated vocabulary items.
    - **createList(listData)**: Creates a new vocabulary list. Returns an object representing the newly created list.
    - **updateList(listId, listData)**: Updates a vocabulary list's information. Returns the updated list object.
    - **deleteList(listId)**: Deletes a vocabulary list and all its associated items.
    - **addWordToList(listId, wordData)**: Adds a new vocabulary item to a list. Returns an object representing the new item.
    - **getVocabularyItemsByAssignmentId(assignmentId)**:Finds the associated vocabulary list for an assignment and returns a collection of its vocabulary item objects.

c. VocabularyItemRepository



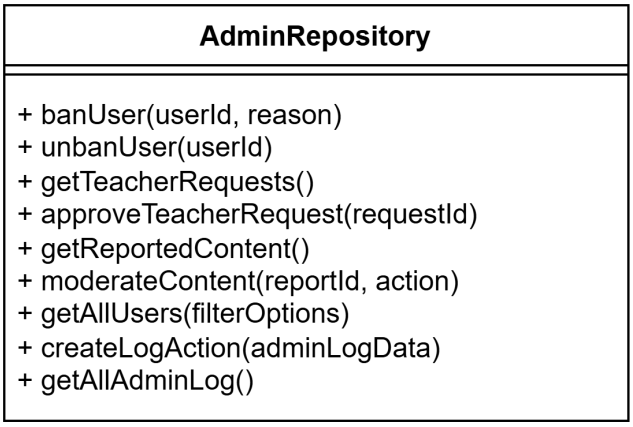
- **Description**: Manages data operations for VocabularyItem entities.
- **Methods**:
    - **findItemById(id)**: Retrieves a single vocabulary item object by its unique ID.
    - **findItemByListId(listId)**: Retrieves a collection of all vocabulary item objects belonging to a specific vocabulary list.
    - **createItem(itemData)**: Creates a new vocabulary item record in the database. Returns an object representing the new item.
    - **updateItem(itemId, itemData)**: Updates the information of a single vocabulary item.
    - **deleteItem(itemId)**: Deletes a single vocabulary item from the database.
    - **deleteItemByListId(listId)**: Deletes all vocabulary items associated with a specific vocabulary list.

d. Class: ClassroomRepository



- **Description:** Manages all data operations related to classrooms, members, and assignments.
- **Methods:**
    - **createClassroom(classroomData)**: Creates a new Classroom.
    - **getClassroomsByTeacherId(teacherId)**: Gets all classrooms managed by a teacher.
    - **getClassroomsByStudentId(studentId)**: Gets all classrooms a student is a member of.
    - **getClassroomDetails(classroomId)**:Gets a single classroom object, potentially enriched with a list of student and assignment information.
    - **addStudent(classroomId, studentId)**: Adds a student to a classroom by creating a ClassroomStudent record.
    - **removeStudent(classroomId, studentId)**: Removes a student from a classroom.
    - **createAssignment(assignmentData)**: Creates a new Assignment for a classroom.
    - **getAssignmentsByClassroomId(classroomId)**: Retrieves a collection of all assignment objects for a classroom.
    - **submitAssignment(submissionData)**: Creates a StudentAssignment record when a student submits their work.
    - **getAssignmentSubmissions(assignmentId)**: Retrieves all StudentAssignment records for a specific assignment.
    - **deleteClassroom(classroomId)**: Deletes a classroom and all member from that classroom.

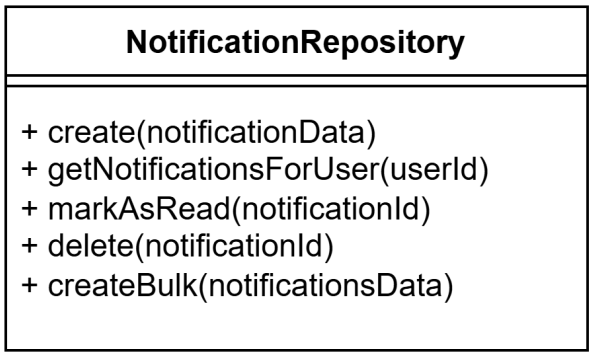
d. Class: AdminRepository



- **Description:** Manages data operations exclusive to administrators.
- **Methods:**
    - **banUser(userId, reason)**: Changes a user's status to 'suspended'.
    - **unbanUser(userId)**: Changes a user's status to 'active'.
    - **getTeacherRequests()**:Retrieves a collection of all pending teacher verification requests..
    - **approveTeacherRequest(requestId):** Approves a teacher request, updating the user's role.

