Group06

VocaBoost
Software Architecture Document

Version 1.2

Revision History

Date	Version	Description	Author
23/06/2025	1.0	Initial Software Architecture Document creation	All team members
20/07/2025	1.1	Complete Implementation View and Deployment Diagram	All team members
03/08/2025	1.2	Revise SAD base on current codebase of modules: Auth, Vocabulary, Classroom and User	All team members

Table of Contents

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

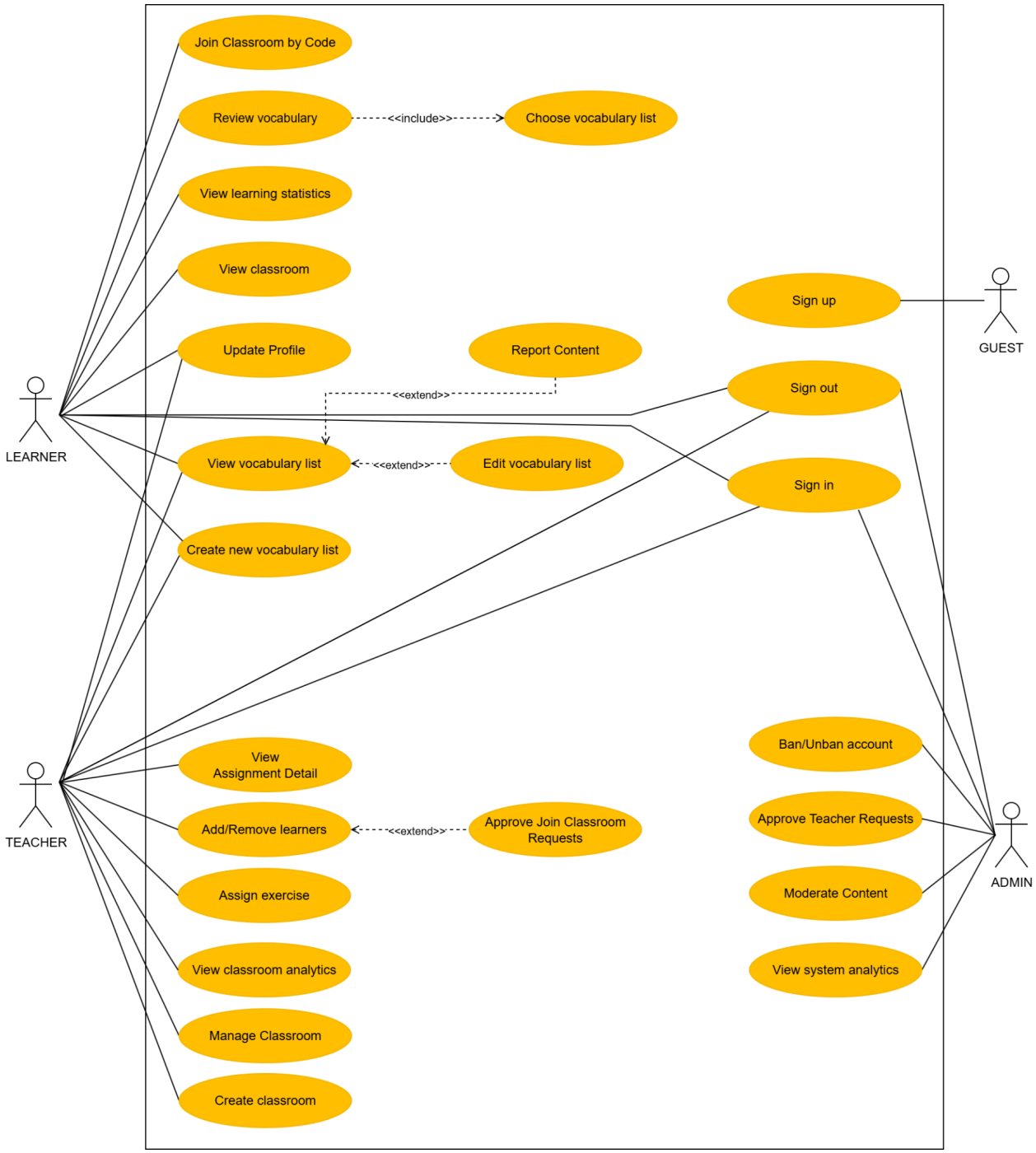
Architectural Goals

- 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.

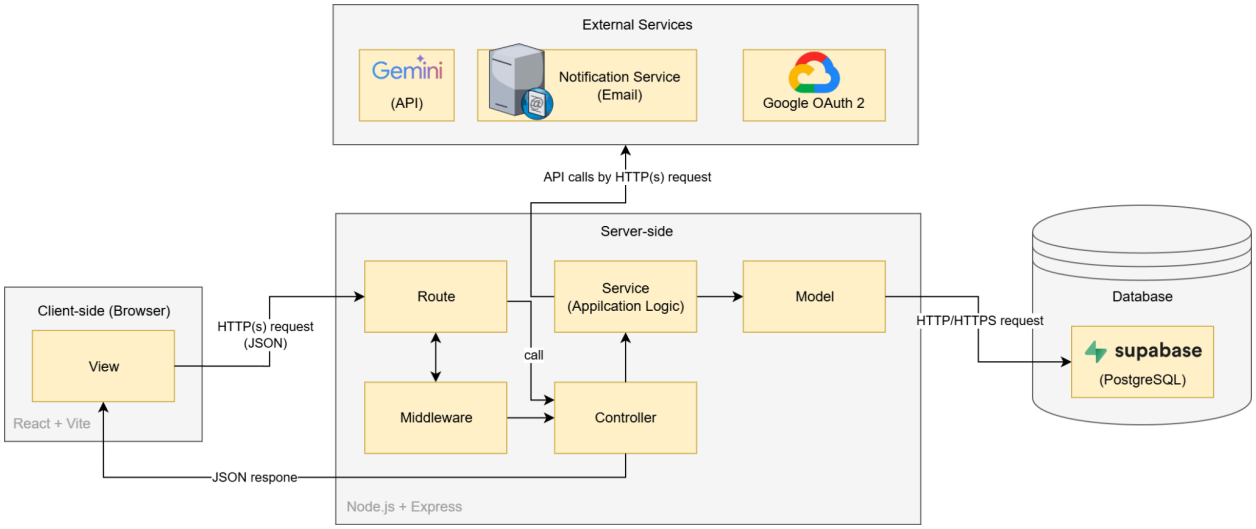
Technical Constraints

- 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

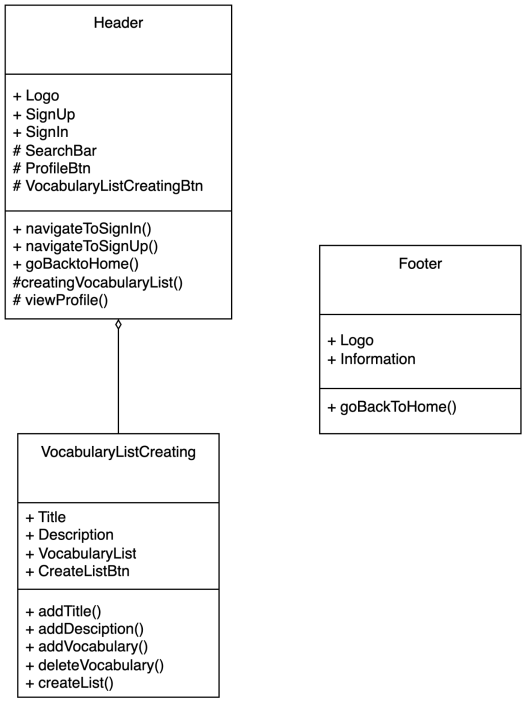
Model	Data access via Supabase JS.	VocabularyRepo.findById(id)
External Service	OAuth, Gemini AI, SMTP Email	API endpoints
Database	Normalised PostgreSQL schema (ERD in 4.7 Component: Database).	SQL interface

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

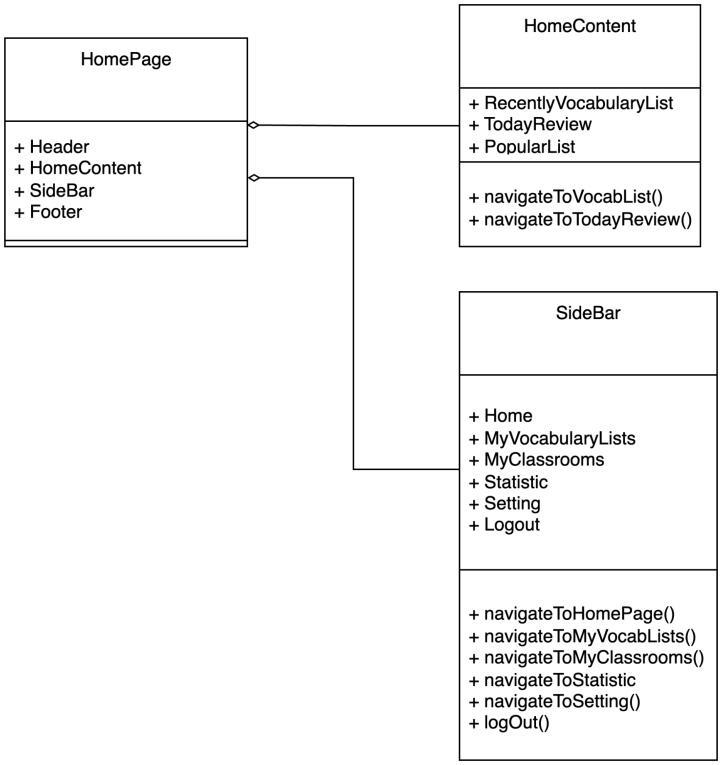


Description: small interface sections that are separated into their own files to be reused across multiple pages in a web application.

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
- **Footer** displays the bottom section of the website provides extra information and support links.

4.1.2 Home Page

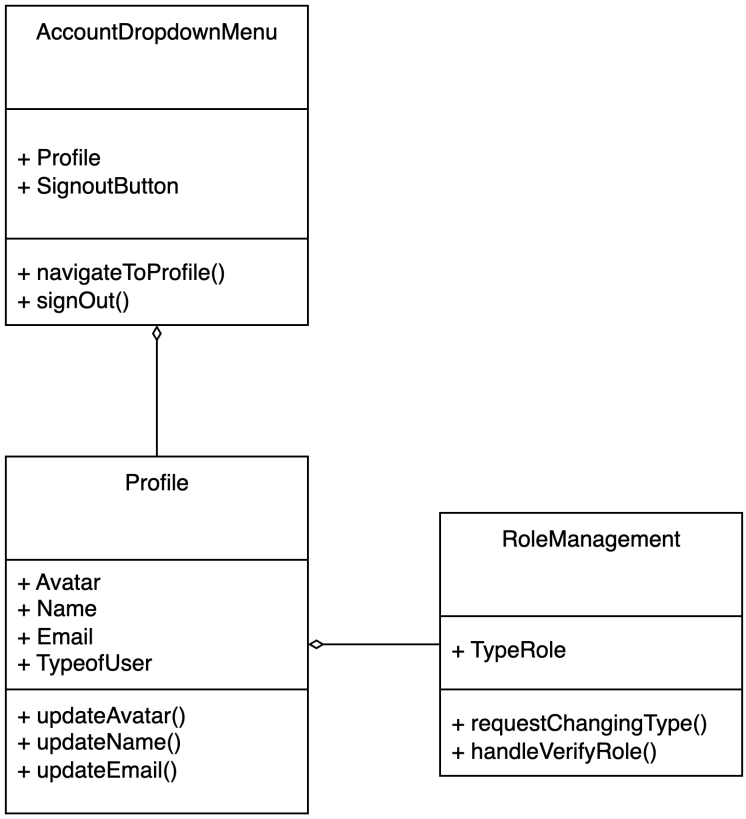


Description: represent the home screen of the website.

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 Dropdown Menu

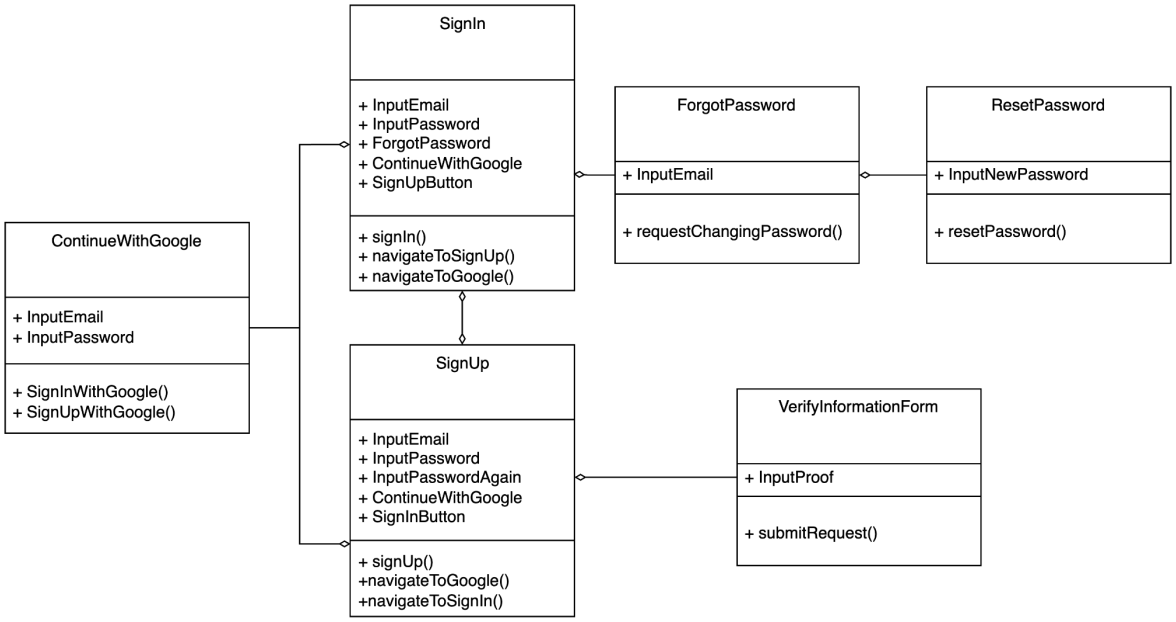


Description: handling user authentication, authorization, and profile personalization.

Classes:

- **AccountDropdownMenu** 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

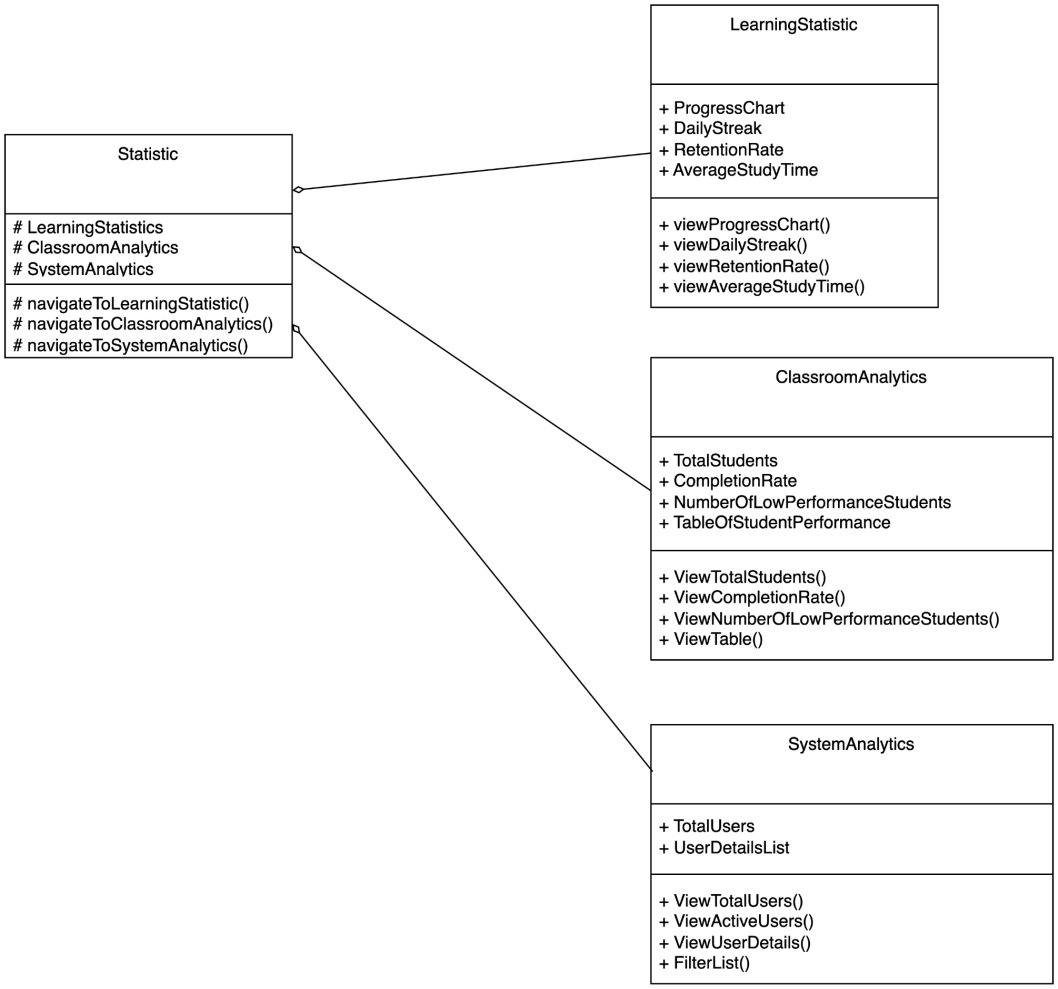


Description: responsible for user access and identity establishment in the application.

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

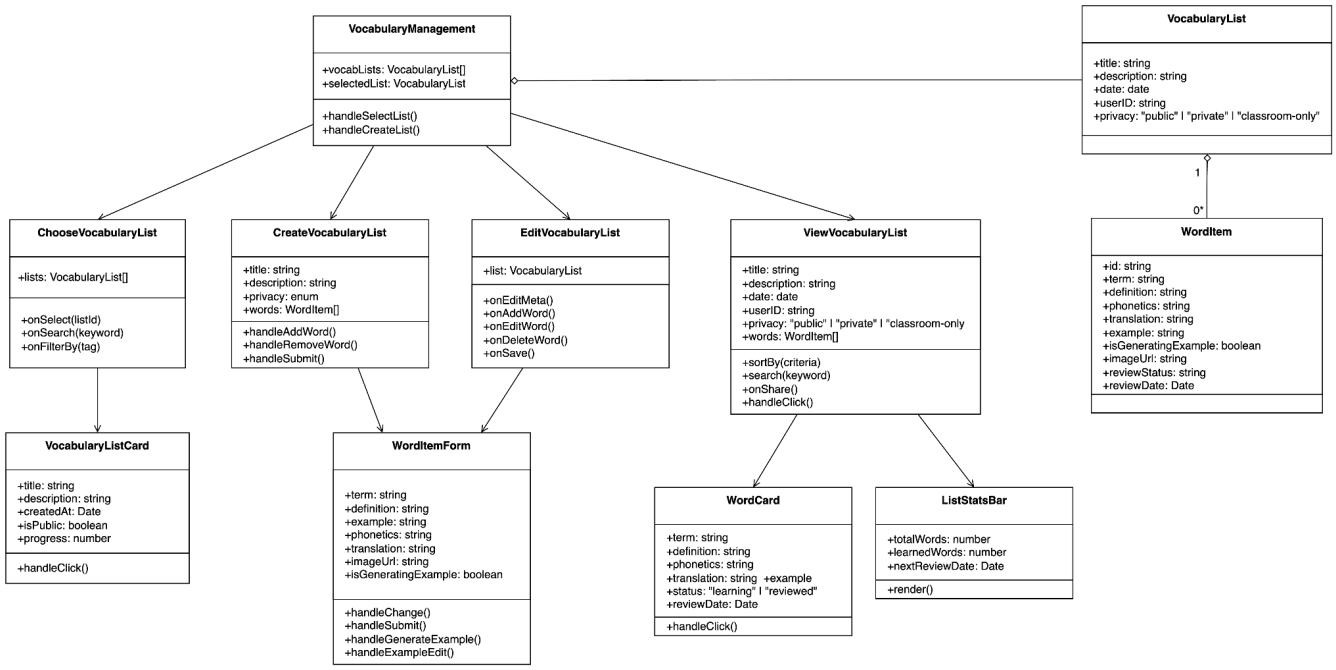


Description: responsible for user access and identity establishment in the website.

