**JetWayz**

**INTERNSHIP REPORT**

*Submitted by:*

**Patel Deep Nimeshbhai**

ENROLLMENT NO: 210280107060

*In partial fulfilment for the award of the degree of*

**BACHELOR OF ENGINEERING**

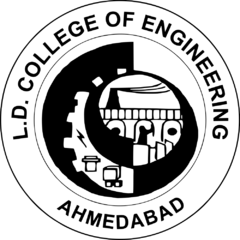
*In*

**Computer Engineering**

**L.D. COLLEGE OF ENGINEERING**

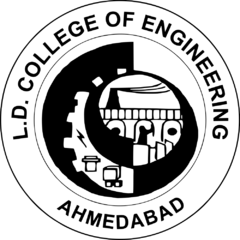
**AHMEDABAD-380015**

**GUJARAT**



**Gujarat Technological University,**

**Ahmedabad [April 2025]**

**L.D. College of Engineering**

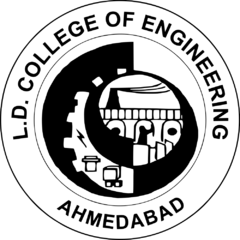
**120, Circular Road, University Area, Ahmedabad, Gujarat 380015.**

# CERTIFICATE

This is to certify that the project report submitted along with the project entitled “JetWayz’’ has been carried out by Gelatar Dharmesh Dhiraj lal under my guidance in partial fulfilment for the degree of Bachelor of Engineering in Computer Engineering, 8th Semester of Gujarat Technological University, Ahmedabad during the academic year 2024-25.

Prof. Nikunj Domadiya Dr. Chirag Thakkar

Internal Guide Head of Department

**L.D. College of Engineering**

**120, Circular Road, University Area, Ahmedabad, Gujarat 380015.**

# DECLARATION

We hereby declare that the Internship/Project report submitted along with the Internship/Project entitled “JetWayz’’submitted in partial fulfilment for the degree of Bachelor of Engineering in **Computer Engineering** to Gujarat Technological University, Ahmedabad, is a Bonafide record of original project work carried out by me at **OPL Pvt Ltd** under the supervision of **Deepak Vishwakarma** and no part of this report has been directly copied from any students’ reports or taken from any other source, without providing due reference.

Place: Ahmedabad

Date: 05 APRIL 2025

Name of the student Signature

**Patel Deep Nimeshabhai**

# ACKNOWLEDGEMENTS

I would like to show my deepest gratitude to **Dr. Nilay N. Bhuptani** , Principal, L.D. College of Engineering, **Dr. Chirag Thakkar**, Head of Department, Computer Engineering, Ahmedabad, for granting me permission to work as an intern in this esteemed IT company and for encouraging us to successfully complete our work.

I would like to express my indebtedness appreciation to our respected guide **Prof.** Pragnesh Patel**,** Assistant Professor, Computer Engineering, L.D. College of Engineering, Ahmedabad & **Mr. Deepak Vishwakarma**, **OPL Pvt. Ltd.** for his invaluable guidance and consistent support during the study. I am grateful to his detailed suggestions and to help clarifying concepts which helped me a lot during the course of this study.

I am grateful to all my **faculty members** of Computer Engineering department, L.D. College of Engineering, Ahmedabad, **my parents** and **my colleagues** for their support, special attention and motivation during the internship and training duration work.

# ABSTRACT

The theoretical foundation of software engineering education gains true value when applied in practical scenarios. Recognizing this principle, Gujarat Technological University mandates internships in the final semester to facilitate students' transition from academic knowledge to professional application. These internships serve as crucial bridges connecting classroom concepts to industry realities.

This report documents my comprehensive 12-week internship experience as an Associate Software Engineer Trainee at OPL Pvt. Ltd., Ahmedabad. During this period, I focused on developing "JetWayz," a sophisticated flight management system designed to streamline airline operations through an integrated digital platform. The system offers specialized interfaces for administrators, airline staff, and passengers, enabling efficient flight scheduling, booking management, and travel planning through an intuitive and responsive user experience.

The project leveraged a modern technology stack comprising Angular for frontend development, Spring Boot with Java for backend services, and MySQL for database management. Working within the MVC architectural pattern, my contributions spanned both frontend and backend development, including implementing secure RESTful APIs, developing robust exception handling mechanisms, and maintaining disciplined version control practices through Git.

This immersive experience provided invaluable insights into enterprise software development lifecycles, enhanced my technical and collaborative capabilities, and developed my adaptability in agile work environments. The internship strengthened my capacity to work both independently and as an effective team member while adhering to project timelines and deliverables.

This report represents a synthesis of academic theory and practical implementation, demonstrating how classroom knowledge transforms into scalable, real-world software solutions. The internship has been instrumental in not only reinforcing my technical foundation but also in shaping my professional identity as an emerging software engineer prepared for industry challenges.

# List of Figures

2.3 Office Environment……………………………………………………………….…………5

2.3 Celebrations...…...….…...……….………………………….………………….……………6

2.3 Travel & Celebrations……………………………………………….………….….………..6

3.1 Object Manager .……………………….…………….……………………….………….….11

3.1 Schema Builder ……………………………………………………………….…………….12

3.1 Flow 1………………………………………………………………………….………….....14

3.1 Flow 2………………………………………………………………………….………….....14

3.1 User Management Salesforce ……………………………………………….…………..…..15

3.1 User Management …………………………………………………………….………….….16

3.1 Salesforce Security …………………………………………………………….……………17

3.1 Security Architecture ………………………………………………………….………….…18

3.1 Permission Sets 1…………………………………………………………….………….…...19

3.1 Permission Sets 2…………………………………………………………….………….…...20

3.2 Developer Console…………………………………………………………….………….….22

3.2 Trigger Code …………………………………………………………….……………….….24

3.2 Visualforce Page……………………………………………………………….………….…25

3.2 LWC Component Structure ………………………………………………….………….…..26

3.2 LWC Code ……..…………………………………………………………….………….…..27

3.2 LWC Component 1…………………………………………………………….………….…27

3.2 LWC Component 2…………………………………………………………….………….…28

3.2 LWC Component 3…………………………………………………………….………….…28

3.2 Test Class………..…………………………………………………………….………….….29

3.2 Integration Code……………………………………………………………………………...30

3.3 Development Flow…………………………………………………………………………...32

3.3 Gearset Tool Org Connection …………………………………………………….………....34

3.3 Gearset Compare & Deploy…….………………………….………………….………….….35

3.3 Gearset Deployment History ……………………………………………….………….…....35

3.3 Gearset Rollback and Reports.……….…………….……………………….………….……36

3.3 Gearset Deployment Successful …………………………………………….………………36

3.3 Changeset Deployment Diagram Overview ………………………………….…………......37

3.3 Changeset Deployment Process Overview ...………………………………….………….....38

3.3 Outbound Changesets ……………………………………………………….…………..…..38

3.3 Inbound Changesets ……………………………………………………….……….…….….39

3.3 Data Import Steps …………………………………………………………….………….….40

3.3 Data Import Step 1………………………………………………………….………….…….40

3.3 Data Import Step 2………………………………………………………….………….….....41

3.3 Data Import Step 3………………………………………………………….………….….....41

3.3 Data Export Step 1…………………………………………………………….………….….42

3.3 Data Export Step 2 …………………………………………………….……………….…....43

4.5 Code Inspection Report ……………………………………………………….………….…46

4.6 Code Optimization Report …………………………………………………….…………….47

5.1 Spotify Search Bar………..…………………………………………………….……………50

5.1 Spotify Result…...………………………………………………………….………..………50

5.2 Create Metadata ………….…………………………………………………….……………51

5.2 Manage Metadata……………………………………………………………………………52

5.2 Get Custom Metadata ..………………………………………….………….……………….52

5.1 Dynamic Component Hepler.js ...…………………………………………….………….….53

5.2 Dynamic Component Component.cmp …………………………………….…….….……...54

# List of Tables

Job Profile Table………………………………………………………………………………....4

Software & Tools Table……………………………………………………………………….…8

# List of Abbreviations

# TABLE OF CONTENT

Certificate…………………………………………………………………..……………………...I

GTU Completion Certificate…………………………………………………..………………… II

Company Certificate………………………………………………………………..……………III

Declaration……………………………………………………………………………..…...……IV

Acknowledgement………………………………………………………………………….…….V

Abstract…………………………………………………………………………..……………... VI

List of Figures……………………………………………………………………….….………VII

List of Tables……………………………………………………………………………..……VIII

List of Abbreviations……………………………………………………..……………….……..IX

Table of Contents……………………………………………………………..…………….…….X

Chapter-1: About the company…………...………………………………………………….…...1

* 1. About……………………………………………………………………….….......1
  2. What we do? ………………………………………………………………………1
  3. Industries……………………………………………………………………...…...2
  4. Mission………………………………………………………………….…………2
  5. Vision………………………………………………………………………....…...2
  6. Website…………………………………………………………………………….2