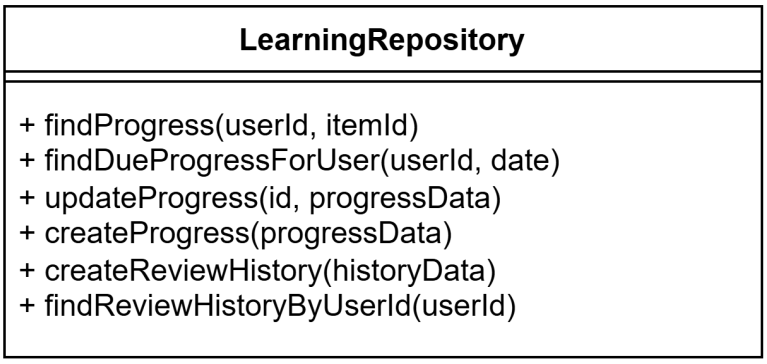
- **getReportedContent()**:Retrieves a list of content reports.
- **moderateContent(reportId, action)**: Processes a content report (e.g., remove content, dismiss report).
- **getAllUsers(filterOptions)**: Retrieves a paginated collection of all user objects.
- **createLogAction(adminLogData)**: Creates a new AdminLog record.
- **getAllAdminLog():** Retrieves a paginated collection of all AdminLog.

e. Class: NotificationRepository



- **Description:** Manages the creation and retrieval of notifications for users.
- **Methods:**
  - **create(notificationData)**: Creates a new Notification.
  - **getNotificationsForUser(userId)**: Retrieves all notifications for a user, usually with pagination.
  - **markAsRead(notificationId)**: Sets the isRead flag to true for one or more notifications.
  - **delete(notificationId)**: Deletes a notification.
  - **createBulk(notificationsData)**: Creates multiple notifications at once (e.g., for a new assignment).

f. LearningRepository



- **Description**: Manages data operations related to a user's learning journey, specifically their progress with vocabulary items (Spaced Repetition System) and their review history.
- **Methods**:
    - **findProgress(userId, itemId)**: Retrieves a single learning progress object for a specific user and vocabulary item.
    - **findDueProgressForUser(userId, date)**: Finds all learning progress records for a user that are due for review on or before a given date.
    - **updateProgress(id, progressData)**: Updates an existing learning progress record, typically after a review session.
    - **createProgress(progressData)**: Creates a new learning progress record when a user learns an item for the first time.
    - **createReviewHistory(historyData)**: Creates a new log entry in the review history.
    - **findReviewHistoryByUserId(userId)**: Retrieves a collection of all review history records for a specific user, often used for detailed learning analytics.

4.7 Component: External Services

Responsibilities

External services are responsible for providing extended functionalities outside of the core application, including:

- **Google OAuth**: Authenticates and authorizes users through their Google accounts, supporting both login and registration via OAuth.
- **Google Gemini AI Service**: Powers AI-driven learning features such as generating example sentences, custom questions, and other learning assistance.
- **Email Notification System (SMTP)**: Sends account verification emails, password reset notifications, class invitations, and broadcast messages.

4.7.1 Google OAuth

- **Integration Approach**: Implemented using Passport.js with the passport-google-oauth20 strategy, or custom-built OAuth2 endpoints.

- **Endpoints**:
  - GET /api/auth/google: Initiates the OAuth flow and redirects to the Google consent screen.
  - GET /api/auth/google/callback: Receives the authorization code from Google and exchanges it for an access token and user profile.
- **Scopes**: profile, email.
- **Token Handling**:
  - Store access_token / refresh_token (if required) in the database (via a Token model).
  - Create a JWT session based on the retrieved Google profile.
- **Error Handling**:
  - 400 Bad Request: Missing or improperly formatted callback code
  - 401 Unauthorized: Invalid token or user not found
  - 500 Internal Server Error: Token exchange failure

