WDD 330 Personal Project

This document serves as your final course assessment.

Introduction

Name: Diogo Rangel Dos Santos

Video Link: https://www.youtube.com/watch?v=yGlRy93cS14
Working Application Link: https://diogorangel.github.io/wdd330/week07/
GitHub Source URL: https://github.com/diogorangel/wdd330/tree/master/week07
Trello Board URL: https://trello.com/b/fyi5BbLH/wdd330-final-project

Course Outcomes

The following are the course outcomes of WDD 330:

1. Become more efficient at applying your innate curiosity and creativity.
2. Become more dexterous at exploring your environment.
3. Become a person who enjoys helping and learning from others.
4. Use a divide and conquer approach to design solutions for programming problems.
5. Finding and troubleshooting bugs you and others will have in the code you write.
6. Developing and debugging HTML, CSS, and JavaScript programs that use medium complexity web technologies.

To complete this course, you need to demonstrate your skill in these areas. Outcomes #1-5 demonstrate your personal development and are most easily shown through self-assessment and sharing experiences. Outcome #6 demonstrates your programming skill and is shown through code and experience in projects.

Skill Development Outcome

Developing and debugging HTML, CSS, and JavaScript programs that use medium complexity web technologies.

This outcome is demonstrated by your skill in the following learning objectives:

Objective	%	Description
JavaScript	25%	Robust programming logic is demonstrated. For example, validating the screen data, looping through an array of JSON data to display to the screen, creating and using events, changing element styles with JS, changing element classes to use different CSS rules.
Third-party APIs	15%	APIs are used effectively, including APIs that provide rich JSON data.
JSON	15%	Demonstrate skill processing JSON data to dynamically update the website.
CSS	15%	Appropriate use of Transforms and Transitions. For example: Add round the edges to DIV, add shadows. enlarge an input field on focus and shrink it on blur, Add borders. CSS should subtly add style to a page.
Events	15%	Use events to enhance the user experience. For example, increase the size of the input field on focus or add a shadow. React to a button click. Initialized the page with data once the onload event triggers.
Local Storage	5%	Local storage is used effectively.

These learning objectives are rated on the following scale:

Rating	Description
Unsatisfactory	Very little if any work was shown in this area.
Developing	The learning objective was shown in very basic ways.
Proficient	Effective use of the learning objective was shown in multiple places.
Mastery	Extensive use of the learning objective was shown in non-trivial ways in many places in the code.

For each learning objective, discuss how the topic was used in your application. List several examples of places where the topics are demonstrated.

The following is an example of what is expected:

Learning Objective	Description	Where can this be seen in your application?

CSS	*I spent a lot of time choosing colors that would complement each other.* *I used CSS to make the input field bigger when it received the focus and to shrink it when it lost focus.*	*This can be seen on the home screen for each input field.*
	Images are enlarged on hover.	*The recipe detail pages have this effect.*
	The search results have alternating colors for the rows for readability.	See the home page after a search is successfully run.

In the following table:

1. Describe how the topics are used.

Have someone test your links to make sure they are accessible by the grader. These links will be to your final personal project.

Feel free to add more rows to this table if needed.

Learning Objective	• **Data Validation:** Validate	**Where can this be seen in your final personal project application?**
JavaScript	**Data Validation:**	Validate user input for searching music (e.g., ensure search query is not empty, sanitize input). If allowing manual entry of Spotify track IDs, validate their format.
	Looping & Display	Loop through an array of JSON music data (your top 5 tracks or search results) to dynamically create and display music cards/list items on the screen, showing details like track name, artist, and album art.
	Events	Handle various user interactions (see "Events" section below), such as adding/removing music, playing/pausing audio previews, and searching for new music.
Third-party APIs		Parse the JSON responses received from the Spotify API.
		Extract relevant data points such as track.name, track.artists[0].name, track.album.images[0].url (for album art), and track.preview_url.
		Use this extracted JSON data to dynamically populate HTML elements (e.g., creating div elements for music cards, img tags for album art, p or span for track/artist names).
JSON		Apply border-radius and box-shadow to music cards, buttons (e.g., play/pause buttons, search button), and input fields.

