GSDC
Global Skill Development Council

JAVA FULL STACK DEVELOPER

Quick Referenc Guide

www.gsdcouncil.org

1. Introduction to Full Stack Development

What is Full Stack Development?

Full Stack Development is the practice of designing and building every layer of a software application — from the visual interface a user sees to the server logic that powers it, the database that stores it, and the infrastructure that runs it. A Java Full Stack Developer owns the entire vertical slice of the technology stack, making them one of the most versatile and sought-after professionals in the industry today.

Unlike specialists who focus on a single layer, a full stack developer must be fluent in client-side technologies, server-side frameworks, relational databases, RESTful API design, and modern deployment pipelines. This breadth of knowledge is exactly what the GSDC CFSD certification validates.

Front-End

Client-side UI rendered in the browser

Back-End

Server-side business logic and processing

Database Layer

Persistent storage and data management

API Integrations

Communication bridges between services

DevOps & Deployment

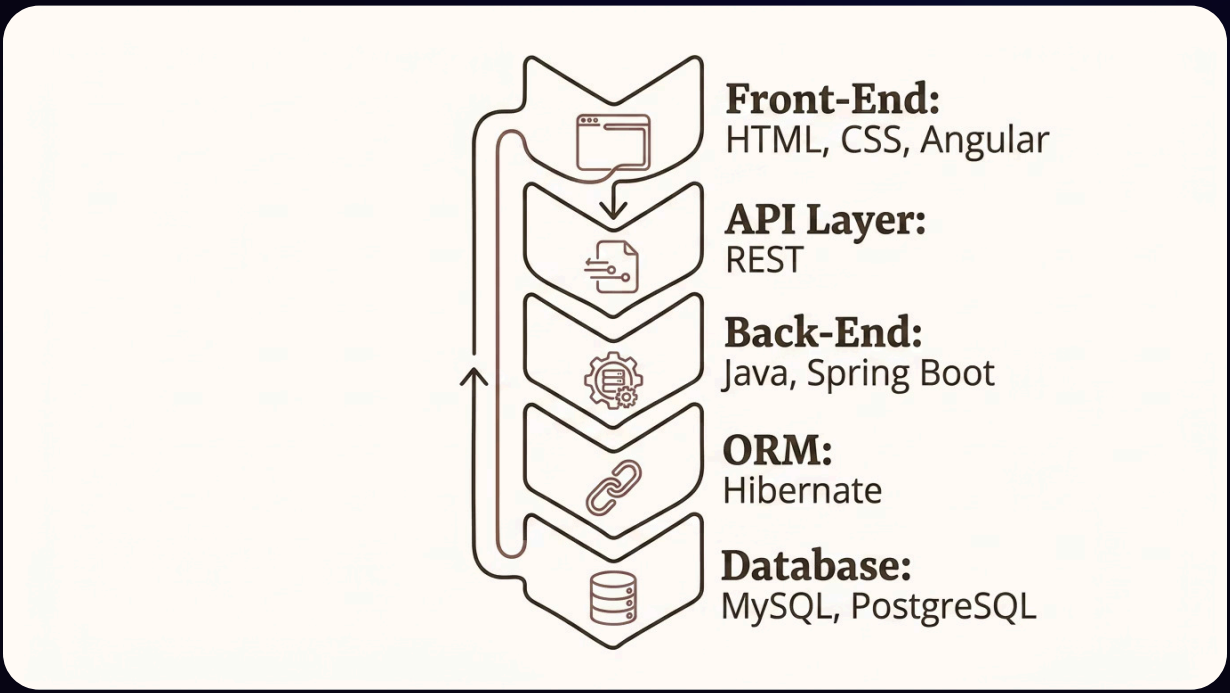
CI/CD pipelines and cloud infrastructure

A Java Full Stack Developer participates in every phase of the software development lifecycle — from initial architecture decisions through coding, testing, deployment, and maintenance. This holistic perspective ensures that technical choices made on the front end harmonize with back-end constraints, and vice versa.

2. Full Stack Architecture Overview

Understanding how all layers fit together is foundational for the CFSD exam. The table below maps each architectural layer to its primary technologies. Memorizing this stack will help you answer scenario-based questions about which tool belongs where.

