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: - 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:** - @/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, AssistantPage, Home

This component is depended on by: N/A

Responsibilities:	**Collaborator:**
- 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	- 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, LoadingPage, OfferingInfo

This component is depended on by: Dashboard

Responsibilities:	**Collaborator:**
- 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	- lucide-react - react-hook-form - @hookform/resolvers/zod - zod - react (createContext, useEffect, useState) - utils (type-utils, format-date-time, useDebounce) - ui (button, form, select, checkbox) - api (coursesApiSlice, offeringsApiSlice, restrictionsApiSlice) - 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

React Component: assistant	
This component depend on: aiController **This component is depended on by:** AssistantPage	
Responsibilities: - 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:** - ui/react - ui/thread

React Component: AssistantPage	
This component depend on: assistant **This component is depended on by:** Dashboard	
Responsibilities: - Render assistant component as the main content of the page	**Collaborator:** -

React Component: LoadingPage	
This component depend on: N/A **This component is depended on by:** N/A	
Responsibilities: - Display a full-screen loading state - Center a spinner component on the page	**Collaborator:** - ui/spinner

React Component: Home	
This component depend on: TimetableCard, TimetableCompareButton, TimetableCreateNewButton **This component is depended on by:** Dashboard	
Responsibilities: - 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:** - ui - lucide-react

React Component: TimetableCard	
This component depend on: TimetableCardKebabMenu **This component is depended on by:** Home	
Responsibilities: - 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:** - ui/button - ui/input - lucide-react - react - react-router-dom

React Component: TimetableCardKebabMenu	
This component depend on: timetablesController **This component is depended on by:** TimetableCard	
Responsibilities: - 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:** - ui/button - react-router-dom - ui/dropdown-menu - ui/dialog - lucide-react - api/timetableSlice

React Component: TimetableCompareButton	
This component depend on: timetablesController **This component is depended on by:** Home	
Responsibilities: - Display a button to open the timetable comparison dialog	**Collaborator:** - ui/button - ui/dialog

- Provide a UI for selecting two timetables to compare - Allow users to cancel or proceed with the comparison	- ui/input - ui/label

React Component: TimetableCreateNewButton	
This component depend on: timetablesController **This component is depended on by:** Home	
Responsibilities: - Display a button to create a new timetable	**Collaborator:** - ui/button - lucide-react - react-router-dom

React Component: Calendar	
This component depend on: N/A **This component is depended on by:** TimetableBuilder	
Responsibilities: - 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:** - @schedule-x/react - @schedule-x/calendar - @schedule-x/drag-and-drop - @schedule-x/event-modal - @schedule-x/theme-default/dist/index.css - ui/button - ui/dialog - ui/input - ui/label - react-hook-form - api/timetableApiSlice - api/restrictionsApiSlice - api/coursesApiSlice - api/eventsApiSlice - api/offeringsApiSlice - react-router-dom - models (timetable-form) - utils (type-utils, semester-utils)

React Component: ViewCalendar	
This component depend on: N/A **This component is depended on by:** CompareTimetables, Home	
Responsibilities: - Renders a timetable that was shared with the user - Lists the courses and daily + weekly + monthly schedule that the shared timetable consists of using a calendar wrapped inside of a dialog	**Collaborator:** - @schedule-x/react - @schedule-x/calendar - @schedule-x/event-modal - react - api/eventsApiSlice - api/authApiSlice - utils (type-utils, semester-utils, calendar-utils) - constants (calendarConstants) - ui/spinner

React Component: GeneratedCalendar	
This component depend on: N/A **This component is depended on by:** TimetableBuilder	
Responsibilities: - Render generated timetables after user submits timetable generate form with provided courses and restrictions - Call backend endpoint to return possible timetables - 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:** - @schedule-x/react - @schedule-x/calendar - @schedule-x/drag-and-drop - @schedule-x/event-modal - @schedule-x/theme-default/dist/index.css - ui/button - ui/dialog - ui/input - ui/label - react-hook-form - api/timetableApiSlice - api/restrictionsApiSlice - api/coursesApiSlice - api/eventsApiSlice - api/offeringsApiSlice - react-router-dom - models (timetable-form) - utils (type-utils, semester-utils)

React Component: OfferingInfo	
This component depend on: N/A **This component is depended on by:** Calendar	
Responsibilities: - Displays a block allowing users to manage meeting section times (e.g. LEC, TUT, and PRA times) for a specific offering from a specific semester - Provides a dropdown for selecting lecture, tutorial, and practical sections	**Collaborator:** - api/offeringsApiSlice - ui/select - models (models, timetable-form) - react - lucide-react - react-hook-form - zod

React Component: TimetableCompareItem	
This component depend on: N/A **This component is depended on by:** CompareTimetables, TiemtableCompareButton	
Responsibilities: - Renders an item in a Select that lists out the timetable's name and the owner of the timetable, allowing for better modularity when used with other components such as CompareTimetables and TimetableComparebutton	**Collaborator:** - semester-icon - ui/select - utils (type-utils) - ui/badge - api/authApiSlice - react

React Component: EditAccountDialog
This component depend on: N/A

This component is depended on by: Dashboard	
Responsibilities: - Displays a Dialog to allow to user to edit their account information such as their username	**Collaborator:** - react - ui/dialog - ui/button - ui/input - ui/label - api/authApiSlice

React Component: TimetableSuccessDialog	
This component depend on: N/A **This component is depended on by:** ShareDialog	
Responsibilities: - Displays a success message regarding a CRUD action applied to a timetable	**Collaborator:** - ui/button - ui/dialog - react

