A REPORT
ON

WEB DEVELOPMENT

Submitted by,

Mr. Rishi Anand - 20211CDV0032

Under the guidance of,

Ms. Meena Kumari K S

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY(DEVOPS)

At

PRESIDENCY UNIVERSITY

BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship report **"WEB DEVELOPMENT"** being submitted by RISHI ANAND bearing roll number 20211CDV0032 in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Technology(DevOps) is a bonafide work carried out under my supervision.

Ms. MEENA KUMARI K S
Assistant Professor
PSCS
Presidency University

Dr. S PRAVINTH RAJA
Professor & HoD
PSCS
Presidency University

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vice Chancellor - Engineering
Dean –PSCS / PSIS
Presidency University

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the work, which is being presented in the report entitled **"WEB DEVELOPMENT"** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Technology(DevOps)**, is a record of my own investigations carried under the guidance of **Ms. Meena Kumari K S, Assistant Professor, Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

Rishi Anand

20211CDV0032

INTERNSHIP COMPLETION CERTIFICATE

ABSTRACT

Web development is a multifaceted and continually evolving discipline that involves the creation, deployment, and maintenance of websites and web-based applications that power the modern digital experience. It is broadly divided into front-end and back-end development. Front-end development focuses on the client side, dealing with the structure, design, and interactivity of web interfaces using HTML, CSS, JavaScript, and modern frameworks such as React, Angular, and Vue.js. Back-end development, in contrast, is concerned with server-side logic, databases, and application architecture, employing technologies like Node.js, PHP, Python, Ruby on Rails, and Java. The synergy between front-end and back-end components results in full-stack development, enabling developers to build end-to-end solutions. With the proliferation of mobile devices and the growing demand for seamless user experiences, responsive and mobile-first design principles have become industry standards. Moreover, the adoption of Single Page Applications (SPAs), Progressive Web Apps (PWAs), and headless CMS architectures has significantly enhanced the functionality and performance of modern websites. Web development also involves integrating third-party services through APIs, utilizing cloud platforms for scalable hosting, and applying DevOps practices for continuous integration and deployment. Security is another critical concern, necessitating measures like HTTPS, secure authentication, and protection against common vulnerabilities such as cross-site scripting and SQL injection. Accessibility and performance optimization further underscore the importance of inclusive and efficient design practices. As artificial intelligence, voice search, and real-time technologies like WebSockets gain traction, web development is set to become even more interactive and user-centric. Ultimately, web development is not just about coding but about creating powerful digital ecosystems that connect users, deliver content, and enable innovation across virtually every industry in the global economy.

ACKNOWLEDGEMENT

First of all, I am indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

I express my sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presiency School of Computer Science and Engineering & Presiency School of Information Science, Presidency University for getting us permission to undergo the project.

I express my heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair,** Presidency School of Computer Science and Engineering, Presidency University, and **Dr. S Pravinth Raja**, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

I am greatly indebted to our guide **Ms. Meena Kumari K S, Assistant Professor** and Reviewer **Mr. Shankar J, Assistant Professor**, Presidency School of Computer Science and Engineering, Presidency University for her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

I would like to convey our gratitude and heartfelt thanks to the CSE7301 Internship Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K,** department Project Coordinators **Ms. Suma N G** and Git hub coordinator **Mr. Muthuraj.**

I thank my family and friends for the strong support and inspiration they have provided us in bringing out this project.

Rishi Anand

LIST OF FIGURES

Sl. No.	Figure Name	Caption	Page No.
1	Figure 6.1	Entity-Relationship diagram for a blog website	39
2	Figure 7.1	Timeline of the Internship	41

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	**v**
	ACKNOWLEDGMENT	**vi**
	LIST OF FIGURES	**vii**
1.	**INTRODUCTION**	1
	1.1 Website Replication Task	1
	1.2 Back-End Form Handling and Email Functionality	2
	1.3 Blog Website Project (EJS + Express.js)	3
2.	**LITERATURE SURVEY**	**4**
	2.1 Web Development: A Practical Introduction	4
	2.2 A Comparative Study of Web Development Frameworks	5
	2.3 Enhancing Web Application Security through Secure Coding Practices	7
	2.4 Responsive Web Design: Techniques and Challenges	9
	2.5 Performance Optimization in Modern Web Applications	12
	2.6 The Role of Progressive Web Apps in Modern Web Development	13
	2.7 Accessibility in Web Development: Standards and Implementation	15
	2.8 Integrating Artificial Intelligence into Web Development	17
	2.9 Microservices Architecture in Web Development	19
	2.10 Web Development Trends: A Survey of Emerging Technologies	21
3.	**RESEARCH GAPS IN EXISTING METHODS**	24
	3.1 User-Centered Design and Accessibility Limitations	24
	3.2 Integration Challenges Between Front-End and Back-End	25
	3.3 Performance Optimization Limitations	25
	3.4 Web Security and Privacy Issues	26
	3.5 Scalability and Maintenance Challenges	27

	3.6. Developer Experience and Tooling Gaps	27
4.	**PROPOSED METHODOLOGY**	**29**
	4.1 Website Replication Task	29
	4.2. Back-End Form Development with Email Capability	30
	4.3. Blog Website Using EJS and Express.js	31
	4.4 Development Workflow	32
5.	**OBJECTIVES**	**33**
6.	**SYSTEM DESIGN AND IMPLEMENTATION**	**37**
	6.1. Website Replication – Front-End System	37
	6.2. Form with Email Functionality – Backend Integration	38
	6.3. Blog Website – Full Stack with Express.js and EJS	39
7	**TIMELINE OF THE INTERNSHIP (GANTT CHART)**	**41**
8.	**OUTCOMES**	**42**
9.	**RESULTS AND DISCUSSIONS**	**45**
	9.1. Website Replication	45
	9.2. Form Submission with Email Integration	46
	9.3. Blog Website Project (EJS + Express.js)	47
	9.4. Cross-Project Observations	47
10.	**CONCLUSION**	**48**
	REFERENCES	**50**
	APPENDIX-A: PSEUDOCODE	**52**
	APPENDIX-B: SCREENSHOTS	**68**
	APPENDIX-C: ENCLOSURES	**71**
	SUSTAINABLE DEVELOPMENT GOALS	**74**

CHAPTER 1

INTRODUCTION

Internships play a crucial role in filling the gap between theoretical knowledge and practical experience. They provide a working environment where aspiring developers can implement theoretical knowledge, hone technical skills, and get exposed to industry standards. This report summarizes the experiences, learnings, and contributions during my Web Development internship at UptoSkills that lasted 3 months.

The internship was mainly centered on building essential front-end and back-end web development skills. I was assigned a series of tasks that challenged both my knowledge of fundamental web technologies as well as my capacity to apply dynamic functionality using server-side scripting. The internship consisted of three main elements:

- Website replication: Familiarity with design concepts and improvement in HTML, CSS, and JavaScript skills through the exercise of replicating an existing website structure and functionality.

- Back-end development for a contact form: Building a server-side application to send automatic emails based on the form data provided.

- Blog website project using EJS and Express.js: An extensive exercise that entails templating, routing, data handling, and displaying dynamic content.

These assignments were designed to solidify the concepts of full-stack web development as well as introduce experience with real-world workflow and development environments.

1.1 Website Replication Task

Replication of an existing website was one of the earliest assignments given during the internship. This assignment was pivotal in comprehending structuring of layouts, responsiveness, and consistency of design.

The intention was not only to replicate the visual features of the website but also to learn the reasoning behind the design components—spacing, grid systems, typography, and interactive behavior. With HTML as structure and CSS (with frameworks like Bootstrap if needed) for styling, I replicated the visual interface and navigation pattern of the example site.

Skills and Tools Utilized:

- HTML5 and CSS3 for structure and styling.
- JavaScript for interactivity.
- Tools used by the developer for inspecting and debugging components.
- Responsive design methods employing Flexbox, Grid, and media queries.

This task presented a great introduction to UI/UX design principles and offered me a firsthand experience of how front-end development is a mix of creativity and logic.

1.2 Back-End Form Handling and Email Functionality

The second big assignment was adding back-end capability to a contact form. The assignment involved creating a server with Express.js that would accept information from a form and process it to send an email to the email address provided in the form.

The process was as follows:

- A user completes a form with a name, message, and email address.
- When the form is submitted, the information is sent as a POST request to the server.
- The server then processes the information and responds with an email confirmation through a mailer service such as Nodemailer.

Skills and Tools Used:

- Node.js and Express.js to create the server and handle HTTP requests.
- Nodemailer to send emails from the server.
- Body-parser middleware to process form data.
- Basic validation and error checking.

This assignment was critical in familiarizing me with server-client interaction and

asynchronous operations. It showcased the practical application of back-end development, particularly in contact systems, feedback forms, and notifications.

1.3 Blog Website Project (EJS + Express.js)

The capstone project of the internship was the development of a blog website using Express.js and Embedded JavaScript (EJS) templating. This project encapsulated all the core learnings of the internship, allowing for both front-end rendering and back-end logic to work together harmoniously.

The blog website featured the following:

- A home page listing all blog posts.
- A dynamic post page generated using EJS, based on unique post IDs.
- A compose page for users to submit new blog entries.
- Routing handled through Express.js.
- Data temporarily stored in memory (with potential future scope for database integration).

Features and Functionalities:

- EJS templating to dynamically insert blog content into HTML pages.
- RESTful routing (GET and POST requests) using Express.js.
- Use of Bootstrap for quick UI development.
- Basic form validation and redirection.
- Modular code structure for scalability.

Skills and Tools Used:

- Express.js for routing and middleware integration.
- EJS as a templating engine to create reusable views.
- JavaScript and Node.js for logic and server-side scripting.
- Version control with Git for code management.

The upcoming sections of this report delve deeper into each of the tasks and projects, outlining the approaches taken, challenges faced, and solutions implemented.

CHAPTER 2

LITERATURE SURVEY

2.1 Web Development: A Practical Introduction

Abstract

The paper provides an extensive overview of contemporary web development methodologies, including front-end and back-end technologies. The paper intends to provide readers with the necessary knowledge to develop responsive and interactive web applications. Major topics covered are HTML, CSS, JavaScript, server-side scripting, database integration, and deployment mechanisms.

Methodology and Techniques

- Front-End Development: Application of HTML5 for content structure, CSS3 for styling, and JavaScript (with frameworks such as React or Angular) for interactivity.
- Back-End Development: Coding with server-side languages like Node.js, Python (Django), or PHP, along with databases like MySQL or MongoDB.
- Development Workflow: Use of version control systems (such as Git), package managers (such as npm), and build tools (such as Webpack) for efficient development.
- Testing and Deployment: Integration of testing frameworks (e.g., Jest, Mocha) and deployment platforms (e.g., Heroku, Netlify) to ensure application reliability and accessibility.

Merits and Advantages

- Comprehensive Skill Set: Offers a whole picture understanding of both client-side and server-side development.
- Practical Application: Focuses on hands-on experience through project-based learning, improving real-world applicability.
- Scalability: Delivers scalable architectures and best practices, equipping developers for big projects.

Limitations and Challenges

- Rapid Technological Changes: The field of web development changes rapidly, possibly making certain practices obsolete.
- Learning Curve: The scope of subjects taught might be daunting to new learners, requiring extra learning materials for proficiency.
- Tooling Complexity: Juggling multiple tools and frameworks brings in complexity and the need for ongoing learning.

Implications and Applications

- Educational Use: As a core book for web development courses and bootcamps.
- Professional Development: Helps prospective developers develop a strong portfolio, making them more employable.
- Entrepreneurial Ventures: Enables users to create and launch their own web applications or startups.

Conclusion

"Web Development: A Practical Introduction" tries to de-mystify the web development process by offering an organized and pragmatic direction. By discussing necessary technologies and workflows, it equips readers to take the leap as capable web developers with the ability to fit into the ever-changing digital scene.

2.2 A Comparative Study of Web Development Frameworks

Abstract

The article concerns the crucial decision-making cycle of choosing suitable web development frameworks. It compares six commonly utilized frameworks—Django, Ruby on Rails, Laravel, Spring Boot, Angular, and React—on important parameters of performance, scalability, usability, adaptability, community support, security, and overall developer experience. The research takes a qualitative as well as quantitative approach through performance benchmarking tests and surveys of developers for a comprehensive analysis. The objective is to help developers and organizations make well-informed, requirement-based decisions when selecting the most appropriate framework for their particular web development initiatives.

