26.06.2025

Team B17

Developer Documentation Modular Skills Assessment Tool – System Documentation by Team B17

Developer Documentation – Table of Contents

1. Project Overview

* 1.1 Project Name and Description
* 1.2 Team Members
* 1.3 Purpose of the System

2. Technologies in Our Project

* 2.1 React (Client Side)
  + **2.1.1 React Components – Creating interfaces through reusable components**
  + **2.1.2 JSX – HTML-like syntax for structuring components in JavaScript**
  + **2.1.3 DOM (Document Object Model) – Tree structure generated from HTML**
  + **2.1.4 Virtual DOM – In-memory copy used to optimize updates**
* 2.2 Tailwind CSS
* 2.3 Vite (Build Tool)
* 2.4 Node.js + Express (Server Side)
* 2.5 MongoDB Atlas (Database)
* 2.6 Vercel Deployment (Monorepo)

3. Application Flow

* 3.1 React SPA Structure
* 3.2 App.jsx (Routing & Structure)

4. Functional Components

*(For each major component, we provide: Functionality, Functional Description, Parameters, Returns, and Used In)*

4.1 Lecturer Components

* 4.1.0 LoginPage.jsx *(Lecturer login)*
* 4.1.1 TeacherHeader.jsx
* 4.1.2 Teacher.jsx
* 4.1.3 QuickActions.jsx
* 4.1.4 RecentActivity.jsx
* 4.1.5 DashboardOverview.jsx
* 4.1.6 Create\_New\_Class.jsx
* 4.1.7 ClassForm.jsx
* 4.1.8 ViewClasses.jsx
* 4.1.9 ClassManager.jsx
* 4.1.10 ClassCard.jsx
* 4.1.11 ConfirmModal.jsx
* 4.1.12 ClassStudentReports.jsx
* 4.1.13 StudentReportCard.jsx
* 4.1.14 SimulationChart.jsx
* 4.1.15 StudentHeader.jsx
* 4.1.16 ExportButton.jsx
* 4.1.17 ClassDetails.jsx
* 4.1.18 SimulationBox.jsx
* 4.1.19 StudentAnswerCard.jsx
* 4.1.20 Student Progress for Teacher Components

• 4.1.20.1 AllReports.jsx  
• 4.1.20.2 ClassProgressCard.jsx  
• 4.1.20.3 pdfExporter.js  
• 4.1.20.4 ProgressOfChosenStudent.jsx  
• 4.1.20.5 StudentCard.jsx  
• 4.1.20.6 StudentOverviewHeader.jsx

* 4.1.21 AIChat.jsx
* 4.1.22 NotificationsContext.jsx

4.2 Student Components

* 4.2.1 StudentHome.jsx
* 4.2.2 StudentLogin.jsx *(Student login)*
* 4.2.3 StudentHeader.jsx *(from studentPages)*
* 4.2.4 MyProgress.jsx
* 4.2.5 MyReports.jsx
* 4.2.6 ShowClasses.jsx
* 4.2.7 StudentSimulation.jsx
* 4.2.8 StudentSimulationBox.jsx
* 4.2.9 AnswerCard.jsx
* 4.2.10 ClassesDoneSimulation.jsx
* 4.2.11 SimulationResult.jsx
* 4.2.12 StudentRecentActivities.jsx
* 4.2.13 StudentAIChat.jsx
* 4.2.14 StudentNotificationsContext.jsx

4.3 Shared Components

* 4.3.1 SharedHeader.jsx
* 4.3.2 Footer.jsx
* 4.3.3 Button.jsx
* 4.3.4 FormInput.jsx
* 4.3.5 Alert.jsx
* 4.3.6 ThemeToggle.jsx
* 4.3.7 ThemeContext.jsx
* 4.3.8 Homepage.jsx
* 4.3.9 AboutSEL.jsx
* 4.3.10 FeatureCards.jsx
* 4.3.11 FeaturesModal.jsx
* 4.3.12 UserContext.jsx

4.4 Auth Components (Lecturer & Student)

* 4.4.1 RegisterPage.jsx
* 4.4.2 ForgotPassword.jsx
* 4.4.3 ResetPassword.jsx
* 4.4.4 VerifyCode.jsx

6. Hooks

* 6.1 React Built-in Hooks
  + useState
  + useRef
  + useEffect
  + useContext
  + useParams
  + useLocation
  + useNavigate
* 6.2 Custom Hooks
  + useNotifications

7. Special Utility Functions

* 7.1 export PDF Functions
* 7.2 7.2 Server Communication Functions
  + POST
  + PATCH
  + DELETE

8. Server-Side API

* 8.1 Server Initialization – app.js
* 8.2 Mongoose Schemas Overview
  + 8.2.1 ClassSchema.js
  + 8.2.2 NotificationSchema.js
  + 8.2.3 StudentNotificationSchema.js
  + 8.2.4 StudentSchema.js
  + 8.2.5 TeacherSchema.js
* 8.3 API Route Files

• 8.3.1 classes.route.js

• 8.3.2 claude.route.js

• 8.3.3 notifications.route.js

• 8.3.4 studentNotifications.route.js

• 8.3.5 students.route.js

• 8.3.6 summary.route.js

• 8.3.7 teachers.route.js

• 8.3.8 teacherStudentProgress.route.js

* 8.4 Services
  + 8.4.1 classInsightService.js
  + 8.4.2 claudeService.js
  + 8.4.3 sendVerificationEmail.js
  + 8.4.4 studentAnalysisService.js

9. Special Functions of React

* + 9.1 createContext
  + 9.2 Context Provider Components
  + 9.3 useContext
  + 9.4 Example – Wrapping the App with Providers
  + 9.5 Example – Accessing Context in a Component

9. Environment and Tools

* 9.1 .env Configuration
* 9.2 Environment Variables on Vercel
* 9.2 Special Configurations (vercel.json)

10. AI Prompts Used in the Project

* 10.1 Class Insight Prompt – Generate Summary for Teacher
* 10.2 Student Response Analysis Prompt – CASEL 5 Evaluation
* 10.3 AI Student Chat Prompt
* 10.4 AI Teacher Chat Prompt
* 10.5 Simulation Generation Prompt – When Creating a New Class

11. Login Credentials for Testing

* 11.1 Teacher Demo
* 11.2 Student Demo

12. References

* 12.1 Code Snippets Taken from External Sources
* 12.2 Documentation / Articles Used

13. Links

* 13.1 Database Connection (MongoDB URI)
* 13.2 GitHub Repository
* 13.3 Live Deployment (Vercel)

14. External APIs and Integrations

• 14.1 Claude AI (via Axios & Anthropic API)  
• 14.2 Email Verification Service (Nodemailer)  
• 14.3 MongoDB Atlas

1. Project Overview (English)

1.1 Project Name and Description

Modular Skills Assessment Tool  
This project focuses on designing a web-based modular skills assessment tool with a strong emphasis on SEL (Social-Emotional Learning) domains.  
The system visually maps students’ academic strengths and weaknesses, especially across SEL competencies, and aims to guide focused recovery learning through AI-based analysis.

1.2 Team Members

* Nahla Aboromy
* Manal Naami
* Roba Hassan
* Nohamed Atamneh
* Fatima Dabbah
* Wael Swaid

1.3 Purpose of the System:

The Modular Skills Assessment Tool is designed to provide a clear and in-depth visual overview of each student’s progress in SEL (Social-Emotional Learning) domains—for both lecturers and students. The system focuses on analyzing the five core CASEL competencies: Self-Awareness, Self-Management, Social Awareness, Relationship Skills, and Responsible Decision-Making. Lecturers receive a comprehensive view of their entire class, including interactive progress maps and visual graphs that highlight both individual and group trends over time. For each student, the system presents detailed CASEL analysis, recurring strengths or persistent weaknesses, and targeted alerts where necessary. Each analysis includes: Overall Score, Strengths, Areas for Improvement, Depth Level, and a Suggested Intervention tailored to the student's needs. These features empower lecturers to gain a deep understanding of class dynamics and to make informed, data-driven educational decisions.

Students also receive personalized, structured feedback after each simulation attempt, including all the analysis elements mentioned above. The system allows students to submit the same simulation multiple times, and each attempt is independently analyzed by the AI model. In addition, students are presented with a personal progress graph that shows how their SEL competencies evolve over time. The feedback is developmental rather than technical, helping students recognize their personal strengths, reflect on areas for improvement, and evaluate their emotional and social decision-making. Altogether, the system supports a process of Focused Recovery Learning, built on continuous reflection, personalized feedback, meaningful growth, and student empowerment.

Technologies in Our Project

2.1 React (Client Side)

2.1.1 React Components

React is a JavaScript library for building user interfaces.

* We use React components to create small, reusable parts of the interface (e.g., buttons, forms, cards).
* Components can be reused across different parts of the project to maintain consistency and reduce code duplication.
* React relies on the Virtual DOM to update only what is needed on the page, improving performance and efficiency.

2.1.2 JSX

JSX (JavaScript XML) is a syntax extension for JavaScript.

* It allows us to write HTML-like code directly within JavaScript.
* JSX makes it easier to visualize component structure and logic in one place.
* It compiles to regular JavaScript using tools like Babel.

2.1.3 DOM (Document Object Model)

The DOM is the structure that the browser creates from our HTML.

* It represents the page as a tree of elements (nodes).
* JavaScript can interact with the DOM to change the page dynamically.
* React interacts with the Virtual DOM first, and then efficiently updates the real DOM.

2.1.4 Virtual DOM

The Virtual DOM is a lightweight copy of the real DOM stored in memory.

* React updates the Virtual DOM first when something changes.
* It then compares it (diffing) to the previous version to detect changes.
* Finally, only the necessary changes are applied to the real DOM.
* This makes the app faster and improves performance.

2.2 Tailwind CSS

Tailwind CSS is a utility-first CSS framework that we use to style our application.

* It provides ready-made classes for spacing, colors, typography, and layout directly in the JSX code.
* Enables us to design custom interfaces quickly without writing custom CSS for each element.
* Helps keep the codebase clean and consistent by reusing style utilities across components.

2.3 Vite (Build Tool)

Vite is a modern build tool that we use to run and build the project.

* It provides fast development with instant reload and quick builds.
* Vite helps us compile and bundle our React code efficiently.
* Out of the box support for React, and modern JavaScript features.

2.4 Node.js + Express (Server Side)

We used Node.js with Express to build the server-side logic and define RESTful APIs.

* Express allows us to define routes that handle client requests (e.g., fetching reports, creating classes, updating meetings).
* The server communicates with MongoDB Atlas to store and retrieve data.

2.5 MongoDB Atlas (Database)

Our project uses MongoDB Atlas, a cloud-hosted NoSQL database that stores data in JSON format.

* Database name: modular\_skills
* The database includes five main collections:

● classes

* Contains class information: classCode, className, subject, situation, question, createdBy, and createdAt.
* Each class includes a students[] array with:
  + studentId
  + answerText – the student's response to the simulation
  + analysisResult – AI-generated analysis with CASEL 5 scores, strengths, weaknesses, overall score, and recommendations

● lecturers

* Lecturer data: username, id, email, password, profilePic

● students

* Basic student information: username, id, email, password, profilePic

● notifications

* Notifications for lecturers
* Fields: teacherId, type, title, read, createdAt, updatedAt

● studentnotifications

* Personal notifications for students
* Fields: studentId, type, title, content, read, createdAt, updatedAt

2.6 Vercel Deployment (Monorepo)

Our entire application was deployed using Vercel, with both frontend and backend hosted in the same monorepo.

* The project is structured into frontend and api folders.
* Server routes are deployed as Serverless Functions.
* The vercel.json file defines deployment settings.
* Every push to Git automatically triggers a Vercel deployment.

3. Application Flow

3.1 React SPA Structure

תמונה שמכילה טקסט, צילום מסך, קו, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

This diagram illustrates the flow of our Single Page Application (SPA).  
The browser initially loads a single index.html file containing an empty <div id="root">.  
From there, main.jsx renders the App component (from App.jsx) into this root div.  
The App component uses React Router to display different components dynamically based on the current URL path - without reloading the entire page.  
When users navigate, React efficiently replaces the component inside the root, creating a seamless, fast user experience typical of SPA architecture.

3.2 App.jsx (Routing & Structure)

The App.jsx file serves as the core routing hub of the application. Using react-router-dom, it defines all URL routes in the Single Page Application (SPA) and dynamically loads the correct component based on the current path – without reloading the page.