		Dynamically search, update details, change element styles (e.g., highlight selected recipes), and toggle classes for different CSS rules.
		Parse the JSON responses from the recipe API. Extract relevant data points such as Author Music, image URL, Name Music, etc.
CSS	CSS (15% - Appropriate use of Transforms and Transitions, subtle style):	Rounded Edges/Shadows: Apply border-radius and box-shadow to recipe cards, buttons, and input fields.
	Button Hover: Add a subtle transform: translateY() (for a slight lift) or box-shadow transition on button hovers.	**Input Field Focus:** Enlarge the music search input field slightly on focus and shrink it back on blur using transform: scale().
	Borders: Use border properties to define sections or interactive elements, subtly adding style.	**Music Card Hover:** Implement a subtle scale or shadow transition when hovering over a music card, or make a play icon appear/fade in.
Events	**onload Event:** Initialize the page by loading your predefined "Top 5" music tracks from local storage or fetching default tracks from the Spotify API once the window.onload event triggers.	**Button Clicks:** React to a "Search" button click, "Add to Top 5" button click, "Remove from Top 5" button click, "Play Preview" button click, and "Pause Preview" button click.
	Input Events: Respond to focus and blur events on the music search input field for visual feedback.	**Dynamic Events:** Attach event listeners to dynamically created elements (e.g., the play/pause buttons on each music card).
Local Storage	**Top 5 Music List:** Store and retrieve your user's "Top 5" music list (e.g., an array of Spotify track IDs or simplified JSON objects containing track name, artist, and ID) in local storage so they persist across browser sessions.	**Recent Searches:** Optionally, store the user's last few music search queries in local storage.

2. Phased Development Plan
3. https://developer.spotify.com/documentation/web-api

Project Idea Proposal: Web Portifolio that contained a application to show your top5 music from spotify

```
https://developer.spotify.com/
// Authorization token that must have been created previously. See :
https://developer.spotify.com/documentation/web-api/concepts/authorization
const token =
'BQAi3i_Z_58cNAE9vwELDpEuOaADcBXSRoZOJKfraLB0BfS2FEjsReJWXkC2Zl0SSgvOnCHna97YtMEdJWLDl
wy2YE7GuPmWKVXj8qAAJH8QplHEevq1YK9QkhGhnQBn2vRnbNjWyCC5p-
4ErB48DZSQmDXR6x3MieR5vCkPD5O_qP0HquJlRyqUAuJF0LcOXsNlRKlT8_K_0KLY-
jMl0bWYNThuSwDHv_oQinVzZkzUZ4gdBK1jZApmj48V1UuJwv4fD5hLgxMmjGlU7tGLZK_x-
LPC2k8sZDujYFEpjDAWnnNBH7uDv57yDhQQUAW_';
async function fetchWebApi(endpoint, method, body) {
  const res = await fetch(`https://api.spotify.com/${endpoint}`, {
    headers: {
      Authorization: `Bearer ${token}`,
    },
    method,
    body:JSON.stringify(body)
  });
  return await res.json();
}

async function getTopTracks(){
  // Endpoint reference : https://developer.spotify.com/documentation/web-api/reference/get-users-
top-artists-and-tracks
  return (await fetchWebApi(
    'v1/me/top/tracks?time_range=long_term&limit=5', 'GET'
  )).items;
}

const topTracks = await getTopTracks();
console.log(
  topTracks?.map(
    ({name, artists}) =>
      `${name} by ${artists.map(artist => artist.name).join(', ')}`
  )
);
```

Phase 1: Project Definition & Planning (Demonstrates Outcomes #1, #2, #4)

Objective: Define your project idea, scope, and initial plan.