Methodology and Techniques

The research utilizes a mixed-method approach:

- Framework Selection: Six widely used frameworks were selected based on their common use and capability to meet varied development requirements in various programming languages.
- Quantitative Analysis: Performance benchmarking tests were run to compare the speed and efficiency of each framework under varying conditions.
- Qualitative Analysis: Surveys from developers and interviews with experts gave insights about the ease of use, learning curve, and practical difficulties faced during the development.

This multi-method approach ensures that the results are based on both empirical evidence and actual developer experiences.

Merits and Advantages

The comparative analysis identifies the following strengths:

- Django: It provides fast development and a neat, pragmatic layout with a myriad of built-in features.
- Ruby on Rails: A convention-over-configuration method for quick development cycles.
- Laravel: Famed for having an elegant syntax and strong features, PHP is more fun to develop.
- Spring Boot: It is strong in developing standalone, production-grade applications with few configurations.
- Angular: Rich framework for crafting dynamic single-page applications.
- React: Flexible and efficient way to create interactive user interfaces.
- These frameworks are backed by robust communities, large documentation, and active forums, which make it easier to solve problems and share knowledge.

Limitations and Challenges

The research finds a number of challenges that are related to these frameworks:

- Learning Curve: Frameworks such as Spring Boot possess a higher learning curve, particularly for developers who are not Java experts.
- Configuration Complexity: Laravel is more manually configured compared to frameworks such as Django.
- Performance Variability: Performance may differ drastically based on the particular use case and the degree to which the framework is optimized.
- Such challenges require serious consideration of the project requirements and the expertise of the development team.

Implications and Applications

The results of this research have real-world implications for different stakeholders:

- Developers: Can make educated choices when choosing a framework that suits their project requirements and individual expertise.
- Project Managers: Get an insight into the trade-offs among various frameworks, supporting resource allocation and project planning.
- Organizations: Are able to match framework choice with long-term objectives, ensuring scalability, maintainability, and performance of their web applications.
- Knowing the strengths and weaknesses of each framework, stakeholders can streamline the development process and improve the quality of the end product.

Conclusion

The research concludes that no framework fits all; the best option will depend on certain project requirements, team skills, and long-term goals. Frameworks such as Django and Spring Boot are appropriate for projects demanding high scalability and strong security, whereas Laravel and React are most suited for rapid prototyping and adaptability. The detailed analysis presented in this research will be a good resource for informed decision-making in the fast-moving web development area.

2.3 Enhancing Web Application Security through Secure Coding Practices

Abstract

The paper highlights the critical role of secure coding in countering vulnerabilities in web

applications. It stresses proactive approaches, such as threat modeling and following secure coding guidelines, to anticipate likely exploits. Through incorporating these methods, developers can strengthen applications against advanced cyber threats.

Methodology and Techniques

The research takes a multi-pronged approach towards secure coding:

- Threat Modeling: Determining potential vulnerabilities at the design stage to foresee and avoid risks.
- Secure Coding Guidelines: Conforming to best practices, such as input validation, adequate authentication measures, and secure session management.
- Code Reviews and Static Analysis: Code regularly reviewed and static analysis tools used to identify and fix security vulnerabilities early during development.
- Penetration Testing: Simulated attacks to test the application for vulnerabilities against possible attacks.

Merits and Advantages

Adopting secure coding practices has a number of advantages:

- Improved Security Posture: Decreases the potential for common vulnerabilities like SQL injection and cross-site scripting.
- Cost Efficiency: Fixing security problems early in development costs less than patching after deployment.
- Regulatory Compliance: Compliant with industry standards and regulation, keeping it legal and ethical.
- Enhanced Trust: Users will be more willing to trust secure applications, building better reputation and user loyalty.

Limitations and Challenges

Even though it has its benefits, secure coding practice implementation has limitations:

- Developer Awareness: Not all developers receive adequate training on security best practices.

- Resource Constraints: Allocating time and resources for proper security steps proves to be difficult in high-speed development environments.
- Evolving Threat Landscape: Staying current with emerging threats involves ongoing learning and adjustment.

Implications and Applications

The implementation of secure coding practices has far-reaching implications:

- Software Development Lifecycle (SDLC): Invigorating security at the beginning ensures an integrated approach towards application development.
- Organizational Policy: Implementing security protocols and training programs creates a culture of security awareness.
- Industry Standards: Helps in developing standardized security patterns throughout the tech industry.

Conclusion

The article concludes that incorporating secure coding habits into the development process is key to creating resilient web applications. By actively counteracting potential weaknesses, organizations are able to secure themselves against cyber attacks, keeping their digital offerings intact and trustworthy.

2.4 Responsive Web Design: Techniques and Challenges

Abstract

Responsive Web Design (RWD) is a methodology that allows websites to respond and adapt to different devices and screen sizes and still present the best user experience on desktops, tablets, and smartphones. This article discusses the fundamental techniques of RWD, including flexible grid systems, fluid images, and media queries, and how it addresses challenges during execution, including performance and design consistency. The research seeks to offer insights into the successful implementation of RWD principles to maximize accessibility and user interaction in the changing digital world.

Methodology and Techniques

The article presents some of the most important techniques involved in adopting Responsive

Web Design:

- Flexible Grid Layouts: Using relative units such as percentages rather than fixed units to enable layouts to change dynamically according to varying screen sizes.
- Fluid Images: Scaling images proportionally within their containing elements to avoid distortion or over-spilling across devices.
- Media Queries: Using CSS declarations that adjust the appearance of content in response to device properties like width, height, and orientation.
- Mobile-First Approach: Developing the mobile version of a site first and then scaling up for wider screens, with key content accessible across all devices.

These practices are substantiated by diverse sources, such as the topic of flexible grid structures and media queries.

Merits and Advantages

Responsive Web Design has the following advantages:

- Better User Experience: Since RWD adjusts to various screen sizes, it offers an intuitive and consistent user interface on devices.
- Cost-Effectiveness: Having a single responsive website eliminates the necessity for individual mobile and desktop versions, hence cutting down development and maintenance expenses.
- Enhanced SEO: Google prefers mobile-compatible sites, with the potential for enhanced search visibility and rankings.
- Greater Coverage: A responsive design provides a wider audience's access, extending to users from different devices and platforms.

These benefits lead to increased user interaction and can result in higher conversions and customer satisfaction.

Limitations and Challenges

In spite of its advantages, RWD carries some challenges:

- Performance Problems: Loading all resources on every device can result in lengthy load times, particularly on mobile networks.
- Design Complexity: Developing a design that can function well on every device is complex and takes time to develop and test.
- Browser Compatibility: Less contemporary browsers can sometimes be incompatible with RWD methods, providing inconsistent user experiences.
- Navigation Problems: Designing intuitive navigation on smaller screens is challenging, and this may affect usability.
- Solving these issues necessitates a strategic solution and rigorous testing on different devices and browsers.

Implications and Applications

The usage of Responsive Web Design has important implications:

- Business Growth: By ensuring a consistent user experience, companies are able to improve customer satisfaction as well as conversion rates.
- Educational Platforms: Educational institutions can guarantee that learning resources are made available to students on any device, ensuring inclusive education.
- E-commerce: Online stores are able to reach more people and deliver a consistent shopping experience, which can increase sales.
- Government Services: Government websites can provide citizens with improved access to information and services, enhancing engagement and transparency.

Organizations need to implement RWD to remain competitive and address the changing needs of their users.

Conclusion

Responsive Web Design is a critical strategy in the multi-device era, making websites usable, accessible, and effective on multiple platforms. Although it poses some challenges, the long-term advantages of improved user experience, wider reach, and cost efficiency make it an investment worth considering. As technology advances, adopting RWD principles will be critical for organizations that want to stay relevant and satisfy the needs of their diverse user base.

2.5 Performance Optimization in Modern Web Applications

Abstract

This essay explores key tactics for improving web application performance on both front-end and back-end levels. The essay covers strategies like lazy loading, caching, database indexing, and query optimization to enhance usability and efficiency. Through the adoption of these measures, developers are able to markedly improve the efficacy and responsiveness of web applications.

Methodology and Techniques

The essay presents a two-pronged approach to performance optimization:

- Front-End Optimization:
  - Lazy Loading: Delays the loading of non-critical resources until they are actually required, cutting initial load times.
  - Caching: Saves often-used data locally to avoid server requests and delays.

- Back-End Optimization:
  - Database Indexing: Establishes indexes to speed up data retrieval operations.
  - Query Optimization: Optimizes database queries to improve execution performance and response times.

Merits and Advantages

Application of these optimization methods has various advantages:

- Improved User Experience: Smoother interactions and faster load times result in higher user satisfaction.
- Enhanced Efficiency: Better resource utilization decreases server load and operational expenses.
- Scalability: Scalable applications are able to manage more traffic without a decline in performance.

Limitations and Challenges

Though the mentioned techniques are helpful, they are accompanied by some challenges:

- Complexity of Implementation: Optimizing strategies might need extensive modification of current codebases.
- Overhead of Maintenance: Ongoing monitoring and revising are required to maintain performance improvements.
- Potential for Over-Optimization: Over-optimization can result in code obfuscation and make maintenance a challenge.

Implications and Applications

The above optimization methods are relevant across most domains:

- E-Commerce Platforms: Increasing page loading speeds can result in increased conversions.
- Content-Heavy Websites: Content and media loading efficiently enhances user interaction.
- Enterprise Applications: Optimized performance guarantees reliability and responsiveness for high-priority business operations.

Conclusion

The article stresses the need for performance optimization in contemporary web development. Through the use of methods such as lazy loading, caching, database indexing, and query optimization, developers can design applications that are not only efficient but also offer a better user experience. Ongoing assessment and adjustment of these practices are necessary to keep up with changing user demands and technological evolution.

2.6 The Role of Progressive Web Apps in Modern Web Development

Abstract

The article examines Progressive Web Applications (PWAs), explaining what they are for and contrasting them with other relevant web technologies such as responsive sites and native apps. It establishes main differences and similarities, debates benefits and potential drawbacks, and provides guidance on whether PWAs are the best for a business's web presence.

Methodology and Techniques

The research takes a comparative analysis stance, looking at PWAs against responsive websites and native apps. It explores the technical building blocks of PWAs, such as:

- HTML5, CSS3, and JavaScript: They are used to construct the user interface, i.e., the app shell, which is stored on the device of the user for faster loading.

- Service Workers: Background JavaScript code that provides support for features such as offline usage and push notifications.
- Web Service Calls: Used to load and display content on the client side, just like native applications.
- Frameworks: Packages such as Angular and React, which support integrated service workers, making it easy to develop PWA.
- This approach offers a complete knowledge of how PWAs work and where they stand in contemporary web development.

Merits and Advantages

The paper discusses some merits of PWAs:

- Cost-Effectiveness: PWAs obviate the necessity to create different apps for various platforms, saving on development and maintenance expenses.
- SEO-Friendly: PWA content is displayed in search engine results, which improves visibility.
- Independence from App Stores: PWAs can be reached directly via browsers, avoiding app store limitations.
- Less Data Usage: PWAs use less data than standard websites, which improves performance, particularly in places with poor connectivity.

Limitations and Challenges

Though they have many benefits, PWAs do have some limitations:

- Development Complexity: Implementing PWA features into a standard website requires extra time and effort.
- Limited Hardware Access: PWAs have limited access to device hardware capabilities

because of browser limitations.

- Browser Compatibility: Older browsers do not support all PWA features, which reduces their audience.
- iOS Support: Although PWAs are supported on iOS, the functionality is partially restricted compared to Android.
- App Store Visibility: Lack of visibility on app stores can lower discoverability and perceived legitimacy.

Implications and Applications

PWAs provide an appealing solution for companies wanting an affordable, cross-platform web presence. They are especially useful for:

- Businesses: To give a native app-like experience without the cost.
- Users: In areas with low internet coverage, since PWAs have offline support.
- Developers: To target a wider audience without having to create various platform-based apps.

But for apps needing strong hardware integration, native apps would still be the way to go.

Conclusion

The paper concludes that although PWAs display many benefits, they are not universally applicable. Their capacity to be a mainstay in web development is high, but adoption is contingent on overcoming existing limitations and adding support across multiple platforms.

2.7 Accessibility in Web Development: Standards and Implementation

Abstract

Web accessibility guarantees websites' usability by everyone, including people with disabilities. This paper gives a thorough introduction to applying accessibility methods for web design, focusing on major standards and best practices. It covers the Web Content Accessibility Guidelines (WCAG), which provide a system for making web content perceivable, operable, understandable, and robust. Legal laws like the Americans with Disabilities Act (ADA) and the Canadian Accessibility Act (ACA) are also addressed.