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

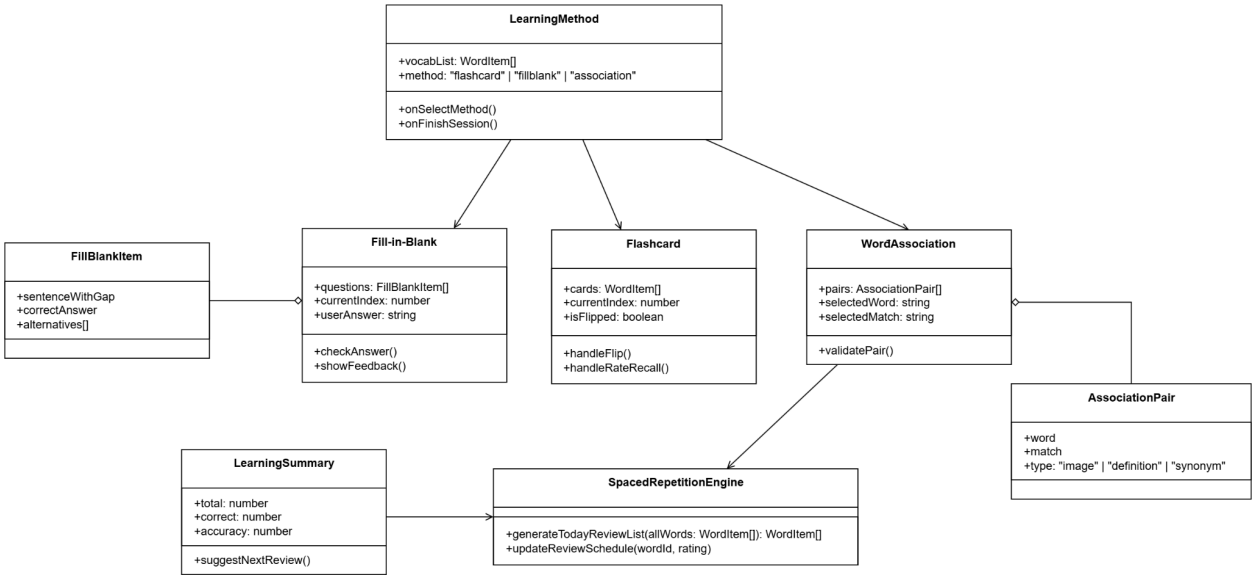


Description: responsible for enabling users to create, edit, view, and organize vocabulary lists within the VocaBoost system. It empowers users to manage their personal vocabulary collections, track learning progress, and leverage AI-generated example sentences to enhance learning effectiveness.
Role: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

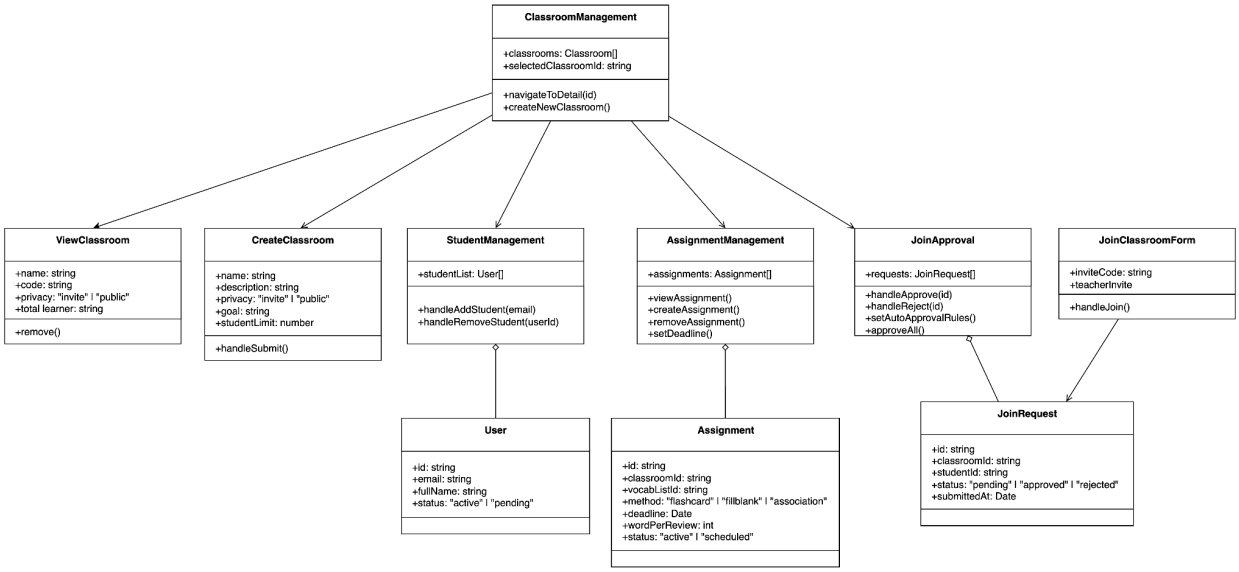
Role: Learner

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:

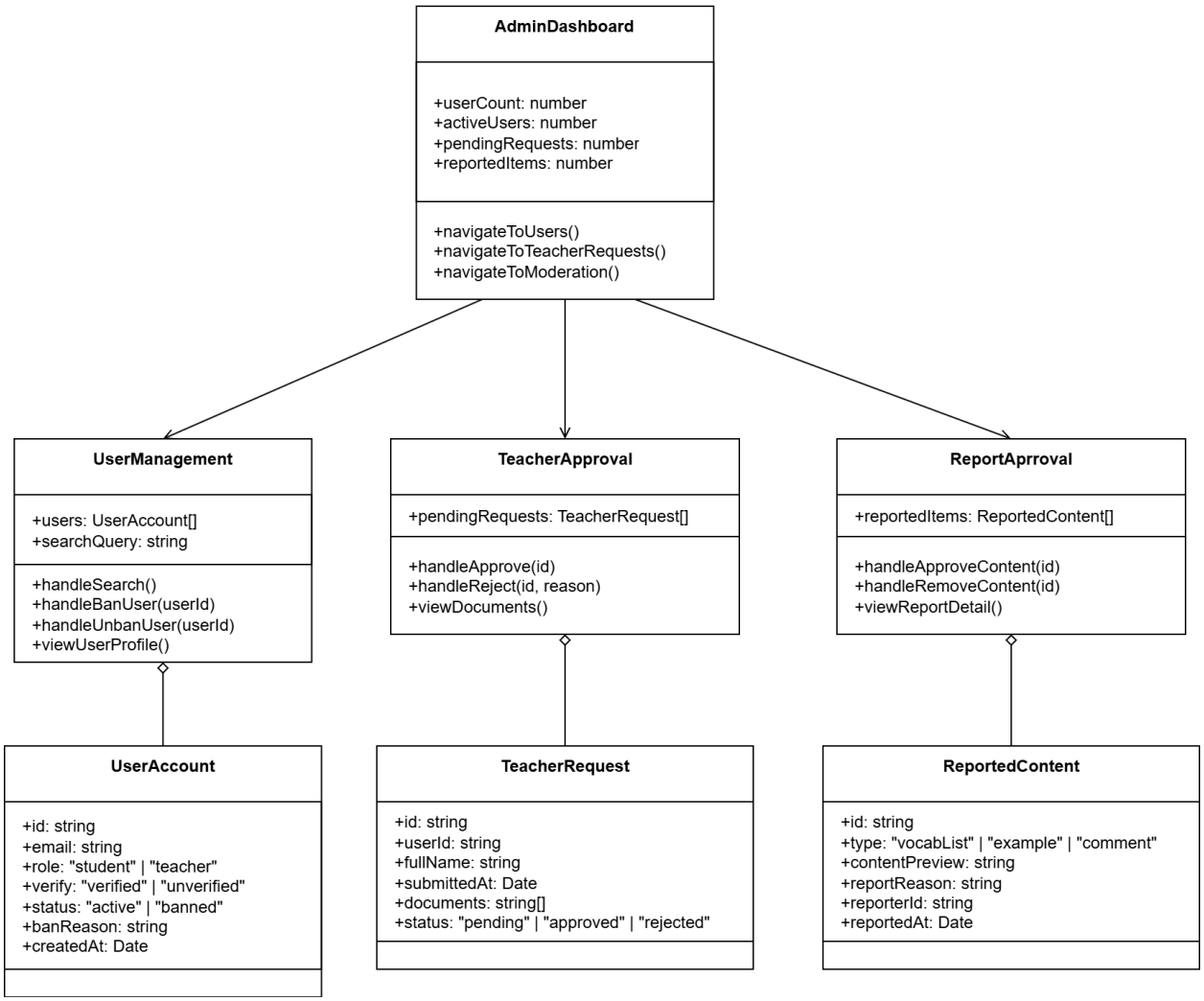
The component is composed of multiple interconnected classes that serve specific responsibilities:

- **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

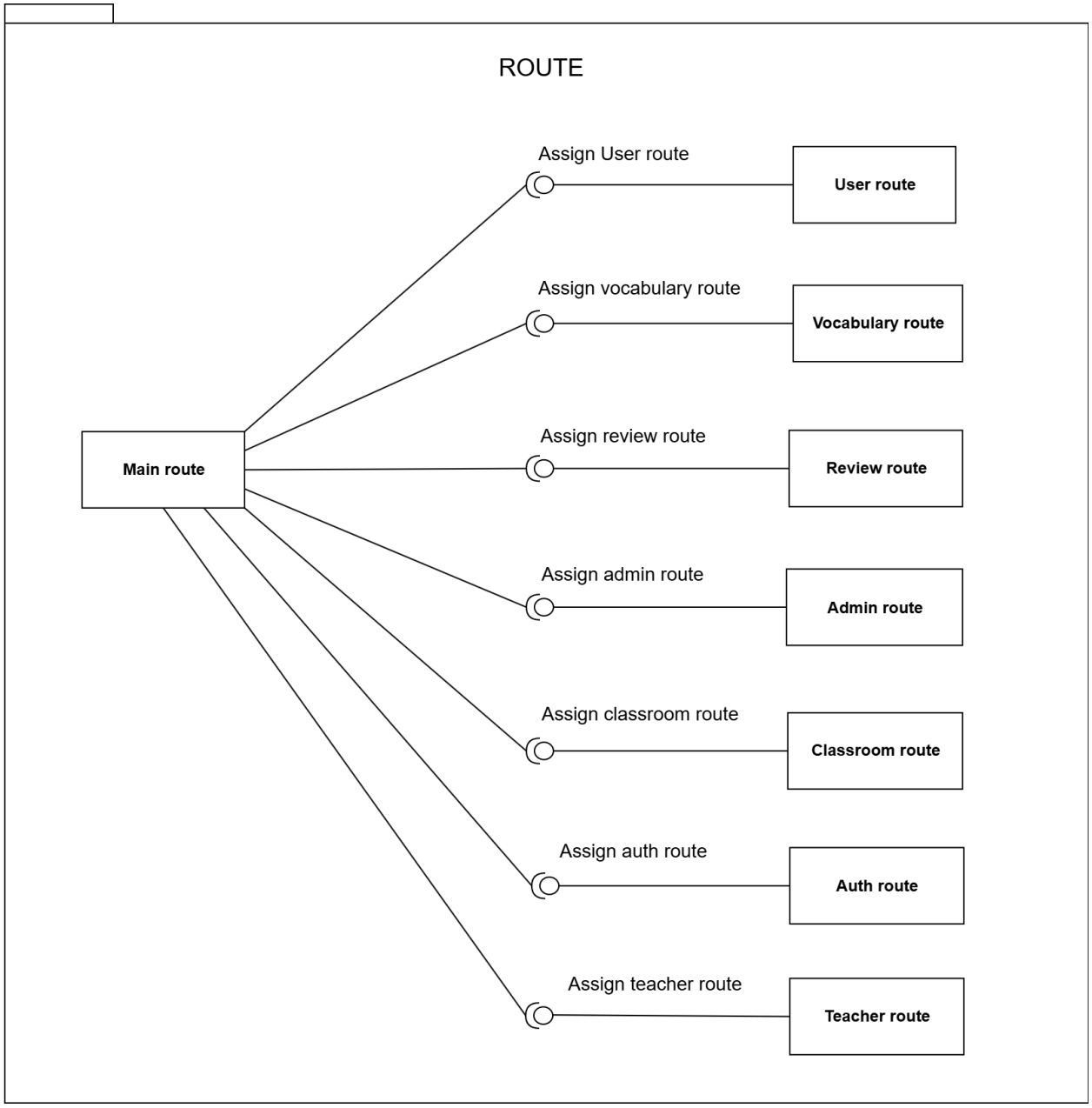
Class:

- **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 7 controllers AuthController, VocabularyController, AdminController, ClassroomController, ReviewController, UserController and TeacherController so there will be 7 corresponding routes. There will be a Main route responsible for categorizing requests and assigning them to one of the 7 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 VocabularyModel to process data as requested. When done,

the controller returns data, through the routes, to the client.

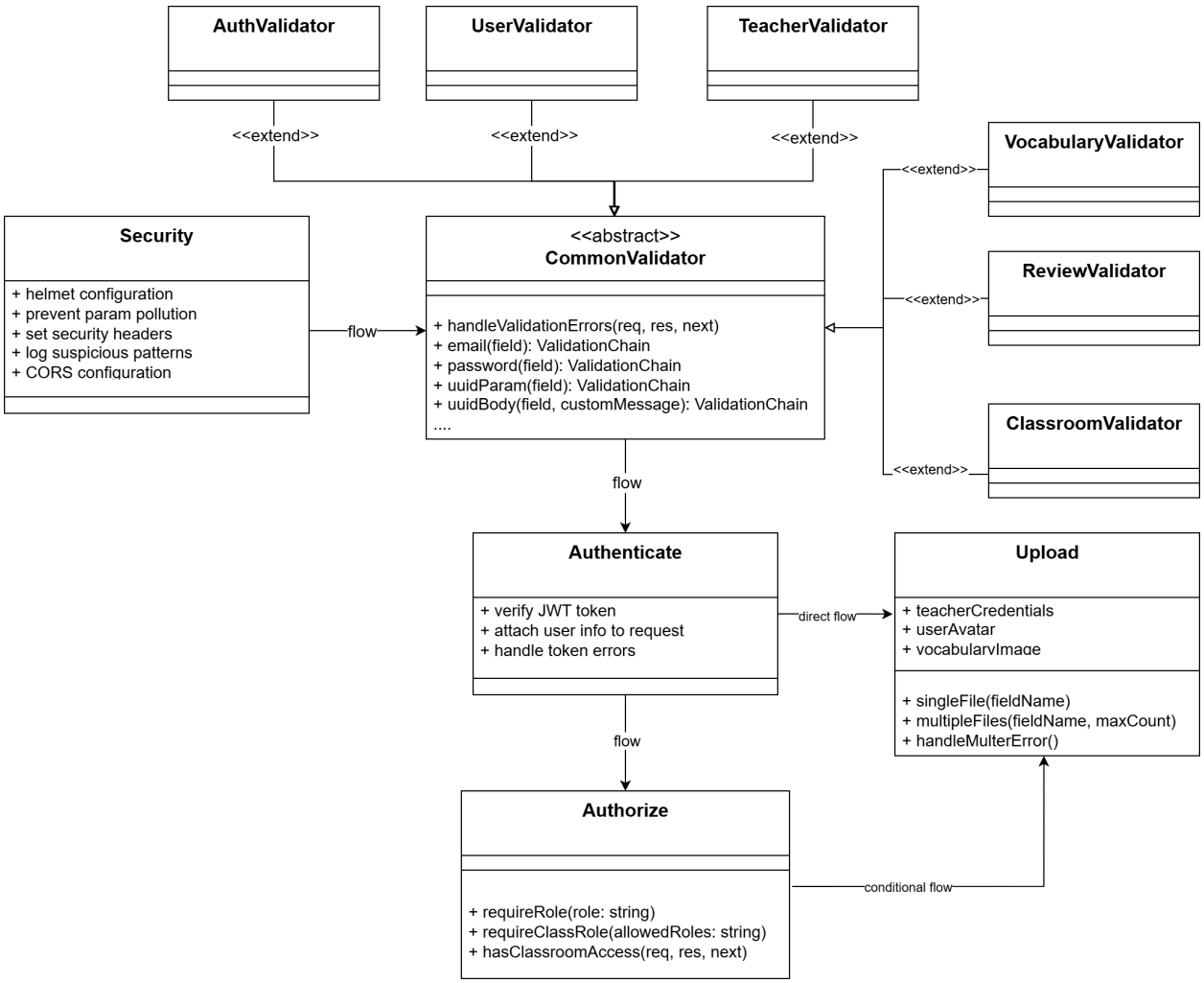
4.3 Component: Middleware (Business Logic Layer)

4.3.1 Responsibilities

Middleware components intercept and process requests before they reach controllers, handling cross-cutting concerns such as authentication, authorization, validation, security, and file uploads.

- **Authentication & Authorization** (JWT, Google OAuth, role-based)
- **Validator** (input checking, error collection)

4.3.2 Class diagram



Middleware flow patterns:

Pattern	**Middleware flow**	**Example endpoint**	**For endpoints that**
Public Endpoints	1. Security 2. XValidator 2. Controller	/health, /api/auth/login	No authentication required
Authenticated	1. Security 2. XValidator 3. Authenticate 4. Controller	/api/user/profile	Basic user authentication
Role-based Access	1. Security 2. XValidator 3. Authenticate 4. Authorize 5. Controller	/api/classroom/create-classroom	Requires specific role: learner/ teacher/ admin/ owner/…
File Upload (Basic)	1. Security 2. XValidator 3. Authenticate 4. Upload 5. Controller	/api/teacher/verification/submit	User credentials + file upload
Complex Authorization + Upload	1. Security 2. XValidator 3. Authenticate 4. Authorize 5. Upload 6. Controller	/api/classroom/:id/assignment (with file)	Sensitive endpoints

4.3.3 Key Middleware Classes

- **Security**: Provides comprehensive security protection for all requests like: helmet for security headers, CORS configuration, parameter pollution prevention, XSS protection,...
- **CommonValidator** (abstract): provides a shared handler and basic validation rules (UUID, email, password, pagination,…).
- **XValidator** (Auth/User/Teacher/Vocabulary/Review/Classroom): extend CommonValidators, adding specific schemas/logic for each route group.
- **Authenticate**: JWT token verification using the jwt.helper
- **Authorize:** responsible for role-based access control
- **Upload:** responsible for MIME type checking and file size limits

4.3.4 Detailed Method Descriptions

a) Class: SecurityMiddleware

1. securityMiddleware(app)

- **Parameters**: Express app instance
- **Description**: Configures comprehensive security middleware for the application
- **Features**:
    - Helmet for security headers (CSP disabled in development)
    - X-Powered-By header removal
    - Parameter pollution prevention
    - Basic XSS protection headers
    - CORS with origin validation
    - Suspicious pattern detection and logging
- **Implementation**: Applied globally to all routes

b) Class: AuthenticateMiddleware

1. authenticateMiddleware(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Validates JWT token from Authorization header and attaches user context
- **Error Responses**:
    - 401: "No token provided" - Missing Authorization header
    - 401: "Invalid token" - Malformed or invalid JWT
    - 401: "Token expired" - Expired JWT
    - 401: "User not found" - User no longer exists
    - 403: "Account {status}" - Inactive/suspended/banned account
- **Implementation**: Applied globally to all routes

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).

c) Class: AuthorizeMiddleware

1. requireRole(...roles)

- **Parameters**: ..roles - Array of allowed user roles

- **Returns**: Middleware function
- **Description**: Factory function creating role-based access control middleware
- **Supported Roles**: ‘learner’, ‘teacher’, ‘admin’
- **Error Responses**:
    - 401: "Authentication required" - Missing req.user
    - 403: "Insufficient permissions" - User role not in allowed roles

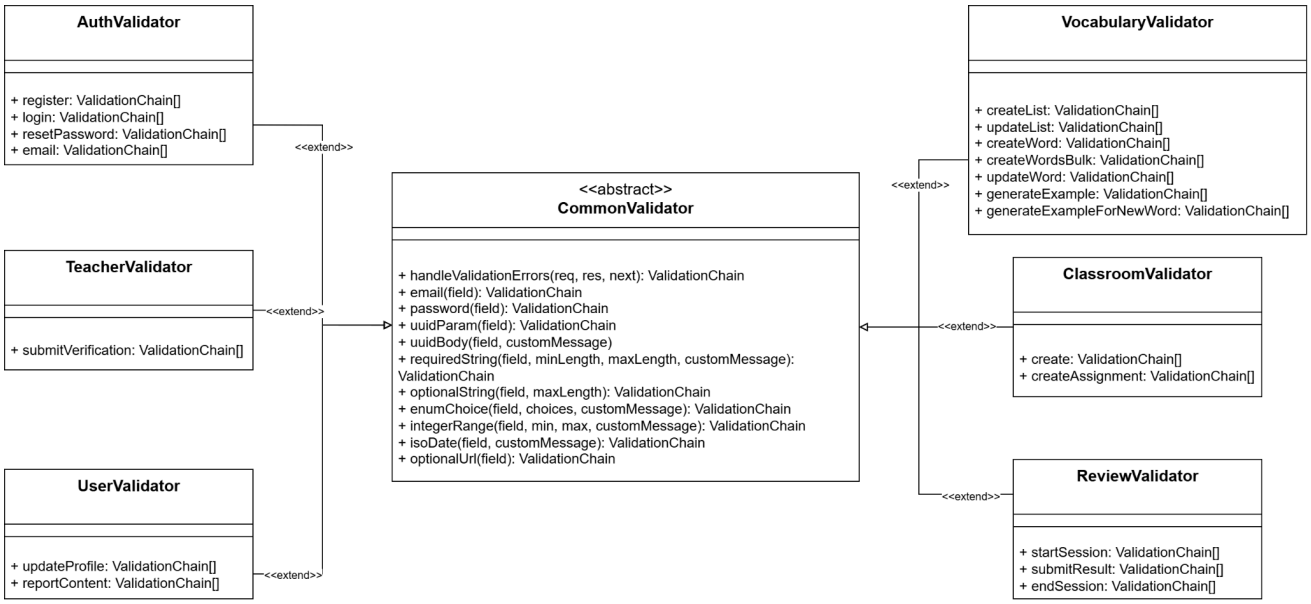
2. hasClassroomAccess(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Complex classroom access permission checking
- **Access Rules**:
    - Admin: Full access to all classrooms
    - Teacher: Access to owned classrooms
    - Learner: Access to joined classrooms only
- **Process Flow**:
    1. Extract classroomId from req.params
    2. Fetch classroom from database
    3. Check classroom status (deleted classrooms are inaccessible)
    4. Verify user access based on role and relationship
    5. Attach classroom and user role to request context
- **Error Responses**:
    - 404: "Classroom not found or has been deleted"
    - 403: "You do not have access to this classroom"

3. requireClassRole(...allowedRoles)

- **Parameters**: …allowedRoles - Array of allowed classroom roles
- **Returns**: Middleware function
- **Description**: Classroom-specific role authorization (must be used after hasClassroomAccess)
- **Supported Roles**: ‘learner’, ‘teacher’, ‘admin’
- **Error Responses**: 403: "Insufficient permissions"

d) Class: CommonValidator



1. handleValidationErrors(req, res, next)

- **Parameters**: Express request, response, next middleware
- **Description**: Checks the validation results from express-validator. If there are any errors, return 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
- **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
- **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
- **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. uuidParam(field: string)

- **Parameters**: field: name of the parameter to validate as UUID v4
- **Returns**: express-validator chain
- **Description**: Validates that the provided parameter is a valid UUID version 4
- **Error Message**: “Invalid UUID format for {field}.”

