ChatGPT

6-Month Full-Stack Developer Study Plan (React to Java Backend Transition)

Note: Each weekday offers ~2 hours for new learning (evenings), and each weekend day ~3 hours. *Every morning*, spend **30 minutes** reviewing past material (flashcards, notes, or revisiting yesterday's code). Regular review will reinforce your knowledge. We've also built in **catch-up/rest days** (typically Sundays every few weeks) – use these to consolidate learning or recharge. By the end of 6 months, you'll have a solid Java/Spring Boot backend foundation, an updated React skillset, a completed full-stack project (deployed on AWS), and be prepared to apply to jobs. The schedule is organized by week (with date ranges) and lists daily tasks with recommended resources.

Month 1 (June 2025): Solidify Java Fundamentals

Focus on core Java programming skills and object-oriented programming (OOP). A strong grasp of Java basics is essential for Spring Boot development [1]. You'll also start brushing up on data structures to prepare for later coding challenges [2].

Week 1 (June 2 – June 8, 2025) – *Java Basics & Syntax*

- **Day 1 (Mon, June 2)** – **(2h)** *Environment Setup & "Hello World"*:
- Install the latest **JDK (Java Development Kit)** and set up an IDE like IntelliJ IDEA or Eclipse. Ensure Java is added to your PATH.
- Write a simple **"Hello World"** program to verify the setup. Run it from the IDE and the command line.
- *Resource:* Oracle's official Java Getting Started guide (sections on installation & basic syntax) [3].
- **Day 2 (Tue, June 3)** – **(2h)** *Language Fundamentals – Variables & Types*:
- Learn Java **primitive types** (int, float, boolean, etc.) and **operators**. Practice by declaring variables and performing basic calculations (e.g., compute a circle's area).
- Read about Java's **control flow**: `if/else` conditions and `switch` statements. Write a program using conditional logic (e.g., a number guessing hint).
- *Resources:* Oracle Java Tutorial on language basics (data types, operators) and control statements [1].
- **Day 3 (Wed, June 4)** – **(2h)** *Control Structures – Loops*:
- Master **loops** (`for`, `while`, `do-while`). Write small snippets: e.g., print multiplication tables using a for-loop, or sum an array of numbers.
- Learn to use **input/output** (e.g., `Scanner` for console input). Prompt the user for a number and output a computed result (to practice I/O and loops together).
- *Resource:* "Java Tutorials – Control Flow Statements" on Oracle Docs (covers loops and conditionals) [3].
- **Day 4 (Thu, June 5)** – **(2h)** *Functions and Arrays*:
- Understand **methods** in Java: define simple functions (e.g., a method to add two numbers, and one to find max of an array). Learn about method parameters, return types, and method overloading.
- Introduce **arrays** – create arrays, iterate over them, and perform operations (e.g., find min/max, reverse an array). This builds a foundation for data structure manipulation [2].
- *Resource:* W3Schools Java Tutorial on Methods and Arrays (for examples and exercises).

- **Day 5 (Fri, June 6)** – **(2h)** *Intro to Object-Oriented Programming (OOP)*:
- **Classes & Objects:** Learn how to define a class in Java (fields, methods) and instantiate objects. Understand the concept of a “blueprint” class vs. actual object instance [4] [5].
- Write a simple class (e.g., `Person` with attributes `name`, `age` and a method `introduce()`). Create instances and call methods to ensure you grasp object usage.
- *Resource:* Oracle Java Tutorial – *Classes and Objects* (explains class structure and object instantiation) [5].
- **Day 6 (Sat, June 7)** – **(3h)** *OOP Deeper – Encapsulation & Constructors*:
- **Encapsulation:** Learn about access modifiers (`private`, `public`) and why they’re used to hide internal state. Practice by making class fields private and adding getter/setter methods.
- Implement **constructors** to initialize objects easily. Overload a constructor in your `Person` class (e.g., one constructor with name+age, another default constructor).
- If time permits, explore the `this` keyword and how it refers to the current object instance.
- *Resource:* Tutorial on Java OOP Concepts (encapsulation and constructors) [3].
- **Day 7 (Sun, June 8)** – **(3h)** *Inheritance & Polymorphism*:
- **Inheritance:** Create a subclass (e.g., `Employee` extends `Person`). Learn how subclasses inherit fields/methods and how to use `super`. Add a new field to `Employee` (like `salary`) and a method overriding `introduce()` to include job info.
- **Polymorphism:** Practice treating an `Employee` object as a `Person` (demonstrating polymorphism). Read about dynamic method dispatch (how overridden methods are chosen at runtime).
- Get familiar with **abstract classes** and **interfaces**: read about their role in Java design (you will use these in frameworks later).
- *Resource:* GeeksforGeeks article on Inheritance in Java (with examples of subclasses and method overriding) [6].

Week 2 (June 9 – June 15, 2025) – *Java Collections & Core APIs*

- **Day 8 (Mon, June 9)** – **(2h)** *Java Collections Framework – Lists*:
- Start with **ArrayList** (part of Java’s Collections). Learn how to create lists, add/remove elements, iterate, and use common methods (`size()`, `contains()`, etc.).
- Implement a simple program using a list (e.g., maintain a to-do list or list of numbers and sort them). Understand why collections are preferable to basic arrays for dynamic data.
- *Resource:* Official Java Collections tutorial (focus on List interface and ArrayList usage) [2].
- **Day 9 (Tue, June 10)** – **(2h)** *Collections – Sets and Maps*:
- Learn about **HashSet** (stores unique elements) and **HashMap** (key-value pairs). Understand use-cases: sets for uniqueness, maps for lookups by key.
- Practice: use a HashSet to filter duplicates from a list of strings. Use a HashMap to count frequencies of words in an array (simulate a word count).
- *Resource:* Baeldung’s guide on Java Collections Framework (sections on Set and Map).
- **Day 10 (Wed, June 11)** – **(2h)** *Exception Handling & Debugging*:
- Understand Java’s **exception hierarchy**. Learn to use `try-catch` blocks and `finally`. Differentiate between checked vs unchecked exceptions.
- Write code that deliberately throws an exception (e.g., divide by zero or array out-of-bounds) and handle it gracefully. Also practice using the debugger in your IDE to step through code and inspect variables.
- *Resource:* Oracle tutorial on Exceptions (covers try/catch, custom exceptions).
- **Day 11 (Thu, June 12)** – **(2h)** *Java 8+ Features – Lambdas & Streams*:
- Get introduced to **Lambda expressions** and the concept of functional programming in Java. Write a simple lambda for a comparator or use Java’s `Runnable` with a lambda.

- Learn the **Streams API** for processing collections. Practice filtering, mapping, and reducing a list of objects (e.g., given a list of `Person`, stream it to filter by age and collect names).
- *Resource:* Tutorial or blog on Java 8 Streams and Lambdas (e.g., Oracle's Lambda Expressions guide).
- **Day 12 (Fri, June 13)** – **(2h)** *Multithreading Basics*:
- Learn what **threads** are in Java. Create a basic thread by extending `Thread` or implementing `Runnable`. Start a couple of threads and see how they run concurrently.
- Understand synchronization issues at a high level (no need to dive too deep). For example, if two threads increment a counter without synchronization, what problems might arise?
- *Resource:* Java Brains video or article on Java Multithreading basics (for beginners).
- **Day 13 (Sat, June 14)** – **(3h)** *Mini-Project – Console Application*:
- Build a small **console-based project** to apply Week 1–2 learnings. Ideas: a **simple calculator**, a **task manager**, or a **library book manager** (with options to add, list, delete items).
- Use OOP design: e.g., for a library manager, create `Book` and `Library` classes. Use collections to store books, and methods to add/remove/search.
- Handle user input in a loop (text-menu interface) and include basic error handling (e.g., if user tries to remove a non-existent item, catch the error).
- This project will reinforce Java syntax, OOP, collections, and flow control in an integrated way [7].
- *Resource:* "Java Project: Console CRUD Example" (a blog or video demonstrating a simple console CRUD app).
- **Day 14 (Sun, June 15)** – **(3h)** *Review and Data Structures Refresher*:
- **Review:** Go over all topics from Weeks 1–2 (OOP concepts, collections, etc.). Ensure you remember key concepts like inheritance vs interface, list vs array differences, exception handling patterns.
- **Data Structures & Algorithms:** As a prep for coding interviews, spend some time on basic algorithms. Try 1-2 easy problems on HackerRank/LeetCode (e.g., array sum, string reversal) to practice using Java for problem-solving. Focus on using the correct data structures (list vs set, etc.) for each problem [2].
- If you felt Week 1–2 was rushed, use today to **catch up** on any incomplete exercises or revisit tough topics. If you're on track, take a **rest** or explore optional topics (e.g., read about Java Collections source implementation out of curiosity).

