ChatGPT

9-Month Full-Stack Developer Roadmap (Sept 2025 – May 2026)

Comprehensive Skills and Knowledge to Acquire

Programming Foundations

Start by solidifying the basics of coding and computer science. Choose JavaScript/TypeScript as your primary language (for full-stack and front-end work), with Python as your second language and Java as a third for object-oriented foundations. Key concepts include variables, data types, control structures, functions, and error handling. Make sure you understand **object-oriented programming (OOP)** principles (classes, inheritance, polymorphism) and fundamental **design patterns**. Gain familiarity with how memory and the call stack work, and practice writing clean, readable code (following naming conventions and style guides). This foundational mastery will make learning frameworks and advanced topics much easier [1] [2]. Also ensure your development environment is set up (VS Code or IntelliJ, browser dev tools, etc.), and you're comfortable using the command line and package managers (npm, pip, Maven/Gradle for Java).

Data Structures & Algorithms (Daily Practice)

Continuous algorithm practice is crucial for technical interviews. Every day, dedicate time to solving **data structure and algorithm problems** (e.g. on LeetCode, HackerRank). Start with basic data structures (arrays, strings, linked lists, stacks, queues) and gradually cover trees, graphs, heaps, hash tables, and advanced algorithms (dynamic programming, search/sort, greedy, etc.). Learn Big-O complexity analysis to reason about performance. Consistency is key – solving around *300 quality problems* over the next 9 months will significantly sharpen your skills [3]. Focus on covering common problem patterns (two-pointer, BFS/DFS, recursion, sliding window, etc.) rather than just raw quantity [4]. The goal is to deeply **understand** solutions, not memorize them [5]. By May, you should aim for 300+ problems solved, which has been observed as a threshold for strong interview performance [6]. This daily practice will also make you comfortable in writing code in an interview setting and thinking through problems methodically.

Front-End Development (HTML, CSS, JavaScript, React, TypeScript)

The front-end is about building the user interface and experience of web applications [7]. Start with **HTML** for structure and **CSS** for styling: learn how to create semantically structured pages with forms, tables, images, lists, and media, and how to style them with CSS selectors, the box model, layouts (flexbox, grid), and responsive design techniques. Build a few static pages to practice these. Next, focus heavily on **JavaScript**, as it is the language that brings interactivity to the web. Master the fundamentals of JS (ES6+ syntax, DOM manipulation, events, AJAX/fetch, promises and async/await, and browser APIs). Since JS will be your strongest skill, spend significant time building a strong grasp of it – understanding closures, prototypes, the event loop, and how to debug in the browser. Once comfortable with vanilla JS, learn **TypeScript** for type-safe development (many modern projects use TS).

After the basics, move to a modern **frontend framework**. This roadmap uses **React** (a popular library for building UI components). Learn React fundamentals (JSX, component state and props, lifecycle hooks or effects, routing, context, etc.), and then more advanced patterns (state management with something like Redux or React Context, handling forms, optimizing performance). Also get familiar with a CSS library or framework – e.g. **Tailwind CSS** or **Bootstrap** – to speed up styling and learn about component-based design [8]. By mastering HTML/CSS/JS first and then React, you ensure you aren't skipping the core knowledge [2]. Build multiple frontend projects, from small ones like a calculator or to-do list (to practice DOM and basic JS) to larger single-page applications in React (to practice component architecture). Incorporate **TypeScript** in your React projects to solidify your TS skills.

Back-End Development (Node.js, Express, Python, Java)

Back-end development involves building the server-side logic, APIs, and database interactions that power web applications [7]. Start with **Node.js** (so you can use JavaScript on the server) and the **Express** framework for building RESTful APIs. Learn how to set up an Express server, define routes and middleware, handle requests/responses, and connect to databases. Understand how to structure a backend project (with controllers, services, etc.), manage environment variables, and implement authentication (e.g., JWT or OAuth basics). Practice building simple APIs (for example, a CRUD API for a to-do list or a blog).

Next, gain proficiency in **Python** for backend scripting and automation. Learn Python basics (if not already known from CS courses) including its syntax, standard library, and idiomatic style. Use Python for scripting tasks and explore a lightweight web framework like **Flask** or **FastAPI** to create a simple API, or **Django** for a full-featured MVC approach. Python will also be useful for writing scripts (like web scrapers or automation scripts) and for exploring data/AI libraries if you choose.

For **Java**, focus on core Java syntax and OOP, and then learn the basics of building a backend with it. This could be as simple as creating a console application or using **Spring Boot** to set up a basic web service. While you don't need to master enterprise Java in 9 months, having one solid Java project (e.g. a REST API or a small application) will demonstrate your ability to work in strongly-typed, OOP-heavy languages. Java experience will reinforce concepts like design patterns, memory management, and multi-threading which are valuable in many large companies.

By learning back-end in **multiple languages**, you broaden your understanding: Node/Express for JavaScript (common in startups and full-stack roles), Python for quick scripting and popular in automation/ML, and Java for enterprise-level development. You'll be prepared to discuss and use all three. Make sure to understand how to handle **APIs** (both designing your own and consuming external APIs), how to implement server-side rendering vs. REST vs. GraphQL (just basic concepts), and how authentication & authorization are handled on the backend (sessions, tokens, OAuth flows). Build a couple of full-stack projects that tie front-end and back-end together (for example, a React frontend with a Node/Express API and database) to practice end-to-end integration.

Databases (SQL and NoSQL)

Working with data is a core part of back-end development. Learn **SQL** for relational databases and **NoSQL** for alternative data stores. Start with SQL fundamentals: tables, queries (SELECT, INSERT, JOIN, etc.), and normalization. Use a database like **PostgreSQL** or **MySQL** to practice writing queries and integrating with your backend (for instance, use Node.js with PostgreSQL to store user info). Understand how ORMs (Object-

Relational Mappers) work (e.g. using Prisma or Sequelize in Node, SQLAlchemy in Python, Hibernate in Java) versus raw SQL queries.

Also learn about **NoSQL** databases, especially **MongoDB**, which is often used with Node (as part of the popular MERN stack) [9]. Learn how data is stored in JSON-like documents in MongoDB, and practice basic CRUD operations with a MongoDB database (perhaps using it in one of your Node projects). Compare when to use relational vs NoSQL (e.g. Mongo for flexible schemas or distributed data, SQL for structured relational data). Additionally, get exposure to **Redis** (an in-memory data store useful for caching) and basic data modeling for key-value stores. By mastering at least one SQL database and one NoSQL database, you'll be able to design the data layer for your applications confidently.

Cloud Platforms and Deployment (AWS Focus, plus Azure & GCP)

Modern full-stack developers are expected to know how to deploy and run applications on the cloud. Focus deeply on **Amazon Web Services (AWS)**, as it's in high demand. Start with the basics of cloud computing: understand concepts like IaaS vs PaaS, regions/zones, and common services. Learn core AWS services: EC2 (virtual machines), S3 (storage), RDS (managed SQL databases), AWS Lambda (serverless functions), API Gateway, and IAM (permissions). Practice deploying at least one project on AWS – for example, host your backend on an EC2 instance or deploy a container to AWS (using ECS or a simple Elastic Beanstalk app). Learn how to use AWS's console and the AWS CLI, and understand the basics of networking on AWS (VPC, security groups). Earning the **AWS Certified Cloud Practitioner** will validate foundational cloud knowledge [10], and the **AWS Solutions Architect – Associate** will demonstrate ability to design and deploy AWS solutions (it validates skills to build and deploy cloud applications) [11]. These certs will ensure you've covered a broad range of AWS topics (compute, storage, databases, security, architecture best practices).

While AWS is the focus, also get a **working knowledge of Azure and Google Cloud**. You don't need to dive as deep, but do go through an overview of each to understand their offerings and how they compare to AWS (for instance, know that AWS EC2 ≈ Azure VM ≈ GCP Compute Engine). Earning **Microsoft Azure Fundamentals (AZ-900)** is a quick way to learn Azure's basics (cloud concepts, core Azure services, pricing) and get a certificate proving that knowledge. Similarly, explore Google Cloud – possibly aim for the **Google Cloud Digital Leader** or **Associate Cloud Engineer** certification (or at least study the material) to understand GCP's approach to networking, compute, and services. If possible, complete Google's **"IT Automation with Python" Professional Certificate**, which also covers using Python for cloud & automation tasks [12] [13]. This will reinforce your Python skills and teach practical cloud automation (using APIs, managing VMs, etc.). By having familiarity with all three major clouds, you can confidently list multi-cloud knowledge on your resume, with AWS as your strength.

Version Control (Git and GitHub)

Using Git for version control is mandatory in modern development. Learn Git basics: initializing a repo, staging and committing changes, branching and merging, and resolving merge conflicts. Use Git throughout all your project work – commit code daily so that by the end, you have a rich history of contributions on GitHub. Also learn how to use GitHub (or other Git platforms like GitLab) for collaboration: pushing to remotes, creating pull requests, reviewing code, and using issues/project boards for task tracking. In the first few weeks, create a GitHub repository for each project and practice good Git hygiene (meaningful commit messages, organized branching). By stage two of your learning, you should be comfortable enough with Git to collaborate with others or contribute to open source. **Upload all your**

projects to GitHub; recruiters will often check your GitHub profile, so having well-structured code repositories is key [14]. Also consider learning how to write a good README for each project and using GitHub features like GitHub Projects or Actions later on. Version control skills also include understanding .gitignore, rebasing vs merging, and tagging releases. These show professionalism in managing code. (Git will become second nature with daily use.)

Testing and Quality Assurance

Developers are expected to test their code. Learn how to write **unit tests** and perform basic **integration testing** for your projects. In JavaScript, get familiar with testing frameworks like **Jest** or **Mocha/Chai**. In Python, use **unittest or PyTest**, and in Java, **JUnit**. Begin by writing simple test cases for functions (e.g. testing a utility function with various inputs). As your projects grow, write tests for API endpoints (e.g. using Supertest for Node or testing Flask routes). Learn about mocking external services in tests, and how to test frontend components (using something like React Testing Library) for bonus points. You don't need exhaustive coverage on personal projects, but having some tests shows you value code quality.

Also learn to use linters and formatters (ESLint/Prettier for JS, Pylint/black for Python) to maintain code quality. Additionally, familiarize yourself with **CI tools** (like GitHub Actions) to automate tests on each commit – this ties into DevOps. There are freeCodeCamp certifications (e.g. *Quality Assurance*) that cover testing methodologies and even some QA tools; completing those will ensure you've touched on key QA concepts. By the end of this journey, you should be comfortable writing basic tests and understand the importance of QA in software projects (this is something you can discuss in interviews to show maturity).

Security Best Practices

While you don't have to be a security expert, recruiters value developers who are mindful of security. Ensure you learn **web security fundamentals**: common vulnerabilities like the OWASP Top 10 (SQL injection, XSS, CSRF, etc.) and how to mitigate them. For example, understand the importance of input validation and using parameterized queries (to prevent injections), how to escape or sanitize user input for HTML (to prevent XSS), and how to handle authentication securely (storing passwords hashed, using HTTPS, etc.). When building projects, implement features like secure password storage, role-based access control, and use HTTP headers to improve security (e.g. Content Security Policy). Learn about **CORS** and how to configure it on your backend when your front-end is on a different domain. Also familiarize yourself with basic encryption and hashing concepts, and how to safely use third-party libraries (keeping dependencies updated to avoid known vulnerabilities).

On the network side, since you'll work with cloud, understand basics like setting up a firewall/security group, using SSH keys for server access, and not exposing sensitive credentials (use environment variables or secret managers). A part of security is also **secure coding practices** – for instance, avoiding using eval in JS, handling errors so they don't leak sensitive info, etc. Completing a short course or module on security (for example, freeCodeCamp's Information Security lessons or a LinkedIn Learning course on secure web development) would be beneficial. By being security-aware, you can mention in interviews how you built your projects with security in mind, which is a professional touch.