Strategies such as semantic HTML use, color contrast improvement, and code validation are also emphasized in the paper. Tools like WAVE, Axe, and Lighthouse are reviewed, along with the significance of user testing to detect real-world problems. Successful implementation case studies offer real-world advice for web designers and developers.

Techniques and Methodology

- Standards Referenced: The paper is focused on WCAG guidelines, specifically versions 2.0 and 2.1, which describe principles to make web content accessible.

- Techniques Used:
    - Using semantic HTML for improving the structure of content.
    - Color contrast optimization to help people with visual disabilities.
    - Code validation to ensure compliance with accessibility guidelines.
    - Applying user testing to detect and fix accessibility issues.

- Tools Under Discussion: The article analyzes tools such as WAVE, Axe, and Lighthouse that aid in analyzing and enhancing web accessibility.

Advantages and Benefits

- Improved User Experience: Adoption of accessibility guidelines enhances usability for everybody, not disabled users alone.
- Compliance with Law: Following guidelines such as WCAG ensures that businesses are legally compliant, less likely to be in court.
- Wider Audience Coverage: Accessible websites can be accessed by a wider audience, including people with different disabilities.
- Better SEO: Most accessibility practices, like semantic HTML, also improve search engine optimization efforts.

Limitations and Challenges

- Difficulty in Implementation: Implementing accessibility features is complex, particularly for older websites that need retrofitting.

- Limitations of Tools: Although tools like WAVE and Axe are useful, they might not detect all accessibility problems, so manual checks are necessary.
- Evolving Standards: Accessibility standards are continually evolving, requiring ongoing education and adaptation by developers.

Implications and Applications

- Policy Development: The insights from the paper can inform organizational policies to prioritize accessibility in web development projects.
- Educational Resource: The paper serves as a valuable resource for educating developers and designers on best practices in web accessibility.
- Framework for Audits: The organizations can adopt the above-mentioned standards and tools as a framework to conduct accessibility audits.

Conclusion

The paper highlights the necessity of incorporating accessibility guidelines into web development to establish accessible digital spaces. By complying with guidelines such as WCAG and employing tools like WAVE and Axe, developers can increase the usability of websites for everybody. Continued attention to accessibility is a means to ensure compliance with legal requirements and to create a more inclusive web.

2.8 Integrating Artificial Intelligence into Web Development

Abstract

The research delves into the influence of Artificial Intelligence (AI) on web development, with a focus on automation, personalization, and decision-making. It examines how AI-based tools and practices streamline development work by automating code generation, testing, and maintenance, resulting in enhanced efficiency and minimized human errors. The article also discusses the way AI improves customized user experiences by leveraging data-driven insights to personalize content, suggestions, and interactions based on individual user behavior and preferences. Moreover, it also talks about the impact of AI on web development decision-making processes, namely project management and user experience design, through data-driven recommendations and predictive analytics.

Methodology and Techniques

The research utilizes a qualitative research approach with a review of current literature and case studies for the analysis of AI application within web development. It explores some of the AI technologies, including machine learning, natural language processing, and computer vision, and how they are being incorporated into web development workflows. The research also evaluates the efficacy of AI tools in automating code generation and testing, among others, as well as their use in user experience personalization and decision-making in project management and design.

Merits and Advantages

- Increased Automation: AI automates repetitive tasks, including code generation and testing, making it more efficient and less prone to human error.
- Personalized User Experiences: AI processes user data to personalize content, recommendations, and interactions, resulting in more engaging and relevant user experiences.
- Enhanced Decision-Making: AI offers predictive analytics and data-driven insights that aid project management and user experience design, enabling informed decision-making.
- Higher Efficiency: Automating repetitive work, AI enables developers to concentrate on more intricate and innovative parts of web development.

Limitations and Challenges

- Data Security and Privacy: Web development involving AI frequently deals with large sets of user data, thus invoking data privacy and security issues. Compliance with laws such as GDPR and CCPA is critical to protect the information of users.
- Integration Complexity: Integrating AI technologies into existing web development processes may be difficult and capital-intensive, involving large investment in infrastructure and skills.
- Ethical Considerations: The integration of AI raises ethical concerns, including algorithmic bias and job loss, and therefore the imperative to develop equitable and transparent AI systems.

Implications and Applications

The incorporation of AI in web development has far-reaching consequences for different sectors. It allows for the development of more responsive, adaptive, and user-friendly web solutions. AI-based tools can increase engagement by offering personalized content and experiences. AI also facilitates optimizing development operations, resulting in quicker deployment of web applications. The uses of AI in web development extend to e-commerce, healthcare, education, and entertainment, among others, with customized experiences and effective delivery of service.

Conclusion

Artificial Intelligence is revolutionizing web development by automating processes, tailoring the user experience, and generating data-driven insights that augment decision-making. Although challenges like data privacy, integration complexity, and ethical issues abound, the advantages of AI incorporation in web development are immense. The future of web development is set to be all the more driven by AI, resulting in more innovative and efficient web solutions.

2.9 Microservices Architecture in Web Development

Abstract

Microservices Architecture (MSA) is a design pattern that organizes an application into a group of loosely interconnected services, one for each distinct business function. This style strengthens modularity by making applications more straightforward to write, test, deploy, and scale. MSA has also become popular because it can allow agile development styles and continuous delivery. It complicates service-to-service communication, data consistency, and system observability.

Methodology and Techniques

Implementation of MSA entails:

- Service Decomposition: Decomposing the application into numerous, independent services following business capabilities.

- Containerization: Leveraging technologies such as Docker to encase services for ensuring consistency in environments.
- Orchestration: Leveraging tools like Kubernetes for orchestrating the deployment, scaling, and network configuration of services.
- Continuous Integration/Continuous Deployment (CI/CD): Leverage automated testing and deployment pipelines for maximizing the efficiency of development.
- API Gateways: Use of gateways for processing requests, routing, and load balancing.
- Service Discovery: Utilizing registries to allow services to discover and communicate dynamically with one another.

Merits and Benefits

- Scalability: MSA enables services to be scaled separately, ensuring optimal utilization of resources.
- Flexibility: It allows different technologies and frameworks to be utilized for various services.
- Resilience: Failure in one service is not critical to others, promoting system dependability.
- Faster Time-to-Market: Separate deployment of services ensures quicker feature delivery.
- Better Maintainability: Smaller codebases for each service make debugging and upgrading easier.

Limitations and Challenges

- Complexity: Having many services complicates the system.
- Data Consistency: Maintaining consistency among distributed services may be tricky.
- Inter-Service Communication: Processing communication among services has overhead and may incur latency.
- Deployment Overhead: Deploying many services requires strong orchestration tools.
- Monitoring and Debugging: Monitoring problems across services requires sophisticated monitoring tools.

Implications and Applications

MSA is especially useful for:

- Large-Scale Applications: Like e-commerce websites and social networks.
- Organizations Implementing Agile Practices: Supports iterative development and rapid deployment.
- Cloud-Native Platforms: Designed for cloud infrastructure and services.
- But MSA won't be suitable for small applications because of its complexity.

Conclusion

Microservices Architecture has considerable benefits in scalability, flexibility, and maintainability, and it is appropriate for large-scale and complex applications. But organizations should be ready to face the related challenges, such as higher complexity and the necessity of strong infrastructure. Proper consideration and planning are necessary when implementing MSA to maximize its benefits.

2.10 Web Development Trends: A Survey of Emerging Technologies

Abstract

This article covers several new technologies in web development, including Progressive Web Applications (PWAs), Single Page Applications (SPAs), AI-generated frontends, serverless architectures, and innovations such as CSS card designs and VR integration. It contrasts these technologies with the current ones on characteristics such as focus, performance, scalability, security, and innovation. It also incorporates graphical data, including bar graphs, on the growing use of AI and Machine Learning in web development. Ending on findings from research and literature studies, it is intended to educate and motivate web development enthusiasts.

Methodology and Techniques

- Literature Review: The paper does an exhaustive review of literature and research work to find and discuss new web development technology.
- Comparative Analysis: It compares new technologies with old ones on different parameters like performance, scalability, and security.
- Data Visualization: Uses bar graphs to represent trends, like increasing use of AI and Machine Learning in web development.

- Expert Insights: Formulates conclusions based on a diverse set of research papers from all over the world, giving an all-around view of the topic.

Merits and Advantages

- Improved User Experience: Technologies such as PWAs and SPAs provide more interactive and smoother user interfaces.
- Improved Development Efficiency: Frontends generated by AI automate UI/UX, thus decreasing development effort and time.
- Scalability and Flexibility: Serverless architectures ensure scalable solutions without having to maintain server infrastructure.
- Innovative Design Capabilities: Improvements in CSS and VR integration allow for more innovative and immersive web designs.
- Data-Driven Insights: The presence of visual data assists in comprehending the trends of adoption of different technologies.

Limitations and Challenges

- Implementation Complexity: The implementation of new technologies can involve drastic changes to current systems and processes.
- Compatibility Issues: Merging new technologies with older systems can present compatibility issues.
- Learning Curve: Developers can need to learn new skills and knowledge to be able to properly use advanced technologies.
- Resource Intensive: Certain technologies, such as AI and VR, can be very resource-intensive and may need large computational resources and infrastructure.

Implications and Applications

- Industry Adoption: The results of the paper can help businesses adopt appropriate technologies to improve their web platforms.
- Educational Resources: Is a useful resource for students and teachers in web development.
- Future Research Directions: Points out areas for future research, including the use of blockchain in web development.

Conclusion

The paper wraps up by stressing the revolutionary effect of future technologies on web development. It underscores the need to keep up with the latest technological developments to avoid falling behind in the digital age. With its elaborative examination of present trends and future implications, it is a complete manual for developers and organizations looking to innovate and improve their web development processes.

CHAPTER 3

RESEARCH GAPS OF EXISTING METHODS

Web development has seen tremendous growth in the last two decades. From static HTML websites to dynamic full-stack applications driven by robust JavaScript frameworks and cloud computing, the field has consistently evolved to accommodate new technologies and user needs. Despite the advancements, there are some research gaps in existing web development practices. These gaps impede performance, security, scalability, accessibility, developer productivity, and user satisfaction.

This part discusses some of the most important research gaps in web development, including user experience (UX) design, front-end and back-end integration, performance optimization, accessibility, web security, scalability, and developer tooling.

3.1 User-Centered Design and Accessibility Limitations

Although contemporary development patterns support responsive and beautiful designs, they fall behind in true user-centered design (UCD) and worldwide accessibility. In the majority of developers, there's a preference mainly for the looks and correct function, even bypassing or demoting accessibility.

Research Gaps:

- Lack of smarter tools to autonomously test and optimize web accessibility.
- Poor implementation of accessibility guidelines (such as WCAG) within mainstream development streams.
- Inadequate support for users with cognitive disabilities or alternative input device users.
- Inadequate study of culturally adaptive interfaces that customize UI/UX according to geographic, linguistic, or sociocultural contexts.

Potential Directions:

- AI-based accessibility testing tools.
- Design-time constraint frameworks that incorporate UCD principles.

- Machine learning model-based dynamic accessibility layering.

3.2 Integration Challenges Between Front-End and Back-End

Despite full-stack frameworks on the rise, front-end to back-end system integration remains problematic—particularly for large-scale or distributed systems.

Research Gaps:

- Poorly optimized or unnecessary data binding between UI components and backend APIs.
- No standardization of API contracts resulting in integration bugs and added debugging time.
- Performance bottlenecks as a result of tight coupling or excess roundtrips between server and client.
- Fragmentation in real-time data synchronization across front-end and back-end (e.g., WebSockets, polling, SSE).

Potential Directions:

- Schema-aware integrated development environments (IDEs) which auto-sync API contracts between stacks.
- Research towards more unified runtime environments (e.g., WebAssembly-based shared logic).
- New architectural patterns facilitating backend-less logic with edge functions and serverless paradigms.

3.3 Performance Optimization Limitations

Performance is one of the essential pillars of successful web development. Yet current optimization methods still lag behind in providing predictable and scalable performance, especially on mobile or low-end hardware.

Research Gaps:

- Limited availability of automated performance bottleneck detection tools that take into consideration real-world usage environments.

- Very little adaptive resource loading methods across various network conditions or device capabilities.
- Poor JavaScript bundling and tree-shaking practices that end up producing unnecessarily large payloads.

- Inadequate management of third-party scripts (plugins, ads, analytics) that slow page speed and responsiveness.