React Component: ShareDialog	
This component depend on: TimetableSucessDialog **This component is depended on by:** TimetableCardKebabMenu, TimetableBuilder	
Responsibilities:	**Collaborator:**

- Displays a dialog allowing the users to specify who to share their timetable with - Provides an email input field for the user to specify the email of the account that they would like to share to	- ui/button - ui/dialog - ui/input - ui/label - ui/spinner - react - api/authApiSlice

React Component: TimetableCardShareKebabMenu	
This component depend on: TimetableErrorDialog **This component is depended on by:** TimetableCard	
Responsibilities: - Provide a dropdown menu to allow users to configure their shared timetables - Displays a confirmation dialog when the user wants to remove a shared timetable - Handle shared timetable deletion and refreshes the shared timetables list after deletion	**Collaborator:** - ui/button - ui/dropdown-menu - ui/dialog - lucide-react - api/sharedApiSlice - react

Component: EmailNotificationSettings	
This component depend on: timetablesController **This component is depended on by:** TimetableCard	
Responsibilities: - Shows current status of timetable's email notification	**Collaborator:** - React - shadcn/ui

enabled - Allow user to toggle and save this status - Sends update request if status updated	- timetableApiSlice

Component: CompareTimetables	
This component depend on: timetablesController **This component is depended on by:** Dashboard	
Responsibilities: - Gets url params for timetable ids and user ids - Gets timetables and events based on timetable and user ids - Renders two timetables side by side - Renders dropdown menus to select timetables to compare - Renders Compare button that when clicked triggers recompare	**Collaborator:** - React - shadcn/ui - timetableApiSlice - TimetableCompareItem - ViewCalendar

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 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:
	- 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: - 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 - Gets the total number of meeting sections for a list of courses. For example, if a course has lectures, tutorials, and practicals, then that course would have 3 meeting sections.	**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 - Process and return the department data as a JSON response - Handle errors and return appropriate HTTP status codes	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

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: timetableController	
This component depend on: N/A **This component is depended on by:** TimetableCardKebabMenu, TimetableCompareButtons, TimetableCreateNewButton, EmailNotificationSettings	

Responsibilities:
- Handle HTTP Request received from frontend:
  - Create new timetable
  - Get (fetch) all timetables belong to the current user in session
  - Update timetable
  - Delete timetable
	Collaborator:
	- express
	- middleware/asyncHandler
	- db/setupDb

Component: eventController
This component depend on: N/A **This component is depended on by:** Calendar

Responsibilities:
- Find the next occurrence date based on week day of a lecture
- Generate weekly events based on week day of 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:
  - Add new events
  - 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)
	Collaborator:
	- express
	- middleware/asyncHandler
	- db/setupDb

Component: restrictionController	
This component depend on: N/A **This component is depended on by:** N/A	
Responsibilities: - 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	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

database - Return appropriate HTTP responses based on success or failure	

Component: generatorController	
This component depend on: N/A **This component is depended on by:** availableFunctions	
Responsibilities: - Handle timetable generation based on input constraints - Validate user permissions before performing any action - Processes and filters out based on given restrictions. - Generates a series of timetables using the processed data - Return appropriate HTTP responses based on success or failure	**Collaborator:** - express - middleware/asyncHandler - db/setupDb

Component: aiController	
This component depend on: N/A **This component is depended on by:** assistant	
Responsibilities: - Handle incoming chat requests from the front end - Process user message and interact with OpenAI's GPT-4o-mini model - Use function calling to interact with system functions - Refine user query before calling vector database - Call vector database to perform similarity search on refined user query	**Collaborator:** - express - middleware/asyncHandler - @ai-sdk/openai - ai

- Stream AI-generated responses back to the client - Enforce system instructions to keep responses relevant to course selection and timetabling	

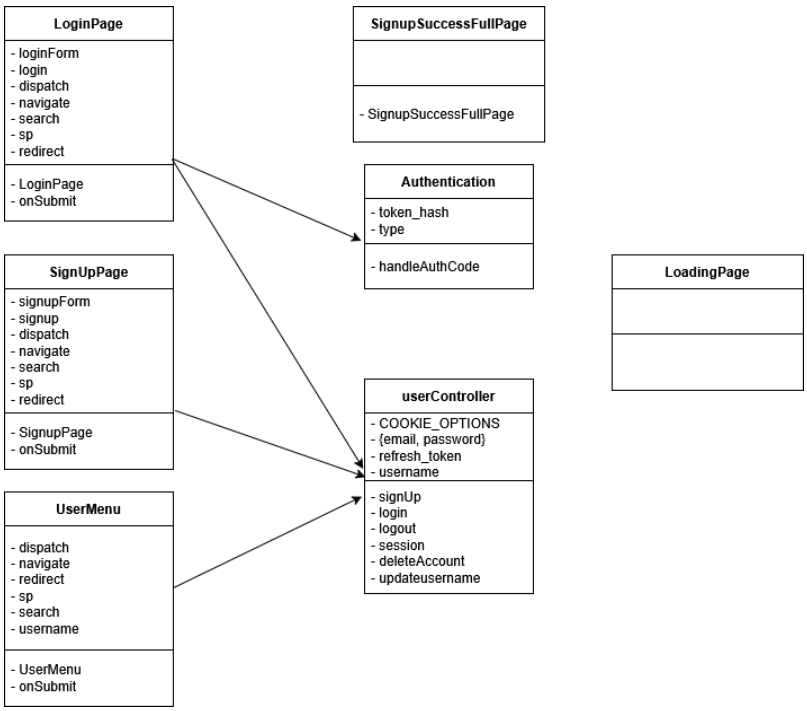
Component: availableFunctions	
This component depend on: generatorController **This component is depended on by:** t	
Responsibilities: - Provides function definitions for tool calls that AI uses to manipulate timetables - generateTimetables uses helper functions for generatorController to generate possible timetables and then chooses one to save in database - updateTimetable updates timetable name or semester - deleteTimetable deletes a timetable - getTimetables gets multiple timetables - getCourses gets course information	**Collaborator:** - supabase - generatorTypes - generatorHelpers - eventsController - express