The entire app is wrapped with several context providers:

* ThemeProvider – manages light/dark mode across all pages
* UserProvider – provides the current user’s authentication context
* NotificationsProvider – manages alerts for lecturers
* StudentNotificationsProvider – manages alerts for students

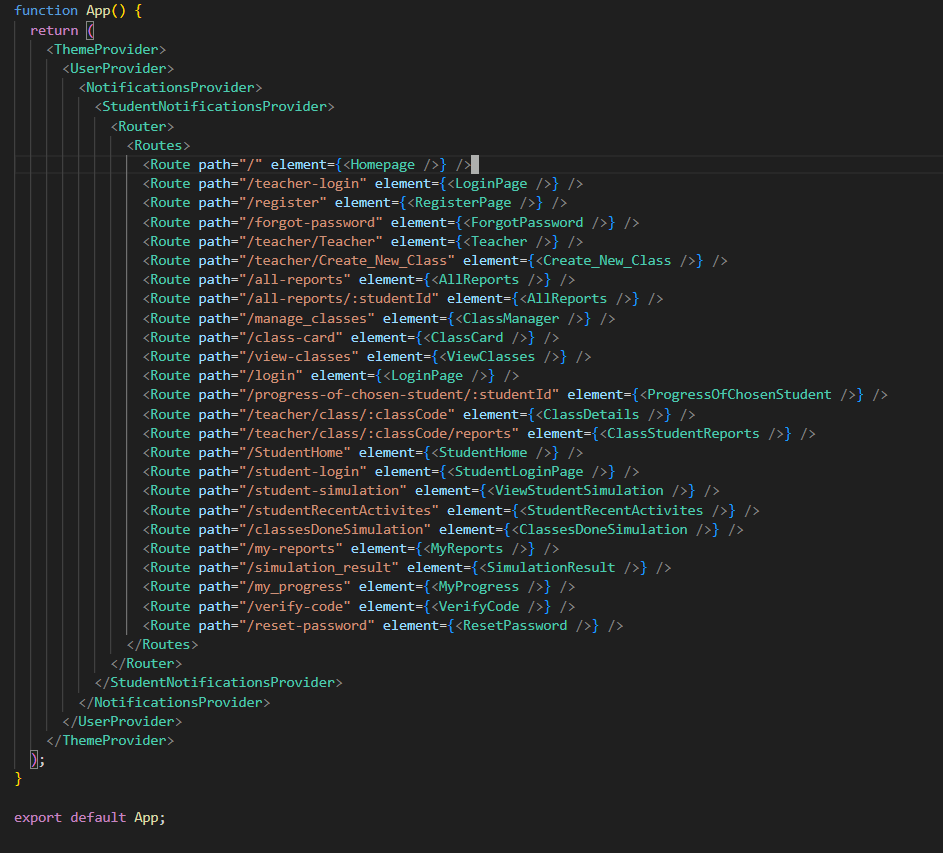
Each route corresponds to a specific component:

* /register renders the RegisterPage component
* /student-login renders the StudentLoginPage
* /all-reports/:studentId displays the detailed progress reports for a specific student

This routing logic ensures smooth transitions, preserves application state, and provides a fast user experience – a hallmark of SPA architecture.

Route Configuration in App.jsx

The following screenshot from App.jsx shows the full list of routes and their corresponding components. It visually represents how the SPA navigates without full page reloads:



Special Libraries

* import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'

react-router-dom is the library that handles routing inside the application.  
It allows us to define URL paths and specify which component should be displayed for each path.  
Thanks to this, our app is a SPA — navigation between pages happens without reloading the entire page from the server.

* import { StrictMode } from 'react'

**StrictMode** is a React tool that helps identify potential problems in the code during development.  
It does not affect how the app looks or works, but it displays warnings in the console if something is written incorrectly or in an outdated way.

* import { createRoot } from 'react-dom/client'

**createRoot** is used to create the root of the app — the place where React injects all components into the browser’s DOM (inside the <div id="root">).

* import React, { useState, useEffect, useRef, useContext } from 'react';
* React: Main library for building user interfaces.
* useState: Stores dynamic values in a component.
* useEffect: Runs side effects (e.g., data fetching) on component load or update.
* useRef: Holds mutable values that don’t trigger re-renders. Often used to reference DOM elements.
* import html2canvas from 'html2canvas';
* Captures any visible part of the app (like charts or cards) as a canvas image.
* Typically used as a first step before converting UI to a PDF.
* import { jsPDF } from 'jspdf';
* Creates downloadable PDF files directly in the browser.
* Often used with html2canvas to turn screen elements into PDF documents.
* const express = require('express');
* Web framework for Node.js to build server-side routes and APIs.
* Used to define endpoints like /login, /api/student-progress, etc.
* const mongoose = require('mongoose');
* Connects to MongoDB and defines data schemas like User, Class, or Notification.
* Enables querying and updating data in MongoDB from the Node.js backend.
* const nodemailer = require('nodemailer');
* Sends emails from the backend, such as OTPs (one-time passwords) or password reset links.
* Uses services like Gmail or SMTP configurations.
* const dotenv = require('dotenv');
  + Loads environment variables from a .env file (like database URIs or API keys).
  + Keeps sensitive data secure and out of the source code.
* const axios = require('axios');
* Sends HTTP requests (GET, POST, etc.) to external APIs.
* it's used for communication with Claude AI, Firebase, or other services.

Components in Our Project

4.1.0 LoginPage.jsx (Lecturer login)

Functionality

Login page for lecturers – authenticates teacher credentials and navigates to the dashboard.

Functional Description

This component renders a styled login form specifically for teachers. It includes:

* Form fields for ID and Password
* Error handling and validation for empty fields
* Integration with backend API at /api/teachers/login
* On successful login: stores teacher info in UserContext and redirects to /teacher/Teacher
* Theme adaptation (dark/light) using ThemeProvider and ThemeContext
* Includes reusable components: FormInput, Button, Alert, SharedHeader, and Footer

Parameters

None (does not accept props).  
Uses:

* UserContext – to access the login() method
* ThemeContext – to determine theme mode
* React Router’s useNavigate() – to redirect after login

Returns

A full login page with:

* Header and footer
* Centered login form with validation
* Navigation links for “Forgot password” and “Register”
* Responsive and theme-aware styling

Used In

* Route: /login?role=teacher
* Called when a lecturer signs in to the system

5.1.1 TeacherHeader.jsx

Functionality

Main header and navigation bar for the teacher dashboard.

Functional Description

This component displays the main navigation links for teachers (Home, Classes, Reports, Create Class), a theme toggle button, a real-time notification bell with dropdown, and user profile information. It also supports logout and adapts its design to light/dark themes using context.

Parameters

This is a functional component — it does not receive props. It uses multiple contexts:

* UserContext – to get the teacher info
* NotificationsContext – to manage and fetch notifications
* ThemeContext – to handle dark/light mode

Returns

A fully responsive <header> element that includes:

* Navigation bar for teacher features
* Notifications system (badge + dropdown)
* Theme toggle
* Profile avatar
* Logout button

Used In

* Teacher dashboard pages
* All pages that require teacher level navigation

4.1.2 Teacher.jsx

Functionality

Main teacher dashboard page, combining navigation, summary, recent activity, and AI assistant.

Functional Description

This component serves as the main dashboard view for teachers. It:

* Greets the logged-in teacher by name
* Displays dashboard summary metrics (DashboardOverview)
* Offers quick navigation options via QuickActions
* Shows recent student activity (RecentActivity)
* Embeds a floating AI assistant (AIChat)
* Wraps all content inside a theme-aware layout (ThemeProvider)

Parameters

This component does not accept props directly.  
Instead, it relies on:

* UserContext – to get the current teacher
* ThemeContext – for dark/light mode settings

Returns

A full-page teacher dashboard including:

* Header (TeacherHeader)
* Personalized welcome message
* Dashboard sections: overview, actions, activity
* AI Assistant button (if logged in)
* Footer

Used In

* This is the entry point for all teacher dashboard functionality

4.1.3 QuickActions.jsx

Functionality

Displays a set of quick action cards for teachers on their dashboard.

Functional Description

The component shows three interactive cards, each representing a key teacher action:

1. Create New Class – opens the class creation form.
2. Manage My Classes – navigates to view and manage existing classes.
3. Student Progress – leads to the full report page showing student performance.

Each card includes:

* An icon
* A short description
* A button that routes to the related feature

The layout is responsive (single column on mobile, 3-column on desktop).

Parameters

None. The component defines a local array (actions) with icon, title, description, and link for each card.

Returns

A responsive <div> element containing 3 stylized cards, each with its own title, description, icon, and button.

Used In

* Teacher.jsx – under the "Quick Actions" section on the teacher dashboard

4.1.4 RecentActivity.jsx

Functionality

Displays the most recent teacher notifications as a summary of recent activities.

Functional Description

This component shows the 3 latest notifications relevant to the teacher, such as exam events, messages, or schedule changes. It:

* Fetches notifications using useNotifications() on initial render (useEffect)
* Sorts them by creation time (createdAt)
* Displays a styled list with icons, types, titles, and timestamps

It uses conditional rendering: if there are no notifications, it shows a fallback message.

Parameters

None (this is a self-contained component).  
Internally uses:

* useNotifications() – to get and fetch notifications

Returns

A styled container (div) with:

* Title: "Recent Activities"
* A list of up to 3 most recent notifications
* For each notification: icon, title, date

Used In

* Teacher.jsx – in the dashboard page, under the "Recent Activities" section

4.1.5 DashboardOverview.jsx

Functionality

Displays key summary statistics on the teacher's dashboard.

Functional Description

This component fetches and displays a summary of the teacher's data from the backend API. It shows:

* The number of active classes
* The total number of students
* The most frequently taught topic

It uses UserContext to get the teacher’s ID and sends a request to /api/teacher/:id/summary.  
A loading message is shown until the data is retrieved.  
Each stat is presented as a card with an icon, label, and value, using a responsive layout.

Parameters

None (does not accept props).  
Uses:

* UserContext for the teacher's ID
* Internal state (useState) to manage summary data and loading

Returns

A responsive grid of three cards displaying:

* Active Classes
* Total Students
* Most Frequent Topic

Each card includes an icon, label, and value.

Used In

* Teacher.jsx – inside the main dashboard content section

4.1.6 Create\_New\_Class.jsx

Functionality

Allows the teacher to create a new class with AI-generated scenario-based questions.

Functional Description

This component is the class creation interface for teachers. It includes:

* Page header and welcome
* Informational popup ("How it works") explaining the AI-based class creation
* Integration with ClassForm for collecting class details
* AI-generated situational question explanation based on the CASEL 5 framework
* Responsive layout with theme support (dark/light)
* Floating AIChat assistant
* Uses state (useState) to toggle info popup
* Uses UserContext and ThemeContext

Parameters

None (no props).  
Internally uses:

* ThemeContext – for color mode
* UserContext – to access teacher info
* Local state: showInfo boolean

Returns

A full-page layout including:

* TeacherHeader
* Title and explanatory paragraph
* Button to show popup info on "How it works"
* ClassForm for actual input
* AIChat assistant
* Footer

Used In

* Route: /teacher/Create\_New\_Class
* Used when a teacher wants to create a new class

4.1.7 ClassForm.jsx

Functionality

Multi-step form that allows a teacher to create a new class with an AI-generated scenario and question.

Functional Description

This component implements a two-step class creation process:

1. Step 1 – Class Details: The teacher enters a class code, class name, and either selects or writes a topic.
2. Step 2 – AI Generation & Approval: The system uses AI to generate a situation and question related to the topic. The teacher can regenerate or approve and create the class.

Features include:

* Topic selection or custom topic input
* Integration with Claude API for AI generation
* Class creation request to backend
* Notifications on success/failure
* Input validation and error handling
* Toast messages for feedback
* Theme support (dark/light)
* Progress bar and step management

Parameters

None (does not accept props).  
Uses:

* UserContext – for teacher ID
* NotificationsContext – to refresh notifications
* Internal state (useState) for form data, step, errors, loading, situation/question

Returns

A two-step responsive UI:

* Step 1: Form with inputs for code, name, and topic
* Step 2: Display of generated situation/question and final confirmation buttons
* Toast feedback on top-right
* Reusable LoadingSpinner component

Used In

* Create\_New\_Class.jsx – renders this form inside the class creation page

4.1.8 ViewClasses.jsx

Functionality

A simple wrapper that renders the ClassManager component.

Functional Description

This component acts as a clean routing endpoint. It doesn’t contain any logic, props, or state of its own.  
Its purpose is to connect the route /view-classes with the ClassManager interface.

Parameters

None

Returns

* Renders: <ClassManager />

Used In

* Route: /view-classes
* Used when a teacher wants to view and manage all their classes

4.1.9 ClassManager.jsx

Functionality

Displays, filters, and manages the list of classes created by the lecturer.

Functional Description

The ClassManager component is the central hub for teachers to view and manage their classes.  
It fetches class data from the backend using the current teacher's ID, allows filtering by name and class code, and renders the classes as individual ClassCards.  
Also includes a button to create a new class and integrates the AI assistant and theme support.

Parameters

None (Relies on global UserContext and ThemeContext)

Returns

* Renders:
  + TeacherHeader, Footer
  + A search + filter interface
  + A list of ClassCards based on fetched and filtered classes
  + A link to /teacher/Create\_New\_Class
  + The AIChat button for lecturers

Used In

* Route: /manage\_classes
* Used when a teacher wants to view, filter, and manage their classes and student participation

4.1.10 ClassCard.jsx

Functionality  
Displays a summary card for each class, showing key statistics and action buttons for the lecturer.

Functional Description  
The ClassCard component presents an individual class in a visually distinct card format. It displays the class code, name, subject, and creation date. It also calculates and shows:

* The number of unique students in the class
* The number of simulation attempts
* The average SEL score (out of 5), with a visual progress bar

Additionally, it includes three main actions:

1. View Details – navigates to the class detail view
2. Student Reports – navigates to the reports page
3. Delete – opens a confirmation modal to delete the class, and if confirmed, triggers an API call and calls onDeleteSuccess.

The component handles edge cases such as missing scores or empty student lists and visually adapts to light/dark mode.

Parameters  
• classData: an object containing the full details of the class  
• onDeleteSuccess: an optional callback function to handle UI updates after successful deletion

Returns  
Renders:  
• A styled card containing the class information  
• Simulation statistics (attempts, SEL score)  
• Action buttons for view, reports, and delete  
• A modal confirmation before deletion

Used In  
• Used inside ClassManager.jsx  
• Appears in the route /manage\_classes as part of the class list displayed to the lecturer

4.1.11 ConfirmModal.jsx

Functionality  
Displays a modal dialog for confirming critical actions – primarily deletion.