6. uuidBody(field: string, customMessage?: string)

- **Parameters**:
  - field: name of the body field to validate as UUID v4
  - customMessage (optional): custom error message
- **Returns**: express-validator chain
- **Description**: Validates that the provided body is a valid UUID version 4
- **Error Message**: “Invalid UUID format for {field}.”

7. requiredString(field: string, minLength: number, maxLength: number, customMessage?: string)

- **Parameters**:
  - field: name of the required string field
  - minLength: minimum allowed length
  - maxLength: maximum allowed length
  - customMessage (optional): custom error message
- **Returns**: express-validator chain
- **Description**: Validates that the provided string field is non-empty, trimmed, and within the specified length range
- **Error Message**: “{field} must be between {minLength} and {maxLength} characters.”

8. optionalString(field: string, maxLength: number)

- **Parameters**:
  - field: name of the optional string field
  - maxLength: maximum allowed length
- **Returns**: express-validator chain for the param
- **Description**: Validates that the provided string field, if present, is trimmed and does not exceed the specified maximum length
- **Error Message**: "{field} must not exceed {maxLength} characters."

9. enumChoice(field: string, choices: string[], customMessage?: string)

- **Parameters**:
  - field: name of the field to validate
  - choices: array of allowed string values
  - customMessage (optional): custom error message
- **Returns**: express-validator chain for the param
- **Description**: Validates that the provided field value is one of the predefined enumerated choices
- **Error Message**: "{field} must be one of [{choices.join(', ')}]."

10. integerRange(field: string, min: number, max: number, customMessage?: string)

- **Parameters**:
  - field: name of the integer field
  - min: minimum allowed value
  - max: maximum allowed value
  - customMessage (optional): custom error message
- **Returns**: express-validator chain for the param
- **Description**: Validates that the provided integer field falls within the specified numeric range
- **Error Message**: "{field} must be between {min} and {max}."

11. isoDate(field: string, customMessage?: string)

- **Parameters**:
  - field: name of the date field to validate
  - customMessage (optional): custom error message
- **Returns**: express-validator chain for the param
- **Description**: Validates that the provided date field conforms to ISO 8601 format
- **Error Message**: "{field} must be a valid ISO 8601 date."

12. optionalUrl(field: string)

- **Parameters**: field: name of the URL field to validate
- **Returns**: express-validator chain for the param

- **Description**: Validates that the provided URL field, if present, is a well-formed URL.
- **Error Message**: "{field} must be a valid URL."

e) Class: AuthValidator

1. register

- **Type**: Middleware array
- **Validates**:
  - email: Required, valid email format, normalized
  - password: Required, 8+ characters with uppercase, lowercase, and number
  - role: Optional, must be 'learner' or 'teacher'
- **Includes**: handleValidationErrors

2. login

- **Type**: Middleware array
- **Validates**:
  - email: Required, valid email format
  - password: Required, non-empty
- **Includes**: handleValidationErrors

3. resetPassword

- **Type**: Middleware array
- **Validates**:
  - token: Required, non-empty JWT token (reset token)
  - newPassword: Required, same strength requirements as registration
- **Includes**: handleValidationErrors

f) Class: UserValidator

1. updateProfile

- **Type**: Middleware array
- **Validates**:
  - displayName: Optional, 1-100 characters
  - removeAvatar: Optional, boolean
  - dailyGoal: Optional, integer 1-100
- **Includes**: handleValidationErrors

2. reportContent

- **Type**: Middleware array

- **Validates**:
    - wordId: Required, valid UUID
    - reason: Required, 1-500 characters
- **Includes**: handleValidationErrors

g) Class: TeacherValidators

1. submitVerification

- **Type**: Middleware array
- **Validates**:
    - fullName: Required, 1-100 characters
    - institution: Required, 1-200 characters
    - schoolEmail: Required, valid email format
    - additionalNotes: Optional, max 1000 characters
- **Includes**: handleValidationErrors

h) Class: VocabularyValidators

1. createList

- **Type**: Middleware array
- **Validates**:
    - title: Required, 1-200 characters, trimmed
    - description: Optional, max 1000 characters
    - privacy_setting: Optional, 'public' or 'private'
    - tags: Optional, array of existing tag names
- **Includes**: handleValidationErrors

2. createWord

- **Type**: Middleware array
- **Validates**:
    - term: Required, 1-100 characters
    - definition: Required, 1-1000 characters
    - translation: Optional, max 500 characters
    - phonetics: Optional, max 200 characters
    - exampleSentence: Optional, max 500 characters
    - synonyms: Optional, array of strings
- **Includes**: handleValidationErrors

i) Class: ReviewValidators

1. startSession

- **Type**: Middleware array
- **Validates**:
  - listId: Required, valid UUID
  - sessionType: Required, valid session type
- **Includes**: handleValidationErrors

2. submitResult

- **Type**: Middleware array
- **Validates**:
  - wordId: Required, valid UUID
  - result: Required, 'correct' or 'incorrect'
  - responseTimeMs: Optional, positive integer
- **Includes**: handleValidationErrors

3. endSession

- **Type**: Middleware array
- **Validates**:
  - sessionId: Required, valid UUID (from params)
- **Includes**: handleValidationErrors

j) Class: ClassroomValidators

1. create

- **Type**: Middleware array
- **Validates**:
  - name: trim, length 1-100 chars
  - description: optional, trim, max 500 chars
  - classroom_status: Optional, 'public' or 'private'
  - capacity_limit: Optional, integer 1-100
- **Includes**: handleValidationErrors

2. createAssignment

- **Type**: Middleware array
- **Validates**:
  - vocabListId: Required, valid UUID
  - title: Required, 1-200 characters
  - exerciseMethod: Required, 'flashcard', 'fill_blank', or 'word_association'
  - wordsPerReview: Required, integer 5-30

    - startDate: Required, valid ISO date
    - dueDate: Optional, valid ISO date after startDate
- **Includes**: handleValidationErrors

i) Class: UploadMiddleware

1. teacherCredentials

- **Type**: Middleware array with multer configuration
- **Configuration**:
    - Single file upload with field name 'teacher-credentials'
    - Allowed MIME types: images, PDFs, Word documents
    - Maximum file size: 10MB

2. userAvatars

- **Type**: Middleware array with multer configuration
- **Configuration**:
    - Single file upload with field name 'user-avatars'
    - Allowed MIME types: image formats only
    - Maximum file size: 5MB

3. vocabularyImages

- **Type**: Middleware array with multer configuration
- **Configuration**:
    - Single file upload with field name 'vocabulary-images'
    - Allowed MIME types: image formats only
    - Maximum file size: 5MB

3. handleMulterError(err, req, res, next)

- **Parameters**: Error object, Express req, res, next
- **Description**: Comprehensive multer error handling
- **Error Types Handled**:
    - LIMIT_FILE_SIZE: "File size exceeds the maximum allowed limit"
    - LIMIT_FILE_COUNT: "Too many files uploaded"
    - LIMIT_UNEXPECTED_FILE: "Unexpected field name"
    - Custom file type errors from MIME validation
- **Response Format**: Standardized error response using ResponseUtils

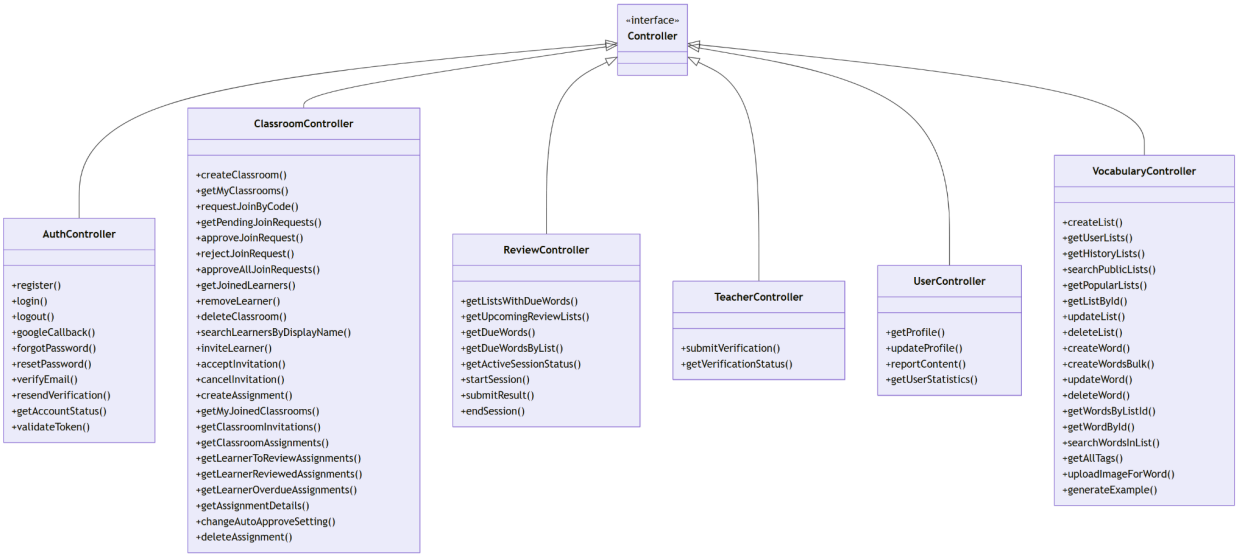
4.4 Component: Controller (Business Logic Layer)

4.4.1 Responsibilities

Controllers handle HTTP requests, coordinate with services, and return formatted responses. They act as thin orchestration layers, delegating business logic to services. Each controller maps to a specific group of primary endpoints:

- **AuthController**: Authentication, registration, OAuth, password management, email verification
- **UserController**: User profiles, settings, content reporting, statistics
- **TeacherController**: Teacher verification workflows and status management
- **VocabularyController**: Vocabulary lists and words management, AI example generation, file uploads
- **ReviewController**: Spaced repetition sessions, progress tracking, due word management
- **ClassroomController**: Classroom lifecycle, member management, assignments, invitations
- **AdminController**: Covers system administration features, content moderation, analytics, and broadcasting messages.

4.4.2 Class diagram



4.4.3 Key Controller Classes

- **AuthController**: Manages user authentication flows including traditional login/registration, Google OAuth integration, password reset workflows, and email verification processes.
- **UserController**: Handles user profile management, personal settings updates, learning statistics retrieval, and content reporting functionality.
- **TeacherController**: Manages the teacher verification process, allowing users to submit credentials and track approval status for role elevation.
- **VocabularyController**: Coordinates vocabulary management including CRUD operations on

lists and words, AI-powered example generation, image uploads, and search functionality.

- **ReviewController**: Orchestrates the spaced repetition learning system, managing review sessions, progress updates, and due word calculations.
- **ClassroomController**: Handles comprehensive classroom management including creation, member enrollment, assignment distribution, and invitation workflows.
- **AdminController**: Manages system administration features, including user management, teacher request approvals, content violation report handling, and system-wide data analytics.

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
3. googleCallback(req, res, next)
    - **Description:** Handles Google OAuth callback and session creation.
    - **Execution Flow:**
        1. Process OAuth result via Passport
        2. Create or update user profile
        3. Generate JWT and redirect to frontend with token
4. forgotPassword(req, res)
    - **Description:** Initiates password reset workflow.
    - **Input Parameters:**

1. req: Contains email
2. res: Used to send the response

- **Return Value:**
- **Execution Flow:**
  1. Generate secure reset token
  2. Send password reset email
  3. Return success response (no user disclosure)

5. resetPassword(req, res)
   - **Description:** Completes password reset with new credentials.
   - **Input Parameters:**
     1. req: Contains reset token and new password
     2. res: Used to send the response
   - **Return Value:** Confirmation message indicating password reset success
   - **Execution Flow:**
     1. Validate reset token and expiration
     2. Update password hash
     3. Invalidate reset token

6. verifyEmail(req, res)
   - **Description:** Confirms email ownership and activates account.
   - **Input Parameters:**
     1. req: Contains email
     2. res: Used to send the response
   - **Return Value:** Confirmation message indicating account activation status
   - **Execution Flow:**
     1. Generate secure reset token
     2. Send password reset email
     3. Return success response (no user disclosure)

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, daily goal)
  2. res: Used to send the response
- **Return Value:** The updated user profile
- **Execution Flow:**
  1. Process avatar upload if provided
  2. Update individual profile fields
  3. Return updated profile data

3. reportContent(req, res)
   - **Description**: Submits content violation reports for moderation.
   - **Input Parameters:**
     1. req: Contains wordId and reason
     2. res: Used to send the response
   - **Return Value:** Confirmation message with the report ID and status
   - **Execution Flow:**
     1. Validate word existence
     2. Check for duplicate reports
     3. Create report record and return confirmation

4. getUserStatistics(req, res)
   - **Description**: Retrieves learning analytics and progress data.
   - **Input Parameters:**
     1. req: Contains the data to update (e.g., full name, avatar, daily goal)
     2. res: Used to send the response
   - **Return Value:** Comprehensive learning statistics including:
     1. Summary metrics (words learned, streaks, retention rate)
     2. Progress over time charts
     3. Completion rates by list
     4. Study consistency calendar

c) Class: TeacherController

1. submitVerification(req, res)
   - **Description**: Submits or updates teacher verification request.
   - **Input Parameters:**
     1. req: Contains teacher's credentials: fullName, institution, schoolEmail, additionalNotes and credentials file
     2. res: Used to send the response
   - **Return Value:** Status indicating if a new request was created or an existing request was updated
   - **Execution Flow:**
     1. Upload credentials file to secure storage

2. Create or update verification request
3. Update user profile and role
4. Return request status

2. getVerificationStatus(req, res)
    - **Description**: Retrieves current verification request status.
    - **Input Parameters:**
        1. req: Contains user information
        2. res: Used to send the response
    - **Return Value:** Verification status with submission details and feedback
    - **Execution Flow:**
        1. Query verification request status
        2. Compile relevant details and feedback
        3. Return verification status

c) Class: VocabularyController

1. createList(req, res)
    - **Description:** Creates a new vocabulary list.
    - **Input Parameters:**
        1. req: Contains the list details (title, description, etc.) in the body and the creator's ID from the authenticated user.
        2. res: Used to send the response.
    - **Return Value:** The newly created list object.
    - **Execution Flow:**
        1. Extracts list data and creator ID from the request.
        2. Calls the vocabulary service to create the list in the database.
        3. Returns the new list object with a 201 status code.
        4. Handles validation and server errors.

2. getUserLists(req, res)
    - **Description:** Retrieves all vocabulary lists created by the authenticated user.
    - **Input Parameters:**
        1. req: Contains the user's ID and optional query parameters for filtering, sorting, and pagination.
        2. res: Used to send the response.
    - **Return Value:** A paginated list of the user's vocabulary lists.
    - **Execution Flow:**
        1. Extracts user ID and query parameters from the request.
        2. Calls the vocabulary service to find the user's lists.
        3. Returns the lists and pagination information.
        4. Handles server errors.

3. getHistoryLists(req, res)
    - **Description:** Retrieves the user's recently viewed vocabulary lists.
    - **Input Parameters:**
        1. req: Contains the user's ID and optional query parameters for pagination.
        2. res: Used to send the response.

- **Return Value:** A paginated list of recently viewed lists.
- **Execution Flow:**
    1. Extracts user ID and pagination parameters from the request.
    2. Calls the vocabulary service to find the user's list history.
    3. Returns the lists and pagination information.
    4. Handles server errors.

4. searchPublicLists(req, res)
    - **Description:** Searches all public vocabulary lists.
    - **Input Parameters:**
        1. req: Contains optional query parameters for searching, filtering by tags, sorting, and pagination.
        2. res: Used to send the response.
    - **Return Value:** A paginated list of public vocabulary lists that match the search criteria.
    - **Execution Flow:**
        1. Extracts query parameters from the request.
        2. Calls the vocabulary service to search public lists.
        3. Returns the matching lists and pagination information.
        4. Handles server errors.

5. getPopularLists(req, res)
    - **Description:** Retrieves a list of the most popular public vocabulary lists.
    - **Input Parameters:**
        1. req: Contains optional query parameters for pagination.
        2. res: Used to send the response.
    - **Return Value:** A paginated list of popular vocabulary lists.
    - **Execution Flow:**
        1. Extracts pagination parameters from the request.
        2. Calls the vocabulary service to find popular lists.
        3. Returns the lists and pagination information.
        4. Handles server errors.

6. getListById(req, res)
    - **Description:** Retrieves a single, specific vocabulary list by its ID.
    - **Input Parameters:**
        1. req: Contains the listId in the URL parameters and the authenticated user's ID.
        2. res: Used to send the response.
    - **Return Value:** The detailed vocabulary list object.
    - **Execution Flow:**
        1. Extracts listId and user ID from the request.
        2. Calls the vocabulary service to find the list and check permissions.
        3. Returns the list object if found and accessible.
        4. Handles "not found" and "forbidden" errors.

7. updateList(req, res)
    - **Description**: Updates the details of an existing vocabulary list.
    - **Input Parameters:**
        1. req: Contains the listId in the URL parameters, the user's ID, and the update data

in the body.
2. res: Used to send the response.
- **Return Value:** The updated vocabulary list object.
- **Execution Flow:**
1. Extracts parameters and data from the request.
2. Calls the vocabulary service to update the list, which includes a permission check.
3. Returns the updated list object.
4. Handles "forbidden" and validation errors.

8. deleteList(req, res)
- **Description:** Deletes a vocabulary list.
- **Input Parameters:**
1. req: Contains the listId in the URL parameters and the user's ID.
2. res: Used to send the response.
- **Return Value:** A success message.
- **Execution Flow:**
1. Extracts listId and user ID from the request.
2. Calls the vocabulary service to delete the list, which includes a permission check.
3. Returns a success response.
4. Handles "forbidden" errors.

8. createWord(req, res)
- **Description:** Adds a new word to a specific vocabulary list.
- **Input Parameters:**
1. req: Contains the listId in the URL parameters, the user's ID, and the new word's data in the body.
2. res: Used to send the response.
- **Return Value:** The newly created word object.
- **Execution Flow:**
1. Extracts parameters and data from the request.
2. Calls the vocabulary service to create the word in the database.
3. Returns the new word object with a 201 status code.
4. Handles "forbidden" errors.

9. createWordsBulk(req, res)
- **Description**: Adds multiple words to a specific vocabulary list in a single operation.
- **Input Parameters:**
1. req: Contains the listId in the URL parameters, the user's ID, and an array of word objects in the body.
2. res: Used to send the response.
- **Return Value**: A summary of the bulk operation (created count, failed count, errors).
- **Execution Flow:**
1. Extracts parameters and data from the request.
2. Calls the vocabulary service to perform the bulk insert.
3. Returns the operation summary with a 201 status code.
4. Handles "forbidden" errors.