- **Brainstorm Project Ideas:**
    - Think about problems you want to solve, interests you have, or previous assignments you could expand upon.
    - Consider what constitutes "medium complexity" – it should go beyond basic static pages but doesn't need to be a full-stack application. Examples: a simple game, a calculator, a task manager, a recipe finder, a small data visualization tool, a simple e-commerce product page with interactive elements.
- **Define Minimum Viable Product (MVP):**

    - What is the absolute core functionality your application *must* have to be considered complete? For the Music Portfolio: Display your predefined Top 5 music, allow searching for new music via Spotify API, add/remove tracks from the Top 5, persist Top 5 in local storage, and play audio previews.
    - List "nice-to-have" features separately for potential future additions (e.g., user authentication for Spotify, more detailed track info, genre filtering).
    - This helps prevent scope creep and ensures you deliver a working product.
- **Set Up Trello Board (or preferred PM tool):**
    - Create lists: "Ideas/Backlog," "To Do," "In Progress," "Done."
    - Break down your MVP into small, actionable tasks (cards). Assign due dates if helpful.
    - This is crucial for demonstrating Outcome #4 (divide and conquer).
- **Initial GitHub Repository Setup:**
    - Create a new public repository.
    - Add a basic README.md file with your project title and a brief description.
    - Commit your initial index.html, style.css, and main.js files (even if empty).
- **Initial Research & Exploration:**
    - Identify a suitable Spotify Web API endpoints (e.g., /search and /tracks/{id}). Read its documentation to understand how to make requests, handle authentication (if needed for advanced features, otherwise public search is fine), and what JSON data it returns.
    - Explore potential solutions or libraries (Outcome #2).

Phase 2: Core Development (HTML, CSS, JavaScript) (Demonstrates Outcomes #4, #6)

Objective: Build the main functionality and structure of your application.

- **HTML Structure:**
    - Create the semantic HTML foundation for your portfolio application (e.g., header, main with sections for "About Me," "Skills," "Projects," and a dedicated "My Top 5 Music" section).
    - Include a search input field and a button for finding music.
    - Create a container div where music cards/list items for your Top 5 and search results will be dynamically inserted.
    - Ensure accessibility best practices are considered.
- **CSS Styling:**
    - Apply styling to make your application visually appealing and user-friendly.
    - **Focus on responsive design:** Use media queries, flexbox, or grid to ensure your application looks good on various screen sizes (mobile, tablet, desktop).
    - Implement the CSS Transforms and Transitions as described in the "Project Idea Proposal" section.
    - Consider a consistent color scheme and typography that fits a personal portfolio.
- **JavaScript Functionality:**
    - Implement the core logic of your application, including Spotify API calls, JSON parsing, and dynamic DOM manipulation.
    - Break down complex JavaScript tasks into smaller functions (Outcome #4).
    - Use modern JavaScript features (e.g., async/await for API calls).
    - Handle user interactions (clicks, input, etc.) using event listeners.
    - Implement local storage logic for saving and retrieving your "Top 5" music list.
    - Implement audio playback for track previews.
- **Version Control (Git):**
    - Commit frequently with meaningful commit messages.
    - Push your changes to GitHub regularly.

Phase 3: Testing & Debugging (Demonstrates Outcomes #5, #6)
Objective: Ensure your application is robust and bug-free.

- **Thorough Testing:**
  - Test all features and user flows (displaying Top 5, searching music, adding/removing tracks, playing previews, local storage persistence).
  - Test on different browsers and devices (if possible).
  - Test edge cases (e.g., no search results, API errors, empty local storage, invalid track IDs).
  - Ask a peer to test your application if feasible (Outcome #3).
- **Identify and Troubleshoot Bugs:**
  - Use browser developer tools (console, debugger, network tab for API calls) to identify issues.
  - Practice systematic debugging techniques (Outcome #5).
  - Document bugs on your Trello board and track their resolution.
- **Refactoring:**
  - Clean up your code: remove redundant code, improve variable names, add comments.
  - Ensure your code is well-organized and readable.

Phase 4: Documentation & Deployment (Demonstrates Outcome #6)
Objective: Prepare your project for final submission and presentation.

- **Finalize GitHub Repository:**
  - Ensure all your code is pushed to GitHub.
  - Update your README.md file:
    - Detailed description of your project (Web Portfolio with Top 5 Music).
    - How to run/use it.
    - Technologies used (HTML, CSS, JavaScript, Spotify Web API).
    - Any known issues or future improvements.
    - Attribution for any external resources used (API, icons, etc.).
- **Deploy Working Application:**
  - Choose a hosting service (e.g., Netlify, Vercel, GitHub Pages) and deploy your application.
  - Verify the link works correctly.
- **Prepare Video Demonstration:**
  - **Part 1: Application Walkthrough:** Clearly demonstrate all features of your working application. Show your Top 5 music, searching for new music, adding/removing tracks, playing previews, and how local storage works for persistence.
  - **Part 2: Reflection on Course Outcomes:** Discuss how you demonstrated Outcomes #1-5 throughout the project. Be specific with examples from your Web Portfolio project.
  - Keep it concise and professional.

