

CourseMatrix

System Design Document

Thomas Yang

Kevin Lan

Austin Xu

Masahisa Sekita

Minh Tran

CourseMatrix.....	0
System Design Document.....	0
System CRC Card.....	3
Front-End.....	3
React Component: LoginPage.....	3
React Component: SignUpPage.....	3
React Component: Dashboard.....	4
React Component: TimetableBuilder.....	4
React Component: CourseSearch.....	5
React Component: CreateCustomSetting.....	5
React Component: SearchFilters.....	6
React Component: OfferingContent.....	6
React Component: UserMenu.....	7
React Component: SignupSuccessfulPage.....	7
React Component: assistant.....	7
React Component: AssistantPage.....	8
React Component: LoadingPage.....	8
React Component: Home.....	8
React Component: TimetableCard.....	9
React Component: TimetableCardKebabMenu.....	9
React Component: TimetableCompareButton.....	9
React Component: TimetableCreateNewButton.....	10
React Component: Calendar.....	10
Back-End.....	10
Component: authentication.....	10
Component: userController.....	11
Component: coursesController.....	12
Component: departmentController.....	12
Component: offeringsController.....	12
Component: timetableController.....	13
Component: eventController.....	13
Component: restrictionController.....	14
Component: aiController.....	15
Component Relationship Diagram.....	15
Dependencies.....	17
Dev-Dependencies.....	18
Software Configuration Summary.....	19
System Architecture.....	19
Three-Tier Architecture.....	19
System Decomposition.....	20
1. User Management.....	20

2. Course Display/Course Entries.....	23
3. AI Workflow.....	25
4. Create, Read, Update and Delete timetables.....	26
Timetable Management.....	26
Event Management.....	26
Application Deployment.....	27
Deployment Plan.....	27

System CRC Card

Front-End

React Component: LoginPage	
This component depend on: N/A	
This component is depended on by: N/A	
Responsibilities: <ul style="list-style-type: none">- Render a 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: <ul style="list-style-type: none">- @/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: <ul style="list-style-type: none">- 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: <ul style="list-style-type: none">- @/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, AssistantPage, Home	
This component is depended on by: N/A	
Responsibilities: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - app-sidebar - ui (breadcrumb, Sidebar) - react (useLocation) - @radix-ui/react-separator - react-router-dom

React Component: TimetableBuilder	
This component depend on: CourseSearch, CreateCustomSetting, SearchFilters, Calendar	
This component is depended on by: Dashboard	
Responsibilities: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - @/api/offeringsApiSlice - ui/table - models

React Component: UserMenu	
This component depend on: N/A	
This component is depended on by: Dashboard	
Responsibilities: <ul style="list-style-type: none"> - Renders user dropdown menu - Handles editing account details such as email and password - Provides user the option to delete their account 	Collaborator: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - Renders a signup successful message - Redirected to this page after clicking email confirmation link sent via the auth flow 	Collaborator: <ul style="list-style-type: none"> - shadcn/ui - components/logo - react-router-dom

React Component: assistant	
This component depend on: aiController	
This component is depended on by: AssistantPage	
Responsibilities: <ul style="list-style-type: none"> - Initialize AI chat runtime with the appropriate API endpoint - Provide the chat runtime context to child components - Render the layout for assistant UI, including the thread list and active thread 	Collaborator: <ul style="list-style-type: none"> - ui/react - ui/thread

React Component: AssistantPage	
This component depend on: assistant	
This component is depended on by: Dashboard	
Responsibilities: <ul style="list-style-type: none"> - Render assistant component as the main content of the page 	Collaborator: <ul style="list-style-type: none"> -

React Component: LoadingPage	
This component depend on: N/A	
This component is depended on by: N/A	
Responsibilities: <ul style="list-style-type: none"> - Display a full-screen loading state - Center a spinner component on the page 	Collaborator: <ul style="list-style-type: none"> - ui/spinner

React Component: Home	
This component depend on: TimetableCard, TimetableCompareButton, TimetableCreateNewButton	
This component is depended on by: Dashboard	
Responsibilities: <ul style="list-style-type: none"> - Fetch and display user's timetable - Provide options to create new timetables and compare existing ones - Render a UI for filtering timetables (All, Mine, Shared) - Handle loading state while fetching timetables 	Collaborator: <ul style="list-style-type: none"> - ui - lucide-react

