**Chubb Tech Ready Program : Java Full-stack Track**

**Duration: 10 weeks × 3 days = 30 training days (240 hours total)  
Schedule: Thursday, Friday, Saturday (8 hours/day)**

**Program Structure**

**Week 1: Common Foundation (All Tracks)**

Purpose:Bridge college-to-corporate knowledge gap, establish fundamentals.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 1 | Enterprise Fundamentals ~~+ SRE Basics~~ | Git Fundamentals ~~Advanced~~  SQL Foundations  ~~Monitoring Tools (Prometheus, Grafana)~~ | Version control best practices  Database normalization  SQL joins, indexing, optimization  ~~Enterprise architecture concepts~~  ~~SRE principles & observability~~ | Git Commands  Git workflow simulation  Database design exercise  SQL optimization challenges  ~~Basic monitoring setup~~ | Gamified Quiz challenges – on each subtopics |
| Fri 2 | Development Best Practices ~~+ AI Enablement~~ | Clean Code Principles  Design Patterns  SOLID Principles  Testing Fundamentals  Code Reviews  ~~Prodigy GenAI Platform~~  ~~AI-Assisted Development~~ | Writing maintainable code  Common design patterns  Object-oriented principles  Unit vs integration testing  Code quality metrics  ~~AI-powered development workflows~~  ~~Prompt engineering for developers~~  ~~Code generation best practices~~ | Code refactoring exercises  Design pattern implementations  Test writing practice  Peer code review session  ~~Prodigy platform hands-on~~  ~~AI-assisted coding exercises~~ | Gamified Quiz challenges – on each subtopics |
| Sat 3 | Refreshers for Weekly topics  Microservices Architecture & REST API  ~~Chubb Ecosystem Deep Dive~~ | Monolithic vs Microservices    REST API Concepts  Postman  Database Design  ~~Insurance Fundamentals~~  ~~DuckCreek Platform~~  ~~Global Data Platform (GDP)~~  ~~Chubb Business Overview~~ | Architectures Overview  API testing methodologies  ~~P&C insurance concepts~~  ~~Policy lifecycle in DuckCreek~~  ~~Claims processing workflows~~  ~~GDP architecture & data flows~~  ~~Chubb's market position~~  ~~Risk assessment fundamentals~~  ~~Regulatory compliance basics~~ | API testing scenarios  ~~Insurance terminology quiz~~  ~~DuckCreek navigation lab~~  ~~GDP data exploration~~  ~~Policy workflow mapping~~  ~~Claims scenario analysis~~  ~~Risk calculation exercise~~ | **Assessment Day:**  Assessment 1: Git + SQL ~~+ SRE Basics~~ (30 pts)  Assessment 2: Code Quality + Testing ~~+ AI Tools~~ (30 pts)  Assessment 3: REST API + API Testing  ~~Chubb Domain + Platforms (30 pts)~~ |

**Week-1 Assignment:**

- Fork & clone a repo, simulate branching/merging conflicts.  
- Optimize SQL queries for performance & explain execution plans.

**Track Specialization- Java FSD (Weeks 2-10)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Track** | **Target Roles** | **Primary Technologies** | **Team Alignment** |
| **Java Track** | Backend Developers, Microservices Engineers | Java, Spring Boot, Reactive Programming, Angular | Java Platform Teams |

**JAVA-FSD TRACK (Weeks 2-10)**

**Week 2: Core Java Foundations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu  4 | Java Foundations | Core Java | Java Basics  Branching (if, switch)  Looping (while, for)  OOPs in Java  Classes, fields, methods, constructors  Keywords this, super & Modifiers  Interfaces & Inheritance | Setup Java development environment    Create a Student class with fields, constructors, and methods to display details.    Create an interface Payable and implement it in Employee and Contractor classes. | Gamified Quiz challenges – on each subtopics |
| Fri  5 | Java Foundations | Core java - Part 2 | Method overloading & Overriding  Abstract classes  Packages  Access modifiers  Exception Handling (try-catch, throws, custom exceptions)  Collections Framework (List, Set, Map, Queue)  Generics | Extend a base class Shape to Circle and Rectangle, override methods to calculate area and perimeter.  Create abstract class Employee with abstract method calculateSalary().Implement concrete subclasses FullTimeEmployee and Contractor. | Lab 1 : Implement a small project to store and manage employee/student data using collections.(15 pts) |
| Sat 6 | Java Foundations | Core Java - Part 3 | Memory management & Garbage collection  Comparable & Comparator  File Handling  MultiThreading | Monitor memory usage using Runtime and profiling tools.  Create List, Set, Map, and Queue objects, and perform add, remove, search, iteration.  Create generic classes, methods, and interfaces. | **Assessment Day:**  Weekly Topics-based Assessment |