System Design Fundamentals

As you approach job-readiness, gain an understanding of high-level **system design**. While new graduates are not always grilled on system design, having a grasp of it will set you apart [15]. Learn how large-scale systems are structured: for example, how a typical web request flows through load balancers to servers to databases. Key concepts to cover include **scalability** (vertical vs horizontal scaling), **caching** (using systems like Redis or CDNs), **database indexing and replication**, **load balancing**, and basics of microservices vs monoliths. Understand the trade-offs in system design (CAP theorem for databases, consistency vs availability). Study a few classic system design interview questions (design a URL shortener, design a social media feed, etc.) and outline how you would approach them – focusing on defining requirements, proposing an architecture with various components (web servers, databases, caches), and discussing how to handle large number of users or data.

Familiarize yourself with concepts like message queues (e.g. RabbitMQ or AWS SQS) and when to use them, and **design patterns** for scalability (like sharding, master-slave DB, stateless services). You don't need to implement these, but you should be able to discuss them. A good resource is the **"Grokking the System Design Interview"** material [16] or similar courses, which cover common design problems and solutions. Aim to be able to do a 30-minute high-level design discussion. This knowledge will also inform how you build your capstone projects (you can incorporate some elements like caching or load testing to demonstrate system design considerations). By understanding system design fundamentals, you'll be more confident in interviews and can tackle any system-oriented questions even as a new grad.

DevOps and CI/CD

Having DevOps skills will **seriously impress recruiters** for a new grad. Get hands-on with **Docker**: learn to containerize your applications (create a Dockerfile for your Node app and your Python app). Understand how containers work and practice running multi-container setups with **Docker Compose** (for example, a web app container + database container). This will teach you how to package and run apps consistently across environments. Next, learn the basics of **Continuous Integration/Continuous Deployment (CI/CD)**. Using **GitHub Actions** (or another CI service), set up workflows that automatically run tests and linting on each push, and even deploy your app. For instance, configure an Action to build your Docker image and push it to Docker Hub or to deploy to a cloud service – automating deployments **before they become emergencies** [17]. This demonstrates an ability to streamline the development process.

Additionally, explore **Infrastructure as Code (IaC)** with a tool like **Terraform**. Learn how to write a Terraform script to provision resources on AWS (e.g., a VPC, an EC2 instance, and an RDS database). Even a simple Terraform project to spin up a server and database for one of your projects will give you insight into how cloud infrastructure is managed in a professional setting. You don't need to be a Terraform guru, but knowing the basics (providers, resources, state) is valuable.

Gain familiarity with the concept of **Kubernetes** – you don't have to deploy a K8s cluster, but know what it is (container orchestration) and the basic terminology (pods, services, deployments). Maybe try deploying your Docker container to a local Minikube or use Docker's built-in Kubernetes just to see it in action. This way, you can speak about containers and even touch upon Kubernetes in interviews if asked (many companies like to know you're aware of it).

By integrating DevOps into your skill set, you become the kind of full-stack developer who not only writes code but can also deploy and maintain it. It shows you can work in a team environment where CI/CD pipelines and containerization are used. Highlight these in your resume (e.g. "Containerized applications with Docker and set up CI/CD pipelines using GitHub Actions"). This practical DevOps exposure will make you stand out.

Professional Skills and Networking

Technical skills alone aren't enough – companies highly value **soft skills** and professional practices. Communication, teamwork, and adaptability are crucial: in fact, *92% of hiring professionals say soft skills are as important as tech skills* [18]. Work on explaining complex ideas clearly, both in written form (documentation, README files for your projects) and orally (practice talking through your code and solutions). Collaborate whenever possible – if you can find a partner or an open-source project, contributing will show you can work in a team. Agile methodologies are common in industry, so understand the basics of Agile/Scrum (sprints, stand-ups, user stories) and perhaps use a Trello or GitHub Projects board to track your tasks during this self-study, as if you were your own project manager.

Begin building your professional online presence: update your **LinkedIn** to reflect your new skills and projects, and consider writing a few short articles (on LinkedIn or a personal blog) about things you learned – this can demonstrate communication skills and passion. Networking can also help: connect with peers, join tech groups or forums (like contributing on Stack Overflow or the freeCodeCamp forum), and perhaps attend virtual meetups or hackathons to meet other developers.

Resume and interview skills: As you near the end of the 9 months, you'll create a polished resume highlighting your projects, skills, and certifications. But well before that, keep note of your achievements so you can craft strong bullet points (e.g. "Developed a full-stack web app using React, Express, and MongoDB, implementing user authentication and RESTful APIs"). Prepare for behavioral interviews by reflecting on experiences from your projects – for example, challenges you overcame debugging an issue, or how you planned and executed a complex feature (use the STAR method to structure these stories). Employers will ask about teamwork, learning from failure, etc., so draw on school projects or any collaboration you did (or even organizing a study group).

Lastly, **time management and consistency** are soft skills to cultivate during this self-study – demonstrate (to yourself and eventually employers) that you can set long-term goals and follow through daily. This entire plan is like a marathon; by sticking to it, you'll also prove your dedication and self-motivation, which are traits every recruiter loves to see.

Key Certifications to Pursue

Earning industry-recognized certifications can greatly strengthen your resume (especially since you lack internships). Here are valuable certs to target during this 9-month journey, aligned with your learning:

- **AWS Certified Cloud Practitioner (AWS)** – *Foundational cloud certification*. Validates understanding of AWS core services, security, and cloud concepts [10]. Ideal as a first AWS cert; shows you know cloud basics.
- **AWS Certified Solutions Architect – Associate (AWS)** – *Associate-level cloud architect cert*. Demonstrates ability to design and deploy AWS solutions, including selecting the right services and

understanding architecture best practices [11] . Highly regarded by employers; aim to achieve this after Cloud Practitioner.

- **Google IT Automation with Python Professional Certificate (Google)** – A 6-course certificate focused on Python scripting, Git, IT automation, and using cloud/tools with Python [12] [13] . This will reinforce Python skills and show experience with automation and cloud (plus it's by Google, which stands out).
- **Google Cloud Certification** – *Optional:* Consider the **Google Cloud Digital Leader** or **Associate Cloud Engineer** cert to validate knowledge of GCP. If aiming higher, the **Google Professional Cloud Architect** is prestigious (but heavy). Even if you don't get the cert, having some GCP training allows you to mention familiarity.
- **Microsoft Azure Fundamentals (AZ-900)** – Entry-level cert for Azure. Shows you grasp Azure's services and cloud concepts. Quick to obtain and gives you the trifecta of cloud providers on your resume.
- **Meta Front-End Developer Professional Certificate (Coursera)** – A comprehensive program by Meta covering HTML, CSS, JavaScript, React, and more (including a capstone project). Completing this not only teaches you front-end thoroughly but also earns a certificate from a FAANG company (Meta) which is a great talking point. It consists of about 9 courses (e.g. Intro to Front-End, Programming with JS, Version Control, HTML/CSS in depth, React basics, React advanced, UX design, and a final project). By finishing it, you'll have additional projects and a certificate to show for your front-end skills.
- **freeCodeCamp Certifications (free)** – freeCodeCamp offers multiple developer certifications that you should earn as you progress:
- *Responsive Web Design* (covers HTML/CSS)
- *JavaScript Algorithms and Data Structures* (covers JS basics and algorithm scripting practice)
- *Front End Development Libraries* (covers React, Redux, and related front-end tools)
- *Data Visualization* (covers D3.js – optional but good for expanding front-end skills)
- *Back End Development and APIs* (covers Node, Express, and MongoDB)
- *Relational Database (SQL)* (covers SQL and relational DB fundamentals)
- *Quality Assurance* (covers testing with Chai, and some Node security)
- *(Others:* freeCodeCamp also has Python, Data Science, InfoSec, etc., but the above are most relevant to web dev.)*

  Each freeCodeCamp cert requires completing lessons and projects. By earning these, you not only gain skills but also have 5-10 small projects (per cert) that you can showcase. Aim to collect the full stack-oriented ones (responsive web design through QA). These certifications are well-known in the dev community and show you've done extensive self-driven work [19] [20] .
- **IBM Professional Certificates (Data Science, AI Engineering)** – *Optional:* IBM's certificates on Coursera (IBM Data Science, IBM AI Engineering) are valuable if you want to demonstrate knowledge in data science or machine learning. They cover Python, SQL, machine learning techniques, and deep learning. These are **not required** for a full-stack web developer, but if you have interest and time, doing one of them can add a differentiator on your resume (showing breadth into AI/ML). For example, IBM Data Science covers data analysis with Python, which reinforces your Python skills and shows you can handle data. IBM AI Engineering covers machine learning with TensorFlow/PyTorch – if you're curious about AI, this could be worthwhile. Prioritize these after the core web-dev and cloud certs, or consider them stretch goals.
- **HackerRank Skill Certifications** – HackerRank offers free skill certification tests (30-60 min exams) in topics like Problem Solving, Java, Python, JavaScript, SQL, etc. These are great quick wins to prove your coding abilities. Plan to take:
- *Problem Solving (Intermediate)* – shows your general algorithm proficiency.

- *Java (Basic or Intermediate)* – to certify your Java knowledge.
- *Python (Basic or Intermediate)* – to certify Python skills.
- *JavaScript (Intermediate)* – to certify JS ability.
  Each certification you pass gives you a badge on your HackerRank profile which you can share on LinkedIn. They signal to recruiters that an external platform verified your skills. Attempt these once you feel comfortable in each language (likely mid-way or later in the plan).
- **LinkedIn Learning and Skill Badges** – Utilize LinkedIn Learning for courses on any topics you need extra help with (e.g. a course on Git, or on AWS prep, or soft skills like interview prep). When you complete courses or learning paths, you can often add certificates to your LinkedIn profile. Additionally, LinkedIn has *Skill Assessments* for things like Java, Python, JS, React – passing these quizzes gives you a badge on your profile (e.g. "JavaScript Skill – Expert"). These are worth doing to polish your profile. While not as heavyweight as the above certs, they show you're proactive. For example, complete the "Become a Full-Stack Web Developer" learning path on LinkedIn Learning for a structured review and a shiny certificate of completion.

In summary, by May 2026 you should have a collection of **high-value certifications**: AWS (definitely), one from Google, one from Azure, the Meta certificate, several freeCodeCamp certs, and any others you choose to pursue. These, combined with your projects, will reassure recruiters of your self-taught expertise. Remember not to chase certifications at the *expense* of practical skills – use them to structure your learning and validate your knowledge, but always pair them with hands-on projects to apply what you learn.

Week-by-Week Roadmap (Sept 2025 – May 2026)

Below is a detailed week-by-week plan. Each week lists focus areas, daily practice, project work, and certification goals. Plan on ~4–8 hours of work **every day** (adjust tasks based on whether you have 4 or 8 hours that day). This roadmap assumes you can dedicate significant time daily; if you need to adjust, ensure the sequence is maintained. Treat it like a structured bootcamp: each week builds on previous ones. You'll do **data structures & algorithms practice every week**, continuously, alongside the highlighted focus.

(For brevity, we use Mon-Sun as the week format, starting today. Adjust exact dates as needed.)

Week 1 (Sept 23–30, 2025) – Setup and Foundations

- **Focus – Programming Basics & CS Refresh:** Set up your development environment (install VS Code, Node.js, Python, Java JDK, Git). Refresh core CS concepts: read up on Big-O notation, review basic data structures (arrays, linked lists) and algorithms (searching/sorting). If your CS theory is rusty, spend some hours reading a primer or watching a crash course on algorithms.
- **Web Foundations:** Begin **HTML & CSS**. Go through an HTML tutorial and learn about elements, tags, and attributes. Similarly, learn basic CSS selectors, properties, and how to link CSS to HTML. Practice by building a very simple personal homepage (e.g., an "About Me" static page with some paragraphs, images, and links). This will warm you up in actual coding.
- **Daily DS & Algo Practice:** Start easy: each day solve 1-2 easy problems on arrays or strings. Use Python or JavaScript for coding them. For example, solve problems like "two sum" or "reverse a string". Focus on understanding how to translate problem description into code and practice writing solutions without looking at answers. Keep track of every problem solved (perhaps maintain a spreadsheet or use LeetCode's tracking). Aim for ~5 problems solved this week.