10. updateWord(req, res)
    - **Description:** Updates the details of an existing word.
    - **Input Parameters:**
        1. req: Contains the wordId in the URL parameters, the user's ID, and the update data in the body.
        2. res: Used to send the response.
    - **Return Value:** The updated word object.
    - **Execution Flow:**
        1. Extracts parameters and data from the request.
        2. Calls the vocabulary service to update the word, which includes a permission check.
        3. Returns the updated word object.
        4. Handles "forbidden" errors.

11. deleteWord(req, res)
    - **Description:** Deletes a word from a vocabulary list.
    - **Input Parameters:**
        1. req: Contains the wordId in the URL parameters and the user's ID.
        2. res: Used to send the response.
    - **Return Value:** A success message.
    - **Execution Flow:**
        1. Extracts wordId and user ID from the request.
        2. Calls the vocabulary service to delete the word, which includes a permission check.
        3. Returns a success response.
        4. Handles "forbidden" errors.

12. getWordsByListId(req, res)
    - **Description:** Retrieves all words belonging to a specific vocabulary list.
    - **Input Parameters:**
        1. req: Contains the listId in the URL parameters, the user's ID, and optional query parameters for pagination.
        2. res: Used to send the response.
    - **Return Value:** A paginated list of words from the specified list.
    - **Execution Flow:**
        1. Extracts parameters from the request.
        2. Calls the vocabulary service to find the words, which includes a permission check.
        3. Returns the words and pagination information.
        4. Handles "not found" and "forbidden" errors.

13. getWordById(req, res)
    - **Description:** Retrieves a single, specific word by its ID.
    - **Input Parameters:**
        1. req: Contains the wordId in the URL parameters and the user's ID.
        2. res: Used to send the response.
    - **Return Value:** The detailed word object, including user progress.
    - **Execution Flow:**

1. Extracts parameters from the request.
2. Calls the vocabulary service to find the word and its associated user progress.
3. Returns the enriched word object.
4. Handles "not found" and "forbidden" errors.

14. searchWordsInList(req, res)
    - **Description:** Searches for words within a specific vocabulary list.
    - **Input Parameters:**
        1. req: Contains the listId in the URL, the user's ID, and query parameters for search term (q), sorting, and pagination.
        2. res: Used to send the response.
    - **Return Value:** A paginated list of words that match the search criteria.
    - **Execution Flow:**
        1. Extracts parameters from the request and validates the presence of the search query q.
        2. Calls the vocabulary service to perform the search.
        3. Returns the matching words and pagination information.
        4. Handles "forbidden" errors.

15. getAllTags(req, res)
    - **Description:** Retrieves a list of all available tags in the system.
    - **Input Parameters:**
        1. req: Standard request object.
        2. res: Used to send the response.
    - **Return Value:** An array of tag names.
    - **Execution Flow:**
        1. Calls the vocabulary service to fetch all tags from the database.
        2. Returns the array of tags.
        3. Handles server errors.

16. generateExample(req, res)
    - **Description:** Generates an example sentence for an existing word using an AI service.
    - **Input Parameters:**
        1. req: Contains the wordId in the URL, the user's ID, and optional context in the body.
        2. res: Used to send the response.
    - **Return Value:** The generated example object.
    - **Execution Flow:**
        1. Extracts parameters from the request.
        2. Calls the vocabulary service to generate the example, which includes permission checks.
        3. Returns the new example.
        4. Handles "forbidden" and server errors.

17. generateExampleForNewWord(req, res)
    - **Description:** Generates an example sentence for a new, unsaved word.
    - **Input Parameters:**
        1. req: Contains the word's term and definition in the body.

2. res: Used to send the response.
- **Return Value**: The generated example object.
- **Execution Flow:**
  1. Extracts term and definition from the request body.
  2. Calls the vocabulary service to generate the example.
  3. Returns the new example.
  4. Handles server errors.

d) Class: ReviewController

1. getListsWithDueWords(req, res)
   - **Description:** Retrieves a paginated list of vocabulary lists that contain words due for review today.
   - **Input Parameters:**
     1. req: Contains the user's ID and optional query parameters for pagination (page, limit).
     2. res: Used to send the response.
   - **Return Value**: A paginated object containing lists with their due word counts.
   - **Execution Flow:**
     1. Extracts user ID and pagination parameters from the request.
     2. Calls the review service to fetch the lists with due words.
     3. Returns the paginated list data.

2. getUpcomingReviewLists(req, res)
   - **Description:** Retrieves a paginated list of vocabulary lists that have words scheduled for review in the next 7 days.
   - **Input Parameters:**
     1. req: Contains the user's ID and optional query parameters for pagination.
     2. res: Used to send the response.
   - **Return Value:** A paginated object containing upcoming lists and their next review day.
   - **Execution Flow:**
     1. Extracts user ID and pagination parameters from the request.
     2. Calls the review service to fetch the upcoming lists.
     3. Returns the paginated list data.

3. getDueWords(req, res)
   - **Description:** Retrieves a summary of all words due for review across all of the user's lists.
   - **Input Parameters:**
     1. req: Contains the user's ID.
     2. res: Used to send the response.
   - **Return Value**: An object grouping due words by list and providing a total count.
   - **Execution Flow:**
     1. Extracts the user ID from the request.
     2. Calls the review service to get the global due words summary.
     3. Returns the summarized data.

4. getDueWordsByList(req, res)
   - **Description:** Retrieves all words that are currently due for review within one specific

vocabulary list.
- **Input Parameters:**
  1. req: Contains the user's ID and the listId from the URL parameters.
  2. res: Used to send the response.
- **Return Value:** An array of detailed word objects that are due for review in that list.
- **Execution Flow:**
  1. Extracts user ID and listId from the request.
  2. Calls the review service to fetch the due words for the specified list.
  3. Returns the array of word objects.

5. getActiveSessionStatus(req, res)
   - **Description:** Checks if the user has an ongoing (active or interrupted) review session.
   - **Input Parameters:**
     1. req: Contains the user's ID.
     2. res: Used to send the response.
   - **Return Value:** The active session object (if one exists) including remaining words, or null.
   - **Execution Flow:**
     1. Extracts the user ID from the request.
     2. Calls the review service to check for an active session.
     3. Returns the session status.

6. startSession(req, res)
   - **Description**: Creates and starts a new review session for a specific list.
   - **Input Parameters:**
     1. req: Contains the user's ID, and the listId and sessionType in the request body.
     2. res: Used to send the response.
   - **Return Value:** A new session object, including the sessionId and a shuffled array of words to be reviewed.
   - **Execution Flow:**
     1. Extracts user ID, listId, and sessionType from the request.
     2. Calls the review service to create the session, which fetches due words and shuffles them.
     3. Returns the new session object with a 201 status code.

7. submitResult(req, res)
   - **Description:** Records the result (correct/incorrect) for a single word within an active review session.
   - **Input Parameters:**
     1. req: Contains the user's ID, the sessionId from the URL, and the wordId and result in the request body.
     2. res: Used to send the response.
   - **Return Value:** A success message.
   - **Execution Flow:**
     1. Extracts all necessary IDs and the result from the request.
     2. Calls the review service to record the result and update the word's spaced repetition progress.
     3. Returns a success response.

8. endSession(req, res)
    - **Description:** Ends an active review session and retrieves the performance summary.
    - **Input Parameters:**
        1. req: Contains the user's ID and the sessionId from the URL.
        2. res: Used to send the response.
    - **Return Value:** A summary object containing stats like total words, accuracy, and completion time.
    - **Execution Flow:**
        1. Extracts user ID and sessionId from the request.
        2. Calls the review service to finalize the session and calculate the summary.
        3. Returns the session summary.

e) Class: ClassroomController

1. createClassroom(req, res)

- **Description:** Creates a new classroom.
- **Input Parameters:**
    - req: Contains classroom details (name, description, classroom_status, capacity_limit) in the body and userId from the authenticated user.
    - res: Used to send the response.
- **Return Value:** Information about the newly created classroom.
- **Execution Flow:**
    - Extract classroom data and the teacher's ID from the request.
    - Call the service to create a classroom with the provided details.
    - Return the newly created classroom object.

2. getMyClassrooms(req, res)

- **Description:** Retrieves all classrooms created by the current teacher.
- **Input Parameters:**
    - req: Contains the teacher's userId from the authenticated user.
    - res: Used to send the response.
- **Return Value:** An array of classroom objects.
- **Execution Flow:**
    - Get the teacher's ID from the request.
    - Call the service to fetch all classrooms associated with the teacher's ID.
    - Return the list of classrooms.

3. requestJoinByCode(req, res)

- **Description:** Submits a request to join a classroom using a join code.
- **Input Parameters:**

- req: Contains the joinCode in the body and the authenticated user's information (user).
- res: Used to send the response.

- **Return Value:** A confirmation message indicating that the join request was submitted or that the user has successfully joined the classroom.
- **Execution Flow:**
  - Extract the joinCode from the request body and the user's information.
  - Check if the joinCode is provided.
  - Call the service to handle the join request by code.
  - Return a success message indicating the request was sent or the user has joined.

4. **getPendingJoinRequests(req, res)**

- **Description:** Retrieves the list of pending join requests for a classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object.
  - res: Used to send the response.
- **Return Value:** An array of pending join request objects.
- **Execution Flow:**
  - Get the classroom ID from the request.
  - Call the service to get all pending join requests for that classroom.
  - Return the list of pending requests.

5. **approveJoinRequest(req, res)**

- **Description:** Approves a learner's request to join the classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object and the learnerId in the body.
  - res: Used to send the response.
- **Return Value:** A confirmation message that the request has been approved.
- **Execution Flow:**
  - Extract the classroom ID and learnerId from the request.
  - Check if learnerId exists in the request body.
  - Call the service to approve the join request.
  - Return a success message.

6. **rejectJoinRequest(req, res)**

- **Description:** Rejects a learner's request to join the classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object and the learnerId in the body.

  - res: Used to send the response.
- **Return Value:** A confirmation message that the request has been rejected.
- **Execution Flow:**
  - Extract the classroom ID and learnerId from the request.
  - Check if learnerId exists in the request body.
  - Call the service to reject the join request.
  - Return a success message.

7. approveAllJoinRequests(req, res)

- **Description:** Approves all pending join requests for a classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object.
  - res: Used to send the response.
- **Return Value:** A message indicating the number of learners that were approved.
- **Execution Flow:**
  - Get the classroom ID from the request.
  - Call the service to approve all pending join requests.
  - Return a success message with the count of approved requests.

8. getJoinedLearners(req, res)

- **Description:** Retrieves a list of all learners who have joined a classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object.
  - res: Used to send the response.
- **Return Value:** An array of learner objects.
- **Execution Flow:**
  - Get the classroom ID from the request.
  - Call the service to get the list of joined learners for the classroom.
  - Return the list of learners.

9. removeLearner(req, res)

- **Description:** Removes a learner from the classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object and the learnerId in the body.
  - res: Used to send the response.
- **Return Value:** A confirmation message that the learner has been removed.
- **Execution Flow:**

- Extract the classroom ID and learnerId from the request.
- Check if learnerId exists in the request body.
- Call the service to remove the learner from the classroom.
- Return a success message.

10. deleteClassroom(req, res)

- **Description:** Deletes a classroom.
- **Input Parameters:**
  - req: Contains the classroom object to be deleted.
  - res: Used to send the response.
- **Return Value:** A confirmation message that the classroom has been deleted.
- **Execution Flow:**
  - Get the classroom object from the request.
  - Call the service to delete the classroom.
  - Return a success message.

11. searchLearnersByDisplayName(req, res)

- **Description:** Searches for learners in a classroom by display name and status.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object, and query parameters status and q (search query).
  - res: Used to send the response.
- **Return Value:** An array of learner objects matching the search criteria.
- **Execution Flow:**
  - Extract the classroom ID, status, and q from the request.
  - Call the service to search for learners by display name and status.
  - Return the search results.

12. inviteLearner(req, res)

- **Description:** Sends an invitation to a learner to join the classroom via email.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object and the learner's email in the body.
  - res: Used to send the response.
- **Return Value:** A confirmation message that the invitation has been sent.
- **Execution Flow:**
  - Extract the classroom ID and email from the request.

- Check if the email is provided.
- Call the service to create and send an invitation to the learner.
- Return a success message.

13. acceptInvitation(req, res)

- **Description:** Accepts an invitation to join a classroom.
- **Input Parameters:**
  - req: Contains the invitation token in the body and the authenticated user's information (user).
  - res: Used to send the response.
- **Return Value:** Information about successfully joining the classroom.
- **Execution Flow:**
  - Extract the token and user information from the request.
  - Check if the token is provided.
  - Call the service to process the invitation acceptance.
  - Return a success message.

14. cancelInvitation(req, res)

- **Description:** Cancels a sent invitation to join a classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object and the email in the body.
  - res: Used to send the response.
- **Return Value:** A confirmation message that the invitation has been canceled.
- **Execution Flow:**
  - Extract the classroom ID and email from the request.
  - Check if the email is provided.
  - Call the service to cancel the invitation.
  - Return a success message

15. _extractAssignmentData(req)

- **Description:** An internal helper function to extract assignment data from the request.
- **Input Parameters:**
  - req: The request object containing assignment information.
- **Return Value:** An object containing the extracted assignment data.
- **Execution Flow:**
  - Extract classroomId, teacherId, vocabListId, title, exerciseMethod, wordsPerReview, startDate, and dueDate from the request.

- Return an object with this data.

16. createAssignment(req, res)

- **Description:** Creates a new assignment for the classroom.
- **Input Parameters:**
  - req: Contains the classroom id, teacher's userId, and assignment details in the body (vocabListId, title, exerciseMethod, wordsPerReview, startDate, dueDate).
  - res: Used to send the response.
- **Return Value:** The newly created assignment object.
- **Execution Flow:**
  - Call _extractAssignmentData to extract assignment data from the request.
  - Call the service to create a new assignment with the extracted data.
  - Return the newly created assignment object.

17. getMyJoinedClassrooms(req, res)

- **Description:** Retrieves all classrooms that the current learner has joined.
- **Input Parameters:**
  - req: Contains the learner's userId from the authenticated user.
  - res: Used to send the response.
- **Return Value:** An array of classroom objects.
- **Execution Flow:**
  - Get the learnerId from the request.
  - Call the service to get all classrooms the learner has joined.
  - Return the list of classrooms.

18. getClassroomInvitations(req, res)

- **Description:** Retrieves the list of pending invitations for a classroom.
- **Input Parameters:**
  - req: Contains the classroom id from the classroom object.
  - res: Used to send the response.
- **Return Value:** An array of invitation objects.
- **Execution Flow:**
  - Get the classroom ID from the request.
  - Call the service to get the list of invitations for the classroom.
  - Return the list of invitations.

19. getClassroomAssignments(req, res)

- **Description:** Retrieves all assignments for a classroom.

- **Input Parameters:**
    - req: Contains the classroom id from the classroom object.
    - res: Used to send the response.
- **Return Value:** An array of assignment objects.
- **Execution Flow:**
    - Get the classroom ID from the request.
    - Call the service to get all assignments for that classroom.
    - Return the list of assignments.

20. getLearnerToReviewAssignments(req, res)

- **Description:** Retrieves the assignments a learner needs to complete in a classroom.
- **Input Parameters:**
    - req: Contains the classroom id and the learner's userId.
    - res: Used to send the response.
- **Return Value:** An array of to-review assignments.
- **Execution Flow:**
    - Extract the classroom ID and learner ID from the request.
    - Call the service to get the list of assignments to be reviewed by the learner.
    - Return the result.

21. getLearnerReviewedAssignments(req, res)

- **Description:** Retrieves the completed assignments of a learner in a classroom.
- **Input Parameters:**
    - req: Contains the classroom id and the learner's userId.
    - res: Used to send the response.
- **Return Value:** An array of reviewed assignments.
- **Execution Flow:**
    - Extract the classroom ID and learner ID from the request.
    - Call the service to get the list of reviewed assignments by the learner.
    - Return the result.

22. getLearnerOverdueAssignments(req, res)

- **Description:** Retrieves the overdue assignments of a learner in a classroom.
- **Input Parameters:**
    - req: Contains the classroom id and the learner's userId.
    - res: Used to send the response.
- **Return Value:** An array of overdue assignments.

- **Execution Flow:**
    - Extract the classroom ID and learner ID from the request.
    - Call the service to get the list of overdue assignments for the learner.
    - Return the result.

23. getAssignmentDetails(req, res)

- **Description:** Retrieves the details of a specific assignment.
- **Input Parameters:**
    - req: Contains the classroom id and the assignmentId from the URL parameters.
    - res: Used to send the response.
- **Return Value:** The detailed assignment object.
- **Execution Flow:**
    - Extract the classroom ID and assignment ID from the request.
    - Call the service to get the details of the assignment.
    - Return the result.

24. changeAutoApproveSetting(req, res)

- **Description:** Changes the auto-approval setting for join requests for a classroom.
- **Input Parameters:**
    - req: Contains the classroom id and the isAutoApprovalEnabled (boolean) value in the body.
    - res: Used to send the response.
- **Return Value:** The new state of the auto-approval setting.
- **Execution Flow:**
    - Extract the classroom ID and isAutoApprovalEnabled from the request.
    - Validate that isAutoApprovalEnabled is a boolean.
    - Call the service to change the auto-approval setting.
    - Return a success message with the new status.

25. deleteAssignment(req, res)

- **Description:** Deletes an assignment from a classroom.
- **Input Parameters:**
    - req: Contains the classroom id, the assignmentId from URL parameters, and the user's userId.
    - res: Used to send the response.
- **Return Value:** A confirmation message that the assignment has been deleted.
- **Execution Flow:**

- Extract the classroom ID, assignment ID, and user ID from the request.
- Call the service to delete the assignment.
- Return a success message.

26. leaveClassroom(req, res)

- **Description:** Allows a learner to leave a classroom.
- **Input Parameters:**
  - req: Contains the classroom id and the learner's userId.
  - res: Used to send the response.
- **Return Value:** A confirmation message that the learner has left the classroom.
- **Execution Flow:**
  - Extract the classroom ID and learner ID from the request.
  - Call the service to handle the learner leaving the classroom.
  - Return a success message.

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

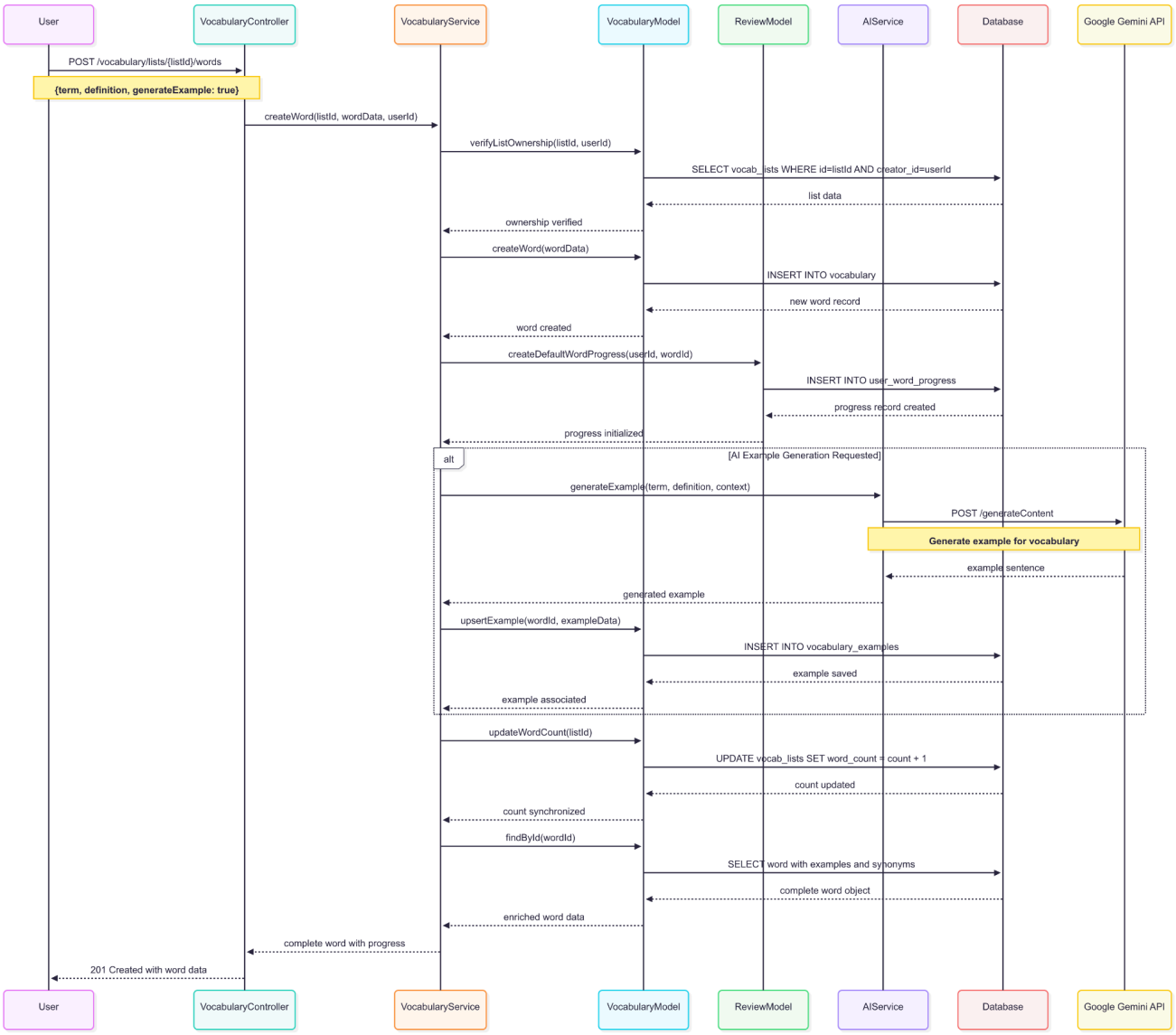
4.5 Component: Service (Business Logic Layer)

4.5.1 Responsibilities:

Services contain core business logic, separated from controllers and data access layers. They orchestrate complex workflows involving multiple models and external services.