Chapter-2: Internship Overview...……………………………………………………………...…3

2.1 Objectives………...………………………………………………………………..3

2.2 Job Description……...……………………………………………….…………….4

2.3 Work Environment….……………………………………………………………..5

2.4 Software/tools table………………………………………………………………..8

Chapter-3: Learning…………...…………………………………………………………………9

3.1 Programming Language Java 8 ………………….……………………………...9

3.1.1OOP Concepts...………………………………….………..…………...…10

3.1.2 Lambda Expressions...………………………….…..…………….............11

3.1.3Stream API...…………………………………….……….……………..…13

3.1.4 Functional Interfaces...………………………………..…….….............…15

3.1.5 Method References...…………………………………………………..…16

3.1.6 Collections & Generics...……………………………………..……..……18

3.1.6 Exception Handling...………………………………………………..……18

3.2 Backend Development–Spring Boot………………………………...….…….…20

3.2.1 Spring Boot Introduction……...……………..……….………….…..……21

3.2.2 REST API Development…...…..………………………………….…...…22

3.2.3 Dependency Injection (@Autowired)……………..…………….……..…23

3.2.4 Spring Data JPA………………….………………………….…..………..24

3.2.5 Microservices Architecture…………………………………….……...….25

3.2.6 Service Registration (Eureka)………………………………….……...….28

3.2.7 API Gateway (Spring Cloud Gateway)………………………..….…...….29

3.2.8 JWT Authentication & Authorization…………………………..……...….29

3.2.9 Exception Handling (@ControllerAdvice)……………….…….……...….29

3.2.10Application Properties Configuration…………….…………………...….29

3.3 Frontend Development – Angular.………...…………………..……………..…30

3.3.1 Angular Overview………………...………………..…..…….……......….33

3.3.2 **Project Structure** ……………………………………..………...……..….36

3.3.3 **Components & Modules** ………………………...….….……...……...….39

**3.3.4 Services and Dependency Injection** …………………………….………………34  
**3.3.5 Data Binding & Event Handling** ………………………….………………….…35  
**3.3.6 Routing & Navigation** ……………………………………………………….….36  
**3.3.7 Reactive Forms & Validation** …………………………………….……….……37  
**3.3.8 API Integration using HttpClient** …………………………………………….…38  
**3.3.9 Consuming JWT-Protected APIs** ……………………………………….………39

Chapter-4: Project – JetWayz-Flight Management System ......................................................... 40

4.1 Introduction & Objective ....................................................................................... 40

4.2 Tech Stack Overview ............................................................................................. 40

4.3 Project Features ...................................................................................................... 41

4.4 MVC Architecture ................................................................................................. 42

4.5 Folder Structure Explanation ................................................................................. 43

4.6 Authentication & Authorization ............................................................................ 45

4.7 Middleware and Utilities ....................................................................................... 46

4.8 Screenshots & Output Pages .................................................................................. 47

4.9 UML Diagrams ...................................................................................................... 48

Chapter-5: My Skills & Work……………………………………………...……………………55

6.1 Skills….….….….….….….….……….….………….………………………....…55

6.2 Work………………………………...………………………………………....…56

Chapter-6: Conclusion…………………………...………………………...……………………

Chapter-7: References……………………………………………………...…………………….

Chapter-8: Mentor’s Feedback.…………………………………………...………………….…



# 

# **OPL.Pvt.Ltd.**

## **About:**

* OPL is a pioneering digital infrastructure and financial technology company committed to transforming the financial ecosystem through innovation and technology. With a strong focus on simplifying credit access, streamlining financial services, and supporting MSMEs, corporates, and large financial institutions, OPL delivers tailor-made, high-impact solutions across the digital lending and fintech space..

## **What we do?**

* Digital Lending Platforms:  
  Revolutionizing credit accessibility with end-to-end, high-efficiency platforms like PSB Loans in 59 Minutes, enabling fast-tracked application processing, automated creditworthiness scoring, and instant approvals.
* Custom Tech Solutions for Financial Institutions:  
  Empowering banks, NBFCs, and corporates with white-label technology products that automate and optimize loan disbursement, enhance customer service, and boost operational scalability.
* MSME & Retail Credit Support:  
  Dedicated tools and platforms to enable growth for micro, small, and medium businesses, providing access to Mudra loans, home loans, personal loans, and auto financing with seamless digital journeys.
* Government & Social Sector Integration:  
  Building national-level portals like *JanSamarth.in* and *JanSuraksha* to support government-linked credit and insurance schemes, streamlining citizen access to welfare and empowerment programs.
* Data-Driven Intelligence & Predictive Analytics:  
  Using real-time data, advanced analytics, and proprietary models like *Fit Rank* to help lenders make smarter, faster, and more informed decisions, while driving innovation in credit assessment and risk management.

## **Industries:**

* Finance
* Banking
* Fintech
* RegTech
* Education
* MSMEs

## **Mission:**

* At OPL, our mission is to revolutionize the financial ecosystem by delivering innovative, accessible, and intelligent digital solutions. We aim to bridge the gap between institutions and individuals by streamlining processes, accelerating credit access, and empowering growth across all sectors through advanced technology.

## **Vision:**

* We envision a world where innovation drives inclusion. OPL is committed to becoming a global leader in digital finance by continuously evolving our platforms and solutions to empower financial institutions, support underserved communities, and shape the future of finance with transparency, speed, and impact.

## **Website:**

* <https://www.oplinnovate.com>

# Internship Overview

## **Objective:**

* **To build strong programming fundamentals** by gaining hands-on experience in Core Java and Java 8 concepts, which are essential for a successful career in Java development.
* **To develop and integrate RESTful APIs** using Spring Boot, enabling effective communication between the application and database layers—an essential component of modern backend development.
* **To understand and design relational databases** using MySQL, learning to manage different types of relationships and create efficient, scalable data models.
* **To gain practical knowledge of frontend development** by building dynamic and interactive user interfaces using the Angular framework, including concepts like server-side rendering and component-based architecture.
* **To learn microservices architecture** by modularizing application code into independently deployable services and building inter-service communication while managing multiple databases.
* **To implement robust security features** including user authentication and authorization, and securing APIs and API gateways to ensure data privacy and application integrity.
* **To experience real-world software development practices**, working collaboratively with a team of developers on full-stack projects, applying principles like MVC and modular coding.

## **Job Description:**

**Java Software Developer** [OPL Innovate]

**The role**

* Backend Developer:  
  Responsible for designing, developing, and maintaining server-side logic using Java and Spring Boot. Involved in building scalable APIs, integrating databases, and ensuring high performance and responsiveness of backend services.
* Full-Stack Contributor  
  Collaborates across the stack to support frontend integration using Angular, contributes to RESTful API development, and ensures smooth communication between client-side and server-side components.

