0

Preface

Chris Hundhausen

The ascension of the Internet as an essential medium of commerce, social interaction and computation has ushered in the modern era of *computational connectivity*. Through a broad range of computing devices—including, most prominently, *mobile* devices—humans are increasingly accessing software applications that support all manner of human activity, ranging from work, to play, to social interaction. And when they do, the applications they are using are most often running in a web browser.

Accordingly, *web development* has become one of the most relevant and highly sought after skills in the software industry today. Given the rapidly growing demand, software companies are struggling find enough developers to fill web development positions, which require at least some of the following skills:

1. *Front-end development*, which focuses on developing the client-side user interfaces to web applications.
2. *Back-end development*, which focuses on developing the web application programming interfaces (APIs) through which front-end applications persistently store, access, and manipulate data.
3. *User interface design* (also called UX or "user experience" design), which focuses on making web applications that are easy to learn, easy to use, and address user needs.

4. *DevOps engineering*, which focuses on implementing the pipelines and scripts for deploying and maintaining, securing, and fine-tuning the performance of web applications.

While *full-stack web development* encompasses all these skills, the first two—front- and back-end development—figure most prominently in the construction of functional web applications. While this book focuses most intently on those skills, it also emphasizes a core aspect of the third skill: *web accessibility*. Indeed, given the U.S. legal requirement [1], and the ethical imperative, for web applications to be accessible by people who use assistive technologies, facilitating accessibility is foundational to web development. DevOps, the fourth of the above quartet of skills, is the least emphasized in this book, receiving some coverage in two of the book's later chapters.

Audience

This book is written for intermediate to advanced undergraduate students who have taken at least an introductory sequence in software development fundamentals. As such, the book assumes an audience that is both comfortable with structured programming in at least one language, and that is capable of rapidly migrating to the JavaScript language even if they have not studied it previously. While the book does discuss the most important syntactic features of the JavaScript language, the book's focus is decidedly *not* on teaching JavaScript, but rather on *applying* JavaScript, along with its siblings, HTML and CSS, to the practice of web development. Students who find themselves struggling with JavaScript as they work through this book should avail themselves of the numerous online tutorials and references on the language. Great

starting places include the W3 Schools JavaScript Tutorial [2], The Modern JavaScript Tutorial [3], and the Mozilla Developer Network's comprehensive reference materials [4].

Technologies

When I began this textbook in 2019, I was under the illusion that it might be possible to write a web development textbook focused on the latest technology versions. I was quickly disabused of this notion when, having drafted initial versions of several of the book's chapters, I discovered that the React framework was changing course: class components were being abandoned in favor of functional components [5]. No problem, I reasoned: I can just rewrite that part of the book. Some five years later, three book drafts later, and multiple versions of the web technologies used in this book later—not to mention the rise of generative AI programming assistants!—I have thankfully given up on keeping up with the latest technologies. The book is grounded in *principles* and *practices* of web development that should withstand the test of time, even if the versions of the technologies used to teach web development in the book will not. With that caveat in mind, let's take a look at the modern (for now) web technologies used in this book.

JavaScript

As the language of the web, JavaScript has become the most popular programming language in use today, with some 62 percent of developers using it [6]. With a storied history that dates to its initial release in 1995, the language continues to evolve, with major updates occurring in 2009 and 2015 and new versions being released annually starting in 2016. As of this writing, the latest release is ES15, also known as

ECMAScript 2024 [7]. Since the release of ES6 JavaScript in 2015—the start of JavaScript's modern era—the core of the language has remained relatively stable. This book adopts post-ES6 JavaScript, including `let` statements, constants, arrow functions, spread syntax, classes, modules (with `import` syntax), and `async/await` syntax for asynchronous operations, while steering clear of esoteric or experimental language features.

On the topic of JavaScript, I would be remiss not to acknowledge the rapid rise of TypeScript [8] as a modern superset of JavaScript widely beloved for its type system and enhanced tooling. If I had begun work on this textbook just a year or two later, I may well have embraced TypeScript as the language of the book. Despite TypeScript's growing popularity, modern JavaScript remains the dominant language for web development. Moreover, students who learn web development in modern JavaScript should have a solid foundation for transitioning to TypeScript if and when the situation calls for it.