Phase 5: Self-Assessment & Reflection (Demonstrates Outcomes #1, #2, #3, #5)
Objective: Reflect on your learning journey and personal development.

- **Review Course Outcomes:** Go back to the list of course outcomes and specifically articulate how your project and your process for building it demonstrate each one.
  - **Outcome #1 (Curiosity & Creativity):** How did you apply these? Did you explore different ways to display your music, or design the portfolio layout creatively?
  - **Outcome #2 (Exploring Environment):** How did you explore the Spotify Web API's documentation? Did you look into different ways to manage audio playback or optimize DOM updates for music cards?

    - **Outcome #3 (Helping & Learning from Others):** Did you consult online resources, tutorials, or forums when implementing API calls or CSS transitions for the music player/cards?
    - **Outcome #4 (Divide and Conquer):** How did you break down the Music Portfolio into smaller tasks (e.g., "Set up Spotify API search," "Render music cards," "Implement 'add to Top 5' button," "Save Top 5 to local storage," "Implement audio playback")?
    - **Outcome #5 (Finding and Troubleshooting Bugs):** Describe a specific bug you encountered (e.g., "Spotify API data wasn't parsing correctly," or "Audio previews weren't playing due to mixed content issues," or "Local storage wasn't persisting the Top 5 list"). Explain the symptoms, the tools you used (e.g., console.log, debugger, network tab), the process you followed to isolate the problem, and the solution. This shows your systematic debugging process.
    - **Outcome #6 (Developing and Debugging HTML, CSS, JS):** Your working Web Portfolio application and its code are the primary evidence here.

3. Connecting to Course Outcomes

Here's how you can specifically demonstrate each outcome:

- **1. Become more efficient at applying your innate curiosity and creativity.**
    - **Demonstration:** Choose a project that genuinely interests you. Show how you explored different approaches or added unique creative touches to your design or functionality. Discuss any "rabbit holes" you went down out of curiosity that ultimately helped your project.
- **2. Become more dexterous at exploring your environment.**
    - **Demonstration:** Detail how you used browser developer tools, researched documentation (MDN, W3Schools, the specific Spotify Web API's docs), or explored new JavaScript APIs/concepts beyond what was explicitly taught in class. Mention any new libraries or frameworks you briefly looked into.
- **3. Become a person who enjoys helping and learning from others.**
    - **Demonstration:** If you collaborated with peers, mention it. Even if not, discuss how you used online communities (Stack Overflow, Discord, etc.) or tutorials to learn and solve problems. You can also mention if you helped anyone else with their projects.
- **4. Use a divide and conquer approach to design solutions for programming problems.**
    - **Demonstration:** Your Trello board (or similar tool) is key here. Show how you broke down your project into smaller, manageable tasks (e.g., "Set up Spotify API fetch for search," "Render music cards," "Implement 'add to Top 5' button," "Save Top 5 to local storage," "Add audio playback"). Explain how tackling one small piece at a time made the larger project less daunting.
- **5. Finding and troubleshooting bugs you and others will have in the code you write.**
    - **Demonstration:** Describe a specific bug you encountered (e.g., "Spotify API data wasn't rendering correctly because of a typo in the JSON path," or "Local storage wasn't persisting because I forgot to JSON.stringify"). Explain the symptoms, the tools you used (e.g., console.log, debugger, network tab), the process you followed to isolate the problem, and the solution. This shows your systematic debugging process.
- **6. Developing and debugging HTML, CSS, and JavaScript programs that use medium complexity web technologies.**
    - **Demonstration:** Your working Web Portfolio application itself is the primary evidence. Ensure it has interactive elements, good responsive design, and clean, functional JavaScript that integrates with an external API and uses local storage. The complexity should be evident in the features you implement