Potential Directions:

- AI/ML-powered build tools that pre-optimize front-end payloads according to usage prediction.
- Real-time user-focused performance monitoring dashboards.
- Creation of intelligent CDNs that compress dynamically based on client profile or bundle/deffer resources accordingly.

3.4 Web Security and Privacy Issues

The growing use of web platforms for sensitive operations (healthcare, banking, identity) demands watertight security measures. Yet, the majority of existing approaches are reactive, patch-based, or too dependent on third-party libraries.

Research Gaps:

- Limited automation in the identification of logic errors or business logic vulnerabilities.
- Excessive dependency on client-side validation with insufficient server-side protection.
- Unstandardized security practices across development teams and absence of standardized security policies in small-to-medium web projects.
- Poor data privacy practices in the face of rising regulatory requirements (e.g., GDPR, CCPA).

Potential Directions:

- Studies on automated threat modeling tools embedded within development pipelines.

- Compliance-generating privacy-by-design tools for producing compliant web forms and storage logic.
- Smart validation frameworks that impose both syntactic and semantic validation rules.

3.5 Scalability and Maintenance Challenges

As software becomes larger, scalability and maintainability become the primary challenges, especially for monolithic or badly modularized codebases.

Research Gaps:

- Lack of effective tools for automated refactoring of legacy web applications.
- Inadequate observability in microfrontend architectures resulting in broken error tracking and erratic UI state.
- Insufficient mature support for reusable, modular components between ecosystems (e.g., code reuse across React, Vue, Angular).
- State management pattern fragmentation in large-scale applications.

Potential Directions:

- Component architecture frameworks facilitating effortless code reuse between UI ecosystems.
- Economically scalable testing approaches developed specifically for microfrontend applications.
- Exploration of self-healing UIs that can recover or adapt to inconsistent states in real-time.

3.6. Developer Experience and Tooling Gaps

Modern web development environments' complexity may lead to sharp learning curves, particularly for novice users or small teams.

Research Gaps:

- Sparse application of AI and NLP in code completion beyond autocompletion (e.g., developer intent detection).
- Suboptimal real-time collaboration tools with embedded development, design, and

deployment workflows.

- Variable documentation habits and absence of tooling to automatically generate relevant documentation from codebases.
- Gaps in hybrid app or server-side rendering (SSR) app debugging abilities.

Potential Directions:

- AI-driven code partners providing contextual coding proposals, refactoring, and documentation.
- Auto-create visual flowcharts or UI wireframes from code IDE plugins.
- Unified cloud-based Integrated Development Environments (IDEs) combining development, testing, and deployment.

CHAPTER 4

PROPOSED METHODOLOGY

The approach used for the internship assignments and final project was iterative and modular development. Each assignment was addressed in sequential steps—planning, design, development, testing, and deployment—to facilitate organized execution, maintainability, and ongoing learning during the internship. The Agile philosophy was loosely applied, with room for flexibility and enhancements based on feedback and testing.

4.1 Website Replication Task

Objective

To replicate the design and layout of an existing website using HTML, CSS, and JavaScript in order to develop skills in front-end development methods and responsive design principles.

Methodology

Phase 1: Requirement Analysis

- Chose a website to replicate.
- Utilized browser developer tools (Inspect Element) to examine page structure, layout grids, and styling methods.
- Located essential elements like headers, navigation menus, hero sections, cards, forms, and footers.

Phase 2: Design and Layout Planning

- Broke the web page into reusable sections.
- Created a wireframe of the layout for clarity.

Phase 3: Front-End Development

- Used semantic HTML5 to build the page structure.
- Used CSS3 for styling and responsiveness.
- Used Flexbox and Grid systems for layout.
- Used media queries for screen size adaptability.

Phase 4: Testing and Debugging

- Cross-checked for consistency in layout across browsers.
- Tested the design responsiveness across different device viewports.

Tools and Technologies Used

1. HTML5, CSS3, JavaScript
2. Chrome Developer Tools
3. Git for version control

4.2. Back-End Form Development with Email Capability

Objective

Create a backend system that processes the form submission and sends an auto-email to the email address submitted by the user.

Methodology

Phase 1: Requirement Gathering

- Built a simple contact form with inputs for name, email, subject, and message.
- Specified expected behavior: form submission would initiate an email with the filled-in details.

Phase 2: Server Setup

- Built a Node.js environment and initialized a project using npm init.
- Installed and set up Express.js for HTTP request handling.

Phase 3: Form Handling

- Added POST route to receive form data through middleware such as body-parser.
- Verified input fields to maintain data integrity prior to processing.

Phase 4: Email Integration

- Utilized Nodemailer for sending emails through SMTP or service-based transport.
- Set up sender credentials and dynamic message content depending on form inputs.

Phase 5: Testing and Validation

- Manually tested with different input values and email addresses.
- Checked both success and failure points (e.g., invalid email, blank fields).

Tools and Technologies Used

- Node.js, Express.js
- Nodemailer
- Postman (used for API testing)
- JavaScript (ES6+)

4.3. Blog Website Using EJS and Express.js

Objective

To develop a complete blog web application where users can create, read, and navigate posts using templating and server-side rendering with EJS.

Methodology

Phase 1: Project Planning

Outlined core features:

- Home page featuring all blog posts
- Dynamic post page for single articles
- Compose page to post new posts
- Designed folder structure and MVC-style architecture.

Phase 2: Environment Setup

- Created Node.js project and installed Express.js and EJS.
- Organized the project in views, public, and routes directories.

Phase 3: Front-End with EJS

Developed dynamic templates for:\

- Home page (index.ejs)

- Compose form (modify.ejs)
- Utilized partials for reusable elements such as headers and footers.

Phase 4: Routing and Logic Development

Established routes:

- GET / – Home page with all posts
- GET /posts/:id – Dynamic blog post
- GET /compose and POST /compose – Creation of posts

Temporarily stored the posts in an in-memory array for ease (or else used a JSON file or low-level DB like NeDB or MongoDB for storage).

Phase 5: Testing and Iteration

- Tested all CRUD operations manually.
- Confirmed template rendering and display of dynamic data.

Tools and Technologies Used

1. Node.js, Express.js
2. EJS templating engine
3. Bootstrap (for styling)
4. JavaScript (for server code)
5. Git and GitHub for tracking versions

4.4 Development Workflow

A broad iterative process was used for every task:

- Planning: Establish scope and task breakdown.
- Design: Wireframe or sketch the layout or logic.
- Implementation: Code and implement necessary functionality.
- Testing: Manually debug and validate output.
- Feedback and Iteration: Improve based on results or mentor feedback.
- Documentation: Keep code comments, Git commits, and progress notes.

CHAPTER 5

OBJECTIVES

The internship was created to allow practical exposure to full-stack web development, with the application of theoretical learning in practice. The internship was intended to facilitate a transition from elementary academic knowledge to hands-on practice by participating in well-structured activities encompassing both front-end and back-end development. The primary goals of the internship were technical and professional in nature, aimed at developing competency, confidence, and clarity in web application development.

1. Consolidating Core Web Development Abilities

A major objective of the internship was to enhance my knowledge of fundamental web technologies—HTML, CSS, and JavaScript—through recreating an existing website. This activity enabled me to gain insight into how actual sites are styled and structured, how responsiveness is attained, and how interactivity is achieved.

Sub-objectives:

- In order to enhance semantic HTML structuring for improved accessibility and SEO.
- To develop CSS styling skills, including layout systems (Flexbox, Grid).
- To learn the principles of responsive design with media queries and adaptive design patterns.
- To use JavaScript to implement dynamic behavior in the front end (e.g., interactive components, DOM manipulation).

This phase formed the foundation for subsequent full-stack activities and contributed to the building of a keen eye for detail and design.

2. Getting Hands-on Experience in Backend Development

Another major goal was to get past front-end logic and obtain working experience with server-side development with Node.js and Express.js. The task of backend development was focused on adding logic to process form data and send out email notifications.

Sub-objectives:

- To be familiar with the organization of a Node.js project and utilize npm for package management.
- To have an Express server set up with the ability to handle routes, middleware, and form data.
- To include third-party modules like Nodemailer for managing automated emails.
- To use input validation, error handling, and feedback for form submission.
- To test real-world scenarios like contact forms, feedback forms, or registration emails.

This goal was crucial in learning how client-server communication is achieved and how user-provided data can be processed securely and efficiently.

3. Using Full-Stack Development in a Real-World Project

The last and most advanced objective was to create a full-stack blog website using EJS for dynamic page rendering and Express.js as the web server framework. This project sought to integrate all learning outcomes from previous tasks and apply them in a cohesive, functional application.

Sub-objectives:

- To implement a scalable and modular project organization with MVC (Model-View-Controller) architecture.
- To utilize EJS templating to render dynamic server-side data in HTML views.
- To create routing logic for posting, reading, and showing blog posts.
- To process user input through form submission and store data in memory (or ready for integration with a database).
- To get familiar with server-side rendering concepts and understand differences with client-side rendering.
- To construct reusable components (partials) such as headers and footers for maintainable design.

This objective served as a capstone to the internship, encouraging a deeper understanding of

full-stack workflows, logic handling, and project management.

4. Developing Problem-Solving and Debugging Skills

An important personal and professional goal was to build strong debugging and troubleshooting skills. Through trial, error, and feedback, I learned to identify issues, understand error messages, and trace bugs across the stack.

Sub-objectives:

- To utilize browser dev tools for front-end debugging and inspection.
- To use console logs and terminal outputs for back-end debugging.
- To mock and test API calls using tools such as Postman.
- To use version control properly for keeping changes and testing code without danger.

These are all essential skills for any real-world developer and were learned organically through the experiential nature of the internship.

5. Enhancing Project Planning and Development Discipline

The internship also sought to inculcate a professional workflow mentality, such as version control, task decomposition, and time management. Although the projects were individual, they were done in a semi-structured Agile-like manner.

Sub-objectives:

- To decompose tasks into manageable modules and schedule development accordingly.
- To employ Git for source control and have a clean commit history.
- To document the development process through comments and GitHub README files.
- To learn and implement Express.js for server creation and routing.
- To learn how middleware is used in Express-based applications.

This assisted in gaining exposure to web development tools and frameworks that will be useful in future team-based development or freelance projects.

6. Exposure to Web Development Tools and Frameworks

During the internship, various tools, frameworks, and libraries were exposed to simplify

development and improve productivity.

Sub-objectives:

- To learn and implement Express.js for server creation and routing.
- To learn how middleware is utilized in Express-based applications.
- To leverage EJS as a template engine for the dynamic content generation.
- To implement use of npm as a module/package manager for dealing with dependencies.
- To familiarize oneself at the basic level with deployment readiness (file organization, environment preparation).

These tasks provided an increased exposure to the ecosystem of web development and the interoperability between different tools and frameworks to deliver scalable applications.

CHAPTER 6

SYSTEM DESIGN & IMPLEMENTATION

The system architecture and implementation strategy for the internship projects were structured to demonstrate real-world web development practices, involving clear separation of concerns between front-end and back-end layers, RESTful routing, and modular file organization. Each project was implemented as a standalone unit with focused design decisions to suit the task's scope.

6.1. Website Replication – Front-End System

Architecture Overview:

The task of website replication used only client-side with HTML, CSS, and JavaScript. None of it contained server logic or dynamic data.

Design Components:

- HTML: Designed the page layout with semantic elements.
- CSS: Managed layout, color palettes, font choices, and responsiveness.
- JavaScript: Controlled interactivity like menus or buttons.
- Media Queries: Controlled responsiveness on various devices.
- Developer Tools: Were used to analyze and reproduce layout and design logic.

Implementation Details:

Organized in folders:

- /index.html
- /styles/style.css
- /scripts/main.js

Utilized browser dev tools to determine CSS properties.

Refactored code to be reusable and responsive, and for responsiveness.

6.2. Form with Email Functionality – Backend Integration

Architecture Overview:

This application utilized a client-server architecture in which a static front-end form communicated with a Node.js/Express back-end server that would send an email when the form was submitted.

Design Components:

- HTML Form: Front-end input fields for name, email, and message.
- Express Server: Processes form data using a POST request.
- Nodemailer: Sends an auto email based on SMTP configuration.
- Validation Middleware: Validates fields submitted are properly formatted.

Implementation Details:

Server routes:

- GET / – Shows the form.
- POST /submit – Processes form submission.

Email logic involved dynamic message construction.

Used environment variables for SMTP credentials (best practice).