The MERN Stack

According to the most recent Stack Overflow Developer Survey available at the time of this writing [6], the two most popular web development frameworks and technologies are Node.js and React.js, which are used by some 40% of software developers. The fifth most popular web technology is Express.js; 18% of developers use it. The fifth most popular database platform is MongoDB, with 25% of developers using it.

These four popular web development technologies form the MERN (MongoDB, Express.js, React.js, and Node.js) technology stack for web development. It is arguably the most popular web development stack in use at the time of this writing, making it a logical choice for a textbook on full stack web development. There is also a good pedagogical

reason to adopt the MERN stack to teach and learn web development: Unlike some other tech stacks that use different languages for front-end and back-end development, the MERN stack uses one language—JavaScript—for all web development. This means that students do not need to switch between programming languages when moving from front-end to back-end development, reducing cognitive load and facilitating a sharper focus on the web development tasks at hand.

Git and GitHub

Modern web development is most often done in a team environment with a shared code base. Accordingly, this textbook relies extensively on a version control system (Git) and a cloud repository hosting system (GitHub) to teach web development. The book assumes that students have basic Git and GitHub skills, including the ability to create and clone a repository; create and switch branches; commit code; and push and pull from a remote. If students do not already have these skills, course instructors should consider adding a supplementary unit on Git/GitHub at the start of the course, or referring students to one of the many wonderful Git/GitHub tutorials on the internet (see, e.g., [9], [10]).

Courses

This book is suitable for a variety of intermediate to advanced web development courses in undergraduate computer science, software engineering, information science, information technology, and related degree programs (henceforth referred to collectively as "computing degree programs"). When considering whether this course is suitable for adoption in a web development course, the key question to ask is this: *Does the course require, as a prerequisite, an introductory sequence in software development*

fundamentals? Adopting the Association for Computing Machinery's terminology, computing educators often call the courses in this sequence *CS 1* ("Introduction to Programming"), *CS 2* ("Intermediate Programming"), and *CS 3* ("Data Structures and Algorithms") If this course sequence or a similar one is a prerequisite, then some pathway through this book should be a good fit for the course.

Pathways through the Book

Table 1 presents pathways through this book for four common types of web development courses offered in computing degree programs:

- **Front End (FE)**—A 200 or 300-level course focused on front-end web development.
- **Full Stack (FS)**—A 300 or 400-level course focused on full-stack web development.
- **Front End–Advanced Elective (FE-AE)**—An advanced elective (400-level) course focused on front-end development.
- **Back End–Advanced Elective (BE–AE)**—An advanced elective (400-level) course focused on back-end development.

	Web Development Course			
Book Section	**FE**	**FS**	**FE-AE**	**BE–AE**
Chapter 1. Welcome to Full Stack Web Development	✓	✓	✓	✓
Part I: Front End Development in HTML, CSS, and JavaScript Chapters 2–11	●	◓	●	
Part II: Front End Development in React Chapters 12-16	◓	◓	●	
Part III: Back-End Development with Node, Express, and MongoDB Chapters 17-24		◓		●
Team Web Development Project (Appendix A and B)	✓	✓	✓	✓

Table 1. Pathways through the book for four different web development courses: FE = 200 or 300-level Front-End Course, FS = 300 or 400-level Full Stack Course, FE-AE = Advanced elective (400-level) course on front-end development, usually part of a two-course sequence;, BE-AE = Advanced elective (400-level) course covering back-end development, usually part of a two-course sequence. ● = Cover chapters in full, ◓ = Cover chapters selectively based on course constraints

FE: 200- or 300-level Course on Front-End Development

Some computing degree programs offer a web development course at the 200- or 300-level, emphasizing web development as a core programming skill. Early web development courses tend to focus on using HTML, CSS, and JavaScript to build front-end web applications, although some coverage of front-end JavaScript programming frameworks and back-end programming may be present. For instance, at my current institution (Oregon State University), CS 290 ("Web Development") is a required course for several degree pathways and focuses most heavily on front-end development. An early course emphasizing front-end development could start with the general introduction to full stack web development in Chapter 1, and then move through the 10 chapters in Part I of the book ("Front End Development in HTML, CSS, and JavaScript") at the pace of one to two chapters per week. If the course wanted to provide coverage of a front-end JavaScript framework, it could also work through some or all of the five chapters in Part II of the book ("Front-End Development in React"). An appropriate culmination to such a course would be an individual or team web development project, as described in Appendix B.