Week 3 (June 16 – June 22, 2025) – *Advanced Java & Prep for Spring*

- **Day 15 (Mon, June 16)** – **(2h)** *Generics and Advanced Java APIs*:
- Learn about **Generics** in Java (e.g., `List<T>`): why they exist and how to create generic classes or methods. Refactor a collection usage in your mini-project to use generics properly (if you used raw types).
- Explore the Java **Collections API deeper**: iterators, `Collections.sort()`, and utility classes like `Collections` or `Arrays`. Practice sorting a list of custom objects (implement `Comparable` on your `Person` or `Book` class).
- *Resource:* Java Tutorials – Generics (covers generic methods and classes).
- **Day 16 (Tue, June 17)** – **(2h)** *Build Tools – Maven/Gradle*:
- Introduce yourself to **Maven** (a build automation tool commonly used in Java projects). Learn about `pom.xml`, dependencies, and the Maven lifecycle.
- Convert your console mini-project into a Maven project (or start a new simple project with Maven) to understand how to manage dependencies. For example, add a logging library (like Log4j) as a dependency and use it for logging messages.
- If time permits, also read briefly about **Gradle** (another build tool). Focus on understanding that these tools help in compiling, packaging, and dependency management.

- *Resource:* Maven Quick Start Tutorial (on Maven official site or Baeldung) for setting up a simple Maven project.
- **Day 17 (Wed, June 18)** – **(2h)** *Intro to Spring Framework Concepts*:
- Before diving into Spring Boot, grasp core **Spring Framework** ideas: **Inversion of Control (IoC)** and **Dependency Injection (DI)**. Read how Spring manages objects (beans) for you, rather than instantiating with `new` [8].
- Understand what a **Bean** is in Spring, and the lifecycle of a Spring bean (creation, initialization, destruction) [8].
- Read about **ApplicationContext** vs BeanFactory, and how Spring configuration can be done via XML or annotations. (No coding yet, just conceptually prepare for Spring Boot's magic by learning the underlying Spring basics [9].)
- *Resource:* GeeksforGeeks – *Spring Core Concepts* (covers IoC, DI basics) [8].
- **Day 18 (Thu, June 19)** – **(2h)** *Web Fundamentals – HTTP & REST*:
- Review how web communication works: the **HTTP protocol** (methods like GET, POST, etc., status codes, and JSON format for data). This is crucial for understanding RESTful APIs.
- Learn about **RESTful API design** principles: resources, endpoints, using HTTP verbs properly (GET for read, POST for create, etc.), and statelessness [10].
- Use a tool like Postman or cURL on a sample public API (e.g., a public "JSON placeholder" API) to practice making GET/POST requests and inspecting responses.
- *Resource:* REST API design tutorial (REST basics and best practices) – e.g., "REST API Tutorial" on restapitutorial.com [10].
- **Day 19 (Fri, June 20)** – **(2h)** *Spring Boot Introduction*:
- **What is Spring Boot?** Read about how Spring Boot builds on Spring to simplify development (auto-configuration, embedded server, opinionated defaults) [11]. Note that many companies use Spring Boot for its efficiency and production-ready setup [11].
- Go through a "Hello World" Spring Boot example: use **Spring Initializr** (start.spring.io) to generate a simple project (with dependencies for web). Understand the project structure (especially the `Application` class with `@SpringBootApplication`).
- Run the Spring Boot app (it comes with an embedded Tomcat server). Verify it starts up on, say, `localhost:8080` (though no functionality yet).
- *Resource:* Official Spring Boot Quick Start guide (Spring.io "Building a RESTful Web Service" guide) – covers initial setup and running a simple app.
- **Day 20 (Sat, June 21)** – **(3h)** *Spring MVC and First Controller*:
- Learn the basics of **Spring MVC** in Spring Boot: how incoming HTTP requests are handled by controller classes.
- Create your first **Controller** class with Spring annotations: e.g., `@RestController` and define a GET endpoint like `/hello` that returns a "Hello, World" string or JSON. Use `@GetMapping`, `@PostMapping` etc. for mapping URLs to methods.
- Understand the role of **Model** (data) and **Controller** (request handling) in the MVC pattern [12]. Although we won't use a View here (for REST APIs, the "view" is usually the JSON output), it's good to know the concept.
- Test your endpoint using Postman or a web browser to ensure it returns the expected output.
- *Resource:* Spring Boot official guide – *Serving Web Content* (to understand controllers and mappings) [12].
- **Day 21 (Sun, June 22)** – **(3h)** *Catch-Up & OOP Review in Practice*:
- **Catch-Up Day:** You've covered a lot in Java and just started Spring Boot. Use today to **review** and **reinforce**. Revisit any Java concepts that were tricky (e.g., lambdas or generics) and ensure you can relate them to what's coming (for example, understanding collections will help with handling data in APIs).
- If you fell behind on setting up Spring Boot or writing the controller, this is a buffer day to complete that. Make sure your development environment for Spring Boot is fully working.

- **Optional:** If on track, read a bit more on Spring Boot features that intrigue you. For instance, look at what **@SpringBootApplication** annotation does under the hood, or how Spring Boot auto-configuration works (just skim for now).
- *Resource:* Personal notes and Java flashcards for review, plus Spring Boot docs on core features for curiosity [13] [14]. (And take a breather if needed – burnout helps no one!)

Month 2 (July 2025): Spring Boot and Databases

In Month 2, you'll dive into building a **RESTful API** with Spring Boot, integrating a database for persistence. Focus on understanding how Spring Boot simplifies common tasks (like setting up web servers and connecting to databases) and how to design clean REST endpoints. We'll also update some front-end React knowledge later this month to keep your full-stack skills sharp.

Week 4 (June 23 – June 29, 2025) – *Building CRUD APIs with Spring Boot*

- **Day 22 (Mon, June 23)** – **(2h)** *Spring Boot CRUD – Project Setup*:
- Initialize a Spring Boot **CRUD project** (e.g., a simple "Task Manager" API). Use Spring Initializr to include **Spring Web** dependency (for REST controllers).
- Plan the data model: e.g., a `Task` entity with fields like id, title, description, status. (For now, we'll use a simple in-memory list to store tasks before integrating a database.)
- Create a **Controller** class for `Task` with basic endpoints: `getAllTasks`, `getTaskById`, `createTask`, `updateTask`, `deleteTask`. Just define method stubs returning dummy data or simple messages for now.
- *Resource:* Spring.io guide "Building a RESTful Web Service" – up to creating a controller and some dummy endpoints.
- **Day 23 (Tue, June 24)** – **(2h)** *Spring Boot CRUD – Implementing Endpoints*:
- Implement the logic for each CRUD endpoint in the `TaskController`. Use a static list or `ArrayList<Task>` as a fake database to store tasks in memory.
- For example, implement `getAllTasks()` to return the list, `createTask()` to add a new Task to the list (assigning an ID), `updateTask()` to modify an existing Task, etc. Manage edge cases (e.g., if `getTaskById` doesn't find the task, return a 404 status).
- Test these endpoints using Postman: perform GET, POST, PUT, DELETE requests and ensure the expected behavior (simulate a client interacting with your API).
- *Resource:* Baeldung's tutorial on building a simple Spring Boot REST API (for reference on in-memory CRUD logic).
- **Day 24 (Wed, June 25)** – **(2h)** *Spring Boot Layers – Service & Repository*:
- Refactor the controller: introduce a **Service layer**. Create a `TaskService` class that contains the business logic (e.g. methods like `findAllTasks()`, `addTask()`, etc.). Move the in-memory data (the task list) and CRUD logic into this service.
- Modify `TaskController` to **inject** `TaskService` (using Spring's `@Autowired` or constructor injection) and call service methods instead of directly manipulating the list. This layered approach follows best practices [15], making your code cleaner and easier to maintain.
- Verify everything still works via testing endpoints. The behavior shouldn't change, but your code structure is now more realistic for enterprise apps (Controller for routing, Service for logic, data store separate).
- *Resource:* Medium article's mention of layered architecture (Controller-Service-Repository-Model) in Spring Boot [15].
- **Day 25 (Thu, June 26)** – **(2h)** *Input Validation & Error Handling*:
- Implement basic **validation** for your API inputs. For example, ensure `Task` title is not empty when creating a task. Use Spring's **Bean Validation** (JSR 380) by adding `@Valid` and constraints like `@NotNull`, `@Size` on the `Task` fields.