Functional Description  
The ConfirmModal component renders a popup window with a customizable title, description, and action buttons (Confirm and Cancel).  
It supports a loading state (isProcessing) to indicate ongoing actions (like deletion).  
The modal only appears if isOpen is true. It triggers the provided callback functions (onConfirm, onCancel) when buttons are clicked.

Parameters  
• title: Modal title (default: "Are you sure?")  
• description: Additional description (optional)  
• confirmText: Confirm button text (default: "Confirm")  
• cancelText: Cancel button text (default: "Cancel")  
• isOpen: Whether the modal is open (Boolean)  
• isProcessing: Whether to show a loading state (Boolean)  
• onConfirm: Callback function triggered on confirmation  
• onCancel: Callback function triggered on cancel

Returns  
• Renders a styled modal that includes:  
• A title and optional description  
• Cancel and Confirm buttons  
• Optional loading indicator  
• If isOpen is false – renders nothing.

Used In  
• Used inside ClassCard.jsx  
• Triggered when a lecturer attempts to delete a class

4.1.12 ClassStudentReports.jsx

Functionality  
Displays all student report cards for a specific class based on the class code.

Functional Description  
This component fetches class data using the classCode from the URL. It groups student submissions by studentId and renders a StudentReportCard for each student who has analysis data. Includes header, summary, AI chat button, and footer.

Parameters  
None (relies on ThemeContext and UserContext).

Returns  
• TeacherHeader  
• Class and student summary info  
• List of StudentReportCards  
• AIChat button  
• Footer

Used In  
• Route: /teacher/reports/:classCode

4.1.13 StudentReportCard.jsx

Functionality  
Displays a detailed report card for an individual student, including profile, simulation attempts, and export option.

Functional Description  
Receives all simulation attempts for one student, shows their name, ID, and profile picture. Renders a SimulationChart for each attempt and includes an export-to-PDF button. Uses default image if profilePic is missing.

Parameters  
• studentGroup – an array of all simulations for the student

Returns  
• StudentHeader  
• Multiple SimulationChart components  
• ExportButton for downloading the report

Used In  
• ClassStudentReports.jsx

4.1.14 SimulationChart.jsx

Functionality  
Renders a bar chart for a single simulation attempt, showing five SEL competencies.

Functional Description  
Displays the index and overall score of a simulation. Uses Recharts to draw a responsive bar chart with colors and tooltips. Adapts to dark/light mode.

Parameters  
• simulation – simulation attempt object  
• index – index of the simulation  
• isDark – boolean flag for theme

Returns  
• Simulation header  
• Recharts BarChart with 5 competency scores

Used In  
• StudentReportCard.jsx

4.1.15 StudentHeader.jsx

Functionality

Displays a student’s profile picture, name, student ID, and number of completed simulations. Used as a header section in student report cards or similar views.

Functional Description

The StudentHeader component shows a summary block with:

* Profile image (with fallback on load error).
* Username (or default fallback if missing).
* Student ID.
* Count of completed simulations (singular/plural handled).  
  It dynamically adapts styles based on the isDark flag for dark/light mode.

Parameters

* profilePic: URL of the student’s profile picture.
* onImageError: Callback if image loading fails.
* username: Student’s name.
* studentId: Unique identifier of the student.
* simulationCount: Number of completed simulations.
* isDark: Boolean indicating whether dark mode is active.

Returns

JSX block showing the student’s profile image, name, ID, and simulations summary.

Used In

* StudentReportCard
* Report or progress-related views

4.1.16 ExportButton.jsx

Functionality

Renders a button used for exporting content (typically as a PDF). The component relies entirely on props for its behavior and appearance.

Functional Description

The ExportButton component:

* Triggers onExport when clicked.
* Shows a different label while exporting (Exporting... vs Export PDF).
* Disables itself when isExporting is true to prevent multiple triggers.
* Adapts styling based on the current theme (isDark).

Parameters

* onExport: Function to call when the button is clicked.
* isExporting: Boolean that disables the button and changes label during export.
* isDark: Boolean for applying dark mode styles.

Returns

A styled <button> element that initiates an export operation and reflects the export status.

Used In

* Detailed student reports
* Summary views or dashboards that support PDF export

4.1.17 ClassDetails.jsx

Functionality

Displays all details for a selected class, based on its code. Shows the simulation situation, a button to get AI insight, and all student answers with SEL analysis.

Functional Description

* Retrieves classCode from useParams and fetches the class details from the server.
* Allows the teacher to request AI-generated insight about the class.
* Displays student answers using StudentAnswerCard.
* Uses ThemeContext to apply dark/light mode.
* Combines multiple components: TeacherHeader, Footer, SimulationBox, AIChat.

Parameters

* No direct props – uses context and URL parameters.

Returns

A full class details view including simulation, question, AI insights, and all student responses.

Used In

* Detailed teacher view of a specific class and its students’ answers.

4.1.18 SimulationBox.jsx

Functionality

Displays the simulation's situation and question, and provides a button to request AI-generated class insight.

Functional Description

* Renders situation text and main question.
* Provides a button that triggers onGetClassInsight when clicked.
* While loading, disables the button and shows a spinner.
* Adjusts styles for dark/light themes.

Parameters

* simulationText: Main question for the simulation.
* situation: Background context of the simulation.
* onGetClassInsight: Function to fetch AI insight.

Returns

A styled box containing simulation details and a button to get AI insight.

Used In

* Displayed inside the ClassDetails view.

4.1.19 StudentAnswerCard.jsx

Functionality

Displays a student's answer with full SEL analysis including strengths, improvement areas, AI recommendation, and estimated depth level.

Functional Description

* Accepts a student answer and renders all analysis visually.
* Fetches missing student data (e.g. username, profile pic).
* Shows profile image, student ID, submission date, and overall score.
* Renders SEL category scores with color indicators and bars.
* Displays suggested intervention and depth level (if available).

Parameters

* answer: Student answer object with analysis.
* student: Student details (optional).
* isDark: Dark mode toggle.

Returns

A detailed card with the student’s response and AI-driven SEL evaluation.

Used In

* Rendered as part of the class detail page to evaluate each student's progress.

• 4.1.20 Student Progress for Teacher Components

Components designed for lecturers to monitor and analyze student progress. These include tools for filtering, reviewing, and visualizing student performance. Each component focuses on a specific aspect of progress management and reporting.

• 4.1.20.1 AllReports.jsx

Functionality  
Displays the progress of all students assigned to the logged-in lecturer, with the ability to search by student ID.

Functional Description  
The AllReports component fetches all students who participated in the lecturer's classes (based on user.id from context), allows filtering by student ID, and displays summary cards for each student using the StudentCard component. It also supports loading states, dark/light mode, and includes an AIChat assistant tied to the lecturer's ID.

Parameters  
None (Relies on UserContext and ThemeContext)

Returns

* TeacherHeader, Footer
* Search field for Student ID
* Loading indicator / No results message
* Grid layout of student cards (StudentCard)
* AIChat button based on user.id

Used In

* Route: /all-reports
* Used as the main teacher page for viewing all students across their classes

• 4.1.20.2 ClassProgressCard.jsx

Functionality  
Displays a student’s progress in a specific course, including improvement chart, depth level, CASEL domain scores, and latest submission date.

Functional Description  
The ClassProgressCard component receives course data and the student’s attempts. It calculates improvement between the first and last attempts, shows a line chart of overall scores if multiple attempts exist, and displays the depth level (Advanced/Intermediate/Basic) of the latest attempt. Below the chart, it renders the CASEL scores across five competencies. It also shows the submission date and the total number of attempts.

Parameters

* classData: An object containing course details and student attempts
* isDark: Boolean indicating whether dark mode is active

Returns

* Course title and subject
* Score improvement between first and last attempts
* Progress line chart (if more than one attempt)
* CASEL competency scores
* Depth level, last submission date, and total attempts count

Used In

* Inside StudentCard to display individual student progress per class

• 4.1.20.3 pdfExporter.js

Functionality  
Exports a DOM element as an A4-sized PDF file, including automatic scaling and multi-page handling.

Functional Description  
The exportElementAsPDF function uses the html2canvas and jsPDF libraries to capture a selected DOM area (elementRef) as a canvas, convert it to an image, and embed it into a PDF file.  
If the content height exceeds one page, the image is split into multiple pages while preserving scale.  
A 0.5-second delay is added before capturing to ensure the element has finished rendering.

Parameters

* elementRef – The DOM element to export as PDF
* filename – The name of the output file (default: 'report.pdf')

Returns

* A downloadable PDF file
* Console error messages in case of failure

Used In

* Export PDF buttons in student report components
* Components that use ref to capture and export content as PDF

• 4.1.20.4 ProgressOfChosenStudent.jsx

Functionality  
Displays the full progress of a selected student, including all enrolled courses, simulation attempts, progress over time, SEL domain scores, and the depth level of the last attempt.

Functional Description  
The ProgressOfChosenStudent component retrieves the studentId from the route, fetches data from /api/teacher/:teacherId/student/:studentId/progress, and presents detailed analytics.  
It uses StudentOverviewHeader to show general info and ClassProgressCard to visualize per-class progress.  
Includes an Export PDF button to save the view as a printable file, and shows the AI chat for the teacher using teacherId.  
It supports loading, error states, and light/dark themes.

Parameters

* studentId via useParams()
* location.state.student – optional navigation state
* user, theme – from context

Returns

* Student summary header (StudentOverviewHeader)
* Class progress charts (ClassProgressCard)
* Export PDF button
* AI teacher chat

Used In

* Route: /all-reports/:studentId
* Detailed student progress report from teacher's side

• 4.1.20.5 StudentCard.jsx

Functionality  
Displays a student summary card showing their photo, name, ID, average score, number of unique simulations, total attempts, latest activity, and a button to navigate to full details.

Functional Description  
The StudentCard component receives a student object and displays a visual card. It includes the student's image (with a fallback avatar), name and ID, statistics, and average score.  
Clicking View Details navigates to the detailed progress page (/progress-of-chosen-student/:id) and passes the full student object via route state.  
The card adapts to light/dark themes and logs the student state for debugging.

Parameters

* student – Object containing:
  + id, username, profilePic, averageScore, uniqueSimulations, totalAttempts, latestActivity, overallScore

Returns

* A visual card presenting the student's basic data
* A button to navigate to detailed student progress

Used In

* AllReports.jsx – to render all student summaries
* Can be reused in class-level or teacher views

• 4.1.20.6 StudentOverviewHeader.jsx

Functionality  
Displays the top section of a student's detailed progress report, including their profile image, name, number of enrolled classes, and stats for total attempts, unique simulations, and average score.

Functional Description  
The StudentOverviewHeader component receives a student object and renders a visual summary.  
It shows:

* Profile image (with fallback handling)
* Full name and student ID
* Number of enrolled classes
* Key statistics: total submissions, unique simulations, and average score  
  Supports dark/light mode via prop or context.

Parameters

* student: object containing
  + id, username, profilePic, classes, totalAttempts, uniqueSimulations, averageScore
* isDark (optional): boolean indicating dark mode preference

Returns

* A styled visual header summarizing student identity and performance

Used In

* ProgressOfChosenStudent.jsx – as the top header for an individual student's progress report

• 4.1.21 AIChat.jsx

Functionality  
Provides a floating AI-powered chat assistant for lecturers to ask questions related to students, classes, or educational topics.

Functional Description  
The AIChat component renders a floating button that opens a chat box.  
It supports sending user messages to an AI backend and displaying AI responses.  
Features include:  
• Toggleable chat window  
• Persistent message state with user and AI roles  
• Auto-scroll to latest message  
• Loading indicator with typing animation  
• Dark/light theme support via context

Parameters  
• teacherId: string identifying the lecturer (sent with API requests)

Returns  
• A floating button and an optional expandable chat window with input, message history, and AI interaction

Used In  
• Displayed globally on the teacher dashboard for real-time AI help

• 4.1.22 NotificationsContext.jsx

Functionality  
Manages notification state for lecturers across the app, including fetching, counting unread items, and marking as read.

Functional Description  
The NotificationsContext provides centralized state management for all teacher notifications.  
It includes:  
• Automatic fetching on user ID change  
• Storing all notifications in state  
• Counting unread notifications  
• Marking individual or all notifications as read  
• Exposing state and actions via a custom hook

Parameters  
• None (internally accesses the current user via UserContext)

Returns  
• Context values:  
– notifications: array of notification objects  
– notificationCount: number of unread notifications  
– fetchNotifications(): function to refetch data  
– markNotificationAsRead(id): marks one as read  
– markAllAsRead(): marks all as read  
– isLoading, error: status flags

Used In  
• TeacherHeader.jsx – to show the notification bell and unread count  
• Other lecturer components requiring real-time notification updates

• 4.2.1 StudentHome.jsx  
Functionality  
Displays the main home page for a logged-in student, showing a personalized welcome, their enrolled classes, recent activities, and a floating AI chat button.

Functional Description  
The component uses ThemeContext to dynamically apply light or dark styling.  
It uses UserContext to fetch and display the logged-in student’s name and ID.  
The page includes:

* StudentHeader: the student's page header
* ShowClasses: lists the student’s enrolled classes
* StudentRecentActivities: displays their latest actions
* StudentAIChat: floating AI chat assistant (shown only if user is logged in)
* Footer: bottom section

Parameters

* No external props. Uses internal contexts: UserContext, ThemeContext.

Returns

* A full student homepage including header, content, and footer.

