“Web Dojo DPS”

A Major Project Report

Submitted in Partial Fulfillment of Requirements for the Award of Degree of Bachelor of Engineering in Computer Science & Engineering
Submitted to

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL (M.P)

Submitted By
Pranjal Kumar Dwivedi
0132CS211115

Under the guidance of
Dr. Prachi Tiwari

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RADHARAMAN INSTITUTE OF TECHNOLOGY & SCIENCE, BHOPAL (M.P.)

Session: April 2025

RADHARAMAN INSTITUTE OF TECHNOLOGY & SCIENCE BHOPAL (M.P.)

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the work embodies in this dissertation entitled **"Web Dojo DPS"** being submitted by **"Pranjal Kumar Dwivedi" 0132CS211115** for partial fulfillment of the requirement for the award of **"Bachelor of Technology in Computer Science and Engineering"** discipline of "Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)" during the academic year 2024-25 is a record of bonafide piece of work, carried out by him/her under my supervision and guidance in the **Department of Computer Science and Engineering, RADHARAMAN INSTITUTE OF TECHNOLOGY & SCIENCE, Bhopal (M.P.).**

Supervisor	**HOD**	**Director**
Dr. Prachi Tiwari	**Prof. Chetan Agarwal**	**Dr. Ajay K Singh**
CSE Dept.	**CSE Dept.**	**RITS,Bhopal**

RADHARAMAN INSTITUTE OF TECHNOLOGY & SCIENCE BHOPAL (M.P.)

Department of Computer Science & Engineering

APPROVAL CERTIFICATE

This dissertation work entitled **"Web Dojo DPS"** being submitted by **"Pranjal Kumar Dwivedi" 0132CS211115** is approved for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering**.

Internal Examiner **External Examiner**

Date: **Date:**

RADHARAMAN INSTITUTE OF TECHNOLOGY & SCIENCE BHOPAL (M.P.)

Department of Computer Science & Engineering

CANDIDATE DECLARATION

I **"Pranjal Kumar Dwivedi" 0132CS211115** students of **Bachelor of Technology in Computer Science & Engineering, Radharaman Institute Of Technology & Science**, **Bhopal (M.P.),** hereby declare that the work presented in this dissertation "**Web Dojo DPS**" is the outcome of my own work, is bonafide and correct to the best of my knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any patented work and has not been submitted to any other university or anywhere else for the award of any degree or any professional diploma.

Date:

"Pranjal Kumar Dwivedi"
0132CS211115

ACKNOWLEDGEMENT

This project work is the result of guidance and support of various people at RITS without whom all our effort would have been directionless and fruitless. We sincerely thank all of them, for assisting us in completing the dissertation.

We express our ardent and earnest gratitude to our guide, **Dr Prachi Tiwari**, Department of Computer Science & Engineering, RITS Bhopal and **Prof. Chetan Agarwal**, HOD, Department of Computer Science & Engineering, RITS Bhopal for their help and encouragement at all the stages of our Work. Their guidance and motivation helped us to be fruitful in our effort.

We also express my heartfelt and profound gratitude to our Director **Dr. Ajay K Singh** for his valuable suggestion and ample resources at all stages of the research work.

Finally, we would like to say that we are indebted to my parents for everything that they have done for us. All of this would have been impossible without their constant support. And I also thank to God for being kind to me and driving me through this journey.

"Pranjal Kumar Dwivedi"
0132CS211115

CHAPTER 1: INTRODUCTION

1.1 Introduction

WebDojo DPS is an advanced web-based code editor designed to provide an interactive and real-time coding experience for front-end developers. It enables users to write, test, and execute HTML, CSS, and JavaScript code efficiently. The system features a structured user interface with dedicated input areas for each language and an output window that instantly reflects code changes. WebDojo DPS aims to bridge the gap between learning and practical implementation by offering a seamless environment for developers to experiment with front-end technologies.

With increasing demand for interactive coding platforms, WebDojo DPS ensures a user-friendly experience by integrating smooth animations powered by GSAP. The tool caters to both beginners and experienced developers, allowing them to write and test code without external dependencies. Unlike traditional editors, WebDojo DPS focuses on simplicity, responsiveness, and ease of use, making it an ideal choice for learning, teaching, and rapid prototyping of web applications.