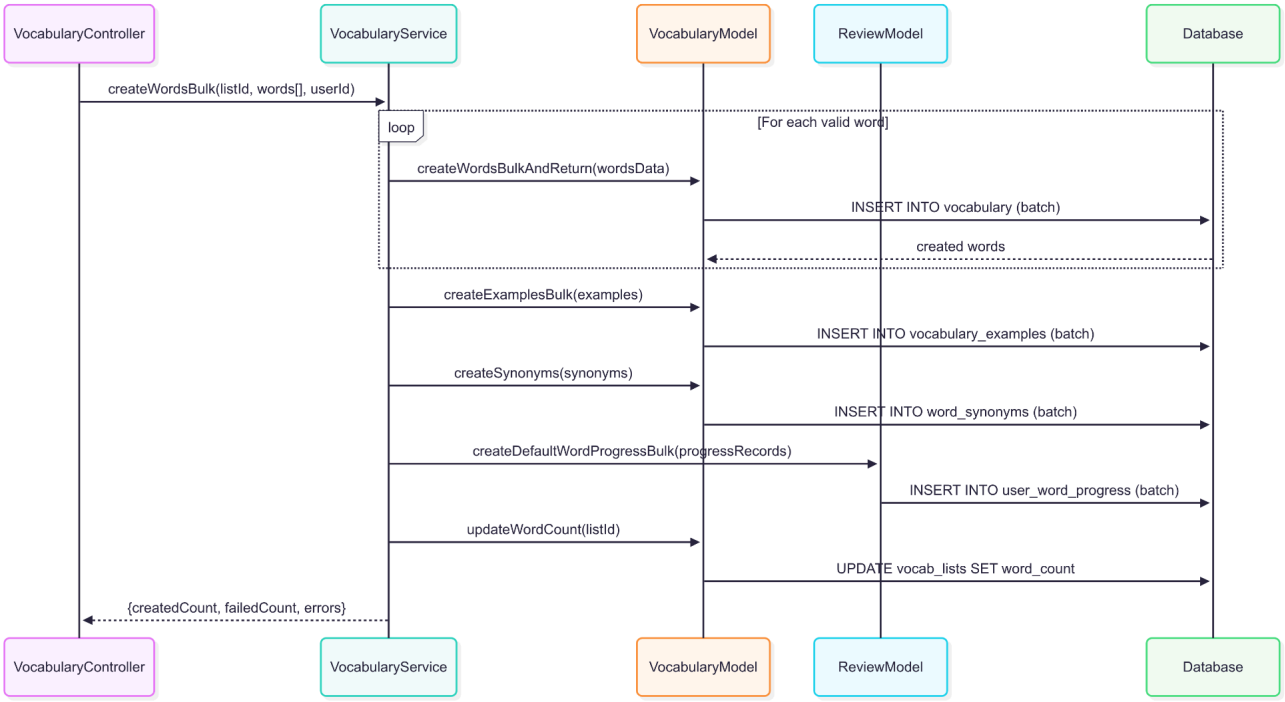
4.5.2 Main Service Flow - Vocabulary Creation, generate example with AI

The following sequence diagram illustrates the primary workflow when a user creates vocabulary with AI-generated examples (Click this link to view the diagram clearly)

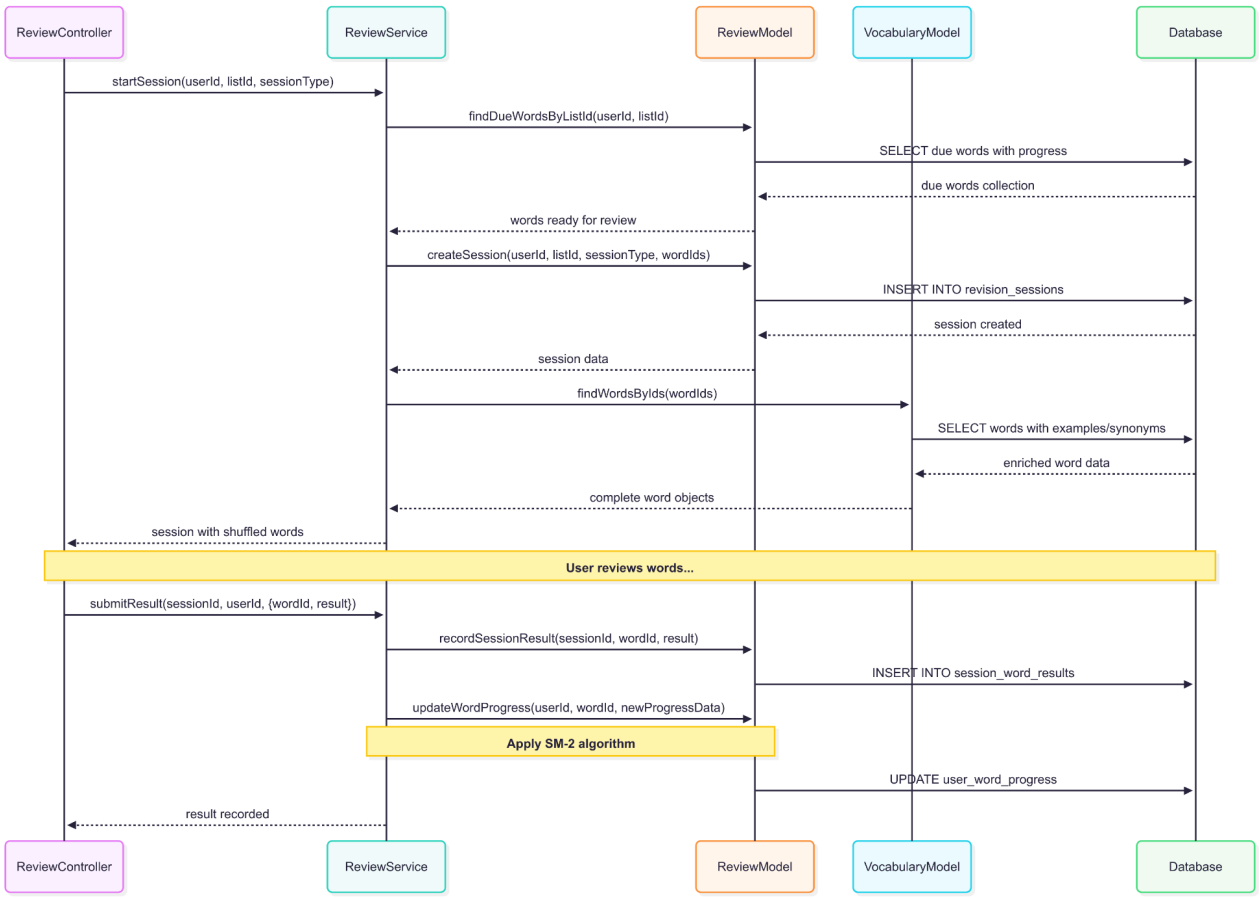

User
VocabularyController
VocabularyService
VocabularyModel
ReviewModel
AIService
Database
Google Gemini API
POST /vocabulary/lists/{listId}/words
{term, definition, generateExample: true}
createWord(listId, wordData, userId)
verifyListOwnership(listId, userId)
SELECT vocab_lists WHERE id=listId AND creator_id=userId
list data
ownership verified
createWord(wordData)
INSERT INTO vocabulary
new word record
word created
createDefaultWordProgress(userId, wordId)
INSERT INTO user_word_progress
progress record created
progress initialized
alt
[AI Example Generation Requested]
generateExample(term, definition, context)
POST /generateContent
Generate example for vocabulary
example sentence
generated example
upsertExample(wordId, exampleData)
INSERT INTO vocabulary_examples
example saved
example associated
updateWordCount(listId)
UPDATE vocab_lists SET word_count = count + 1
count updated
count synchronized
findById(wordId)
SELECT word with examples and synonyms
complete word object
enriched word data
complete word with progress
201 Created with word data

4.5.3 Alternative Flows

Bulk Vocabulary Creation (Click this link to view the diagram clearly):

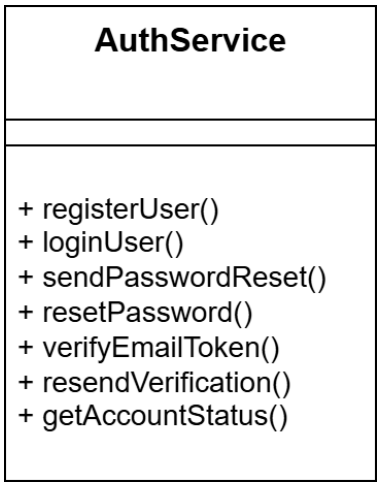


Review Session Flow (Click this link to view the diagram clearly):



4.5.3 Key Service Classes:

a. AuthService

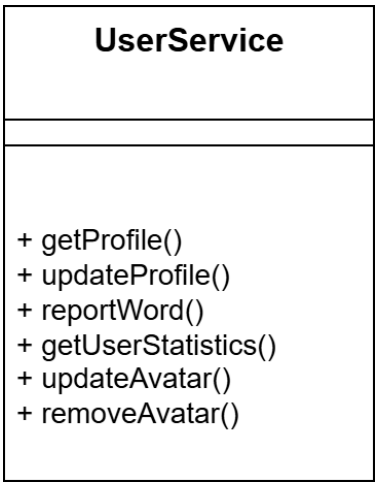


Description: Manages all authentication-related business logic including user registration, login, password management, and email verification workflows. Acts as the core authentication layer interfacing with user models and external services.

Methods:

- **registerUser(email, password, role):** Creates new user accounts with email verification. Validates email uniqueness, hashes passwords, generates verification tokens, and sends confirmation emails. Returns user data with JWT token.
- **loginUser(email, password):** Authenticates user credentials and establishes sessions. Validates password hashes, checks account status (active/suspended), and generates JWT tokens for authenticated sessions.
- **sendPasswordReset(email):** Initiates password reset workflow by generating secure reset tokens and sending reset emails. Implements security best practices by not revealing account existence.
- **resetPassword(token, newPassword):** Completes password reset process by validating reset tokens, updating password hashes, and invalidating used tokens to prevent replay attacks.
- **verifyEmailToken(token):** Processes email verification tokens to activate user accounts. Validates token authenticity and expiration, updates email verification status, and returns authentication tokens.
- **resendVerification(email):** Regenerates and sends new verification emails for unverified accounts. Includes validation to prevent spam and ensure account eligibility.
- **getAccountStatus(email):** Retrieves account verification and status information without exposing sensitive data, supporting frontend conditional rendering.

b. UserService

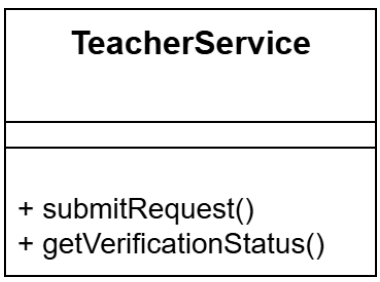


Description: Handles user profile management, settings, and user-generated content operations. Coordinates with storage services for file uploads and integrates with statistics models for analytics.

Methods:

- **getProfile(userId):** Retrieves comprehensive user profile data including settings, role-specific information (teacher verification status, classroom counts), and ensures default settings exist.
- **updateProfile(userId, updateData, avatarFile):** Manages profile updates including display name, daily goals, and avatar management. Coordinates file uploads and maintains data consistency.
- **reportWord(reporterId, wordId, reason):** Processes content violation reports with duplicate prevention and word existence validation. Creates moderation records for admin review.
- **getUserStatistics(userId):** Aggregates learning analytics including summary metrics, progress charts, completion rates, and study consistency data from multiple sources.
- **updateAvatar(userId, avatarFile):** Handles avatar image uploads through storage service integration with proper file validation and URL generation.
- **removeAvatar(userId):** Safely removes user avatar images and updates profile references to maintain data consistency.

c. TeacherService



Description: Manages teacher verification workflows enabling role elevation from learner to teacher.

Handles credential uploads and status tracking for administrative approval processes.

Methods:

- **submitRequest(userId, data, file):** Processes teacher verification submissions including credential file uploads, creates or updates verification requests, and manages user role transitions.
- **getVerificationStatus(userId):** Retrieves current verification request status with submission details, approval status, and rejection reasons for user feedback.

d. VocabularyService

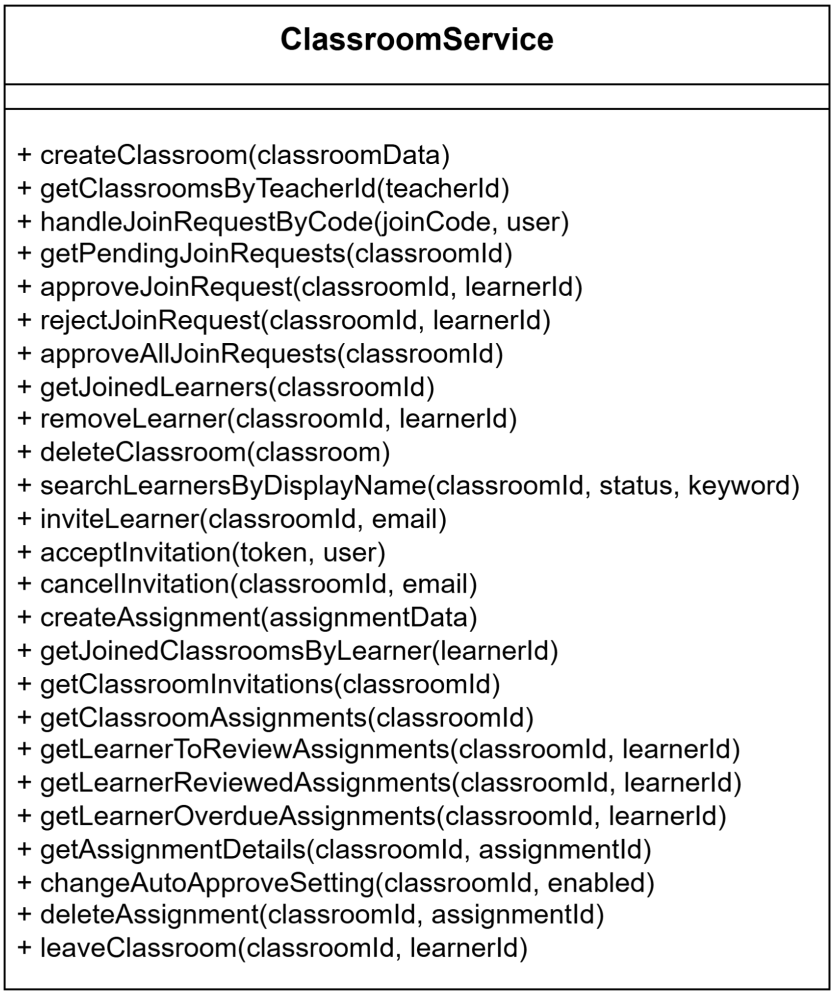
Description: Manages all business logic related to vocabulary lists and words. This includes creation, retrieval, updates, deletion, searching, and advanced features like AI-powered example generation and tracking user interaction history.

Methods:

- **createList(listData, creatorId):** Creates a new vocabulary list and associates it with existing tags using a database function for atomicity.
- **findUserLists(userId, options):** Retrieves a paginated list of all vocabulary lists created by a specific user, with support for searching, sorting, and filtering.
- **searchPublicLists(options):** Retrieves a paginated list of all public vocabulary lists, with support for searching by keyword and filtering by tags.

- **findListById(listId, userId):** Retrieves a single, detailed vocabulary list by its ID. It enforces privacy rules (a user can only see their own private lists) and updates the user's viewing history.
- **findHistoryLists(userId, options):** Retrieves a paginated list of vocabulary lists that the user has recently viewed, ordered by the most recent access.
- **findPopularLists(options):** Retrieves a paginated list of public vocabulary lists, ordered by their popularity (view count).
- **updateList(listId, userId, updateData):** Updates the properties (title, description, tags, etc.) of an existing vocabulary list after verifying user ownership.
- **deleteList(listId, userId):** Deletes a vocabulary list and all its associated words after verifying user ownership.
- **createWord(listId, wordData, userId):** Creates a single new word, including its optional example and synonyms, and adds it to a specified list. It also initializes the user's spaced repetition progress for the new word.
- **createWordsBulk(listId, words, userId):** Adds multiple words to a list in a single, efficient operation. It processes valid words and returns a summary of any that failed validation.
- **updateWord(wordId, userId, updateData):** Updates the properties of a single existing word, including its example and synonyms, after verifying user ownership.
- **deleteWord(wordId, userId):** Deletes a single word from a list after verifying user ownership and updates the list's word count.
- **findWordsByListId(listId, userId, options):** Retrieves a paginated list of all words within a specific vocabulary list, enriching each word with its example and synonyms.
- **findWordById(wordId, userId):** Retrieves the full details of a single word, including its example, synonyms, and the authenticated user's specific learning progress for that word.
- **searchWordsInList(listId, userId, options):** Performs a search for words within a specific list based on a query string, with support for sorting and pagination.
- **generateExample(wordId, userId, context):** Uses an AI service to generate a new example sentence for an existing word and saves it to the database.
- **generateExampleForNewWord(term, definition, context):** Uses an AI service to generate an example sentence for a word that has not yet been saved to the database.
- **findAllTags():** Retrieves a list of all unique tags available in the system.

e. ClassroomService



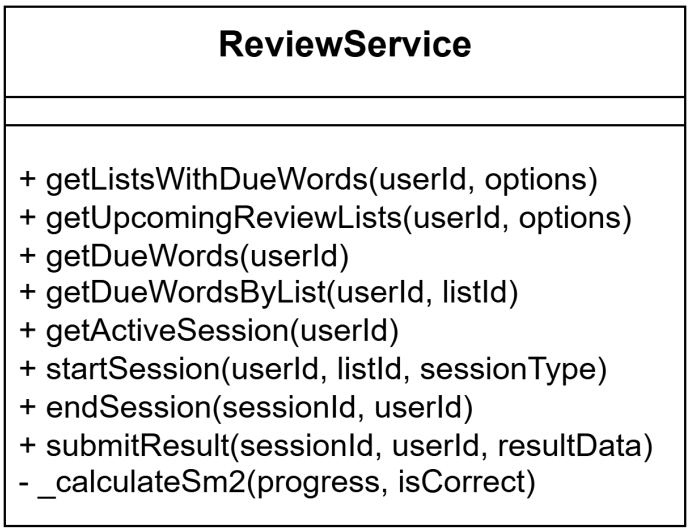
Description: Manages all business logic related to classrooms, including creation, membership management (joining, approving, rejecting), invitations, assignments, and data retrieval.

Methods:

- **createClassroom(classroomData)**: Creates a new classroom with a unique join code.
- **getClassroomsByTeacherId(teacherId)**: Retrieves all classrooms for a given teacher, including assignment counts.
- **handleJoinRequestByCode(joinCode, user)**: Processes a learner's request to join a classroom using a join code, handling validation and auto-approval.
- **getPendingJoinRequests(classroomId)**: Retrieves all pending join requests for a classroom.
- **approveJoinRequest(classroomId, learnerId):** Approves a pending join request for a specific learner, changing their status to 'joined'.
- **rejectJoinRequest(classroomId, learnerId)**: Rejects a pending join request for a specific learner.
- **approveAllJoinRequests(classroomId)**: Approves all pending join requests for a classroom, up to its capacity limit.