Used In

* Used as the main page rendered after student login.

4.2.2 StudentLogin.jsx (Student login)  
Functionality  
A login page for students, featuring a form to input student ID and password, error handling, and links to register or reset password.

Functional Description  
The component displays a centered login form with responsive styling and theme support via ThemeProvider.  
When the form is submitted, it sends a request to the backend /api/students/login endpoint.  
On successful login, the student is stored in UserContext and redirected to their home page.

The component includes:

* Usage of UserContext for login handling
* Usage of ThemeContext for dark/light mode styling
* Utilizes FormInput, Button, and Alert components
* Includes links to registration and password reset

Parameters

* No external props. Uses internal state and context (useState, useContext).

Returns

* A full login page layout with form, links, and footer.

Used In

* Main login page for students accessing the platform.

4.2.3 StudentHeader.jsx

Functionality  
Displays the header for all student pages. Includes navigation links, a theme toggle button, a dynamic notification bell with dropdown, and a student profile section with logout option.

Functional Description  
This component provides the following features:

* Navigation bar with links to:
  + Student Home (/StudentHome)
  + Completed Simulations (/classesDoneSimulation)
  + My Progress (/my\_progress)
* Theme toggle button to switch between light and dark mode using ThemeContext.
* Notification bell:
  + Shows the count of unread notifications using StudentNotificationsContext
  + Animates when there are unread notifications
  + Dropdown panel listing notifications (loaded dynamically)
  + Each notification shows a type-based icon and timestamp
  + Allows marking a single notification or all notifications as read
* Profile section:
  + Shows student’s name and role (Student)
  + Displays profile picture (uses default icon if missing)
* Logout button: Redirects to /student-login

Parameters  
• No external props. Uses internal contexts:

* ThemeContext – to determine dark/light styling
* UserContext – to get student data
* StudentNotificationsContext – to manage notification logic

Returns  
• A fully functional and styled header with navigation, theme toggle, and notification features.

Used In  
• Used in all major student views including StudentHome, MyProgress, StudentSimulation, etc.

4.2.4 MyProgress.jsx

Functionality  
Displays a detailed progress dashboard for the student, including simulation statistics, SEL skill averages, interactive graphs, a PDF export button, and summaries of recent attempts.

Functional Description  
The component uses UserContext to retrieve student data and ThemeContext to adjust theme styling (dark/light).  
It includes:  
• StudentHeader – displays the student’s name and ID  
• Statistics card – total attempts, unique simulations, average score  
• Bar Chart – visualizes average score per SEL skill  
• Line Chart – shows skill trends across all submissions  
• Simulation cards – show latest responses, first vs. latest scores, and full CASEL breakdown  
• PDF Export Button – allows exporting the dashboard as a PDF report  
• StudentAIChat – floating AI assistant for support  
• Footer – bottom of the page

Parameters  
• No external props  
• Uses UserContext, ThemeContext, and StudentNotificationsContext  
• Fetches data from /api/classes/get-classes-done-simulation/:id

Returns  
• A full progress dashboard with visual charts, simulations breakdown, export features, and interactive help

Used In  
• Serves as the student’s personal progress page, accessible from the student home or navigation

4.2.5 MyReports.jsx

Functionality  
Displays a student's personal reports for a selected class, including simulation details, date filtering, and submitted answers.

Functional Description  
This component uses ThemeContext for styling and UserContext to identify the logged-in student.  
It fetches the class data from the server based on the classCode, and includes:  
• StudentHeader – top header with student info  
• StudentSimulationBox – shows the simulation situation and question  
• Date filter input to show submissions by selected date  
• AnswerCard – cards showing the student’s responses and analysis  
• StudentAIChat – floating AI assistant button  
• Footer – page bottom section

Parameters  
• No external props.  
• Uses UserContext, ThemeContext, useLocation, and backend API to get class data by class code.

Returns  
• A full student report page showing simulation info, answer cards, and date filtering. Includes header, AI support, and footer.

Used In  
• Used in the My Reports view to show a student's submissions for a selected class.

4.2.6 ShowClasses.jsx

Functionality  
Displays a list of available classes for the student, with search filters by name or code, and allows starting a simulation for a selected class.

Functional Description  
This component uses ThemeContext for dark/light theme support and UserContext to identify the current logged-in student.  
It fetches class data from the server (/api/classes/get-all-classes) and normalizes it for display.  
The page includes:  
• Input fields for filtering by class name and code  
• Class cards showing name, code, subject, and a "Start Simulation" button  
• "Load More" button to reveal more classes  
• "Show Less" button to collapse the list back

Parameters  
• No external props  
• Uses UserContext, ThemeContext, and useNavigate for routing  
• Fetches data using fetch from the backend API

Returns  
• A responsive grid of available classes for the student, including filter functionality and navigation to simulations

Used In  
• Used inside the StudentHome.jsx page to display joinable classes for the student

4.2.7 StudentSimulation.jsx

Functionality  
Allows a student to participate in a selected simulation by reading the scenario, answering the question, and submitting their response for analysis and notification.

Functional Description  
The component uses UserContext and ThemeContext to render a personalized and themed experience.  
It performs the following:  
• Fetches simulation details (situation and question) using classCode  
• Displays a form to write and submit an answer  
• Submits the answer to the backend API  
• Creates a notification for the student after submission  
• Navigates to /simulation\_result page after successful submission  
• Displays a loading overlay during fetch or submit actions

Parameters  
• Receives from location.state:

* studentId – student identifier
* classCode – class code  
  • Uses context providers:
* UserContext
* ThemeContext
* StudentNotificationsContext

Returns  
• An interactive form to complete a simulation, including scenario, question, answer textarea, and a submit button with loading and success handling

Used In  
• Used when the student starts a simulation from the class list in ShowClasses.jsx

4.2.8 StudentSimulationBox.jsx

Functionality  
Displays the simulation scenario and the corresponding question in a clean, styled content box for the student to read.

Functional Description  
This component receives two props:  
• situation – the scenario description of the simulation  
• simulationText – the question the student must answer  
It renders both inside a styled container with light/dark mode compatibility for visual clarity.

Parameters  
• simulationText – the question text  
• situation – the scenario description  
(Both passed in as props from parent components)

Returns  
• A display box containing the scenario and question in a structured layout

Used In  
• Used in simulation-related pages like StudentSimulation.jsx and MyReports.jsx to show the core simulation content

4.2.9 AnswerCard.jsx

Functionality  
Displays a student’s submitted answer for a simulation, along with the overall score, full CASEL analysis, strengths, areas for improvement, suggested intervention, and estimated depth level.

Functional Description  
The component receives the answer object and conditionally renders it only if analysisResult exists.  
It includes:  
• Formatted submission date  
• Overall score with color indicators  
• The student’s answer text  
• CASEL analysis with scores, icons, feedback, and progress bars for 5 domains  
• Lists of strengths and areas for improvement  
• A suggested intervention section  
• Optional estimated depth level display

Parameters  
• answer – student answer object containing text, submission date, and analysis result  
• isDark – a boolean to indicate whether dark mode is active

Returns  
• A complete card view of a student’s submission, feedback, and analysis in a styled layout

Used In  
• Used in reporting page my-reports to display previously submitted simulations

4.2.10 ClassesDoneSimulation.jsx

Functionality  
Displays a list of classes in which the student has completed simulations, with search and filter options and navigation to detailed reports.

Functional Description  
This component uses UserContext to identify the logged-in student and ThemeContext to apply dark/light theming.  
It fetches data from /api/classes/get-classes-done-simulation/:userId and displays class cards.  
Includes:  
• Search input for class name  
• Filter input for exact class code  
• "View Details" button – navigates to /my-reports with the relevant classCode  
• StudentAIChat – floating AI assistant for support  
• Footer – bottom section of the page

Parameters  
• No external props  
• Uses UserContext, ThemeContext, and useNavigate  
• Fetches data from backend API using fetch

Returns  
• A dynamic student view listing past simulation classes with search/filter and report navigation

Used In  
• Used as a dedicated page for students to review completed simulations and access detailed reports

4.2.11 SimulationResult.jsx

Functionality  
Displays the result of a submitted simulation for the student, showing the full analysis and answer using the AnswerCard component. Handles loading and error states.

Functional Description  
This component uses UserContext to identify the student and ThemeContext for theme styling.  
It retrieves the student's answer for the given classCode (from location.state) using a backend API call.  
Includes:  
• StudentHeader – page header  
• API call to fetch the latest answer for the class  
• Displays AnswerCard with CASEL analysis if data is available  
• Error handling with return-to-dashboard button  
• Floating AI assistant (StudentAIChat)  
• Footer at the bottom of the page

Parameters  
• No external props  
• Uses classCode from location.state  
• Uses UserContext, ThemeContext

Returns  
• A complete result screen including the submitted answer, full analysis, and support UI

Used In  
• Redirect target after student submits a simulation in StudentSimulation.jsx

4.2.12 StudentRecentActivities.jsx

Functionality  
Displays the three most recent activities performed by the student, based on notification data, including timestamp, action type, and styled icons.

Functional Description  
This component uses StudentNotificationsContext to fetch and sort the student's notifications.  
It performs the following:  
• Fetches notifications using fetchNotifications()  
• Parses and sorts them by date (newest first)  
• Displays the top three most recent notifications  
• Applies color and icon styling based on notification type (submitted, exam, export)  
• Displays a fallback message if no activities exist

Parameters  
• No external props  
• Uses context: StudentNotificationsContext  
• Uses useLocation from React Router (not essential to logic)

Returns  
• A styled list of the latest three student activities with visual indicators and timestamps

Used In  
• Used in the student dashboard (StudentHome.jsx) to show recent actions like simulation submission, test alerts, or PDF export

4.2.13 StudentAIChat.jsx

Functionality  
A floating chat widget that allows the student to interact with an AI assistant. Provides personal support regarding simulations, classes, and performance.

Functional Description  
This component renders a circular floating button. Upon click, it opens a chat interface that includes:  
• A personalized AI greeting with the student's name  
• A scrollable message area with history  
• A typing indicator when the AI is generating a response  
• An input box and send button for the student  
• AI communication via POST to /api/claude/student-chat-insight with student ID and chat context  
• Dark/light theme support via ThemeContext

Parameters  
• studentId – student identifier for context-aware queries  
• studentName – used for the initial AI greeting

Returns  
• A responsive floating AI assistant UI with full message flow and input controls

Used In  
• Used across all student-facing pages (e.g., StudentHome, MyProgress, SimulationResult) to offer continuous AI-based support

4.2.14 StudentNotificationsContext.jsx

Functionality  
Manages the notification state for the student, including fetching notifications from the backend, tracking unread count, and marking notifications as read.

Functional Description  
This file defines and exports StudentNotificationsContext and its provider.  
It provides:  
• notifications – list of all student notifications  
• notificationCount – count of unread notifications  
• fetchNotifications() – fetches notifications for the current user  
• markNotificationAsRead(id) – marks a specific notification as read  
• markAllNotificationsAsRead() – marks all notifications as read for the current student  
• isLoading, error – loading and error indicators

The data is fetched automatically when the userId changes using useEffect.  
All children wrapped in the provider gain access to these values via React context.

Parameters  
• No external props  
• Uses UserContext to get the logged-in student’s ID  
• Accepts children – nested React components

Returns  
• A global provider that supplies notification data and actions to student-facing components

Used In  
• Used across the student interface in components like StudentHeader to provide real-time notification functionality

4.3.1 SharedHeader.jsx

Functionality  
Displays a consistent top header used across authentication pages. Includes logo, AI badge, theme toggle, and modals for features and SEL information.

Functional Description  
The component renders a styled header with:  
• An SVG icon and “Edu Map” branding  
• An “AI POWERED” badge with gradient background  
• A ThemeToggle button to switch light/dark modes  
• A FeaturesModal to show app capabilities  
• An AboutModal to display SEL-related information  
The theme is dynamically styled based on ThemeContext.

Parameters  
• No external props  
• Uses ThemeContext to determine current theme

Returns  
• A responsive header used across all auth pages, including branding, modals, and UI settings

Used In  
• Used in shared layout across all lecturer/student authentication pages:

* Login.jsx
* Register.jsx
* ResetPassword.jsx
* VerifyCode.jsx

4.3.2 Footer.jsx

Functionality  
Provides a consistent footer for all pages, displaying the current year and system name.

Functional Description  
The component renders a themed footer:  
• Displays the current year dynamically using JavaScript  
• Shows the app name: *Modular Skills Assessment Tool*  
• Styled using Tailwind to match the current light/dark theme

Parameters  
• No external props

Returns  
• A visually consistent, responsive footer bar across all pages

Used In  
• Used at the bottom of most pages, such as:

* StudentHome.jsx
* MyProgress.jsx
* Teacher.jsx
* Dashboard
* Any page with layout footer inclusion

4.3.3 Button.jsx

Functionality:  
A reusable button component that supports multiple states including loading, disabled, light/dark themes, and different design variants.

Functional Description:  
The button displays dynamic content using children, and shows a spinner with the text "Loading..." when in the loading state. It supports different styling variants: primary (blue), secondary (gray), and danger (red). You can make it full-width using the fullWidth prop. The button adapts to the current theme (light or dark) using the context.

Parameters:

* type (default: 'button') — the type of the button (button, submit, reset)
* onClick — function to execute when clicked
* isLoading — if true, shows a loading spinner and disables the button
* disabled — disables the button
* variant — determines the button's style ('primary', 'secondary', 'danger')
* fullWidth — if true, makes the button take full width
* className — custom CSS classes for additional styling

Returns:  
A styled <button> element with theme-aware design, click behavior, loading indicator, and proper accessibility and transitions.