4.7.2 Google Gemini service

- **Integration Approach**: Create an AIService that communicates with the Google Gemini REST API via HTTP.
- **Configuration**:
  - AI_API_KEY is stored in environment variables.
  - Base URL: https://gemini.googleapis.com/v1/models/text-bison-001:generateText.
- **Use Cases**:
  - Generate example sentences for new vocabulary words (VocabularyController)
  - Suggest answers for fill-in-the-blank exercises (ReviewController)
- **Request/Response Flow**:
  - Controller calls AIService.generateExample(word).
  - AIService builds the payload and sends an HTTP POST request to Gemini
  - Handles the response and extracts candidates[].content.
  - Returns the result back to the controller
- **Error Handling & Retry**:
  - 429 Too Many Requests: Retry using exponential backoff
  - 5xx: Log the error and return a fallback response or client error

4.7.3 Email Notification System

- **Integration Approach**: Use an EmailService built with the nodemailer library to send emails directly via an SMTP server.
- **Configuration**:
  - SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_PASS are configured via .env.
  - EMAIL_FROM specifies the default sender address.
- **Templates**: Use handlebars or ejs libraries to render email templates
- **Use Cases**:
  - Account verification (AuthController)

- Password reset (AuthController)
- Invitations (ClassroomController)
- Broadcast (AdminController).

- **Error Handling**:
  - 401 Authentication Failed: Incorrect SMTP credentials
  - 550 Relay Denied: Invalid recipient address
  - 5xx Server Error: Log the error, optionally retry

4.8 Component: Database (Data Access Layer)

The **VocaBoost Database Schema** provides the complete data backbone for VocaBoost, implemented on Supabase Postgres for scalability and reliability. It manages structured storage for users, vocabulary, learning progress, classrooms, and system utilities, ensuring secure, efficient data access and consistency to power VocaBoost's personalized learning experience.

Full ER Diagram: ER Diagram.png

4.8.1 User management

Key responsibilities:

- Storing user account details, including email, hashed passwords, Google ID, and role assignments.
- Tracking user status (active, pending verification, suspended) for access control.
- Managing user settings such as daily learning goals, preferred language, theme, and timezone.
- Recording user learning statistics, including vocabulary learned, review counts, streaks, and total study time.
- Handling user deactivation, including deactivation reasons and tracking who performed the deactivation.
- Supporting teacher role requests with verification workflows for classroom management.
- Managing authentication tokens for email verification and password resets.