**Week-2 Assignment:**

|  |
| --- |
| - Implement a Library Mgmt System (books, members, borrowing). - Build a CLI-based Student Grade Tracker (inheritance + collections). - Debug & fix an intentionally broken Java program. - Write custom exception hierarchy for business rules. |

|  |
| --- |
|  |

**Week 3 : Java 8 Foundations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 7 | Java 8 & latest | Java 8/11 – Part 1 | Intro to Functional programming  lambda expressions  Functional interfaces  Method references  Optional Class  Streams API  Data & Time API | Implement functional interfaces (Predicate, Consumer, Function) in small programs.  Demonstrate isPresent(), ifPresent(), orElse(), orElseGet(), orElseThrow().  Create a list of employees/students and perform stream methods | Gamified Quiz challenges – on each subtopics |
| Fri 8 | Java 8 & latest versions | Java 8/11/17 – Part 2 | Callable & Future Interfaces  Completable Future & Completion Stage  Brief intro Java 9-17 features  Collectors API & Immutable Collections  var keyword  File APIs  Switch expressions  Text blocks  Records  Logging: Log4j framework utilization | Create immutable collections using List.of(), Set.of(), [Map.of](http://map.of)().  Apply switch expressions with yield. | Lab 2 :  Build a Functional Employee Analytics System:  Filter employees based on department/salary using streams and lambdas. Safely handle optional fields like manager or email. Display reports with formatted dates and durations.  (15  points) |
| Sat 9 | Java Essentials | Maven/Gradle      IntelliJ IDEA    JUnit 5  Mockito | Build tool configuration    IDE productivity    Unit testing fundamentals    Mocking strategies | Build automation scripts        Write comprehensive unit tests | **Assessment Day:**  Weekly Topics-based Assessment |

**Week-3 Assignment:**

|  |
| --- |
| - Use streams to generate reports from a CSV dataset (e.g., sales data). - Implement async calls with CompletableFuture for API simulation. - Create a log parser app that uses lambdas & streams. - Refactor imperative code into functional-style Java. |

|  |
| --- |
|  |

**Week 4: Advanced Java & Spring Boot Foundation ~~+ SRE Integration~~**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 10 | Modern Java Essentials ~~+ Observability~~ | Java 17/21 LTS  Maven/Gradle  IntelliJ IDEA  JUnit 5  Mockito  ~~Micrometer~~  ~~OpenTelemetry~~ | Modern Java features (Records, Switch expressions)  Build tool configuration  IDE productivity  Unit testing fundamentals  Mocking strategies  ~~Application metrics & tracing~~ | Setup Java development environment  Practice modern Java syntax  Write comprehensive unit tests  Build automation scripts  ~~Implement observability in Spring Boot~~ | Lab 1: Java Development Setup ~~+ Observability~~ (25 pts) |
| Fri 11 | Spring Boot Core ~~+ GDP Integration~~ | Spring Boot 3.2+  Spring Core  Spring Data JPA  PostgreSQL  ~~H2 Database~~  ~~GDP Connectors~~ | Dependency injection deep dive  Auto-configuration mechanics  JPA entity relationships  Database configuration  Spring profiles  ~~GDP data access patterns~~ | Build comprehensive CRUD API  Configure multiple data sources  Implement complex entity relationships  ~~Environment-specific configurations~~  ~~Connect to GDP APIs~~ | Lab 2: Spring Boot CRUD ~~+ GDP~~ (35 pts) |
| Sat 12 | Exception Handling & TDD ~~+ DuckCreek API~~ | Spring Boot Validation  Custom Exceptions  Global Exception Handler  TestContainers  Spring Boot Test  ~~DuckCreek REST APIs~~ | Exception hierarchy design  @ControllerAdvice patterns  Custom validation annotations  TDD workflow in Spring  ~~Integration testing strategies~~  ~~DuckCreek policy operations~~ | Build comprehensive exception framework  Implement global error handling  Practice TDD workflow  Write integration tests  ~~Build DuckCreek integration layer~~ | Project 1: TDD-Built ~~DuckCreek Integration API~~ (45 pts) |