Layer	Technologies
Front-End	HTML, CSS, JavaScript, Angular
Back-End	Java, Spring Boot
Database	MySQL, PostgreSQL
ORM	Hibernate
APIs	REST APIs
Build Tools	Maven, Gradle
Version Control	Git, GitHub
Deployment	Docker, Cloud Platforms



Each layer communicates with the adjacent layers through well-defined contracts — HTTP requests between front-end and API, JDBC/ORM between back-end and database. A solid understanding of this layered model is the mental framework that ties all other CFSD topics together.

3. Java Fundamentals Quick Notes

Java Core Features

- **Object-Oriented**

  Everything modeled as objects with state and behavior

- **Platform Independent**

  Write once, run anywhere via JVM bytecode

- **Secure**

  Built-in security manager and bytecode verification

- **Multi-threaded**

  Concurrent execution via Thread API

- **Robust**

  Strong type checking and exception handling

Primitive Data Types

Type	Example
`int`	10
`double`	10.5
`char`	'A'
`boolean`	true

Control Statements

Type	Example
Conditional	`if`, `else`, `switch`
Looping	`for`, `while`, `do-while`
Jump	`break`, `continue`

Java's platform independence stems from compilation to bytecode rather than native machine code. The JVM (Java Virtual Machine) interprets this bytecode at runtime on any supported operating system, making Java applications inherently portable. This characteristic, combined with the language's strict type system, is what has sustained Java's dominance in enterprise back-end development for decades.

4. Object-Oriented Programming (OOP)

OOP is the conceptual backbone of Java development and a major exam focus area. The four pillars — encapsulation, inheritance, polymorphism, and abstraction — are not just theoretical concepts; they shape how every Java application is architected. Understanding them deeply means being able to recognize design decisions in code snippets and justify architectural choices in open-ended questions.

Encapsulation

Data hiding — bundling data and methods, restricting direct access via private fields and public getters/setters.

Inheritance

Reusability — a child class extends a parent class, inheriting its fields and methods while optionally overriding them.

Polymorphism

Multiple forms — the same method name behaves differently based on the object type (overloading vs. overriding).

Abstraction

Hiding implementation — exposing only essential behavior through abstract classes and interfaces.

Key OOP Keywords

Keyword	Purpose
`this`	References the current object instance
`super`	References the parent class constructor or method
`final`	Declares a constant, non-overridable method, or non-extendable class
`static`	Belongs to the class, shared across all instances

5. Exception Handling

Exception handling is Java's structured mechanism for managing runtime errors without crashing the application. The try-catch-finally block is the core construct, and mastering it — along with the distinction between checked and unchecked exceptions — is essential for both the exam and real-world development. The CFSD exam often presents code snippets and asks whether a given exception is caught, propagated, or suppressed.

Exception Keywords

Keyword	Purpose
`try`	Block to monitor for exceptions
`catch`	Handles a specific exception type
`finally`	Always executes, even if exception is thrown
`throw`	Explicitly throws an exception
`throws`	Declares exceptions a method may throw

Common Runtime Exceptions

Exception	Cause
`NullPointerException`	Null object access
`ArithmeticException`	Divide by zero
`ArrayIndexOutOfBoundsException`	Invalid array index

ⓘ Checked exceptions must be declared or caught at compile time. Unchecked exceptions (RuntimeException subclasses) do not require explicit handling but should still be managed defensively.

6. Java Collections Framework

The Collections Framework is one of the most heavily tested topics in the CFSD exam. It provides a unified architecture for storing and manipulating groups of objects. Knowing which collection to use for a given scenario — ordered vs. unordered, duplicates allowed vs. not, key-value pairs vs. sequences — is a critical practical skill that maps directly to daily Java development work.

Key Interfaces & Implementations

Interface	Common Implementations
List	ArrayList, LinkedList
Set	HashSet, TreeSet
Map	HashMap, LinkedHashMap

Comparison Matrix

Collection	Ordered?	Duplicates?
ArrayList	Yes	Yes
HashSet	No	No
HashMap	Key-value	Keys unique

Java 8 Stream Features

Lambda Expressions

Write concise functional-style code with inline anonymous functions

Streams API

Chain operations like filter, map, and reduce on collections declaratively

Functional Interfaces

Single-method interfaces like Predicate, Function, and Consumer that power lambdas

Java 8 was a watershed release that introduced functional programming patterns into Java. Lambda expressions and the Streams API allow developers to process collections with minimal boilerplate, dramatically improving code readability and maintainability. CFSD candidates should be comfortable reading and writing basic stream pipelines.