1.2 Presently Available System for the Same

1.2.1 Features

- Real-time code execution without external compilation tools.
- Supports HTML, CSS, and JavaScript within a single platform.
- User-friendly interface with an intuitive and structured layout.
- Uses GSAP animations to enhance the user experience.
- Provides instant preview of code changes, eliminating the need for refreshing.
- Allows developers to test small components and snippets before implementing them in full projects.

1.2.2 Limitations

- Limited to front-end technologies, lacking support for backend languages.
- No built-in debugging or error-highlighting tools.
- Requires a large screen for the best user experience; mobile usability is limited.
- Lacks collaboration features for multiple users.

1.3 Problem Statement

Many online code editors either lack a well-structured user interface or do not provide real-time rendering of front-end code. Developers often struggle with inefficient workflows where they need to switch between different applications to test their code. Additionally, traditional editors can be complex for beginners, making it difficult to grasp front-end development concepts quickly.

WebDojo DPS aims to solve these challenges by offering a web-based platform that provides an interactive, smooth, and efficient coding experience. It focuses on real-time execution, ease of use, and visual representation, making front-end development more accessible and productive.

1.4 Proposed Solution

The proposed system provides an interactive code editor where users can write, edit, and execute HTML, CSS, and JavaScript code in real-time. By integrating GSAP animations and a structured layout, WebDojo DPS enhances the user experience, making front-end development more engaging and efficient. The tool ensures minimal distractions and a clean workspace for developers to focus on writing and testing their code effortlessly.

1.5 Aim of the Project

To develop an intuitive and efficient web-based code editor that allows users to write and test HTML, CSS, and JavaScript code with real-time execution, seamless visualization, and an enhanced user experience through smooth animations and a structured layout.

1.6 Objectives of the Project

- Provide a responsive and interactive development environment for front-end coding.
- Implement real-time execution of code within an embedded output window.
- Ensure a user-friendly interface with a structured and intuitive design.
- Integrate GSAP animations for a seamless and visually appealing experience.
- Allow users to write and test front-end code without needing an external server or additional setup.
- Minimize latency in code execution for an uninterrupted workflow.
- Optimize the editor for both beginner-level learning and professional prototyping.

1.7 Applications

- **Learning and Practicing Front-End Technologies**: WebDojo DPS serves as an excellent tool for beginners and students to learn HTML, CSS, and JavaScript interactively.
- **Instant Preview of Code Changes**: Developers can visualize their code execution in real-time without needing to reload the page.
- **Teaching and Demonstrations for Web Development Courses**: Instructors can use WebDojo DPS to demonstrate front-end development concepts during live sessions.
- **Rapid Prototyping**: Developers can quickly test new UI components before integrating them into full-scale projects.
- **Freelance and Hobby Projects**: Useful for individual developers looking to experiment with front-end designs without setting up a complex development environment.

CHAPTER 2: METHODOLOGY

2.1 Introduction

This chapter provides an in-depth discussion of the approach used to develop WebDojo DPS, including the methodology, process model, and project planning strategies. The methodology focuses on an iterative and structured approach to ensure efficient implementation, flexibility, and user-centric development. The model chosen for development allows for regular testing, enhancements, and improvements based on feedback, ensuring a robust and refined system.

2.2 Proposed Methodology

The development process follows an iterative approach, enabling continuous testing, evaluation, and improvement. The methodology ensures that the core functionalities are developed first, followed by additional enhancements in later phases. Each stage undergoes rigorous testing to validate its efficiency, usability, and responsiveness. The approach also involves gathering feedback from users and stakeholders, allowing for modifications and refinements throughout the development cycle.

2.3 Process Model Adopted

2.3.1 Name of Adopted Model

The **Incremental Model** was selected for the development of WebDojo DPS.

2.3.2 Reason for Selecting Model

- Allows for continuous development and refinement in manageable increments.
- Enables testing and debugging of individual components before full integration.
- Provides flexibility to incorporate additional features and improvements in later iterations.
- Ensures early detection of issues, allowing for quick resolution.
- Facilitates user feedback at various stages, improving the final product's usability and functionality.

