Tech Stack

Frontend:

- Typescript
- React
- Vite

Backend:

- Node.js

State Management:

- Redux

Sprint 1

Scope:

File Structure Created

Searching:

- User can type in a search box and see class suggestions in a dropdown

Dropdown:

- After the user searches display the results in a dropdown (fake data at this point)

Selection:

- Selecting a class from the dropdown appends it to the selected list

File Structure Breakdown:

The core application directory is “SmartScheduler” which contains configuration files and the following directories: “public” which contains the Vite logo in vite.svg, “server” which has the backend entry point with index.js, and “src” which is the main application source.

Configuration files:

- package.json / package-lock.json: define dependencies and scripts
- vite.config.ts: configuration for the Vite build tool
- tsconfig*.json: TypeScript compiler settings
- eslint.config.js: defines linting rules for code style and quality
- index.html: root HTML entry for the web app

src:

Contains the key files App.tsx, main.tsx, and types.ts as well the following directories: assets, components, features, redux, styles, and utils

- App.tsx: main application component
- main.tsx: app bootstrap file
- types.ts: shared TypeScript types

assets:
- Holds images, SVGs, and other media

components:
- ClassCard.tsx: displays individual class info in left column

- Grid.tsx: layout for schedule visualization

- Header.tsx: top navigation or app banner

- ScheduleCard.tsx, UnscheduledCard.tsx: cards for scheduled and unscheduled classes

- SearchBar.tsx: course search input

- SelectedCourses.tsx, TotalCreditHours.tsx: displays user-selected courses and totals

- UnscheduledTable.ts: classes with no scheduled time

features:
- scheduleSlice.ts: a Redux slice for managing schedule-related state

redux:
- store.ts: configures the Redux store
- hooks.ts: provides typed hooks for Redux operations

styles:
- App.css, HeaderStyles.css, SearchBarStyles.css: scoped styling for UI consistency

utils:
- getCourses.ts: helper for fetching or filtering course data

Components:

Search and Selection Loop:

-Handle events: The user types a course name into the SearchBar component. Input events trigger calls to the SearchServer, which gets course data.

-Update state: The returned course data is dispatched through Redux

Actions, updating the Redux State store with new search results and selected course data.

Search:

SearchBar:

- Manages search input, dropdown visibility, and user interactions
- Updates Redux state with the results through dispatched actions
- Handles UI logic for showing/hiding dropdowns

SearchServer:

-Backend proxy between frontend and university

Selection and State Management:

ReduxAction:

- Manages global state changes triggered by the SearchBars
- Handles updating selected class lists.

Redux State:

- Stores all selected courses
- Supplies state context to components like SelectedClasses via Redux hooks