300- or 400-level Course on Full-Stack Web Development

In lieu of, or in addition to, an early course in web development, many computing degree programs offer one or more *advanced electives* in web development for students in the third or fourth year of the program. One approach is a single advanced (300- or 400-level) course on full-stack web development. When I was on the faculty at Washington State University, I taught CS 489 ("Web Development"), which covers front-end and back-end development in a semester-long course. This was the course for

which I wrote early drafts of this textbook, and for which this textbook is ideally suited. In the first week, the course could begin with the general introduction to full stack web development in Chapter 1. In the remainder of the semester, the course could divide its time roughly equally between front- and back-end development before launching a team web development project. A plausible schedule might look like this:

- **Week 1**: Introduction to full-stack web development (Chapter 1).
- **Weeks 2 through 8**: Cover selected chapters of Part I ("Front End Development in HTML, CSS, and JavaScript") and Part II ("Front-End Development in React"). There are 15 chapters in these two parts, so a pace of one to two chapters per week would be reasonable.
- **Weeks 9 through 13**: Cover selected chapters of Part III ("Back-End Development with Node, Express, and MongoDB"). At a pace of one to two chapters per week, a course could cover all eight chapters in this part within five weeks.
- **Weeks 14 through 15**: Students work on final team web development project using agile development process with two one-week sprints. Appendix A presents a primer on team agile practices for web development, while Appendix B presents detailed instructions for running and evaluating a web development project associated with the SpeedScore code base.

Advanced Electives on Front-End and Back-End Web Development

Some computing degree programs offer two advanced electives (usually 400-level courses) on web development: one focused on front-end development, and a second focused on back-end development. For example, at my current institution, Oregon State

University, we offer CS 493 (“Cloud Application Development”) and CS 494 (“Advanced Web Development”) as 10-week courses (we are on the quarter system). While some institutions may require courses like these to be taken in a certain order—typically, the front-end course comes before the back-end course—the courses at Oregon State University can be taken in any order.

This book has a unique pedagogical feature that supports independent front-end and back-end web development courses like the ones at Oregon State University. Part III of the book, which focuses on back-end development, is quasi-independent from Parts I and II, which focus on front-end development. While the entire book focuses on developing the same full-stack web application, the book’s coverage of front-end development relies exclusively on a front-end mechanism—*local storage*—to store application data persistently. Thus, Chapters 2 through 16 never refer to or depend on a back-end web application. This means that, after an introduction to full stack web development and a brief orientation to the web application that serves as the unifying focus of the book (Chapter 1), students could begin their study of back-end web development with Part III (Chapters 17–24). In fact, Chapter 17’s treatment of computer networking concepts for the web, including HTTP and TCP/IP, provides an appropriate introduction to server-side programming, laying a foundation for a course focused on back-end web development.

Pedagogical Perspectives

This textbook is entitled *Full Stack Web Programming from the Ground Up* for a reason: It starts with the lowest-level building blocks of web programming, and then moves to

higher-level frameworks. For front-end web programming, this means presenting detailed coverage of the fundamental building blocks of front-end web development—HTML, CSS, and JavaScript—and then shifting to React, a higher-level JavaScript framework. For back-end web programming, this means introducing the Node.js execution environment prior to delving into Express.js, a popular back-end web application framework built on top of Node.js.