7. JDBC — Java Database Connectivity

JDBC is Java's standard API for interacting with relational databases. Before ORM frameworks like Hibernate abstracted the details, JDBC was the primary mechanism for executing SQL from Java code. The CFSD exam tests your understanding of the JDBC connection lifecycle, the interfaces involved, and the SQL operations they enable. A strong foundation in JDBC also helps you understand what Hibernate is doing under the hood.

1. **Load Driver**
   Register the JDBC driver class with the DriverManager
2. **Create Connection**
   Obtain a `Connection` object using a JDBC URL, username, and password
3. **Prepare Statement**
   Use `PreparedStatement` to build parameterized SQL queries
4. **Execute Query**
   Call `executeQuery()` or `executeUpdate()` depending on operation type
5. **Process Results**
   Iterate the `ResultSet` to extract returned rows and columns
6. **Close Connection**
   Release all resources — `ResultSet`, `Statement`, `Connection` — in reverse order

Core JDBC Interfaces

Interface	Purpose
`Connection`	Represents the DB connection session
`Statement`	Executes static SQL statements
`PreparedStatement`	Parameterized, pre-compiled SQL
`ResultSet`	Cursor over query result rows

CRUD SQL Operations

Operation	Purpose
`SELECT`	Read / retrieve data
`INSERT`	Add new records
`UPDATE`	Modify existing records
`DELETE`	Remove records

8. Hibernate Framework

Hibernate is the most widely used ORM (Object Relational Mapping) framework in the Java ecosystem. It bridges the impedance mismatch between Java's object-oriented model and the relational model used by databases. Instead of writing verbose JDBC boilerplate, developers annotate their Java classes and let Hibernate generate the corresponding SQL automatically. This significantly accelerates development and reduces the risk of SQL-related bugs.

Key Advantages

Less Boilerplate

Auto-generates SQL from annotated POJOs

Simplified Operations

CRUD via `Session` API, no raw SQL required

Caching Support

First and second-level caches reduce DB round trips

DB Portability

Switch databases by changing dialect configuration

Core Annotations

Annotation	Purpose
`@Entity`	Marks class as a persistent entity
`@Table`	Maps class to a specific DB table
`@Id`	Designates the primary key field
`@Column`	Maps a field to a specific column

Entity Relationships

Relationship	Example
One-to-One	User → Profile
One-to-Many	Author → Books
Many-to-Many	Students ↔️ Courses

9. Spring Framework

The Spring Framework is the dominant enterprise Java framework, providing comprehensive infrastructure support for building robust, scalable, and maintainable applications. Its core innovation — Dependency Injection (DI) — fundamentally changes how Java objects are wired together, replacing manual object creation with container-managed dependencies. This results in loosely coupled, easily testable code.

Spring Core

IoC container and Dependency Injection — the foundation all other modules build upon

Spring MVC

Model-View-Controller framework for building web applications with clean separation of concerns

Spring AOP

Aspect-Oriented Programming for cross-cutting concerns like logging, transactions, and security

Spring Security

Comprehensive authentication and authorization for Spring-based applications

Types of Dependency Injection

1. **Constructor Injection**

   Dependencies passed via constructor — preferred for mandatory dependencies

2. **Setter Injection**

   Dependencies set via setter methods — suitable for optional dependencies

3. **Field Injection**

   Direct field injection via `@Autowired` — convenient but harder to test

Spring Bean Scopes

Scope	Description
Singleton	One shared instance per Spring container (default)
Prototype	New instance created every time the bean is requested

10. Spring Boot Quick Notes

Spring Boot is an opinionated extension of the Spring Framework that eliminates the verbose XML configuration that historically made Spring applications tedious to set up. With auto-configuration, an embedded servlet container, and production-ready actuator endpoints, Spring Boot enables developers to go from zero to a running REST API in minutes. It is the de facto standard for modern Java back-end development and a central topic in the CFSD exam.