React Component: TimetableCard	
This component depend on: TimetableCardKebabMenu	
This component is depended on by: Home	
Responsibilities: <ul style="list-style-type: none"> - Display a timetable card with its title, last edited date and owner - Allow users to edit the timetable title and save changes - Provide access to additional options through a kebab menu 	Collaborator: <ul style="list-style-type: none"> - ui/button - ui/input - lucide-react - react

React Component: TimetableCardKebabMenu	
This component depend on: N/A	
This component is depended on by: TimetableCard	
Responsibilities: <ul style="list-style-type: none"> - Provide a dropdown menu for additional timetable actions - Allow users to navigate to the edit page for specific timetable - Display a confirmation dialog for deleting a timetable - Handle timetable deletion and refresh the timetable list after deletion 	Collaborator: <ul style="list-style-type: none"> - ui/button - react-router-dom - ui/dropdown-menu - ui/dialog - lucide-react - api/timetableSlice

React Component: TimetableCompareButton	
This component depend on: N/A	
This component is depended on by: Home	
Responsibilities: <ul style="list-style-type: none"> - Display a button to open the timetable comparison dialog 	Collaborator: <ul style="list-style-type: none"> - ui/button - ui/dialog

<ul style="list-style-type: none"> - Provide a UI for selecting two timetables to compare - Allow users to cancel or proceed with the comparison 	<ul style="list-style-type: none"> - ui/input - ui/label
--	--

React Component: TimetableCreateNewButton	
This component depend on: N/A	
This component is depended on by: Home	
Responsibilities: <ul style="list-style-type: none"> - Display a button to create a new timetable 	Collaborator: <ul style="list-style-type: none"> - ui/button - lucide-react - react-router-dom

React Component: Calendar	
This component depend on: N/A	
This component is depended on by: TimetableBuilder	
Responsibilities: <ul style="list-style-type: none"> - Parse and format course and user events for display in the calendar - Initialize and configure the calendar with different views and plugins - Display the calendar component - Provide a button for saving the timetable 	Collaborator: <ul style="list-style-type: none"> - @schedule-x/react - @schedule-x/calendar - @schedule-x/drag-and-drop - @schedule-x/event-modal - @schedule-x/theme-default/dist/index.css - ui/button

Back-End

Component: authentication
This component depend on: N/A

This component is depended on by: LoginPage	
Responsibilities: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

Component: userController	
This component depend on: N/A	
This component is depended on by: SignUpPage, LoginPage	
Responsibilities: <ul style="list-style-type: none"> - Handle user registration, user login/logout and session refresh - Handle user registration <ul style="list-style-type: none"> - Extract user inputted email and password - Call supabase.auth.signup() to register a new user - Send user email for account verification - Handle user login <ul style="list-style-type: none"> - Extract user inputted email and password - Verify email and password by connecting to the 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 	Collaborator: <ul style="list-style-type: none"> - cookie-parser - express - middleware/asyncHandler - db/setupDb

Component: coursesController	
This component depend on: N/A	
This component is depended on by: CourseSearch, CreateCustomSetting, SearchFilters	
Responsibilities: <ul style="list-style-type: none"> - Fetch courses and offering data from the database - Map course codes to their respective semesters for easier filtering - Filter 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: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

Component: departmentController	
This component depend on: N/A	
This component is depended on by: SearchFilters	
Responsibilities: <ul style="list-style-type: none"> - Handle HTTP requests for fetching department data by querying the departments table from the database - Process and return the department data as a JSON response - Handle errors and return appropriate HTTP status codes 	Collaborator: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

Component: offeringsController

This component depend on: N/A	
This component is depended on by: OfferingContent	
Responsibilities: <ul style="list-style-type: none"> - 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: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

Component: timetableController	
This component depend on: N/A	
This component is depended on by:	
Responsibilities: <ul style="list-style-type: none"> - Handle HTTP Request received from frontend: <ul style="list-style-type: none"> - Create new timetable - Get (fetch) all timetables belong to the current user in session - Update timetable - Delete timetable 	Collaborator: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

