CourseMatrix

System Design Document

Thomas Yang
Kevin Lan
Austin Xu
Masahisa Sekita
Minh Tran

System CRC Card

Front-End

React Component: LoginPage	
This component depend on: N/A **This component is depended on by:** N/A	
Responsibilities: - Rendera login interface for users to input account information - Validate and handle user credentials: email and password - Communicate with the backend API to authenticate users - Store user authentication credentials in Redux state - Navigate users to the dashboard page after login	**Collaborator:** - @/api/authApiSlice - component (button, card, input, label, form, logo, password-input) - models(login-form - @hookform/resolvers/zod - react-hook-form - react-router-dom - zod - @/stores/authslice - react-redux - react (useState)

React Component: SignUpPage	
This component depend on: N/A **This component is depended on by:** N/A	
Responsibilities: - Render a sign-up interface for new users - Validate user input (email, password, confirm password) - Handle user registration by making API requests - Navigate users to login page after successful sign-up	**Collaborator:** - @/api/authApiSlice - component (logo, password-input, button, card, form, input, label) - models(signup-form) - @/stores/authslice - @hookform/resolvers/zod - react (useState) - react-hook-form - react-redux - react-router-dom - zod

React Component: Dashboard	
This component depend on: TimetableBuilder, UserMenu **This component is depended on by:** N/A	
Responsibilities: - Render the main dashboard layout - Manage routing within the dashboard - Provide sidebar functionality - Display breadcrumb navigation based on the current route - Redirect invalid routes to a “Not Found” page - Render specific page likes Home, Timetable Builder, and AI Assistant	**Collaborator:** - app-sidebar - ui (breadcrumb, Sidebar) - react (useLocation) - @radix-ui/react-separator - react-router-dom

React Component: TimetableBuilder	
This component depend on: CourseSearch, CreateCustomSetting, SearchFilters **This component is depended on by:** Dashboard	
Responsibilities: - Render the timetable builder UI with form fields and interactive elements allowing users to build personalized schedules - Manage course selection by handling course search, addition, and removal - Allow users to apply filters via the SearchFilters component - Handle timetable restrictions by adding or removing custom settings - Use debounce logic for search query optimization - Applies a query limit to prevent large course queries from rendering - Send API requests to fetch course data dynamically - Manages form state using react-hook-form	**Collaborator:** - lucide-react - react-hook-form - @hookform/resolvers/zod - zod - react (createContext, useEffect, useState) - utils (format-date-time, useDebounce) - ui (button, form, select) - api (coursesApiSlice) - models (models, timetable-form, filter-form)

React Component: CourseSearch	
This component depend on: TimetableBuilder, courseController, OfferingContent **This component is depended on by:** TimetableBuilder	
Responsibilities: - Render a search input field for searching course - Display a dropdown panel with search results as user type - Allow users to add a course to their form - Display course details via a hover-card - Handle search filtering options - Close dropdown panel when clicking outside	**Collaborator:** - lucide-react - utils (useClickOutside, convert-breadth-requirement) - ui (input, hover-card) - react (useContext, useEffect, useRef, useState) - models (models)

React Component: CreateCustomSetting	
This component depend on: TimetableBuilder, courseController **This component is depended on by:** TimetableBuilder	
Responsibilities: - Render a model form for creating new schedule restrictions - Handle users' different restriction types (time-based, day-based, time-off) - Provide form fields for selecting restriction details - Validate and submit form data - Close modal when requested	**Collaborator:** - lucide-react - @hookform/resolvers/zod - react-hook-form - zod - ui (card, select, form, button, checkbox, input) - react (useContext, useState) - models (timetable-form) - time-picker-hr

React Component: SearchFilters	
This component depend on: TimetableBuilder departmentController, courseController **This component is depended on by:** TimetableBuilder	
Responsibilities: - Render a modal form for applying filters to course searches - Provide dropdowns for filtering by semester, breadth requirement, and credit weight - Handle form submission and validation - Allow users to reset filters - Close the modal when requested	**Collaborator:** - lucide-react - zod - react-hook-form - utils (convert-breadth-requirement) - ui (button, card, form, select) - models (filter-form, timetable-form)