6.3. Blog Website – Full Stack with Express.js and EJS

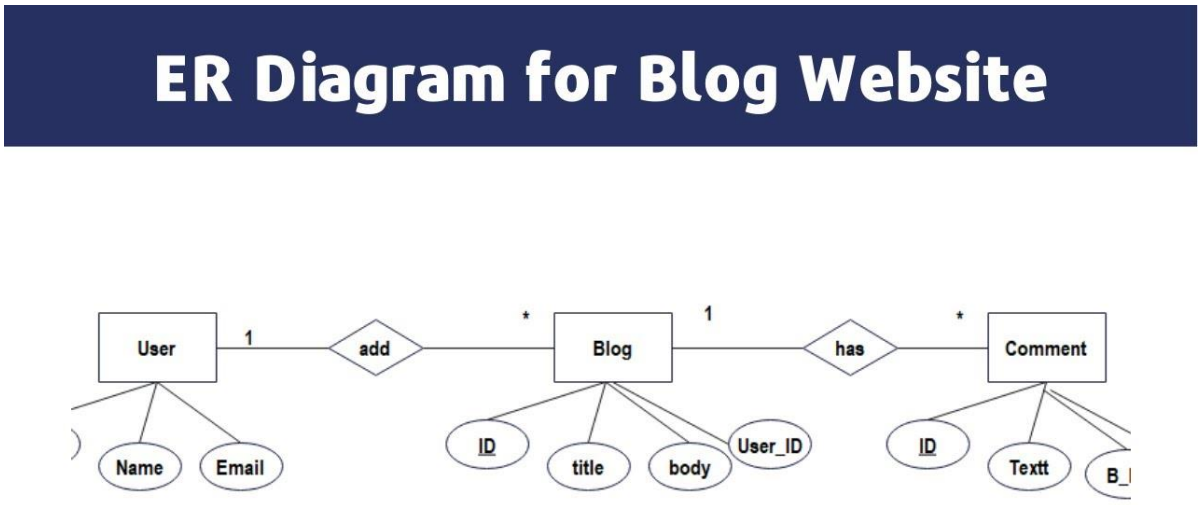


FIG. 6.1: Entity-Relationship diagram for a blog website

Architecture Overview:

This was a full-stack MVC-type system. It had dynamic routing, template rendering, and long-term blog post storage (in-memory for simplicity).

Design Components:

EJS Templates: Dynamic HTML rendered depending on route information.

Express.js Server: Handled routes and rendered views.

Routing Layer:

- GET / – Renders list of posts.
- GET /compose – Renders form to create new post.
- POST /compose – Saves blog content when submitted.
- GET /posts/:postid – Renders individual blog post.

In-Memory Data Store: Saved blog posts as JavaScript objects (arrays).

Implementation Details:

Folder structure:

- /views: EJS files (home.ejs, post.ejs, compose.ejs)
- /public: CSS and assets
- /routes: (Optional) logic routing
- Utilized EJS partials for reusable headers/footers.
- Added basic error handling and form validation.

CHAPTER-7

TIMELINE OF THE INTERNSHIP

(GANTT CHART)

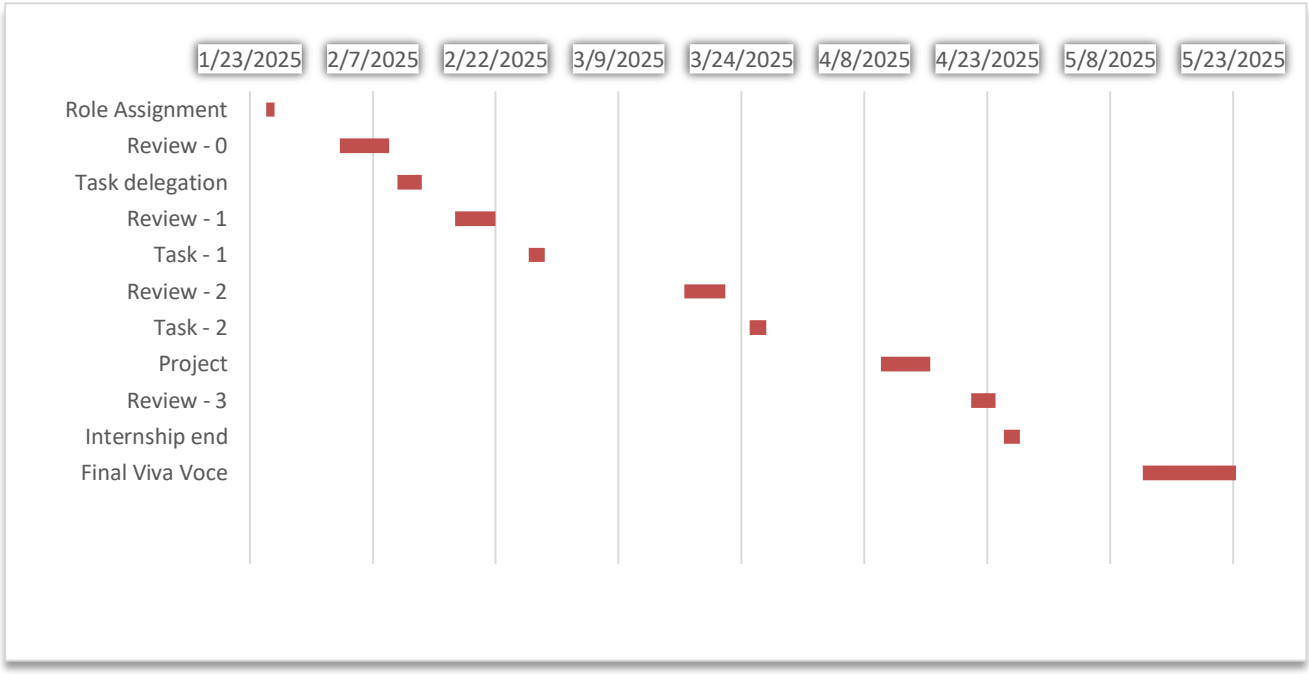


FIG. 7.1: Timeline of the Internship

CHAPTER 8
OUTCOMES

The internship was an all-encompassing learning experience that filled the gap between theoretical knowledge and industry-standard practice in web development. By accomplishing three significant tasks—website replication, form backend with email support, and a blog website project using EJS and Express.js—several significant outcomes were achieved. These outcomes range across technical, analytical, and soft skills, along with a better understanding of development workflows.

1. Improved Front-End Development Skills

The website cloning exercise served to solidify core principles of HTML, CSS, and JavaScript. By meticulous observation and manual recreation of a live website, I achieved:

- A better grasp of semantic HTML structuring for readability and accessibility.
- Hands-on experience in designing responsive layouts with Flexbox, CSS Grid, and media queries.
- Enhanced confidence in debugging layout bugs and verifying cross-device compatibility.
- Practical experience with UI/UX principles through examination of real-world layout techniques and visual hierarchy.

2. Acquired Practical Backend Development Experience

The backend form submission assignment exposed me to the fundamentals of server-side programming and routing, data handling, and asynchronous operations.

- Constructed a Node.js and Express.js server from ground up.
- Acquired knowledge of how to use middleware such as body-parser to handle incoming data.
- Implemented Nodemailer successfully to send automatic emails on form submission.
- Acquired knowledge on how to validate input data, manage errors nicely, and safeguard sensitive credentials through environment variables.
- Gained hands-on exposure to tools such as Postman for endpoint testing and emulating API calls.

3. Built Full-Stack Application Skills

The blog site project enabled the combination of front-end and back-end skills into a functional web application.

- Built and deployed a multi-page, dynamic web site with EJS templating and Express.js routing.
- Managed user-submitted content and displayed blog posts dynamically.
- Found out how to organize projects using the Model-View-Controller (MVC) pattern for maintainability and scalability.
- Built a solid understanding of RESTful routing, dynamic parameters, and in-memory data storage.
- Used partial templates to maintain UI elements such as headers and footers throughout pages.

4. Enhanced Debugging and Problem-Solving Abilities

Across all assignments, numerous issues were faced—such as layout inconsistencies, server errors, and input validation bugs—that were valuable learning experiences.

- Become proficient at utilizing browser developer tools for CSS and DOM inspection.
- Resolved server errors using terminal logs and custom error messages.
- Understood how to isolate bugs by dividing large problems into small testable units.
- Established a more systematic method of reading documentation and debugging issues independently.

5. Development Tool Exposure and Best Practices

This internship involved the first practical experience with industry-applicable tools and processes:

- Used Git and GitHub for version control, learning how to commit changes, roll back, and keep track of code repositories.
- Used basic project structuring conventions for readability and scalability.
- Adopted techniques such as modular coding, configuration with.env files, and code commenting for ease of understanding.

6. Time and Task Management Skills

- Successfully working on various tasks with timelines enhanced my skill in planning, prioritization, and work delivery.
- Divided each task into rational steps: planning, developing, testing, and reviewing.
- Maintained balancing learning new tech with delivering usable outcomes.
- Maintained juggling personal growth and internship deliverable expectations.

7. Increased Confidence and Readiness for Future Projects

At the end of the internship, I gained not only technical skills, but also the self-confidence to:

- Start and finish a project independently.
- Work with full-stack technology in an integrated setup.
- Explain technical work through documentation and presentation.
- Consult open-source communities and official documentation when stuck.

The internship was highly beneficial and productive, offering a strong foundation for a future career in web development. It provided:

- Practical experience in designing and building functional websites.
- Confidence in using both front-end and back-end technologies.
- A clear understanding of the development lifecycle, from idea to deployment.

These outcomes have positioned me well for more advanced roles or freelance projects in the web development domain, with the ability to adapt quickly to real-world problems and development environments.

CHAPTER 9

RESULTS AND DISCUSSIONS

The internship gave me a comprehensive experience in full-stack web development by implementing three primary projects: website duplication, form with backend email support, and a blog web application using EJS and Express.js. Each project had distinct deliverables and learning slopes, and successful implementation and extensive skill development ensued.

9.1. Website Replication

Results:

- A working front-end version of the target site was successfully duplicated.
- The design was identical to the original in structure, responsiveness, and design aesthetics.
- The site was responsive on various screen sizes such as mobile, tablet, and desktop.
- All interactive elements like navigation menus and buttons functioned correctly.

Discussion:

This assignment greatly enhanced my front-end development abilities. At first, issues were replicating spacing, responsive behavior, and supporting various screen resolutions. These were addressed by:

- Utilizing Chrome DevTools for layout inspection and debugging.
- Utilizing CSS Flexbox and Grid to deal with intricate layouts.
- Utilizing media queries for responsive styling.

I also came to understand refactoring CSS into modular pieces for better maintainability. Overall, this exercise sharpened my structural and visual grasp of front-end design and UI/UX best practices in real-world scenarios.

9.2. Form Submission with Email Integration

Results:

- A web form was built to gather user information (name, email, message).
- A backend using Node.js was established by Express.js to process form submissions

through POST requests.

- The submitted form data was successfully used to trigger an email by employing Nodemailer.
- Input validation was done to avoid wrong or blank submissions.
- Form responses were indicated through success/error messages to notify the user.

Discussion:

This assignment familiarized me with backend processing and real-time client-server communication. Main challenges and how I solved them are:

- SMTP authentication errors: Resolved using secure app passwords and environment variables.
- Processing form data: Learned to utilize Express middleware (express.urlencoded) for the processing of form input.
- Email formatting: Exercised crafting plain-text and HTML-based emails.

The exercise provided actual exposure to REST practices, async programming, and how backend systems process user inputs and third-party integrations.

9.3. Blog Website Project (EJS + Express.js)

Results

- A dynamic, full-stack blog website was created using EJS templates and Express.js routing.
- The main page showed all blog posts created.
- There was a "Compose" page where users could create new blog posts through a form.
- Single blog posts were displayed on separate pages using dynamic routing (/posts/:postTitle).
- Content was temporarily saved in an in-memory array (no database used).

Discussion:

This project brought together my learning into a whole full-stack application. Main learnings and thoughts:

- Dynamic Routing: Learned how to leverage route parameters to render specific

content.

- Server-side Rendering: EJS enabled me to inject dynamic content into HTML, giving the app an interactive and lively feel.
- Template Reusability: I employed partials to prevent code duplication for headers, footers, and page structure.
- Data Handling: Storing and working with data in arrays while not using a database gave insight into logic and scalability concerns in the future.

Challenges were:

- Dealing with undefined routes or blank post submissions (resolved with conditional checks and error messages).
- Understanding route hierarchy and preventing URL conflicts (e.g., /compose vs /posts/:title).
- This project provided a good foundation for grasping MVC design concepts, even without official database integration.

9.4. Cross-Project Observations

Technology Stack Growth:

Throughout the internship, I evolved from working with front-end tools alone to encompassing full-stack functionality. This multi-layered learning enabled me to comprehend how various technologies and layers communicate in a web application.