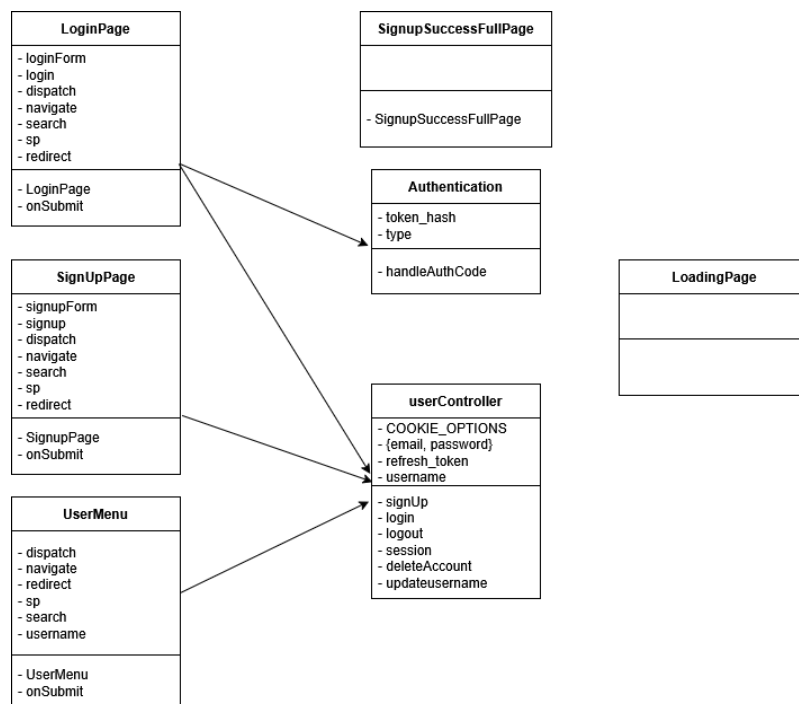
Component: eventController	
This component depend on: N/A	
This component is depended on by:	
Responsibilities: <ul style="list-style-type: none"> - Find the next occurrence date based on week day of a lecture - Generate weekly events based on week day of 	Collaborator: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

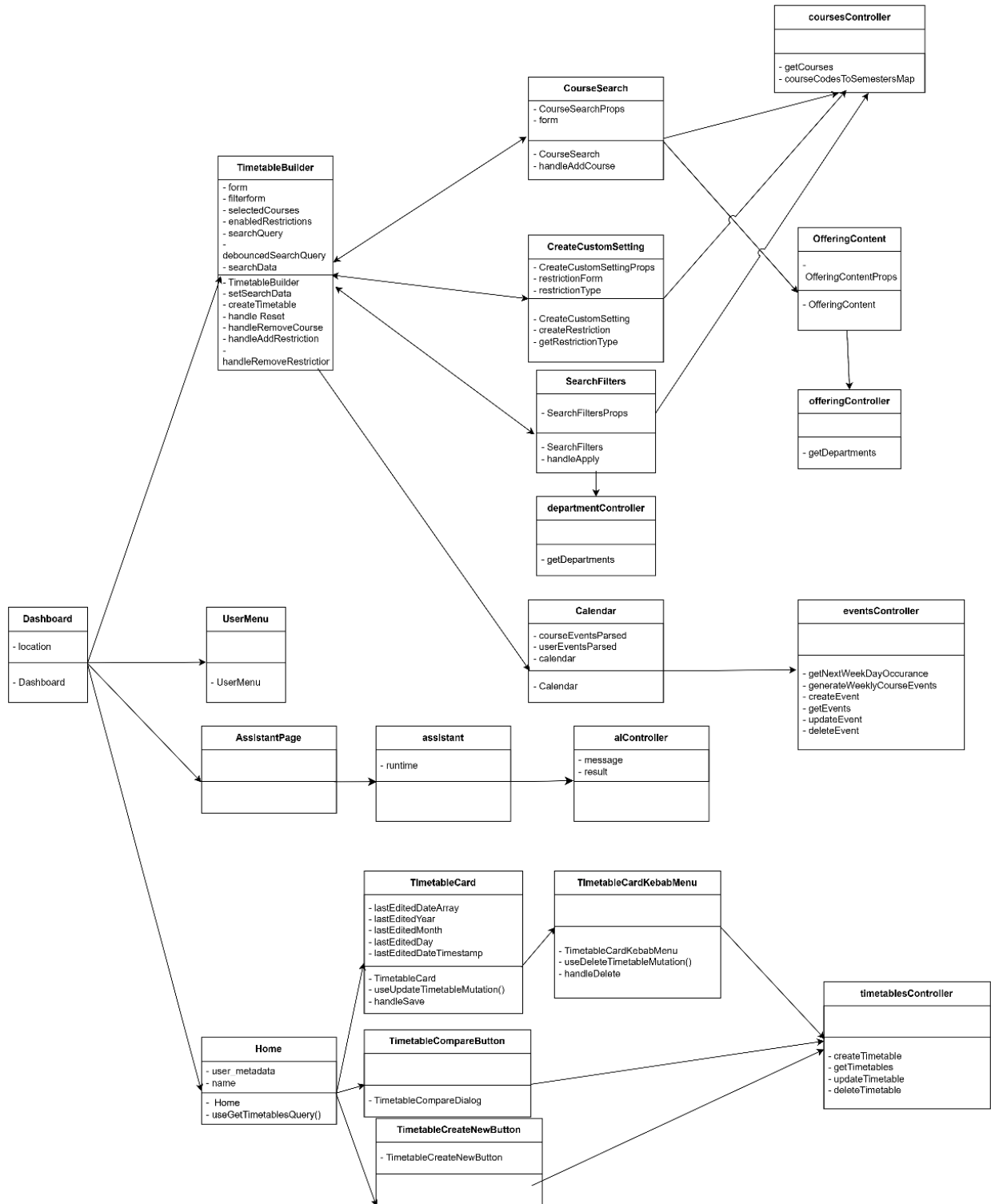
lecture from start date to end date of a semester - Distinguish between course event (lectures) and user personal events based on offering_id - Handle HTTP Request received from frontend: <ul style="list-style-type: none"> - Add new events <ul style="list-style-type: none"> - user event will be added based on the specific date - course event will be added weekly on the weekday of the lecture from the start date to the end date of the semester - Get (fetch) all events in a the user selected timetable - Update timetable (Entire course event sequence can be updated at once) - Delete timetable (Entire course event sequence can be deleted at once) 	
---	--