**Week-4 Assignment:**

|  |
| --- |
| - Develop a CRUD API for Inventory Management with SQL. - Write unit + integration tests for a user authentication module. - Compare memory/performance of Java 11 vs Java 17 code. |

|  |
| --- |
|  |

**Week 5: Spring WebFlux & Reactive Programming + AI Integration**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 13 | Reactive Programming ~~+ Prodigy Integration~~ | Spring WebFlux  Project Reactor  Reactive Streams  ~~Netty Server~~  ~~WebClient~~  ~~Prodigy SDK~~ | Reactive programming concepts  Mono vs Flux operations  Non-blocking I/O  Reactive HTTP clients  Error handling in streams  AI service integration patterns | Convert REST API to reactive  Implement reactive HTTP clients  Handle reactive error scenarios  Performance comparison tests  ~~Integrate Prodigy for code analysis~~ | Lab 3: Reactive API + AI Integration (35 pts) |
| Fri 14 | Advanced Reactor Patterns ~~+ SRE Monitoring~~ | Project Reactor Advanced  Custom Operators  Backpressure  Schedulers  Hot/Cold Streams  ~~Prometheus Java Client~~  ~~Custom Metrics~~ | Backpressure handling strategies  Custom operator creation  Threading models  Stream temperature concepts  ~~Performance optimization~~  ~~Business metrics collection~~ | Handle backpressure scenarios  Create custom reactive operators  Implement retry/timeout patterns  Optimize reactive performance  ~~Build comprehensive metrics dashboard~~ | Lab 4: Advanced Reactive ~~+ SRE Metrics~~ (40 pts) |
| Sat 15 | MongoDB Reactive ~~+ GDP Analytics~~ | Spring Data MongoDB Reactive  MongoDB 7.0+  Aggregation Pipeline  Change Streams  ~~GridFS~~  ~~GDP Analytics APIs~~ | Document database modelling  Reactive MongoDB operations  Complex aggregation queries  ~~Real-time data streaming~~  ~~File storage patterns~~  ~~Analytics pipeline integration~~ | Design NoSQL document schemas  Build reactive MongoDB repositories  Implement aggregation pipelines  ~~Create real-time data streams~~  ~~Build GDP analytics dashboard~~ | Project 2: Reactive ~~GDP Analytics Service~~ (45 pts) |

**Week-5 Assignment:**

|  |
| --- |
| - Convert a blocking e-commerce order API to reactive. - Implement real-time stock price streaming with WebFlux. - Build retry/recovery flows for failed API calls in reactive pipelines. |