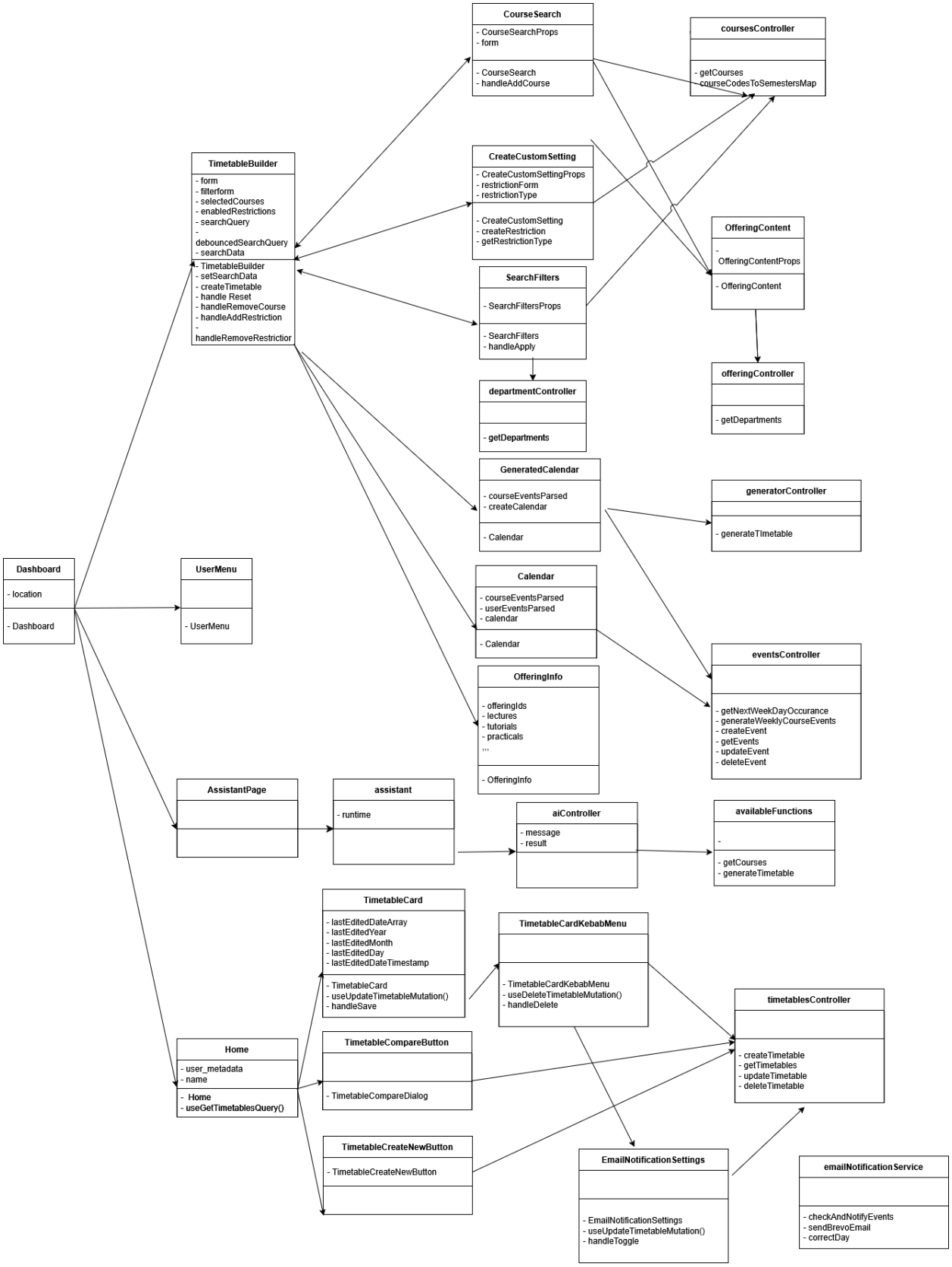
Component: sharesController	
This component depend on: N/A **This component is depended on by:** N/A	
Responsibilities: - Handle CRUD operation for shares tables to manage timetable sharing between users - Validate requested data - Ensure that sharing is eligible: not duplicated and not to the current user - Retrieve shared user's ID based on their email	**Collaborator:** - express - asyncHandler - supabase

- Check calendar eligibility for sharing	

Component: emailNotificationService	
This component depend on: **This component is depended on by:**	
Responsibilities: - Provides function checkAndNotifyEvents which is ran by server during cron job - Checks for course events happening in next 15 min - If a course event is valid and is happening in next 15 min, then notify user by sending HTTP request to Brevo SMTP which then sends email to user - Ensure day and times are valid and that email is successfully sent	**Collaborator:** - Axios - supabase