React Component: OfferingContent	
This component depend on: offeringController **This component is depended on by:** SearchFilters	
Responsibilities: - Fetch Offering Data using useGetOfferingQuery to retrieve course offering details based on course_code and semester - Handle loading state by displaying a loading message while data is being fetched - Render offering tables containing information such as section, day, time, location, enrollment, waitlist, delivery mode, instructor and notes	**Collaborator:** - @/api/offeringsApiSlice - ui/table - models

React Component: UserMenu	
This component depend on: N/A **This component is depended on by:** Dashboard	
Responsibilities: - Renders user dropdown menu - Handles editing account details such as email and password - Provides user the option to delete their account	**Collaborator:** - shadcn/ui - lucide-react - react-router-dom

React Component: SignupSuccessfulPage	
This component depend on: N/A **This component is depended on by:** N/A	
Responsibilities: - Renders a signup successful message - Redirected to this page after clicking email confirmation link sent via the auth flow	**Collaborator:** - shadcn/ui - components/logo - react-router-dom

Back-End

Component: authentication	
This component depend on: N/A **This component is depended on by:** LoginPage	
Responsibilities: - Handle authentication code verification - Extract and validate query parameter - Use Supabase's verifyOtp method to verify the authentication token - Redirect the user upon successful authentication - Return appropriate error responses for invalid or missing parameter	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

Component: userController	
This component depend on: N/A **This component is depended on by:** SignUpPage, LoginPage	

Responsibilities:
- Handle user registration, user login/logout and session refresh
- Handle user registration
  - Extract user inputted email and password
  - Call supabase.auth.signup() to register a new user
  - Send user email for account verification
- Handle user login
  - Extract user inputted email and password
  - Verify email and password by connecting to the
	Collaborator:
	- cookie-parser
	- express
	- middleware/asyncHandler
	- db/setupDb

database - Register user cookies for session - Handle logout by clearing cookies - Handle user sessions by fetching the user's refresh_token through cookies and refreshing the user's session. - Send success or error responses to the client	

