Course Name: EECS 2311 Z
Team Number: 8
Student:
Lukasz Nowosad 220250403

Date: March 24, 2025

User Story	Main Developers	Assigned Tester
Creating a group	Haaris / Kashif	Farzin
Viewing course Schedules	Marko Zovic	Lukasz / Kashif
Chat	Farzin	Abdullah
Create an account	Lukasz / Abdullah	Marko / Haaris

Manual Test Cases:

Searching up a course outside of Lassonde/Bethune:

- Try searching up a language course such as GER 1000
- Error message shows up as expected: “No courses found with the given parameters.”

Searching up a course within Lassonde Bethune:

- Try searching for any course within science/engineering
- Shows up as expected, make sure you enter everything correctly

Searching up a course that doesn’t exist:

- Try inputting random letters into the input fields
- Behaves as expected, gives an error: “No courses found with the given parameters.”

Inputting /schedule into the url:

- Make sure you are logged out
- This shouldn't take you to the schedule page if you are not logged in
- Bug: currently /schedule can be accessed whilst being logged out

Running the courses web scraper:

- Run the web scraper, make sure you have:
    - Python 3.x (Ensure it's installed)
    - Google Chrome (latest version)
    - Chrome WebDriver (same version as Chrome)
    - Python dependencies:
        - Selenium
        - beautifulsoup4
- Works as expected mostly
- Bug: first entry is a basic input(course: course, time: time)

The Problem Report Form

- Problem report number: PR-001
- Reported by: Lukasz
- Date reported: March 24, 2025
- Programname: Course Search Feature
- Release number: v2.0.0
- Configuration(s): All browsers
- Report type: Functionality bug
- Can reproduce: Yes
- Severity: Medium
- Priority: High
- Problem summary: /schedule accessible while logged out
- Problem description and how to reproduce it:

1. Navigate to the application URL.
2. Ensure you are logged out.
3. Type /schedule in the URL bar.
4. The page incorrectly loads instead of redirecting to the login page.

- Suggested fix: Implement authentication check before accessing the `/schedule` route.
- Status: Open
- Resolution: Pending
- Resolved by: [To be determined]

The Problem Report Form

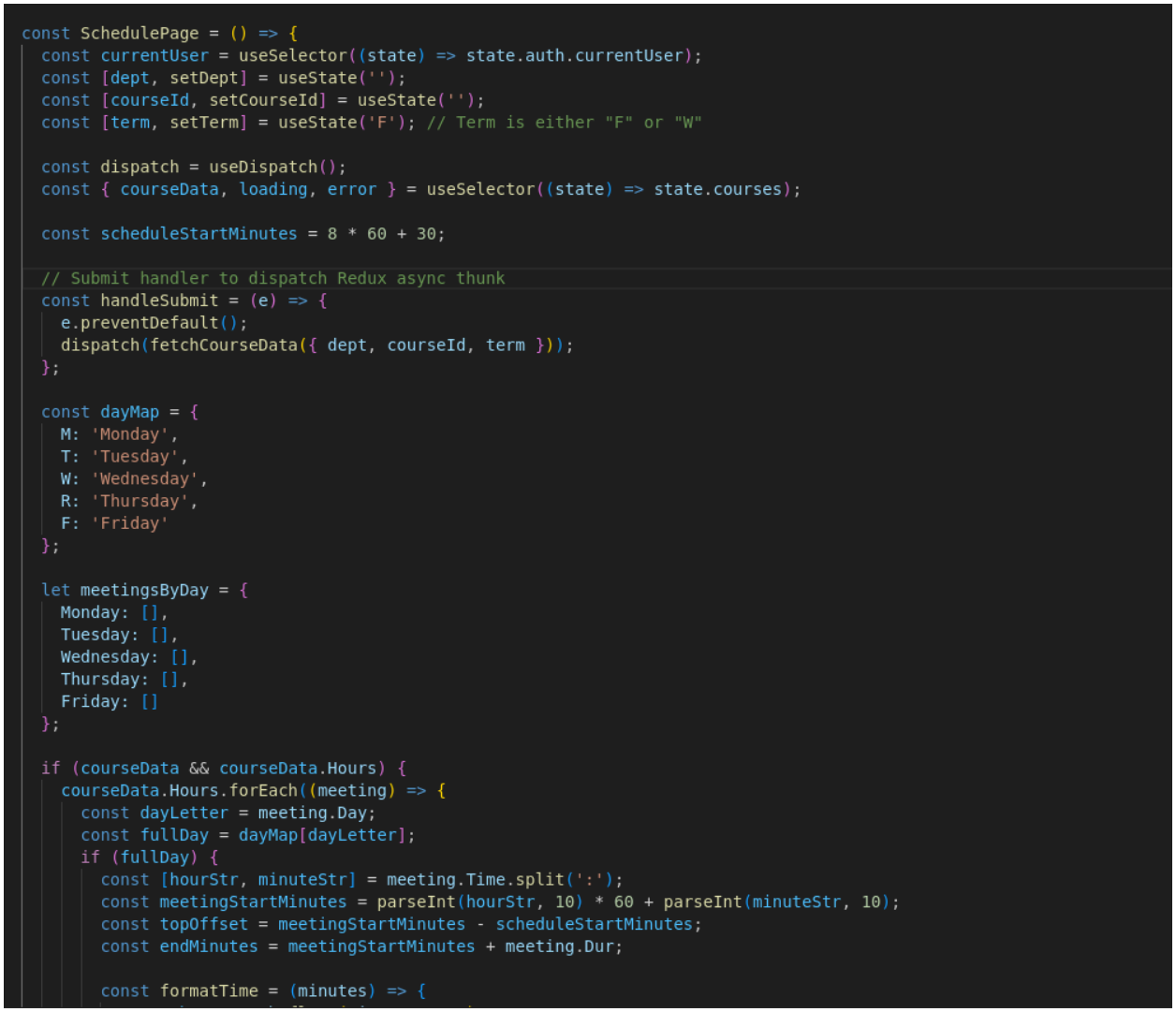
- Problem report number: PR-002
- Reported by: Lukasz
- Date reported: March 25, 2025
- Program (or component) name: Web Scraper
- Release number: v2.0.0
- Configuration(s): Python 3.x, Google Chrome (latest), Chrome WebDriver (latest)
- Report type: Data handling bug
- Can reproduce: Yes
- Severity: Low
- Priority: Medium
- Problem summary: First entry in scraper output shows "course: course, time: time"
- Problem description and how to reproduce it:

1. Run the web scraper with appropriate dependencies installed.
2. Observe the first entry in the output.
3. The first entry consistently shows placeholder data: "course: course, time: time."

- Suggested fix: Ensure the scraper correctly skips empty or default entries before populating data.

- Status: Open
- Resolution: Pending
- Resolved by: [To be determined]

Code Review:



```
const SchedulePage = () => {
  const currentUser = useSelector((state) => state.auth.currentUser);
  const [dept, setDept] = useState('');
  const [courseId, setCourseId] = useState('');
  const [term, setTerm] = useState('F'); // Term is either "F" or "W"

  const dispatch = useDispatch();
  const { courseData, loading, error } = useSelector((state) => state.courses);

  const scheduleStartMinutes = 8 * 60 + 30;

  // Submit handler to dispatch Redux async thunk
  const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(fetchCourseData({ dept, courseId, term }));
  };

  const dayMap = {
    M: 'Monday',
    T: 'Tuesday',
    W: 'Wednesday',
    R: 'Thursday',
    F: 'Friday'
  };

  let meetingsByDay = {
    Monday: [],
    Tuesday: [],
    Wednesday: [],
    Thursday: [],
    Friday: []
  };

  if (courseData && courseData.Hours) {
    courseData.Hours.forEach((meeting) => {
      const dayLetter = meeting.Day;
      const fullDay = dayMap[dayLetter];
      if (fullDay) {
        const [hourStr, minuteStr] = meeting.Time.split(':');
        const meetingStartMinutes = parseInt(hourStr, 10) * 60 + parseInt(minuteStr, 10);
        const topOffset = meetingStartMinutes - scheduleStartMinutes;
        const endMinutes = meetingStartMinutes + meeting.Dur;

        const formatTime = (minutes) => {
```

A check should be added to make sure the user is logged in. If they are not logged in, they should be sent to the starter page.

Refactoring:

1. Long Function / Code Duplication

- Smell: The logic inside the if (courseData && courseData.Hours) block is lengthy and includes complex logic for mapping meetings, formatting times, and calculating offsets.
- Solution: Extract this logic into a separate utility function called processMeetings() to improve readability and reusability.

2. Violation of Single Responsibility Principle (SRP)

- Smell: The SchedulePage component handles multiple responsibilities — fetching data, transforming data, rendering UI, and maintaining state.
- Solution: Split this component:
    - Extract data fetching logic into a custom hook (e.g., useCourseSchedule).
    - Extract the JSX rendering logic for individual days/meetings into child components (e.g., DaySchedule and MeetingBlock).

3. Inefficient Mapping

- Smell: The code unnecessarily iterates over courseData.Hours and manually maps each entry to meetingsByDay.
- Solution: Refactor this to use reduce() for better readability and efficiency.

4. Hardcoded Day Mapping

- Smell: The dayMap object is manually defined, which introduces potential maintenance issues.
- Solution: Create a utility function like getDayName() to handle day conversion dynamically.