Component: restrictionController	
This component depend on: N/A	
This component is depended on by: N/A	
Responsibilities: <ul style="list-style-type: none"> - Handle CRUD operations for timetable restrictions - Validate user permissions before performing any action - Ensure data integrity by verifying timetable and restriction associations - Interact with the Supabase database to store and retrieve restriction data - Format and sanitize input data before storing it in the database - Return appropriate HTTP responses based on success or failure 	Collaborator: <ul style="list-style-type: none"> - express - middleware/asyncHandler - db/setupDb

Component: aiController	
This component depend on: N/A	
This component is depended on by: assistant	
Responsibilities: <ul style="list-style-type: none"> - Handle incoming chat requests from the front end - Process user message and interact with OpenAI's GPT-4o-mini model - Refine user query before calling vector database - Call vector database to perform similarity search on refined user query - Stream AI-generated responses back to the client - Enforce system instructions to keep responses relevant to course selection and timetabling 	Collaborator: <ul style="list-style-type: none"> - express - middleware/asyncHandler - @ai-sdk/openai - ai

Component Relationship Diagram





Dependencies

Front-End

```
1. "@ai-sdk/openai": "^1.1.13",
2. "@assistant-ui/react": "^0.7.86",
3. "@assistant-ui/react-ai-sdk": "^0.7.16",
4. "@assistant-ui/react-markdown": "^0.7.20",
5. "@assistant-ui/react-ui": "^0.1.7",
6. "@hookform/resolvers": "^3.10.0",
7. "@radix-ui/react-accordion": "^1.2.3",
8. "@radix-ui/react-avatar": "^1.1.3",
9. "@radix-ui/react-checkbox": "^1.1.4",
10. "@radix-ui/react-dialog": "^1.1.6",
11. "@radix-ui/react-dropdown-menu": "^2.1.6",
12. "@radix-ui/react-hover-card": "^1.1.6",
13. "@radix-ui/react-label": "^2.1.2",
14. "@radix-ui/react-select": "^2.1.6",
15. "@radix-ui/react-separator": "^1.1.2",
16. "@radix-ui/react-slot": "^1.1.2",
17. "@radix-ui/react-tooltip": "^1.1.8",
18. "@reduxjs/toolkit": "^2.5.1",
19. "ai": "^4.1.45",
20. "class-variance-authority": "^0.7.1",
21. "clsx": "^2.1.1",
22. "lucide-react": "^0.474.0",
23. "react": "^18.3.1",
24. "react-dom": "^18.3.1",
25. "react-hook-form": "^7.54.2",
26. "react-redux": "^9.2.0",
27. "react-router-dom": "^7.1.3",
28. "remark-gfm": "^4.0.1",
29. "tailwind-merge": "^2.6.0",
30. "tailwindcss": "^3.4.17",
31. "tailwindcss-animate": "^1.0.7",
32. "zod": "^3.24.1"
```