Component Relationship Diagram




CourseSearch
- CourseSearchProps
- form
- CourseSearch
- handleAddCourse
coursesController
- getCourses
courseCodesToSemestersMap
TimetableBuilder
- form
- filterform
- selectedCourses
- enabledRestrictions
- searchQuery
- debouncedSearchQuery
- searchData
- TimetableBuilder
- setSearchData
- createTimetable
- handle Reset
- handleRemoveCourse
- handleAddRestriction
- handleRemoveRestrictior
CreateCustomSetting
- CreateCustomSettingProps
- restrictionForm
- restrictionType
- CreateCustomSetting
- createRestriction
- getRestrictionType
OfferingContent
- OfferingContentProps
- OfferingContent
SearchFilters
- SearchFiltersProps
- SearchFilters
- handleApply
offeringController
- getDepartments
departmentController
- getDepartments
GeneratedCalendar
- courseEventsParsed
- createCalendar
- Calendar
generatorController
- generateTImetable
Dashboard
- location
- Dashboard
UserMenu
- UserMenu
Calendar
- courseEventsParsed
- userEventsParsed
- calendar
- Calendar
eventsController
- getNextWeekDayOccurance
- generateWeeklyCourseEvents
- createEvent
- getEvents
- updateEvent
- deleteEvent
OfferingInfo
- offeringIds
- lectures
- tutorials
- practicals
- OfferingInfo
AssistantPage
assistant
- runtime
aiController
- message
- result
availableFunctions
- getCourses
- generateTimetable
TimetableCard
- lastEditedDateArray
- lastEditedYear
- lastEditedMonth
- lastEditedDay
- lastEditedDateTimestamp
- TimetableCard
- useUpdateTimetableMutation()
- handleSave
TimetableCardKebabMenu
- TimetableCardKebabMenu
- useDeleteTimetableMutation()
- handleDelete
timetablesController
- createTimetable
- getTimetables
- updateTimetable
- deleteTimetable
Home
- user_metadata
- name
- Home
- useGetTimetablesQuery()
TimetableCompareButton
- TimetableCompareDialog
TimetableCreateNewButton
- TimetableCreateNewButton
EmailNotificationSettings
- EmailNotificationSettings
- useUpdateTimetableMutation()
- handleToggle
emailNotificationService
- checkAndNotifyEvents
- sendBrevoEmail
- correctDay

Dependencies

Front-End

```
"@ai-sdk/openai": "^1.1.13",
 "@assistant-ui/react": "^0.7.91",
 "@assistant-ui/react-ai-sdk": "^0.7.16",
 "@assistant-ui/react-markdown": "^0.7.20",
 "@assistant-ui/react-ui": "^0.1.7",
 "@hookform/resolvers": "^3.10.0",
 "@radix-ui/react-accordion": "^1.2.3",
 "@radix-ui/react-avatar": "^1.1.3",
 "@radix-ui/react-checkbox": "^1.1.4",
   "@radix-ui/react-dialog": "^1.1.6",
   "@radix-ui/react-dropdown-menu": "^2.1.6",
   "@radix-ui/react-hover-card": "^1.1.6",
   "@radix-ui/react-label": "^2.1.2",
   "@radix-ui/react-select": "^2.1.6",
   "@radix-ui/react-separator": "^1.1.2",
   "@radix-ui/react-slot": "^1.1.2",
   "@radix-ui/react-switch": "^1.1.3",
   "@radix-ui/react-tooltip": "^1.1.8",
   "@reduxjs/toolkit": "^2.5.1",
   "@schedule-x/drag-and-drop": "^2.21.1",
   "@schedule-x/event-modal": "^2.21.1",
   "@schedule-x/events-service": "^2.21.0",
   "@schedule-x/react": "^2.21.0",
   "@schedule-x/scroll-controller": "^2.23.0",
   "@schedule-x/theme-default": "^2.21.0",
   "ai": "^4.1.45",
   "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-hook-form": "^7.54.2",
   "react-redux": "^9.2.0",
   "react-router-dom": "^7.1.3",
   "redux-mock-store": "^1.5.5",
   "remark-gfm": "^4.0.1",
   "tailwind-merge": "^2.6.0",
   "tailwindcss": "^3.4.17",
   "tailwindcss-animate": "^1.0.7",
   "zod": "^3.24.1"
```

Back-End

```
"@ai-sdk/openai": "^1.1.13",
 "@getbrevo/brevo": "^2.2.0",
 "@langchain/community": "^0.3.32",
 "@langchain/openai": "^0.4.4",
 "@langchain/pinecone": "^0.1.3",
 "@pinecone-database/pinecone": "^5.0.2",
 "@supabase/supabase-js": "^2.48.1",
 "@types/express": "^5.0.0",
 "@types/node-fetch": "^2.6.12",
   "ai": "^4.1.45",
   "axios": "^1.8.4",
   "cookie-parser": "^1.4.7",
   "cors": "^2.8.5",
   "csv-parser": "^3.2.0",
   "d3-dsv": "^2.0.0",
   "dotenv": "^16.4.7",
   "express": "^4.21.2",
   "langchain": "^0.3.19",
   "node-cron": "^3.0.3",
   "node-fetch": "^2.7.0",
   "openai": "^4.85.4",
   "pdf-parse": "^1.1.1",
   "stream": "^0.0.3",
   "swagger-jsdoc": "^6.2.8",
   "swagger-ui-express": "^5.0.1",
   "validator": "^13.12.0"
```

Dev-Dependencies

Front-End

```
"@eslint/js": "^9.17.0",
 "@jest/globals": "^29.7.0",
 "@testing-library/jest-dom": "^6.6.3",
 "@testing-library/react": "^16.0.0",
 "@tsconfig/node20": "^20.1.4",
 "@types/jest": "^29.5.14",
 "@types/node": "^22.10.10",
 "@types/react": "^18.3.18",
 "@types/react-dom": "^18.3.5",
   "@types/redux-mock-store": "^1.5.0",
   "@types/testing-library__react": "^10.2.0",
   "@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",
    "identity-obj-proxy": "^3.0.0",
    "jest": "^29.7.0",
    "jest-environment-jsdom": "^29.7.0",
    "postcss": "^8.5.1",
    "ts-jest": "^29.2.3",
    "ts-node": "^10.9.2",
    "typescript": "~5.6.2",
    "typescript-eslint": "^8.18.2",
    "vite": "^6.0.5",
    "vitest": "^3.0.9"
```