2.4 Time Plan and Team Structure

2.4.1 Team Structure and Assigned Task

The project was developed through collaboration among various team members, each responsible for specific tasks:

- **Front-end Development**: UI design, implementation, and optimization.
- **Logic Implementation**: JavaScript functionality for real-time execution and smooth integration with the output window.
- **Styling and Animations**: CSS and GSAP integration for an enhanced user experience.
- **Testing and Debugging**: Ensuring smooth operation, identifying bugs, and improving performance.

2.4.2 Time Plan

The development of WebDojo DPS followed a structured timeline, dividing tasks into multiple phases:

- **Week 1-2**: UI Design and initial prototyping, including layout structuring.
- **Week 3-4**: Implementation of JavaScript functionality for real-time execution and integration of GSAP animations.
- **Week 5**: Testing and debugging, performance optimization, and user feedback evaluation.
- **Week 6**: Final review, deployment, and documentation.

By following this methodology, WebDojo DPS ensures a structured development process, enabling efficient implementation, scalability, and user satisfaction. The iterative model ensures that the system remains adaptable to new requirements while maintaining its core functionalities.

CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

3.1 Introduction

This chapter provides a comprehensive analysis of the system requirements, design, and specifications of WebDojo DPS. It covers the system's functional and non-functional requirements, module descriptions, architectural components, and design diagrams to illustrate how the system operates. WebDojo DPS is developed with a structured approach to ensure seamless execution, user-friendly interaction, and optimal performance for front-end developers.

3.2 Requirement Analysis

Requirement analysis is crucial in understanding the needs and expectations of the system. The requirements are categorized into functional and non-functional requirements.

3.2.1 Requirement Gathering

To build an effective web-based code editor, the following requirements were gathered:

- Handling user input for HTML, CSS, and JavaScript code.
- Enabling real-time execution of code using an iframe.
- Providing smooth animations to enhance the user experience.
- Ensuring a structured and responsive UI for different screen sizes.

3.2.2 Feasibility Study

The feasibility study evaluates the practicality and implementation of WebDojo DPS based on three aspects:

3.2.2.1 Economical Feasibility

- The system is cost-effective, requiring only a web browser to function.
- No need for additional software or hardware resources, reducing expenses.
- Open-source technologies ensure low development and maintenance costs.

3.2.2.2 Technical Feasibility

- Developed using widely-used technologies like HTML, CSS, JavaScript, and GSAP.
- Utilizes an iframe for real-time code execution, ensuring efficiency.
- Compatible with modern web browsers without requiring additional plugins.

3.2.2.3 Operational Feasibility

- Users can write, edit, and test their code instantly without prior setup.
- Simple and intuitive interface, reducing the learning curve for beginners.
- Provides real-time preview functionality, improving the development workflow.

3.3 System Specification

The system consists of several functional components that ensure smooth operation and usability.

3.3.1 Functional Description

- **Three dedicated text areas**: Separate sections for writing HTML, CSS, and JavaScript code.
- **Real-time execution**: Immediate visualization of changes using an iframe.
- **Output window**: Displays the rendered result of the written code.
- **GSAP animations**: Enhances UI interactions and transitions.
- **Auto-update mechanism**: Executes code changes without requiring manual refresh.

3.3.2 Subsystem Description (Project Modules)

WebDojo DPS is divided into multiple modules, each handling a specific functionality:

3.3.2.1 Name of Modules

- **Input Handling Module**: Accepts user input for HTML, CSS, and JavaScript.
- **Output Rendering Module**: Displays the live preview of the written code.
- **Animation Module**: Implements GSAP animations for a smoother experience.
- **User Interface Module**: Manages the layout, structure, and responsiveness.
- **Execution Module**: Ensures proper execution of the code in real-time.

3.4 System Requirements

System requirements define the essential functionalities and constraints for WebDojo DPS.

3.4.1 Functional Requirements

- Ability to execute HTML, CSS, and JavaScript code in real-time.
- Dynamic rendering using an iframe for immediate feedback.
- Dedicated input areas for each coding language.
- Smooth animations for UI interactions.
- Error handling to prevent execution failures.