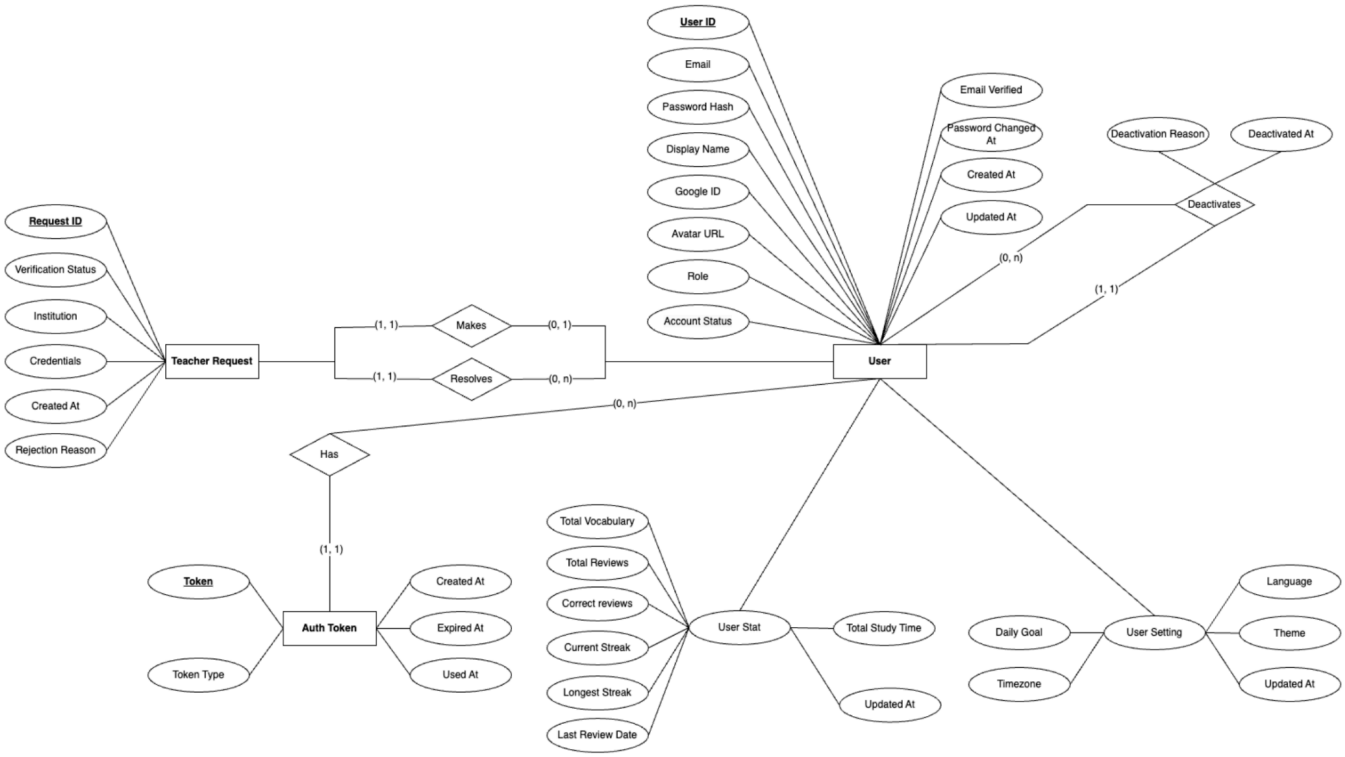


Table Overviews:

- **User:** Stores user account details including email, hashed passwords, Google ID, role assignments, and account status for managing user identities and access control.
- **User Settings:** Stores user-specific preferences such as daily learning goals, preferred language, theme, and timezone, allowing personalized learning experiences.
- **User Stats:** Records user learning statistics including total vocabulary learned, review counts, streaks, and total study time, supporting personalized progress tracking.
- **User Deactivation:** Tracks user deactivation events with reasons and who performed the deactivation, maintaining an audit trail for moderation and security.
- **Teacher Request**: Stores teacher role verification requests with status tracking and optional rejection reasons, enabling controlled elevation of user roles for classroom management.
- **Auth Token:** Manages authentication tokens for actions like email verification and password resets, supporting secure user authentication workflows.

4.8.2 Vocabulary Management

Key responsibilities:

- Storing vocabulary lists with metadata including titles, descriptions, word counts, and privacy settings.
- Managing individual vocabulary words with definitions, phonetics, and optional images for richer learning.
- Allowing multiple example sentences for each word to improve contextual understanding.
- Storing synonyms for words to aid in advanced vocabulary practice and association.
- Supporting a tagging system for vocabulary lists to enhance organization and searchability.