**Key responsibilities**

* Strong knowledge of Core Java, OOPs Concepts, and Collections Framework
* Hands-on experience in building RESTful APIs using Spring Boot and managing relational databases with MySQL
* Familiarity with microservices architecture, including modular design and inter-service communication
* Proficient in writing clean, maintainable code with version control using Git and following MVC architecture

**Job details**

|  |  |  |
| --- | --- | --- |
| Java Software Developer | 5 days working etc. | 8.5 hours/day |
| Full-time | ~~Part-time~~ | Ashwamegh Elegance - 3, Opp. SBI Corporate Office, SM Road, Ambawadi, Ahmedabad - 380015 | Fresher |
| ~~Overtime~~ | ~~travel~~ | ~~evening/weekend work~~ |

[2.2.0.1 Job Profile Matrix]

## **Work Environment:**

* At OPL, we create an environment where innovation and collaboration thrive, empowering our employees to reach their full potential. We believe in crafting groundbreaking solutions that address the challenges of the financial world, from simplifying credit access to transforming digital lending systems. If you're passionate about using technology to drive meaningful change and overcome real-world challenges, OPL is the place for you. Join our team and be a part of the transformation, as we work together to revolutionize the future of finance. Explore our current openings, and take the first step towards an exciting career with OPL



[2.3.0.1 Office Environment]

* My Company values diversity and inclusion and has a diverse workforce, including employees from different backgrounds and cultures. My Company encourages open communication and feedback, and employees are encouraged to speak up and share their ideas.



[2.3.0.2 Celebrations ]

* My Company also celebrates various events, festivals and employee birthday celebration, food parties, yearly vacations etc.



[2.3.0.3 Travel & Celebrations ]

* My Company also provides opportunities for employees to grow and develop their skills through training programs, mentorship, and career advancement opportunities. My Company promotes work-life balance and offers flexible work arrangements, such as remote work options and flexible schedules.
* Overall, My Company work environment is positive, collaborative, and focused on growth and development, while also valuing employee well-being and work-life balance.

## **Software tool’s table:**

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Tool Name | Logo/Image | Use Case |
| 1. | Visual Studio Code |  | Visual Studio Code is a free coding editor that helps you start coding quickly. Use it to code in any programming language, without switching editors. Visual Studio Code has support for many languages, including Python, Java, C++, JavaScript, and more. |
| 2. | **STS (Spring Tool Suite)** to run your **Spring Boot** backend |  | For backend development, used **Spring Tool Suite (STS)** to run and manage **Spring Boot** applications. STS provided an integrated development environment tailored for building enterprise-level Java applications with Spring. It helped streamline backend development, REST API creation, and seamless integration with the frontend built using Angular. |
| 3. | GitHub |  | GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. |

[2.4.0.1 Software & Tools Table]

**3.1 Programming Language - Java 8**

* **Theoretical Foundation**

Java 8, released in March 2014, represents a paradigm shift in Java's evolution by introducing functional programming concepts to a traditionally object-oriented language. This update was driven by several key industry trends:

1. **Multicore Processing**: As CPU manufacturers shifted toward multicore architectures instead of faster clock speeds, programming languages needed better tools for concurrent programming.
2. **Big Data Processing**: The explosion of data volumes required more efficient ways to process large collections of information.
3. **Code Maintainability**: The increasing complexity of applications demanded more expressive, less verbose coding patterns.
4. **Competing Languages**: The rise of functional languages and frameworks was challenging Java's dominance in enterprise development.

Java 8 addressed these challenges by introducing functional interfaces, lambda expressions, the Stream API, and method references—features that fundamentally changed how Java code is written while maintaining backward compatibility. This update balanced innovation with stability, allowing developers to adopt functional programming techniques gradually without breaking existing systems.

The importance of Java 8 cannot be overstated; it represents the most significant change to the language since its inception, enabling more concise, readable, and maintainable code while providing better support for parallel processing and asynchronous programming patterns essential for modern applications.

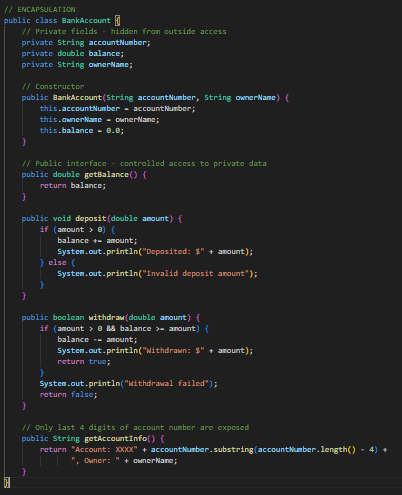
**3.1.1 OOP Concepts** 

* **Theoretical Foundation**

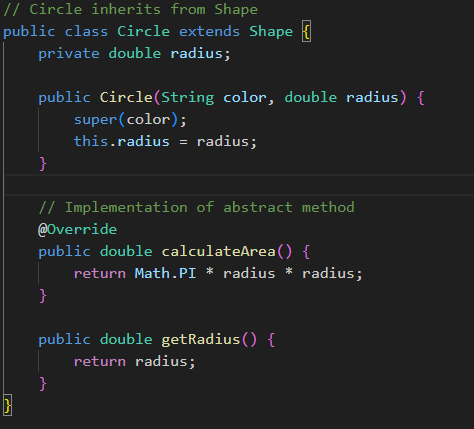
Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects" containing data and procedures. Java was designed as an OOP language from its inception, implementing four core principles that help manage complexity in software systems:

* **Key Principles:**

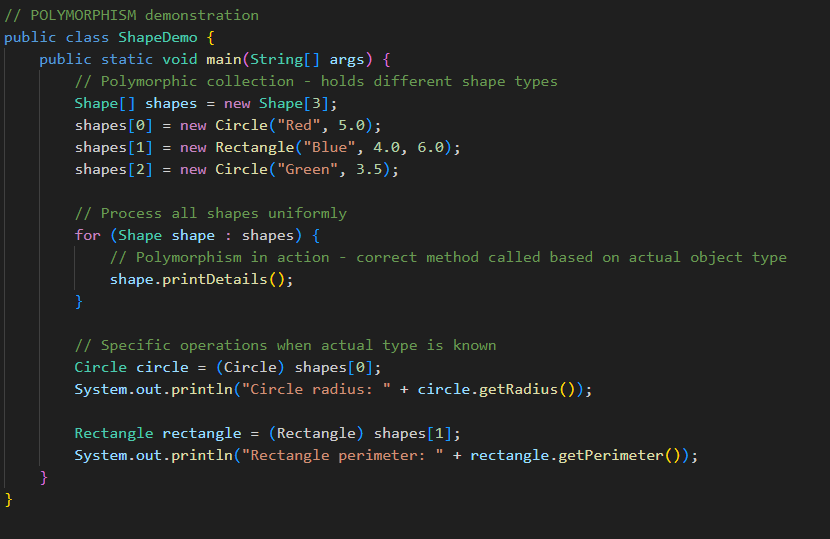
1. **Encapsulation**: Bundles data with methods that operate on that data, hiding internal state and requiring all interactions to occur through well-defined interfaces. This creates information hiding and reduces system interdependencies.



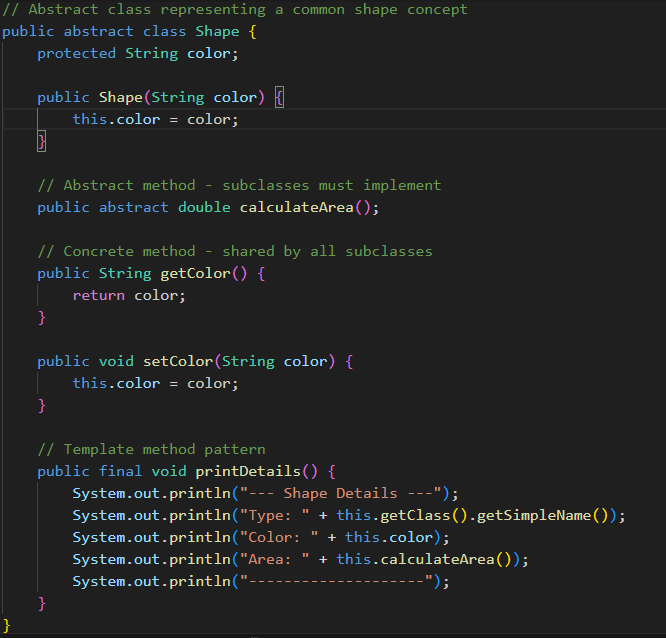
1. **Inheritance**: Enables a class to inherit properties and behaviors from a parent class, establishing "is-a" relationships that promote code reuse and logical hierarchies.