3.4.2 Non-Functional Requirements

- **Performance**: Minimal delay in real-time execution.
- **Responsiveness**: Adaptive design for different screen sizes.
- **Scalability**: Future support for additional features like code collaboration.
- **Security**: Ensuring safe execution of JavaScript without security vulnerabilities.

3.5 UML Diagrams

UML diagrams are used to visually represent the system's architecture, interactions, and workflows.

3.5.1 Use Case Diagram: Brief Description

- Illustrates the interaction between the user and WebDojo DPS.
- Shows the actions a user can perform, such as writing, executing, and previewing code.

3.5.2 Activity Diagram: Brief Description

- Displays the flow of actions within the system, from code input to real-time rendering.
- Represents how data flows between modules during execution.

3.5.3 Sequence Diagram: Brief Description

- Demonstrates the sequence of interactions between the user, input areas, execution module, and output window.
- Shows how code is processed and displayed in real-time.

3.5.4 Class Diagram: Brief Description

- Defines the structure of the system in terms of objects, attributes, and functions.
- Represents different classes responsible for handling input, execution, and output.

3.5.5 Collaboration Diagram: Brief Description

- Describes the communication between various modules.
- Shows how different components of WebDojo DPS work together to achieve real-time execution.

3.6 Data Flow Diagram (Level 0, 1, 2)

The Data Flow Diagram (DFD) illustrates how data moves within the system:

- **Level 0**: High-level overview showing user interaction and code execution flow.
- **Level 1**: Detailed breakdown of data movement between the input, execution, and output modules.
- **Level 2**: In-depth representation of internal data processing and rendering.

3.7 ER Diagram

- Since WebDojo DPS is a front-end tool, an ER diagram is not required.
- If future versions incorporate backend features, an ER diagram may be added.

3.8 Database Design

- Currently, WebDojo DPS does not involve database storage.
- In future iterations, database integration may be considered for saving user sessions or code snippets.

CHAPTER 4: IMPLEMENTATION

4.1 Technology Used

The development of WebDojo DPS involves multiple technologies to ensure a seamless user experience. The chosen technologies provide a robust foundation for front-end development, real-time execution, and enhanced interactivity.

- **Frontend Technologies: WebDojo DPS is built using standard web technologies:**
  - **HTML:** Structures the user interface and defines elements such as text areas and the output window.
  - **CSS:** Styles the editor, ensures responsiveness, and integrates animations for an enhanced experience.
  - **JavaScript:** Implements logic for real-time code execution, syntax handling, and dynamic updates.
- **Animation Library:**
  - **GSAP (GreenSock Animation Platform):** Used for smooth transitions, UI interactions, and animations that enhance user experience.
- **Execution Engine:**
  - **Iframe-based execution:** Renders the user's code in real-time without requiring server-side processing.
- **Additional Libraries:**
  - **CodeMirror or Ace Editor (optional):** Can be used for syntax highlighting and an improved coding experience.
  - **LocalStorage API:** Stores and retrieves user code for persistence.

4.2 Coding Implementation

The implementation of WebDojo DPS follows a modular approach, ensuring clear separation of concerns. The key aspects of coding include:

- **Code Execution Module:**
  - Listens for user input changes in HTML, CSS, and JavaScript text areas.
  - Updates the iframe output dynamically to reflect code changes in real time.
  - Uses JavaScript event listeners to optimize execution flow and prevent unnecessary re-renders.
- **UI and Styling Module:**
  - Implements a modern and intuitive layout using CSS Flexbox and Grid.
  - Ensures responsiveness across different screen sizes, from desktops to mobile devices.
  - Uses GSAP animations for UI elements like buttons, input fields, and transitions between themes.
- **Code Persistence Module:**
  - Saves the user's written code in the browser's local storage.
  - Allows users to reload and access previously written code without data loss.
- **Theme and Customization Module:**
  - Provides light and dark mode options for user preferences.
  - Implements font size adjustments and customizable layouts for better usability.