Debugging Experience:

- As complexity grew, I became more skilled at debugging:
- Front-end layout problems through DevTools.
- Backend server problems through terminal logs and console.log.
- Email send failure using Nodemailer's error logs and SMTP configurations.
- Code Management and Clean Practices

I exercised keeping code modular and legible. Structuring files became more organized, with separation of concerns between views, routes, and assets.

CHAPTER 10

CONCLUSION

The web development internship has been an incredibly enriching and life-changing experience. Throughout the internship, I was able to apply theoretical concepts to practical application by participating in hands-on development activities, each of which made a meaningful contribution to my development as a developer.

By completing the website replication exercise, I had a solid grounding in front-end development, such as responsive design, layout structuring, and UI analysis. The exercise honed my detail-orientedness and improved my skill to translate design to code effectively.

The email and form submission functionality project exposed me to back-end development. Through interactions with Node.js, Express.js, and Nodemailer, I was taught how to work with user input, validate data, and send requests to third-party services. This provided a real-world look at server-side logic and the flow of form data within a web application.

The most informative and detailed aspect of the internship was creating a blog website with EJS and Express.js. The full-stack project assisted me in bringing client-side rendering and server-side logic together, structuring code through the MVC pattern, and grasping dynamic routing. It was the culmination of everything I had acquired so far, and it allowed me to create an actual application from the ground up that managed user input, dynamically rendered pages, and organized content.

Apart from technical competence, this internship provided me with useful soft skills like time management, self-teaching, problem-solving, and doing tasks in a systematic way. It also exposed me to development best practices such as modular coding, exception handling, and the value of clean and sustainable code.

Overall, the internship has equipped me with the confidence to handle more challenging web development tasks. It has not only consolidated my current knowledge but also extended my skill as a full-stack developer. I now feel better positioned to contribute positively in a

professional development setup and handle freelance or team-based web projects in the future.

REFERENCES

1. **A Comprehensive Review of Web Designing and Web Development: Concepts, Practices and Trends,** Elakiya K, International Journal of Research Publication and Reviews, Vol 4, No 4, April 2023
https://ijrpr.com/uploads/V4ISSUE4/IJRPR11646.pdf

2. **"Evaluating Web Frameworks: A Comparative Study for Selecting the Optimal Technology Based on Development Requirements"** by Ayod Bhad (February 2025)
https://www.researchgate.net/publication/389129746_Evaluating_Web_Frameworks_A_Comparative_Study_for_Selecting_the_Optimal_Technology_Based_on_Development_Requirements?utm_source=chatgpt.com

3. N. Teodoro and C. Serrão, "**Web application security: Improving critical web-based applications quality through in-depth security analysis,"** *International Conference on Information Society (i-Society 2011)*, London, UK, 2011, pp. 457-462, doi: 10.1109/i-Society18435.2011.5978496.
https://ieeexplore.ieee.org/document/5978496

4. **Responsive Web Design Techniques**, Waseem I. Bader, Abdelaziz I. Hammouri, nternational Journal of Computer Applications (0975 – 8887) Volume 150 – No.2, September 2016
https://www.ijcaonline.org/archives/volume150/number2/bader-2016-ijca-911463.pdf

5. **"Performance Optimization in Web Applications"** by Prathyusha Kosuru, published in the *European Journal of Advances in Engineering and Technology* in 2019
https://zenodo.org/records/13919484

6. "**Exploring the Role of Progressive Web Applications in Modern Web Development**" by Viskhan Magomadov, published in the *Journal of Physics:*

Conference Series (2020)
https://www.researchgate.net/publication/347180526_Exploring_the_role_of_progressive_web_applications_in_modern_web_development

7. Ara, J., Sik-Lanyi, C. & Kelemen, A**. Accessibility engineering in web evaluation process: a systematic literature review**. *Univ Access Inf Soc* **23**, 653–686 (2024). https://link.springer.com/article/10.1007/s10209-023-00967-2

8. "**Artificial Intelligence in Web Development: Enhancing Automation, Personalization, and Decision-Making**" by Nitesh Upadhyaya, published in the *International Journal of Advanced Research in Science, Communication and Technology* in August 2024.
https://www.researchgate.net/publication/383170137_Artificial_Intelligence_in_Web_Development_Enhancing_Automation_Personalization_and_Decision-Making

9. "**Microservices Architecture: Accelerating Feature Development And Scalability Through Monolith Decomposition**" by Kuciuk Artiom
https://ijecs.in/index.php/ijecs/article/view/4958

10. Vayadande, Kuldeep. (2024). **Development of Latest Technologies in Web Development: A Survey of Methods and Trends**.
https://www.researchgate.net/publication/378143924_Development_of_Latest_Technologies_in_Web_Development_A_Survey_of_Methods_and_Trends

APPENDIX-A

PSUEDOCODE

A.1. Website Replication Task

Header.html:

BEGIN HTML Document

SET language to English

SET character encoding to UTF-8

SET viewport for responsive layout

SET document title to "Document"

INCLUDE external resources:

- Font Awesome stylesheet for icons
- Local stylesheet "style.css"

BEGIN BODY

DEFINE HEADER SECTION:

DISPLAY contact information:

- "Palam, New Delhi, Delhi 110077"
- Divider
- "info@uptoskills.com"
- Divider
- "+91-9319772294"

ADD login buttons:

- Button labeled "Employer Login" (id="btn1")
- Button labeled "Candidate Login" with user icon (id="btn2")

DEFINE social media icon links:

- Twitter → link to Uptoskills Twitter page
- Facebook → link to Uptoskills Facebook page
- LinkedIn → link to Uptoskills LinkedIn posts
- YouTube → link to Uptoskills YouTube channel
- Instagram → link to Uptoskills Instagram
- Telegram → link to Uptoskills Telegram

END HEADER

END BODY

END HTML Document

Footer.html:

BEGIN HTML Document

SET language to English

SET character encoding to UTF-8

SET viewport for responsive design

SET title to "Document"

INCLUDE external stylesheets:

- Font Awesome for icons
- Local CSS file "style.css"

BEGIN BODY

DEFINE FOOTER SECTION

CREATE footer container with multiple columns:

COLUMN 1: Contact & Social Links

- Display logo image
- Display address: "Palam, New Delhi, Delhi 110077"

- Display email: "info@uptoskills.com"
- Display phone number: "+91-9319772294"
- DISPLAY social media icons with links:
  * Twitter
  * Facebook
  * LinkedIn
  * YouTube
  * Instagram
  * Telegram

COLUMN 2: Quick Links

- Heading: "Quick Links"
- List of navigational links:
  * About Us
  * Courses
  * Events
  * Jobs and Internships
  * Become a Trainer

COLUMN 3: Short Links

- Heading: "Short Links"
- List of important site policy links:
  * Terms & Conditions
  * Privacy Policy
  * Refund & Cancellation
  * Sitemap
  * Login to Download Certificate

COLUMN 4: Location

- Heading: "Find Us Here"
- (Placeholder for future map or address widget)

DISPLAY footer copyright:

- Text: "Copyright © 2025 UptoSkills. All Rights Reserved"

END BODY

END HTML Document

Home.html:

BEGIN HTML Document

SET metadata and import external styles/scripts:

- Set character encoding to UTF-8
- Set viewport for responsiveness
- Set document title to "Document"
- Link to local CSS files: home.css and style.css
- Link to Bootstrap CDN for styling
- Link to Animate.css for animation support
- Link to Font Awesome for icons
- Include jQuery library for dynamic loading

DEFINE jQuery script:

- On document ready:

  LOAD "header.html" into element with id "header"

  LOAD "footer.html" into element with id "footer"

BEGIN BODY

ADD header section (loaded dynamically into "header" div)

CREATE navigation bar (home_nav):

- Display logo image
- Add navigation buttons:
  * Internship (link placeholder)
  * Jobs (link to job.html)

* Learn & Earn (link placeholder)
* Discover (link placeholder)

CREATE main container:

- ADD heading section with animations:
  * Subtitle: "SKILLS FOR LIFETIME"
  * Main title: "Unleash Your Career" (with color emphasis)
  * Description: Platform tagline
- ADD action buttons with animation:
  * "Hire From Us"
  * "HR TPO Podcast"
  * YouTube icon linking to a channel

- ADD promotional message:
  * Info about "LEARN & EARN" program and referral rewards

- CREATE info container with 4 content blocks:
  1. For Colleges
     - Icon
     - Title
     - Description
     - Link to more details
  2. Free Jobs
     - Icon
     - Title
     - Description
     - Link to more details
  3. Free Internships
     - Icon
     - Title
     - Description
     - Link to more details
  4. Degree Programs

- Icon
- Title
- Description
- Link to more details

ADD footer section (loaded dynamically into "footer" div)

END BODY

END HTML Document

Job.html:

BEGIN

```
// Get references to the dropdown selectors
SET qualificationSelect TO element with id "qualification-select"
SET subQualificationSelect TO element with id "sub-qualification-select"

// Listen for changes on qualification selector
ON qualificationSelect change DO
  SET selectedQualification TO selected value from qualificationSelect

  // Hide all job listings initially
  FOR EACH jobElement IN elements with class "job-details"
    SET jobElement display to "none"
  END FOR

  // Show jobs that match selected qualification
  FOR EACH jobElement IN elements with class "job-details"
    IF jobElement's data-qualification attribute EQUALS selectedQualification THEN
      SET jobElement display to "block"
    END IF
  END FOR
```

```
  // Show/hide sub-qualification dropdown
  IF selectedQualification EXISTS in subQualifications THEN
    SHOW subQualificationSelect
    CLEAR subQualificationSelect options
    FOR EACH subQ IN subQualifications[selectedQualification]
      ADD subQ as option to subQualificationSelect
    END FOR
  ELSE
    HIDE subQualificationSelect
  END IF
 END ON

 // Optionally: Listen for changes on sub-qualification selector
 ON subQualificationSelect change DO
  SET selectedSubQualification TO selected value from subQualificationSelect

  FOR EACH jobElement IN elements with class "job-details"
    IF jobElement includes selectedSubQualification THEN
      SET jobElement display to "block"
    ELSE
      SET jobElement display to "none"
    END IF
  END FOR
 END ON
END
```

A.2. Back-End Form Handling and Email Functionality

Main.jsx:

```
BEGIN

 IMPORT StrictMode from React
 IMPORT createRoot from React DOM
```

IMPORT global styles from 'index.css'
IMPORT main App component from 'App.jsx'

FIND the root HTML element by ID "root"

CREATE a React root using createRoot(root element)

RENDER the App component inside StrictMode wrapper
into the root element

END

App.jsx:

BEGIN App Component

IMPORT stylesheet "App.css"
IMPORT image "teamLogo" from images folder

RETURN:
HTML structure:
BODY:
DIV with class "container":
DISPLAY image using teamLogo with class "pic"

DISPLAY heading "DETAILS OF CANDIDATE" in bold

CREATE a form with id "candidateForm":

LABEL and TEXT INPUT for "Name"
- placeholder: "Enter full name"
- required

LABEL and TEXT INPUT for "Role"

- placeholder: "Enter job role"
- required

LABEL and NUMBER INPUT for "Salary"
- placeholder: "Enter salary amount"
- required

LABEL and DATE INPUT for "Joining Date"
- required

LABEL and EMAIL INPUT for "Email"
- placeholder: "Enter email address"
- required

LABEL and TEL INPUT for "Phone"
- placeholder: "Enter contact number"
- required

SUBMIT BUTTON with text "🚀 Submit"

END RETURN

END App Component

Index.js:

BEGIN Express.js Application

IMPORT express
IMPORT body-parser
IMPORT nodemailer

CREATE express app

CONFIGURE app to use body-parser for parsing form data

CREATE POST route for "/submit-form":

1. Receive form data:
   - Retrieve form data (e.g., name, role, salary, joining date, email, phone) from the request body.

2. Prepare the email content:
   - Create email content using the received form data.
   - Structure the email body (e.g., name, role, salary, joining date, etc.).

3. Configure Nodemailer:
   - Set up the transporter with email service (e.g., Gmail) and authentication details.
   - Define email options (e.g., from, to, subject, text, HTML content).

4. Send the email:
   - Use the transporter.sendMail() function to send the email to the recipient's email address.
   - Handle success or error callbacks.
   - If successful: Send a success response to the client (e.g., "Form submitted successfully").
   - If error: Send an error response to the client (e.g., "Error sending email").

CREATE GET route for "/" (to serve the HTML form if needed).