Observe that this perspective differs from the one commonly taken by web development bootcamps and online courses, which tend to teach high-level front-end frameworks (e.g., React, Vue, Angular) and back-end frameworks (Express, Flask, Ruby) right out of the gate, in the interest of getting students rapidly up to speed with web development. The pedagogical perspective underlying this book is that, if students learn the low-level building blocks of web development first, they are in a better position to learn the higher-level frameworks on which they are built. The approach dictated by this perspective requires more time and effort, as students first learn lower-level constructs (HTML, CSS, JavaScript, Node) that will ultimately be replaced by higher-level abstractions such as *components* and *JSX* (in front-end React) and *routes*, *controllers*, and *services* (in back-end Express). However, the extra time and effort are deemed to be worth it, since an understanding of the underlying building blocks, and of how higher-level frameworks build on them, should lead to more resilient and robust web development skills.

Another key pedagogical perspective embodied by this book is that a productive and engaging way to learn web programming is to build an *authentic* web application from the ground up. In the first chapter, the book introduces SpeedScore, a full-stack web

application for speedgolf [11]. Under continuous development since 2017, most recently by a team of undergraduate and graduate students at Oregon State University as part of its Vertically Integrated Projects Program [12], the production version of the application (see https://speedscore.org) is hundreds of thousands of lines of source code, has gone through multiple versions, and is used by a vibrant international community of speedgolfers. The hope is that, by engaging in the development of a real full-stack application that is used by an international audience, readers will gain the sense that they are learning relevant skills that can make a difference.

A third perspective, closely related to the previous one, is that it is imperative for students to learn how to contribute to *legacy* code bases, rather than always writing their own code from scratch. This perspective has been heavily influenced by my involvement in a multi-year computing education research project [13] funded by the U.S. National Science Foundation's Improving Undergraduate STEM Education (IUSE) Program [14]. The project focuses on developing and evaluating novel pedagogical approaches in which students learn software engineering by engaging in so-called *brownfield* software development projects, in which they collaborate on a team to contribute to a legacy code base written by others. This type of development more closely aligns with what students will ultimately encounter in the software industry, where they will join teams that grow and maintain legacy code bases. This book emphasizes brownfield development by requiring students to build on, and test, the SpeedScore code base developed in the book, rather than create new web applications from scratch.

Pedagogical Features

We now consider the key pedagogical features of the book, some of which follow from the pedagogical perspectives described in the previous section.

Code Listings

Perhaps the most prominent pedagogical feature of the book is its numerous code listings, which present key segments of solutions that address the development problems explored in the book's chapters. Indeed, much of the book is dedicated to motivating, developing, and explaining these listings, which ground the book firmly in concrete code solutions that address key web development problems.

SpeedScore Code Repository with Tags

The book develops a scaled down, pedagogical version of the SpeedScore application from start to finish, with each chapter using the application as a backdrop against which to motivate, contextualize and explore a different aspect of web development. In each chapter, the book carefully walks the reader through the development of key components of the application, step-by-step, and reasons through the associated design and implementation considerations and tradeoffs.

To further ground its explorations of web development, the book includes a companion GitHub code repository whose history documents the progressive development of the book's code. The code developed in each book chapter is captured in a separate branch of the repository (e.g., `ch3`, `ch4`, `ch5`). Moreover, strategically-labeled Git tags [15] reference the key code blocks developed within each chapter. Tag markers appearing in the text (e.g., `ch5s1` for the first snapshot of Chapter 5) allow readers to

access the code that was just developed. Thus, the book's source code repository plays a valuable role in supporting exploratory learning, enabling users to view, execute and test the code developed in the book.

Flow Diagrams

In Part III of the book, where server-side APIs are developed to address the needs of a client-side application, understanding the complex interactions between the client and server is essential. Annotated flowcharts that graphically present these interactions feature prominently in this part of the book, providing valuable visual aids for explaining how various API routes work.

Boxes

Many of the chapters feature boxes—side discussions of key technologies, considerations, and tradeoffs related to the chapter content. While boxes are generally not required reading, they add perspective and invite the reader to contemplate the use of practices and technologies that can enhance web development.

Exercises

End-of-chapter exercises provide opportunities for readers to check their understanding of the concepts, solutions, and design and implementation decisions presented in the chapter. They also invite readers to perform additional research related to the chapter content.

Programming Tasks

End-of-chapter programming tasks invite readers to expand upon the code developed in the chapter. This might entail implementing important functionality that the chapter chose

not to implement; writing test suites to evaluate code developed within the chapter; or using alternative technologies, libraries, or approaches to implement functionality implemented within the chapter. In all cases, the tasks are *brownfield* in the sense that they must build on the existing SpeedScore code in the chapter's repository branch.