Used In:  
This button is used across the system in login forms, registration, password reset, code verification, submitting forms, exporting reports, and more.

4.3.4 FormInput.jsx

Functionality:  
A reusable input field component for forms that supports labels, validation errors, and adapts to dark/light themes.

Functional Description:  
This component accepts props such as id, name, type, value, onChange, required, label, error, readOnly, and optional className.  
It renders a label if provided, an input field with styling based on the current theme (dark/light), and displays an error message below the field when needed.

Parameters:

* id, name – Input field identifier and name
* type – Field type (text, email, password, etc.)
* placeholder – Placeholder text inside the input
* value – Current value of the input
* onChange – Callback function for value changes
* required – Boolean to mark field as required
* label – Optional label text displayed above the input
* error – Optional error message
* readOnly – If true, disables editing
* className – Custom CSS classes

Returns:  
A <div> with a label, a styled <input>, and an error message (if any), styled to match the current theme.

Used In:  
Used in login, registration, reset password, and verification forms.

4.3.5 Alert.jsx

Functionality:  
Displays a styled alert box for messages like info, success, error, or warning. It shows an icon according to the type and can be dismissed with a close button.

Functional Description:  
This reusable alert component is used to notify users about system states or actions. It dynamically applies styles and icons based on the type prop (info, success, error, warning) and only renders if a message is provided. The onClose function allows dismissing the alert manually.

Parameters:

* type (string) – The style and icon to display (default: 'info').
* message (string) – The text content of the alert.
* onClose (function) – Callback when user clicks the close button.
* showIcon (boolean) – Whether to show the leading icon.
* className (string) – Additional styles.

Returns:  
A styled alert box containing the icon and message, optionally closable.

Used In:  
All pages that need user feedback after actions (form submissions, login errors, success messages, etc.).

4.3.6 ThemeToggle.jsx

Functionality:  
A toggle button that switches between light and dark mode. It shows a moon icon when the theme is light and a sun icon when the theme is dark.

Functional Description:  
This component lets the user change the UI theme. It reads the current theme from ThemeContext and calls toggleTheme() on click. It also adapts styling and icons based on the current mode.

Parameters:  
No props – context is used internally to access and modify theme state.

Returns:  
A styled circular button that changes the theme on click and updates the icon accordingly.

Used In:  
Header components, where users can switch between light and dark mode.

4.3.7 ThemeContext.jsx

Functionality:  
Provides and manages the global theme context (light/dark mode) across the entire application.

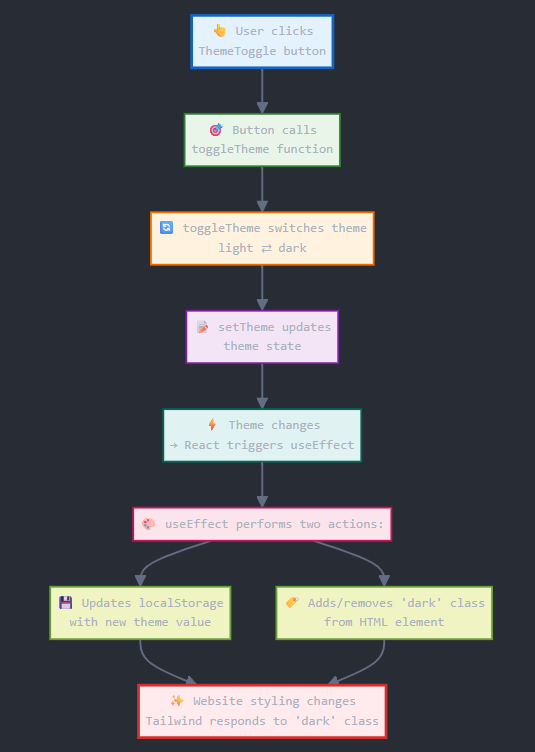
Functional Description:  
This file defines ThemeContext and a ThemeProvider component. It reads the initial theme from localStorage or system preferences, and updates both localStorage and the <html> class list on theme changes. The context includes the current theme and a toggleTheme() function.

Parameters:

* children (ReactNode) – Components that will receive the context.

Returns:  
A context provider (<ThemeContext.Provider>) that wraps the app and supplies theme and toggleTheme to consumers.

Used In:  
Wraps the app in App.jsx to give all components access to dark/light mode logic.



4.3.8 Homepage.jsx

Functionality:  
The homepage component renders the main landing interface, including a hero section, navigation buttons for students and lecturers, and a three-step visual walkthrough of the platform's process.

Functional Description:  
It includes a header, a vibrant introduction section with heading and subtitle, two login buttons, a feature walkthrough with text and images, and a footer. All styles adapt to light/dark themes.

Parameters:  
None – the component uses internal layout and styles.

Returns:  
A fully styled homepage layout with responsive design and theme awareness.

Used In:  
Rendered on the main root route (/) as the first view for users.

4.3.9 AboutSEL.jsx

Functionality:  
Displays a modal with information about SEL (Social and Emotional Learning) when the user clicks "About SEL".

Functional Description:  
This component controls a modal that explains SEL and lists the five core CASEL competencies. It supports dark/light themes and includes a button to close the modal.

Parameters:  
No props – it uses useState to manage modal visibility.

Returns:  
An interactive link and popup modal containing informative content about SEL.

Used In:  
Homepage.

4.3.10 FeatureCards.jsx

Renders a grid of feature cards, each displaying a title, icon, and description of one core functionality in the platform.

Functional Description:  
The component uses a static array of features and maps over it to generate styled cards, with light/dark theme support and responsive layout.

Parameters:  
None – features are hardcoded in the component.

Returns:  
A styled responsive grid of feature cards.

Used In:  
Homepage to introduce key benefits of the platform.

4.3.11 FeaturesModal.jsx

Functionality:  
A component that displays a "Features" link. When clicked, it opens a modal listing the key features of the platform.

Functional Description:  
The component uses internal state (useState) to control modal visibility. On click, a modal appears showing core features such as user management, PDF exports, interactive charts, class statistics, and dark mode support. The modal can be closed via a "Close" button.

Parameters:  
No props – data is hardcoded in the component.

Returns:  
An interactive link and a modal popup listing platform features.

Used In:  
Homepage.

4.3.12 UserContext.jsx

Functionality:  
Manages user state across the app. Stores the current user, provides login/logout functions, and saves the user in localStorage to persist between page reloads.

Functional Description:  
Defines UserContext and a UserProvider component that wraps the app and supplies user-related values. When a user logs in, their data is saved in localStorage; when they log out, it's removed. The context provides access to user, setUser, login(), and logout() functions for any component within the provider.

Parameters:  
• children (ReactNode) – Components that receive access to user context.

Returns:  
A context provider (<UserContext.Provider>) wrapping the app with user management logic.

Used In:  
The root of the app (App.jsx) to provide user data and actions globally.

4.4 Auth Components (Lecturer & Student)

4.4.1 RegisterPage.jsx

Functionality

A two-step registration form for both students and lecturers, supporting profile image upload and theming.

Functional Description

This component renders a full registration page with:  
• Step 1 – Username, ID, email, and password input fields  
• Step 2 – Profile picture upload with an option to skip  
• Theme support via ThemeContext (light/dark mode)  
• Sends registration data via POST to either /api/students/register or /api/teachers/register  
• Includes field validation (Gmail, password length, exact 9-digit ID)  
• Shows dynamic errors, loading spinner, and progress indicator

Parameters

• None explicitly passed – uses internal state and role from URL query string (role=teacher | student)

Returns

• A responsive, theme-aware registration UI with field validation, profile image preview, and complete registration flow

Used In

• Used on the registration route for both students and lecturers to create new user accounts in the system

4.4.2 ForgotPassword.jsx

Functionality

Password reset request form for both teachers and students. Validates the email and sends a reset code to the user.

Functional Description

This component renders a form that allows users to enter their email and request a password reset code. It includes:  
• Validation for proper email format  
• Sends a POST request to /api/students/forgot-password or /api/teachers/forgot-password based on role  
• Displays success or error messages  
• Redirects to the VerifyCode page with email and role in the URL if the request is successful  
• Light/dark mode support using ThemeContext

Parameters

• role – extracted from the URL (student or teacher)  
• email – entered by the user in the input field

Returns

• A complete email input form with validation, dynamic feedback, and redirect on success

Used In

• Used as the initial step in the password recovery process, before entering the verification code

4.4.3 ResetPassword.jsx

Functionality

A password reset page for both teachers and students, with new password input, validation, and light/dark theme support.

Functional Description

This component renders a secure password reset form after a user clicks a link with email and role in the URL. Features include:  
• Inputs for new password and confirmation  
• Validation to ensure both passwords match  
• POST request to the correct endpoint based on role: /api/teachers/reset-password or /api/students/reset-password  
• Displays success or error messages  
• Supports dark/light theme using ThemeContext  
• Redirects the user to the appropriate login page after success

Parameters

• email – extracted from the URL  
• role – user type (student or teacher) extracted from the URL

Returns

• A styled and responsive password reset UI with validation, alerts, and navigation

Used In

• Used in the password reset flow after the user verifies their email via a code

4.4.4 VerifyCode.jsx

Functionality

Verification form for a 6-digit code sent to the user's email as part of the password reset process. Supports both students and lecturers with light/dark theme support.

Functional Description

This component renders a secure form where the user enters the verification code received by email. Features include:  
• A read-only email input populated from URL query parameters  
• An input for the 6-digit verification code  
• Sends a POST request to the appropriate endpoint: /api/students/verify-code or /api/teachers/verify-code  
• Displays error messages if the code is incorrect  
• Redirects to the reset password page upon successful verification

Parameters

• email – extracted from the URL query  
• role – user type (student or teacher) from the URL

Returns

• A themed UI for entering a verification code with validation and redirect on success

Used In

• Used in the password recovery flow after the email step and before resetting the password

Hooks

6.1 React Built-in Hooks

useState

useState is a built-in React Hook that allows functional components to manage internal state. It returns an array with two elements:

1. The current state value
2. A function to update the state

When the update function is called, React re-renders the component with the new state value.

Code Example from Our Project

תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Explanation of the Example

In this code, we use multiple useState calls to manage the internal state of a multi-step AI simulation creation process:

* step: Tracks the current step (1 = details, 2 = AI generation, 3 = approval). Used to control the UI flow.
* isLoading: A boolean flag indicating whether a loading process is in progress (e.g., communicating with AI).
* loadingAction: Describes which specific action is currently loading (e.g., "Generating").
* situation: Holds the text of the situation generated by the AI.
* question: Holds the AI-generated question associated with the situation.
* generatedSituations: A counter of how many simulations have been generated so far.
* isTransitioning: Indicates if the component is transitioning between steps (can trigger UI animations or disable buttons).

Each state variable is independent, allowing precise control over the component's behavior and UI updates.

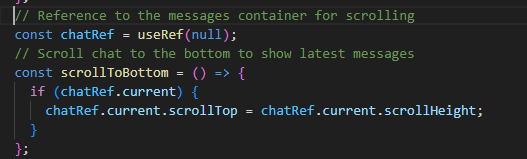
useRef

useRef is a built-in hook in React that returns a stable reference object which persists across renders. It is mainly used for two purposes:

1. Direct access to DOM elements – for example, to scroll, focus, or measure an element.
2. Storing mutable values that don’t cause a re-render when updated – such as timers, counters, or previous values.

The reference object holds a .current property, where the actual value is stored. Updating .current does not trigger a re-render of the component.

Example from Our Code – *AIChat.jsx*



Explanation of the Example

In this example, chatRef is used to keep a reference to the DOM element that displays the chat messages. Whenever a new message arrives from the user or the AI, the scrollToBottom function accesses the element using .current and scrolls it to the bottom.

Thanks to useRef, this behavior is handled without causing a re-render of the component – the scroll action is applied directly to the DOM.

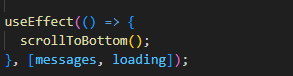
This creates a responsive chat experience where the user always sees the latest message without needing to scroll manually.

useEffect

useEffect is a built-in React hook that allows performing side effects in functional components, such as:

* Fetching data from an API
* Setting up timers
* Updating the DOM
* Synchronizing with external systems

Example from Our Code – *AIChat.jsx*



Explanation of the Example

In the AIChat component, we want the chat window to automatically scroll to the bottom whenever a new message is added or the loading state changes.

The useEffect hook listens for changes to the messages and loading state variables. When either changes, the scrollToBottom function is triggered, which scrolls the chat container to show the latest message.

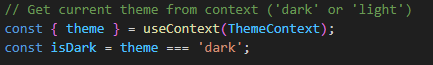
This creates a smooth user experience by ensuring the user always sees the most recent message without needing to scroll manually.

useContext

useContext is a built-in React hook that allows a component to access context values without passing props through every level of the component tree.

It's especially useful for global state like theme mode, authenticated user.

Example from Our Code – *AIChat.jsx*



Explanation of the Example

In the AIChat component, we want the chat box design to reflect the current theme (dark or light). Instead of passing the theme value as a prop, we directly access the global theme state using useContext.

This allows us to check isDark and apply conditional styles accordingly – for background, text, and borders – making the UI adaptive to the theme.

useParams

useParams is a React Router hook that lets you access dynamic route parameters from the URL inside a functional component.

It’s especially useful when you define routes like /class/:classCode and want to extract classCode.

It returns an object containing all parameters defined in the route path.