|  |
| --- |
|  |

**Week 6: Microservices & Spring Cloud ~~+ SRE Production Readiness~~**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 16 | Microservices Architecture ~~+ SRE Design~~ | Spring Boot Microservices  Service Discovery  ~~Load Balancing~~  ~~Circuit Breaker~~  ~~Distributed Tracing~~  ~~SLI/SLO Definition~~ | Microservices design principles  Service communication patterns  ~~Fault tolerance strategies~~  ~~Distributed system challenges~~  ~~Observability patterns~~  ~~Reliability engineering~~ ~~principles~~ | Design microservices architecture  Implement service communication  ~~Add circuit breaker patterns~~  ~~Configure distributed tracing~~  ~~Define service SLIs/SLOs~~ | Lab 5: Microservices ~~+ SRE Design~~ (35 pts) |
| Fri 17 | Spring Cloud Ecosystem ~~+ Production Monitoring~~ | Spring Cloud Gateway  Eureka Discovery  Config Server  ~~Sleuth + Zipkin~~  ~~Actuator~~  ~~ELK Stack~~  ~~Application Performance Monitoring~~ | API Gateway patterns  Service registry  External configuration  ~~Request tracing~~  ~~Health monitoring~~  ~~Log aggregation & analysis~~ | Build API Gateway  Configure service discovery  Externalize configuration  ~~Implement health checks~~  ~~Set up comprehensive monitoring~~ | Lab 6: Spring Cloud ~~+ Production Monitoring~~ (40 pts) |
| Sat 18 | Event-Driven Architecture ~~+ Insurance Events~~ | Apache Kafka  RabbitMQ  Spring Cloud Stream  Event Sourcing  ~~CQRS Patterns~~  ~~DuckCreek Event Streams~~ | Event-driven architecture  Message broker patterns  Event sourcing concepts  Command query separation  ~~Saga patterns~~  ~~Insurance event processing~~ | Implement event-driven flows  Build message producers/consumers  Design event sourcing system  ~~Create CQRS implementation~~  ~~Process DuckCreek events~~ | Project 3: ~~Insurance~~ Event-Driven Microservices (50 pts) |

**Week-6 Assignment:**

|  |
| --- |
| - Build a microservices-based Banking app (Account + Transaction + Notifications). - Implement service registry & discovery using Eureka. |

|  |
| --- |
|  |

**Week 7: Advanced Java & Performance + AI-Enhanced Development**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 19 | Advanced Java Features + AI Code Enhancement | Java 17+ Features  Virtual Threads  Pattern Matching  Records & Sealed Classes  Modules (JPMS)  ~~Prodigy Code Assistant~~ | Modern Java syntax  Concurrency improvements  Type-safe patterns  Immutable data structures  Module system  AI-driven code optimization | Implement modern Java patterns  Use virtual threads  Build modular application  Performance comparisons  AI-assisted refactoring | Lab 7: Modern Java + AI Enhancement (35 pts) |
| Fri 20 | Performance Optimization ~~+ SRE Performance~~ | JVM Tuning  Profiling Tools  Memory Management  Caching Strategies  Database Optimization  ~~Performance SLI Monitoring~~ | JVM memory model  Garbage collection tuning  Application profiling  Redis/~~Hazelcast~~ caching  Query optimization  ~~Performance reliability metrics~~ | Profile application performance  Optimize JVM settings  Implement caching layers  Tune database queries  ~~Set up performance monitoring~~ | Lab 8: Performance ~~+ SRE Monitoring~~ (40 pts) |
| Sat 21 | Security & Production ~~+ Chubb Security Standards~~ | Spring Security  OAuth 2.0/JWT  API Security  Docker  Kubernetes Basics  ~~Chubb Security Framework~~ | Authentication/authorization  Token-based security  API protection strategies  Containerization  Container orchestration  ~~Enterprise security compliance~~ | Implement JWT security  Secure microservices  Containerize applications  Deploy to Kubernetes  ~~Implement Chubb security standards~~ | Project 4: Production-Ready Secure Insurance API (50 pts) |

**Week-7 Assignment:**

|  |
| --- |
| - Implement caching with Redis for frequent DB queries. - Secure a REST API with JWT tokens & refresh token strategy. - Dockerize a microservice and deploy on Kubernetes locally. |