Rapid Development

Spring Initializr and starter dependencies eliminate configuration overhead and get you building faster

Embedded Server

Ships with embedded Tomcat or Jetty — no external server installation required

Auto Configuration

Intelligently configures beans based on classpath dependencies using `@SpringBootApplication`

Production Ready

Built-in Actuator endpoints expose health checks, metrics, and environment info out of the box

Spring Boot Layered Architecture

Layer	Responsibility
Controller	Receives and routes HTTP requests
Service	Implements business logic
Repository	Handles database access via JPA/Hibernate

REST API Methods & Status Codes

Method	Purpose	Status
GET	Fetch data	200 OK
POST	Create data	201 Created
PUT	Update data	200 OK
DELETE	Remove data	200 / 204

⚠ Status 400 = Bad Request, 404 = Not Found, 500 = Internal Server Error. These are common exam distractors — know them cold.

11. Spring Security Basics

Security is a non-negotiable requirement for any production application, and Spring Security is the standard solution in the Spring ecosystem. The CFSD exam tests your ability to distinguish between authentication (who are you?) and authorization (what are you allowed to do?), as well as your familiarity with modern security patterns like JWT-based stateless authentication and role-based access control.

Authentication

Verify identity — confirming that a user or service is who they claim to be, typically via credentials like username and password or a token.

Authorization

Verify permissions — once identity is confirmed, determining what resources and operations the authenticated principal is allowed to access.

Common Security Features in Spring Applications

JWT Authentication

Stateless token-based auth — the server issues a signed JSON Web Token that the client sends with each request, eliminating session state on the server.

Role-Based Access Control

Assign roles like ADMIN or USER and restrict endpoints using `@PreAuthorize` or `HttpSecurity` configuration rules.

Password Encryption

Never store plaintext passwords. Use BCrypt hashing via Spring's `PasswordEncoder` to securely store and verify credentials.

CSRF Protection

Cross-Site Request Forgery tokens prevent malicious sites from submitting requests on behalf of authenticated users without their knowledge.

12. Front-End Development Basics

A Java Full Stack Developer must be equally conversant on the client side. While Java powers the back end, HTML, CSS, and JavaScript are the foundational trinity of every web user interface. The CFSD exam covers the core role of each technology and how they interact to produce the browser-rendered experiences users interact with daily.

HTML Key Elements

Element	Purpose
`<form>`	Captures user input and submits data
`<table>`	Displays tabular / structured data
`<div>`	Generic block-level section container

CSS Layout Concepts

Concept	Purpose
Flexbox	One-dimensional layout along a single axis
Grid	Two-dimensional responsive grid system
Media Queries	Apply styles based on screen breakpoints

JavaScript Core Concepts

DOM Manipulation

Access and modify HTML elements programmatically using `document.querySelector` and event listeners

Events

Respond to user interactions like clicks, key presses, and form submissions via event handlers

Functions

Encapsulate reusable logic; JavaScript supports first-class functions and closures

JSON

JavaScript Object Notation — the universal data exchange format between front-end and REST APIs

13. TypeScript Quick Notes

TypeScript is a statically typed superset of JavaScript developed by Microsoft. It compiles down to plain JavaScript and runs anywhere JavaScript runs, but adds a powerful type system that catches entire categories of bugs at compile time rather than at runtime. Angular — the front-end framework covered in the CFSD curriculum — is built with TypeScript, making TypeScript literacy a prerequisite for Angular development.

Static Typing

Declare variable types explicitly — TypeScript flags type mismatches before code even runs, eliminating entire classes of runtime errors

Better Maintainability

Type annotations serve as inline documentation, making large codebases dramatically easier to navigate and refactor safely

Improved Scalability

Interfaces and type contracts enforce consistency across large teams and multi-module applications

Enhanced IDE Support

TypeScript enables rich autocompletion, inline documentation, and intelligent refactoring in VS Code and other editors

Core TypeScript Types

Type	Example Value	Notes
`string`	"Java"	Text data, use single or double quotes
`number`	100	Covers both integer and floating-point values
`boolean`	true	true / false logical flags
`any`	anything	Disables type checking — use sparingly

14. Angular Framework

Angular is Google's opinionated, full-featured TypeScript-based front-end framework for building single-page applications (SPAs). It provides a complete solution out of the box — routing, state management, HTTP client, form handling, and dependency injection — making it the natural front-end counterpart to Spring Boot in a Java Full Stack architecture. CFSD candidates should understand Angular's component-based architecture and its most important built-in features.