- Add an **exception handler** (using `@ControllerAdvice` or exception handler methods) to catch cases like "Task not found" (throw a custom `TaskNotFoundException` in service and handle it to return a meaningful HTTP 404 with a JSON error message).
- This will make your API more robust and closer to production style error handling.
- *Resource:* Baeldung – *Spring Boot @Valid and Exception Handling* (shows how to use validation annotations and write exception handlers).
- **Day 26 (Fri, June 27)** – **(2h)** *Intro to Relational Databases (SQL)*:
- Shift focus to **databases**: understand why we need a database instead of an in-memory list (data persistence across restarts, more data than memory, etc.).
- Learn database fundamentals: what is **SQL**, what are **tables, rows, columns**. Understand primary keys and foreign keys, and basics of database design (a brief intro to normalization).
- Set up a local database: install **MySQL** or **PostgreSQL** (choose one, e.g., MySQL for ease). Alternatively, use an embedded database (**H2**) initially for simplicity.
- Using your chosen DB, create a simple schema and table manually (e.g., a `tasks` table with columns id, title, description, etc.). This can be via a GUI client (like MySQL Workbench) or command-line SQL.
- *Resource:* W3Schools SQL Tutorial (covers basic SQL commands) – focus on CREATE TABLE and simple SELECT/INSERT queries [16] [17] .
- **Day 27 (Sat, June 28)** – **(3h)** *Spring Data JPA – Integrating Database*:
- Introduce **Spring Data JPA** to your Spring Boot project to replace the in-memory data store with a real database. Add the **Spring Data JPA** and **MySQL/PostgreSQL driver** dependencies to your project (update `pom.xml`).
- Configure database connection in `application.properties` (JDBC URL, username, password for your local database).
- Create a `Task` **entity** class with JPA annotations (`@Entity`, `@Id`, etc.) matching your database table structure.
- Create a **Repository** interface e.g. `TaskRepository` that extends `JpaRepository<Task, Long>`. Spring Data will provide implementations for common CRUD methods automatically.
- Update your Service to use `TaskRepository` for data operations. For example, use `taskRepository.findAll()` instead of the in-memory list.
- Run the application – on startup, Spring Boot (with JPA and H2/MySQL) can auto-create the table based on your entity if configured (`spring.jpa.hibernate.ddl-auto`). Verify that CRUD operations now affect the database (e.g., creating a task inserts a row in the `tasks` table).
- *Resource:* Spring.io guide "Accessing Data with JPA" – shows setting up an entity and repository with Spring Boot [18] [19] .
- **Day 28 (Sun, June 29)** – **(3h)** *Database CRUD & SQL Practice*:
- Test all your CRUD endpoints again, now backed by the database. Use a DB viewer or `SELECT` queries to confirm data is actually being created/updated/deleted in the database.
- Practice some **SQL queries** directly on the DB to reinforce SQL skills: e.g., manually `INSERT` a record, `SELECT` it via the database client, then try retrieving it through your API to see the end-to-end connection.
- If comfortable, experiment with slightly advanced JPA: e.g., write a custom finder method in `TaskRepository` (like `List<Task> findByStatus(String status)` and see it work without implementation due to Spring Data query creation).
- **Review & Catch-up:** This week you connected a lot of dots (Spring Boot, JPA, DB). Take time to ensure you understand how each layer works together. If you fell behind (perhaps the database setup or JPA was tricky), use this day to solidify those steps.
- *Resource:* "Don't be lazy with databases — the more you understand them, the better your app design and performance" [20] . Re-read any relevant Spring Data JPA docs or do an extra SQL exercise if needed to feel confident.

Week 5 (June 30 – July 6, 2025) – *Back-End Augmentation & Modern React Refresher*

Now that you have a basic Java REST API with a database, we'll enrich your backend skills with important features (like authentication) **and** ensure your React knowledge is up-to-date. This week splits focus: first half on backend (authentication), second half on frontend (modern React practices).

- **Day 29 (Mon, June 30)** – **(2h)** *Intro to Authentication – Basics of Spring Security*:
- Learn why **authentication and authorization** are needed: most apps require login and restricted access. Read about Spring Security – a framework to secure Spring applications (it can handle user login, password encoding, role-based access control, etc.) [21] .
- Start simple: enable Spring Security in your app by adding the dependency. By default, Spring Boot will lock down all endpoints (with a generated password). Test this default behavior: try accessing your API endpoints and observe the 401 responses.
- Understand the concept of **Security Filter Chain** – you don't need full detail yet, but know that Spring Security sits in front of your controllers to intercept requests.
- *Resource:* Baeldung – *Spring Security Introduction* (overview of how Spring Security integrates with Boot and default behaviors) [21] .
- **Day 30 (Tue, July 1)** – **(2h)** *Spring Security – Customizing Authentication*:
- Configure Spring Security to use a custom user detail. For now, use an in-memory user store: define a username and password in `application.properties` or configure a `WebSecurityConfigurerAdapter` (if using Spring Security 5.x) to set up an in-memory user with roles.
- Adjust your security settings to permit public access to certain endpoints if needed (e.g., if you have a health check or status endpoint) and secure others. For example, require authentication for all CRUD endpoints of your API.
- Test with Postman: when secured, calls should require a Basic Auth header (or whatever authentication mechanism you set). Verify that with correct credentials you can access, and with none or wrong credentials you get 401.
- *Resource:* Official Spring Security documentation – "Hello Security" guide (for setting up a simple in-memory user authentication) [22] .
- **Day 31 (Wed, July 2)** – **(2h)** *JWT and Modern Auth (Conceptual)*:
- Learn about **JWT (JSON Web Tokens)** – a popular stateless authentication method for APIs (especially for securing RESTful services in a distributed system). Even if not fully implementing, understand the flow: client logs in with credentials -> server returns a JWT -> client sends JWT on each request -> server validates it.
- If time permits, attempt integrating JWT in your Spring Boot app: use Spring Security along with a library or configure manually to generate tokens on a login endpoint and require token auth for other endpoints. (This is advanced; if it's too time-consuming, just follow a tutorial and understand the code rather than writing from scratch.)
- Alternatively, read about **OAuth2** vs JWT, just to be aware of concepts (many companies use OAuth2 for social logins, etc., but JWT is simpler for custom implementations).
- *Resource:* "Spring Boot JWT Authentication Tutorial" (e.g., by Tech Primers or Baeldung) for a step-by-step on adding JWT auth.
- **Day 32 (Thu, July 3)** – **(2h)** *React Modern Practices – Hooks & Functional Components*:
- Switch to **frontend refresh**. React has likely evolved since you started; ensure you're comfortable with **functional components** and **Hooks** (useState, useEffect, etc.) if you haven't been using them exclusively. (As of React v16+ hooks are the standard for managing state instead of class components).
- Take a small old React component you remember (maybe a class-based component from older code) and refactor it into a functional component with hooks. For example, a component that loads a list of items could use `useState` for data and `useEffect` for fetching on mount.