START the server and listen on a specific port (e.g., 3000).

END Express.js Application

A.3. Blog Website Project (EJS + Express.js)

Index.ejs:

BEGIN Blog Page

1. Set up the HTML structure:

- Set the document's language to English (`<html lang="en">`).
- Define the character encoding as UTF-8 (`<meta charset="UTF-8">`).
- Set the viewport to make the page mobile responsive (`<meta name="viewport" content="width=device-width, initial-scale=1.0">`).
- Set the title of the page to "Blog" (`<title>Blog</title>`).

2. Link to an external CSS file for styling the page (`<link rel="stylesheet" href="/styles/main.css">`).

3. Define the body content:

- Create a container div (`<div class="container">`) to hold all elements.

- Display the blog's title as "Rishi's Blog" using an `<h1>` tag.

- Provide a link to create a new blog post:
  - The link's ID is `newPostBtn`, and it points to "/new" to create a new post (`<a id="newPostBtn" href="/new">New Post</a>`).

- List all blog posts within an unordered list (`<ul id="postsList">`):
  - For each post, do the following:
    - Display the title inside an `<h2>` tag.
    - Display the post's date inside a `<small>` tag.
    - Display the content of the post inside a `<p>` tag.
    - Display the author's name using a small text tag.
    - Provide two links for each post:
      - An "Edit" link that directs to the edit page for the post.
      - A "Delete" link that triggers a delete action for the post via the API.

- Close the list and the container div.

END Blog Page

modify.ejs:

BEGIN Edit Blog Page

1. Set up the HTML structure:

- Set the document's language to English (`<html lang="en">`).
- Define the character encoding as UTF-8 (`<meta charset="UTF-8">`).
- Set the viewport to make the page mobile responsive (`<meta name="viewport" content="width=device-width, initial-scale=1.0">`).
- Set the title of the page to "Edit Blog" (`<title>Edit Blog</title>`).

2. Link to an external CSS file for styling the page (`<link rel="stylesheet" href="/styles/main.css">`).

3. Define the body content:

- Create a container div (`<div class="container">`) to hold all elements.

- Display a heading with the value of `heading` (which is dynamically passed from the server, e.g., "Edit Post" or "Create Post").

- Check if the `locals.post` object exists (i.e., if we are editing an existing post):
  - If true:
    - Create an edit form (`<form id="editForm">`) that submits via POST method to `/api/posts/<%= post.id %>`.
    - Pre-fill the form fields with the post's existing data (`title`, `content`, and `author`).
    - Provide a submit button that dynamically displays the value of `submit` (e.g., "Save

Changes").

- If false:

- Create a new post form (`<form id="newPostForm">`) that submits via POST method to `/api/posts`.

- Include input fields for `title`, `content`, and `author` with placeholders.

- Provide a submit button that dynamically displays the value of `submit` (e.g., "Create Post").

4. Close the container div and the body content.

END Edit Blog Page

Index.js:

BEGIN Express.js Server

1. Initialize the Express application:
   - Create an Express app instance.
   - Define the server port as 4000.
   - Set up in-memory data store `posts` to store blog post data.
   - Initialize `lastId` to track the most recent post ID.

2. Set up middleware:
   - Use `bodyParser.json()` to parse incoming JSON data.
   - Use `bodyParser.urlencoded()` to parse incoming URL-encoded data.

3. Define the API routes:

   - GET /posts (Get all posts):
     - Send a JSON response with all the posts from the `posts` array.

   - GET /posts/:id (Get a specific post by ID):
     - Parse the post ID from the request parameters.
     - Search the `posts` array for the post with the given ID.

- If found, send the post data as JSON. If not found, send a 404 response.

- POST /posts (Create a new post):
  - Increment `lastId` to generate a new ID for the post.
  - Collect the title, content, author, and current date from the request body.
  - Create a new post object and add it to the `posts` array.
  - Send the new post as JSON with a 200 status.

- PATCH /posts/:id (Update a specific post's details):
  - Parse the post ID from the request parameters.
  - Find the post with the given ID in the `posts` array.
  - If the post exists, update the fields provided in the request body (e.g., title, content, author).
  - Send the updated post as JSON.

- DELETE /posts/:id (Delete a specific post by ID):
  - Parse the post ID from the request parameters.
  - Search the `posts` array for the post with the given ID.
  - If found, remove the post from the array and return a 200 status.
  - If not found, return a 404 error message.

4. Start the server on the specified port (4000).
5. Log a message when the server is successfully running.

END Express.js Server

Server.js:

BEGIN Express.js Server

1. Initialize the Express application:
   - Create an Express app instance.
   - Define the server port as 3000.
   - Set the `API_URL` as "http://localhost:4000" to connect to the API server.

- Serve static files from the "public" directory.

2. Set up middleware:
   - Use `bodyParser.urlencoded()` to parse incoming URL-encoded data.
   - Use `bodyParser.json()` to parse incoming JSON data.

3. Define the routes:
   - GET / (Home page):
     - Send a GET request to fetch all blog posts from the API server.
     - Render the "index.ejs" template with the list of posts.
     - If an error occurs while fetching posts, send a 500 error response.

   - GET /new (Create a new post page):
     - Render the "modify.ejs" template with a heading ("New Post") and submit button ("Create Post").

   - GET /edit/:id (Edit a specific post page):
     - Fetch the post data from the API server using the provided ID.
     - Render the "modify.ejs" template with a heading ("Edit Post"), submit button ("Update Post"), and the post data.
     - If an error occurs while fetching the post, send a 500 error response.

   - POST /api/posts (Create a new post):
     - Send the form data as a POST request to the API server to create a new post.
     - Redirect to the home page upon successful creation of the post.
     - If an error occurs while creating the post, send a 500 error response.

   - POST /api/posts/:id (Partially update a post):
     - Send the form data as a PATCH request to the API server to update the post with the provided ID.
     - Redirect to the home page upon successful update of the post.
     - If an error occurs while updating the post, send a 500 error response.

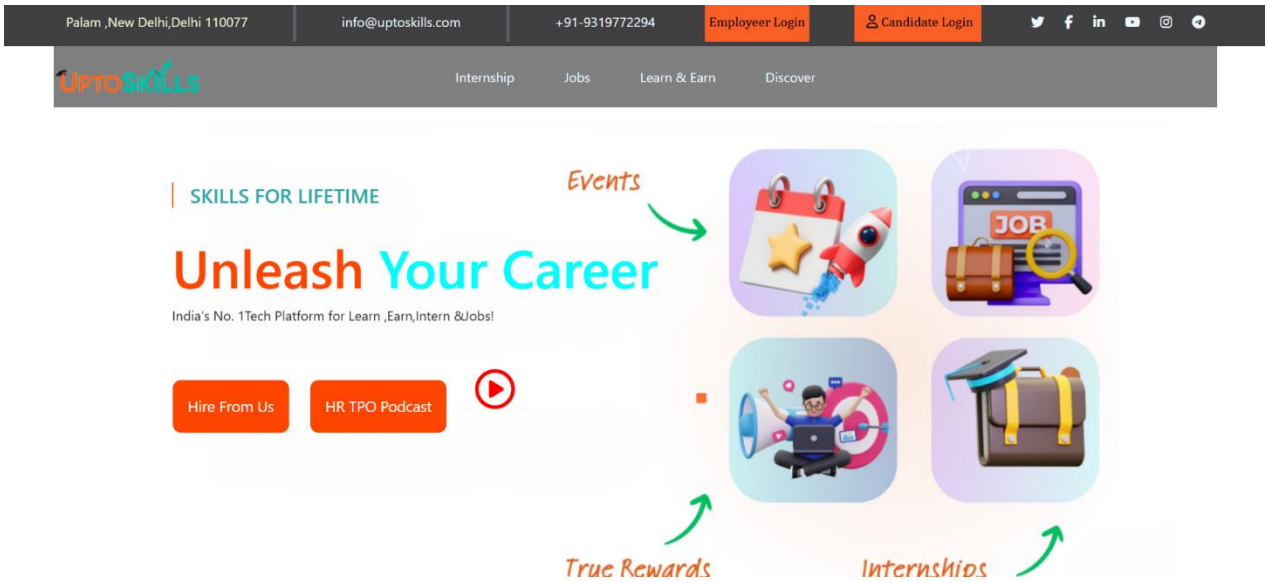
- GET /api/posts/delete/:id (Delete a specific post):
  - Send a DELETE request to the API server to delete the post with the provided ID.
  - Redirect to the home page upon successful deletion of the post.
  - If an error occurs while deleting the post, send a 500 error response.

4. Start the server on the specified port (3000).
5. Log a message when the server is successfully running.

END Express.js Server

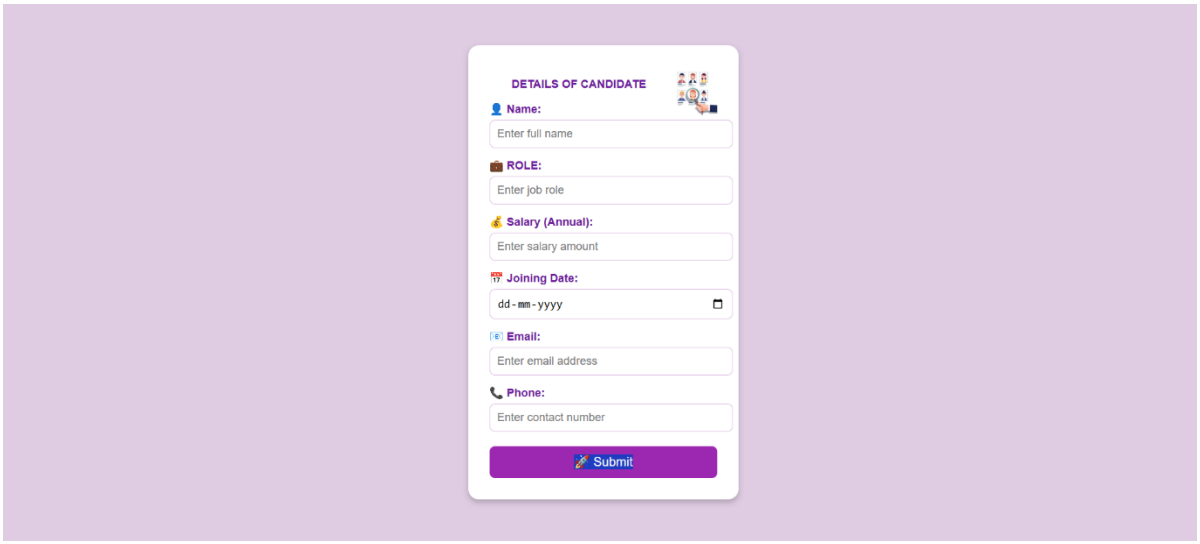
APPENDIX-B

SCREENSHOTS



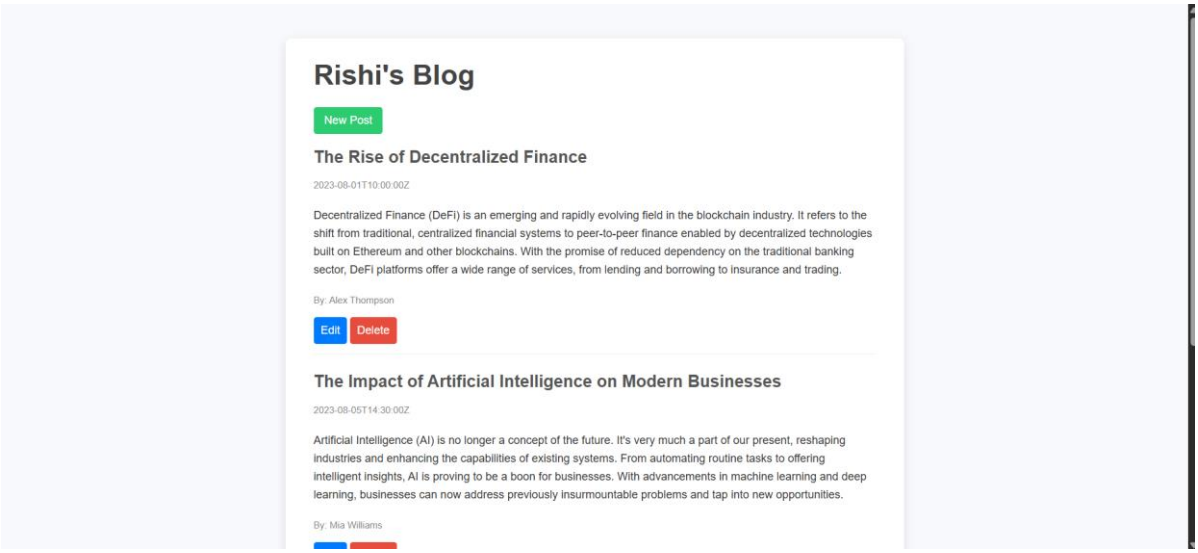
Screenshot B.1: Website Replication Task

Replicated the landing page of Uptoskills using HTML, CSS, and JavaScript, maintaining visual fidelity, responsive design, and interactive UI elements as part of the internship's website replication task.