4.3 Code Structure Overview

The WebDojo DPS project follows a structured codebase to maintain readability and scalability:

- **index.html:** Contains the basic layout, including input areas and output frame.
- **styles.css:** Defines styles, animations, and layout responsiveness.
- **script.js:** Handles code execution, user interactions, and real-time updates.
- **animations.js:** Manages GSAP animations and smooth transitions.

HTML CODE -

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>WebDojo DPS</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <div class="editor-container">

        <textarea id="html-code" placeholder="Write HTML here..."></textarea>

        <textarea id="css-code" placeholder="Write CSS here..."></textarea>

        <textarea id="js-code" placeholder="Write JavaScript here..."></textarea>

    </div>

    <button id="run-code">Run Code</button>

    <iframe id="output"></iframe>

    <script src="script.js"></script>

</body></html>
```

CSS Code -

```
/* General Styles */
```

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
    color: #333;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
}

/* Container */
.editor-container {
    display: flex;
    flex-direction: column;
    width: 80%;
    max-width: 1200px;
    background: #fff;
    box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.2);
    border-radius: 8px;
    overflow: hidden;
}

/* Editor Sections */
```

```
.editor {
    display: flex;
    flex-direction: column;
    padding: 10px;
    border-bottom: 1px solid #ddd;
}

.editor label {
    font-weight: bold;
    margin-bottom: 5px;
}

.editor textarea {
    width: 100%;
    height: 120px;
    border: 1px solid #ccc;
    border-radius: 4px;
    padding: 8px;
    font-size: 14px;
    resize: none;
}

/* Output Section */
.output-container {
    padding: 10px;
    background: #eee;
```

```
}

iframe {
    width: 100%;
    height: 300px;
    border: none;
    background: #fff;
}

/* Button Styling */
.run-button {
    display: inline-block;
    padding: 10px 20px;
    background: #007bff;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
    margin-top: 10px;
}

.run-button:hover {
    background: #0056b3;
}
```

```
/* Responsive Design */
@media (max-width: 768px) {
    .editor-container {
        width: 95%;
    }

    .editor textarea {
        height: 100px;
    }
}
```

Javascript Code -

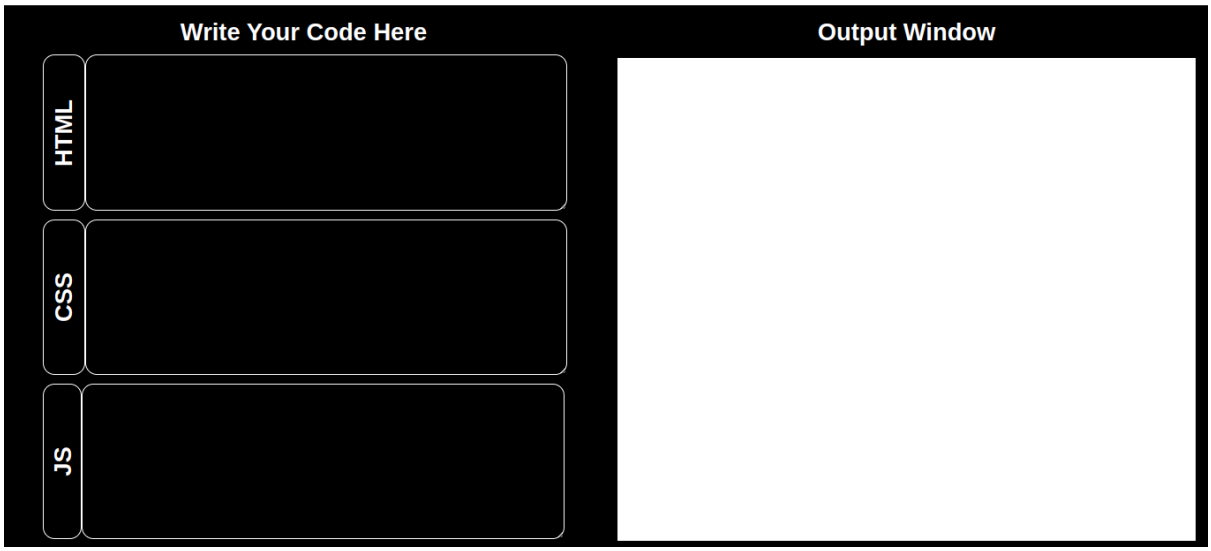
```
document.addEventListener("DOMContentLoaded", () => {
    const htmlInput = document.getElementById("htmlInput");
    const cssInput = document.getElementById("cssInput");
    const jsInput = document.getElementById("jsInput");
    const outputFrame = document.getElementById("outputFrame").contentDocument;
    const themeToggle = document.getElementById("themeToggle");

    function updateOutput() {
        outputFrame.open();
        outputFrame.write(`
            <html>
            <head>
              <style>${cssInput.value}</style>
            </head>
```

```
            <body>
                ${htmlInput.value}
                <script>${jsInput.value}<\/script>
            </body>
            </html>
        `);
        outputFrame.close();
        localStorage.setItem("html", htmlInput.value);
        localStorage.setItem("css", cssInput.value);
        localStorage.setItem("js", jsInput.value);
    }

    [htmlInput, cssInput, jsInput].forEach(input => {
        input.addEventListener("input", updateOutput);
    });

    function loadSavedCode() {
        htmlInput.value = localStorage.getItem("html") || "";
        cssInput.value = localStorage.getItem("css") || "";
        jsInput.value = localStorage.getItem("js") || "";
        updateOutput();
    }
    themeToggle.addEventListener("click", () => {
        document.body.classList.toggle("dark-mode");    });
    loadSavedCode();
});
```