1. **Polymorphism**: Allows objects to be treated as instances of their parent class, enabling a single interface to represent different underlying forms. This creates flexibility through dynamic method binding.



1. **Abstraction**: Focuses on the essential qualities of an object rather than one specific implementation, allowing programmers to think at higher levels of conceptualization.



**3.1.2 Lambda Expressions**

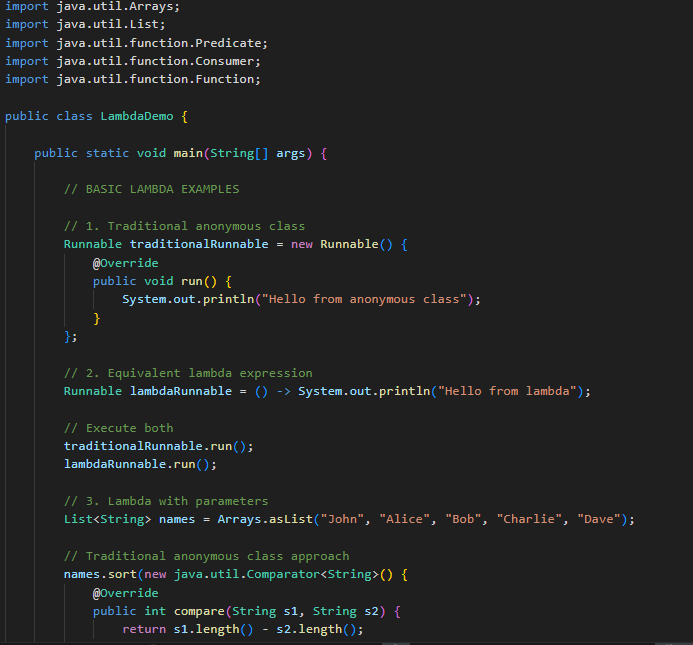
* **Theoretical Foundation**
* Lambda expressions fundamentally changed Java by introducing functional programming concepts to the language. They represent anonymous functions—code blocks that can be passed around as if they were data.
* At their core, lambda expressions are a more concise syntax for implementing single-method interfaces (functional interfaces). They allow developers to treat functionality as a method argument, enabling behavior parameterization without the verbose anonymous inner class syntax.
* Lambda expressions are based on lambda calculus, a formal mathematical system for expressing computation, developed by Alonzo Church in the 1930s. This system forms the theoretical foundation of functional programming languages.
* **How Lambda Expressions Work:**

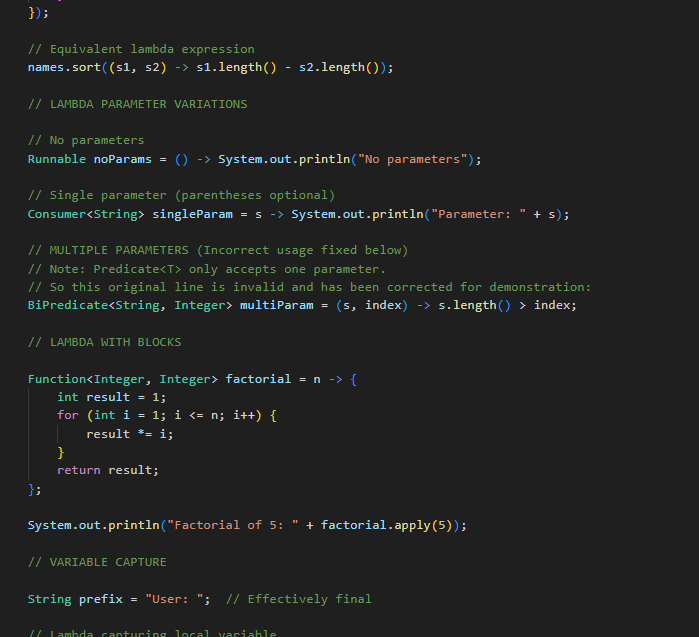
When you write a lambda expression in Java, the compiler:

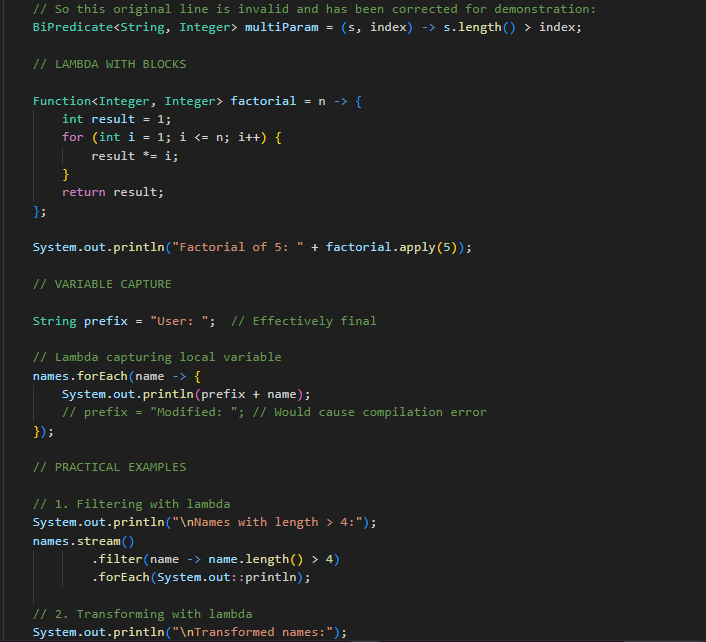
1. Determines the target functional interface based on context
2. Verifies compatibility with the interface's method signature
3. Creates an instance of the functional interface with the lambda body as the method implementation
4. Uses invokedynamic bytecode to defer the actual implementation strategy to runtime

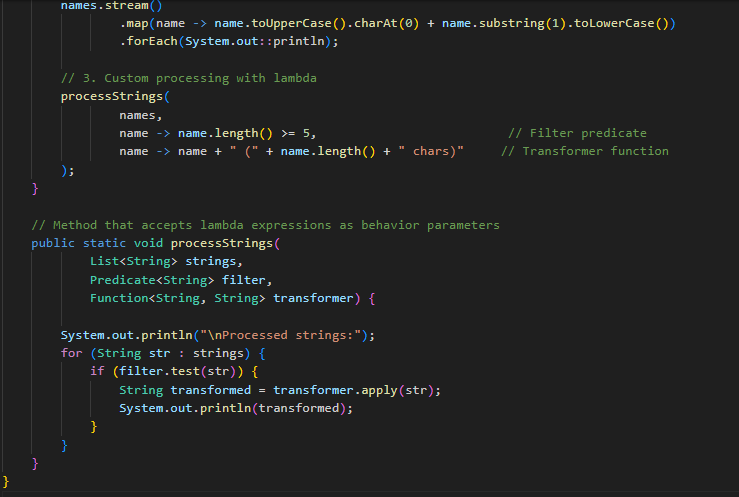
The JVM can optimize lambda execution through techniques like inlining and removing unnecessary object creation, often making lambdas more efficient than equivalent anonymous classes.

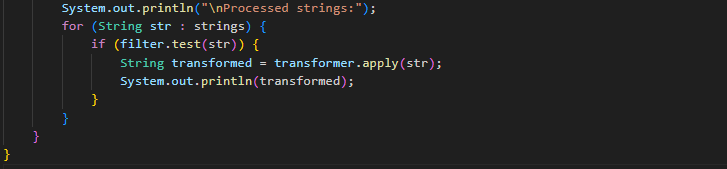
### **Program Example:**











## **3.1.3 Stream API**

### **Theoretical Foundation**

The Stream API represents a paradigm shift in how collections are processed in Java, introducing a functional approach to data manipulation. Unlike collections, which are data structures that store elements, streams are pipelines of computational operations that can transform data.