- Read about the **Context API** for state that needs to be global (to reduce reliance on Redux for simpler apps). If you haven't used Context, practice by creating a simple context (e.g., a Theme context that provides dark/light mode value to components).
- *Resource:* Official React docs – *"Hooks at a Glance"* and *"Using the Context API"* (covers useState/useEffect basics and context usage).
- **Day 33 (Fri, July 4)** – **(2h)** *React Advanced Patterns & TypeScript*:
- Learn about additional React patterns: **React Router** (if you haven't used v6, see what changed), **code-splitting** (lazy loading components), and maybe a touch of performance optimization (using `React.memo` or the concept of virtualized lists if dealing with many items).
- Introduce **TypeScript** in React if you haven't used it: understand how TypeScript can catch errors and make refactoring easier. Set up a small React component in a TypeScript file (e.g., define an interface for props and use it). If your final project might use TypeScript, this is important practice.
- Ensure you can create custom types and use React's types (like `FC` for functional components, etc.).
- *Resource:* React + TypeScript cheatsheet (a well-known GitHub repository with tips on using TS in React).
- **Day 34 (Sat, July 5)** – **(3h)** *Mini React Project – Connect to API*:
- Spin up a new **React app** (you can use Create React App or Vite) to practice integrating with a backend. This will also prepare you for your full-stack final project.
- Implement a simple UI for the Task Manager API you built: e.g., a page to list tasks, a form to add a task, and the ability to mark as done or delete. Use **fetch or Axios** to call your Spring Boot API (running locally). Ensure you handle CORS: enable CORS in your Spring Boot app for development (using `@CrossOrigin` on the controller or a global CORS config) so your React front-end can call it.
- This exercise will highlight how front-end and back-end connect (HTTP calls, JSON data handling on both ends). It will also let you practice React Hooks in a real scenario (e.g., useEffect to fetch tasks on component mount, useState to store tasks).
- Bonus: If your Spring Boot security (from earlier this week) is enabled, you'll need to handle auth in the React app (maybe use basic auth for now or disable security while testing). For now, it's okay to test with security off or a dummy easy setup.
- *Resource:* "React CRUD app with Spring Boot REST" – many tutorials exist (for example, one on dev.to or YouTube) showing how to connect a React frontend to a Spring Boot backend. Use those as reference if needed.
- **Day 35 (Sun, July 6)** – **(3h)** *Catch-Up & Frontend/Backend Sync*:
- Today is a **catch-up/rest day** after an intense week. Spend time to **synchronize your knowledge**: now that you've touched both backend and frontend sides, consider how they work together.
- **Review:** Go over Spring Security and JWT concepts – these can be challenging; ensure you at least conceptually get how authentication will fit into a full-stack app. Also review any React TypeScript issues you encountered.
- If you fell behind on the React refresh or couldn't complete the mini React frontend, use today to wrap that up. Having a basic full-stack mini-demo (Tasks API + React UI) even in rough form is great progress.
- **Rest:** If you are up-to-date, take a break or explore something fun like a new React library (just for curiosity) or glance at Next.js (optional). Re-energize for next week.

Month 3 (August 2025): Advanced Backend Topics & Microservices Concepts

By now you have a working knowledge of building a monolithic Spring Boot application with a database and a basic front-end. This month, focus on broadening your backend expertise: learn about **microservices architecture**, incorporate a NoSQL database (MongoDB), and containerize your applications with **Docker**. These skills will make you versatile in handling modern backend challenges. We'll also plan the **capstone full-stack project** you'll build in Month 4–5.

Week 6 (July 7 – July 13, 2025) – *Microservices Theory & Project Planning*

- **Day 36 (Mon, July 7)** – **(2h)** *Understanding Microservices Architecture*:
- Learn what **microservices** are and how they differ from a monolith. Key idea: breaking a large application into small, independent services with specific responsibilities [23] . Each microservice can be developed, deployed, and scaled independently [24] .
- Read about the benefits: scalability (scale services individually), resilience (one service failure doesn't take down the whole system), and flexibility in using different tech per service [24] . Also note challenges: complexity in orchestration and communication between services.
- Familiarize yourself with common microservices patterns: **API Gateway**, **Service Discovery**, **Circuit Breakers** (just concepts at a high level).
- *Resource:* Martin Fowler's article *"Microservices"* (covers the characteristics and benefits of microservice architecture) [25] .
- **Day 37 (Tue, July 8)** – **(2h)** *Spring Cloud and Microservices in Spring*:
- Explore how Spring Boot is often used for microservices. Learn about **Spring Cloud** – a suite of tools that complement Spring Boot for microservices (e.g., Spring Cloud Netflix components like Eureka for service discovery, Spring Cloud Gateway for API gateway, Config Server, etc.).
- You don't need to implement these yet, but know the landscape. For example, read a bit about **Eureka** (service registry) and how services find each other, and **OpenFeign** (a declarative HTTP client to call other service APIs) [26] .
- Optional hands-on: If curious, you can try a quick Spring Cloud tutorial (like setting up a Eureka server and a dummy client service), but this can be complex. It's okay to focus on conceptual understanding given time constraints.
- *Resource:* Spring.io Blog or Baeldung – *Guide to Spring Cloud* (to see what tools exist for building microservices with Spring).
- **Day 38 (Wed, July 9)** – **(2h)** *Planning Your Capstone Project*:
- Begin planning the **full-stack capstone project** you will build in Month 4–5. This should be a substantial project to showcase on your resume, integrating everything: React front-end, Spring Boot back-end, a SQL database, and maybe a small microservice or NoSQL component for bonus. Choose an idea that excites you and touches on common requirements for jobs.
- Ideas: **E-commerce app** (with product catalog, users, orders), **Blog platform** (users, posts, comments), **Task manager with teams**, **Expense tracker**, etc. Make sure it's complex enough to demonstrate CRUD, auth, relationships, etc., but scoped to 4-6 weeks of work.
- Outline the features and tech stack: e.g., *"I will build a Blog app with a React front-end, Spring Boot REST API, MySQL for persistence, and use Spring Security for auth. I might include a second microservice for a recommendation or notification system as an extension."*
- Design the data model on paper: list entities (for blog example: User, Post, Comment, etc. with relationships). This will guide your implementation.
- *Resource:* "Project Planning for Developers" (freeCodeCamp or similar article on how to plan personal projects – focusing on MVP feature selection).
- **Day 39 (Thu, July 10)** – **(2h)** *NoSQL Introduction – MongoDB Concepts*:

- Learn about **NoSQL databases** and when to use them. Unlike SQL databases, NoSQL stores flexible schemas or key-value pairs. **MongoDB** is a document-oriented NoSQL DB storing JSON-like documents [27] [28].
- Understand use-cases: e.g., storing schema-less data, or when you need to scale writes horizontally. For your knowledge (and interviews), know basic MongoDB terms: *database*, *collection* (equivalent to a table), *document* (a JSON record).
- Install MongoDB locally (or use a cloud MongoDB Atlas free tier) and use a Mongo client or the Mongo shell to create a database and insert a sample document. Practice a simple query (MongoDB query language for filtering by a field).
- *Resource:* Official MongoDB "Basics" tutorial or GeeksforGeeks MongoDB intro [29].
- **Day 40 (Fri, July 11) – (2h)** *Spring Data MongoDB – Hands-on*:
- Extend your knowledge by writing a small Spring Boot service that uses MongoDB. For instance, create a new Spring Boot project (or module) called "Notes Service" where you store simple notes in MongoDB.
- Add **Spring Data MongoDB** dependency. Configure MongoDB connection in `application.properties` (if Mongo is running locally, it might connect to `mongodb://localhost:27017` by default).
- Create a `Note` class and mark it with `@Document` (Spring Data annotation for Mongo collections). Include fields like id, content, date.
- Create a `NoteRepository` interface that extends `MongoRepository<Note, String>`.
- Write a simple `NoteController` with endpoints to create and fetch notes, using the repository. This is similar to JPA but for Mongo – Spring's programming model is very consistent across databases [30] [31].
- Test the endpoints by storing some notes (the data will be saved in Mongo). Observe in the Mongo shell that documents appear in the collection.
- *Resource:* Spring.io guide "Accessing Data with MongoDB" – very similar to the JPA one, but for NoSQL [32] [33].
- **Day 41 (Sat, July 12) – (3h)** *Docker Basics – Containerizing Your App*:
- Learn what **Docker** is and why it's useful: it allows packing your application with its environment into a "container" so it runs consistently across different machines [34]. This is crucial for deploying apps and especially useful in microservices setups (each service in its own container).
- Install **Docker Desktop** on your machine. Go through a beginner tutorial: e.g., containerize a simple "Hello World" or a basic Node app first just to understand `Dockerfile`, `docker build`, and `docker run`.
- Write a **Dockerfile** for your Spring Boot application (the Task API or Notes service). Usually, you'd use an openjdk base image and copy the jar. For example:

```
FROM openjdk:17-jdk-alpine
COPY target/myapp.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

- Build the Docker image (`docker build -t myapp:1.0 .`) and run a container (`docker run -p 8080:8080 myapp:1.0`). Test that you can access your app's endpoints via `localhost:8080` when it's running in Docker. If it works, you've containerized your Spring Boot app!
- *Resource:* Docker's official "Getting Started" guide (covers building your first container) [34].
- **Day 42 (Sun, July 13) – (3h)** *Docker-Compose and Multi-Container Setup*:
- Extend your Docker knowledge by exploring **Docker Compose** (a tool to run multi-container environments easily). Create a `docker-compose.yml` file to run your Spring Boot app

alongside its database. For example, one service for your Spring Boot API and another for a MySQL database. Configure networking so the app can talk to the DB.
- Try bringing up the whole stack with `docker-compose up`. This is practice for deploying multi-tier applications (and a mini simulation of a microservices environment on your machine) [35] [36].
- Ensure your Spring Boot app can indeed connect to the DB container (you might need to adjust the DB hostname to the service name from compose). This can be tricky – use logs to debug connectivity.
- **Review:** Summarize to yourself what you did this week – especially microservices, NoSQL, and Docker. These are heavy topics; it's okay if you don't master them fully. Knowing the basics and having done a small POC (proof of concept) will allow you to speak about them in interviews.
- If something didn't go smoothly (e.g., trouble containerizing or using Docker Compose), use today to troubleshoot or read more. Otherwise, take a short break to recharge. Next week, you'll start the big project in earnest!

Month 4 (September 2025): Capstone Project Development (Backend)

Time to build your capstone **full-stack project**. This month, focus on implementing the backend (Spring Boot) for your chosen project, incorporating robust features: authentication, relational database, and perhaps integrating some of the advanced tech you learned (maybe a touch of NoSQL or an external API) to make it stand out. By the end of Month 4, your backend should be mostly complete and tested, ready for front-end integration in Month 5.

Week 7 (July 14 – July 20, 2025) – *Project Kickoff & Backend Setup*

- **Day 43 (Mon, July 14)** – **(2h)** *Project Bootstrapping*:
- Create a new **Spring Boot project** for your capstone (use Spring Initializr with needed dependencies: Spring Web, Spring Data JPA, Spring Security, etc., depending on your project needs). Set up your project structure (packages for controllers, services, models, repositories as you practiced).
- Initialize a Git repository for version control. This project should be on GitHub (or GitLab) from the start – you'll want to show this to employers, and committing regularly also helps you track progress.
- Define your core entities as Java classes. For example, if doing a Blog: `User`, `Post`, `Comment`. Add JPA annotations (`@Entity`) and outline fields (don't worry if you tweak them later). Focus on the main ones required for MVP (minimum viable product).
- *Resource:* Example GitHub project for a similar app (just to see structure). E.g., search "Spring Boot blog app example" – but remember not to copy, just to get ideas.
- **Day 44 (Tue, July 15)** – **(2h)** *Repository & Service Layer Creation*:
- Create **Repository interfaces** for your entities (e.g., `UserRepository`, `PostRepository`, etc.). Extend `JpaRepository` to inherit basic CRUD operations.
- Set up the **Service layer** for key functionalities. For instance, `UserService` for user-related logic (like registering a user, finding by username), `PostService` for post-related logic (like creating a post, fetching posts with filters). At this point, you might still be using stub data or simple implementations, which is fine.
- If any complex relationships exist (e.g., a User has many Posts), think about how to handle those (via JPA mappings `@OneToMany`, etc.). Add those annotations where appropriate and decide how you'll load data (eager vs lazy loading decisions – e.g., maybe lazily load `comments` on a post).

- *Resource:* Thực hành (practice) reference: if your project idea is common (like e-commerce), look up schema designs for that domain to ensure you're not missing an important field or relationship.
- **Day 45 (Wed, July 16)** – **(2h)** *Implementing Core Features (CRUD)*:
- Start coding the core **CRUD operations** for your main entities. For example, implement user registration (create user), login (we'll do security soon, but you can stub this), create post, edit post, delete post, list posts, etc. Focus on functionality before security: allow operations openly first.
- Use your repository methods or custom queries to fetch the necessary data. For instance, if you need "all posts by a certain user" implement `findByUserId` in `PostRepository` or use a service method that calls the repo.
- Test as you go: write simple unit tests for service methods if comfortable, or just use a main method / Postman to call repository methods via a temporary runner. Ensuring your logic works before adding the web layer is helpful.
- *Resource:* Baeldung – *Testing JPA Repositories* (for guidance if you want to write tests). Not mandatory, but good practice.
- **Day 46 (Thu, July 17)** – **(2h)** *Spring Security – Project Integration*:
- Time to add **authentication** to your project. Decide on your approach: a common scenario is to use JWT for a SPA (single-page app) front-end. Alternatively, you can do session-based auth with Spring Security's form login if you were building a traditional web app. For our full-stack SPA, JWT is modern and convenient.
- Implement an **AuthController** that has endpoints like `/auth/register` and `/auth/login`. For registering, create a new User (hash the password!). For login, verify credentials and return a JWT token. You can use **Spring Security's AuthenticationManager** and **UserDetailsService** for this, or use a library to help with JWT.
- Simplify if needed: It's acceptable to use Basic Auth in development or have a hardcoded user to protect certain endpoints. But ideally, generate JWT tokens on login. There are many tutorials with code you can adapt – just make sure you understand the flow.
- Store passwords securely: include something like Bcrypt encoder (`PasswordEncoder`) to hash passwords.
- *Resource:* "Spring Boot JWT Auth for REST APIs" (e.g., a step-by-step Medium article or YouTube by Java Guru) [37] . Follow it to integrate JWT in your project.
- **Day 47 (Fri, July 18)** – **(2h)** *Completing API Endpoints*:
- Finish implementing all planned **REST endpoints** in your backend. This includes the main resource endpoints (posts, comments, etc.) with proper HTTP methods and response codes, and any additional features (maybe a search endpoint or a user profile endpoint, etc.).
- Ensure to handle edge cases: e.g., return 404 if an entity is not found, 400 for bad input, etc. Reuse the exception handling techniques you learned (ControllerAdvice for global errors, etc.).
- Add **DTOs** if needed: Sometimes you may not want to expose the entire Entity directly. Consider creating Data Transfer Objects for sensitive operations (e.g., a UserDTO that doesn't include password). This is a good practice, but if time is short, you may skip detailed DTO creation and use entities in responses carefully (excluding sensitive fields with annotations like `@JsonIgnore`).