- **getJoinedLearners(classroomId)**: Retrieves a list of all learners who are currently members of a classroom.
- **removeLearner(classroomId, learnerId):** Removes a learner from a classroom by updating their status.
- **deleteClassroom(classroom)**: Soft-deletes a classroom by updating its status.
- **searchLearnersByDisplayName(classroomId, status, keyword)**: Searches for learners within a classroom by their display name and join status.
- **inviteLearner(classroomId, email)**: Creates and sends an email invitation for a user to join a classroom.
- **acceptInvitation(token, user)**: Validates an invitation token and adds the user to the classroom as a learner.
- **cancelInvitation(classroomId, email)**: Cancels a pending invitation that has not yet been used.
- **createAssignment(assignmentData)**: Creates a new assignment, clones the vocabulary list into sublists, and links them.
- **getJoinedClassroomsByLearner(learnerId)**: Retrieves all classrooms a specific learner has joined, including assignment counts.
- **getClassroomInvitations(classroomId)**: Retrieves all pending and used invitations for a classroom.
- **getClassroomAssignments(classroomId**): Retrieves all assignments for a classroom and computes their current status.
- **getLearnerToReviewAssignments(classroomId, learnerId)**: Retrieves all active assignments a learner needs to complete.
- **getLearnerReviewedAssignments(classroomId, learnerId)**: Retrieves all assignments a learner has completed.
- **getLearnerOverdueAssignments(classroomId, learnerId)**: Retrieves all assignments for a learner that are past their due date.
- **getAssignmentDetails(classroomId, assignmentId)**: Retrieves detailed information for a single assignment, including its full vocabulary list.
- **changeAutoApproveSetting(classroomId, enabled)**: Updates the auto-approval setting for a classroom.
- **deleteAssignment(classroomId, assignmentId)**: Permanently deletes an assignment and all its associated data, including sublists.
- **leaveClassroom(classroomId, learnerId)**: Allows a learner to leave a classroom and deletes their associated assignment progress.

f. ReviewService

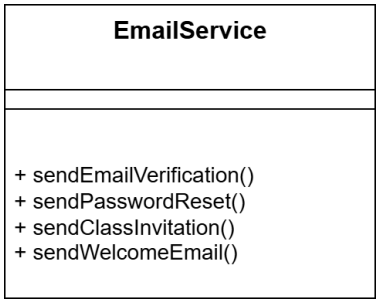


Description: Manages all business logic related to the user's learning and review process. This includes fetching vocabulary items that are due for review based on the Spaced Repetition System (SRS), managing the lifecycle of a review session, processing user answers, and calculating the next review dates for words.

Methods:

- **getListsWithDueWords(userId, options):** Retrieves a paginated list of vocabulary lists that contain words currently due for review, enriched with creator information.
- **getUpcomingReviewLists(userId, options):** Retrieves a paginated list of vocabulary lists that will have words due for review within the next 7 days, excluding today.
- **getDueWords(userId):** Fetches a summary of all words due for review across all of the user's lists, grouped by list.
- **getDueWordsByList(userId, listId):** Retrieves all words currently due for review for a single, specific list after verifying user access permissions. Each word is enriched with its details and user progress.
- **getActiveSession(userId):** Checks if the user has an ongoing review session and, if so, returns its state, including the remaining words to be reviewed.
- **startSession(userId, listId, sessionType):** Initiates a new review session. It fetches a batch of due words from the specified list, creates a session record in the database, and returns the shuffled words to the client.
- **endSession(sessionId, userId):** Finalizes a review session, calculates the final accuracy and other stats, and returns a performance summary.
- **submitResult(sessionId, userId, resultData):** Records a user's answer (correct/incorrect) for a word within a session. This is the core method that triggers the SRS calculation.
- **_calculateSm2(progress, isCorrect):** A private helper method that implements the SuperMemo 2 (SM-2) algorithm to calculate the next review date, interval, and ease factor for a word based on the user's performance.

g. EmailService

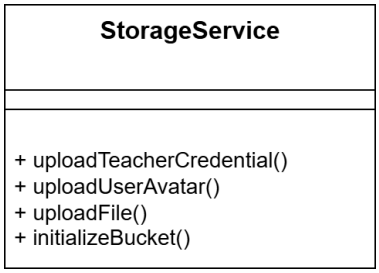


Description: Comprehensive email notification system using Handlebars templates and SMTP integration. Manages transactional emails for authentication, classroom operations, and user communications.

Methods:

- **sendEmailVerification(to, verificationToken):** Sends account verification emails with secure tokens and frontend-compatible verification links using branded templates.
- **sendPasswordReset(to, resetToken):** Delivers password reset emails with time-limited secure tokens and clear instructions for account recovery.
- **sendClassInvitation(to, token, classInfo, teacherName):** Sends classroom invitation emails with join tokens, classroom details, and teacher information for seamless enrollment.
- **sendWelcomeEmail(to, displayName):** Sends welcome emails to new users with onboarding information and dashboard links to improve user engagement.

h. StorageService



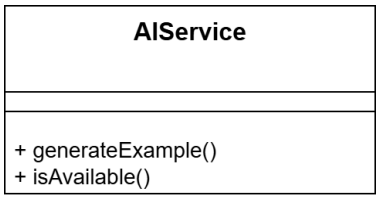
Description: Manages file upload operations with Supabase storage integration. Handles different file types with appropriate validation, security measures, and URL generation for various use cases.

Methods:

- **uploadTeacherCredential(file, userId):** Handles teacher verification document uploads with secure folder organization, file type validation, and appropriate access controls.
- **uploadUserAvatar(file, userId):** Manages user profile image uploads with automatic file naming, overwrite capabilities, and public URL generation for frontend display.
- **uploadFile(file, options):** Generic file upload method supporting various configurations including bucket selection, folder organization, and security settings.

- **initializeBucket(bucketConfig):** Sets up Supabase storage buckets with proper configurations including public access settings, file type restrictions, and size limits.

i. AIService



Description: Integrates with Google Gemini AI for vocabulary learning enhancement. Provides AI-generated content including example sentences and contextual learning materials with error handling and fallback mechanisms.

Methods:

- **generateExample(word, definition, context):** Creates contextually appropriate example sentences for vocabulary words using AI prompts optimized for language learning with retry mechanisms for reliability.
- **isAvailable():** Checks AI service availability and configuration status to enable graceful degradation when external services are unavailable or misconfigured.

j. AdminService

Description: The AdminService handles all administrative operations for system management, user administration, content moderation, and system monitoring. It provides comprehensive tools for administrators to manage users, oversee teacher verification processes, moderate reported content, monitor system health, and generate analytics reports.

Methods:

User Management Methods

- **getUserManagement(filterOptions)** Retrieves a paginated list of all users with filtering options by role, status, registration date, and activity level for comprehensive user administration.
- **banUser(userId, reason, adminId)** Suspends user account, records reason for suspension, logs admin action, and sends notification email to affected user about account status change.
- **unbanUser(userId, adminId)** Reactivates suspended user account, clears suspension reason, logs admin action, and sends account reactivation notification email.
- **updateUserRole(userId, newRole, adminId)** Changes user role (learner/teacher/admin), validates role transition permissions, logs role change, and notifies user of role update.
- **deleteUser(userId, adminId)** Permanently removes user account, anonymizes associated data, handles cascade deletions, and logs deletion action for audit trail.

- **getUserActivity(userId)** Generates comprehensive user activity report including login history, vocabulary usage, classroom participation, and learning progress.

Teacher Verification Methods

- **getTeacherRequests(status, pagination)** Retrieves paginated list of teacher verification requests filtered by status (pending, approved, rejected) with applicant details and submission dates.
- **approveTeacherRequest(requestId, adminId)** Approves teacher verification, updates user role to teacher, updates request status, logs approval action, and sends approval email.
- **rejectTeacherRequest(requestId, reason, adminId)** Rejects teacher verification with detailed reason, updates request status, logs rejection action, and sends rejection email with feedback.

Content Moderation Methods

- **getContentReports(status, pagination)** Retrieves paginated list of content reports filtered by status (open, resolved, dismissed) with reporter details and reported content information.
- **moderateContent(reportId, action, adminId)** Takes moderation action on reported content (remove, edit, warn), updates content status, notifies content creator, and logs moderation decision.
- **dismissReport(reportId, adminId)** Dismisses report as invalid or not requiring action, updates report status, logs dismissal reason, and notifies reporter of decision.

System Analytics Methods

- **getSystemStatistics()** Generates comprehensive system statistics including user counts, vocabulary statistics, classroom metrics, and usage analytics for dashboard.
- **getClassroomOverview()** Provides overview of classroom system including total classrooms, active assignments, learner engagement metrics, and teacher activity.
- **getVocabularyStatistics()** Compiles vocabulary system metrics including total lists, words, most popular lists, and user-generated content statistics.

Audit and Logging Methods

- **getAdminLogs(pagination)** Retrieves paginated admin action logs with details of who performed what actions when for audit trail and compliance.
- **createAdminLog(action, details, adminId)** Creates detailed log entry for admin actions including action type, affected resources, timestamp, and admin user details.

Data Management Methods

- **exportUserData(userId)** Generates complete user data export in compliance with data protection regulations including all user content and activity history.
- **generateSystemReport(reportType, dateRange)** Creates comprehensive system reports (usage, performance, security) for specified date ranges with export functionality.
- System Configuration Methods
- **updateSystemSettings(settings, adminId)** Updates system-wide configuration settings like default user limits, feature toggles, and security policies with change logging.
- **getSystemHealth()** Monitors system health including database performance, service status, error

rates, and resource utilization for operations dashboard.

Communication Methods

- **sendSystemNotification(emails, subject, content)** Sends system-wide notifications or announcements to specified user groups with template support and delivery tracking.
- **broadcastMaintenance(message, scheduledTime)** Broadcasts maintenance notifications to all active users with scheduling and automatic reminder functionality.
- Advanced Analytics Methods
- **getUserEngagementMetrics(dateRange)** Analyzes user engagement patterns including daily active users, session duration, feature usage, and retention rates.
- **getContentQualityMetrics()** Evaluates content quality metrics including report rates, user ratings, and automated quality assessments.
- **getPerformanceMetrics()** Compiles system performance data including response times, error rates, and resource utilization trends.

Security and Compliance Methods

- **reviewSecurityLogs(dateRange)** Analyzes security-related events including failed login attempts, suspicious activities, and security policy violations.
- **generateComplianceReport(regulationType)** Creates compliance reports for various regulations (GDPR, COPPA) including data processing activities and user consent tracking.
- **auditDataAccess(dateRange)** Reviews data access patterns to ensure compliance with privacy policies and identify potential security concerns.

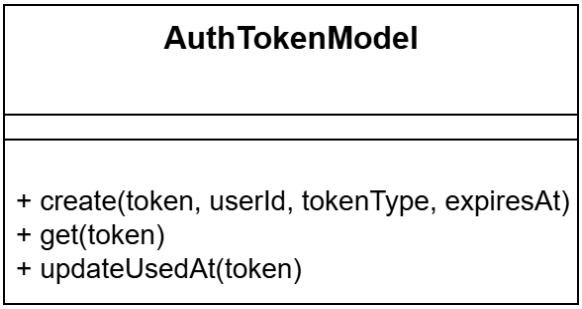
4.6 Component: Model (Data Access Layer)

4.6.1 Responsibilities:

Classes containing methods to perform CRUD (Create, Read, Update, Delete) operations and handle complex business logic related to the data entities. The Model layer abstracts all database interactions using Supabase client, providing clean interfaces for the Service layer to interact with data without direct SQL knowledge.

4.5.2 Key Service Classes:

a. Class: AuthTokenModel

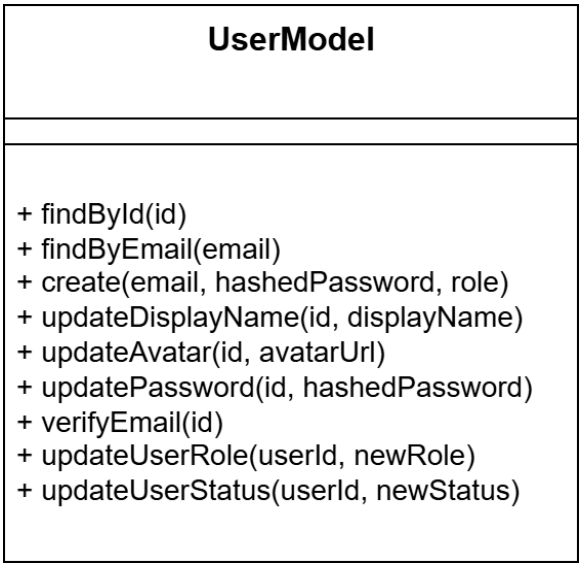


Description: Manages authentication tokens for email verification, password reset, and other time-sensitive operations.

Methods:

- **create(token, userId, tokenType, expiresAt):** Creates new authentication token with expiration time
- **get(token):** Retrieves token details including expiration and usage status
- **updateUsedAt(token):** Marks token as used to prevent replay attacks

b. Class: UserModel

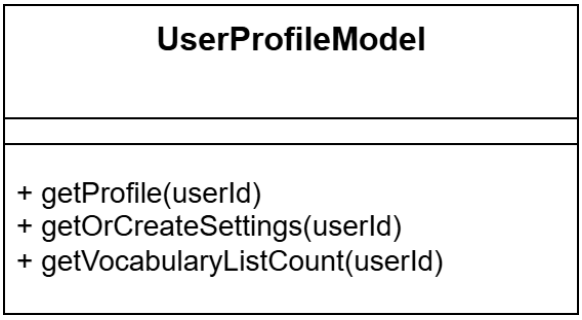


Description: Core user management operations including account creation, authentication data, profile updates, and user status management.

Methods:

- **findById(id):** Retrieves complete user record by primary key
- **findByEmail(email):** Finds user by email address for authentication
- **create(email, hashedPassword, role):** Creates new user account with traditional registration
- **findByGoogleId(id):** Finds user by Google OAuth ID for social login
- **updateGoogleId(userId, googleId):** Links Google account to existing user, marks email as verified
- **updateDisplayName(id, displayName):** Updates user's display name
- **updateAvatar(id, avatarUrl):** Updates user's profile picture URL
- **createGoogleUser(userData):** Creates new user account via Google OAuth with pre-verified email
- **updatePassword(id, hashedPassword):** Updates user password and tracks change timestamp
- **verifyEmail(id):** Marks user email as verified after confirmation
- **hasReportedWord(reporterId, wordId):** Checks if user has already reported specific content
- **createReport(reporterId, wordId, reason):** Creates new content report for moderation
- **updateDailyGoal(userId, dailyGoal):** Updates user's daily learning goal in settings
- **updateUserStatus(userId, newStatus):** Changes user account status (active, inactive, suspended)
- **updateUserRole(userId, newRole):** Updates user role (learner, teacher, admin)

c. Class: UserProfileModel

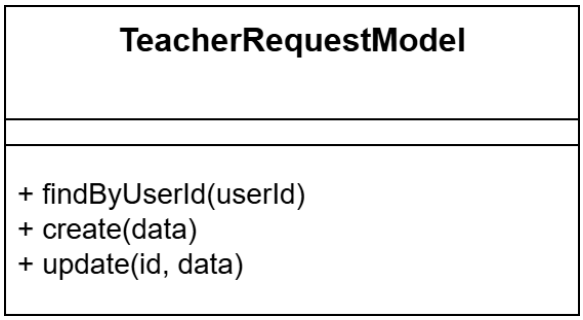


Description: Extended user profile management including settings, preferences, and aggregated statistics.

Methods:

- **getProfile(userId):** Retrieves full user profile including settings and preferences with joins to user_settings table
- **getOrCreateSettings(userId):** Retrieves existing settings or creates default settings for new users
- **getVocabularyListCount(userId):** Counts active vocabulary lists created by user

d. Class: TeacherRequestModel



Description: Manages teacher verification request lifecycle including submission, review, and status tracking.

Methods:

- **findByUserId(userId):** Retrieves latest verification request for user, ordered by creation date
- **create(data):** Creates new teacher verification request with institution and credentials
- **update(id, data):** Updates existing request, resets status to pending and clears rejection reason

e. Class: VocabularyModel

Description: Manages all direct data access operations for vocabulary-related entities including lists, words, examples, synonyms, tags, and user history.

List Management Methods:

- **findListById(listId):** Retrieves single vocabulary list with creator and tags information
- **findUserLists(userId, options):** Retrieves user's vocabulary lists with search, filter, and sort support

- **searchPublicLists(options):** Searches public vocabulary lists by keyword and tags with pagination
- **upsertListHistory(userId, listId):** Records or updates when user last viewed a list for history tracking
- **findHistoryLists(userId, from, to):** Retrieves user's recently viewed vocabulary lists with pagination
- **findPopularLists(from, to):** Retrieves public vocabulary lists sorted by view count
- **updateList(listId, updateData):** Updates vocabulary list properties like title, description, privacy
- **deleteList(listId):** Removes vocabulary list from database
- **findListOwner(listId):** Retrieves creator_id for ownership verification

Word Management Methods:

- **createWord(wordData):** Creates new vocabulary word record in database
- **createWordsBulkAndReturn(wordsToInsert):** Inserts multiple words efficiently and returns created objects
- **findWordWithListInfo(wordId):** Retrieves parent list's creator and privacy for permission checks
- **findWordsByListId(listId, from, to):** Retrieves paginated words for a list with examples and synonyms
- **findWordsByIds(wordIds):** Retrieves multiple complete word objects by their IDs
- **searchInList(listId, options):** Performs text search within specific list with sorting and pagination
- **updateWord(wordId, updateData):** Updates vocabulary word information
- **deleteWord(wordId):** Removes vocabulary word from database
- **findById(id):** Retrieves complete word object with example and synonyms

Example Management Methods:

- **createExamplesBulk(examplesToInsert):** Inserts multiple example sentences efficiently
- **upsertExample(wordId, exampleData):** Creates or updates example sentence for a word
- **deleteExample(wordId):** Removes example sentence for specific word

Synonym Management Methods:

- **createSynonyms(synonymsToInsert):** Inserts multiple synonym records, ignoring duplicates
- **deleteSynonymsByWordId(wordId):** Removes all synonyms for specific word

Tag Management Methods:

- **findAllTags():** Retrieves all available tags in the system
- **findTagsByName(tagNames):** Retrieves tag records for validation
- **associateTagsToList(listTagRelations):** Links tags to vocabulary list
- **disassociateAllTagsFromList(listId):** Removes all tag associations for list

Database Function Methods:

- **createListWithTags(listData):** Atomically creates list and associates tags using database function
- **updateWordCount(listId):** Recalculates and updates word count for list using database function

f. Class: ClassroomModel

ClassroomModel

+ createClassroom(classroomData)
+ isJoinCodeUnique(code)
+ findByTeacherId(teacherId)
+ findByTeacherIdWithAssignmentCounts(teacherId)
+ getClassroomByCode(joinCode)
+ getClassroomCount(teacherId)
+ findMemberStatus(classroomId, userId)
+ createJoinRequest(classroomId, learnerId, email)
+ getClassroomById(classroomId)
+ isJoinedLearner(classroomId, userId)
+ getPendingJoinRequests(classroomId)
+ updateLearnerStatus(classroomId, learnerId, fields)
+ approveAllPendingRequests(classroomId)
+ getJoinedLearners(classroomId)
+ softDeleteClassroom(classroomId)
+ searchLearnersByDisplayName(classroomId, status, keyword)
+ upsertInvitation(invitationData)
+ getInvitationByToken(token)
+ updateInvitationStatus(invitationId, status)
+ addLearnerToClass(classroomId, learnerId, email)
+ findInvitation(classroomId, email)
+ cancelInvitation(classroomId, email)
+ createAssignment(data)
+ createAssignmentSublist(data)
+ getJoinedClassroomsByLearner(learnerId)
+ getJoinedClassroomsByLearnerWithAssignmentCounts(learnerId)
+ getInvitationsByClassroomId(classroomId)
+ getLearnerAssignmentsByClassroomAndLearner(classroomId, learnerId)
+ createLearnerAssignmentsBatch(assignments)
+ getAssignmentsByClassroom(classroomId)
+ getLearnerAssignmentsByStatus(classroomId, learnerId, statusList)
+ hasLearnerAssignment(assignmentId, learnerId)
+ createLearnerAssignment(assignmentData)
+ getAssignmentById(classroomId, assignmentId)
+ getAssignmentVocabulary(vocabListId)
+ countLearnersCompleted(assignmentId)
+ updateAutoApproval(classroomId, value)
+ getAssignmentSublists(assignmentId)
+ deleteAssignmentSublists(assignmentId)
+ deleteLearnerAssignmentsByAssignmentId(assignmentId)
+ deleteLearnerAssignmentsByLearnerId(learnerId)
+ deleteAssignmentRecord(assignmentId)

Description: Handles all database operations for classroom functionality including classroom management, member management, assignments, and invitations.