- **Git & GitHub:** Initialize a git repository for your website project. Make your first commits as you work on the HTML/CSS files. Create a GitHub account (if you haven't) and push your repo. By the end of the week, you should have your first repository on GitHub – a simple website. This also means learning basics of `git add/commit/push`. (If you need a quick Git intro, take a one-hour Git tutorial mid-week.)
- **Certifications/Other:** Enroll in freeCodeCamp's **Responsive Web Design** certification. As you learn HTML/CSS from other resources, start working through the freeCodeCamp HTML/CSS exercises too (they will reinforce your learning). No exam this week, but get into the learning platforms you'll use. Also, set up accounts on LeetCode, HackerRank, etc., and familiarize yourself with their UI for coding practice.

Week 2 (Oct 1–7, 2025) – HTML/CSS Mastery & Basic JS

- **Focus – Responsive Web Design:** Continue and complete the freeCodeCamp **Responsive Web Design** curriculum [21] [19]. This involves lessons on HTML forms, CSS flexbox, CSS grid, etc. By end of this week, build the required FCC projects: a *Tribute Page* and a *Survey Form* (at minimum). Also try the *Product Landing Page* and *Personal Portfolio Webpage* projects if time permits [22]. These projects will solidify your HTML/CSS skills – ensure they are mobile-responsive. Publish these pages (you can use GitHub Pages or Netlify to host static sites) to start building a portfolio.
- **JavaScript Fundamentals:** Begin learning **JavaScript basics**. Work through freeCodeCamp's **JavaScript Algorithms and Data Structures** sections or an equivalent JS tutorial. Cover variables, data types, loops, functions, and basic DOM manipulation (querying elements, responding to button clicks). Write a few simple scripts – for example, add a script to your survey form that validates inputs or a button that toggles content. Hands-on JS practice is crucial here.
- **Daily DS & Algo Practice:** Increase to 1–2 problems per day (still easy-level). Now include problems on linked lists or stacks if you've reviewed those. If you studied an algorithm (say binary search) this week, solve a problem using it. Keep practicing writing clean solutions. (Goal: another ~7 problems this week, running total ~12).
- **Git/GitHub:** Create a new repository for any JS experiments you do (or continue using the site repo). Practice using branches – e.g., create a branch for adding a new feature to your project, then merge it. Also refine your GitHub profile: add a professional photo, write a bio mentioning you're a CS student learning full-stack development. Little touches like these make your profile attractive to recruiters early on.
- **Certifications:** By now you should be nearly done with **FCC Responsive Web Design** – attempt the final certification exam/assessment and claim that certificate. This is your first cert achieved! Start the **FCC JavaScript Algorithms** exercises as well (these double as algorithm practice). No external cert exam yet, but plan out when to tackle AWS Cloud Practitioner (probably in November) – perhaps purchase a course (e.g., on Udemy) or find AWS's free Cloud Practitioner Essentials material, and schedule a tentative exam date.

Week 3 (Oct 8–14, 2025) – JavaScript Deep Dive

- **Focus – JavaScript Programming:** Dive deeper into JS. Cover advanced topics like objects and arrays in depth, JSON, and the basics of ES6 (arrow functions, let/const, template strings). Learn about higher-order array methods (map, filter, reduce) to manipulate data – these are commonly used. Implement a few small JS exercises: e.g., write a function to find primes, or to sort an array of objects. Also get comfortable with the Document Object Model: practice selecting elements and

updating the DOM. For example, enhance one of your previous HTML projects by making something interactive (hide/show sections, validate form fields in real-time, create a simple image slider, etc.).

- **Browser Dev Tools:** Spend some time learning to use Chrome/Firefox Developer Tools – how to inspect elements, debug JS with breakpoints, and view console logs. This skill will help immensely when building front-ends.
- **Daily DS & Algo Practice:** Continue with 1-2 problems daily. This week, include some slightly trickier ones (e.g., two-pointer technique or string manipulation problems). Since you're learning JS deeply, try solving some problems in JavaScript (to kill two birds with one stone). If you get stuck on a problem, read discussions but *then* implement the solution yourself. Running total ~20 problems by end of week.
- **Project – JS Mini-Project:** Build a small purely front-end project using JS. For instance, a **Calculator app** (classic beginner project) or a **To-Do List** that lets users add and remove items and saves them in localStorage. This will reinforce JS knowledge and give you a portfolio piece. Make sure to style it nicely (use your CSS skills) and deploy it (e.g., on GitHub Pages). Commit your code frequently.
- **Certifications:** If you're progressing through freeCodeCamp's **JavaScript Algorithms and Data Structures**, aim to complete it or at least finish significant sections (the algorithm scripting challenges). This not only teaches JS but also gives more algorithm practice. You could potentially earn this FCC certificate by next week. Additionally, consider enrolling in the **Meta Front-End Developer** courses on Coursera now, if you want to follow that structure; the early courses (Intro to Front-End, Programming with JS, Version Control) align with what you're doing now. They can supplement your learning and you'll eventually earn Meta's cert if you complete all. Plan out a schedule for those courses (maybe one course per week or two).

Week 4 (Oct 15–21, 2025) – Interactive Web & Git Mastery

- **Focus – DOM and Events:** This week, focus on making webpages truly interactive. Learn about **DOM events** (onclick, onsubmit, etc., and using `addEventListener`). Practice by adding event handlers in your mini-projects: e.g., form validation messages on submit, or pressing a button changes content. Understand event propagation (bubbling and capturing) conceptually. Also learn about **asynchronous JS**: introductions to callbacks, promises, and `async/await`. Do a simple async task like fetching data from a public API (e.g., use `fetch()` to get weather data or random quotes) and display it on a page. This introduces you to HTTP requests from the front-end.
- **CSS Framework Intro:** Try using a CSS framework like **Bootstrap** or **Tailwind** on one of your projects. For example, rebuild your portfolio page or to-do app using Bootstrap components. This saves design time and makes your projects look more polished. It also prepares you for using UI libraries professionally.
- **Daily DS & Algo Practice:** You should now be comfortable with easy problems; try a couple of medium-level problems towards end of week. Continue solving at least one per day. Possibly tackle problems involving maps/dictionaries or simple recursion. Aim to reach ~30 problems solved total by end of week.
- **Project – Personal Portfolio Site v1:** By now you have a couple of small projects (tribute page, to-do app, etc.). It's time to create a **portfolio website** to showcase everything. You can use a template or build from scratch with your HTML/CSS/JS skills. Include an "About Me", a list of projects (with links and screenshots), and contact info. This will evolve over time, but get a basic version up. Use this project to practice **responsive design** thoroughly (media queries, flexbox/grid). Deploy it to GitHub Pages or Netlify with a custom domain if possible (you can buy a cheap domain for your name). This becomes the central place to send recruiters.

- **Git & Collaboration:** This week, focus on mastering git workflows. Learn how to use branching effectively: create a `dev` branch on your portfolio site repo, make some changes, then open a Pull Request (you can even simulate a team by having a friend review or just review yourself) and merge it. Practice writing clear commit messages. If you're doing the Meta version control course, finish that to solidify these concepts.
- **Certifications:** Achieve the **freeCodeCamp JavaScript Algorithms and Data Structures** certificate by completing its final challenges and test. Now you have two FCC certs (HTML/CSS and JS) – great progress in one month. Also, start lightly studying for **AWS Cloud Practitioner** – perhaps read one or two chapters of AWS documentation or watch intro videos on AWS cloud concepts in the evenings. (Don't worry if it's not coding; you'll accumulate knowledge for the exam by December.)

Week 5 (Oct 22–28, 2025) – Moving to Front-End Frameworks (React)

- **Focus – React Basics:** Transition into **React.js** this week. Begin with the basics: understand why we use frameworks (single-page app needs, state management, etc.). Set up a React development environment (use Create React App or Vite to bootstrap a project). Learn about components, JSX syntax, and how state and props work. Follow a React tutorial or the official docs to create a simple app – for instance, a "Hello World" that dynamically updates, or a small **Calculator** in React (reimplement your earlier vanilla JS project to see the differences). Get comfortable with functional components and hooks like `useState`.
- **TypeScript Intro:** As you start React, also introduce **TypeScript** in a simple context. Try converting a couple of JavaScript examples or small functions to TypeScript, learning about interfaces and types. You might hold off using TS in React until you're comfortable with React itself, but do at least an intro TS tutorial or read on how TS would apply to React components (maybe install TS in your React app toward end of week and convert one component as an experiment).
- **Daily DS & Algo Practice:** Keep the momentum – solve 1-2 problems daily (mix easy and medium). Try to use Python for a few problems this week to keep your Python fresh (for example, solve a string manipulation or math problem in Python). Problems involving two-dimensional arrays or simple graphs (like DFS on a grid) could be introduced now since you have done basics. Total solved by end of week ~38+.
- **Project – React mini-project:** Build a small React project by end of week. Good starter ideas: **React To-Do List** (again, but now using React state), or a **Movie Search App** that calls an API (like OMDB) and displays results. This will teach you React component structure and making network requests in React (`fetch` in useEffect). Aim to practice component composition (maybe create multiple components: search bar, result list, result item, etc.). Style it with CSS or a component library (could try Material-UI or Chakra UI for experience). Deploy this React app using Netlify or Vercel for bonus points.
- **Team Collaboration Simulation:** If possible, collaborate with a peer on the React project – even if it's one other student, pair program or assign tasks to simulate teamwork. Alternatively, contribute to an open-source issue in a beginner-friendly repository (it could be a documentation fix or a small bug). This will give you real-world collaboration experience and something to mention on your resume ("Contributed to open source on XYZ project").
- **Certifications:** Continue the **Meta Front-End** courses focusing on React (they have separate courses for React basics and advanced React – you might finish React Basics course this week). Also, now that you have FCC JS cert, start FCC **Front End Development Libraries** certificate (which covers React and Redux). The projects for this (like building a Random Quote Machine or a Markdown Previewer in React) can be part of your practice – possibly use one as your mini-project. No external exam yet, but *schedule* your **AWS Cloud Practitioner exam** for late November now, to give yourself a deadline

(AWS allows scheduling in advance; schedule about 4-5 weeks out and work backwards [23]). This will motivate you to start studying more earnestly soon.

Week 6 (Oct 29 – Nov 4, 2025) – Advanced React & Front-End Capstone

- **Focus – React Advanced Topics:** Build on your React knowledge: learn about **React Router** for multi-page SPA navigation. Introduce a global state management solution – the simplest might be React's Context API or go for **Redux** if you're feeling adventurous (the FCC curriculum covers Redux in the Front-End Libraries cert). Understand the problem Redux or context solves (prop drilling) and implement a simple global store (for example, in a to-do app, use context to provide the to-do list state to deeply nested components). Also explore React hooks beyond useState: e.g., `useEffect` for side effects (fetching data, etc.), `useContext`, and if comfortable, `useReducer` or custom hooks. This week, also pay attention to optimization: learn about React Developer Tools, and concepts like memoization (`React.memo`, `useMemo`, `useCallback`) just at a high level so you know they exist.
- **Styling React Apps:** Try a CSS-in-JS approach or a component library. For instance, use **Styled Components** or try **Tailwind CSS** in your React project to see different ways to style. This makes your React apps look professional without needing a separate designer.
- **Daily DS & Algo Practice:** Continue daily practice. Focus on any weak areas you've noticed. Maybe dedicate one day to purely revising previously solved problems (to ensure you remember patterns). Solve at least 5-7 new problems this week. Consider upping to 2 problems most days, since by now algorithms should be a habit. Running total ~45.
- **Project – Full Front-End Project:** Build a more complex **front-end application** in React that could be portfolio-worthy. Ideas: a **Personal Portfolio v2 using React** (replace your static site with a React app for practice, maybe with some animations or extra interactivity), or a small **blog frontend** (that fetches posts from a placeholder API), or a **dashboard** that uses charts (could use Chart.js or D3 for a couple of graphs, integrating data visualization skills). Aim to implement at least: multiple routes (pages), stateful components interacting, and API calls. Ensure it's polished in UI/UX (mobile-friendly, no obvious bugs). This project can be one of your highlights to show off React skills.
- **Certifications:** By end of this week, you should finish freeCodeCamp's **Front End Development Libraries** cert, which includes React and Redux, by completing its projects (Random Quote Machine, Markdown Previewer, Drum Machine, JavaScript Calculator, Pomodoro Timer). Completing these will both give you the certificate and additional mini-projects (if you haven't already done similar ones yourself). Also, aim to complete the **Meta Advanced React** course if you're doing Meta's program. Simultaneously, allocate some time this week to AWS Cloud Practitioner study – cover AWS basics like EC2, S3, RDS, IAM in theory (use AWS's free training or Stephane Maarek's course, etc.). You might spend e.g. 1 hour each evening reading AWS whitepapers or doing practice questions [24] [25]. This will ramp up in upcoming weeks, but starting now is good.