Back-End

```
1. "@ai-sdk/openai": "^1.1.13",
2. "@langchain/community": "^0.3.32",
3. "@langchain/openai": "^0.4.4",
4. "@langchain/pinecone": "^0.1.3",
5. "@pinecone-database/pinecone": "^5.0.2",
6. "@supabase/supabase-js": "^2.48.1",
7. "@types/express": "^5.0.0",
8. "ai": "^4.1.45",
```

```
9.   "cookie-parser": "^1.4.7",
10.  "cors": "^2.8.5",
11.  "csv-parser": "^3.2.0",
12.  "d3-dsv": "^2.0.0",
13.  "dotenv": "^16.4.7",
14.  "express": "^4.21.2",
15.  "langchain": "^0.3.19",
16.  "openai": "^4.85.4",
17.  "pdf-parse": "^1.1.1",
18.  "swagger-jsdoc": "^6.2.8",
19.  "swagger-ui-express": "^5.0.1",
20.  "ts-node": "^10.9.2",
21.  "validator": "^13.12.0"
```

Dev-Dependencies

Front-End

```
1.   "@eslint/js": "^9.17.0",
2.   "@types/node": "^22.10.10",
3.   "@types/react": "^18.3.18",
4.   "@types/react-dom": "^18.3.5",
5.   "@vitejs/plugin-react": "^4.3.4",
6.   "autoprefixer": "^10.4.20",
7.   "eslint": "^9.17.0",
8.   "eslint-plugin-react-hooks": "^5.0.0",
9.   "eslint-plugin-react-refresh": "^0.4.16",
10.  "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)

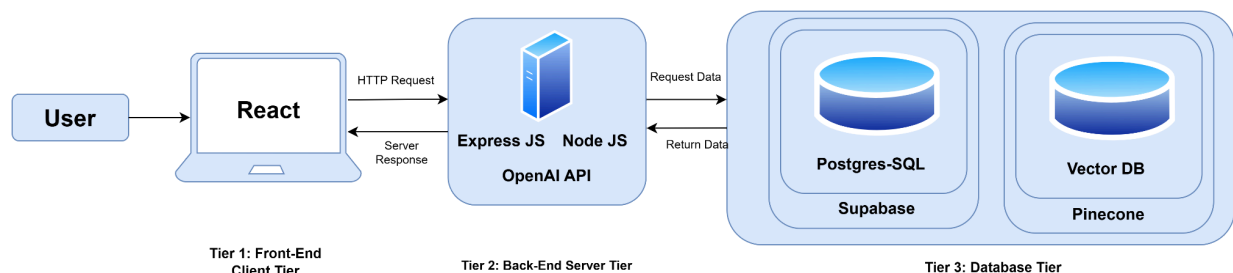
Server Port: 8081

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. The backend will also handle the interaction with the 3rd part OpenAI API. For the Database tier, we will use the PostgreSQL database via Supabase to handle users, courses, and course offerings information. VectorDatabase via Pinecone will be used in the 3rd tier to store required AI information.



System Decomposition

1. User Management

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

1.3 User Account Edit

After logging in, users can update their registered usernames and passwords in the **UserMenu** component, which sends the updated data to **userController** in the backend. **userController** then connects to the database to verify the credentials.

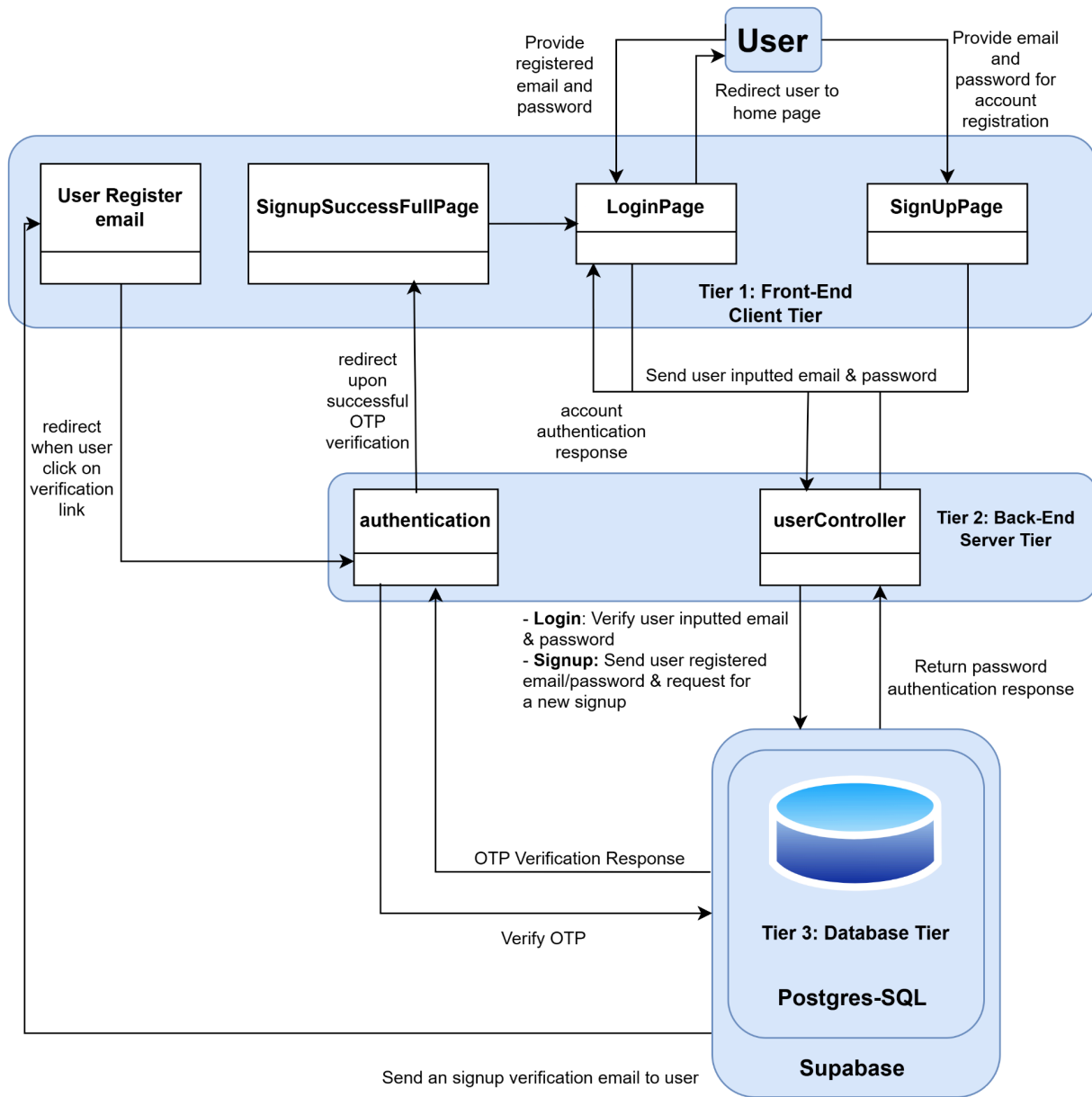