Component

The fundamental building block. Each component owns a TypeScript class (logic), HTML template (view), and CSS styles (presentation).

Service

Shared business logic or data fetching that can be injected into any component. Keeps components focused on presentation.

Module

Groups related components, services, and directives into a cohesive unit. `AppModule` is the root module of every Angular app.

Angular Key Features

- **Two-Way Data Binding**

  Synchronizes model and view automatically via `[(ngModel)]`

- **Dependency Injection**

  Angular's built-in DI system provides services to components seamlessly

- **Routing**

  `RouterModule` enables SPA navigation without full page reloads

- **HTTP Client**

  `HttpClientModule` provides Observable-based REST API calls

- **Reactive Forms**

  Form-driven validation using `FormGroup` and `FormControl`

Lifecycle Hooks

Hook	Purpose
`ngOnInit()`	Runs after component is initialized — ideal for data fetching
`ngOnDestroy()`	Runs before component is destroyed — use for cleanup and unsubscribing

15. API Integration

API integration is the connective tissue that binds front-end and back-end layers together in a full stack application. In modern web architectures, the Angular front end communicates with the Spring Boot back end exclusively via RESTful HTTP APIs, exchanging data in JSON format. Understanding the API contract — the agreed-upon endpoints, HTTP methods, request/response structures, and error codes — is a fundamental full stack skill.

1

Client

Angular front-end sends HTTP requests with JSON payloads to the API layer

2

API Layer

Spring Boot REST controllers receive requests, route to services, and return JSON responses

3

Server / DB

Business logic executes, Hibernate queries the database, results flow back up the chain

Sample JSON Payload

JSON (JavaScript Object Notation) is the universal data exchange format. Every REST API interaction between Angular and Spring Boot involves JSON serialization and deserialization. Spring Boot uses Jackson under the hood to automatically convert Java objects to JSON and vice versa.

```
{
  "name": "John",
  "role": "Developer",
  "active": true
}
```

Spring Boot's `@RestController` annotation combines `@Controller` and `@ResponseBody`, automatically serializing return values to JSON via Jackson. No manual conversion needed.

16. Microservices Basics

Microservices architecture represents a fundamental shift from monolithic application design. Instead of deploying a single large application, microservices decompose the system into small, independent services — each responsible for a specific business capability, deployable and scalable independently. This architectural style has become the dominant pattern for large-scale cloud-native applications and is increasingly tested in modern certification programs like the CFSD.

In contrast to a monolith where a single bug can bring down the entire application, microservices are isolated — a failure in the payment service, for example, does not affect the catalog service. This fault isolation, combined with the ability to scale individual services based on demand, makes microservices highly attractive for production systems with variable load patterns.

Scalability

Scale only the services under load rather than the entire application — dramatically more cost-efficient in production environments

Independent Deployment

Teams deploy their services on their own release cadences without coordinating with every other team in the organization

Faster Development

Small, focused services are easier to understand, build, test, and iterate on than large monolithic codebases

Fault Isolation

A failure in one service is contained and does not cascade to bring down the entire system

17. Build Tools — Maven

Maven is Java's most widely used build automation and dependency management tool. It standardizes the project structure, manages third-party library downloads from Maven Central, and automates the compilation-to-deployment pipeline through a defined lifecycle. Understanding Maven is essential for any Java developer — every Spring Boot project scaffolded via Spring Initializr uses Maven or Gradle by default, and the CFSD exam expects familiarity with both.

pom.xml — The Project Object Model

The `pom.xml` file is the heart of every Maven project. It declares:

- Project coordinates (groupId, artifactId, version)
- Dependencies and their versions
- Build plugins and their configuration
- Repository locations for dependency resolution