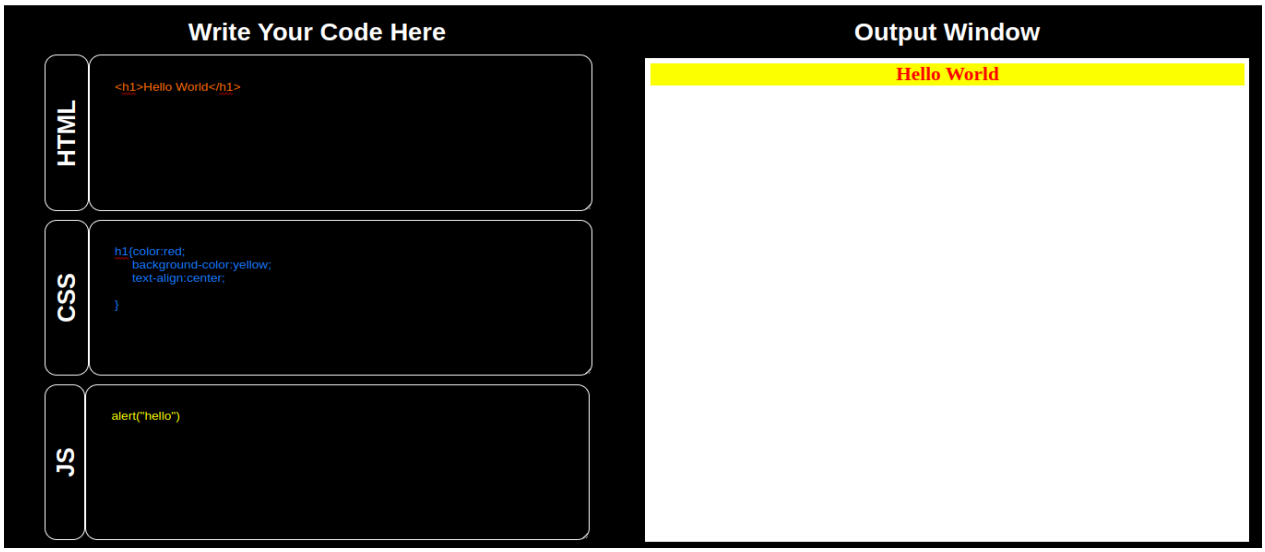
4.4 Snapshots

To provide a visual understanding of WebDojo DPS, the following snapshots are included:

- **Home Screen: Displays the main interface, including code input areas and the output preview.**



Code Execution in Action: Showcases real-time rendering when the user types HTML, CSS, or JavaScript



CHAPTER 5: TESTING

5.1 Testing Objectives

The primary objective of testing WebDojo DPS is to ensure smooth and error-free execution, responsiveness, and a user-friendly interface. The main goals include:

- Verifying the real-time execution of HTML, CSS, and JavaScript code.
- Ensuring animations and UI elements function as expected.
- Testing responsiveness across various screen sizes.
- Checking code persistence using LocalStorage.
- Identifying and resolving potential bugs.