Week 7 (Nov 5–11, 2025) – Back-End Beginnings (Node.js & Express)

- **Focus – Node.js and Express Fundamentals:** Shift focus to the back-end now. Start learning **Node.js** runtime and **Express.js** framework. Begin with Node basics: how the event loop works, using npm to install packages, and writing a simple "Hello World" server without Express (using Node's built-in `http` module) just to understand the fundamentals. Then dive into Express: learn how to define routes (GET, POST, PUT, DELETE), how to parse JSON request bodies, and how to send responses. Follow a basic tutorial to build a simple REST API – for example, an API for a to-do list or notes (since you've done to-do in front-end, doing it in back-end makes sense). Implement routes

like `GET /todos`, `POST /todos`, etc., storing data in memory for now. Test these routes using a tool like Postman or curl.
- **Understanding APIs:** Also learn about JSON and REST principles (resources and HTTP methods). This knowledge will help you design good APIs for your projects.
- **Daily DS & Algo Practice:** Maintain your algorithm routine (don't let it drop as you switch focus to backend). Possibly integrate some **algorithmic thinking in Node**: write a script in Node to solve a coding challenge (so you see how to handle input/output in Node environment). Continue solving problems (another ~7 this week, crossing ~52 total).
- **Project – Express API Project:** Expand your Express learning into a small project: build a **RESTful API** with Express that uses a real database. This could be a backend for a simple app – for example, a **Blog API** (routes for posts and comments) or a **Task Manager API**. Use MongoDB as a quick start for a database (perhaps via **Mongoose** ODM). This week, focus on implementing CRUD operations and testing them via Postman. Don't worry about front-end, just ensure your server works (you'll connect a front-end later). Structure your project neatly (routes folder, controllers, etc.). Learn how to handle errors and send proper HTTP status codes.
- **Databases Intro:** If you haven't used MongoDB before, spend time learning its basics: setting up a local MongoDB or using a cloud MongoDB Atlas, understanding collections and documents. Practice writing queries in Mongo (find, insert, update). Integrate it with your Express app (store and retrieve data from the DB instead of an array). This is your introduction to databases in practice.
- **Certifications:** As you foray into backend, start freeCodeCamp's **Back End Development and APIs** certification. It covers Node, Express, and MongoDB, and has projects like building a URL Shortener, Exercise Tracker, etc. These align well with what you're doing – try to complete one of the FCC projects this week (for example, the URL Shortener project) as it will reinforce Express/Mongo skills. Additionally, consider scheduling the **Azure Fundamentals (AZ-900)** exam for a couple months out, or at least get free study materials for it to use later. Keep chipping away at AWS Cloud Practitioner prep – at this point ensure you cover the basics of AWS security and pricing (common exam topics) while continuing to review services.

Week 8 (Nov 12–18, 2025) – Back-End Development & Database Skills

- **Focus – Deeper into Back-End:** Continue building out your Node/Express backend skills. Learn about **middleware** in Express (for logging, authentication, error handling). Implement a simple middleware (e.g., one that logs request info, or a middleware to handle unknown routes with a 404). Also, learn about **user authentication** basics: try adding a simple registration/login to your API (you can use an npm package like `bcrypt` to hash passwords, and maybe `jsonwebtoken` to issue a JWT for auth). This might be a lot, but even implementing a basic JWT auth for one route (protected route that requires a token) will teach you important concepts.
- **SQL Database Introduction:** Alongside your NoSQL (Mongo) experience, get started with a SQL database. Install **PostgreSQL** or use an online SQL sandbox. Go through SQL query tutorials: learn creating tables, basic SELECT queries with JOINs, etc. Try writing queries against a sample dataset. Then, in code, try using PostgreSQL with Node (perhaps using an ORM like Sequelize or Prisma, or the node-postgres library). If it's too much to integrate in one of your projects now, at least do a mini exercise: write a Node script that connects to a Postgres database, creates a table, inserts a row, and reads it. This will demystify using SQL from code.
- **Daily DS & Algo Practice:** Now that you've covered most basic DS, try more complex problems. Include one or two **linked list** or **binary tree** problems this week (since those often trip people up). Also practice writing code on paper or a whiteboard for one problem to simulate interview conditions. Continue with ~7 problems/week pace (total ~60 by now).

- **Project – Full-Stack Integration (MERN stack):** It's time to tie front-end and back-end together. Take one of your earlier front-end React projects (maybe the blog or task app) and connect it to a back-end API. For example, if you built a blog front-end, use your Express/Mongo backend as the data source. Adjust your React app to fetch from your Express API (you might have to enable CORS on the backend). This will teach you how the front-end and back-end communicate (via HTTP and JSON). Alternatively, build a new **full-stack MERN project**: for instance, a **Simple Social Network** (users can create an account, post messages, and view others' posts) or a **Quiz app** (backend serves questions, front-end displays and posts answers). Keep it small in scope but cover the full-stack: database, API, and UI. Deploy this if possible – you could deploy the front-end on Netlify and backend on Heroku or render.com (since AWS setup might take more work). Seeing a project live end-to-end is rewarding and will highlight any issues in integration.
- **Version Control & DevOps:** Use this project to practice a mini DevOps workflow: set up a basic **CI pipeline** with GitHub Actions – for example, an Action that runs `npm test` (if you have tests) or lints your code on each push. It can simply echo "Tests passed" if none exist, just to get familiar. Additionally, if deploying, figure out an automated way: maybe the Action deploys the front-end to Netlify via their CLI or triggers a build on push. This is optional, but good exposure.
- **Certifications:** Aim to complete the **freeCodeCamp Back End Development and APIs** cert by finishing its projects this week or next. Those projects (like the exercise tracker, file metadata microservice, etc.) can double as learning exercises for you – each covers different facets of backend. Also, since AWS Cloud Practitioner exam is likely in ~2 weeks, ramp up your study: do practice exam questions, and review any weak areas (like CloudWatch, SNS, etc.). By end of this week, you should consistently score ~80%+ on practice exams for Cloud Practitioner. Book a specific date if you haven't – likely in Week 10 – and ensure you have an ID and everything ready for the online proctored exam.

Week 9 (Nov 19–25, 2025) – Python for Scripting & Cloud Fundamentals

- **Focus – Python Programming:** Shift focus this week to your second language: **Python**. Even if you learned Python in school, now you'll apply it in practical ways. Start with a quick refresh of Python syntax (through a "Python for JavaScript developers" comparison if needed). Do a small project or scripts with Python: for instance, write a **Web Scraper** using `requests` and `BeautifulSoup` to fetch data from a website, or an **Automation Script** (maybe something that reads a CSV and performs an operation, or automates a system task on your computer). This shows Python's strength in scripting. Also explore Python's standard library for file I/O, JSON, etc.
- **Python Web Framework:** Try building a **simple web API using Flask**. For example, replicate one of your Express APIs in Flask to see differences. Create a couple of routes and return JSON. This gives you a taste of Python back-end development. If ambitious, experiment with a micro web framework like **FastAPI**, which is modern and uses Python type hints. You don't need a full project, just implement a simple endpoint. This will allow you to say you've built an API in both Node and Python.
- **Daily DS & Algo Practice:** Use Python for solving some algorithm problems this week to reinforce your Python skills. Solve perhaps 5 problems in Python (you'll notice some problems are quicker to implement in Python due to libraries). Since you're juggling a lot this week, even 5 solid problems is fine. If you can, try at least one more challenging problem (maybe a medium that involves combining data structures or more logic). Total ~65 by end of week.
- **Cloud – AWS Core Services Practice:** In preparation for the AWS exam and to solidify cloud knowledge, do a **hands-on AWS exercise**: for example, deploy a simple website or API on AWS. Perhaps upload your portfolio site's static files to **AWS S3** and serve it via an **Amazon CloudFront** CDN (there are tutorials for hosting a static site on S3). Or launch an **EC2 instance** (free tier) and deploy your Express app to it (you'll need to set up Node on the server, etc.). This will teach you

practical AWS skills like using the AWS console, configuring security groups, and SSH into a Linux VM. Additionally, familiarize yourself with AWS Lambda by writing a small Lambda function (maybe a function that returns a greeting) via AWS Console or using AWS SAM. Even just doing the hello-world will help you grasp serverless basics.

- **Project – AWS Integration:** If possible, integrate AWS into one of your projects: for example, modify your full-stack project to store uploaded images in **S3** instead of on the server, or use AWS SES to send an email, etc. A simpler integration: use AWS's free **RDS** tier to host your project's PostgreSQL database on AWS and connect your app to it (this showcases knowledge of cloud databases). These kinds of touches in a project can really impress (showing you can use cloud services).
- **Certifications:** This week, **pass the AWS Certified Cloud Practitioner exam** (if scheduled around now). Dedicate a day or two earlier in the week to final review and take the exam (typically 90 minutes). Once you pass, celebrate – you now have your first major cloud cert! Add the badge to your LinkedIn. Immediately after, begin studying for **AWS Solutions Architect – Associate**, which you plan to take by March. The Associate exam is more in-depth; start by tackling one domain at a time (e.g., networking and VPCs, storage solutions, etc.). But you can do this gradually in coming weeks. Also, since you focused on Python, consider enrolling in the **Google IT Automation with Python** certificate if not already – it might cover things you just did (like using Git, debugging, automation), and completing it will be a nice credential from Google. You could plan to complete those 6 courses over December-January.

Week 10 (Nov 26 – Dec 2, 2025) – Java Programming Foundations

- **Focus – Core Java:** Now turn to **Java**, your supportive language. Install an IDE like IntelliJ or Eclipse. Start with the basics: syntax differences from Python/JS, strong typing, and the Java compile/run cycle. Write simple programs (Hello World, basic loops, functions). Then practice OOP in Java: create a few classes (e.g., a `Student` class with fields and methods) to get a feel for classes and objects. Understand concepts like constructors, getters/setters, inheritance (maybe make a subclass), and interfaces vs abstract classes. This will reinforce OOP understanding that's useful in any language.
- **Java Data Structures & Algorithms:** Use Java to implement a couple of data structures manually (for learning): e.g., write a LinkedList class or a binary search tree insertion. This is mainly for practice. Also, solve 2-3 algorithmic problems in Java this week (perhaps ones you solved in Python/JS before, to compare). This ensures you're somewhat comfortable coding in Java for interviews if needed.
- **Intro to Java Backend (Spring)**: Spring Boot is the common Java backend framework. Set up a simple Spring Boot project (using Spring Initializr). It can be overwhelming, so aim small: create one REST controller with one or two endpoints. You'll need to learn some annotations (@RestController, @GetMapping, etc.) and how to run the app. If time permits, connect this to a database using Spring Data JPA just to see it (or use an in-memory H2 DB). The goal isn't to master Spring, but to be able to say you've created a basic API in Java. This experience will help in interviews if a company uses Java – you can mention you've dabbled in Spring.
- **Daily DS & Algo Practice:** With three languages under your belt now, decide which language you'll use by default in interviews – many choose Python for speed. Continue practicing algorithms in that main language to build speed. This week, try focusing on algorithms that use more advanced techniques: e.g., some dynamic programming (like a simple Fibonacci with memoization, or coin change problem) or more complex graph traversal. Solve at least 5 problems. Total ~70 by end of week.
- **Project – Java Mini-Project:** Do a small project in Java to consolidate what you learned. For example, a **Command-line Address Book** (store contacts, search them) or a mini text-based game.