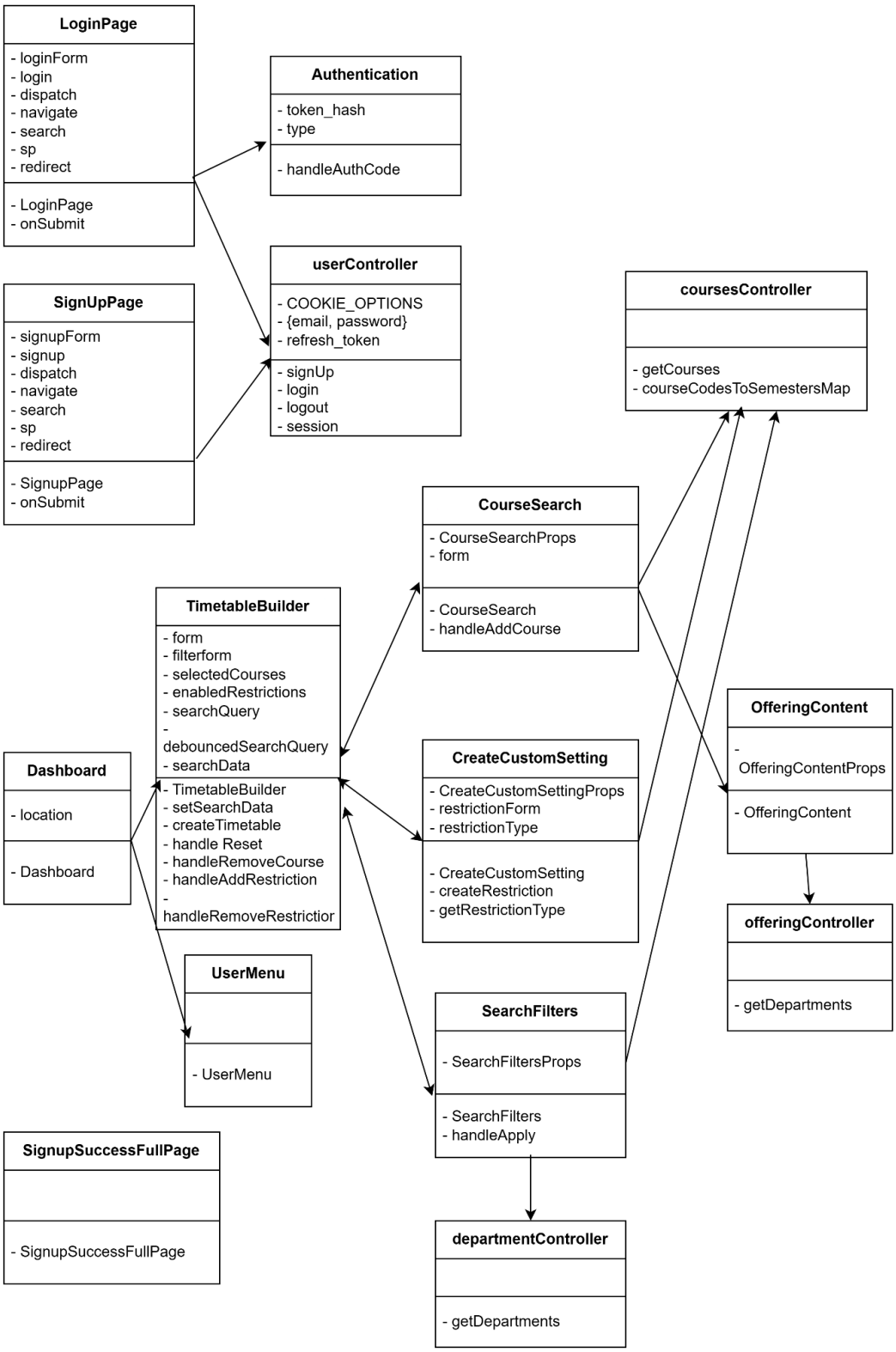
Component: coursesController	
This component depend on: N/A **This component is depended on by:** CourseSearch, CreateCustomSetting, SearchFilters	
Responsibilities: - Fetch courses and offering data from the database - Map course codes to their respective semesters for easier filtering - Filer courses based on query parameter: limit, breadth requirement, credit weight, semester, department, year level - Handle database query errors and ensure smooth API execution - Respond with filtered course data	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

Component: departmentController	
This component depend on: N/A **This component is depended on by:** SearchFilters	
Responsibilities: - Handle HTTP requests for fetching department data by querying the departments table from the database	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

- Process and return the department data as a JSON response - Handle errors and return appropriate HTTP status codes	

Component: offeringsController	
This component depend on: N/A **This component is depended on by:** OfferingContent	
Responsibilities: - Handle HTTP requests for fetching offerings data by querying the offerings table from the database - Process query parameters and return the offerings data as a JSON response - Handle errors and return appropriate HTTP status codes	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

Component Relationship Diagram



Dependencies

Front-End

```
"@radix-ui/react-label": "^2.1.1",
"@radix-ui/react-slot": "^1.1.1",
"@reduxjs/toolkit": "^2.5.1",
"class-variance-authority": "^0.7.1",
"clsx": "^2.1.1",
"lucide-react": "^0.474.0",
"react": "^18.3.1",
"react-dom": "^18.3.1",
"react-redux": "^9.2.0",
"react-router-dom": "^7.1.3",
"tailwind-merge": "^2.6.0",
"tailwindcss": "^3.4.17",
"tailwindcss-animate": "^1.0.7"
```