5.2 Testing Methods and Strategies

To validate WebDojo DPS, multiple testing methods were employed:

- **Unit Testing:** Focused on individual components, such as the code execution module and theme toggle.
- **Integration Testing:** Verified interactions between different modules (e.g., text input updating iframe output).
- **Functional Testing:** Ensured that key functionalities like code execution, UI animations, and LocalStorage persistence work correctly.
- **Compatibility Testing:** Checked performance across different browsers and devices.
- **User Testing:** Collected feedback from users to identify potential UI/UX improvements.

5.3 Test Cases

Below are some key test cases used to validate WebDojo DPS:

Test Case ID	Test Scenario	Expected Result	Status
TC-01	Enter HTML code in editor	Code renders in output window	Pass
TC-02	Enter CSS code in editor	Styles apply correctly in output	Pass
TC-03	Enter JavaScript code in editor	JavaScript executes properly	Pass

TC-04	Modify code dynamically	Output updates in real-time	Pass
TC-05	Switch between dark and light mode	UI theme changes instantly	Pass
TC-06	Reload page	Code persists using LocalStorage	Pass
TC-07	Resize browser window	Layout remains responsive	Pass
TC-08	Check GSAP animations	Transitions and effects work smoothly	Pass
TC-09	Test on different browsers	Works consistently on Chrome, Firefox, and Edge	Pass
TC-10	Mobile Testing	UI remains accessible and usable on mobile devices	Pass

5.4 Bug Fixes and Improvements

During the testing phase, several bugs and UI inconsistencies were identified and resolved:

- Fixed a bug where JavaScript code execution would sometimes not update properly.
- Improved iframe security settings to prevent external code execution risks.
- Enhanced UI responsiveness for better performance on mobile screens.
- Optimized GSAP animations to prevent lagging on lower-end devices.

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

WebDojo DPS successfully provides an efficient and interactive platform for writing, testing, and executing front-end code in real time. The project offers an intuitive interface, seamless integration of HTML, CSS, and JavaScript execution, and smooth animations that enhance the user experience. The implementation of LocalStorage also ensures that users can retain their work without worrying about data loss. Through rigorous testing, WebDojo DPS has been validated for functionality, responsiveness, and performance across different devices and browsers.

6.2 Limitations of the Project

Despite its strengths, WebDojo DPS has some limitations:

- It is limited to front-end technologies and does not support backend processing.
- There are no built-in debugging tools like error highlighting or code suggestions.
- Performance may vary on lower-end devices due to animation processing.
- Requires a large screen for optimal experience, as the UI layout may feel constrained on smaller devices.

6.3 Difficulties Encountered

During development, several challenges were faced and addressed:

- **Real-time synchronization:** Ensuring that the output updates instantly without delays.
- **Optimizing animations:** Avoiding performance issues caused by GSAP animations on low-end devices.
- **Cross-browser compatibility:** Ensuring that the application performs consistently across different web browsers.
- **Storage persistence:** Implementing a reliable way to save and retrieve user code without affecting performance.

6.4 Future Enhancement Suggestions

To further improve WebDojo DPS, the following enhancements are proposed:

- **Backend support:** Allow users to save projects to a database and retrieve them later.
- **Integrated debugging tools:** Provide error detection, code hints, and syntax suggestions.
- **Code collaboration:** Enable multiple users to edit and view code simultaneously.
- **Mobile optimization:** Enhance UI adaptability for a better mobile experience.
- **Theme customization:** Allow users to create and apply custom themes.

These improvements will make WebDojo DPS a more powerful and versatile tool for developers and learners alike.

REFERENCES

- JavaScript Documentation
- GSAP Animation Library