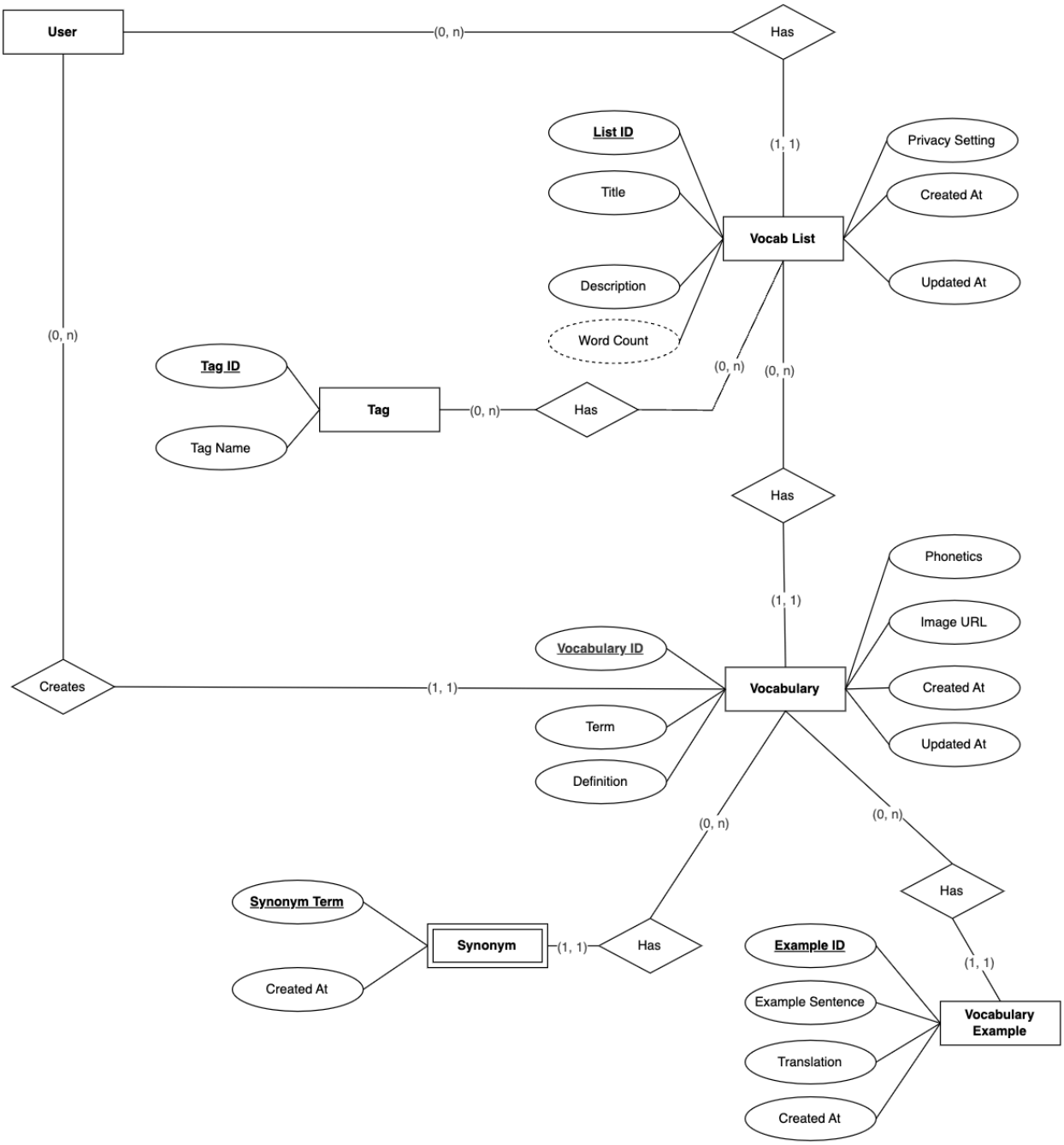

User
(0, n)
Has
(1, 1)
List ID
Title
Description
Word Count
Vocab List
Privacy Setting
Created At
Updated At
(0, n)
Tag ID
Tag Name
Tag
(0, n)
Has
(0, n)
(0, n)
Has
(1, 1)
Phonetics
Image URL
Created At
Updated At
Vocabulary ID
Term
Definition
Vocabulary
Creates
(1, 1)
(0, n)
(0, n)
Has
(1, 1)
Synonym Term
Created At
Synonym
(1, 1)
Has
Example ID
Example Sentence
Translation
Created At
Vocabulary Example

Table Overviews:

- **Vocab List**: Stores vocabulary lists created by users with metadata including titles, descriptions, word counts, privacy settings, and active status.
- **Vocabulary**: Stores individual vocabulary words within lists, including term, definition, phonetics, and optional images, forming the core learning units.
- **Vocabulary Example**: Stores multiple example sentences and optional translations for each vocabulary word, helping learners understand usage in context.
- **Synonym**: Stores synonyms associated with vocabulary words, supporting advanced vocabulary association and learning activities.
- **Tag**: Stores reusable tags for categorizing vocabulary lists to improve organization and filtering.
- **List Tag**: Associates tags with specific vocabulary lists, enabling structured categorization and advanced search filtering.

4.8.3 Revision and Progress Tracking

Key responsibilities:

- Tracking individual learner progress for each vocabulary word, including review intervals and correctness history for effective spaced repetition.
- Recording structured review sessions, capturing start and completion times, scores, and session types for learning analytics.
- Storing detailed results for each word reviewed within a session, including correctness and timestamps, to support granular progress tracking and reporting.