Quasi-Independence of Front-End and Back-End Parts of the Book

As mentioned earlier, a unique pedagogical feature of the book is that the front-end code developed in Parts I and II of the book does not rely on the back-end server application (API) constructed in Part III of the book. This is because the client SpeedScore application relies exclusively on client-side local storage for persistent data storage. Only in the book's final chapter (Chapter 24) are the front-end and back-end applications stitched together into a full-stack application. The practical implication of this pedagogical feature is that it allows front-end and back-end web development to be taught independently of each other, and in any order.

Team Web Development Project

Given that when students become professional web developers, they will most often join software teams that are working on legacy code bases, it makes sense to engage students in team web development projects as a culminating experience in web development courses that adopt this textbook. To support such projects, Appendix A presents best practices for engaging student teams in agile web development practices, while Appendix B presents a prompt, set of issues, starting codebase and assessment rubrics for team web development project in which students add features to the SpeedScore application developed in this book.

Articulation with ABET Learning Outcomes

Many computing degree programs seek accreditation from ABET [16], perhaps the most widely known and respected accreditation board for STEM disciplines. To obtain ABET accreditation, a degree program must demonstrate that students in the program attain the ABET-mandated student learning outcomes (SLOs). Table 2 presents the five ABET SLOs for computing degree programs [17], along with sample performance indicators that could be used to gauge attainment of those outcomes in a web development course that adopts this book. The book's emphasis of seven core web development principles provides a robust framework for principles-based assessment of SLO 1. An assessment of SLO 2 can be split across client-side and server-side implementation concerns. While the remaining three SLOs may not traditionally be assessed in a web development course, Table 2 illustrates possible avenues for assessment that leverage web development activities, concerns, and a team project involving web development.

ABET Student Learning Outcome	Sample Performance Indicators
1. *Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions.*	a. *Responsiveness*: Tailor a web application's user interface to a variety of different display sizes b. *Accessibility*: Use semantic HTML, the Web Content Accessibility Guidelines to construct we applications that are accessible to people who use access them through assistive technologies c. *Usability*: Apply user interface design principles to construct web applications that are easy to learn and efficient to use. d. *Usefulness*: Construct web applications that support functionalities that provide value to their users e. *Correctness*: Construct test cases to ensure that a web application's behavior meets requirements. f. *Robustness*: Apply input constraints and error checking to prevent unexpected, uncommon or incorrect user behaviors from crashing a web application g. *Security*: Use anti-CSRF tokens and http-only cookies to reduce possibility of successful XSS and CSRF attacks against a web application

2. *Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.*	a. Client-side development: Use React to construct a client-side web application that supports creating, modifying, and deleting user data stored in local storage b. Server-side development: Use Express.js and MongoDB to construct a REST API to support CRUD operations on a web application's user data
3. *Communicate effectively in a variety of professional contexts.*	a. *Documentation:* Clearly document a module or function within a web application's source code using best practices
4. *Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.*	a. *Accessibility*: Assess the extent to which a web application is universally accessible to people who use assistive technologies b. *Security*: Implement appropriate security measures to safeguard web application accounts and data.
5. *Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline*	a. Fulfill assigned responsibilities in a team web development project b. Describe ways a team might improve its collaborative practices c. Assist team members who ask for help

Table 2. Sample performance indicators for gauging student attainment of ABET Student Learning Outcomes in a web development course that adopts this textbook

Articulation with CS2023

The latest ACM/IEEE-CS/AAAI Computer Science Curricula (CS2023) [18] classify web development as a form of *specialized platform development (SPD)*—a subarea of the Application Development knowledge area. Table 3 presents the core and non-core knowledge areas (KAs) that make up the *SPD-Web* subarea, along with the book topics and chapters that address those KAs. As the table indicates, all but one KA (3. Software as a Service) are addressed by the book. Conversely, several broad topics not identified by the CS2023 as SPD-Web KAs are addressed by the book:

- Automated testing of front-end web applications (Ch. 7)
- Using Web APIs (Ch. 14)
- Networking concepts and protocols for web development (Ch. 17)
- Automated testing of APIs (Ch. 23)
- Documenting APIs (Ch. 23)

Knowledge Area (C = Core, N = Non-Core)	Relevant Book Topics/Chapters
1. Web programming languages (C)	• Overview: Ch. 1 • HTML, CSS, JavaScript: Ch. 2–11
2. Web platforms, frameworks, or meta-frameworks (C)	• Client-side development frameworks: Ch. 12–16 • Server-side development frameworks: Ch. 18
3. Software as a Service (SaaS) (C)	• Not covered
4. Web standards (C)	• Accessibility: Ch. 3–11
5. Security and Privacy Considerations (C)	• Client data sanitization: Ch. 8 • Authentication: Ch. 21 • API Security: Ch.22
6. Analyzing requirements for web applications (N)	• Front-end app requirements: Ch. 14 • Back-end app requirements: Ch. 18
7. Computing services (N)	• Hosting solutions for client apps: Ch. 17 • Hosting solutions for server apps: Ch. 24
8. Data management (N)	• Accessing app data: Ch. 18
9. Architecture (N)	• Architecting client-side applications: Ch. 3, 4, 15, 16 • API Architectural Frameworks: Ch. 20
10. Storage solutions (N)	• Local Storage: Ch. 10 • SQL and No-SQL Databases: Ch. 19

Table 3. Book topics and chapters that address the core (C) and non-core (N) CS2023 Knowledge Areas (KAs) for SPD-Web subarea

Acknowledgements

Bringing this book to the finish line has proved to be a far greater challenge than I could have ever imagined. That I was able to finally finish the first edition of the book in 2024, over five years after I drafted the first chapters of the book, is nothing short of astonishing, not only because I was able to persevere through multiple drafts and rewrites as the web technology rapidly evolved, but also because my editors, Naomi Robertson and Steve Merken of Elsevier, failed to give up on me after multiple missed deadlines.

On the over five-year journey that has culminated in the publication of the book, I have many people to be grateful for. This journey all started when Washington State University (WSU) agreed to support my odd request for a sabbatical focused on developing a full-stack web application for the esoteric sport of speedgolf and traveling to national and international speedgolf tournaments to facilitate, and collect data on, the

use of the application to support the live-scoring of those tournaments. That sabbatical, which took place in the academic year 2017-18, kindled my passion for web development and planted the seeds for this book. Early drafts of some chapters of this book were used and reworked in the context of teaching CS 489 ("Web Development") at WSU from 2018 through 2021.

I am grateful to Stephen Merken, production editor at Elsevier, who responded enthusiastically to my initial book proposal and agreed to support it. He assigned the project to Naomi Robertson, whose encouragement and unflinchingly positive attitude throughout this process have had a profoundly positive impact. As the book began to consume more years of my life, I thought many times of giving up, but Naomi kept rooting for me to finish it.

I am grateful to three colleagues who contributed to the writing of this book. Thanks go to Craig Miller, a prominent computing education researcher and advocate for web development education, for writing the book's foreword; to my long-time colleague and friend Phillip Conrad, with whom I have passionately pursued research into software engineering education for over a decade, for contributing the appendix on best practices for team web development projects rooted in agile methods; to Rishabh Srivastava, a former graduate student of mine, for adapting a piece of his M.S. project for the appendix on deploying a client-side web application to alternative hosting platforms; and to XXX, a current/former undergraduate/graduate student of mine, for contributing solutions to the end-of-chapter exercises and programming tasks.

The book's pedagogic approach of engaging students in *brownfield* web development, in which they develop features for the legacy SpeedScore code base rather

than writing web applications from scratch, stems from my work on the National Science Foundation-funded research project entitled “Exploring Brownfield Programming Assignments in Undergraduate Computing Education.” [13]. Thus, this book has been supported by the National Science Foundation under grant number 1915196.