Conceptually, streams are built upon:

1. **Declarative programming**: Expressing what should be done rather than how
2. **Functional composition**: Building complex operations from simple ones
3. **Lazy evaluation**: Deferring execution until results are needed
4. **Data pipeline architecture**: Representing multi-step processing as a single flow

Streams leverage the functional interfaces introduced in Java 8 to accept behaviors as parameters, allowing transformations to be expressed as lambda expressions or method references.

#### **Stream Processing Model:**

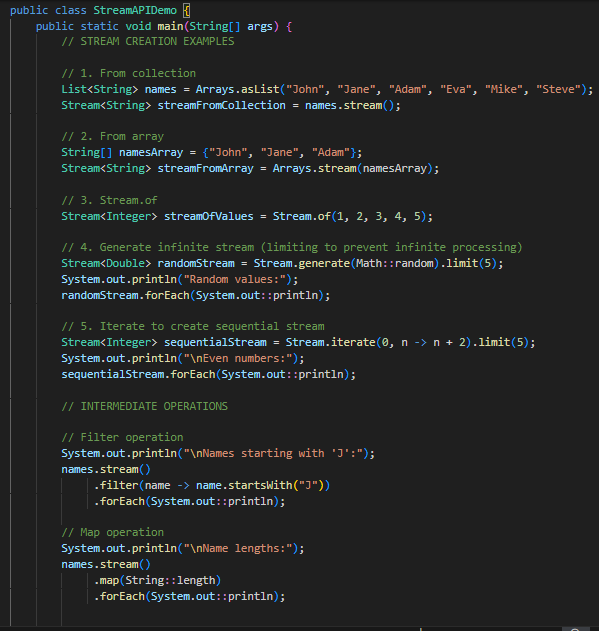
A stream pipeline consists of:

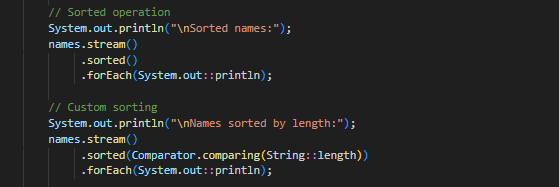
1. **Source**: A data origin (collection, array, generating function, I/O channel)
2. **Intermediate operations**: Transformations that produce new streams (filter, map, etc.)
3. **Terminal operation**: Final processing that produces a result or side effect

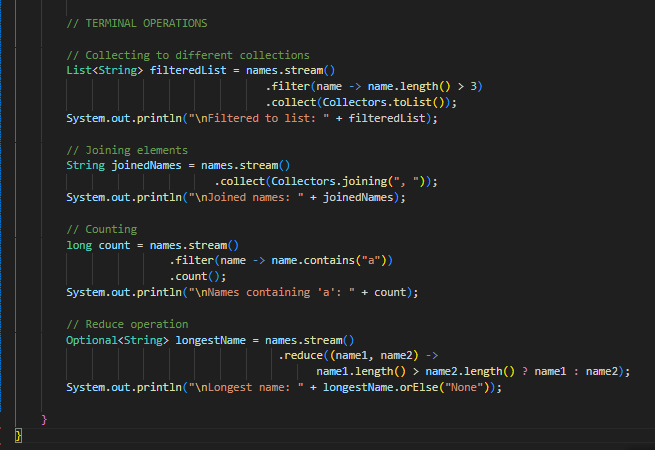
The key insight is that intermediate operations are lazy—they only set up the pipeline but don't process any elements until a terminal operation is invoked. This enables significant optimizations:

* **Short-circuiting**: Stopping processing once a result is determined
* **Loop fusion**: Combining multiple operations into a single pass
* **Parallelization**: Splitting processing across multiple threads automatically

### **Program Example:**







## **3.1.4 Functional Interfaces**

### Theoretical Foundation

* Functional interfaces serve as the cornerstone of Java's implementation of functional programming. A functional interface is an interface that contains exactly one abstract method, making it suitable as a target type for lambda expressions and method references.
* The concept bridges object-oriented and functional paradigms by using Java's existing interface type system to represent functions. This design choice was crucial for maintaining backward compatibility while introducing functional programming concepts.
* Functional interfaces can be viewed as "function types" in Java. While traditional object types describe what an object is, functional interface types describe what an object can do—its behavior.

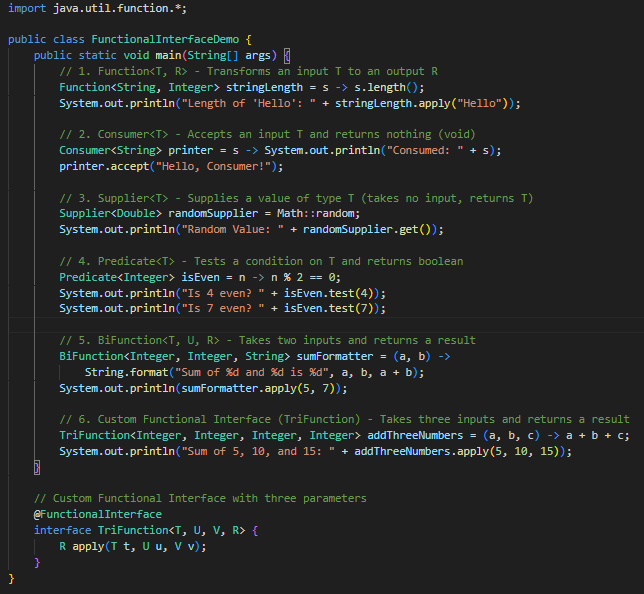
#### **How Functional Interfaces Work:**

When you use a lambda expression or method reference, the Java compiler:

1. Identifies the target functional interface from context
2. Verifies the lambda's parameter types and return type are compatible with the interface method
3. Creates an implementation of that interface with the lambda/method reference as the implementation of the abstract method

The @FunctionalInterface annotation is a marker that ensures an interface meets the requirements of having exactly one abstract method. While optional, it helps detect errors at compile time if additional abstract methods are added accidentally.

### **Program Example:**



# 3.1.5 Method References

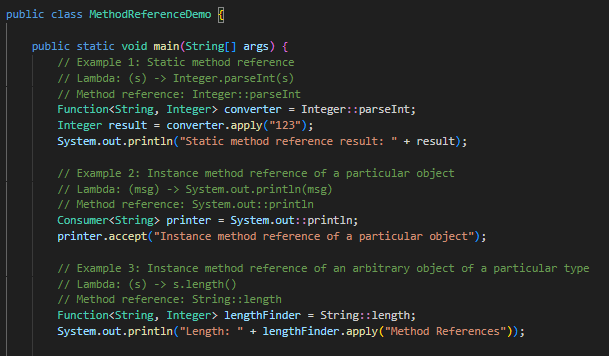
## **Theoretical Foundation**

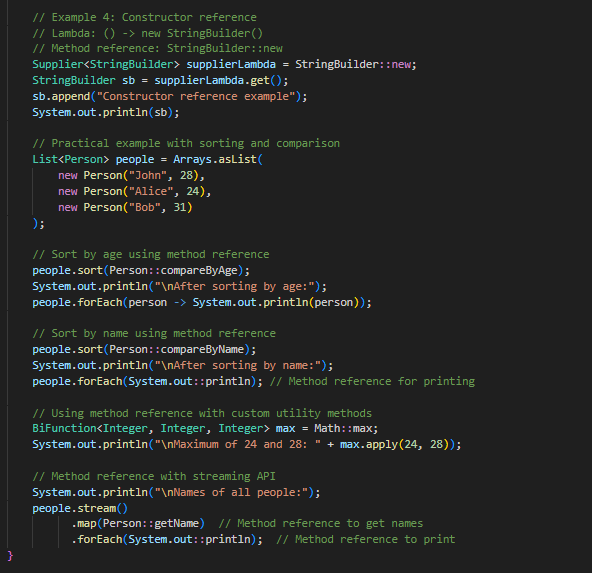
## Method references provide a shorthand notation for lambda expressions that call a specific method. They enhance Java's functional programming capabilities by making code more concise and readable, particularly when working with functional interfaces. Method references, introduced in Java 8, align with the language's evolution toward supporting more functional programming paradigms.