Classroom Management Methods:

- **createClassroom(classroomData):** Creates new classroom record with unique join code
- **isJoinCodeUnique(code):** Validates uniqueness of classroom join code
- **findByTeacherId(teacherId):** Retrieves all non-deleted classrooms for specific teacher
- **findByTeacherIdWithAssignmentCounts(teacherId):** Fetches teacher's classrooms with assignment counts
- **getClassroomByCode(joinCode):** Finds classroom using unique join code
- **getClassroomCount(teacherId):** Counts total classrooms created by teacher
- **getClassroomById(classroomId):** Retrieves single classroom details by ID
- **softDeleteClassroom(classroomId):** Marks classroom as deleted without removing data
- **updateAutoApproval(classroomId, value):** Updates auto-approval setting for join requests

Member Management Methods:

- **findMemberStatus(classroomId, userId):** Checks learner's membership status in classroom
- **createJoinRequest(classroomId, learnerId, email):** Creates or updates join request with pending status
- **isJoinedLearner(classroomId, userId):** Verifies if user is active classroom member
- **getPendingJoinRequests(classroomId):** Retrieves all pending join requests with user details
- **updateLearnerStatus(classroomId, learnerId, fields):** Updates member status and related fields
- **approveAllPendingRequests(classroomId):** Bulk approves all pending join requests
- **getJoinedLearners(classroomId):** Retrieves all successfully joined learners with details
- **searchLearnersByDisplayName(classroomId, status, keyword):** Searches classroom members by name and status

Invitation Management Methods:

- **upsertInvitation(invitationData):** Creates or updates classroom invitation with token
- **getInvitationByToken(token):** Retrieves invitation details using unique token
- **updateInvitationStatus(invitationId, status):** Marks invitation as used or cancelled
- **addLearnerToClass(classroomId, learnerId, email):** Adds learner with joined status after invitation acceptance
- **findInvitation(classroomId, email):** Finds specific invitation by classroom and email
- **cancelInvitation(classroomId, email):** Cancels pending invitation by updating status
- **getInvitationsByClassroomId(classroomId):** Retrieves all pending invitations for classroom

Assignment Management Methods:

- **createAssignment(data):** Creates new assignment record in database

- **createAssignmentSublist(data):** Creates sublist record for assignment division
- **getAssignmentsByClassroom(classroomId):** Retrieves all assignments for specific classroom
- **getAssignmentById(classroomId, assignmentId):** Fetches single assignment details
- **getAssignmentVocabulary(vocabListId):** Retrieves terms and definitions for assignment
- **getAssignmentSublists(assignmentId):** Gets all sublist records for assignment
- **deleteAssignmentSublists(assignmentId):** Removes all sublists for assignment
- **deleteAssignmentRecord(assignmentId):** Deletes main assignment record

Learner Assignment Methods:

- **getLearnerAssignmentsByClassroomAndLearner(classroomId, learnerId):** Gets assignment IDs for learner using database function
- **createLearnerAssignmentsBatch(assignments):** Bulk creates learner assignment records
- **getLearnerAssignmentsByStatus(classroomId, learnerId, statusList):** Retrieves learner assignments by status list
- **hasLearnerAssignment(assignmentId, learnerId):** Checks if learner assignment exists
- **createLearnerAssignment(assignmentData):** Creates single learner assignment record
- **countLearnersCompleted(assignmentId):** Counts learners who completed assignment
- **deleteLearnerAssignmentsByAssignmentId(assignmentId):** Removes all learner progress for assignment
- **deleteLearnerAssignmentsByLearnerId(learnerId):** Removes all assignment progress for learner

Classroom Membership Methods:

- **getJoinedClassroomsByLearner(learnerId):** Retrieves all classrooms learner has joined
- **getJoinedClassroomsByLearnerWithAssignmentCounts(learnerId):** Gets learner's classrooms with active assignment counts

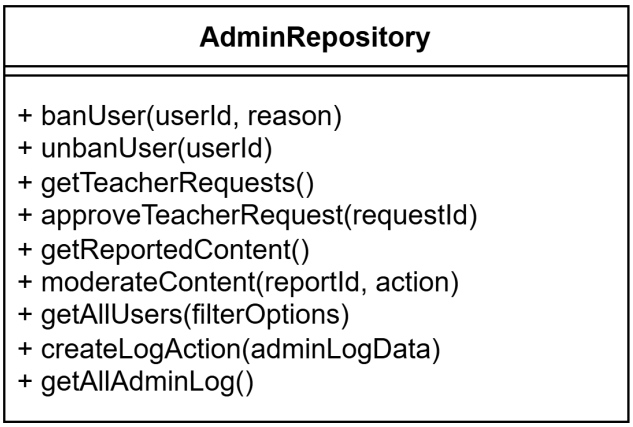
Vocabulary Integration Methods:

- **getWordsByListId(listId):** Retrieves all words for vocabulary list
- **cloneListWithWords(originalListId, creatorId, title, words):** Creates copy of vocabulary list with words for assignment
- **deleteVocabLists(listIds):** Removes multiple vocabulary lists (for assignment cleanup)

User Integration Methods:

- **findById(userId):** Retrieves user record by ID
- **findByEmail(email):** Finds user by email address

g. Class: AdminModel

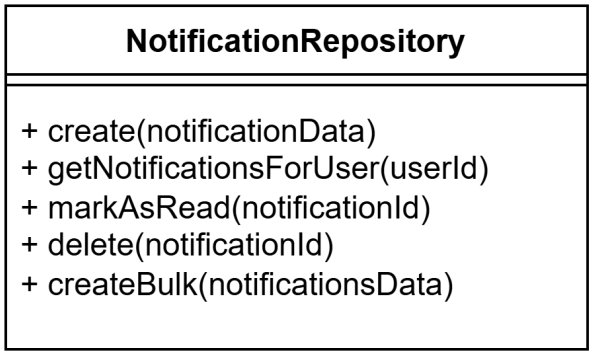


Description: Manages data operations exclusive to administrators.

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.

h. Class: NotificationModel

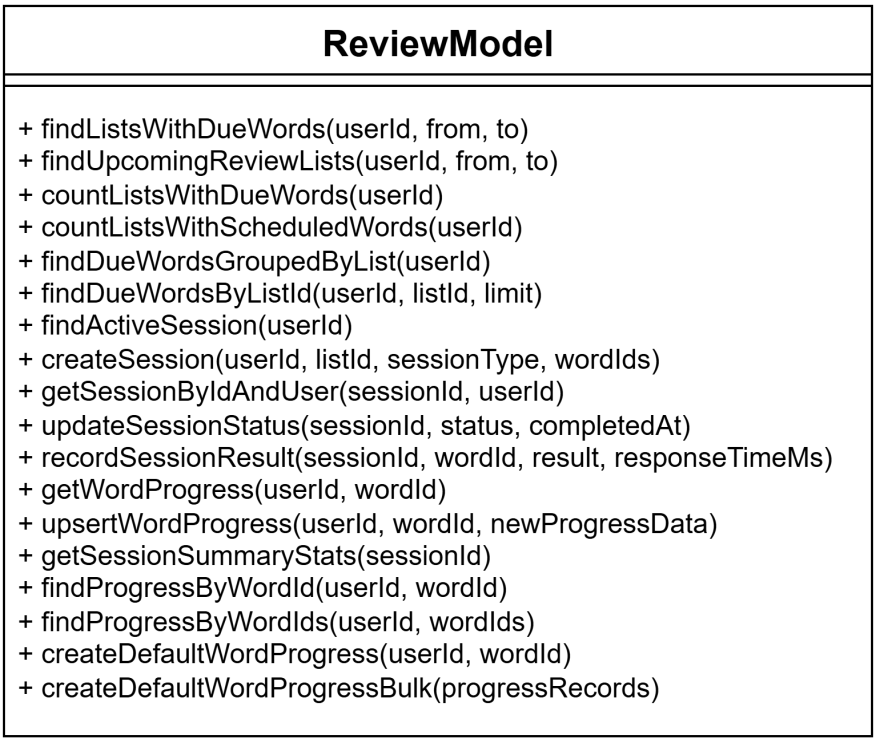


Description: Manages the creation and retrieval of notifications for users.

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).

i. Class: ReviewModel



Description: Manages all direct data access operations for spaced repetition schedules, review sessions, and user progress tracking.

Due Words and Lists Methods:

- **findListsWithDueWords(userId, from, to):** Retrieves vocabulary lists containing words currently due for review
- **findUpcomingReviewLists(userId, from, to):** Retrieves lists with words due for review in next 7 days
- **countListsWithDueWords(userId):** Counts unique lists with due words for pagination

- **countListsWithScheduledWords(userId):** Counts lists with upcoming scheduled reviews
- **findDueWordsGroupedByList(userId):** Fetches all due words summarized by their parent list
- **findDueWordsByListId(userId, listId, limit):** Retrieves due words for specific list with full details

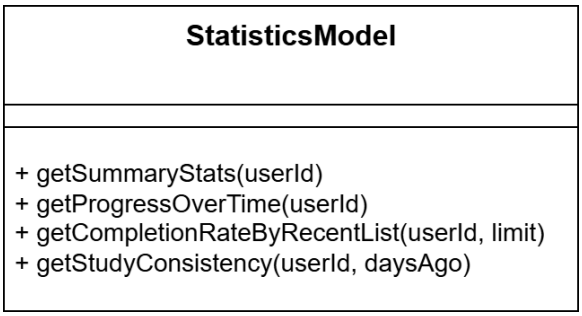
Session Management Methods:

- **findActiveSession(userId):** Checks for ongoing review session for user
- **createSession(userId, listId, sessionType, wordIds):** Creates new review session record
- **getSessionByIdAndUser(sessionId, userId):** Retrieves specific session ensuring user ownership
- **updateSessionStatus(sessionId, status, completedAt):** Updates session status to completed or interrupted

Progress Tracking Methods:

- **recordSessionResult(sessionId, wordId, result, responseTimeMs):** Logs user's answer for word in session
- **getWordProgress(userId, wordId):** Retrieves spaced repetition progress for single word
- **upsertWordProgress(userId, wordId, newProgressData):** Creates or updates word progress after review
- **getSessionSummaryStats(sessionId):** Retrieves all results for session to calculate summary
- **findProgressByWordId(userId, wordId):** Retrieves detailed progress for single word
- **findProgressByWordIds(userId, wordIds):** Efficiently retrieves progress for multiple words
- **createDefaultWordProgress(userId, wordId):** Creates initial progress record for new word
- **createDefaultWordProgressBulk(progressRecords):** Bulk creates default progress for multiple words

j. Class: StatisticModel



Description: Manages user learning statistics, progress tracking, and analytics data for performance insights.

Methods:

- **getSummaryStats(userId):** Retrieves overall user statistics like total vocabulary, streaks, and study time
- **getProgressOverTime(userId):** Fetches monthly progress data showing cumulative words mastered over time
- **getCompletionRateByRecentList(userId, limit):** Calculates completion rates for user's recently

studied lists
- **getStudyConsistency(userId, daysAgo):** Retrieves study session dates for consistency calendar visualization

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.

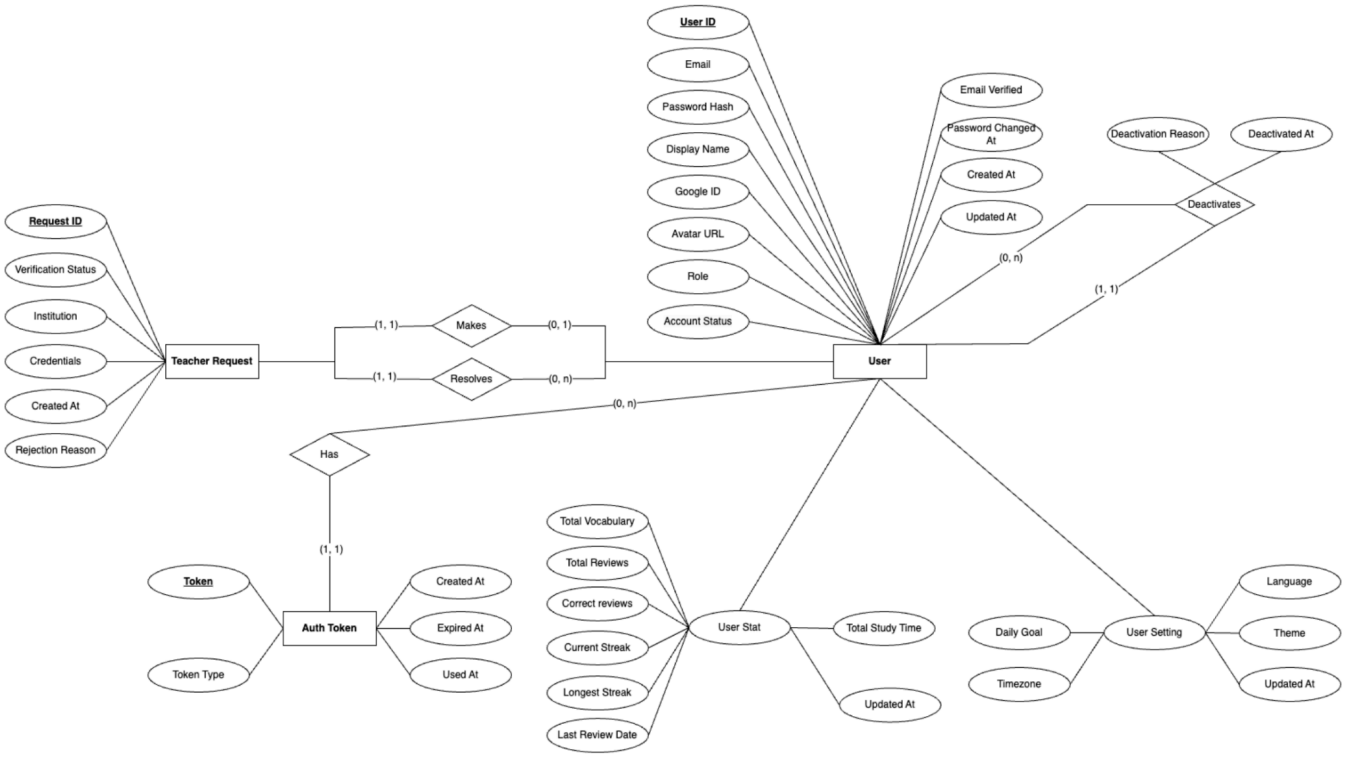


Table Overviews:

- **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.

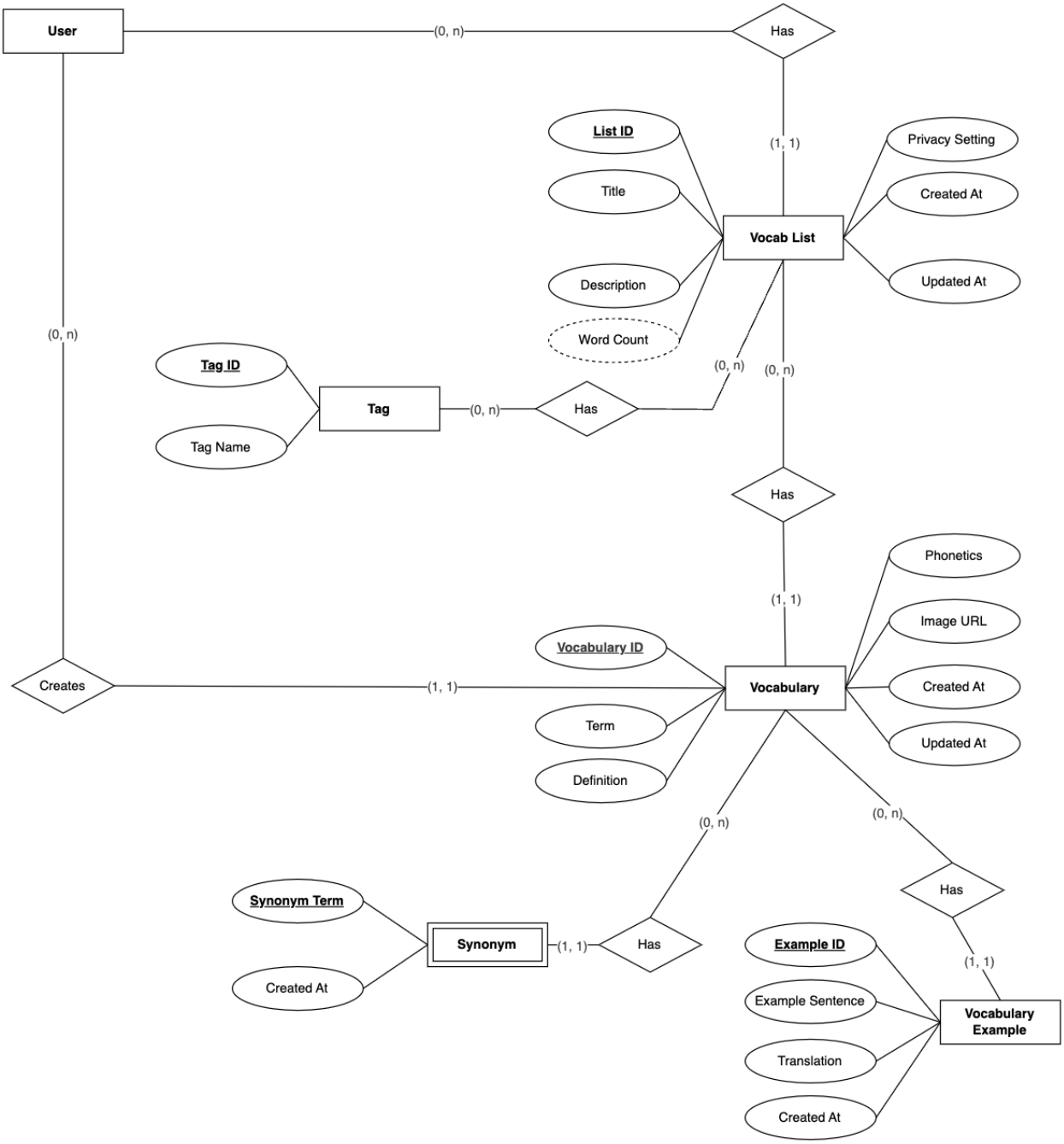


Table Overviews:

- **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.