- Test all endpoints with Postman, using real example data. Essentially, pretend you are the front-end: create a few users, log in, create some posts, fetch them, update them, etc., to simulate usage.
- *Resource:* Postman or curl examples – (you can document some example requests for later use in frontend integration). No external citation needed here, just good practice.
- **Day 48 (Sat, July 19)** – **(3h)** *Integration of NoSQL or External Service (Optional Enhancement)*:
- If your project could benefit from it (and you have time/energy), integrate an advanced component to impress employers: for example, use **MongoDB** for a specific feature. Perhaps

store logs or analytics in Mongo, or user activity streams, separate from the primary MySQL data. Alternatively, integrate an external API – e.g., call a public weather API if your app needs weather info, etc.

- For a blog project, maybe integrate a **search** feature using a library, or store some data in Redis/ Mongo for quick access. For an e-commerce, maybe use a currency conversion API.
- **Keep it limited**: one small integration is enough. The goal is to show you're capable of using multiple tools. If you choose Mongo, perhaps reuse the knowledge from the Notes service exercise: set up a secondary Mongo connection in your app and a repository for that data. If using an external REST API, practice making an HTTP call from your Spring service (use `RestTemplate` or WebClient to call an external URL).
- Document whatever you attempt – even if not fully finished, knowing you tried and understanding how it works is valuable for interviews.
- *Resource:* Varies based on what you choose (e.g., if integrating an API, use its docs; if adding Redis, see Spring Data Redis guide). No matter what, ensure the core app still runs correctly after this enhancement.
- **Day 49 (Sun, July 20)** – **(3h)** *Testing & Backend Code Review*:
- **Catch-Up/Review Day:** Your backend is mostly done! Use today to review and polish:
    - **Testing:** Write a few unit or integration tests for critical pieces (e.g., test that an unauthorized request is rejected, or that a repository method returns expected results). If not comfortable with testing, you can do manual testing but try to at least create one JUnit test class to demonstrate you know how (this could impress interviewers if you mention it).
    - **Code Review:** Go through your code for any bugs or TODOs. Simplify where possible, add comments for clarity, and ensure consistency in error handling and naming.
    - **Documentation:** Create a simple **README** for your project in the repo. Document what the project is, how to run it, and list the API endpoints (you might also add Swagger to your project for API docs if you know it – optional).
    - If you are behind on any backend feature, use today to catch up so that next week you can start front-end without worry.
- *Resource:* Swagger (OpenAPI) Spring Boot integration guide, if you want to auto-document your APIs. Otherwise, a manual README of endpoints is fine.

Week 8 (July 21 – July 27, 2025) – *Front-End Development for Capstone Project*

With the backend ready and tested, turn to the **React front-end** for your project. This week, you'll build the UI and make sure it communicates with the backend API (including handling authentication). By the end of this week, you should have a functional full-stack application running locally.

- **Day 50 (Mon, July 21)** – **(2h)** *Front-End Project Setup*:
- Set up a new **React app** (using Create React App, Vite, or Next.js if you prefer SSR, but CRA is fine for an SPA). If using CRA, decide whether to use JavaScript or TypeScript – given modern practices, using **TypeScript** is recommended to showcase that skill. If you're new to TS, you already practiced a bit last month, so try using the CRA TypeScript template.
- Organize your project structure (e.g., components, pages, services for API calls, context for global state like auth).
- Install UI framework or component library if you plan to use one (e.g., Material-UI, Ant Design, or even Bootstrap) to make styling easier – not required, but it can speed up making a decent-looking UI.
- Commit this initial setup to your repo. Possibly create a separate repo for the front-end, or a subfolder in a monorepo – whatever you prefer, but ensure version control.

- *Resource:* Official React documentation on "Thinking in React" (for structuring components effectively).
- **Day 51 (Tue, July 22)** – **(2h)** *Implementing Authentication in React*:
- Focus on the **authentication flow** on the front-end. Create components for Login (and Signup if your app has registration on the front-end).
- On Login form submit, call the backend `/auth/login` endpoint with credentials. If using JWT, store the returned token (likely in localStorage or an in-memory context/store). Also set up a way to include this token in subsequent API requests (e.g., configure axios with an auth header, or attach manually via fetch).
- Manage the logged-in state: perhaps use React Context or a state management library to keep track of the current user and auth status globally. For simplicity, Context + useReducer can serve as a mini store for auth info.
- Protect routes on the front-end: Implement a mechanism (like a `<PrivateRoute>` component) that checks if user is logged in (token present) and either allows access or redirects to login.
- *Resource:* Blog post "Managing JWT in React" – covers storing tokens and attaching them to API calls securely (with considerations for security).
- **Day 52 (Wed, July 23)** – **(2h)** *Core UI – List and Create Operations*:
- Start building the main parts of your app's UI. For a blog, this could be the Post listing page; for e-commerce, the product catalog; for a task app, the task list, etc.
- Fetch data from your API and display it. E.g., call GET `/posts` and display a list of posts with titles and excerpts. Use `useEffect` to load data on component mount.
- Implement a form to create a new entity (new Post, new Task, etc.). This should gather input and send a POST request to the API. After creation, you might refresh the list or route the user to the new item's detail page.
- Ensure you handle errors in the UI (e.g., show validation messages if API returns error, like "Title is required" or "Unauthorized" if token expired). This will make your app feel more complete.
- *Resource:* React Hook Form library (optional to manage form state/validation easily in React) if the forms are complex. Otherwise, simple controlled components for form inputs will do.
- **Day 53 (Thu, July 24)** – **(2h)** *Remaining Features – Update & Delete*:
- Complete the CRUD on front-end: implement the ability to update existing records (e.g., an "Edit" button on a post or profile). This likely involves populating a form with existing data and sending a PUT request to the backend.
- Implement delete operations with a confirmation prompt (e.g., a delete button that calls DELETE API endpoint, then removes the item from UI state on success).
- Add nice-to-haves: loading states (show a spinner or message while API calls are in progress), and basic error displays (e.g., toast notifications or alert messages on failure).
- If your app has any additional feature (comments on a post, adding items to cart, etc.), integrate those now similarly by calling the respective endpoints you built.
- *Resource:* For a polished UI, you might refer to component library docs (if using one) for how to implement modals, spinners, etc. (e.g., Material-UI's Progress component for loading).
- **Day 54 (Fri, July 25)** – **(2h)** *UI Styling and Polish*:
- Spend time making your app look presentable. Use CSS or a UI framework to style the layout, navigation, and components. This doesn't have to be fancy, but a clean look is important for impression. Ensure mobile-responsiveness if possible (maybe use flexbox or simple grid layouts).
- Implement navigation links or a menu to navigate between pages (e.g., Home, Create New, Profile, etc.). If the user must log in first, ensure the app routes them accordingly.
- Test the full user flow: start from not logged in, go through login, view list, create something, edit it, delete it, log out. Fix any bugs that pop up in this end-to-end test.
- Ensure that restricted actions are indeed restricted on the UI (even though backend also secures them). E.g., hide admin-only buttons from regular users if your app has roles.

- *Resource:* "Responsive Web Design Basics" (Mozilla Developer Network or freeCodeCamp) if you need a refresher on making things responsive.
- **Day 55 (Sat, July 26)** – **(3h)** *Integration Testing & Bug Fixes*:
- Now test the **full stack integration** thoroughly: run both your backend and frontend together and exercise all features. This might reveal bugs (e.g., CORS issues, incorrect API paths, state not updating properly, etc.). Debug and fix these issues.
- Pay attention to the network calls in the browser dev tools – check that JWT tokens are being sent, that responses look correct, and that errors (if any) are handled gracefully.
- If you have a friend or colleague, this is a great time to ask them to pull your project and run it, acting as a user to find issues or confusing parts. Sometimes we miss obvious things because we're too close to the project.
- Work on any missed edge cases. Example: what if a token expires? (Maybe out of scope to fully handle in 6 months, but you could implement a simple "token expired, please log in again" flow by checking 401 responses). Or if using refresh tokens, ensure those work (advanced).
- *Resource:* None needed – use your own application logs, browser console, and testing mindset here.
- **Day 56 (Sun, July 27)** – **(3h)** *Catch-Up & Deployment Prep*:
- **Catch-Up Day:** Ensure your full-stack project is basically feature-complete as planned. If you fell behind on any functionality or styling, use today to finish it up.
- Begin thinking about **deployment**: Next month you will deploy this app (backend on AWS, frontend maybe on AWS S3 or another static host). Prepare for that by checking configurations:
    - Externalize any configuration that shouldn't be hard-coded (like DB credentials, they should be in properties and you might use environment variables in production).
    - Make sure your frontend API URLs are not pointing to `localhost` in a way that's hard to change – you might load them from a config or use relative paths if hosting front and back together.
- If you're totally on track, you can rest or optionally write some documentation for your project (a brief user guide or update the README with screenshots). You could also create sample test data so that when deployed, the app has something to show (e.g., a few demo accounts or items).
- Congratulate yourself – you've built a full stack application! This is a huge milestone.

Month 5 (October 2025): Deployment, Cloud & Final Touches

In Month 5, you will deploy your project and familiarize yourself with basic **AWS cloud services**: EC2 for hosting your backend, S3 for static frontend or file storage, and RDS for a managed database. You'll also polish your project, ensure everything is documented, and start preparing your resume and portfolio for job applications.

Week 9 (July 28 – August 3, 2025) – *AWS Cloud Deployment*

- **Day 57 (Mon, July 28)** – **(2h)** *AWS Essentials and Account Setup*:
- If you haven't already, create an **AWS Account** (ensure you use free tier resources to avoid charges). Familiarize yourself with the AWS Management Console interface.
- Learn about AWS **EC2** (Elastic Compute Cloud) – basically virtual machines in the cloud. Also recall AWS **S3** (Simple Storage Service) for storing files, and **RDS** (Relational Database Service) for managed SQL databases [38] .
- Read an overview of these core services: EC2 provides virtual servers, S3 provides scalable object storage, RDS provides managed database instances [38] . Understand at a high level how they'll fit: we'll host the Spring Boot app on EC2, use either our local DB or an RDS instance, and possibly host the React app on S3.
- *Resource:* GeeksforGeeks – *Introduction to AWS* (explains EC2, S3, RDS in simple terms) [38] .

- **Day 58 (Tue, July 29)** – **(2h)** *Deploying Backend to EC2*:
- Launch an **EC2 instance** (Amazon Linux or Ubuntu). Ensure it's in a region close to you (for latency) and free-tier eligible (t2.micro). Configure key pair for SSH and open necessary security group ports (at least port 22 for SSH, and 80/8080 for your app).
- SSH into the EC2 server. Install Java (if not already) since you need JDK to run your Spring Boot jar. Also install Docker if you plan to deploy via Docker (alternatively, you can run the jar directly with java).
- Transfer your Spring Boot application to EC2. E.g., you can SCP the jar file or docker image, or git clone your project and build on the server. Easiest: package a fat jar (`mvn package`) on your machine, then upload the jar.
- Run the app on EC2: if using jar, `nohup java -jar app.jar &` to run in background; if using Docker, run your container exposing the right port.
- Test the API from your local machine by hitting the EC2's public IP (or public DNS) on the exposed port. If it works (you get a response), congrats – your backend is live on the internet! [34]
- *Resource:* Official AWS tutorial "Deploy a Spring Boot Application to EC2" (if available) or a Medium article with step-by-step instructions.
- **Day 59 (Wed, July 30)** – **(2h)** *Deploying Frontend to S3 (Static Hosting)*:
- Build your React app for production (`npm run build` for CRA). This produces static files (HTML, JS, CSS).
- Create an **S3 bucket** for static website hosting. Enable "Static website hosting" in its properties. Note the endpoint (it will be like `http://<bucket-name>.s3-website-<region>.amazonaws.com`).
- Upload your React build files to the S3 bucket (you can use the AWS Console or CLI). Make sure files are publicly readable (for a static site) or configure the bucket policy to allow public read on the files.
- Test accessing the S3 website URL – your React app should load. It might still not work fully if it's trying to call the API at localhost. So, update the API base URL in your React app to point to your EC2's public address (and redeploy if needed). Ensure CORS on your backend is configured to allow requests from your S3 domain or a wildcard for simplicity during dev.
- Alternatively, you can deploy the React app on EC2 as well (install Nginx and serve static files), but S3 is simpler and scalable for static content.
- *Resource:* AWS Docs – *Hosting a Static Website on Amazon S3* (guides through bucket setup and permissions).
- **Day 60 (Thu, July 31)** – **(2h)** *Setting up RDS (Optional if using cloud DB)*:
- If you want your database in the cloud (not just running on EC2), set up an **RDS** instance for MySQL/PostgreSQL. This is optional; if your app's DB is small, you might continue with a local or EC2-hosted DB. But knowing RDS is good for interviews.
- Launch a free-tier RDS database (MySQL or PostgreSQL). Set master username/password and wait for it to be available.
- Configure your Spring Boot app (on EC2) to connect to RDS: update the DB URL, username, password (consider using AWS Secrets Manager or just keep it simple for now by using plaintext in properties). You might redeploy the app with these new settings.
- Ensure the EC2's security group can reach the RDS instance's security group on the DB port (3306 for MySQL, 5432 for Postgres). If not, adjust security groups.
- Migrate your schema: use your existing SQL scripts or connect remotely to RDS and run the CREATE TABLE statements to set up initial schema (or use Spring JPA auto DDL if enabled). Seed data if necessary.
- Test the application again via the front-end to ensure it works with the cloud DB.
- *Resource:* AWS documentation on RDS setup, or a digitalocean tutorial "How to set up MySQL on AWS RDS".
- **Day 61 (Fri, Aug 1)** – **(2h)** *Docker and Deployment Automation (Optional)*:

- Containerization in production: If you deployed manually, consider making it more streamlined. For example, create a **Docker image** for your backend and push it to **ECR** (Elastic Container Registry) or Docker Hub, so the server can pull and run it easily.
- Alternatively, explore using **AWS Elastic Beanstalk** or **AWS ECS/Fargate** to deploy your containerized app. Elastic Beanstalk can deploy a Spring Boot jar or Docker with minimal setup – it handles provisioning EC2, load balancing, etc. This might be an overkill for now, but it's good to be aware of.
- If you find this too time-consuming, skip to focusing on documentation and portfolio. The key is you have a working deployment. This step is just if you want to add DevOps flavor to your skillset.
- *Resource:* AWS Elastic Beanstalk Spring Boot tutorial – if you want a relatively quick way to deploy by just uploading your jar.
- **Day 62 (Sat, Aug 2)** – **(3h)** *Final Project Polishing*:
- Now that your app is deployed, polish any remaining aspects:
    - **Monitoring**: Set up basic CloudWatch monitoring for your EC2 instance (CPU, memory) or logs if inclined, just to familiarize (optional).
    - **Backup plan**: Ensure you have backups of your code (GitHub) and maybe database (manual export if needed).
    - **App polish**: Maybe create a few default accounts on the live site (for demo purposes) or dummy data so that a recruiter visiting can see something without logging in (if appropriate). For instance, have a read-only demo mode or provide a demo login credential in your README.
    - Check performance: The free-tier instance is small; see that your app pages load reasonably. If the first load is slow due to EC2 cold start, mention in docs that it might take a moment.
- Make a short **video** or screenshots of your running application. This is useful to share along with your resume or on LinkedIn.
- *Resource:* N/A – use this time to really make your project something you're proud to show.
- **Day 63 (Sun, Aug 3)** – **(3h)** *Project Documentation & Reflection*:
- Write detailed **documentation** for your project: update the README with instructions to run locally and a link to the deployed version. Write about the tech stack used (React, Spring Boot, MySQL, etc.), and features implemented. This will help you recall details during interviews and shows professionalism.
- If you have a personal blog or portfolio site, write a short article about this project – the challenges faced and the learning from it. It's great content for recruiters to see your journey. (Optional but highly recommended if you enjoy writing.)
- **Reflection:** Review the past 5 months of learning. Skim through your notes or this schedule and appreciate how far you've come – from Java basics to deploying a full app on AWS! Identify any weak areas you want to revisit in the final month (e.g., "I'm still not very confident about Spring Security" or "I want to practice some coding problems for interviews"). We will tackle those in the next month.
- *Resource:* Your own documented journey – consider posting it on LinkedIn or Medium to signal your new skills to your network.

Month 6 (November 2025): Interview Prep and Job Applications

Congratulations – by the start of Month 6, you have a project and a broad skill set in full-stack development (with a backend focus). This final month is about making you **job-ready**: polishing your resume, applying to jobs, and preparing for technical interviews (including data structures, system design, and commonly asked questions in Java/Spring). By the end of this month, you should be actively applying and ready to handle interviews, which are expected to happen in Month 7.

Week 10 (Aug 4 – Aug 10, 2025) – *Resume, Portfolio, and Job Applications*

- **Day 64 (Mon, Aug 4)** – **(2h)** *Update Resume*:
- Rewrite your **resume** to highlight your new backend/full-stack skills. Add your capstone project under a Projects section: describe the tech stack (React, Spring Boot, Docker, AWS) and what features you implemented (CRUD operations, authentication, deployment) – this will catch recruiters' eyes.
- Under skills, list relevant technologies: Java, Spring Boot, REST APIs, MySQL/PostgreSQL, MongoDB, Docker, AWS (EC2/S3/RDS), React, etc. Mention 4+ years of React experience plus the new backend skills – positioning you as a full-stack developer.