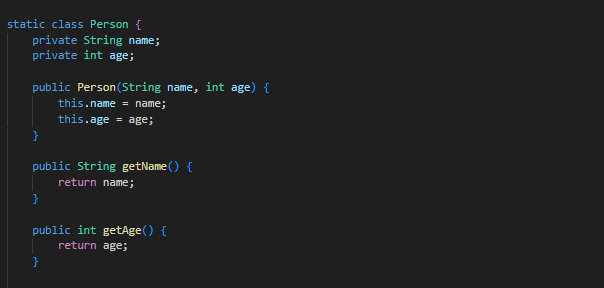
### **Types of Method References:**

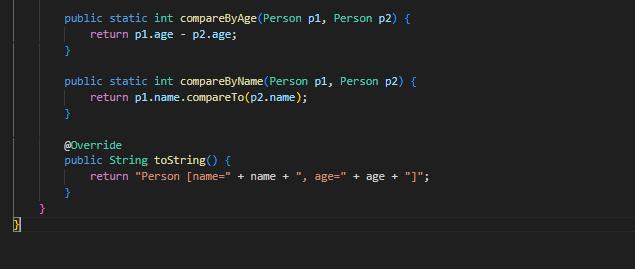
1. **Static method references**: ClassName::staticMethodName
2. **Instance method references of particular objects**: instanceReference::methodName
3. **Instance method references of arbitrary objects of a particular type**: ClassName::instanceMethodName
4. **Constructor references**: ClassName::new

### **Program Example:**









# 3.1.6 Collections & Generics

## **Theoretical Foundation**

Java's Collections Framework provides a unified architecture for representing and manipulating collections of objects. Generics, introduced in Java 5, enable type safety and compile-time type checking, eliminating the need for explicit casting and preventing runtime type errors.

### **Collection Framework Hierarchy:**

* **Collection** (interface): Root interface with basic operations like add, remove, contains.

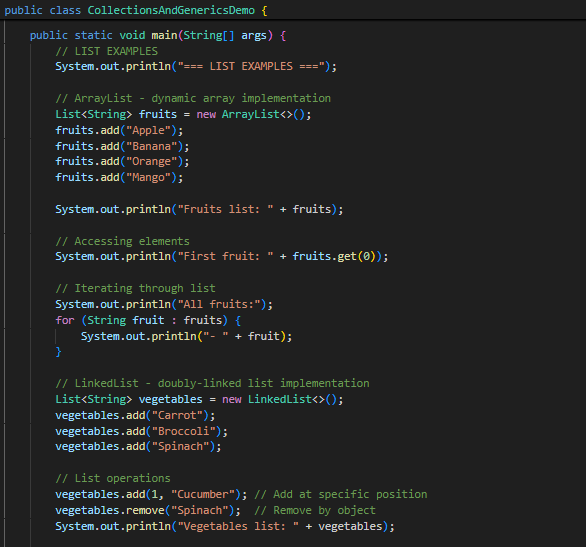
1. **List**: Ordered collection with positional access.
2. **Set**: Collection with no duplicate elements.
3. **Queue**: Collection designed for holding elements prior to processing.
4. **Deque**: Double-ended queue supporting element insertion and removal at both ends.

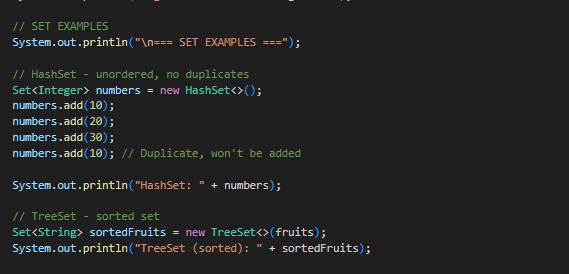
* **Map** (interface): Key-value pairs with unique keys.

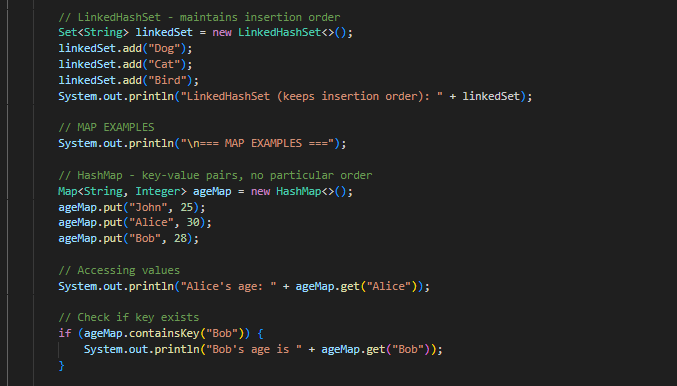
### **Generic Types:**

* **E**: Element (used in collections)
* **K**: Key
* **V**: Value
* **N**: Number
* **T**: Type
* **S, U, V, etc.**: Additional types

## **Program Example:**









# 3.1.7 Exception Handling

## **Theoretical Foundation**

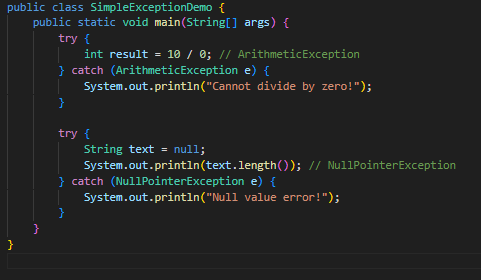
Exception handling in Java provides a structured approach to managing runtime errors, ensuring that program flow isn't disrupted by unexpected conditions. The model separates error detection from error handling, making programs more robust and maintainable.

### **Exception Hierarchy:**

* **Throwable**: Root of exception hierarchy
  1. **Error**: Serious problems, not typically handled by applications (e.g., OutOfMemoryError)
  2. **Exception**: Recoverable conditions that programs might handle
     + **Checked Exceptions**: Must be caught or declared (e.g., IOException)
     + **Unchecked Exceptions**: Subclasses of RuntimeException, don't require explicit handling (e.g., NullPointerException)

### **Key Concepts:**

* **Try-Catch-Finally**: Core mechanism for handling exceptions
* **Throw and Throws**: Mechanisms for generating and declaring exceptions
* **Multi-catch**: Handling multiple exceptions in one catch block (Java 7+)
* **Program Example:**



**3.2 Backend Development – Spring Boot**

## **3.2.1 Spring Boot Introduction**

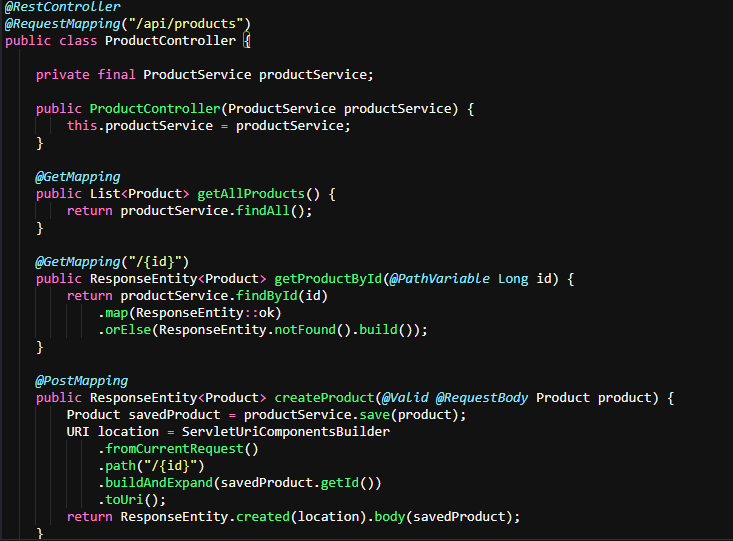
Spring Boot is a framework that simplifies the development of Java-based applications by providing defaults for code and annotation configuration. It follows the "convention over configuration" principle and is designed to get applications up and running as quickly as possible.