- If the updated password and/or username are acceptable (nonempty less than 50 characters long), the system will update the database information and user session accordingly.
- If the account update fails due to either incorrect or unacceptable user credentials, **userController** returns **error code 400** and displays "unable to update user"

- If a server error occurs during an account edit the system will respond with **error code 500** and display the error message on console for debugging.

1.4 User Account Delete

After logging in, users can delete their registered accounts in the **UserMenu** component, which sends the DELETE request to **userController** in the backend. **userController** then connects to the database to verify the credentials.

- If the account delete request is accepted the system will then delete the user's account, clear the user's session from their browser, and send the user to the login page.
- If the delete request is not accepted due to incorrect user credentials the **userController** returns **error code 400** and displays "User ID (uuid) is required" in the console for debugging
- If a server error occurs during an account edit the system will respond with **error code 500** and display the error message on console for debugging.



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.

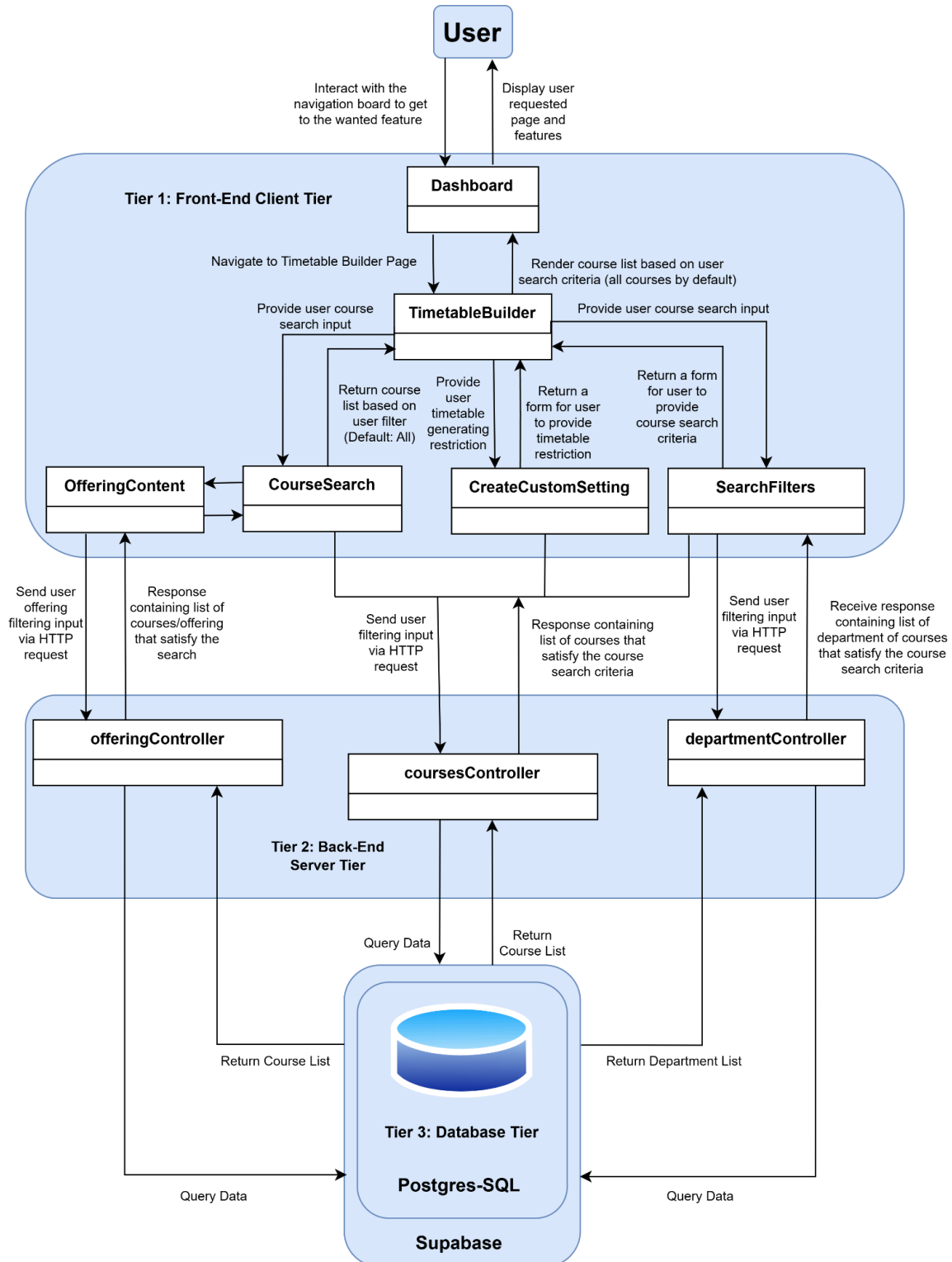
- If the query is successful, the backend returns the data with **response code 200 OK**.
- If an error occurs, the following responses may be triggered:
 - **Error code 500:** "Failed to retrieve course offerings. Please try again later."
 - **Error code 400:** "Missing or invalid query parameters: `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."



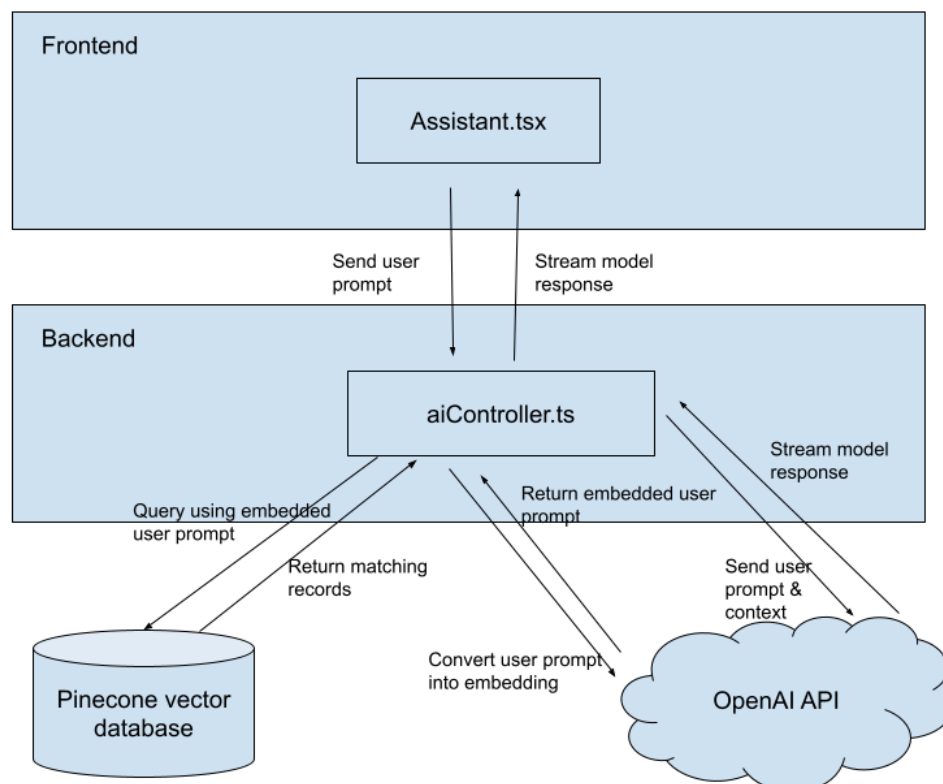
3. AI Workflow

On the **AssistantPage** component, users interact with the AI interface which is managed by a library (assistant-ui). Users can add chats, toggle between chats, delete chats, and rename chats.