Back-End

```
"@supabase/supabase-js": "^2.48.1",
"@types/express": "^5.0.0",
"cookie-parser": "^1.4.7",
"cors": "^2.8.5",
"dotenv": "^16.4.7",
"express": "^4.21.2",
"swagger-jsdoc": "^6.2.8",
"swagger-ui-express": "^5.0.1",
"ts-node": "^10.9.2",
"validator": "^13.12.0"
```

Dev-Dependencies

Front-End

```
"@eslint/js": "^9.17.0",
"@types/node": "^22.10.10",
"@types/react": "^18.3.18",
"@types/react-dom": "^18.3.5",
"@vitejs/plugin-react": "^4.3.4",
"autoprefixer": "^10.4.20",
"eslint": "^9.17.0",
"eslint-plugin-react-hooks": "^5.0.0",
"eslint-plugin-react-refresh": "^0.4.16",
"globals": "^15.14.0",
```

```
11.   "postcss": "^8.5.1",
12.   "typescript": "~5.6.2",
13.   "typescript-eslint": "^8.18.2",
14.   "vite": "^6.0.5"
```

Backend

```
 1.   "@types/cookie-parser": "^1.4.8",
 2.   "@types/cors": "^2.8.17",
 3.   "@types/swagger-jsdoc": "^6.0.4",
 4.   "@types/swagger-ui-express": "^4.1.7"
```

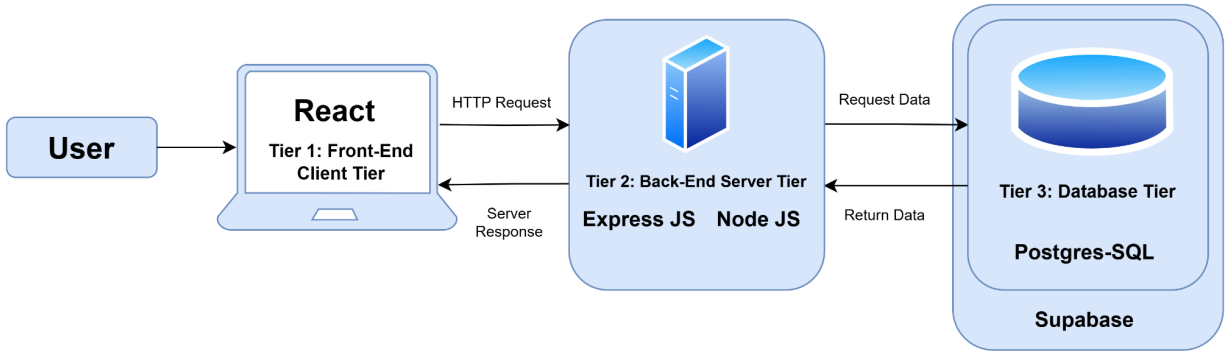
Software Configuration Summary

Operating System: Any
Language: TS/TSX
Compiler: Frontend (Vite + React) Uses esbuild , Backend (Node.js + Express) Uses tsc
Database: Postgres-SQL (Supabase)
Server Port: 8081
Front-End Port: 5173

System Architecture

Three-Tier Architecture

Our team believes that three tier is the most suitable architecture for this software application. The front-end will be handled using React framework for seamless integration between JavaScript and HTML. NodeJS, ExpressJS will be used to handle backend logic. For the Database tier, we will use the PostgreSQL database via Supabase.



System Decomposition

1. User login/registration

1.1 User Registration

First-time users must visit the **SignUpPage** to create an account before accessing the Course Matrix features. Upon submitting their email and password, **SignUpPage** sends this data to the **userController** component in the backend, which forwards the request to the **Supabase** database.

- If the registration is successful, the database will send a verification email to the user's registered email address
- If the email is already registered, **userController will** respond with **error code 400** and the message: "User already exists". The **SignUpPage** will notify the user that the registering email already exists.
- Other errors will be responded to with **error code 500** and message: "Internal Server Error". A message: "An unknown error occurred. Please try again" will be displayed on the front end.