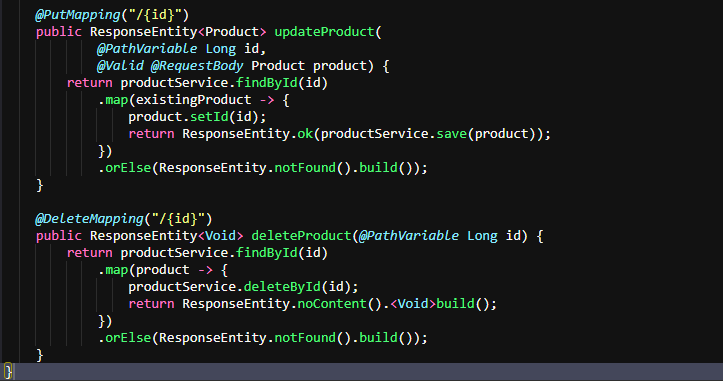
* Spring Boot offers the following advantages:
* Creates stand-alone Spring applications with embedded servers
* Provides opinionated 'starter' dependencies to simplify build configuration
* Automatically configures Spring and third-party libraries when possible
* Requires no XML configuration
* Offers production-ready features such as metrics, health checks, and externalized configuration
* The core components of Spring Boot include:
* Auto-configuration: Automatically configures your application based on dependencies
* Starter dependencies: Curated sets of compatible dependencies for common use cases
* Actuator: Monitoring and management capabilities for production environments
* Spring Boot CLI: Command-line tool for rapid prototyping

Spring Boot promotes a microservices architecture but is equally effective for monolithic applications, providing a foundation that scales with your application needs.

## **3.2.2 REST API Development**

* Spring Boot simplifies the creation of RESTful web services through its powerful annotations and built-in capabilities:
* REST Controller example:



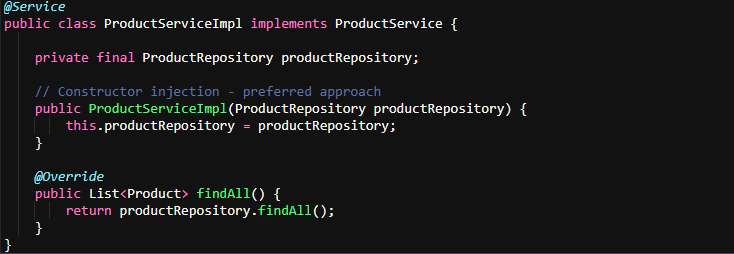


* Key REST-related annotations:
* @RestController: Combines @Controller and @ResponseBody, eliminating the need to annotate each method
* @RequestMapping: Maps requests to handler methods
* HTTP method-specific annotations: @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping
* Parameter annotations: @PathVariable, @RequestParam, @RequestBody, @RequestHeader
* Response annotations: @ResponseStatus, ResponseEntity
* Content negotiation is handled automatically, with Spring Boot converting between JSON and Java objects using libraries like Jackson.

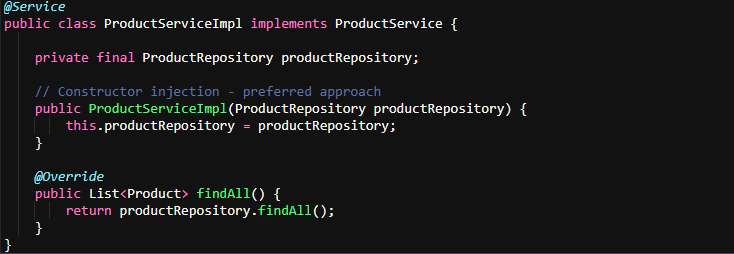
## **3.2.3 Dependency Injection (@Autowired)**

* Dependency Injection (DI) is a core feature of the Spring Framework that Spring Boot leverages. It allows objects to receive their dependencies rather than creating them, leading to loosely coupled code that's easier to test and maintain.
* Spring Boot supports several injection methods:

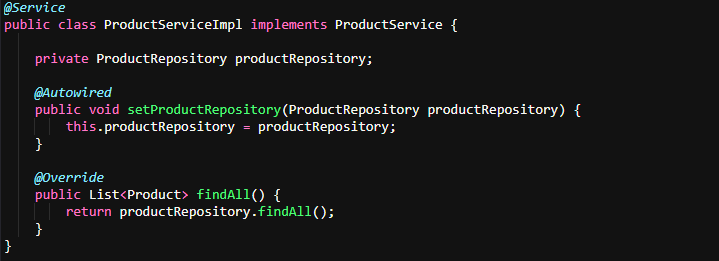
1. **Constructor Injection (Preferred):**



1. **Field Injection:**



1. **Setter Injection:**



* Spring Boot's auto-configuration feature automatically registers many components as beans, but you can define custom beans using annotations:
* @Component: Generic component
* @Service: Business service layer
* @Repository: Data access layer
* @Controller / @RestController: Web layer
* @Configuration: Configuration classes that define beans

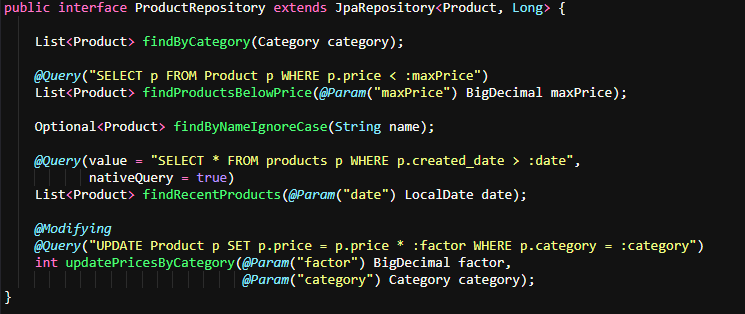
## **3.2.4 Spring Data JPA**

* Spring Data JPA simplifies data access by reducing boilerplate code and providing ready-to-use repository implementations:

1. **Entity Definition:**

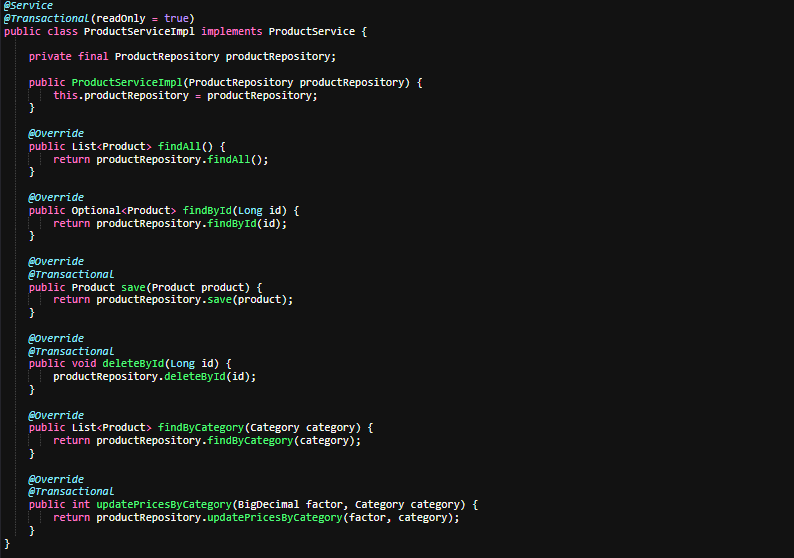


1. **Repository Interface:**



* Spring Data JPA features:
* Out-of-the-box CRUD operations: save(), findById(), findAll(), delete(), etc.
* Query methods: Define queries by method name conventions
* Custom JPQL queries using @Query
* Native SQL queries with nativeQuery = true
* Pagination and sorting support
* Auditing capabilities (@CreatedBy, @LastModifiedDate, etc.)
* Transaction management (@Transactional)