When the user messages the assistant, the frontend calls **aiController** in the backend which establishes an HTTP SSE (server side event) connection. This establishes a 1 way stream of data from **aiController** to the frontend.

aiController then performs several refinement processes to improve the query and determine if it is relevant. If it is relevant, it calls the Pinecone vector database with the user query and it returns several records that match the query. This is then passed to a model instance as context and the model response is streamed back to the client side via HTTP SSE.

- If the query is successful, the backend returns the data with **response code 200 OK**.
- If there is an issue in the backend and the stream fails to return data, the backend responds with:
 - **Error code 500:** [[various error messages as a result of stream failures]]



4. Create, Read, Update and Delete timetables

Users can create, access, and edit their timetables through the main frontend interface (Dashboard) by navigating to the Home page. The system retrieves the user's created timetables via an HTTP request to the **timetableController**, displaying them using the **TimetableCard** component. Each card contains the timetable title, last update timestamp, and owner's username (if the owner does not have a username, it will show the owner's email instead of their username).

Timetable Management

- **Viewing Timetables:** On page load, a request is sent to **timetableController.getTimetables()**. If successful (**200 OK**), timetables are displayed. Errors such as unauthorized access (**401 Unauthorized**) or missing data (**404 Not Found**) are handled accordingly.
- **Editing a Timetable Name:** Clicking the pencil icon triggers a request to **timetableController.updateTimetable()**. Errors such as missing parameters (**400 Bad Request**) or unauthorized edits (**401 Unauthorized**) return appropriate responses.