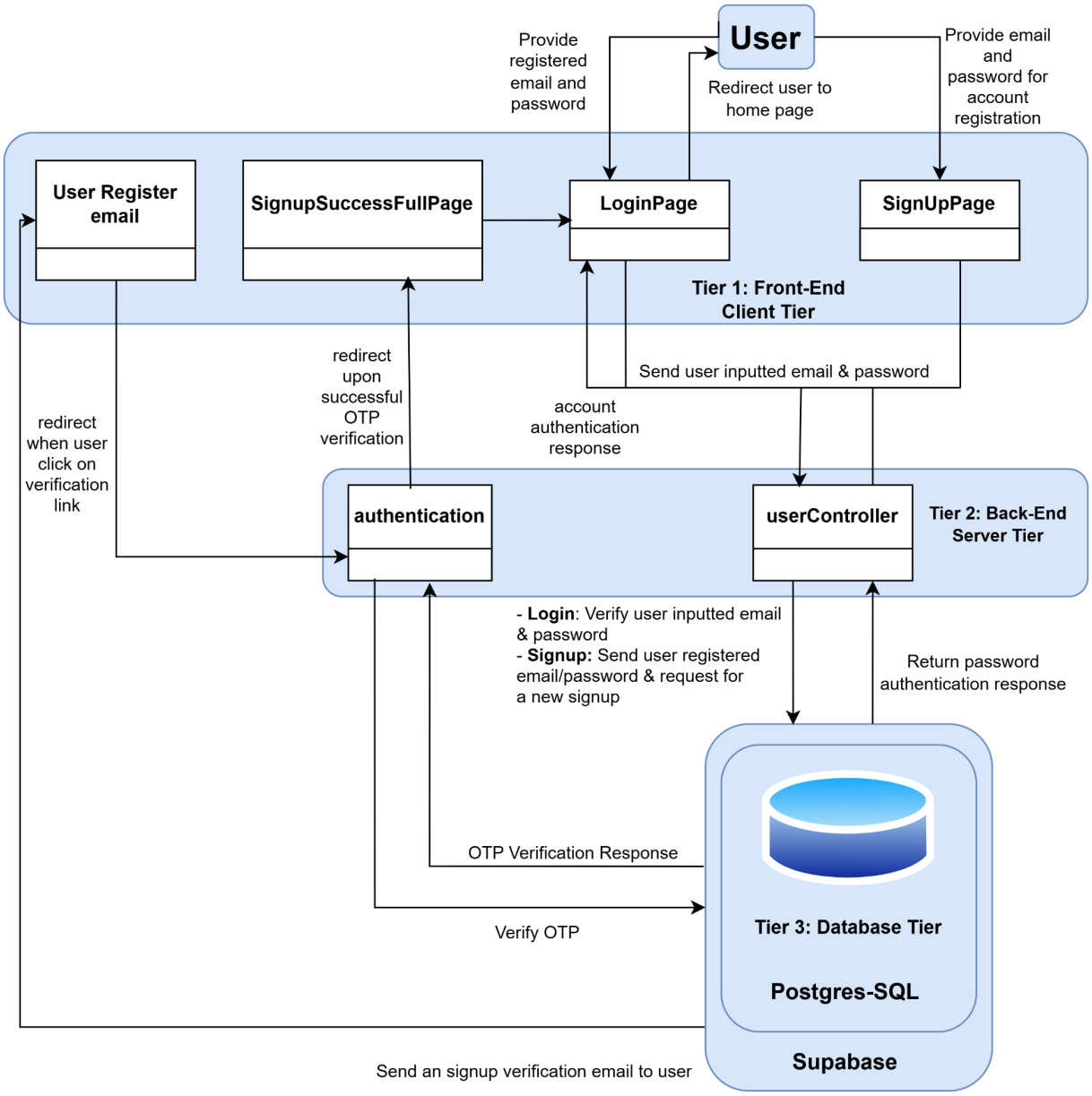
Once the user clicks the verification link in the email, a token is sent to the **authentication** component in the backend, which will then verify the OTP with the database.