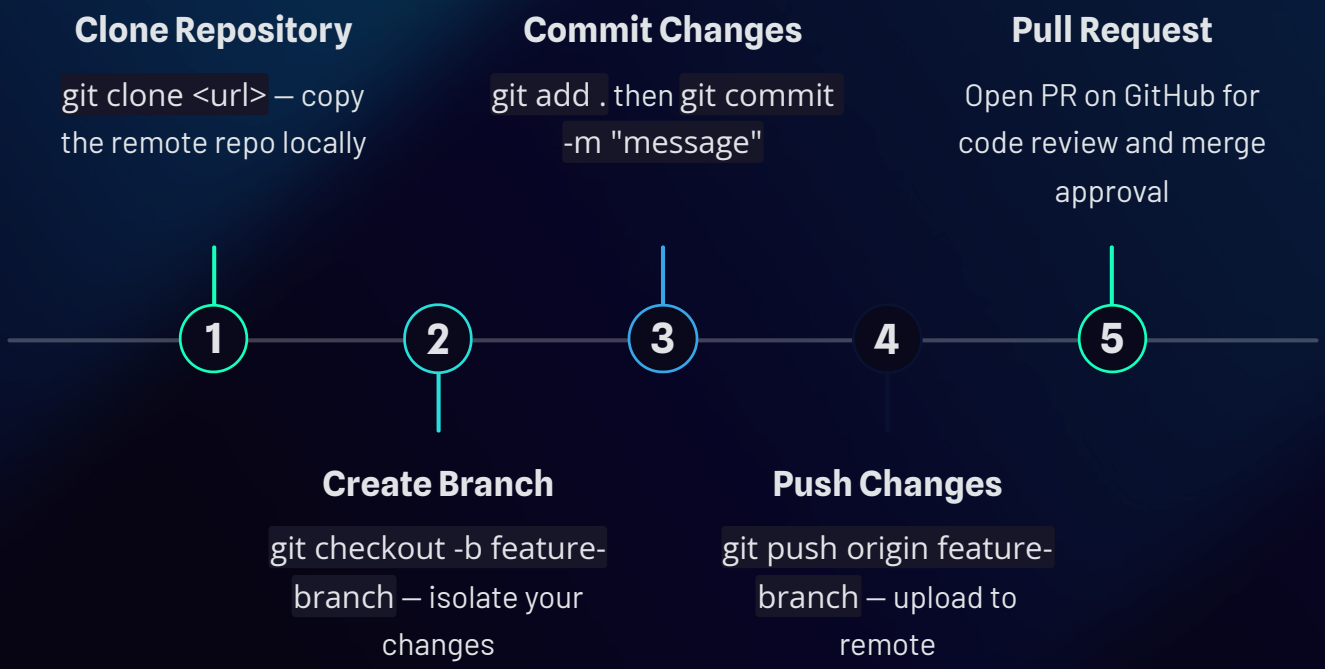
Adding a Spring Boot starter dependency in `pom.xml` is all it takes to pull in an entire curated set of compatible libraries.

Maven Build Lifecycle

Phase	Purpose
`validate`	Check project structure is correct
`compile`	Compile source code to bytecode
`test`	Run unit tests via Surefire plugin
`package`	Bundle into JAR or WAR artifact
`install`	Install artifact to local repository
`deploy`	Push artifact to remote repository

18. Git & Version Control

Git is the industry-standard distributed version control system, and GitHub is the most widely used platform for hosting Git repositories and enabling team collaboration. Every professional Java developer works with Git daily — branching, merging, reviewing pull requests, and resolving conflicts are routine tasks. The CFSD exam tests knowledge of the Git command workflow from initialization through to a merged pull request.



Essential Git Commands Reference

Command	Purpose
`git init`	Initialize a new local Git repository
`git clone <url>`	Copy an existing remote repository locally
`git add <file>`	Stage specific files for the next commit
`git commit -m ""`	Save staged changes with a descriptive message
`git push`	Upload committed changes to the remote branch
`git pull`	Fetch and merge changes from the remote branch
`git status`	Show working directory and staging area status

19. Testing Basics

Testing is not optional in professional software development — it is what distinguishes production-grade code from prototype-quality code. The CFSD exam expects candidates to understand the testing pyramid: unit tests form the base (fast, isolated, numerous), integration tests sit in the middle (verifying module interactions), and system tests sit at the top (end-to-end validation). A well-tested Spring Boot application relies heavily on JUnit 5 for unit tests and Spring Boot Test for integration testing.