Backend

```
  "@testing-library/jest-dom": "^6.4.8",
   "@testing-library/react": "^16.0.0",
   "@types/cookie-parser": "^1.4.8",
   "@types/cors": "^2.8.17",
   "@types/jest": "^29.5.14",
   "@types/node-cron": "^3.0.11",
   "@types/supertest": "^6.0.2",
   "@types/swagger-jsdoc": "^6.0.4",
   "@types/swagger-ui-express": "^4.1.7",
    "identity-obj-proxy": "^3.0.0",
    "jest": "^29.7.0",
    "jest-environment-jsdom": "^29.7.0",
    "supertest": "^7.0.0",
    "ts-jest": "^29.2.6",
    "ts-node": "^10.9.2"
```

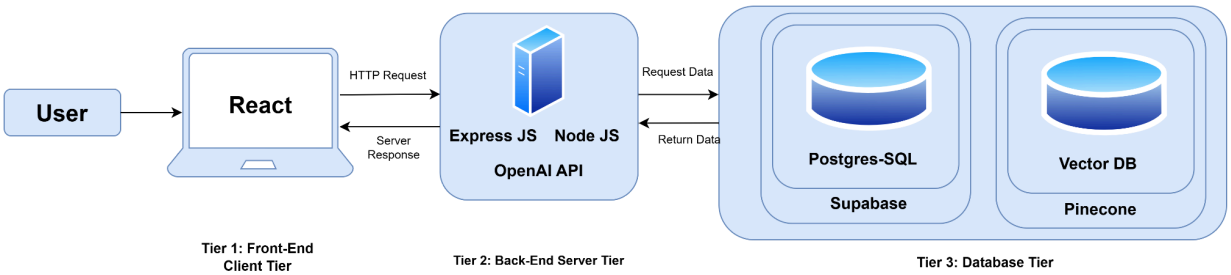
Software Configuration Summary

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.

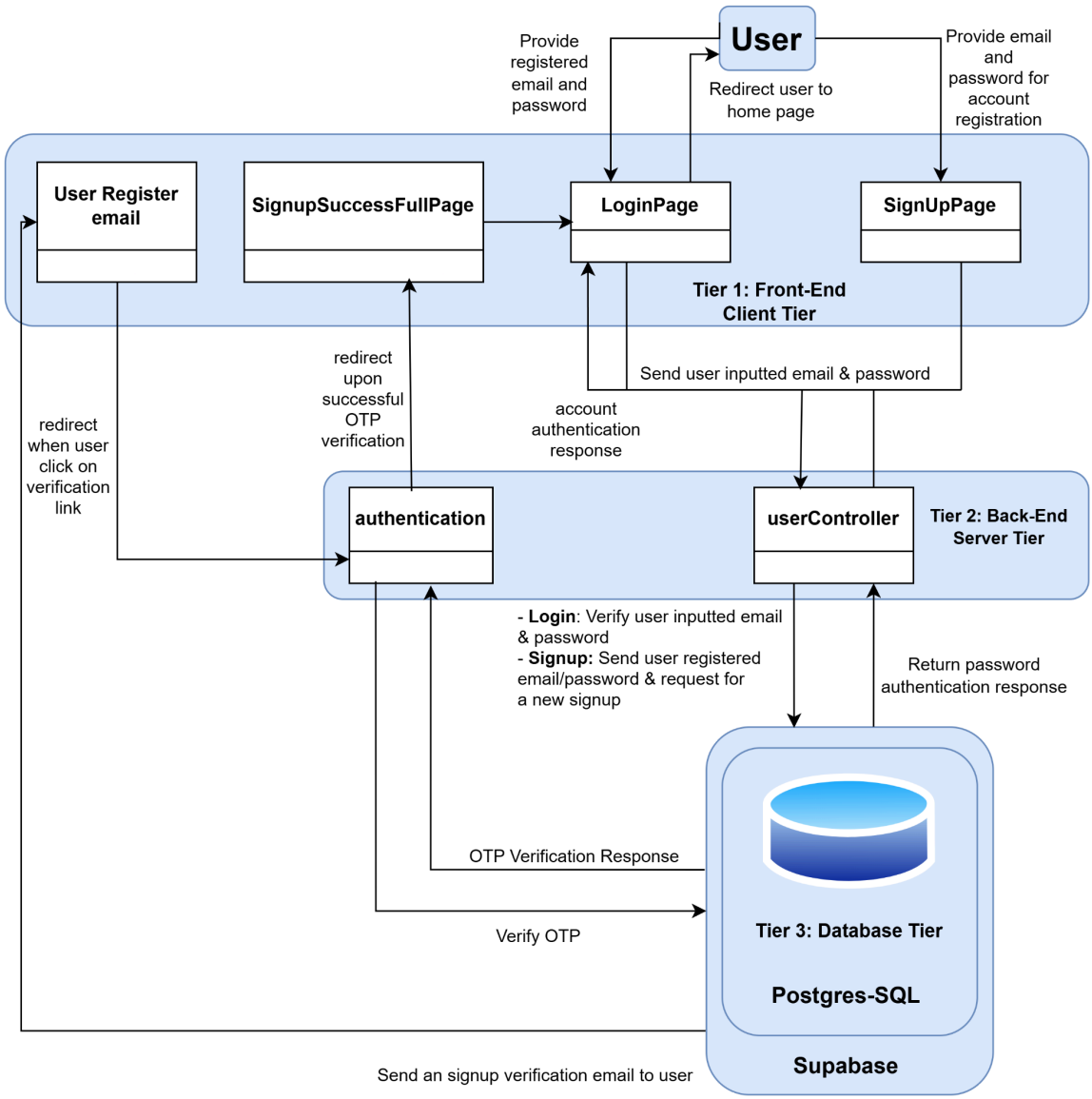
- If the updated password and/or username are acceptable (nonempty less than 50 characters long), the system will update the database information and user session accordingly.
- If the account update fails due to either incorrect or unacceptable user credentials, userController returns **error code 400**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