Example from Our Code – *ClassDetails.jsx*



Explanation of the Example

In ClassDetails, we want to display class information dynamically based on the class code in the URL (like /class/ABC123). By using useParams, we extract the value of classCode.

We then:

* Fetch the class data from the backend using the class code.
* Request an AI-generated insight using that code.  
  This makes the component dynamic and flexible for any class selected by the user.

useLocation

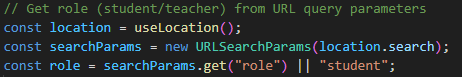
useLocation is a React Router hook that returns the current location object. This object contains information about the current URL — including the pathname, search (query string), and hash.

Basic usage:

const location = useLocation();

Useful when you need to read query parameters from the URL.

Example from your code (ForgotPasswordContent.jsx)



Explanation

You're using useLocation to access the ?role=... query parameter in the URL. This lets the component know whether the user is a teacher or a student, and behave accordingly (like choosing which API endpoint to call or which login page to return to). If the parameter is missing, it defaults to "student".

useNavigate

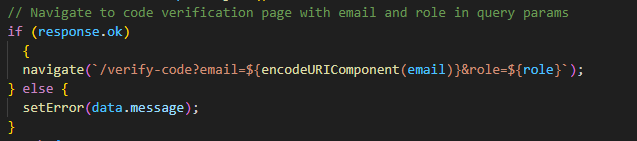
useNavigate is a React Router hook that returns a function used to programmatically navigate to another route.

Basic usage:

const navigate = useNavigate();

navigate("/new-page");

Example from your code (ForgotPasswordContent.jsx)



After a successful password reset request, this line redirects the user to the /verify-code page, while passing the email and role in the query string. This allows the next page to know who the user is and what type of account they're recovering.

6.2 Custom Hook for Notification Management– useNotifications

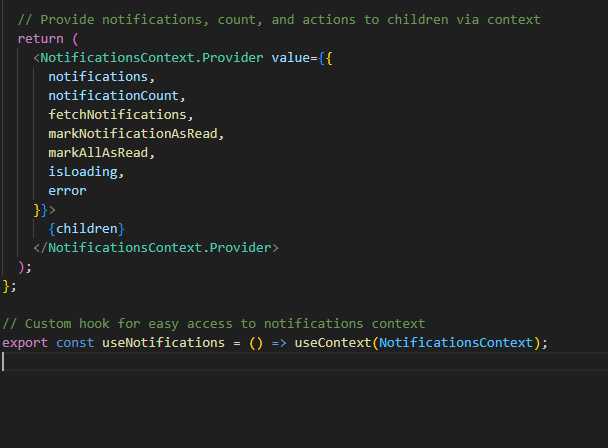
useNotifications is a custom React hook built on top of the React Context API.  
It provides an easy and centralized way for components to access all notification-related data and actions without prop drilling.

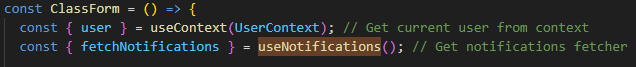
Using this hook, a component can:

* Access the list of all notifications (notifications)
* Know how many are unread (notificationCount)
* Fetch or refresh the list (fetchNotifications)
* Mark one as read (markNotificationAsRead)
* Mark all as read (markAllAsRead)
* Check if a request is loading (isLoading)
* Handle errors (error)

This hook consumes the NotificationsContext and returns the values from the provider.

Code Example:





Explanation in English – Using the Custom Hook useNotifications()

The useNotifications() is a custom React hook designed to provide easy access to the notifications state and actions via NotificationsContext. Instead of accessing the context manually in every component, this hook wraps useContext(NotificationsContext) and gives direct access to properties like notifications, notificationCount, and functions such as fetchNotifications. In the ClassForm component, for instance, useNotifications() is used to extract fetchNotifications, abstracting away the implementation details of how notifications are managed internally.

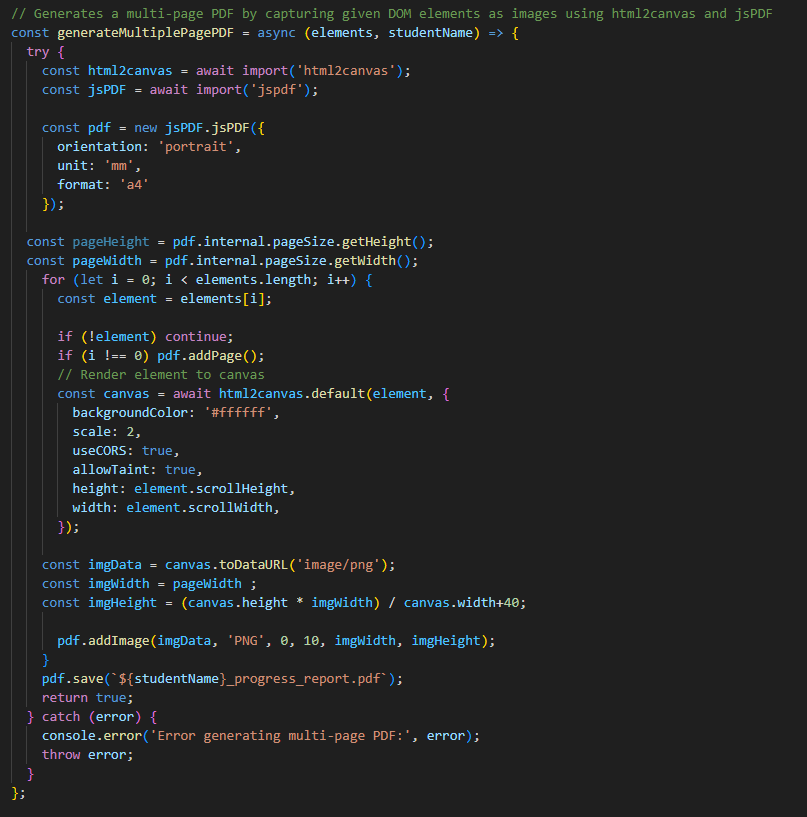
This promotes code reusability and keeps components clean and focused on their UI logic.

7. Special Utility Functions

• 7.1 export PDF Functions

In our system, nearly every page that includes reports or charts (such as MyProgress, or AllReports) supports a PDF export feature. This feature allows both students and lecturers to save the displayed information — including charts, summaries, and AI feedback — as a downloadable report for future use or sharing.

The generateMultiplePagePDF function (shown in the image above) uses the html2canvas and jsPDF libraries to capture DOM elements as canvas images and convert them into a multi-page PDF file. Each visual component is added to a new page in the PDF, which is saved under the student's name.

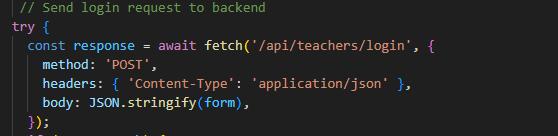


7.2 Server Communication Functions

POST Request

Explanation:  
The POST method is used to send data from the client (browser) to the server. It is typically used when submitting a form, logging in, or registering. The data is securely sent in the request body.

Example from Code:



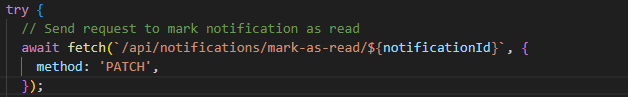
Explanation:

* This line sends the login form with ID and password.
* The data is sent as JSON using the POST method.
* If successful, the user is redirected to the dashboard.

PATCH Request

The PATCH method is used to partially update an existing resource on the server.  
In our example, we use PATCH to mark a specific teacher notification as "read" without modifying other fields.

Example from code:



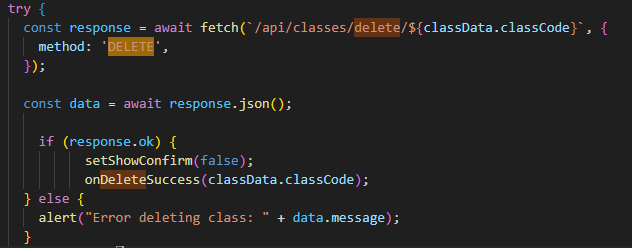
Explanation:

* This line sends a PATCH request to update a single notification.
* Only the read status is changed – everything else remains unchanged.
* Then it calls fetchNotifications() to refresh the list.

DELETE Request

The DELETE method is used to remove a resource from the server, such as a class, user, or file.  
In our system, we use DELETE when a lecturer clicks “Delete” and confirms the removal of a class.

Example from the code:



Explanation:

* This sends a DELETE request to the server using the class code.
* If successful, the class card is removed using onDeleteSuccess.
* The deletion happens only after user confirmation in a modal.

Server-Side API

8.1 Server Initialization – app.js

This file is responsible for bootstrapping the Express server and linking together all server components:

* General setup: Loads required libraries like express, mongoose, cors, body-parser, and environment variables.
* Database connection: Connects to MongoDB using Mongoose.
* CORS configuration: Enables access from both Vercel domains and local development.
* Route registration: All API routes (teachers, students, notifications, etc.) are imported and mounted under /api/....
* Deployment-ready for Vercel: The app object is exported, and app.listen() runs only in local development.

8.2 Mongoose Schemas Overview

8.2.1 ClassSchema.js

* Purpose: Represents a class created by a teacher, including the simulation situation, question, and student answers.
* Main Fields:
  + classCode, className, subject, situation, question
  + createdBy – ID of the teacher who created the class
  + students[] – array of answers including studentId, answerText, analysisResult, submittedAt
* Custom Method:
  + getStudentAnswerCount(studentId) – returns how many times a student submitted answers in this class.

8.2.2 NotificationSchema.js

* Purpose: Represents notifications sent to teachers, such as exam reminders, messages, or success alerts.
* Main Fields:
  + teacherId – ID of the teacher
  + type – one of 'exam', 'message', 'success', etc.
  + title, read
* Note: Uses automatic timestamps (createdAt, updatedAt)

8.2.3 StudentNotificationSchema.js

* Purpose: Represents notifications sent to students – such as answer submission, new exam, or PDF export.
* Main Fields:
  + studentId, type, title, content, time, read
* Note: Includes automatic timestamps.

8.2.4 StudentSchema.js

* Purpose: Represents a student user – includes login and profile information.
* Main Fields:
  + username, id (national ID), email, password, profilePic
* Additional Notes:
  + Stored in collection "students"
  + Versioning (\_\_v) is disabled

8.2.5 TeacherSchema.js

* Purpose: Represents a teacher in the system – includes name, ID, subject, and profile picture.
* Main Fields:
  + username, id (exactly 9 digits), email, password, subject, profilePic
* Additional Notes:
  + Stored in "lecturers" collection
  + Versioning disabled

8.3 API Route Files

8.3.1 classes.route.js

|  |  |  |
| --- | --- | --- |
| Description | Path | Method |
| Create a new class | /create | POST |
| Get all classes created by a specific teacher | /teacher/:teacherId | GET |
| Submit a student’s answer and analyze it | /submit-answer | POST |
| Get class by class code | /get-class-by-code | GET |
| Get classes where a student submitted | GET /api/classes/get-classes-done-simulation/:studentId | GET |
| Get all classes in the system | /get-all-classes | GET |
| Delete a class and notify the teacher | /delete/:classCode | DELETE |
| Generate class insights from AI | /ai-class-insight | POST |
| Get a student’s latest answer in a specific class | /:classCode/student/:studentId | GET |

• 8.3.2 claude.route.js

This route file defines various POST endpoints that interact with Claude AI for generating content, analyzing student responses, simulating emotional learning situations, and facilitating contextual conversations for both teachers and students.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| POST | /ask | Sends a single prompt to Claude for a response | prompt, maxTokens, temperature, model, system | General-purpose Claude interaction |
| POST | /chat | Handles a multi-turn conversation with Claude | messages[], maxTokens, temperature, model, system | Dialogues and multi-message flows |
| POST | /generate-situation | Generates an emotional learning scenario + question | topic, maxWords, previousSituations | SEL-focused scenarios per CASEL framework |
| POST | /analyze-response | Analyzes a student's answer according to CASEL 5 | situation, question, studentResponse, studentName | Provides feedback and scores for SEL competencies |
| POST | /chat-insight | Enables teachers to chat with Claude with access to class/student data | teacherId, messages[] | Teacher-guided Claude insights using real classroom data |
| POST | /student-chat-insight | Enables students to chat with Claude in a contextual way | studentId, messages[] | Student-focused Claude assistant with personal context |

• 8.3.3 notifications.route.js

This route manages lecturer notifications. It supports creating notifications, retrieving all notifications for a given teacher in reverse-chronological order, marking individual notifications as read, and marking all notifications as read.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| POST | /create | Create a new notification for a teacher | teacherId, type, title, read | Used when a student submits an answer or a class is deleted |
| GET | /teacher/:teacherId | Fetch all notifications for a specific teacher | none | Display the teacher's notifications ordered by latest first |
| PATCH | /mark-as-read/:notificationId | Mark a single notification as read | none (just notification ID in URL) | For managing read/unread state of individual notifications |
| PATCH | /mark-all-as-read/:teacherId | Mark all notifications as read for a specific teacher | none | Used when clicking "Mark All as Read" button |

• 8.3.4 studentNotifications.route.js

This route manages student notifications. It allows for creating a new notification, retrieving all notifications for a specific student, marking a single notification as read, and marking all notifications as read.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| POST | /create | Create a new notification for a student | studentId, type, title, content, time, read | Used when a student submits a simulation or receives feedback |
| GET | /student/:studentId | Retrieve all notifications for a specific student | none | Used by the student's notification interface (StudentHeader) |
| PATCH | /mark-as-read/:notificationId | Mark a single notification as read | none (just notification ID in URL) | Helps track read/unread status of notifications |
| PATCH | /mark-all-as-read/:studentId | Mark all notifications as read for a specific student | none | Used when clicking "Mark All as Read" button |