Alternatively, extend your Spring Boot test from earlier into a tiny project: e.g., a **Library API** (with endpoints to add/search books). Even if not full-fledged, having some code in Java on your GitHub is beneficial. Ensure to commit this work as a separate repo.

- **Soft Skills – Communication:** As an interlude, practice explaining one of your projects or a technical concept in simple terms (maybe to a friend or just record yourself). Communication practice is important. You could also write a short Medium or dev.to article about something you learned (maybe "How I built my first MERN app" or "5 things I learned switching from Python to Java"). This will help articulate your experience and can be shared on LinkedIn to show passion.
- **Certifications:** If you're comfortable, attempt some **HackerRank skill tests** this week or next. You just studied Java – try the **HackerRank Java (Basic)** certification test. Also, you can likely clear **Problem Solving (Basic)** easily now; perhaps attempt that too. These are typically one-hour tests. Earning a couple of HackerRank badges in languages you know (Java, Python, JS) will be quick wins. Also, since November is ending, check if any **LinkedIn Skill Assessments** align with what you learned (there's one for Java, one for OOP, etc.) – take a few to add those badges to your profile.

Week 11 (Dec 3–9, 2025) – Full-Stack Project #1 and Test/QA

- **Focus – Build a Major Project (Full-Stack):** Now that you've touched all main technologies, pick one **capstone project** to build in December (and possibly January). A great strategy is to solve a real-world problem or clone a popular application in a simplified form. For this week, plan and start building **Full-Stack Project #1**. Ideas:
- **E-commerce Store**: A mini e-commerce site where users can browse products, add to cart, and place orders. Tech: React front-end, Node/Express backend, MongoDB or PostgreSQL, perhaps Stripe API (test mode) for payments.
- **Social Media/App Clone**: For example, a simple Twitter clone (users can post tweets, follow others, and see a feed) or Instagram clone (with image upload – use AWS S3 for storing images). Tech: MERN or PERN (Postgres, Express, React, Node).
- **Project Management Tool**: Like a Trello clone where users create boards and tasks. Emphasize drag-and-drop UI and real-time updates (maybe use web sockets via libraries like Socket.io for a bonus real-time feature).
- **Personal Portfolio CMS**: A system where you have a backend to write blog posts or add projects, and a front-end that displays them. This can showcase both your dev and content skills.
  Choose one that excites you and is feasible. This week, focus on **planning** (define the scope, data models, and tech stack) and start implementing core features. For instance, set up the repository (monorepo or separate repos for front-end/back-end), implement user authentication (which every app will need), and create the database schema. Aim to get the basic backend endpoints and a skeleton frontend in place.
- **Testing & QA:** As you build this project, begin writing **tests**. Implement unit tests for some backend logic (e.g. test a utility function or a service method). Also write at least one end-to-end test if possible (using a tool like Jest to call an API endpoint and check result). For the front-end, write a simple component test (if using React, maybe with React Testing Library check that a component renders given props). It's okay if test coverage is small; the goal is to practice testing. Also set up a continuous integration workflow for this project: e.g., GitHub Actions to run tests on each push, and maybe automatically deploy the dev version to a cloud service (Netlify/Heroku). This makes your project feel professional.
- **Daily DS & Algo Practice:** This week, balance coding the project with algorithm practice. If you're coding many hours on the project, you can reduce algos to maybe 4-5 problems, focusing on keeping the habit. Ensure you solve at least one problem that involves combining multiple data

structures (e.g., a problem that uses a heap + custom comparator, or a graph traversal with a twist). If preparing for technical interviews, you may also start using a resource like *Blind 75* list or similar to guide your practice, ensuring you cover all categories. Problems solved ~75+ by now.

- **Certifications:** Work on finishing any **freeCodeCamp** certifications left: likely **Relational Database (SQL)** (if available – or just ensure you practice SQL to equivalent level) and **Quality Assurance**. For Relational DB, you might need to go through SQL exercises and a project (like building a trivia database). For QA, you'd do testing projects (like a Node app where you write tests). If time is tight, schedule these for the holiday period coming up. Also, if you haven't yet attempted, try the **HackerRank Problem Solving (Intermediate)** certification – it will test your algorithm skills under time constraint, which is good preparation for interviews. Passing it will give you a nice badge to show.
- **Note:** This week or next includes some holidays (if in the US, Thanksgiving was late Nov; December has possibly end-of-year holidays). Use any holiday breaks to either relax a bit (to avoid burnout) or to double down on coding if you're free – but maintain a sustainable pace. You've been grinding hard for 11 weeks, so short breaks are okay as long as you get back on track.

Week 12 (Dec 10–16, 2025) – Deployments and DevOps

- **Focus – Deployment & DevOps for Project #1:** Take your Full-Stack Project #1 and push it towards deployment. For the backend, set up a production environment. If you haven't yet, consider containerizing the app with **Docker**. Write a Dockerfile for the backend and one for any other service (e.g., database) or use Docker Compose to run them together. Test running your app in a container locally. Next, choose a deployment method:
- **Option 1:** Deploy on a PaaS like **Heroku** or **Railway** (quick and handles infrastructure for you).
- **Option 2:** Deploy on **AWS** for a deeper learning: maybe launch an EC2 for your backend (or use AWS Elastic Beanstalk for simplicity), set up a managed DB (AWS RDS or simply use MongoDB Atlas with AWS). If front-end is separate, host it on Netlify/Vercel as before, or use AWS S3+CloudFront.
- **Option 3:** Use **Docker** containers on a cloud service – e.g., deploy to **AWS ECS** or even attempt **Kubernetes** via a service like Google Cloud GKE or Azure AKS (only if you're eager – not required). Document whatever deployment process you do. The goal is to have at least one project deployed and accessible via a URL for recruiters to try out. Solve any deployment issues that arise (this will teach you about environment variables in production, setting up a domain, etc.).
- **Infrastructure as Code:** If deploying on AWS, try to incorporate **Terraform**: for instance, write Terraform config to provision an EC2, security group, and maybe an S3 bucket. Even if you don't fully use it to deploy (since manual might be faster this time), having some Terraform code in your repo or notes is useful to refer to and mention.
- **Improving Project #1:** Based on testing and deployment, refine your project. Fix any bugs found during testing, optimize any slow part if needed, and add at least one "nice-to-have" feature that you initially skipped (e.g., add search functionality, or add pagination to lists, etc.). This shows iterative improvement. By now, Project #1 should be near completion: a full-stack app, tested, and deployed, with documentation (update your README to include setup steps, technologies, and screenshots).
- **Daily DS & Algo Practice:** Now that one major project is winding down, ramp algorithms back up. Dedicate solid time this week to algorithms: try to solve some **LeetCode medium/hard** problems, especially in categories you find tough (e.g., if DP is still confusing, focus on that pattern). Simulate an actual interview by picking one problem, giving yourself 30 minutes to solve it and writing the solution on paper or a whiteboard, then checking. This kind of practice will build confidence. Aim for 7+ problems (maybe one per day plus extra on weekend). Total ~82+.

- **Certifications:** If you haven't taken it yet, now's a good time to attempt the **LinkedIn skill assessments** for JavaScript, Python, and Java (these are short quizzes, and a score in the top ~30% gives a badge). Also consider the **LinkedIn AWS Skill Assessment** to complement your cert – though you have AWS cert, the LinkedIn badge could attract recruiters searching those keywords. Keep chipping at AWS Solutions Architect Associate reading; December could be a good time to do a focused study sprint or even take a practice exam. Additionally, if you paused the **Google IT Automation with Python** cert, try to complete a couple of its courses during any holiday downtime (it covers Git, troubleshooting, and even a bit on automating with configuration management which is useful in DevOps).

Week 13 (Dec 17–23, 2025) – System Design and Resume Building

- **Focus – System Design Basics:** Now allocate time to formally study **system design**. Use resources like *Grokking the System Design Interview* or free YouTube lectures. Learn to outline the design of a few systems. For practice:
- Pick 2-3 common system design scenarios (e.g., design a URL shortener, design a social media newsfeed, design an online bookstore). For each, write out an approach: define requirements (functional and non-functional), outline the architecture (clients, servers, databases, external services), and consider scaling (caching, load balancing, database sharding, etc.).
- Draw diagrams for these solutions. You can use online tools like draw.io or diagrams.net to make simple architecture diagrams (this will help visual thinking).
- Think about trade-offs: If you were asked "how would you scale X to millions of users?", be ready to mention adding load balancers and horizontal scaling, etc. You might also review the basics of CAP theorem, types of databases, and message queues.
  This study will likely spill into next week as well, and that's fine – plan to cover a bit each week from now on. This knowledge not only helps interviews but can spark ideas to improve your own projects' architecture.
- **Resume Drafting:** With several projects and certs now, start drafting your **resume**. List your 8-10 best projects (you might eventually trim to 6-8 to fit one page, but have them listed first). For each project, write 1-2 bullet points focusing on achievement and tech used (e.g., "Developed a full-stack e-commerce web app using React, Node, and MongoDB, implementing JWT authentication and Stripe API integration"). Quantify if possible (users, performance improvement, lines of code – though with personal projects, focus more on tech stack and what you accomplished). Include a section for **Skills** (list programming languages, frameworks, tools, cloud platforms – you can now confidently list JS/TS, Python, Java, React, Node, Express, SQL, MongoDB, AWS, Docker, etc.). Include a **Certifications** section listing AWS, Meta, Google, etc. (with issue dates). Also include any achievements like "Solved 300+ coding problems on LeetCode" (once you reach that) – that can impress as an indicator of practice.
  Since you're a student, also mention your degree (B.Sc. Computer Science, expected May 2026), but put more emphasis on your self-taught projects and certs since that's your main strength now. Keep the resume concise (1 page). Get feedback on it if possible (maybe use an online resume review or ask a mentor).
- **Daily DS & Algo Practice:** Keep algorithms fresh – at this point, incorporate a variety of problems each week to ensure breadth. For instance, this week ensure you solve at least one problem each from categories: DP, graph, string, and math. This maintains breadth. ~5 problems is okay if you're busy with resume and system design. Total ~87.
- **Project Polish:** If you have any incomplete freeCodeCamp projects or cert projects, finish them this week so you can head into the new year with that done. Also, maybe revisit your older projects (like

your first portfolio or JS apps) – consider updating or refactoring them with your now improved skills (this can be quick touch-ups like better CSS or adding a README). Little improvements can turn a mediocre early project into a polished gem in your portfolio.

- **Certifications:** By now, ideally you have: AWS Cloud Practitioner (done), FCC Responsive, JS, Front-end Libraries, Back-end APIs certs (done), maybe FCC QA, Meta cert in progress, etc. If you haven't yet, try to take the **AWS Certified Cloud Practitioner** if you postponed it or failed (you can retake after 2 weeks if that happened). Also plan for any January exams (maybe schedule Azure Fundamentals for January, and possibly Google's Python cert completion around then). This week, maybe claim the freeCodeCamp **Quality Assurance** cert (the projects there will test your knowledge of writing tests – a nice review of what you did in Project #1).