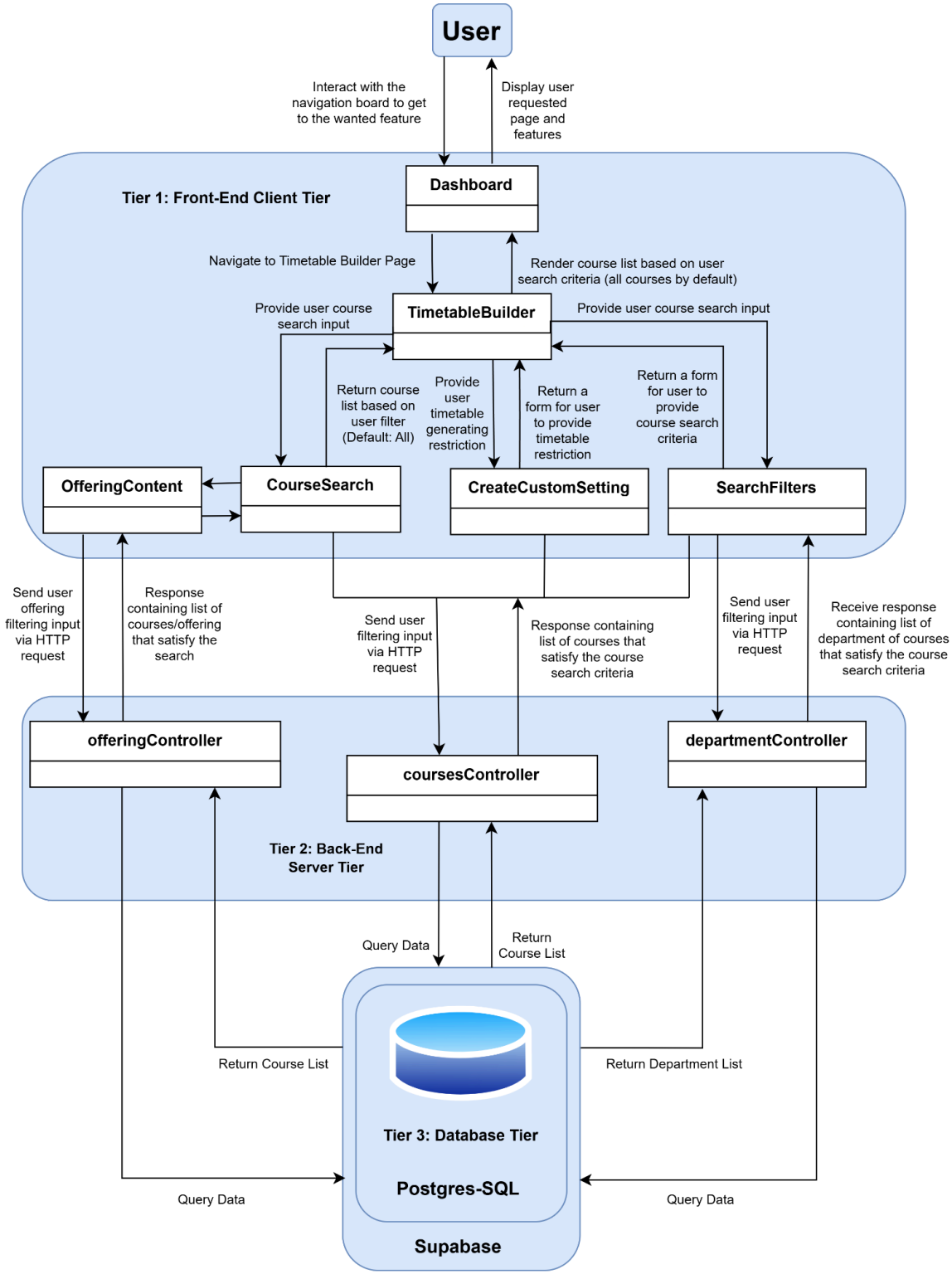
The **departmentsController** handles all department-related data displayed in the filter form. When users interact with department-related filters, the frontend sends a request to retrieve available departments.

- 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."


User
Interact with the navigation board to get to the wanted feature
Display user requested page and features
Tier 1: Front-End Client Tier
Dashboard
Navigate to Timetable Builder Page
Render course list based on user search criteria (all courses by default)
Provide user course search input
TimetableBuilder
Provide user course search input
Return course list based on user filter (Default: All)
Provide user timetable generating restriction
Return a form for user to provide timetable restriction
Return a form for user to provide course search criteria
OfferingContent
CourseSearch
CreateCustomSetting
SearchFilters
Send user offering filtering input via HTTP request
Response containing list of courses/offering that satisfy the search
Send user filtering input via HTTP request
Response containing list of courses that satisfy the course search criteria
Send user filtering input via HTTP request
Receive response containing list of department of courses that satisfy the course search criteria
offeringController
coursesController
departmentController
Tier 2: Back-End Server Tier
Query Data
Return Course List
Return Course List
Return Department List
Tier 3: Database Tier
Postgres-SQL
Supabase
Query Data
Query Data