• 8.3.5 students.route.js

This route handles student registration, login, and information retrieval. It includes creating a student account, logging in with ID and password, fetching student(s) by ID, and managing password reset through email verification.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| GET | / | Retrieve all students | none | For teacher's full student list display |
| GET | /:studentId | Retrieve a student by ID | none (just ID in URL) | For showing student profile or admin view |
| POST | /details | Retrieve multiple students by array of IDs | studentIds[] | Useful for comparative student analysis |
| POST | /register | Register a new student | id, email, password, name, ... | Adds a new student to the system |
| POST | /login | Student login with ID and password | id, password | Validates credentials and grants access |
| POST | /forgot-password | Send a verification code to student email | email | Begins the password reset flow |
| POST | /verify-code | Verify email code for reset | email, code | Step to confirm student identity |
| POST | /reset-password | Set new password | email, newPassword | Final step to update student credentials |

• 8.3.6 summary.route.js

This route provides a dashboard summary for a specific teacher. It retrieves all classes created by the teacher and calculates stats like active class count, number of unique students, and the most frequent topic.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| GET | /teacher/:teacherId/summary | Get dashboard summary for a teacher | none | Displaying teacher's overall dashboard stats |

• 8.3.7 teachers.route.js

This route handles teacher registration, login, password recovery, and fetching all teachers. It includes ID and email validation, and error handling for duplicate records.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| GET | / | Retrieve all teachers | none | - |
| POST | /register | Register a new teacher | id, email, password, name,.. | Creates a new teacher account |
| POST | /login | Teacher login | id, password | Authenticates existing teacher |
| POST | |  | | --- | |  |   /forgot-password | Send verification code by email | email | Initiates password reset |
| POST | /verify-code | Verify email code | email, code | Code validation step |
| POST | /reset-password | Set new password | email, newPassword | Completes password reset process |

• 8.3.8 teacherStudentProgress.route.js

This route allows teachers to track the overall progress of all students in their classes (including average score, number of attempts, last activity, and number of unique simulations), and also to retrieve the detailed simulation attempts of a specific student across the teacher's classes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Route | Description | Request Body Params | Primary Usage |
| GET | /student-progress/:teacherId | Retrieves overall progress data of all students in teacher's classes | none | For displaying All Reports: average, attempts, last date |
| GET | /teacher/:teacherId/student/:studentId/progress | Retrieves detailed simulations of a specific student under a teacher | none | For showing student's detailed report |

8.4 Services

8.4.1 classInsightService.js

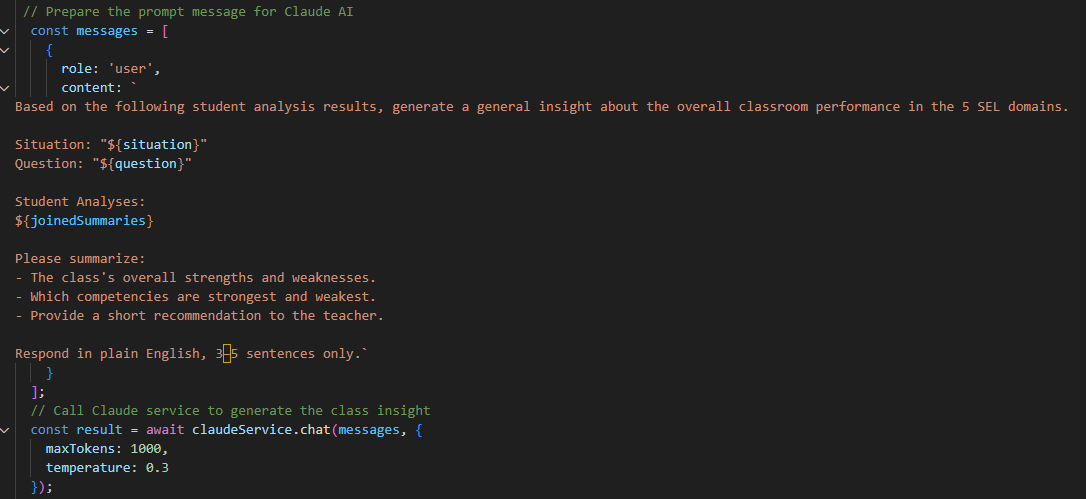
This service is responsible for generating an overall classroom-level insight based on the SEL analysis results of individual students, using the Claude AI model.

Main Function:

generateClassInsightFromClaude({ situation, question, studentAnalyses })

* Accepts simulation situation, question asked, and a list of student SEL analyses.
* Combines the student analyses into a unified prompt.
* Sends the prompt to Claude via claudeService.chat().
* Returns a brief summary of:
  + Overall strengths and weaknesses of the class
  + Strongest and weakest SEL competencies
  + A recommendation for the teacher

Key Prompt Example:



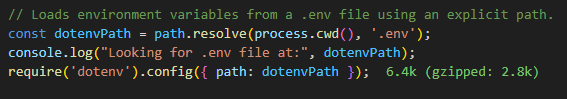
8.4.2 claudeService.js

This service handles communication with the Claude AI API (Anthropic) and is responsible for sending single-prompt or multi-message chat requests.

Key Features:

* Loads API key securely from .env using dotenv
* Builds valid JSON requests for Claude API
* Supports options like model, maxTokens, temperature, and system prompt
* Logs useful debug info and handles error cases gracefully
* Returns a structured success/error object

Example: Loading API Key



Example Request:

תמונה שמכילה טקסט, צילום מסך, גופן, תוכנה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

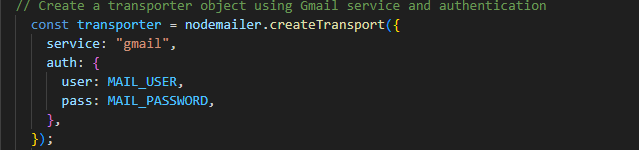
8.4.3 sendVerificationEmail.js

This service handles sending verification codes via email to users who requested a password reset.  
It uses nodemailer with Gmail as the transport mechanism and secure credentials from the .env file.

Key Features:

* Loads sender credentials (MAIL\_USER, MAIL\_PASSWORD) from environment
* Formats a rich HTML email with the verification code
* Sends the email securely using Gmail SMTP
* Logs detailed output and handles errors gracefully

Key Code Snippet:



Email Format:

תמונה שמכילה טקסט, צילום מסך, תוכנה, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

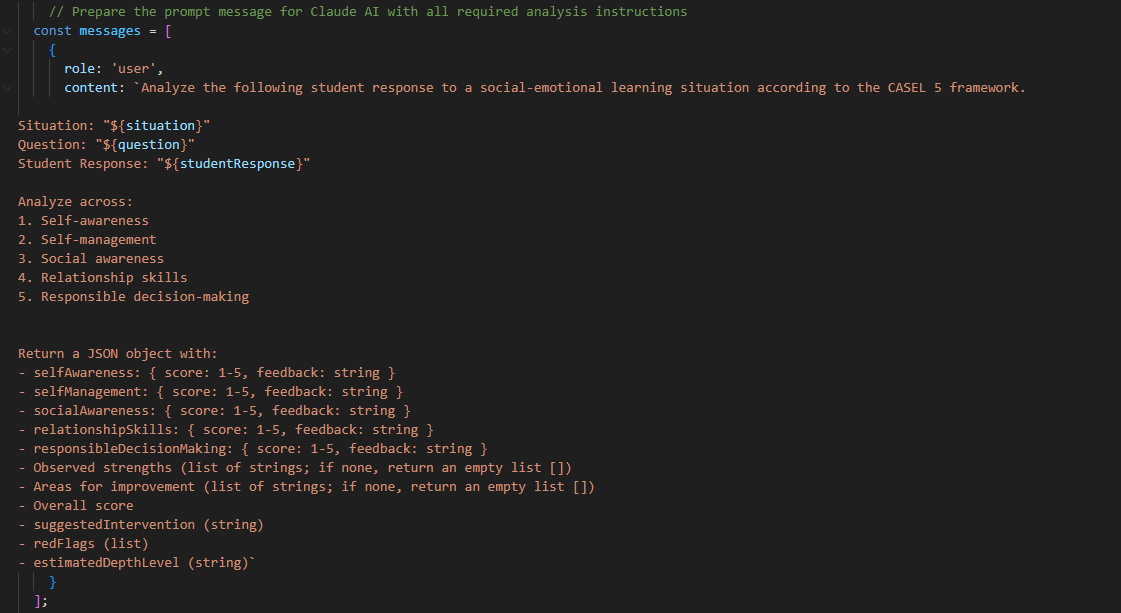
8.4.4 studentAnalysisService.js

This service analyzes a student’s written response to a SEL simulation using the CASEL 5 framework, powered by Claude AI.  
It returns a structured JSON object containing scores, feedback, strengths, weaknesses, and suggested interventions.

Key Functionality:

* Accepts: situation, question, studentResponse, studentName
* Sends a rich prompt to Claude asking for domain-specific evaluation
* Requests a well-structured JSON output from the model
* Parses and returns the JSON safely, or returns null on failure

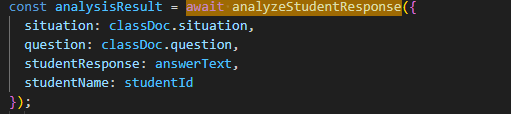
Sample Prompt:



AI Output Parsing:



Example Usage in App:



Special Functions of React

In the project, in addition to Hooks, we used special React functions to manage Context and provide global functionality:

createContext

From:  
ThemeContext.jsx, UserContext.jsx

What it does in the project:  
Used to create a new Context.  
Allows sharing data (such as theme or user) between components without manually passing props through every level of the component tree.

Provider (via Context Provider components)

From:  
ThemeProvider, UserProvider

What it does in the project:  
The components wrap the app and supply values that the Context shares (for example: theme, toggleTheme, user, login, logout).

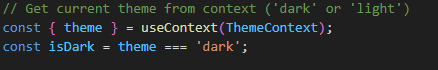


To enable global access to theme state, user data, and notifications, the application is wrapped with several Context Providers. This approach avoids excessive prop drilling and allows any component to easily access key values via useContext().

useContext

How it connects:  
After the Context is created and its values are provided through a Provider, other components in the project can use useContext to access the global values (such as the logged-in user or the current theme), without needing to pass props manually.

Example:



9. Environment and Tools

9.1 .env Configuration

General Overview:

In this project, we use the .env file (Environment Variables) to store sensitive configuration values, such as:

* Claude AI API key
* MongoDB connection URI
* Gmail credentials for sending verification emails

These variables help separate code from secret data and make the app easily configurable across different environments (development, production, etc.).

Environment Variables Used in This Project (Vercel-based Deployment):

|  |  |
| --- | --- |
| Variable Name | Purpose in the Project |
| ANTHROPIC\_API\_KEY | Used for sending requests to Claude AI |
| MONGO\_URI | MongoDB database connection string |
| MAIL\_USER | Sender Gmail address for verification |
| MAIL\_PASSWORD | Gmail password |

Usage in the Code:

Load environment variables:



Example from claudeService.js:

תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Example from sendVerificationEmail.js:

תמונה שמכילה טקסט, גופן, צילום מסך, שעון

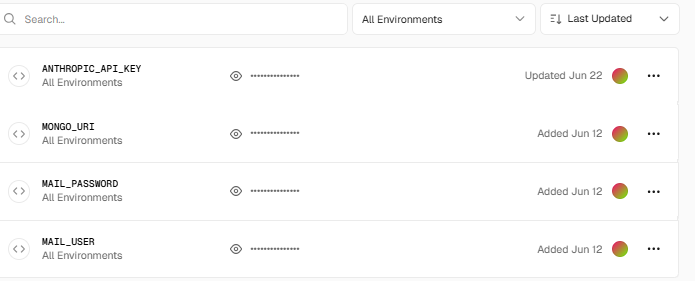
תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

How It Works on Vercel:

Since our project is deployed using Vercel, we do not store the .env file in the codebase.  
Instead, we define environment variables directly in the Vercel Dashboard under:

Project Settings > Environment Variables

At runtime, Vercel injects these values into process.env, allowing the backend code to access them just like in local development.



9.2 Environment Variables on Vercel

Below are the environment variables defined for the project via Vercel’s Environment Settings. These values are injected automatically into process.env and used for secure integration with external services (email, database, AI).

תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תמונה שמכילה טקסט, צילום מסך, קבלה, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תמונה שמכילה טקסט, גופן, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

9.2 Special Configurations (vercel.json)

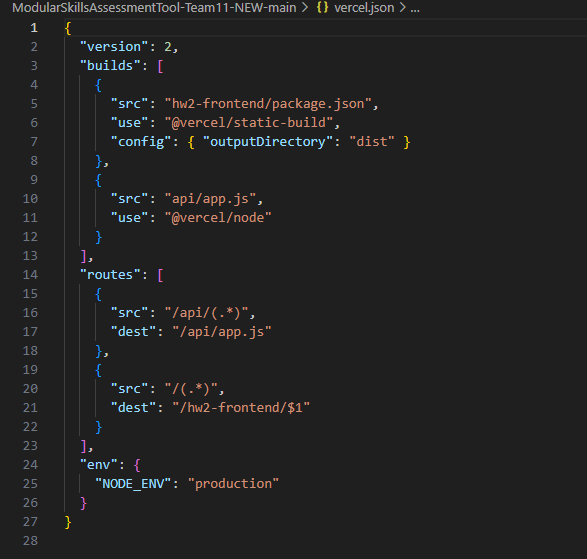
The vercel.json file tells Vercel how to build, route, and serve both the frontend and backend parts of the project.  
In this case, the project is a monorepo with:

* A frontend React app in hw2-frontend/
* A Node.js API in api/app.js

File Breakdown:

|  |  |
| --- | --- |
| Section | Description |
| version | Specifies the Vercel config version (always set to 2) |
| builds | Defines how to build each part – React app as static, Node as serverless |
| routes | Routes API calls to app.js, and all other requests to the frontend |
| env | Defines environment variables used during build time |

Example: vercel.json Configuration File Structure



10. AI Prompts Used in the Project

10.1 Class Insight Prompt – Generate Summary for Teacher

Used in: classInsightService.js  
Prompt:

תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

10.2 Student Response Analysis Prompt – CASEL 5 Evaluation

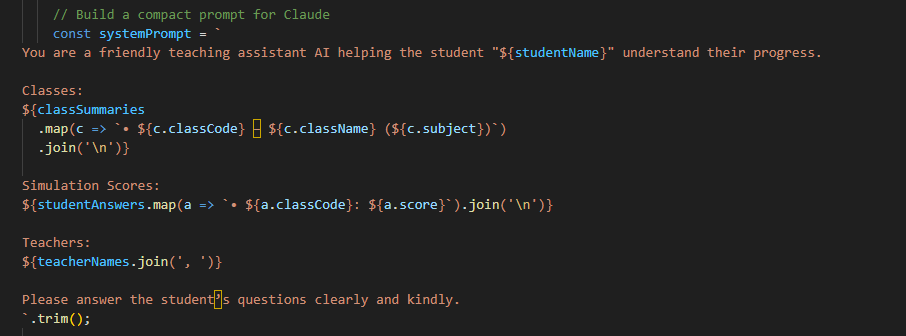
Used in: studentAnalysisService.js  
Prompt:

תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

10.3 AI–Student Chat Prompt *(Dynamic Prompt via API)*

Used in: POST /student-chat-insight (API Route)  
Prompt (system message to Claude):



10.4 AI–Teacher Chat Prompt *(Dynamic Prompt via API)*

Used in: POST /chat-insight (API Route)  
Prompt (system message to Claude):

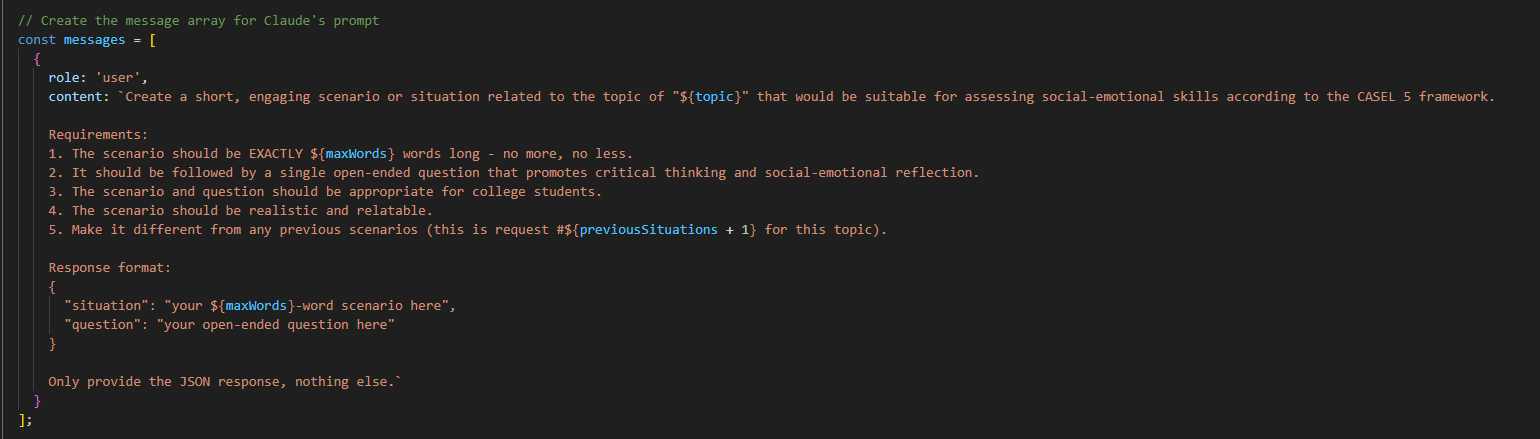
תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

10.5 Simulation Generation Prompt – When Creating a New Class

Used in: POST /generate-situation (API Route)  
Purpose:  
Generate a realistic situation and open-ended question suitable for assessing SEL (Social-Emotional Learning) using the CASEL 5 framework.

Prompt sent to Claude:



11. Login Credentials for Testing

To allow testers and evaluators to explore the system without creating new accounts, we provide demo login credentials for both teacher and student roles.

These demo users have access to fully functional parts of the system and are useful for validating features such as simulations, reports, AI chat, and class management....

11.1 Teacher Demo

* ID: 123456789
* Password: 123456789
* insights

11.2 Student Demo

* ID: 234234234
* Password: 123456789

12. References

12.1 Code Snippets Taken from External Sources

1. StackOverflow / React Docs – useEffect that runs only on mount

Why we used it:  
We needed to trigger actions like data fetching only once when the component loads. The pattern useEffect(() => {...}, []) is a standard solution, widely referenced in both StackOverflow and official docs.

Reference:  
 <https://react.dev/reference/react/useEffect>

Code Example:

תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

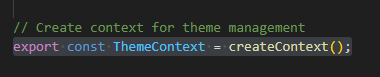
תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

2. React Documentation – Using createContext and useContext

Why we used it:  
To make global values (like theme and user info) accessible throughout the app without prop drilling, we used React Context. We created a context and accessed it using useContext() in various components.

Reference:  
 <https://react.dev/learn/passing-data-deeply-with-context>

Code Example:

תמונה שמכילה טקסט, גופן, צילום מסך

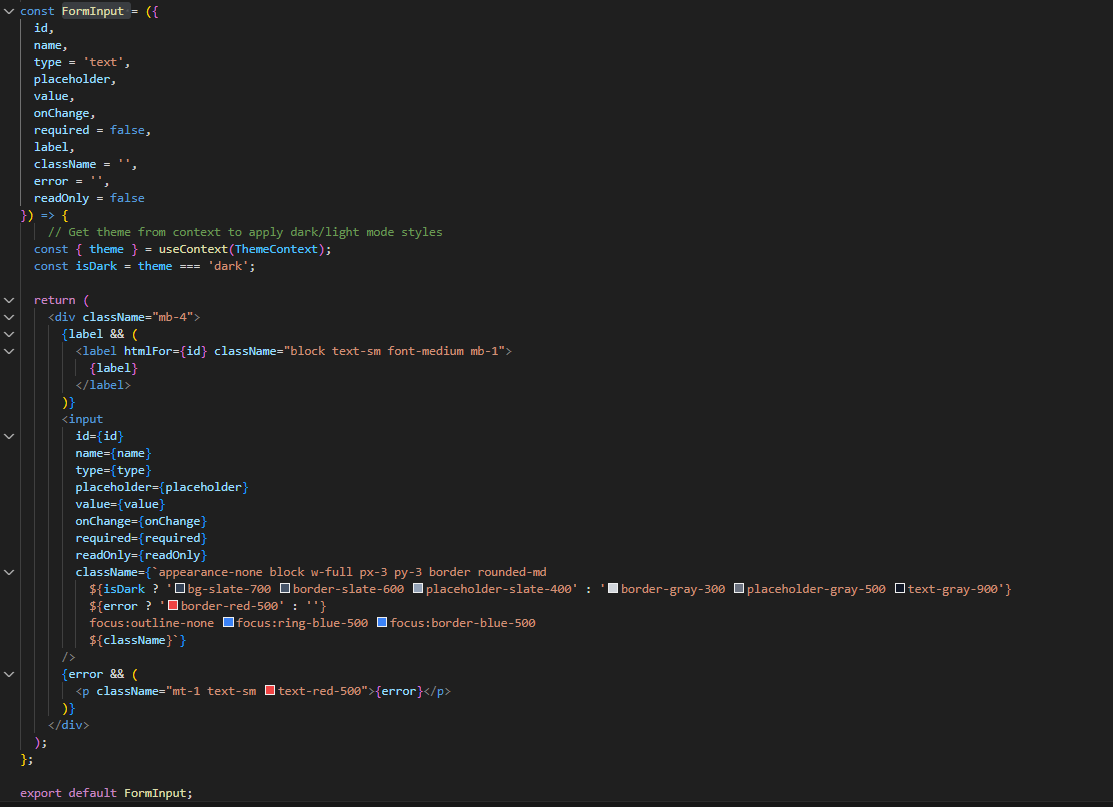
תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

3. React Documentation – Reusable components with props

Why we used it:  
For components like FormInput that are used across multiple forms, we built them to receive props like label, value, and onChange, following the reusable component model described in the React docs.

Reference:  
 <https://reactjs.org/docs/components-and-props.html>

Code Example:



12.2 Documentation / Articles Used

|  |  |  |
| --- | --- | --- |
| Source | Topic | Link |
| Vercel Documentation | Configuring projects with vercel.json | <https://vercel.com/docs/project-configuration> |
| Nodemailer Documentation | Sending emails using Gmail | <https://nodemailer.com/usage/> |
| Anthropic Claude API Docs | Sending messages to Claude via API | <https://docs.anthropic.com/claude/reference/messages_post> |
| CASEL Framework | CASEL 5 model for SEL skill analysis | <https://casel.org/fundamentals-of-sel/> |

13. Links

13.1 Database Connection (MongoDB URI)

The system uses MongoDB Atlas as its cloud-based database.  
The server connects using the following URI:

MongoDB Connection mongodb+srv://n0502898789:Nahla%3F%3F123@cluster0.jvrnjss.mongodb.net/modular\_skills?retryWrites=true&w=majority

Note: In production, this URI is stored securely in the .env file and accessed via process.env.MONGO\_URI.

13.2 GitHub Repository

The full source code for both the frontend (React) and backend (Node.js/Express) is hosted on GitHub under the following repository:  
 [ModularSkillsAssessmentTool-Team11-NEW](https://github.com/NahlaAboromi/ModularSkillsAssessmentTool-Team11-NEW)

13.3 Live Deployment (Vercel)

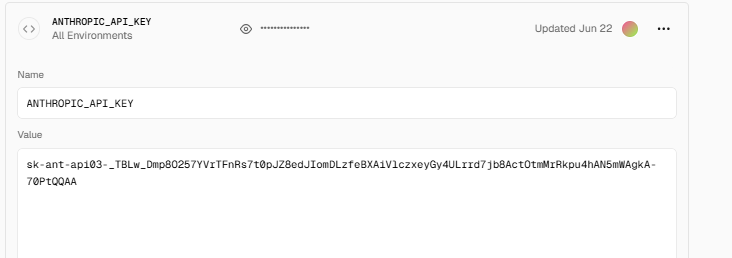
The live version of the system is deployed on Vercel, and can be accessed at:  
<https://modular-skills-assessment-tool-team-two.vercel.app/>

This link allows full access to both student and teacher functionalities and is suitable for testing, demo, and evaluation.

14. External APIs and Integrations

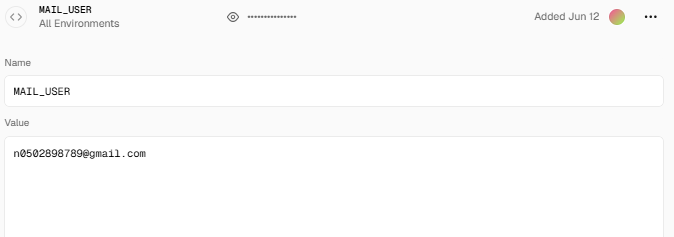
14.1 Claude AI (via Axios & Anthropic API)

* Used In: claudeService.js, /api/claude
* Purpose:  
  Enables intelligent analysis, summarization, and interactive chat based on student responses and simulations using the Claude 3 model by Anthropic.  
  This includes generating AI-driven feedback, insights for teachers, and real-time conversations with students.
* Implementation Details:
  + Communicates with Claude via HTTPS using the endpoint https://api.anthropic.com/v1/messages.
  + The API key is loaded from .env using process.env.ANTHROPIC\_API\_KEY.
  + Uses the axios library to send POST requests with prompt messages (user content) and optional system instructions.
  + Responses are used to generate feedback, summary, or guidance for both students and teachers.
* Environment Variable Required:



14.2 Email Verification Service (Nodemailer)

* Used In: sendVerificationEmail.js
* Purpose:  
  Sends verification codes (OTP) to users when they reset their password.
* Implementation Details:
  + Uses nodemailer to send emails via SMTP services .
  + Email credentials are stored in the .env file to secure them.
  + The message includes a code and basic HTML formatting for user-friendly emails.
* Environment Variables Required:

 תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

14.3 MongoDB Atlas

* Used In: app.js, all routes and models
* Purpose:  
  Stores all persistent data: users, classes, simulations, notifications, and analysis results.
* Implementation Details:
  + Connection established via mongoose.connect() using a URI from .env.
  + Supports both local development and cloud-based storage (MongoDB Atlas).
* Environment Variable Required:

תמונה שמכילה טקסט, גופן, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.