- Tailor your experience section (if you have prior roles) to highlight any full-stack or collaborative backend work you might have done, even tangentially. Emphasize problem-solving and willingness to learn new technologies – since you clearly did in these 6 months.
- Keep the resume concise (1-2 pages) and impactful. Use action verbs and quantify achievements if possible (e.g., "Implemented a full-stack web application with X features using Spring Boot and React, deployed on AWS" is a strong statement).
- *Resource:* Google "full-stack developer resume examples" to see how others phrase their projects and skills.
- **Day 65 (Tue, Aug 5)** – **(2h)** *LinkedIn and Portfolio Cleanup*:
- Update your **LinkedIn** profile to reflect your new skill set. Add a tagline like "Full-Stack Developer (Java Spring Boot & React)". In your About section, mention you've been upskilling in backend technologies. Also, add your project in the Projects section with a brief description and link.
- If you have a GitHub account (you should!), ensure your project repo is public and well-presented (good README, decent commit history). Pin this repository on your GitHub profile. Recruiters often check GitHub for active projects.
- (Optional) If you created a personal portfolio site, update it with your new project and skills. If not, it's okay – your GitHub/LinkedIn can serve as portfolio.
- Start preparing a list of companies or job postings you want to apply to. Identify roles labeled "Java Developer", "Backend Developer", or "Full-Stack Developer (Java/React)". Check that you meet most requirements; note any gaps to address in interview prep (e.g., if many mention Spring Security or AWS, you have those covered!).
- *Resource:* Use LinkedIn job search and filter for "Java Spring Boot developer Chennai" or remote, start saving those jobs.
- **Day 66 (Wed, Aug 6)** – **(2h)** *Job Applications (Part 1)*:
- Begin applying to jobs. Aim to send out a few applications today. Prioritize roles that excite you and match your new skills. Customize your cover letter or email for each: briefly mention your 4 years of React experience and highlight the backend skills you've acquired (e.g., "...recently built a Spring Boot RESTful API with a React front-end, deployed on AWS [39] ").
- Use job boards: LinkedIn, Indeed, Glassdoor, Stack Overflow Jobs, etc. Also consider reaching out to recruiters or contacts in Chennai who might know openings (leverage your network – let them know you've upskilled in Java backend).
- Keep track of applications in a spreadsheet (company, role, date applied, status). This will help you follow up later.
- *Resource:* "How to get a job as a Java developer" articles might give tips on what recruiters look for – ensure you highlight those points in your applications.
- **Day 67 (Thu, Aug 7)** – **(2h)** *System Design Basics*:
- Some interviews (especially for senior or full-stack roles) might include high-level **system design** discussions. Prepare by reviewing basic system design concepts: how a web request flows, fundamentals of scalability (vertical vs horizontal scaling, load balancers), caching, and microservices vs monolith trade-offs.

- Think about the design of your own project: how would you scale it if needed? What if you had 10x traffic – would you add another app server and a load balancer? What if you needed to break it into microservices – how might you separate responsibilities?
- Learn a few classic system components: e.g., design a URL shortener, or an online library system (these are common design interview scenarios). Focus on identifying core components and their interactions (client -> server -> database, etc.).
- *Resource:* Educative.io or free system design prep articles (e.g., "System Design Primer" on GitHub) for beginners.
- **Day 68 (Fri, Aug 8)** – **(2h)** *Java and Spring Interview Q&A*:
- Revisit core **Java concepts** that are commonly asked in interviews: OOP principles (polymorphism, inheritance – can you explain them clearly?), the difference between an interface and abstract class, how garbage collection works generally, etc. Also thread basics (what is a thread, synchronization).
- Prepare for **Spring/Spring Boot questions**: e.g., *"What is Spring Boot and how is it different from Spring?"* (Answer: Spring Boot is built on Spring to simplify configuration and setup, provides embedded servers, starters, auto-configuration [11]), *"What is dependency injection?"*, *"What is a Spring Bean?"*, *"How do you handle database transactions in Spring?"*, *"What is REST and have you designed REST APIs?"* (Yes, you have!).
- Review a list of **Spring Boot interview questions** [39] to see what topics are covered. For example, understand how auto-configuration works, what Actuator is, how to externalize config, etc., at least on a surface level.
- Also refresh on JPA/Hibernate questions: lazy vs eager fetch, what is an Entity, how do you write JPQL, N+1 problem (just basics). And some SQL questions (e.g., join vs union difference, indexing).
- *Resource:* GeeksforGeeks Spring Boot interview Q&A [39] and Java interview questions reference. Focus on understanding rather than memorizing.
- **Day 69 (Sat, Aug 9)** – **(3h)** *Data Structures & Algorithms Practice*:
- Spend a solid block practicing coding problems, as many tech interviews for developers include DS&A questions. Focus on problems that can be solved in 15-20 minutes. Use HackerRank, LeetCode or similar.
- Topics to prioritize: **arrays and strings** (since those are common and you can solve via simple loops, use cases like two-sum problem, string reversal, removing duplicates, etc.), **hashmaps** (e.g., counting frequencies, two-sum via map), **linked list basic operations**, and **perhaps binary search or sorting logic**.
- Also practice at least one problem involving a **stack or queue** (e.g., valid parentheses check using stack), and one simple **tree traversal** (if time – maybe just understanding how to do a DFS/BFS). For a backend role with 4+ years experience, they may not grill heavy algorithms like LeetCode hard, but you should be comfortable with basics and writing code in Java on a whiteboard or shared editor.
- Simulate a timed interview: pick an easy/medium LeetCode problem, set a 20-30 minute timer, and solve it in Java. Then check solutions and understand any mistakes.
- *Resource:* LeetCode "Top Interview Questions" list or HackerRank easy challenges section. Emphasize reasoning and clarity over cleverness.
- **Day 70 (Sun, Aug 10)** – **(3h)** *Mock Interview and Q&A*:
- It's a catch-up and **mock interview** day. If you know someone in the industry, request a mock interview session for both technical and behavioral questions. If not, do it yourself: speak answers out loud as if in an interview, for common questions like "Tell me about yourself" (focus on your transition story and project), "Describe a challenge you overcame", and technical ones ("Explain your project's architecture", "How does Spring Dependency Injection work?", etc.).

- Review any remaining weak points you've noticed. For example, if you struggled in algorithm practice with a certain data structure, review how it works. If you are unsure about a Spring annotation usage, quickly look it up.
- Prepare a few good questions to ask interviewers (e.g., about the tech stack, team practices). This shows interest.
- Finally, ensure you have everything ready for interviews: If coding round on your machine, have an IDE or environment ready; if system design on a whiteboard, have a checklist of key points (like clarify requirements, etc.). If remote, test your internet and audio/video setup.
- *Resource:* "Cracking the Coding Interview" book summaries (for behavioral questions and general tips), and any notes you've taken. Also consider using Pramp or other mock interview platforms for a final practice.

Wrapping Up and Looking Ahead

By the end of Month 6, you should be actively interviewing. Continue applying to new jobs each week (job searching is a bit of a numbers game) and keep your skills sharp by reviewing notes or doing small coding exercises. In **Month 7**, focus on the interview process itself: for each interview, revisit relevant topics (for a Java role, anticipate Java and system design questions as we covered). Also, work on **communication** – explain your thoughts clearly and confidently, leaning on the projects and knowledge you've gained.

Keep in mind, learning is continuous. Even as interviews proceed, you can refine your knowledge on-the-fly: if an interviewer asks something unfamiliar (say, a specific AWS service or a tricky algorithm), note it down, study it that evening, and you'll be even more prepared for the next one.

Good luck! With this dedicated 6-month preparation, you have built a strong foundation in Java backend development on top of your React experience. You're now well-equipped to land a Java backend or full-stack role – go forth and showcase your new skills with confidence!

---

[1] [3] [6] [8] [9] [10] [12] [13] [14] [16] [17] [18] [19] [21] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] Best Way to Master Spring Boot – A Complete Roadmap | GeeksforGeeks
https://www.geeksforgeeks.org/best-way-to-master-spring-boot-a-complete-roadmap/

[2] [7] [15] [20] [22] [37] 6 Months to Backend Developer: My Journey From Zero IT Knowledge to Coding Professionally | by Rizky Ananda | May, 2025 | Medium
https://medium.com/@rizkyand/6-months-to-backend-developer-my-journey-from-zero-it-knowledge-to-coding-professionally-5b2b3516c824

[4] [5] Lesson: Classes and Objects (The Java™ Tutorials > Learning the Java Language)
https://docs.oracle.com/javase/tutorial/java/javaOO/index.html

[11] [39] Spring Boot Interview Questions and Answers | GeeksforGeeks
https://www.geeksforgeeks.org/spring-boot-interview-questions-and-answers/

[34] [35] [36] Docker for Web Developers: Starting with the Basics | Docker
https://www.docker.com/blog/docker-for-web-developers/

[38] Amazon Web Services (AWS) Tutorial | GeeksforGeeks
https://www.geeksforgeeks.org/aws-tutorial/