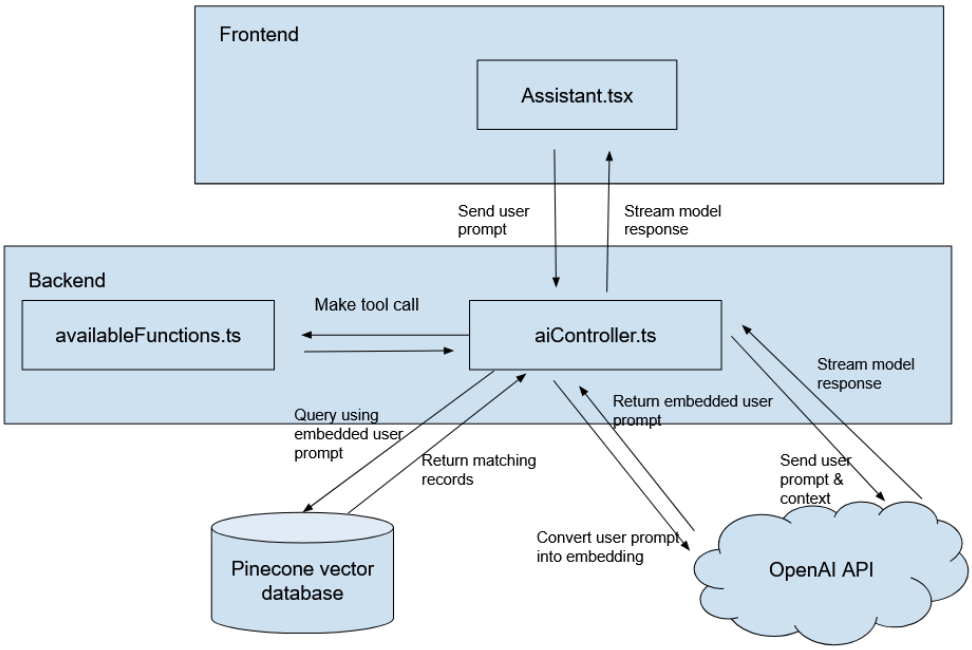
3. AI Workflow

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 first checks if the user has used the special command /timetable in their prompt. If so , it performs function calls using their prompt and available tools. Tool calls are defined in **availableFunctions** and have specific input schemas that the AI must adhere to when making tool calls. Results are returned as messages to the user through SSE. If the command is not given, then the regular AI chatbot behavior takes over which first 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.

The OfferingInfo component is used to populate available lectures, tutorials, and practicals for courses in the timetable. It fetches course offerings and filters them based on the offering type (LEC, TUT, PRA)

Populate the Calendar with Course Offerings:

When a timetable is opened or created, OfferingInfo loads user selected lecture, tutorial and practical from the filter form response onto an interim calendar. From here users can trigger the

option to change course sections using the dropdown boxes (SelectTrigger, SelectContent). The selected section will be changed automatically on the interim calendar for better visualization.

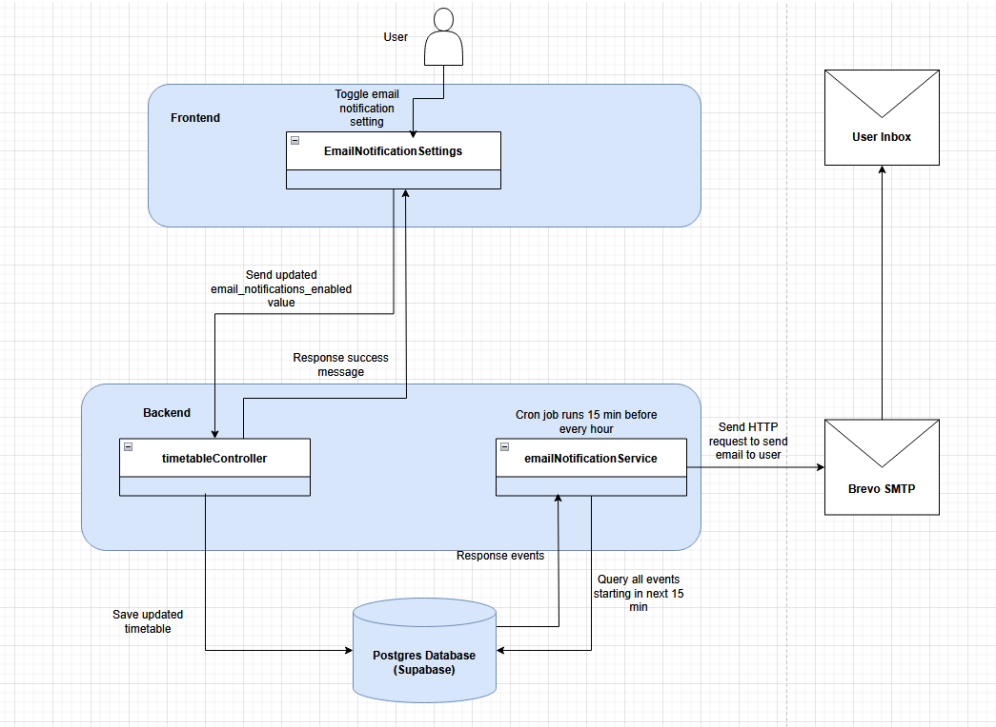
Once finished users can save the calendar and the selected option will be saved in the database as a courseEvent.