Week 14 (Dec 24–31, 2025) – Review and Reflection (Holiday Week)

- **Focus – Review & Fill Gaps:** Use this end-of-year week to review what you've learned and fill any gaps. Go over topics to ensure you haven't missed anything important:
- Revisit **data structures**: can you recall the implementation details and common operations for each?
- Revisit **algorithms**: sorting algorithms, traversal algorithms, etc. Maybe implement one from scratch (like quicksort or BFS) in a language of your choice to keep the knowledge fresh.
- Re-read your notes or summaries from each major topic (front-end, back-end, etc.). This helps retention.
- **Side Skills:** If you find a gap, address it. For example, if you realize you haven't spent much time on **CSS Flexbox/Grid**, do a quick freeCodeCamp or Wes Bos tutorial to strengthen that. Or if your knowledge of **Redux** is shaky (maybe you used only context), do a mini tutorial on Redux to solidify the concept (since many job listings mention it).
  Another gap could be **algorithmic math** (like bit manipulation or combinatorics) – if so, do a couple of problems in that area. Essentially, this week is flexible to polish any weak spots.
- **System Design Continued:** If you didn't finish going through system design scenarios, continue that. Perhaps write a mock design doc for one of your own projects as if you were scaling it to millions of users – thinking about what you'd do (e.g., if your social app went big, how would you handle more traffic? Maybe add caching, split services, etc.). This exercise can help you discuss your projects in an interview in terms of scalability (which will really impress interviewers, as students rarely do that).
- **Career Prep – LinkedIn and Networking:** Update your **LinkedIn profile** comprehensively: add all your new certifications (AWS, etc.), add the projects in the Projects section (link to the GitHub and live demo if available), update your Skills section to include all technologies you know (people can endorse you too). Write an "Open to work" blurb in your bio like "CS student graduating May 2026, actively building full-stack projects and certified in AWS, seeking Software Engineer roles". If you haven't, make a personal portfolio site live (you did earlier as a project – ensure it's updated with latest projects & certs). This will be your hub to show others.
  Start connecting with people: join some LinkedIn groups for software engineers or new grads, and maybe reach out to a few recruiters or alumni from your school who work in tech – just to start building connections (you can mention you'll be seeking opportunities in a few months). This networking can pay off when you start applying.
- **GitHub Review:** Ensure your GitHub is organized: pin your top 6 repositories (the major projects and any standout smaller ones). Archive or delete any trivial/test repos to keep it clean. Write short descriptions for each repo. If you want, add a **GitHub README profile** (a special README that appears on your profile) summarizing you and showing stats – not necessary, but a nice touch. The idea is that if a recruiter opens your GitHub, they immediately see a well-curated selection of your

best work and proof of constant activity (you should have commits on many days – lots of green squares on the contribution graph, especially over the last 3-4 months).

- **Daily DS & Algo Practice:** If you have free time over the holidays, do some **fun coding challenges** or maybe partake in **Advent of Code** (a yearly December coding challenge event) to keep things light and fun. You can also revisit old problems you got wrong and try them again to see improvement. But it's okay to reduce intensity this week and recharge. Try to solve at least 3 problems to not lose the rhythm. Total ~90 by year's end.
- **Celebrate Progress:** You've come a long way in 3+ months! Reflect on everything you built and learned. Maybe write a summary of your journey so far (could be a LinkedIn post or a blog). This not only is personally rewarding but also shows your network what you've achieved (someone might take notice).

Week 15 (Jan 1–7, 2026) – Certifications Sprint and Project #2

- **Focus – Certifications Sprint:** Kick off the new year by securing more certs. If you planned to get the **Azure Fundamentals (AZ-900)** certification, dedicate a couple of days to study (if you haven't already) and take the exam (it's not too difficult if you know AWS well; focus on Azure-specific services and terminology). Earning Azure Fundamentals will give you the trifecta of cloud basics. Next, if you enrolled in **Google IT Automation with Python**, push to complete it this month – perhaps finish 2 of the remaining courses this week (they cover things like config management with Puppet, which also adds to your DevOps knowledge [26]). By finishing, you'll get the certificate from Coursera/Google to list. Also, if interested in a Google Cloud cert, consider taking the **Google Cloud Digital Leader** exam around this time (since you studied AWS and Azure, a lot of concepts carry over; just brush up on GCP's service names and nuances). This will allow you to mention a GCP cert on your resume.
  Additionally, IBM's certificates: decide if you want to tackle one. If yes, you might not complete it fully while doing everything else, but you could at least start the **IBM Data Science** certificate courses as a "downtime learning" activity to slowly progress. It's lower priority for full-stack, so treat it as optional exploration (perhaps do one course out of 9 each week if you find time, or skip if overloaded).
- **Project #2 Planning:** Time to initiate **Full-Stack Project #2**. Choose another idea that showcases different skills or stack elements than Project #1. For example, if Project #1 was MERN, maybe do Project #2 with **Python/Flask + React**, or **Java/Spring + React**, to diversify. Or if Project #1 was heavy on relational DB, maybe use NoSQL or some new tech in Project #2 (like GraphQL for the API or Redux on front-end if not used before, etc.). Some ideas:
- **Real-time Chat App:** Use Node and Socket.io to build a web chat application (like a mini Slack). Real-time functionality will impress (and you can mention knowledge of WebSocket architecture).
- **Issue Tracker System:** Build a bug tracker (like a simplified JIRA). This could involve more complex relationships (projects, issues, comments, users with roles).
- **Mobile-Friendly PWA:** Build a Progressive Web App that feels like a mobile app (maybe a task manager or habit tracker). Use React with service workers for offline capability. This shows front-end prowess.
- **Data Visualization Dashboard:** If you want to highlight data skills, create a dashboard that visualizes data from some API or dataset (could tie in Python data analysis on the back-end with charts on the front-end).
  Plan out this project and begin development. Try a different approach if possible (for instance, if last time you did everything yourself, maybe this time use a UI framework like Next.js for React, or try a different database). Starting now gives you enough time to finish by early spring.

- **Daily DS & Algo Practice:** Keep solving algorithms regularly. Now focus more on quality and interview simulation: for each problem, after solving, explain your solution aloud as if to an interviewer and analyze its complexity. This will build your communication for technical interviews. Problems count ~5-7 this week, crossing ~95 total.
- **Professional Prep:** It's time to also start thinking about **mock interviews**. Perhaps schedule a mock coding interview on a platform like Pramp or with a friend. Doing one this month will reveal where you need to improve (maybe it's explaining, or coding speed). Use feedback to guide your practice the next weeks. Also, refresh your understanding of behavioral questions – compile a list of common behavioral questions (why do you want to work here, tell me about a challenge, etc.) and outline answers using STAR method. No need to perfect now, but start thinking of examples (most will come from projects or school experiences since you didn't intern).
- **Certifications:** If you haven't already, attempt the **HackerRank Python (Intermediate)** or **JavaScript (Intermediate)** skill tests by now. Also, verify if your freeCodeCamp progress qualifies you for their Full Stack Certification (which one gets after completing multiple certs) – they may have something like "Legacy Full Stack Certification" if you did all front-end + back-end certs. If yes, claim it and add to resume. It might not be widely known, but it encapsulates that you did a lot on freeCodeCamp.

Week 16 (Jan 8–14, 2026) – Project #2 Development & Advanced Topics

- **Focus – Building Project #2:** Continue heavy development on your second major project. By now you should have core functionality underway. Aim to get a working **MVP (Minimum Viable Product)** by end of this week. That means the primary use case of the app is functional end-to-end. For example, if it's a chat app: users can register/login, and send messages in a chat room in real-time. If it's a PWA app: it works offline for basic features. Whatever the main goal, get it in place. Then you'll spend following weeks enhancing and polishing.
  As you code, apply good practices: mobile-responsive design, write unit tests for critical functions, and use Git diligently (maybe try a GitFlow workflow – develop branch, feature branches, etc.). If not already, consider making this project open-source on GitHub and share progress on LinkedIn or Twitter to potentially attract interest (building an audience can indirectly help job prospects).
- **Advanced Topic – GraphQL & Alternative Tech:** This week, try to learn one new tech concept to broaden your knowledge. For example, **GraphQL**: set up a simple GraphQL server (maybe using Apollo Server) and see how it differs from REST. Or learn about **Next.js** (React framework with SSR) by converting a small portion of your app to Next for experimentation. Alternatively, explore a bit of **CI/CD pipeline with Jenkins or GitLab CI** if you only used GitHub Actions so far. The idea is to peek into another tool or approach so you can mention familiarity. Don't invest more than a day or two; just enough to say "I've used X in a toy project" which might spark good conversation in interviews or give you perspective.
- **Daily DS & Algo Practice:** Possibly start timed sets: pick 2-3 medium problems and give yourself 1 hour to solve all (to mimic interview pressure). Also revisit any problems you struggled with historically and ensure you can solve them smoothly now. This week aim for ~7 problems, passing the milestone of 100+ problems solved total. Hitting 100 is great, but keep eyes on 300 goal. Now that interviews may be approaching in a couple months, try to identify any algorithm topics you haven't covered (maybe bit manipulation, or some specific data structure like trie) and do a problem or two there.
- **Behavioral Prep – Stories:** Write out a few **STAR stories** from your projects:
- A challenge you faced in a project and how you solved it (e.g., debugging a tough error, dealing with an unfamiliar technology).

- A leadership or teamwork experience (if you collaborated with anyone or even helped fellow students, use that).
- A time you had to learn something quickly (this whole journey is an example – explain how you tackled this ambitious plan).
- A success you're proud of (maybe deploying your app and getting positive feedback).
  Having these stories written will help you answer questions like "Tell me about a difficult bug you encountered" or "What's a project you're proud of and why?" etc. Practice saying them out loud in a clear, structured way.
- **Networking:** This week, identify target companies you'd like to apply to (maybe mid-size tech companies as per your goal, or some larger ones). Start reaching out: for each company, find a connection or recruiter on LinkedIn, and send a friendly note expressing your interest and asking for advice on their hiring process or if they have upcoming new grad roles. This early outreach can sometimes lead to a referral when you actually apply. Keep it light and professional. Even if you don't get responses, it's good practice.

Week 17 (Jan 15–21, 2026) – Finishing Project #2 & AWS Associate Prep

- **Focus – Complete Project #2:** Aim to have Project #2 feature-complete and in a polishing stage by the end of this week. Finish implementing all intended features. Then do a thorough **testing/debugging pass**: click every button, try to break your app with wrong inputs, fix any bugs. Add any final tests for major components or routes. Ensure security basics: for instance, if your app has authentication, confirm protected routes are truly protected; if it has forms, ensure you validate input on server too. This attention to detail will make your project shine. After that, prepare it for deployment similar to Project #1: containerize if applicable and deploy on a chosen platform. If Project #1 was on AWS, maybe deploy Project #2 on a different platform (like Heroku or Azure) to expand your experience. Or if Project #2 uses a different stack (say Python), try deploying it on a service suitable for that (maybe PythonAnywhere or GCP App Engine). The more deployment exposure, the better you can discuss various environments.
- **Documentation & Presentation:** Write comprehensive documentation for the project. Update the README with clear setup instructions, technologies used, and maybe a short post-mortem of what you learned. Also, prepare a short demo (could be a video or slides) of your project that you can use in interviews or just to showcase. Practicing a 5-minute walkthrough of your project will help in career fairs or interviews when you might be asked about your work.
- **AWS Solutions Architect Associate – Intensive Study:** The AWS SAA exam is tougher than the Cloud Practitioner and is scheduled for March (approx). Use some time this week to intensify study. Go through a popular course or book's curriculum domain by domain. Focus on architecture scenarios: e.g., how to design a highly available web app (know about multi-AZ deployments, load balancers, auto scaling). Understand more services like AWS CloudFormation (infrastructure as code, which you touched via Terraform), AWS CICD tools (CodePipeline, etc.), and advanced database options (Aurora, DynamoDB best practices). Do practice exams; identify weak areas (maybe VPC networking is often tricky – focus there if needed). Plan to allocate a few hours each week to AWS until exam day.
- **Daily DS & Algo Practice:** You're likely quite strong in algorithms by now. Start mixing easy problems for warm-up with real interview-style questions (some can be long or multi-step). Also practice explaining multiple approaches (if you solve something with brute force, can you then improve it and explain that?). This skill is critical in interviews to demonstrate optimization thinking. Solve another ~6-8 problems. If you can push your total solved to ~120 by now, you're in great shape going into the heavy interview season soon.