|  |
| --- |
|  |

**Week 8: Web Skills & Angular Fundamentals**

| **Day** | **Focus Area & Topics Covered** | **Hands-on Labs / Activities** | **Assessment & Learning Outcomes** |
| --- | --- | --- | --- |
| Thu 22:  Web Foundations (HTML & CSS) | Introduction to Web Development - Client-Server Model - Role of Frontend, Backend, Database | Walkthrough: Browser–Server communication | Outcome: Understand how web apps work end-to-end |
| HTML Basics - Structure (head, body, tags) - Forms, input elements - Semantic HTML | Lab: Build a registration form | Short quiz on HTML elements |
| CSS Fundamentals - Selectors, box model - Flexbox & Grid layouts - Colors, fonts, spacing | Lab: Style the registration form with modern layout | Peer review of styled form |
| Responsive Design - Media queries - Mobile-first approach | Lab: Make the form responsive across devices | Practical assessment: Responsive webpage |
| Mini Project | Personal Portfolio webpage (HTML+CSS) | Outcome: Ability to create responsive web pages |
| Fri 23: JavaScript Essentials | Core JavaScript - Variables, operators, datatypes - Functions & scope | Lab: Simple Calculator | Quiz on JS basics |
| DOM Manipulation - Selecting elements - Event handling - Modifying DOM dynamically | Lab: Interactive form validation | Demonstration of validation |
| Objects & Arrays - JSON basics - Iteration methods - Using Fetch API for external data | Lab: To-do list app with add/delete functionality | Assessment: To-do app functionality |
| ES6+ Features - let/const - Arrow functions - Template literals - Array methods (map, filter) | Lab: Refactor To-do app with ES6+ | Code review |
| Mini Project Wrap-up | Consolidate JS exercises into one app | Outcome: Build interactive webpages with JS |
| Sat 24: Angular Foundations (Part 1) | Angular Intro & Setup - Angular CLI - Project structure & files | Lab: Initialize Angular project | Short quiz on Angular basics |
| Components & Templates - Creating components - Interpolation & data binding | Lab: Create Employee component | Hands-on check |
| Directives & Pipes - \*ngIf, \*ngFor - Built-in & custom pipes | Lab: Employee list with filters | Demo submission |
| Services & Dependency Injection - Service creation - Sharing data between components | Lab: Central Employee Service |  |
| Mini Project | Build Employee Directory App with components + services | Outcome: Comfort with Angular component-driven development |

**Week 8: Assignment**

**Part 1: Web Foundations**

* Create a **multi-page website** with the following:
  + **Home Page** → Introduction + navigation bar
  + **Registration Page** → Student registration form (name, email, course, year, photo upload)
  + **About Page** → Information about the portal (use semantic HTML tags)
* Apply **CSS styling**:
  + Use **Flexbox/Grid** for layout
  + Ensure **mobile responsiveness** using media queries
* Deliverable → A **responsive Student Portal UI (HTML+CSS)**

**Part 2: JavaScript Essentials**

* Add **JavaScript functionality** to the Student Portal:
  + **Form Validation** → Validate registration inputs (e.g., required fields, email format)
  + **Dynamic Student List** → Store submitted student data in an array and display it on the page
  + **Interactivity** → Ability to edit/delete students
  + **Persistence** → Use **localStorage** to save student data across sessions
  + **Bonus:** Fetch a random “Motivational Quote of the Day” from a free API and display on the Home page
* Deliverable → A **dynamic, interactive Student Portal (HTML+CSS+JS)**

**Part 3: Angular Foundations**

* Rebuild the Student Portal in **Angular**:
  + Create components:
    - student-list (shows all students)
    - student-detail (view/edit a single student)
    - student-form (add new student)
  + Use **directives** (\*ngIf, \*ngFor) to display student data dynamically
  + Create a **custom pipe** to filter students by course/year
  + Implement a **central service** to manage student data (mock array in memory)
* Deliverable → A **modular Angular Student Management App**

**Evaluation Criteria (Rubric):**

1. **Functionality (40%)** – Features implemented correctly (forms, validation, CRUD, filtering, responsiveness)
2. **Code Quality (20%)** – Clean, modular code with proper comments & ES6/Angular best practices
3. **UI/UX (20%)** – Responsive design, intuitive navigation, user-friendly interface
4. **Completeness (20%)** – Meets all requirements, works end-to-end