- If the verification is successful, users are redirected to the **SignUpSuccessfulPage**, where they can navigate back to the **LoginPage** to sign in
- If verification fails due to an invalid or expired token, users are redirected to an **AuthErrorPage** to retry the verification process

1.2 User Login

When logging in, users enter their registered email and password in the **LoginPage** component, which sends this data to the **userController** in the backend. **userController** then connects to the database to verify the credentials.

- If the email and password are correct, the system will generate and store the needed information for the session and redirect users to the course matrix homepage (**Dashboard** component)
- If the login attempt fails due to incorrect credentials, **userController** returns **error code 401** and displays "Login invalid. Please check your email or password" in the **LoginPage** component.
- If a server error occurs during login, the system responds with **error code 500** and displays the error message on the console for debugging


User
Provide registered email and password
Redirect user to home page
Provide email and password for account registration
User Register email
SignupSuccessFullPage
LoginPage
SignUpPage
Tier 1: Front-End Client Tier
redirect upon successful OTP verification
Send user inputted email & password
account authentication response
redirect when user click on verification link
authentication
userController
Tier 2: Back-End Server Tier
- Login: Verify user inputted email & password
- Signup: Send user registered email/password & request for a new signup
Return password authentication response
OTP Verification Response
Verify OTP
Tier 3: Database Tier
Postgres-SQL
Supabase
Send an signup verification email to user

2. Course Display/Course Entries

Upon logging in, users are directed to the **Dashboard** component, the main frontend interface. From here, users can navigate to various features within the course matrix. In Sprint 1, the team is primarily focused on developing the Timetable Builder feature.

On the **TimetableBuilder** page, users can search for courses using the search bar. By default, all courses available in the selected semester are displayed in the dropdown, rendered by the **CourseSearch** component. As users type, this frontend component sends **debounced HTTP requests** to the **courseController** in the backend, which queries the **Supabase** database to retrieve courses matching the user's input.

- If the request is successful, **courseController** returns the relevant courses with a **200 OK** status.
- If there is an issue retrieving course data from the database, the backend responds with:
    - **Error code 500:** "Failed to retrieve course data. Please try again later."
    - **Error code 400:** "Invalid course search parameters."

To refine their search, users can apply filters using the **SearchFilter** component. When they submit their filter options, an HTTP request is sent to the **courseController**, which processes the query and returns the filtered results. If the filter options contain invalid data, the backend responds with **error code 400**: "Invalid filter parameters. Please check your input."

The **OfferingContent** component retrieves and displays course offering information using the **offeringController**. When users request course offerings, an HTTP request is sent to the backend with the course_code and semester as query parameters. The **offeringController** queries the **Supabase** database to fetch matching offerings.

- If the query is successful, the backend returns the data with **response code 200 OK**.
- If an error occurs, the following responses may be triggered:
    - **Error code 500:** "Failed to retrieve course offerings. Please try again later."
    - **Error code 400:** "Missing or invalid query parameters: course_code or semester."
    - **Error code 404:** "No course offerings found for the specified course and semester."

The **departmentsController** handles all department-related data displayed in the filter form. When users interact with department-related filters, the frontend sends a request to retrieve available departments.

- If the request is successful, the backend returns the department list with **response code 200 OK**.
- If an error occurs, possible responses include:
    - **Error code 500:** "Failed to fetch department data. Please try again later."
    - **Error code 404:** "No departments found in the database."

To personalize the timetable, users can add scheduling restrictions using the **CreateCustomSetting** component. These constraints (e.g., unavailable time slots) are sent via an **HTTP request** to the **courseController**, ensuring that the generated timetable aligns with the user's preferences

- If the request contains invalid constraints, the backend responds with **error code 400**: "Invalid scheduling constraints. Please check your input format."
- If a database error occurs, the system returns **error code 500:** "Failed to save custom timetable settings. Please try again later."