In the countless hours I spent researching modern web development, writing the SpeedScore source code, and writing the book, I became a cave dweller in front of the triple monitors in my home office, sacrificing many weekends and recreational opportunities with my family. Nonetheless, my daughter Lily and my partner Angela remained supportive, allowing me to present show-and-tell sessions about book sections or SpeedScore code I had developed, and listening to my venting about how daunting the project had become. They gave me the space to reflect and the strength to keep going.

My mom, Patricia Hundhausen, and dad, David Hundhausen, were both alive when I floated the idea to write this book. Both were proud and supportive of their son’s new textbook venture and encouraged me to pursue it. Sadly, I lost both of my parents before this book could be completed. I have fond memories of giving my dad weekly updates about this book on my drives back from speedgolf rounds. He would always ask how things were going and urge me to keep at it. His belief in me motivated me not to give up, even when it seemed I would never finish.

References Cited

[1] “Fact Sheet: New Rule on the Accessibility of Web Content and Mobile Apps Provided by State and Local Governments,” ADA.gov. Accessed: Dec. 12, 2024. [Online]. Available: https://www.ada.gov/resources/2024-03-08-web-rule/

[2] “JavaScript Tutorial,” w3schools. Accessed: Feb. 11, 2022. [Online]. Available: https://www.w3schools.com/js/

[3] “The Modern JavaScript Tutorial.” Accessed: Dec. 13, 2024. [Online]. Available: https://javascript.info/
[4] moz://a, “typeof - JavaScript | MDN,” MDN Web Docs. Accessed: Jul. 13, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof
[5] “Introducing Hooks,” React. Accessed: Nov. 12, 2022. [Online]. Available: https://reactjs.org/docs/hooks-intro.html
[6] “2024 Stack Overflow Developer Survey,” Stack Overflow. Accessed: Dec. 13, 2024. [Online]. Available: https://survey.stackoverflow.co/2024/
[7] “ECMAScript® 2024 Language Specification.” Accessed: Dec. 14, 2024. [Online]. Available: https://262.ecma-international.org/
[8] “JavaScript With Syntax For Types.,” typescriptlang.org. Accessed: Feb. 11, 2022. [Online]. Available: https://www.typescriptlang.org/
[9] “Git Tutorial.” Accessed: Dec. 14, 2024. [Online]. Available: https://www.w3schools.com/git/default.asp
[10] “Git and GitHub Tutorial – Version Control for Beginners,” freeCodeCamp.org. Accessed: Dec. 14, 2024. [Online]. Available: https://www.freecodecamp.org/news/git-and-github-for-beginners/
[11] “PlaySpeedgolf.com. It’s game time!,” PlaySpeedgolf.com. It’s game time! Accessed: Dec. 14, 2024. [Online]. Available: https://www.playspeedgolf.com/
[12] “Vertically Integrated Projects | Electrical Engineering and Computer Science | College of Engineering | Oregon State University,” Oregon State University: School of Electrical Engineering and Computer Science. Accessed: Dec. 14, 2024. [Online]. Available: https://engineering.oregonstate.edu/EECS/vip
[13] C. D. Hundhausen and O. Adesope, “Collaborative Research: Exploring Brownfield Programming Assignments in Undergraduate Computing Education,” U.S. National Science Foundation. Accessed: Dec. 15, 2024. [Online]. Available: https://www.nsf.gov/awardsearch/showAward?AWD_ID=1915196
[14] “Improving Undergraduate STEM Education: Directorate for STEM Education (IUSE: EDU) | NSF - National Science Foundation.” Accessed: Dec. 15, 2024. [Online]. Available: https://new.nsf.gov/funding/opportunities/iuse-edu-improving-undergraduate-stem-education-directorate-stem
[15] S. Chacon and B. Straub, “Git Basics - Tagging,” in *Pro Git*, Apress, 2014, pp. 55–60. Accessed: Dec. 15, 2024. [Online]. Available: https://git-scm.com/book/en/v2/Git-Basics-Tagging
[16] “Home,” ABET. Accessed: Dec. 15, 2024. [Online]. Available: https://www.abet.org/
[17] “Criteria for Accrediting Computing Programs, 2023 - 2024,” ABET. Accessed: Dec. 15, 2024. [Online]. Available: https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2023-2024/
[18] A. N. Kumar *et al.*, *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024.