**Week 9: Angular (Part 2) + Spring Boot Integration**

| **Day** | **Focus Area & Topics Covered** | **Hands-on Labs / Activities** | **Assessment & Learning Outcomes** |
| --- | --- | --- | --- |
| Thu 25: Angular (Part 2) + Spring Boot Backend | Routing & Forms - Router module - Template-driven vs Reactive Forms | Lab: Add navigation & employee add/edit form | Routing quiz |
| Spring Boot Basics - Controllers, Services, Repositories - Project setup (Spring Initializr) | Lab: Create REST API for Employees | Test with Postman |
| CRUD with Spring Data JPA - Entity creation - Save, update, delete, retrieve | Lab: Employee API with DB persistence (H2/MySQL) | Postman testing submission |
| Integration Angular + Spring Boot - Angular HttpClient - Fetch data from backend | Lab: Connect Angular frontend to Spring Boot API | Build working REST APIs and connect Angular frontend |
| Fri 26: Full Stack Integration & Capstone Project Assignment to Participants | API Consumption in Angular - GET, POST, PUT, DELETE requests - Error handling & CORS | Lab: Connect frontend forms with backend CRUD | Debugging check |
| End-to-End Workflow - Employee Management (Add/View/Edit/Delete) - DB connectivity | Lab: Fully functional Employee App | Demo |
| Deployment Basics - Run full stack app locally - Testing & debugging techniques | Lab: Package & run Angular + Spring Boot app | Q&A check |
| Project task - Full Stack Employee Management System (Angular + Spring Boot + DB) - Team-based build & presentation | Final project demo & presentation | Deploy a full stack web app with frontend + backend + DB |
| Sat 27: Product Engineering with Java & Agile Principles | Introduction: Product Engineering  Understanding Users, Business Goals & Value Creation  Problem-Solving vs. Feature-Building  Designing Products for Scale, Reliability & Maintainability  Quality & Security as Core Product Principles  Collaboration with Product Managers, Designers & Stakeholders  Continuous Feedback, Iteration & Product Evolution.  ● Agile Methodology- Agile principles and methodologies; Agile frameworks such as Scrum and Kanban; sprint planning, daily stand-ups, sprint reviews, and retrospectives | Build a simple login API.  First as a feature (just code). Then as a product (security, validations, error handling, logging).  Compare both implementations | Gamified Quiz challenges – on each subtopics |

**Week-9 Assignment:**

|  |
| --- |
| - Re-engineer a simple feature into a productized version (e.g., from plain login → secure login with validations/logging). - Use Copilot to refactor legacy Java code into modern Java. - Compare Copilot vs manual coding productivity with a mini-task. - Write product design doc for a feature (API spec, user stories, test plan). |

|  |
| --- |
|  |

**Week 10: Final Capstone Project & Presentation**

* Full-stack insurance platform using Java ecosystem
* Multi-service architecture
* AI-enhanced features throughout the application
* Production-ready deployment with fundamental monitoring

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Focus Area** | **Technologies** | **Topics Covered** | **Hands-on Labs** | **Assessment** |
| Thu 28 | Capstone Project | Java FSD, with Angular | * Doubts clarification | * Learner-specific review | * Learner-specific review |
| Fri 29 | Capstone Project | Java FSD, with Angular | * Doubts clarification | * Learner-specific review | * Learner-specific review |
| Sat 30 | Capstone Project | Java FSD, with Angular | * Doubts clarification | * Learner-specific review | * Learner-specific review |

**Project variations - Sample Applications:**

**MealDB App**

Use Postman to hit the meals db api to query the data from: https[://www.themealdb.com/](http://www.themealdb.com/)  An API that fetches a list of meals from TheMealDb.com and returns a meal that

requires the least number of ingredients:

https[://www.themealdb.com/api/json/v1/1/search.php?s=Arrabiata](http://www.themealdb.com/api/json/v1/1/search.php?s=Arrabiata)

**Spotify App**

Clone the Spotify app. Users can able to listen to the songs, search songs, search album, search artists, follow artist, unfollow artists, create a playlist, update the playlist, add the song to playlist, remove the song from playlist and updateuser profile info.

Reference: https://developer.spotify.com/documentation/web-api/

**FreshDesk App-**

Clone the Freshdesk product. User can able to view a ticket, search the ticket, create a newticket, update the ticket, delete ticket, view a contact, search the contacts, create a new contact, update the contact, and delete a contact. Reference: https://developers.freshdesk.com/api/