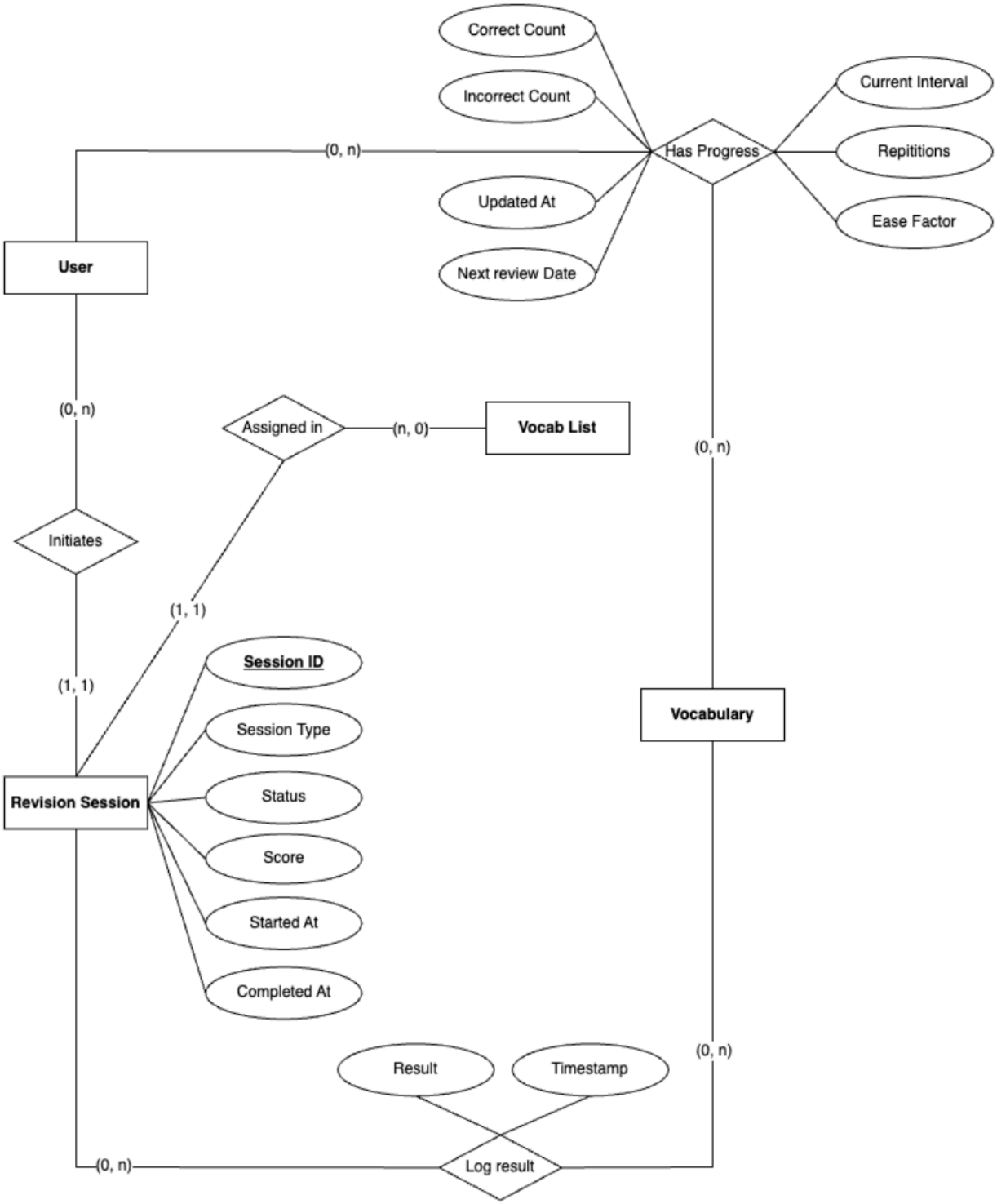

Correct Count
Incorrect Count
Updated At
Next review Date
Has Progress
Current Interval
Repititions
Ease Factor
(0, n)
User
(0, n)
Assigned in
(n, 0)
Vocab List
(0, n)
Initiates
(1, 1)
(1, 1)
Session ID
Session Type
Status
Score
Started At
Completed At
Revision Session
Vocabulary
(0, n)
Result
Timestamp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

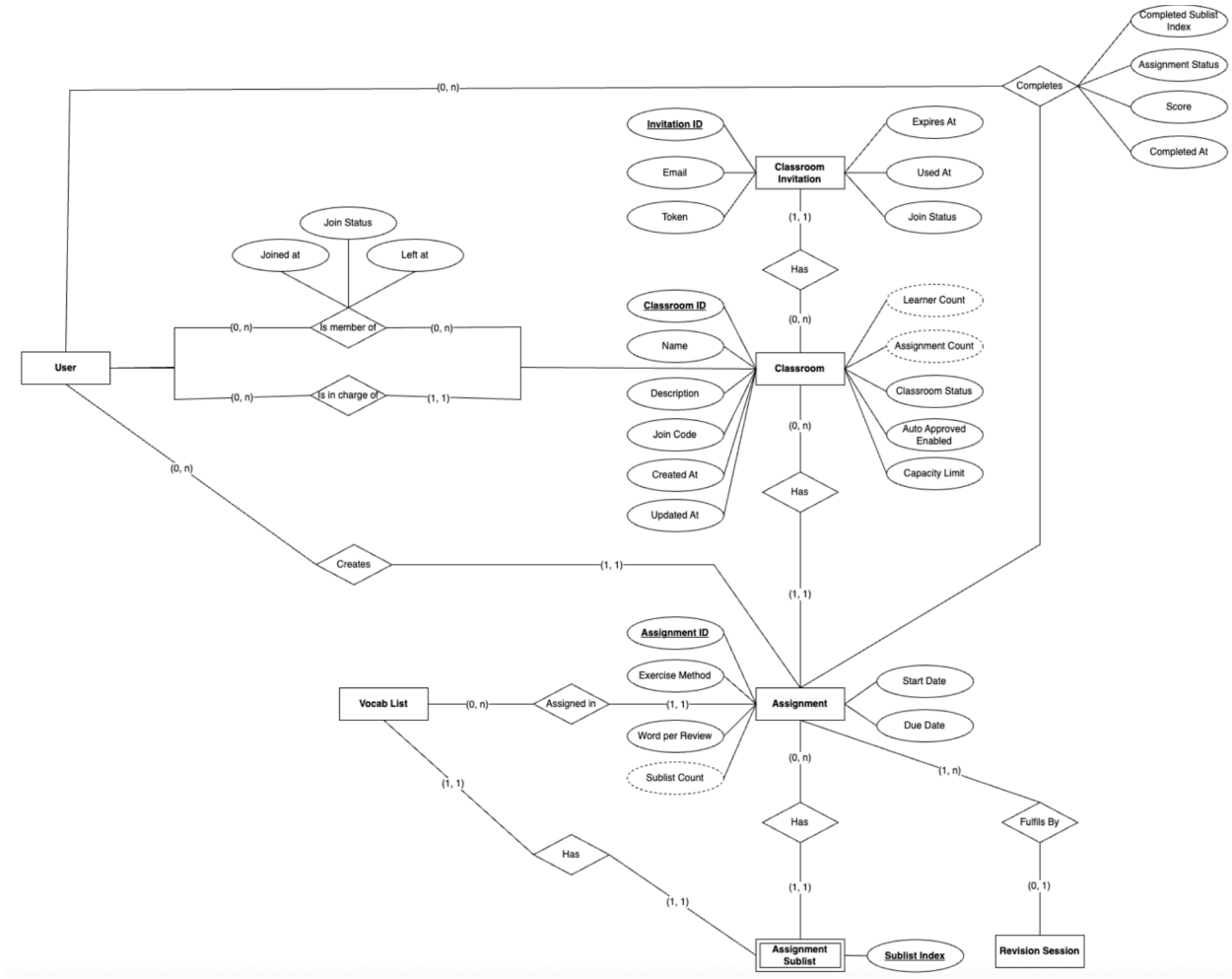


Table Overviews:

- **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.

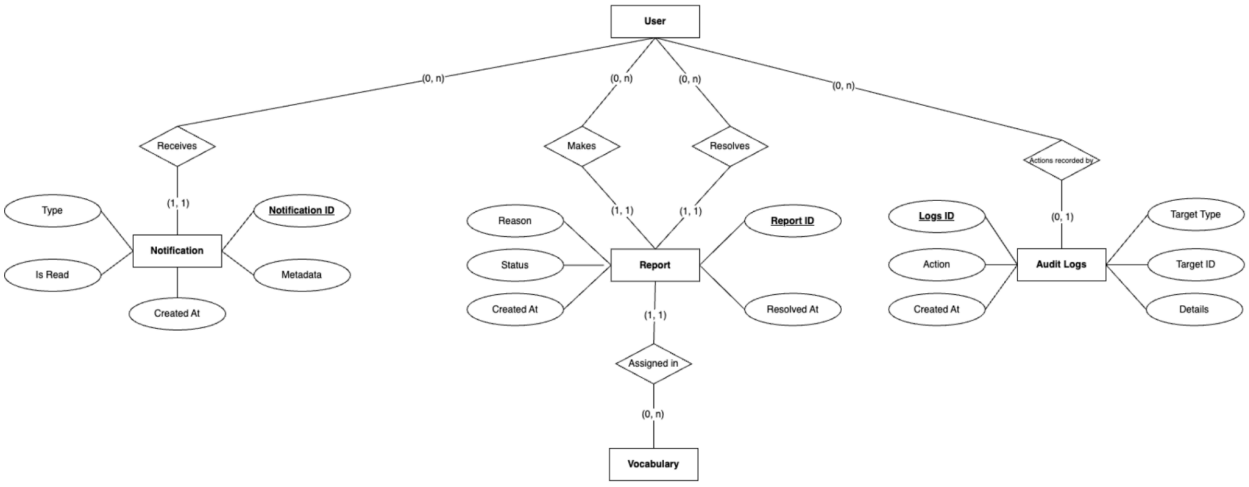
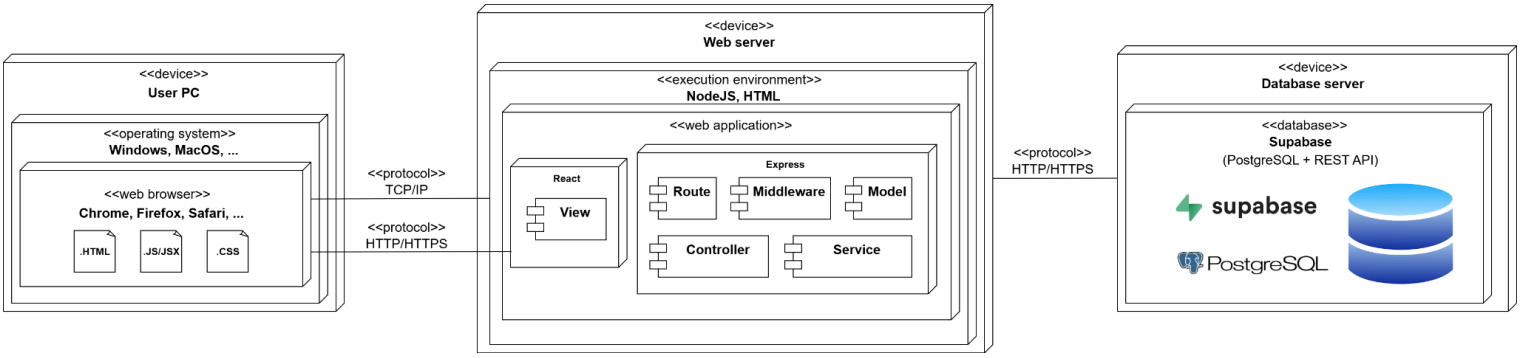


Table Overviews:

- **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

Deployment diagram of web application



5.1 Client Tier

Devices:

- Users access VocaBoost via various devices such as personal computers, laptops.

Operating Systems and Browsers:

- Compatible with popular operating systems (Windows, macOS, Linux).
- Supports modern browsers like Chrome, Firefox, Safari, and Edge.

Execution Environment

- The frontend application is developed with **React 19**, compiled and bundled by **Vite**, and executed within the user's browser environment.

Content Delivered

- HTML, CSS, and JavaScript files, resulting from transpilation (JSX → JS), bundling, and rendering performed by Vite.

Communication Protocols

- Uses HTTP/HTTPS to communicate with the Web Server.
- Employs TCP/IP as the underlying network protocol for establishing browser-server connections.

5.2 Web Server Tier

Hosting Environment: Deployed on cloud infrastructure such as Vercel, Daytona, or similar cloud providers.

Server Environment

- **Node.js 20** runtime environment.
- Backend application powered by **Express.js** version 5.

Communication Protocol: Communicates with **Supabase** via secure REST API endpoints over HTTP/HTTPS, facilitated by the supabase-js client library.

5.3 Database Tier

Database System: Hosted on Supabase, a managed cloud-based database built upon PostgreSQL.

Data Storage: Employs normalized, structured schemas optimized for scalability, integrity, and

efficient querying.

Key Database Schemas

- **User**: Stores user credentials, profile information, roles, and status.
- **Vocabulary**: Stores vocabulary lists and vocabulary items, including metadata and contextual examples.
- **Progress**: Tracks learners' progress and review schedules, utilizing a spaced repetition system.
- **Classroom**: Manages classroom structures, student enrollments, and interactions.
- **Assignment**: Tracks assignments issued within classrooms and student submissions.
- **Notification**: Manages user notifications for system interactions and alerts.
- **Token**: Handles verification workflows for email, password resets, and teacher approval processes.
- **Report**: Records and manages inappropriate content.

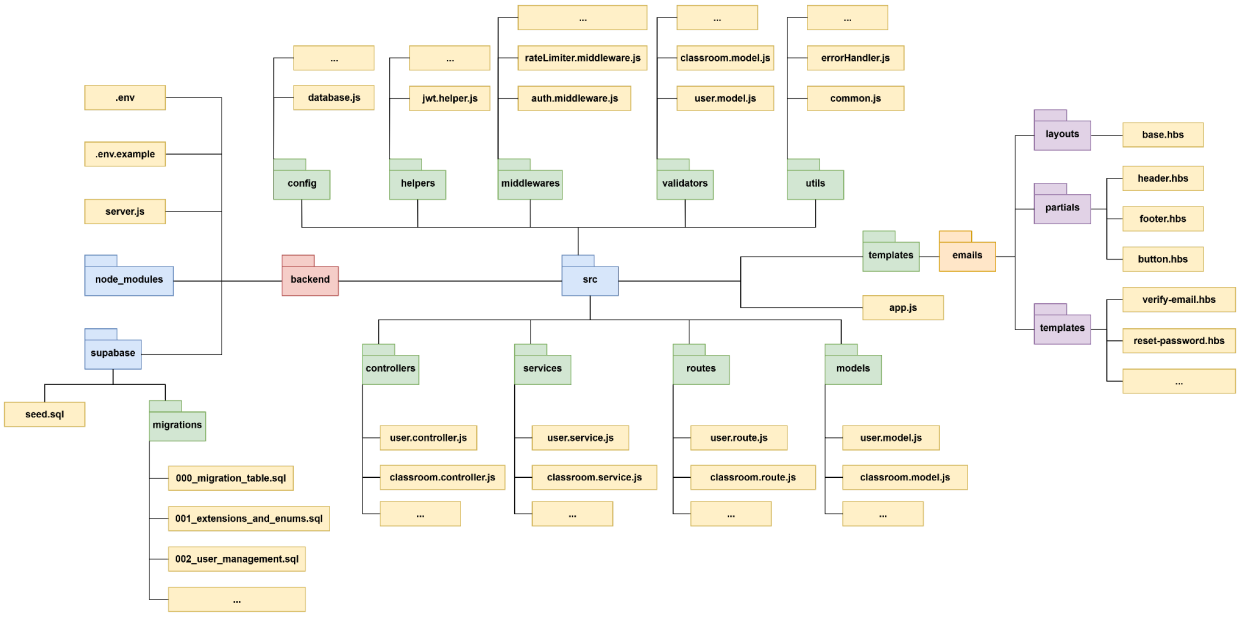
Communication Protocol: Accessible exclusively through secure REST APIs provided by Supabase, utilizing HTTP/HTTPS.

5.4 End-to-End Communication Flow

1. **Client Interaction**: Users initiate interaction via web browsers, loading the React-based frontend from the Web Server.
2. **Frontend-Backend Communication**: React application sends HTTP/HTTPS requests to the backend API endpoints provided by Node.js/Express server.
3. **Backend Processing**: Express backend processes requests through middleware, routes them to controllers, and delegates core processing to business logic within services.
4. **Database Interaction:** Model uses the supabase-js library to interact with the Supabase database via secure RESTful calls, ensuring robust data handling and security.
5. **External Service Integration**: The backend communicates with external services such as Google Gemini AI for AI-driven vocabulary suggestions and SMTP providers for sending email notifications.
6. **Response to Client**: Processed data is returned securely through the backend to the React frontend, which dynamically renders the information for the user.

6. Implementation View

1. Backend

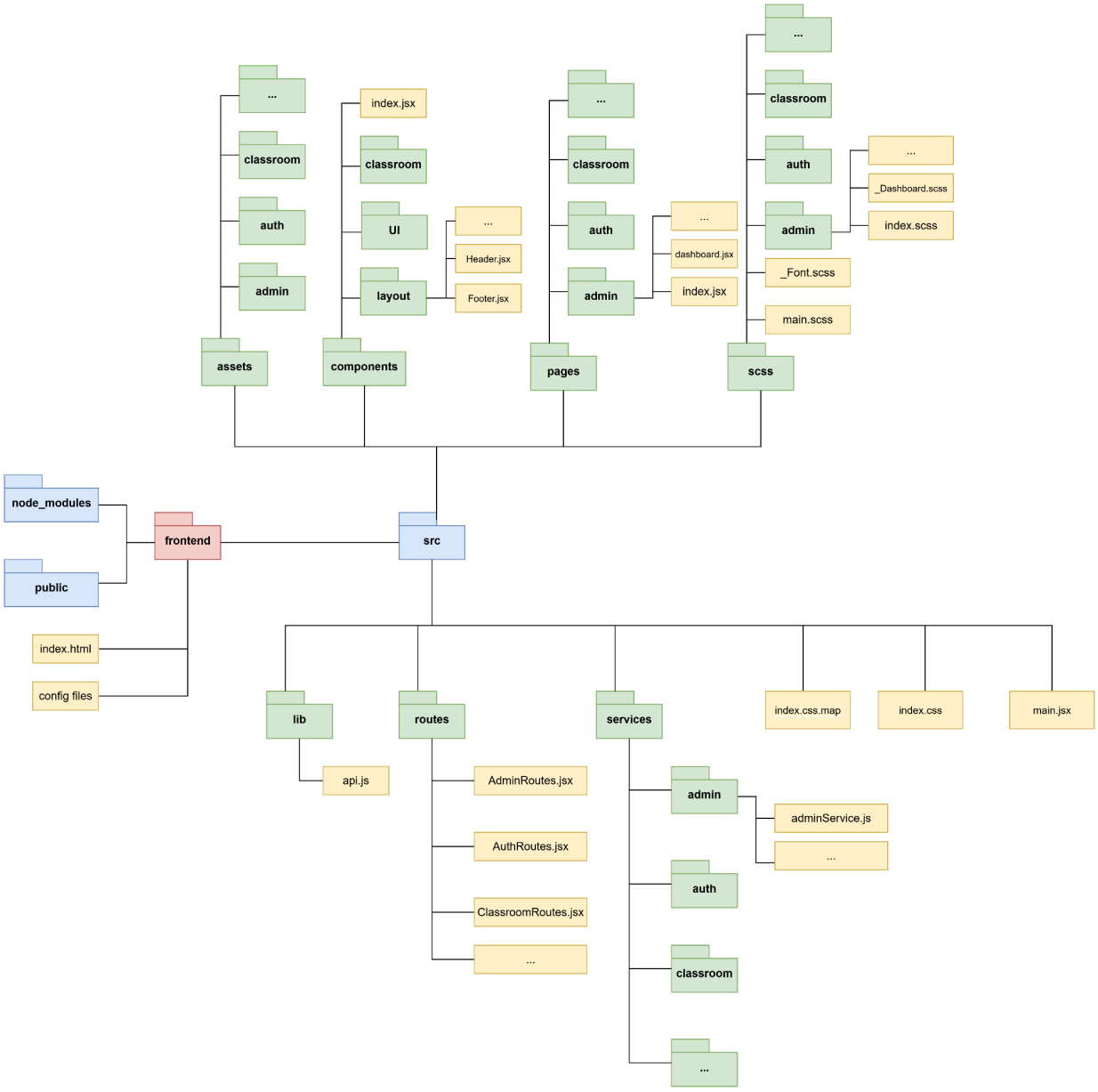


- **src/**: The core source directory for backend logic, organized into modules based on functionality
  - **app.js**: The main application file. It is responsible for setting up the Express framework, configuring global middlewares, and mounting the main application routes.
  - **config/**: Contains configuration files. For example, database.js might hold the database connection.
  - **utils/**: Contains general-purpose utility functions that support common tasks like logger formatting or error handling.
  - **helpers/**: Contains small, reusable utility functions that can be used across the application. These functions typically do not belong to any specific business logic.
  - **routes/**: Defines the API endpoints. Each route file maps a URL and an HTTP method (GET, POST, PUT, DELETE) to a specific handler function in a controller.
  - **middlewares/**: Contains middleware functions. Middleware are functions that execute between receiving a request and passing it to the controller. They are commonly used for tasks like user authentication, authorization, logging, or rate limiting.
  - **validators/**: Contains the logic for validating incoming data from the client. Before processing a request, validators check if the request body, params, or query is valid and conforms to the defined data schema.
  - **controllers/**: Handles incoming HTTP requests and delegates work to the corresponding service layer
  - **services/**: This layer contains all the core business logic of the application, sitting between controllers and models.
  - **models/**: Handles direct interaction with the database
- **server.js**: The application's entry point. It is responsible for creating a server instance, listening

for incoming connections on a specific port, and starting the application.

- **.env**: Environment variables (not in version control)
- **.env.example**: Template for environment variables
- **node_modules/**: The standard Node.js directory containing all project libraries and dependencies managed by npm.
- **supabase/**: This directory is dedicated to managing the database and Supabase-related configurations.
    - **seed.sql**: Contains SQL scripts to insert seed data into the database. This file is useful for initializing a development or testing environment with initial data.
    - **migrations/**: Contains SQL migration files. Each file represents a sequential change to the database structure.
- **templates/** Contains Handlebars email templates for the notification system
    - **emails/layouts/**: Base layout templates providing consistent email structure
    - **emails/partials/**: Reusable components (headers, footers, buttons) for email composition
    - **emails/templates/**: Specific email templates for various system notifications

2. Front end



- **src/:** the main directory housing the frontend application logic
  - **assets/:** contains ".svg" patterns which are used to decorate websites.
  - **components/:** contains components which are modular and promote reusability.
  - **pages/:** represents page-level React components, each file is a single page managed by a folder and imported to "index.jsx" to import to routes file easily.
  - **scss/:** contains "scss" files used to style the website. Each subfolder represents a module or component group and includes its own "scss" files. Each of these folders has an index.scss file that forwards all internal styles. Finally, all ".scss" files are imported into a central main.scss.

- **lib/:** contains “api.js” which is in contact with the back end.
- **routes/:** defining route groupings which manage navigate pages without reloading pages.
- **services/:** contains all logic related to making HTTP requests to retrieve, create, update, or delete data on the server.
- **index.css.map:** mapped file “scss” to compile to one css file to include to index.html
- **index.css:** a file where “index.css.map” is mapped.
- **main.jsx:** the application entry point
- **index.html:** matches standard usage in React apps to serve static assets and the base HTML.
- **node_modules/**: The standard Node.js directory containing all project libraries and dependencies managed by npm.
- **public/:** store static assets that are not processed by Webpack or Vite.
- **config files:** runtime config, static metadata, or hosted API keys.