- **Deleting a Timetable:** Clicking the three-dot menu opens the **TimetableCardKebabMenu**, which provides options to edit events or delete the timetable. The deletion request is sent to **timetableController.deleteTimetable()**, with error handling for invalid or unauthorized deletions.
- **Creating a New Timetable:** Clicking the **TimetableCreateNewButton** redirects the user to the **Calendar** component. A new timetable is created by sending a request to **timetableController.createTimetable()**, handling errors such as missing fields (**400 Bad Requests**).

Event Management

- **Fetching Events:** When a user chooses to edit a timetable's events, they are redirected to the **Calendar** component. An HTTP request is sent to **eventsController.getEvents()** to retrieve all events associated with the selected timetable and user session. If the timetable ID is invalid (**404 Not Found**) or unauthorized (**401 Unauthorized**), an error response is returned.

By integrating error handling directly into the request flow, the system ensures robust feedback and security across timetable and event operations.

Application Deployment

Deployment Plan

Our application is to be deployed on the web using a Google cloud virtual machine to host our website. Our application's front end will be deployed on the port 5173 of the virtual machine while our backend server will be running on port 8081 to ensure proper design architecture is maintained.

On the Google cloud virtual machine our application will be launched using docker containers (one for frontend React app, and one for backend express server), as taught in class, to ensure the ease of deploying our application, ensure consistency across environments, and that when deployed it will remain running reliably.

Currently a simple version of our application is deployed in the following address:
<http://34.130.253.243:5173>. The current instance being run is our sprint 1 application.