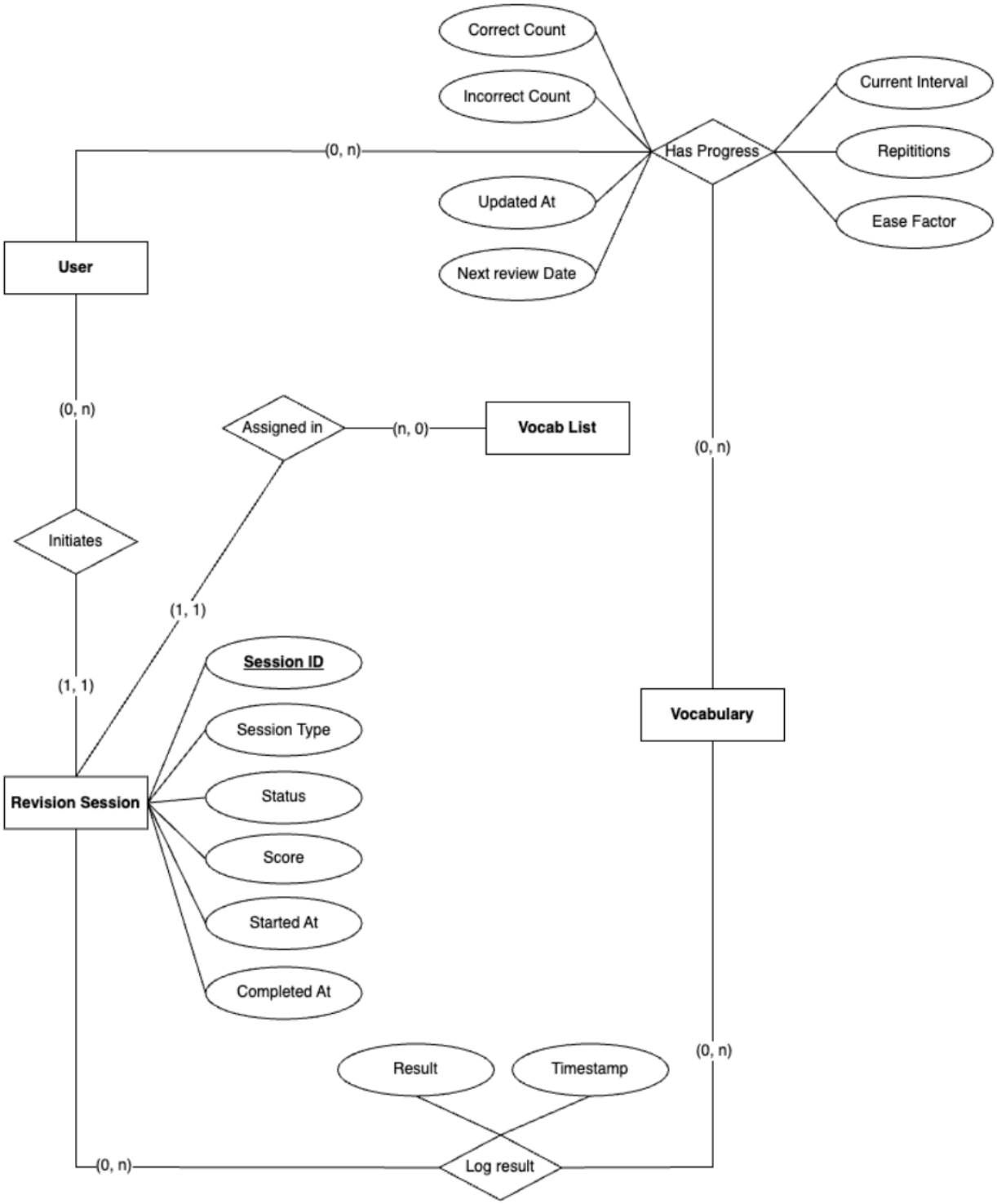

Correct Count
Incorrect Count
Updated At
Next review Date
Has Progress
Current Interval
Repitions
Ease Factor
(0, n)
User
(0, n)
Assigned in
(n, 0)
Vocab List
(0, n)
Initiates
(1, 1)
(1, 1)
Session ID
Session Type
Status
Score
Started At
Completed At
Revision Session
Vocabulary
(0, n)
Result
Timestamp
(0, n)
Log result

Table Overviews:

- **User Word Progress:** Tracks user-specific progress for each vocabulary word, including next review dates, review intervals, ease factors, repetition counts, and correctness history to power personalized spaced repetition.
- **Revision Session:** Records review sessions for learners, including session type, associated vocabulary list, assignment link (if applicable), session status, scores, and timestamps for tracking learning activities.
- **Session Word Result:** Stores the results of individual vocabulary words reviewed during a revision session, capturing correctness and review timestamps to provide detailed session analytics.