Unit Testing

Test individual methods and classes in isolation using mock dependencies. Fast to run, easy to pinpoint failures. Primary tool: JUnit with Mockito.

Integration Testing

Test interactions between multiple modules — for example, verifying that a Spring Service correctly persists data through a Repository to the database.

System Testing

End-to-end tests that validate the complete application flow — from HTTP request through all layers to the database response — in a near-production environment.

JUnit 5 Core Annotations

Annotation	Purpose
`@Test`	Marks a method as a test case to be executed
`@BeforeEach`	Runs setup code before each individual test method
`@AfterEach`	Runs cleanup code after each individual test method
`@BeforeAll`	Runs once before all test methods in the class
`@AfterAll`	Runs once after all test methods in the class

Write tests first (Test-Driven Development) to clarify requirements before implementing code. Even if you don't practice strict TDD, having tests for every service method is a professional baseline that the CFSD exam strongly favors.

20. Cloud & Deployment Basics

Modern Java applications are deployed to cloud infrastructure rather than on-premise servers. Docker containerization and CI/CD pipelines have fundamentally changed the deployment experience — what once required manual server configuration now happens automatically through code. The CFSD exam covers deployment concepts at a conceptual level, expecting candidates to understand what Docker, CI/CD, and cloud platforms do and why they matter in a full stack context.

Docker — Containerization

Package your Spring Boot application and all its dependencies into a portable container image. A Docker container runs identically on any machine — from a developer's laptop to a production cloud server — eliminating "it works on my machine" problems entirely.

CI/CD — Automated Deployment

Continuous Integration / Continuous Deployment pipelines automatically build, test, and deploy code every time a developer pushes to the main branch. Tools like Jenkins, GitHub Actions, and GitLab CI are common in Java shops.

Cloud Hosting — Scalable Infrastructure

Deploy containerized Java applications to AWS (ECS, Elastic Beanstalk), Azure (App Service), or Google Cloud (Cloud Run) for on-demand scalability, managed databases, and global availability without owning physical hardware.

AWS

Amazon Web Services — the largest cloud provider, with EC2, RDS, S3, ECS, and Lambda among key services

Azure

Microsoft Azure — strong enterprise integration, Azure DevOps, App Service, and AKS Kubernetes

Google Cloud

GCP — leading AI/ML services, Cloud Run for containerized apps, and BigQuery for analytics

21. Full Stack Security Best Practices

Security must be built into a full stack application from the ground up — not bolted on as an afterthought. The CFSD exam covers both the defensive practices that protect applications and the common vulnerabilities that attackers exploit. Understanding both sides of the security equation enables developers to write code that is resilient against the OWASP Top 10 and other common attack vectors.

Security Best Practices Checklist

Practice	Purpose
Input Validation	Reject malformed or malicious data before it reaches business logic
HTTPS	Encrypt all data in transit using TLS certificates
JWT Tokens	Stateless, signed authentication tokens for securing REST APIs
Password Hashing	BCrypt one-way hashing — never store or transmit plaintext passwords

Common Vulnerabilities & Prevention

SQL Injection

Use `PreparedStatement` or JPA/Hibernate queries — never concatenate user input into SQL strings

XSS (Cross-Site Scripting)

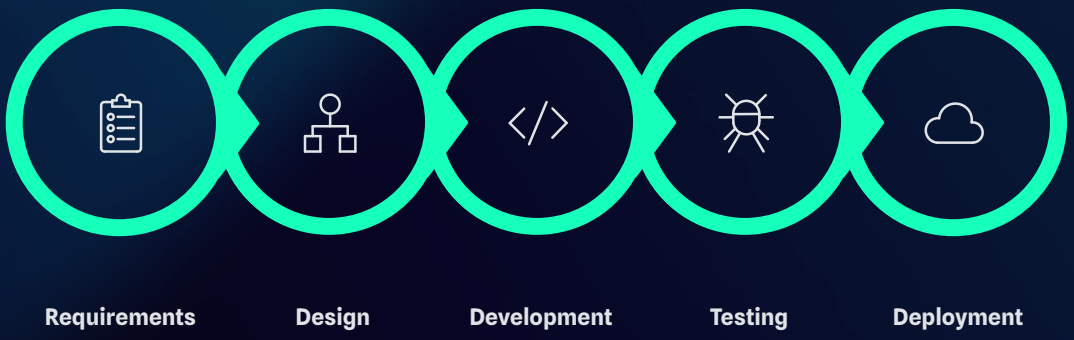
Sanitize and escape all user-generated content before rendering it in HTML templates