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

5. Email Notifications

Users can enable/disable email notifications per timetable. If email notifications are enabled for a given timetable, the user will receive emails 15 min before course events of that timetable. When a user chooses to enable email notifications in the settings panel **(EmailNotificationSettings.tsx)**, an HTTP request is sent to **timetablesController.updateTimetable()** with the new email notification enabled field set as true.

On the server side, a cron job is run in our express server 15 min before every hour. In **emailNotificationService.ts,** each scan checks for course events starting in the next 15 min and for each of those events. For each course event, If the user is not found or an error occurred fetching offerings/events, then the next course event is checked. If a course event is deemed valid and is happening within the next 15 minutes, an email request is sent to Brevo (3rd party SMTP server) to deliver an email to the user whose event is starting soon.**400-series** errors are caught and an error response is logged, indicating if something went wrong with sending the email. Otherwise the email is successfully sent and logged on the server.



6. Timetable generation

Users will be able to generate timetables based on their user constraint via an HTTP request to the **generatorController**. The generation is a multistep process, as follows:

1. In **TimetableBuilder,** the user selects courses and restrictions. They then click the Generate button to trigger an HTTP POST request to **generatorController.generateTimetable,** passing in the request body the timetable name, semester, courses, and restrictions
2. In **generatorController**, first all offerings for all relevant courses will be fetched through **getOfferings**.
3. Then, all offerings are grouped together via function **groupOfferings** by their `course_id` and `meeting_section` since sometimes, a single lecture can happen in multiple days and they must be chosen together.
4. Next, invalid offerings will be filtered out with function **getValidOfferings**, this will check all offerings to see if they satisfy all “Restrict Before”, “Restrict After”, “Restrict Between” and “Restrict Day” restrictions.
5. Then, offerings will be categorized into lectures, tutorials and practicals via **categorizeValidOfferings** and one of each will be chosen by the algorithm, if it exists.
6. **getValidSchedules** will be called to implement a backtracking algorithm, searching for all valid schedules.
    a. Base case: if all courses have been considered, then we check to see if it violates any “Restrict Days Off” or “Max Gap” restrictions, if not, add it to the list of valid timetables, furthermore, if the size of the valid timetables is over 200, then exit the function immediately.
    b. Iterate through all elements in the current list, if said element can be inserted into the current schedule, we insert it into the stack and use recursion to insert the next series of offerings.
    c. After the recursion, we pop the stack and try the next element in the list.
    d. Lastly, after the loop has been exited, we exit the function.
7. We have the result generated, if the result is empty, then no valid timetables has been generated (**404 Not Found**).
8. Otherwise, we return the number of valid timetables found, alongside with the timetables themselves.
    a. If there are more than 10 valid timetables, then a random subset of 10 timetables will be returned instead.
9. **Timetable Builder** will then render **GeneratedCalendar** which provides a calendar UI for all the generated timetables. The user can toggle between generated timetables until they find one that they like
10. The user can then save the timetable by pressing the save button.They then name the timetable and make a POST request to **timetableController.create** which saves the timetable

7. Timetable Compare

Users are able to compare timetables in a side-by-side view by selecting two timetables including timetables shared to them by other users. The process is as follows:

1. In **Home.tsx**, the user can click on the Compare Timetables button (**TimetableCompareButton.tsx)** which opens a dialog menu prompting them to select two timetables. The list of timetables is obtained from making a GET HTTP request to /api/timetables to get a list of all timetables that the user has viewing access to
2. After the user clicks submit, they are redirected to `` `/dashboard/compare?id1=${timetableId1}&id2=${timetableId2}&userId1=${userId1}&userId2=${userId2}` `` which renders **CompareTImetables.tsx**. URL params are parsed to obtain the two timetable ids and their respective user's id.
3. **CompareTImetables.tsx** then renders **ViewCalendar.tsx** which gets via HTTP request all the timetable events for the two timetables and renders them on the calendar
4. **400-series errors** such as **404 Bad request** or **401 Unauthorized** are propagated from **timetableController** in the backend. Otherwise, if the data from the database is returned without error then a **200** status code is sent.
5. The user can then switch timetables on **CompareTImetables.tsx** by navigating the dropdown menus, selecting timetables, and clicking the compare button again

8. Timetable Share

Users are able to share timetables with other users by selecting the email of the account that the user would like to share their timetable with. The process is as follows:

1. In **Home.tsx**, the user can see their owned timetables by going on the “Mine” tab
2. After the user clicks on the kebab menu for one of their owned timetables, there will be an option named “Share Timetable”
3. After clicking on the “Share Timetable” button, a dialog (**ShareDialog.tsx)** will open, showing an email input field where the user can enter the email of the person that they want to share their timetable with.
4. If the email that the user entered is valid and is not equal to the email of the current account (because users cannot share timetables with their own accounts), then after the user clicks Share, the dialog (**ShareDialog.tsx)** will close and a new dialog **(TimetableSuccessDialog.tsx)** will pop up showing the success message “Timetable shared successfully!”. A POST HTTP request will be made to /api/timetables/shared to share the user’s selected timetable with the other account.
5. The list of timetables that the user has shared with others is obtained from making a GET HTTP request to /api/timetables/shared/owner and the list of timetables that has been shared with the user is obtained from making a GET HTTP request to /api/timetables/shared/me. All of the timetables that have been shared with the user are listed in the “Shared With Me” tab from the Home Page.

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 application's latest functional version.

Continuous Integration and Deployment

Whenever a push is made to our project's develop repo it is then unit tested (both frontend and backend) using github actions. If the new pushed changes pass all unit and integration tests it is then pushed forwards in the CI/CD pipeline. A new docker image of the application is created (with the pushed changes) and then stored in docker hub with a version tag. After that the new image is pushed to our Google Cloud virtual machine and deployed.