- **Mock Interviews:** Schedule at least one **mock technical interview** this week or next (if you haven't done one recently). Use a platform or a friend who has strong coding skills to simulate a 45-minute coding interview. Treat it seriously (dress up a bit, communicate as you would in a real one). After, note what you did well and what you need to improve (maybe you forgot to consider edge cases, or you were quiet – work on that). Also, try a **mock system design or behavioral interview** if you can get someone experienced to do that with you. Feedback is gold.
- **Professional Skills:** Reflect on any behavioral feedback from mocks. For example, if you used too much jargon, practice simplifying. Or if you got a behavioral question you hadn't thought of (like a conflict or failure example), prepare an answer for next time. Also, ensure by now you have a solid answer to "Tell me about yourself" that quickly pitches your journey: e.g. "I'm a CS student graduating in May. Over the last 9 months I've undertaken a rigorous self-guided bootcamp where I built full-stack projects using React, Node, Python, and even deployed them to AWS. I've solved over 100 algorithmic problems and earned certifications like AWS Solutions Architect Associate and Meta Front-End. I'm excited to bring this strong foundation and learn even more on the job." – something along those lines, concise and focused on your strengths. Practice delivering it confidently.

Week 18 (Jan 22–28, 2026) – Iterating and Expanding Portfolio

- **Focus – Portfolio & Open Source:** Now with two big projects done, update your **portfolio website** to prominently feature them. Add screenshots, descriptions, and links (GitHub and live demo). Ensure the design looks professional – consider using a template or modern design cues (lots of white space, good typography). If you built it in React earlier, maybe incorporate a new library or even convert it to Next.js for practice, but only if time permits. The goal is that anyone visiting your site immediately sees 8-10 projects with the top 2-3 highlighted and detailed. It should also list your certifications and have a downloadable PDF resume. Make your contact info clear. Possibly start a blog section on your site where you wrote a couple of short articles (you can cross-post anything you wrote on Medium earlier). This shows communication skills.
- **Project #3 (Small) or Open Source:** If you have another project idea in mind or want to dabble in something new, you can start a **small Project #3**. However, at this stage it might be wiser to contribute to **open source** or community projects to show teamwork. Identify a repository on GitHub (maybe something from freeCodeCamp or a library you used) with beginner-friendly issues. Make a contribution (fix a bug or add a small feature, or improve documentation). Getting a merged PR in an open source project is a nice resume item. Even if it's small, it demonstrates collaboration and initiative. If you prefer doing your own project, choose a quick one that adds novelty – e.g., a **browser extension** or a **CLI tool** in Node/Python – something different to diversify your skill set. Keep it limited in scope so it doesn't distract from interview prep.
- **Daily DS & Algo Practice:** Continue daily problems. If you find yourself getting bored, spice it up with new sources – try some CodeSignal or HackerRank contests, or do a timed LeetCode contest to simulate pressure. By now, new problems should often remind you of patterns you've seen – that's good. Also consider focusing on **speed and accuracy**: pick an easy problem and solve it as fast as possible (to simulate those first-round phone screens where they expect a quick solution). But also pick a complex problem and ensure you can go deep into analyzing it (for onsites). Another ~7 problems this week (total ~130).
- **Behavioral Interviews – Practice:** Either with a friend or by yourself, practice answering common behavioral questions out loud. For example, "Tell me about a time you had to learn something quickly" – you can talk about how you self-taught all these technologies. "Describe a project you're proud of" – have a crisp summary of one of your big projects ready. "Tell me about a failure or bug you encountered" – have that story ready. Practicing speaking these will make you more fluent and

reduce nerves later. Try to weave in the STAR format naturally (Situation, Task, Action, Result). Also prepare a couple of thoughtful questions to ask interviewers (e.g., about the company's tech stack or culture), since that usually comes at the end of interviews.

- **Certifications:** If you haven't done so, this is a good time to finalize the **Meta Front-End Developer** certificate if you followed it. The last step in that program is usually a capstone project and a final course on coding interview prep – completing it will give you a credential from Coursera/Meta to list. Also, check if IBM or others offer any quick badges for things you've done (IBM sometimes gives digital badges for completing certain courses). Perhaps try the **IBM Cloud Core** badge (IBM offers a free Cloud Essentials course that gives a badge – not as known as AWS, but anything cloud-related adds to your profile). These are optional extras.

Week 19 (Jan 29 – Feb 4, 2026) – System Design Practice & Interview Drills

- **Focus – System Design Q&A:** Dedicate a couple of sessions to doing mini **system design interviews** with a friend or just by yourself on paper. Take a prompt (say "Design Instagram") and spend an hour writing down how you'd design it, then talk it through. Focus on articulating your assumptions and decisions. If possible, get feedback from an experienced engineer on your approach. Also, review any cheat sheets or key points for system design (e.g., how to calculate storage or bandwidth for given requirements – sometimes interviewers like seeing a bit of back-of-envelope math, like "if we have 1 million users posting X per day…" to gauge scaling). This might be overkill for entry roles, but being prepared won't hurt.
- **Interview Drills:** Each day this week, simulate a specific aspect of interviews:
- One day, do a full 1-hour coding session where you talk through as you code a medium problem.
- Another day, do a session of 2-3 behavioral questions back-to-back, answering as if in an interview panel.
- Another, do a system design prompt for 30 minutes, focusing on high-level only (just the main components).

  Identify any nervous tics or gaps and adjust. Perhaps record yourself to critique clarity and filler words. Aim to be concise but thorough in explanations.
- **Daily DS & Algo Practice:** You might scale back a bit on new problems if you're confident and instead focus on reviewing patterns and ensuring you can solve known common problems quickly. However, if there are some notorious hard problems (like some LeetCode hards that are well-known) you haven't tried, attempt one or two for the challenge – it can make medium ones feel easier. Keep your problem-solving muscles warm with ~5 problems. Total ~135.
- **Project Maintenance:** Use this time to address any final improvements to your projects based on feedback or personal review. Perhaps someone tried your app and suggested a feature – implement a quick version if feasible. Or do a performance pass: check if any API is slow and consider caching or optimization (mention in documentation if you did something like that). These small touches can be interview talking points ("Initially my page load was 3s, so I implemented lazy loading to cut it to 1s"). Also ensure all projects are up-to-date on GitHub with latest code and that all tests pass, etc.
- **Apply for Jobs (start)**: Some companies start posting new-grad roles around this time (if they didn't in fall). Begin searching job boards for "Software Engineer New Grad 2026" or similar. Make a list of positions to apply to, and start applying to a few (especially mid-size companies which may not have super early timelines). Tailor your resume slightly if needed for each (highlight relevant skills if job description mentions specific tech). Even if you feel not 100% ready, getting your application in early is beneficial. Keep track of applications in a spreadsheet. Also continue networking follow-ups: message contacts at companies where you applied to mention your application and interest.

Week 20 (Feb 5–11, 2026) – Interview Season & Continuous Learning

- **Focus – Interviews and Feedback Loop:** By now, you might start getting some interview opportunities (phone screens or coding tests). Dedicate time this week to any such scheduled interviews. Before each, review the company's profile and typical interview format (Glassdoor or forums can help). After each, honestly assess what you did well or not. If you stumbled on a particular algorithm or concept, make note to practice that again. Use any feedback given to improve. This week's focus is to treat each interview as a learning experience to sharpen for the next.
- **Continuous DS Practice:** Don't stop practicing algorithms because interviews have started; continue a maintenance routine. Solve a couple of problems every other day just to keep sharp (especially if interviews are spread out). If you get an interview question you couldn't solve perfectly, go back and master it afterwards.
- **Learning New Stuff (small doses):** Even while interviewing, continue to learn in small ways to show growth. For instance, spend a day exploring **Kubernetes** basics (just read or watch a video) so you can mention that you're aware of container orchestration (ties into DevOps knowledge). Or play with a new JavaScript feature or Python library briefly, to show you stay current. Since AI/ML is hot, you might even take a day to experiment with an **AI API** (like integrate a GPT-3 API into a toy script) – this could be a cool thing to mention if relevant, and gives you insight into AI engineering (plus you have IBM AI engineering cert context if you did that). Balance this so it doesn't distract from main prep, but shows you're still passionate about learning.
- **Portfolio & Resume Updates:** If you achieve something new (say you cracked a hard open source issue or learned a new skill), update your resume/LinkedIn accordingly. By now your resume might have a lot; consider if something can be trimmed to highlight what the specific job values (maybe list fewer projects but more detail on the best ones). Ensure your resume keywords match job descriptions (some use ATS software).
- **Soft Skills – Teamwork Story:** If by chance you haven't actually worked with others yet in this journey, maybe try to do a quick collab now – even a pair programming session with a friend on one of your projects or participating in a short hackathon (there are online hackathons you can join for a weekend). This can give you a real example of teamwork under pressure to talk about. Employers do ask about team experiences; you can draw from class projects if needed, but having one from your self-learning period (like "I collaborated with an online peer to build an open source feature") could be useful.
- **Mock Behavioral Panel:** If possible, have a couple friends conduct a mock behavioral interview as a panel (over video or in person). Sometimes facing 2-3 people asking questions can feel different than one-on-one. This helps reduce anxiety for real on-site (or final round) interviews. They should ask a mix of behavioral questions (including a curveball or two). Practice keeping your cool and answering. Solicit feedback on your clarity and demeanor.
- **Certifications:** This week, consider doing any final certification you planned: e.g., if you prepared for **Google Cloud Associate Engineer** or **Professional Cloud Architect**, you might take it now or soon (though these are tough – only do if you prepared well). Or perhaps the **IBM Data Science** certificate if you actually went through it – you might finish it around now. These will be cherries on top; ensure they don't distract from interview prep though. If an exam is too much now, you can defer it – your core focus is getting the job, certs can be finished even after securing a job offer.

Week 21 (Feb 12–18, 2026) – Job Applications & Strengthening Weak Areas

- **Focus – Application Blitz:** Many new grad roles for Summer 2026 might be in full swing. Dedicate significant time this week to applying to a broad range of companies (if you haven't already applied

widely). Aim to send out applications or networking emails to any remaining targets, including mid-size tech companies, promising startups, and even big companies if they still have openings or if you missed some in the fall. Customize cover letters or emails where appropriate to stand out (mention your portfolio and key projects). Given your strong GitHub and portfolio now, highlight those in applications. This is a numbers game to some extent, so don't be shy to apply to 20-30+ positions.

- **Interview Weak Areas:** By now you have a sense of any areas in interviewing where you feel less confident. Spend this week remedying that. For example:
- If your coding speed in interviews is a bit slow, do daily timed problem drills to speed up.
- If you struggle with dynamic programming, take one day to specifically review classic DP problems (knapsack, longest increasing subsequence, etc.) and the thought process to solve them.
- If behavioral answers feel awkward, refine them and perhaps practice with new people to get comfortable telling your stories naturally.
  Use this week to really turn weaknesses into at least average, and strengths into super-strengths.
- **Daily DS & Algo Practice:** Keep solving or reviewing. You might reduce new problems and instead focus on patterns/techniques: e.g., take one day to write template solutions for common patterns (two-pointer, BFS, DFS, DP, etc.) from memory. Or teach someone else a concept – teaching is a great way to solidify knowledge. If you can, solve a couple new problems still to maintain a growth mindset. Total solved might be ~150 by now.
- **Project Usage and Monitoring:** If your deployed projects have any analytics (like you put Google Analytics or simple logging), see if they're actually getting traffic (maybe you shared them on forums or with friends). If yes, that's cool – you can mention that e.g. "My project X has ~100 daily users" (even if they're mostly peers, it's a metric!). If not, consider showcasing them in relevant communities to get some user feedback (e.g., post about your app on Reddit r/learnprogramming showcase thread or a dev community). Even minor real-world usage can be a differentiator and will give you any last-minute bugs to fix which is a learning experience.
- **DevOps/Cloud Knowledge Refresh:** In case interviews veer into DevOps (some might if they notice your cloud certs), be prepared to discuss what you did. Refresh in your mind the deployments you set up: how Docker works, what CI/CD pipeline you wrote, why you chose AWS services, etc. Also prepare for conceptual questions like "What is CI/CD and why is it useful?" – you can answer referencing your experience setting up GitHub Actions (e.g., "I set up CI/CD for my projects which automatically ran tests and deployed on merge; this ensured integration issues were caught early and made deployment routine [17]."). For cloud, be ready for "What is your experience with cloud?" – you can mention your AWS projects and Cloud Practitioner cert, and maybe an example of using a cloud service like S3 or RDS in your project. These specifics make your answers credible.
- **Confidence and Mindset:** By now, you should feel a significant transformation from where you started. Trust in your preparation. This week, focus on maintaining a positive mindset. Continue healthy routines (sleep well, short exercise breaks, etc.) to keep stress low. Remind yourself that you've built a strong portfolio and skill set – something many senior CS students won't have. This confidence will come through in interviews (without arrogance – stay humble and eager to learn).