CSRF

Use security tokens to verify that state-changing requests originate from your own front end

22. Software Development Lifecycle (SDLC)

The Software Development Lifecycle is the structured process that governs how software is conceived, built, tested, deployed, and maintained. Every CFSD candidate should be able to name the SDLC phases in order and describe the primary activities within each. Understanding SDLC also contextualizes how Agile methodologies like Scrum fit into the broader development process.



The SDLC provides a predictable, repeatable framework that reduces project risk and improves team coordination. Each phase has defined inputs, outputs, and quality gates — for example, the Testing phase cannot begin until Development produces a deployable build, and Deployment cannot proceed until Testing sign-off is obtained.

Agile / Scrum Concepts

Concept	Purpose
Sprint	Time-boxed iteration (typically 2 weeks) for delivering working software
Scrum	Agile framework with defined roles: Product Owner, Scrum Master, Dev Team
Daily Stand-up	15-minute synchronization meeting: what did I do, what will I do, any blockers?
Backlog	Prioritized list of user stories and tasks to be implemented in future sprints

Agile does not replace the SDLC — it implements it iteratively. Each sprint cycles through a mini-SDLC: requirements (sprint planning) → design → development → testing → mini-deployment (sprint review). Understanding this connection is a common CFSD exam question type.

23. Full Stack Best Practices

Best practices represent the accumulated wisdom of the software engineering profession — patterns and habits that experienced developers have found consistently lead to maintainable, performant, secure, and scalable applications. For CFSD candidates, these practices are not just exam topics; they are the professional standards you will be expected to demonstrate on the job from day one. Internalizing these principles now pays dividends throughout an entire career.

Write Clean Code

Follow naming conventions, keep methods short and focused on a single responsibility, eliminate dead code, and write self-documenting code that colleagues can maintain without needing an explanation.

Use Layered Architecture

Strictly separate Controller, Service, and Repository layers. Each layer should have a single, well-defined responsibility, and dependencies should only flow downward — never skip layers.

Follow REST Standards

Use correct HTTP verbs, return appropriate status codes, version your APIs (`/api/v1/`), and document endpoints using Swagger/OpenAPI so consumers know exactly how to integrate.

Validate Inputs

Never trust data coming from the client. Use Bean Validation (`@NotNull`, `@Size`, `@Pattern`) in your DTOs and validate at both the controller layer and the service layer.

Handle Exceptions Properly

Use `@ControllerAdvice` to create global exception handlers. Return meaningful error responses with appropriate HTTP status codes rather than leaking stack traces to the client.

Secure APIs & Write Reusable Components

Apply JWT authentication and role-based authorization to every sensitive endpoint. On the front end, extract repeated UI logic into Angular services and reusable components to maintain the DRY principle across the entire stack.

> Clean code is not written by following a set of rules. You don't become a software craftsman by learning a list of heuristics. Professionalism and craftsmanship come from values that drive disciplines. — Robert C. Martin

GSDC
Global Skill Development Council

CERTIFIED JAVA FULL STACK DEVELOPER (CFSD)

ABOUT GSDC CERTIFICATION

EBOOK

Extensive and exclusive Ebook created by world's experts to help you with understanding core concepts.

LEARNING MATERIALS

Get access to learning materials such as videos, ebooks, templates, and practice exams, which will help you clear the certification exam.

CREATED BY EXPERTS

GSDC certifications are created and authored by world's leading experts in the field.

LEARNING OBJECTIVE

- Gain insights into autonomous decision-making processes
- Apply knowledge using ready-to-implement templates
- Demonstrate ability to work with Agentic AI models
- Validate your skills wit

Enroll now with the code **LEARN20** To avail **20%** discount

Enroll Now

www.gsdcouncil.org