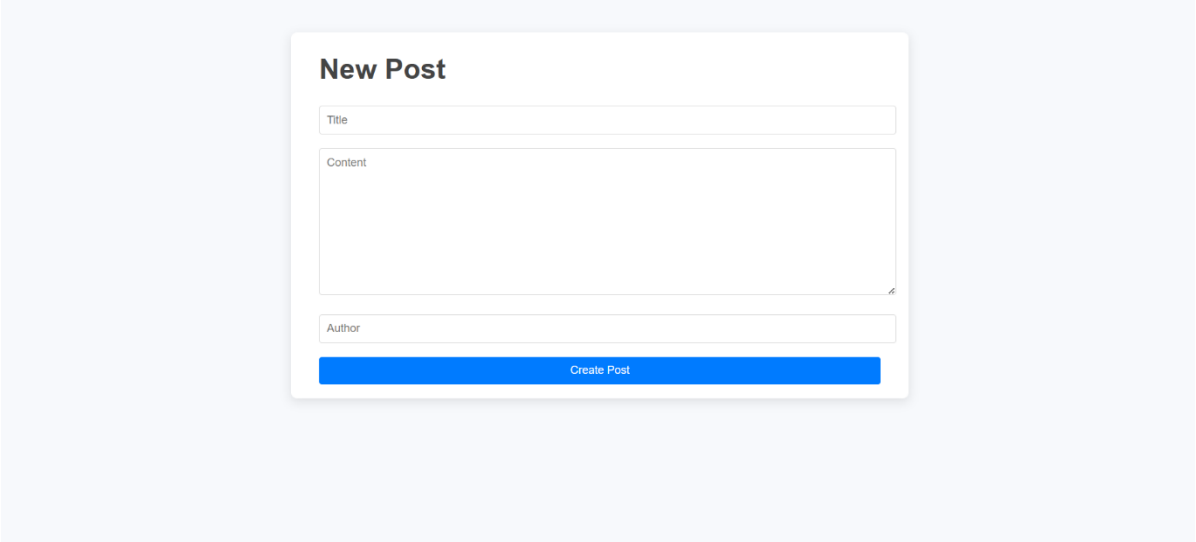
Screenshot B.2: Back-End Form Handling and Email Functionality

This image shows a candidate detail submission form developed as part of the second internship task. Upon submission, the form data is handled by a backend built using Express.js, which validates the input and sends the details to the entered email address via Nodemailer, demonstrating seamless frontend-backend integration.



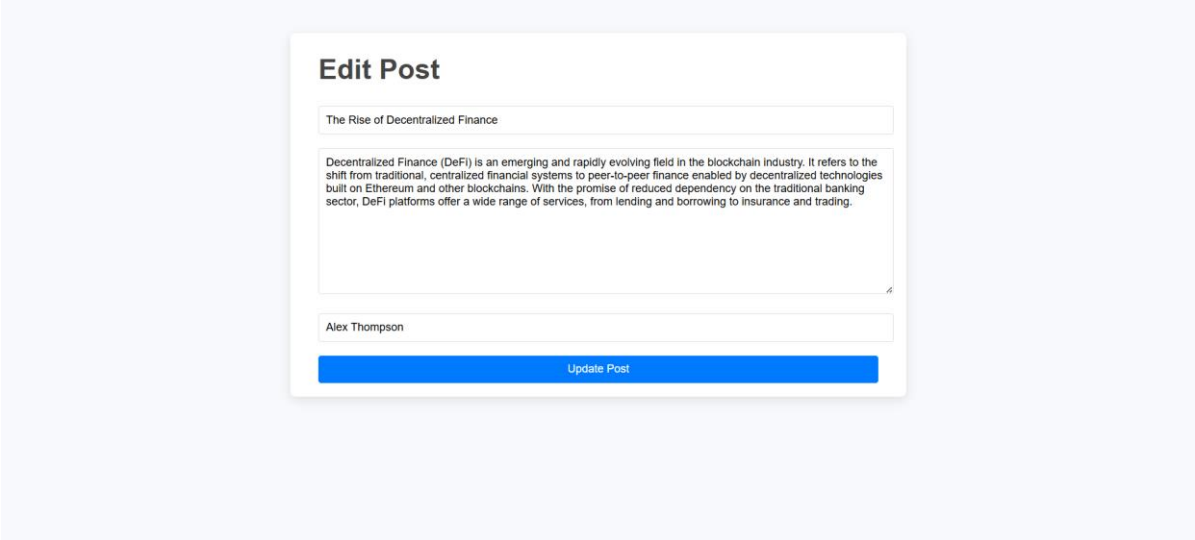
Screenshot B.3: Blog Website Project(EJS + Express.js)

This image represents the blog website developed as part of the internship project using EJS and Express.js. The platform allows users to create, view, edit, and delete blog posts dynamically, showcasing full-stack development skills with server-side rendering, routing, and CRUD functionality.



Screenshot B.4: Create feature in Blog project

This image displays the "New Post" interface of the blog website developed using EJS and Express.js. It allows users to enter a blog title, content, and author name, and upon submission, dynamically creates and stores a new blog post, demonstrating effective implementation of form handling and server-side rendering.

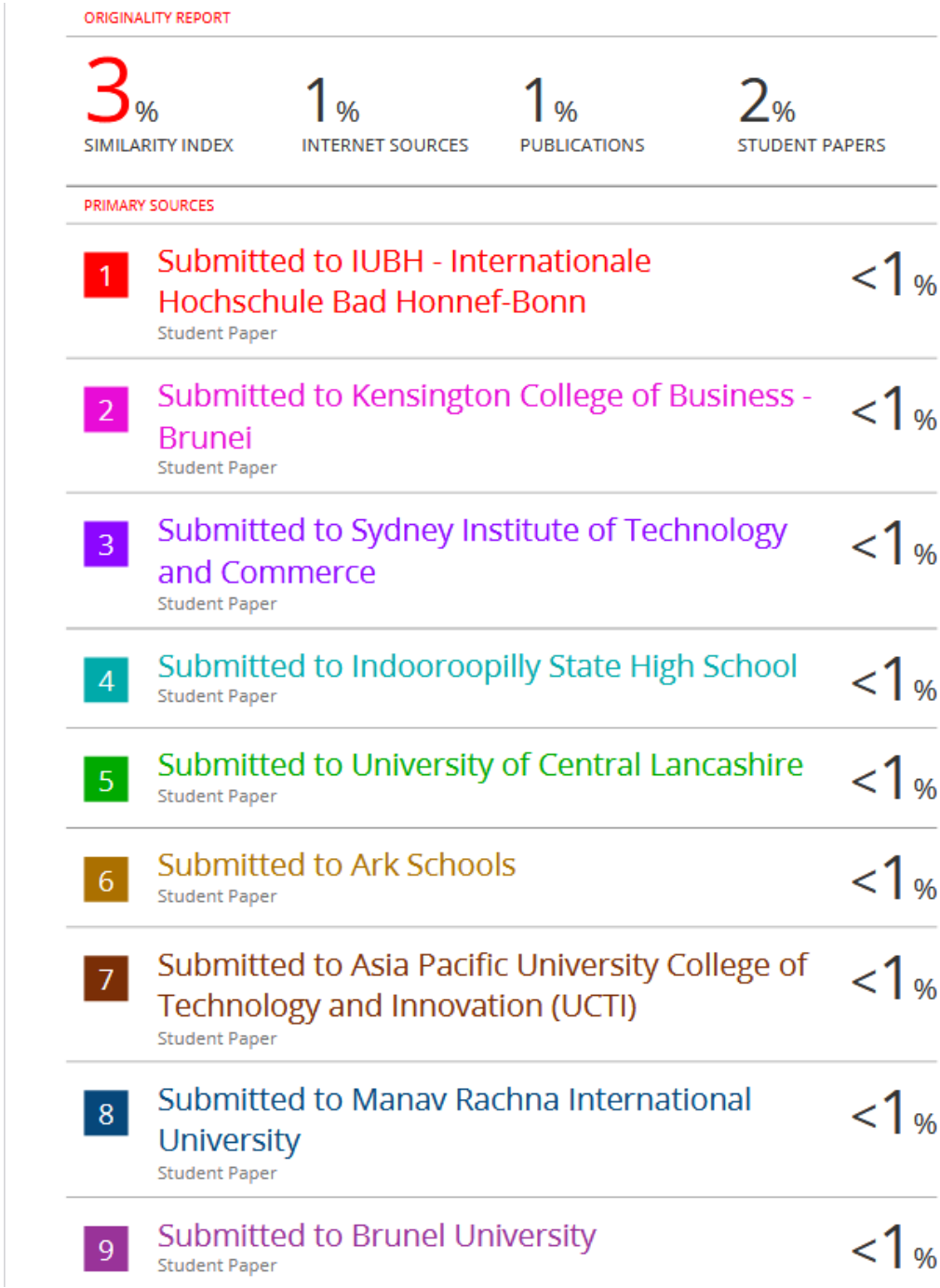


Screenshot B.5: Update feature of Blog project

This image showcases the "Edit Post" functionality in the blog project, enabling users to modify the title, content, or author of an existing blog post. It highlights the implementation of dynamic routing and update operations using Express.js, supporting full CRUD capabilities in the web application.

APPENDIX-C

ENCLOSURES



ORIGINALITY REPORT

3%	1%	1%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Student Paper	<1%
2	Submitted to Kensington College of Business - Brunei Student Paper	<1%
3	Submitted to Sydney Institute of Technology and Commerce Student Paper	<1%
4	Submitted to Indooroopilly State High School Student Paper	<1%
5	Submitted to University of Central Lancashire Student Paper	<1%
6	Submitted to Ark Schools Student Paper	<1%
7	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1%
8	Submitted to Manav Rachna International University Student Paper	<1%
9	Submitted to Brunel University Student Paper	<1%

10	blog.joshsoftware.com Internet Source	<1 %
11	www.ijsr.net Internet Source	<1 %
12	Tulsi Pawan Fowdur, Lavesh Babooram. "Chapter 6 Video Quality Assessment Application Development with JavaScript", Springer Science and Business Media LLC, 2024 Publication	<1 %
13	jets.innovascience.uz Internet Source	<1 %
14	www.ebizneeds.com Internet Source	<1 %
15	www.reformedvoice.com Internet Source	<1 %
16	blog.informationgeometry.org Internet Source	<1 %
17	nlistsp.inflibnet.ac.in Internet Source	<1 %
18	www.oneplanetnetwork.org Internet Source	<1 %
19	Fernandes, Porfírio Afonso. "Desenvolvimento de um ERP com CI/CD, Autenticação e Auditoria do Sistema", Instituto Politecnico do Porto (Portugal), 2024 Publication	<1 %
	Submitted to University of Wales Swansea	

20	Student Paper	<1%
21	"Computers Helping People with Special Needs", Springer Science and Business Media LLC, 2024 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

SUSTAINABLE DEVELOPMENT GOALS

According to the tasks and learning outcomes of your web development internship, the following Sustainable Development Goals (SDGs) from the image can be reasonably linked to your work:

1. Goal 4 – Quality Education:

   My internship is directly linked to SDG 4, as it encourages skills development, technical education, and learning through practice.

   By creating projects with HTML, CSS, JavaScript, Node.js, EJS, and Express.js, I acquired skills that conform to lifelong learning and quality technical education.

2. Goal 8 – Decent Work and Economic Growth:

   This internship equips me for decent work in the digital economy.

   It improved my employment chances in technology fields, particularly the web development sector, which is pivotal for inclusive and sustainable economic development.

3. Goal 9 – Industry, Innovation, and Infrastructure:

   By developing backend applications, form handling, and dynamic websites, I

contributed to digital infrastructure.

My work facilitates innovation in the technology area by creating scalable and organized web solutions, setting a foundation for more complex systems.

4. Goal 17 – Partnerships for the Goals:

   Being involved in an internship and working with supervisors or teams (if applicable) assists in building partnerships for development, particularly in knowledge sharing and mentoring in the tech community.