4.8.4 Classroom and Assignment

Key responsibilities:

- Managing classrooms created by teachers, including descriptions, join codes, capacity limits, and privacy status.
- Handling learner membership within classrooms, including invitations, join requests, and membership statuses.
- Enabling teachers to create and manage assignments linked to vocabulary lists within classrooms for structured practice.
- Supporting sublist management within assignments to break down larger vocabulary lists into manageable learning chunks.
- Tracking learner-specific assignment progress, completion status, scores, and completion timestamps.
- Managing secure invitation workflows for learners to join classrooms using token-based invitations.

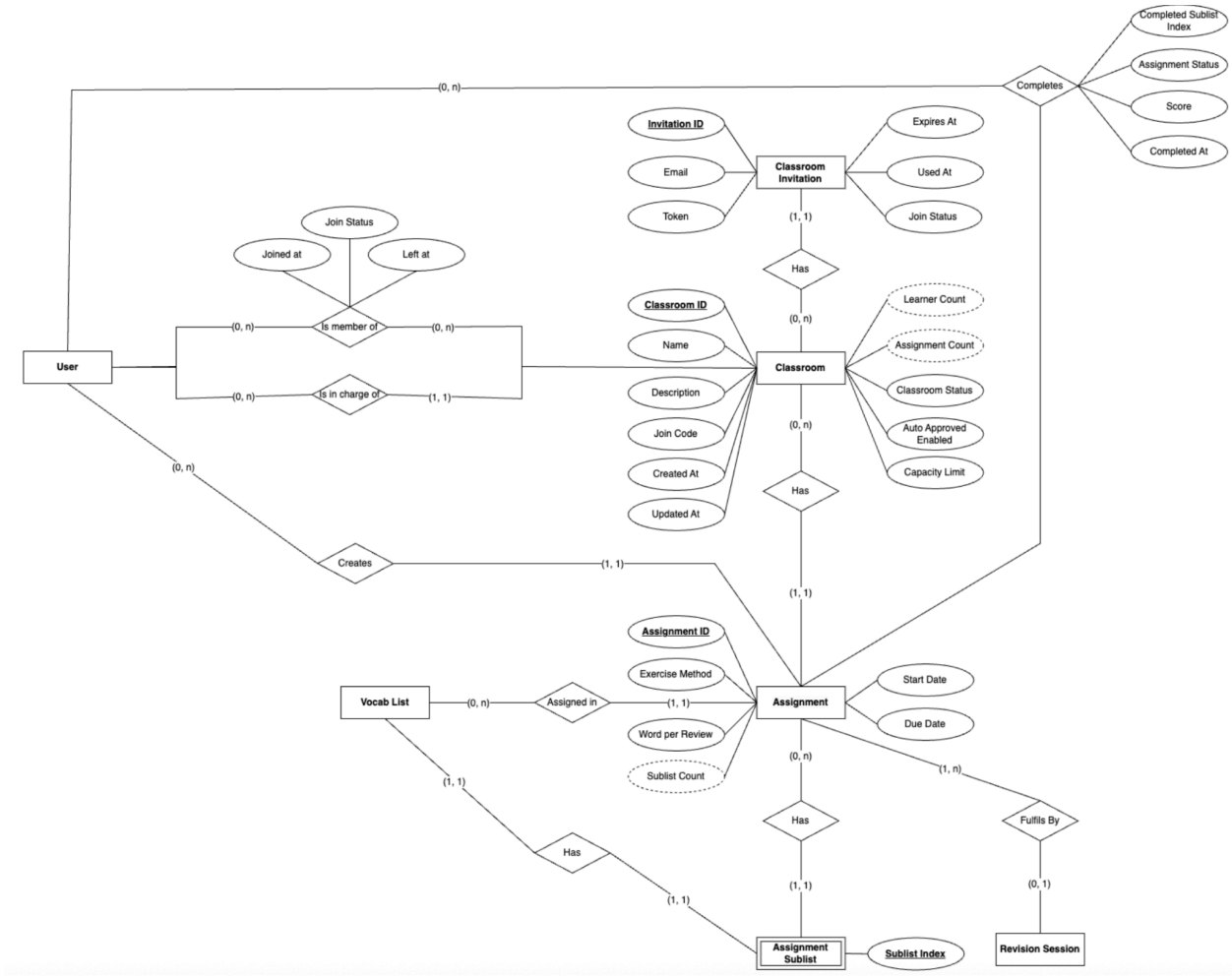


Table Overviews:

- **Classroom**: Stores classroom entities managed by teachers, including names, descriptions, join codes, learner counts, capacity limits, and classroom status for structured learning groups.
- **Classroom Member**: Manages learner membership within classrooms, tracking join statuses, invitation workflows, and membership timestamps.
- **Assignment**: Stores assignments created by teachers within classrooms, including associated vocabulary lists, exercise methods, word counts per review, and due dates to structure learner activities.
- **Assignment Sublist**: Manages sublist structures within assignments, allowing large vocabulary lists to be divided into smaller, trackable subsets for easier learning.
- **Learner Assignment**: Tracks learner-specific assignment progress, including completed sublists, status, scores, and completion timestamps for detailed tracking of learner performance.
- **Classroom Invitation**: Manages secure, token-based invitation workflows for learners to join classrooms, supporting email-based invitations and tracking their usage and statuses.

4.8.5 System Utilities and Notifications

Key responsibilities:

- Managing user-submitted reports on vocabulary content for quality control and moderation

workflows.

- Storing audit logs of user and admin actions for transparency, debugging, and accountability.
- Managing notifications sent to users, including messages, metadata, and read statuses to support in-app alerts and user engagement.

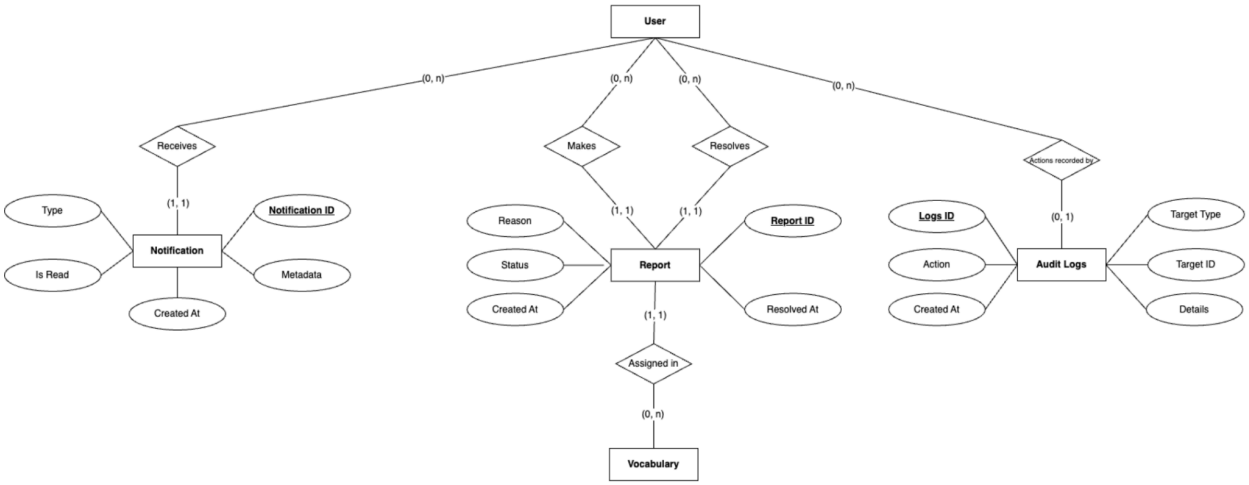


Table Overviews:

- **Report**: Stores user-submitted reports related to vocabulary content, including the reporter, reported word, reasons, statuses, and resolver references for moderation and quality control.
- **Audit Logs**: Stores logs of user and admin actions within the system, including action types, target entities, and JSON details to provide an auditable trail for transparency and debugging.
- **Notification**: Manages in-app notifications for users, including recipient tracking, message contents, metadata, read status, and timestamps to support timely user engagement and alerts.

5. Deployment

[This section will be completed in PA4 as specified in the template requirements]

6. Implementation View

[This section will be completed in PA4 as specified in the template requirements]