Week 22–35 (Feb 19 – May 25, 2026) – Continued Practice, Interviews, and Final Milestones

(Combining the remaining weeks, as they will involve a mix of interviews, wrapping up any remaining goals, and transitioning to job landing.)

- **Interviews & Offers:** During these weeks, you'll likely be in various stages of interviewing. Some weeks may be heavy with coding tests, technical phone screens, or on-site (virtual) interview loops. Keep up your algorithm practice in between to stay sharp. For each interview, prepare by reviewing relevant material (if it's a front-end heavy role, expect some JS/React specific questions; if full-stack, expect anything from coding to maybe system design). Leverage your projects in interviews: often you'll be asked about them – use that chance to demonstrate your knowledge (for example, explain how you designed your project's database or how you set up CI/CD – showing system thinking). As offers (hopefully) come in, evaluate them not just on salary but also learning opportunities (since you want to continue growing to maybe reach those big tech companies in future). Keep applying/interviewing until you secure a role you're happy with.
- **Completing 300+ Problems:** Continue solving problems regularly to reach the **300+ solved problems** goal by May. If you're at ~150 by Feb, that means solving around 50 per month for Mar, Apr, May – about 12 per week, which is doable especially if interview activity slows down by April after you (hopefully) have an offer. Use remaining time to maybe tackle more difficult problems or even competitive programming challenges to further improve. Reaching this milestone is a personal achievement and will absolutely reflect in your coding ability. Also, you can show your LeetCode or HackerRank profile to recruiters if it's impressive (some recruiters do ask for it, as seen in some cases [27]). Having 300+ problems and maybe some contest ranks will wow them.
- **AWS Solutions Architect Associate Cert:** Around March (perhaps Week 24 or 25), take the **AWS SAA exam** as planned. With all the preparation and your hands-on experience, you should pass. This is a big one – list it immediately on resume/LinkedIn. By now you likely have multiple certs (AWS CP, AWS SAA, Azure, Google Python, Meta, etc.) – you might condense them on resume to most relevant, but on LinkedIn you can list all.
- **Graduation Requirements:** Don't forget to keep up with any remaining schoolwork since you're still a student. Ensure you'll pass all courses and graduate on time in May. Perhaps one of your self-driven projects can double as a senior project or independent study for credit, if applicable – that could ease academic load. Balance your time if any finals or academic deadlines come up in April/May.
- **Final Projects & Portfolio Polish:** You now have a robust portfolio. As a final touch, you might do one more small project or improve an existing one in these weeks just to keep your skills fresh and show continuous development. For instance, if you haven't touched machine learning at all, you could do a tiny project like training a simple model in Python (maybe integrate it into one of your apps, like a recommendation feature). That ties in your IBM AI Engineering cert if you completed it. But this is optional; only pursue if you're curious and have time.
- **Soft Skills & Behavioral Mastery:** Leading up to May, practice mock HR interviews focusing on behavioral and fit questions, especially if you target large companies that put weight on these (like Amazon's Leadership Principles – you can practice framing answers using those if applying there). By now you should have a repository of personal experiences to draw from. Keep them fresh in mind.
- **Networking & Job Fair:** If your college has a spring job fair or if there are local tech meetups, attend them. Use your portfolio and GitHub as your talking points. By being able to discuss your 8-10 projects passionately and show them on your website, you will stand out even without internships.

Many students with internships don't have such extensive personal projects, so you'll be a unique candidate. Emphasize the consistency of your GitHub contributions and your ability to self-learn – companies value engineers who can pick up new skills quickly (and you're now evidence of that).

- **Resume Finalization:** Update your resume one last time in May with any new achievements (e.g., "Solved 300+ LeetCode problems" – you can put that under a achievements section, as some recruiters actually appreciate seeing that metric [27]). Ensure all information (GPA if you list it, graduation date) is correct as you graduate. Have a peer or mentor review it for any last improvements.
- **Prepare for Transition:** In the final weeks of this plan (late April/May), hopefully you have secured a job offer or have some in pipeline. Use time to prepare for that transition: if you got a job, see if they use any tech you're less familiar with and study that to get a head start. If you're still interviewing in May, intensify efforts with whatever companies remain and leverage your now nearly graduate status (some companies only consider you once you're very close to graduation). You can also consider short-term freelance or internship in summer as a fallback – but given your preparation, a full-time offer is likely.
- **Confidence for Interviews:** By end of this journey, you should feel **confident in both technical and behavioral interviews**. You'll have a strong GitHub (daily contributions, numerous repos), a polished portfolio with deployed projects, a resume packed with skills and certs, and hundreds of hours of coding practice under your belt. This confidence will help you tackle interviews calmly. Remember to also prepare for common behavioral questions like teamwork and conflict resolution – draw from any group work at school or open source collabs. But also leverage your self-directed project work to show you're self-motivated and passionate.

Week 36 (May 26–31, 2026) – Wrap-Up and Next Steps

- **Goal Check:** It's May 2026, time to evaluate outcomes. By this point, you should have:
- **Robust Knowledge Base:** You've learned full-stack web development thoroughly – front-end (HTML, CSS, JS/TS, React), back-end (Node/Express, plus Python/Flask and some Java/Spring), databases (SQL and MongoDB), cloud (AWS deeply, plus Azure/GCP basics), DevOps (Docker, CI/CD, Terraform basics). This comprehensive list matches the plan we outlined at the start – nothing forgotten. You can confidently claim skills in all these areas on your resume now.
- **Daily Practice Habit:** You solved 300+ algorithm problems, touching every major topic [3]. This not only prepped you for interviews but also ingrained problem-solving habits that will benefit you on the job.
- **Certifications Achieved:** Ideally, you now hold multiple certs: AWS Cloud Practitioner & Solutions Architect Associate [10] [11], Azure Fundamentals, Google IT Automation with Python, perhaps a Google Cloud cert, Meta Front-End, several freeCodeCamp certs, and maybe IBM Data Science/AI if you pursued them. These are listed on your resume/LinkedIn and lend credibility to your self-taught expertise.
- **Portfolio of Projects:** You have 8–10 polished, deployed projects. At least 2 of them are substantial full-stack applications you can demo and discuss in depth, showing off different aspects of your skill set. The others (small apps, certification projects, etc.) round out your experience (covering things like front-end design, API microservices, data visualization, etc.). This portfolio is showcased on your personal website which itself is an example of your work.
- **Strong Online Presence:** Your GitHub profile shows continuous activity, collaboration, and quality code. You possibly have contributions to open source. Your LinkedIn is populated with certs and projects, and you've built a network of contacts. There may also be HackerRank/LinkedIn skill badges

on your profile verifying your coding abilities. All of this combined makes you *stand out to recruiters*, even without internship experience, because it demonstrates initiative and real-world skills.

- **Employment Outcome:** Ideally, by end of May you have landed a job offer as a full-stack or software engineer at a mid-size or large tech company – fulfilling your goal. If not yet, you are very close and equipped with everything needed to land one soon. Keep applying and interviewing; sometimes hiring cycles vary, but rest assured your preparation will pay off.

- **Graduation and Beyond:** As you graduate, finalize your resume with your B.Sc. and any Latin honors or notable awards. Then it's about starting your career. Remember that learning never really ends – you've built a strong foundation in 9 months, but on the job you'll continue growing. Your familiarity with DevOps, system design, and broad tech stack means you can quickly adapt to whatever tech your new team uses. Also, you can leverage your projects in the job – perhaps you can suggest solutions you learned or even share code if applicable. You should also continue algorithm practice occasionally to be ready for future interviews (like 1-2 years down the line if aiming for bigger companies).

- **Confidence in Role:** When you start your full-time job, imposter syndrome might creep in given you took an unconventional path (no internships). But you can be confident because you effectively gave yourself a comprehensive bootcamp and more [28] [21]. You likely have more practical experience than many fresh grads who only did class projects. Trust in your training; ask questions and be a team player, and you'll do great. Your portfolio and certs got you in the door; now your continuous learning habit will help you excel in your role, and those behavioral skills (communication, adaptability [18] [29]) you developed will help you integrate well into the team.

- **Lifelong Learning:** Finally, set some post-May goals. Perhaps aim to tackle that Kubernetes or advanced DevOps fully, or contribute regularly to open source, or build an even larger personal project using cutting-edge tech (maybe an AI-driven app) to keep your skills sharp. With the strong base you have, you can pivot or learn any new technology faster now. Keep up with industry trends, maybe pursue that IBM AI Engineering certificate over the summer if AI interests you, or start preparing for higher-level certs (AWS professional or specialty) as a next milestone. The key is to never become stagnant. Given how far you came in 9 months, imagine where you could be after a year on the job plus continued self-learning – perhaps ready for interviews at top-tier companies if that remains a goal.

In summary, by adhering to this structured plan, you've transformed from feeling underprepared to becoming a **job-ready full-stack developer** with a standout profile. You learned every key skill area – from programming basics to advanced cloud deployment – and reinforced them with extensive practice and projects. You acquired numerous certifications that validate your knowledge officially, and you built a rich portfolio that proves your abilities to any recruiter or hiring manager. You also honed soft skills like communication, problem-solving, and self-management along the way, which employers highly value [18] [30]. Now, as May 2026 arrives, you can confidently step into the professional world, **ready to ace technical interviews and thrive in a full-time software engineering role**. Good luck, and congratulations on how far you've come!

---

1 2 8 14 28 From Zero to Full Stack: A Complete Developer’s Roadmap... | by Emmanuel Cobbinah | Medium
https://medium.com/@ecobbinahbuz/from-zero-to-full-stack-a-complete-developers-roadmap-937b75eaa182

3 4 5 I solved 1583 LeetCode problems. But you only need these 300. | by Ashish Pratap Singh | Medium
https://medium.com/@ashishps/i-solved-1583-leetcode-problems-but-you-only-need-these-300-db17e9297e00

6 27 Google recruiter asked for Leetcode profile and told me to complete 200-300 mid-hard level problems | Tech Industry - Blind
https://www.teamblind.com/post/google-recruiter-asked-for-leetcode-profile-and-told-me-to-complete-200-300-mid-hard-level-problems-mqeyhdry

7 9 Full Stack Developer Roadmap [2025 Updated] - GeeksforGeeks
https://www.geeksforgeeks.org/blogs/full-stack-developer-roadmap/

10 11 23 24 25 Steps to start your AWS Certification journey | AWS Training and Certification Blog
https://aws.amazon.com/blogs/training-and-certification/steps-to-start-your-aws-certification-journey/

12 13 26 Google IT Automation with Python Professional Certificate | Coursera
https://www.coursera.org/professional-certificates/google-it-automation

15 16 System design interview guide for Software Engineers | Tech Interview Handbook
https://www.techinterviewhandbook.org/system-design/

17 Why Every Full-Stack Developer Should Learn DevOps (I Learned ...
https://levelup.gitconnected.com/why-every-full-stack-developer-should-learn-devops-i-learned-the-hard-way-2e822ac9fc27

18 29 30 Tech Hiring Trends 2025: Why Soft Skills Matter More Than Ever
https://www.cogentuniversity.com/post/tech-hiring-trends-2025-why-soft-skills-matter-more-than-ever

19 20 21 What do I need to learn to become a fullstack developer/engenieer? - Career Advice - The freeCodeCamp Forum
https://forum.freecodecamp.org/t/what-do-i-need-to-learn-to-become-a-fullstack-developer-engenieer/694830

22 My Review of freeCodeCamp's Responsive Web Design Projects
https://dev.to/colinintj/my-freecodecamp-responsive-web-design-projects-1893