5. CSS Class Selection Logic in Separate Function

- Smell: The getMeetingClass() function could be integrated with the JSX itself using conditional logic directly in the className attribute.
- Solution: Inline the logic directly for simpler code or replace the function with a concise mapping object.

6. Error Handling Improvement

- Smell: The error display logic only shows a message but does not guide users with actionable steps.
- Solution: Provide additional guidance or retry options in the UI.

```
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { fetchCourseData, clearCourseData } from
'../features/courseSlice';
import RedShape from './components/RedShape';
import PurpleShape from './components/PurpleShape';
import PinkShape from './components/PinkShape';
import Header from '../Header';
import "./styles/SchedulePage.css"

const SchedulePage = () => {
 const currentUser = useSelector((state) => state.auth.currentUser);
 const [dept, setDept] = useState('');
 const [courseId, setCourseId] = useState('');
 const [term, setTerm] = useState('F'); // Term is either "F" or "W"

 const dispatch = useDispatch();
 const { courseData, loading, error } = useSelector((state) =>
state.courses);

 const scheduleStartMinutes = 8 * 60 + 30;

 // Submit handler to dispatch Redux async thunk
 const handleSubmit = (e) => {
   e.preventDefault();
   dispatch(fetchCourseData({ dept, courseId, term }));
 };

 const dayMap = {
   M: 'Monday',
   T: 'Tuesday',
   W: 'Wednesday',
   R: 'Thursday',
   F: 'Friday'
 };

 let meetingsByDay = {
   Monday: [],
   Tuesday: [],
```

```
    Wednesday: [],
    Thursday: [],
    Friday: []
  };

  if (courseData && courseData.Hours) {
    courseData.Hours.forEach((meeting) => {
      const dayLetter = meeting.Day;
      const fullDay = dayMap[dayLetter];
      if (fullDay) {
        const [hourStr, minuteStr] = meeting.Time.split(':');
        const meetingStartMinutes = parseInt(hourStr, 10) * 60 +
parseInt(minuteStr, 10);
        const topOffset = meetingStartMinutes - scheduleStartMinutes;
        const endMinutes = meetingStartMinutes + meeting.Dur;

        const formatTime = (minutes) => {
          const hrs = Math.floor(minutes / 60);
          const mins = minutes % 60;
          return `${hrs.toString().padStart(2,
'0')}:${mins.toString().padStart(2, '0')}`;
        };

        meetingsByDay[fullDay].push({
          ...meeting,
          topOffset,
          endTime: formatTime(endMinutes)
        });
      }
    });
  }

  return (
    <div className="padding-container">
      <Header currentUser={currentUser} />
      <RedShape color="#58C8D7" />
      <PurpleShape color="#E6487F"/>
      <PinkShape color="#F6960A"/>
      <div className="header-container">
        <a href="/" className="back-button">&larr; Back</a>
```

```
      <h1>Schedule Viewer</h1>
    </div>
    <div className="filter-container">
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Dept"
          value={dept}
          onChange={(e) => setDept(e.target.value)}
        />
        <input
          type="text"
          placeholder="Course ID"
          value={courseId}
          onChange={(e) => setCourseId(e.target.value)}
        />
        <select value={term} onChange={(e) => setTerm(e.target.value)}>
          <option value="F">F</option>
          <option value="W">W</option>
        </select>
        <button type="submit">Show Schedule</button>
      </form>
    </div>

    {loading && <p>Loading course data...</p>}
    {error && <p className="error-message">{error}</p>}

    {courseData && (
      <div>
        <h2>
          Schedule for {courseData.Dept} {courseData["Course ID"]} -
{courseData["Course Name"]} ({courseData.Term})
        </h2>
        <div className="schedule">
          {Object.keys(meetingsByDay).map((dayName) => (
            <div className="day" key={dayName}>
              <div className="date-block">{dayName}</div>
              {meetingsByDay[dayName].map((meeting, index) => (
                <div
                  key={index}
```

```
                    className={`time-block
${getMeetingClass(meeting.Type)}`}
                    style={{
                      top: meeting.topOffset,
                      height: meeting.Dur
                    }}
                  >
                    <p>
                      {meeting.Type} {meeting.Meet ? `(${meeting.Meet})` :
''}
                    </p>
                    <p>
                      {meeting.Time} - {meeting.endTime}
                    </p>
                    <p>
                      {meeting.Campus} {meeting.Room && `Room:
${meeting.Room}`}
                    </p>
                  </div>
                ))}
              </div>
            ))}
          </div>
        </div>
      )}
    </div>
  );
};

// Helper function to choose a CSS class based on the meeting type.
function getMeetingClass(type) {
  if (type.includes('LECT')) {
    return 'lecture';
  } else if (type.includes('LAB')) {
    return 'lab';
  } else if (type.includes('TUT')) {
    return 'seminar';
  }
  return '';
}
```

```
export default SchedulePage;
```