1. **Service Layer with Transaction Management:**



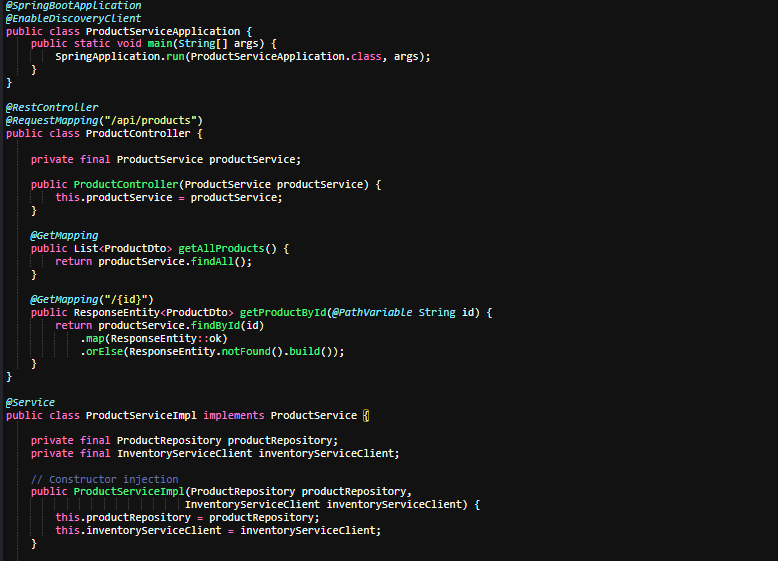
## **3.2.5 Microservices Architecture**

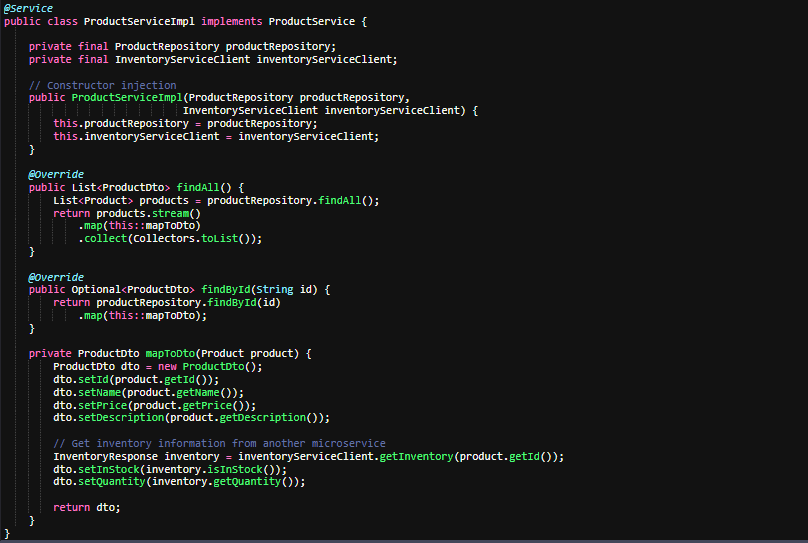
* Microservices architecture is an approach to software development where an application is built as a collection of small, loosely coupled services. Spring Boot excels in implementing this architecture through its integration with Spring Cloud.
* **Key components of a microservices architecture using Spring Boot:**

1. **Service Discovery**: Registration and discovery of services
2. **API Gateway**: Single entry point for clients
3. **Configuration Server**: Centralized configuration management
4. **Circuit Breaker**: Handling failures gracefully
5. **Distributed Tracing**: Tracking requests across services

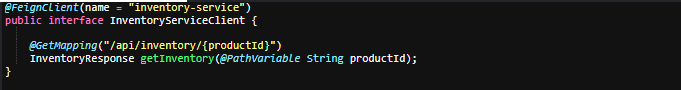
* **Example of a microservice application structure:**





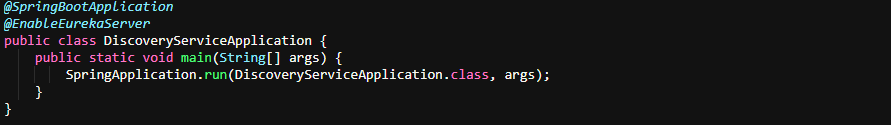


* Inter-service Communication:



## **3.2.6 Service Registration (Eureka)**

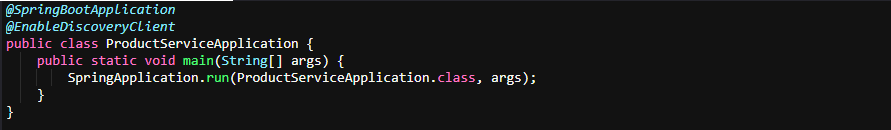
* Eureka is a service registration and discovery server that allows microservices to find and communicate with each other without hardcoding their hostnames and ports.
* **Setting up Eureka Server:**



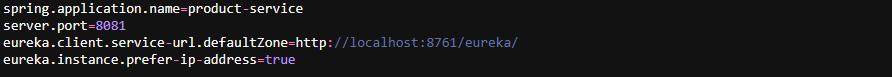
* **application.properties for Eureka Server:**



* **Configuring a service as Eureka Client:**



* **application.properties for Eureka Client:**



# Code Inspection and Optimization Reports

### **Code inspection:**

* Code inspection is a process in which a team of developers reviews software code to identify defects and issues that could cause problems later on. The goal is to catch problems early and ensure that the code is of high quality before it is deployed.

### **Code analysis:**

* Code analysis is the automated process of reviewing software code to identify potential issues and improve overall code quality. This can involve a range of techniques, from simple syntax checking to complex algorithms that analyze code for potential security vulnerabilities, performance bottlenecks, and other issues.

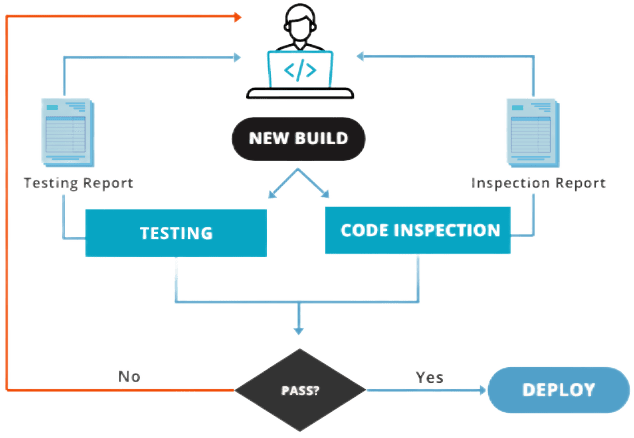
### **Code optimization:**

* Code optimization is the process of improving software code to make it faster, more efficient, and more resource-friendly. This can involve techniques such as removing unnecessary code, simplifying algorithms, and using more efficient data structures.

### **Code debugging:**

* Code debugging is the process of finding and fixing errors, bugs, and other issues in software code. This can involve using specialized tools to step through the code and identify problems, as well as writing additional code to address issues.

### Code Inspection Report:

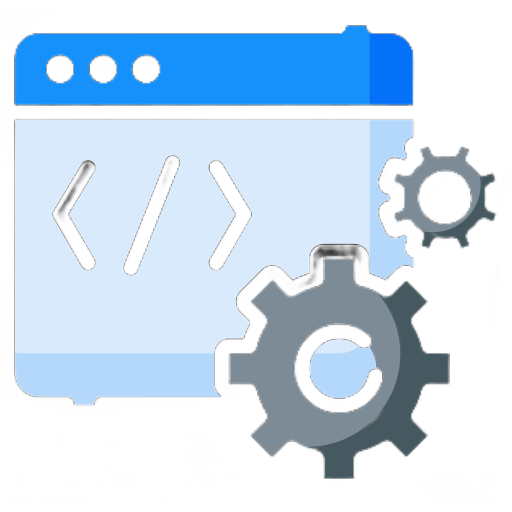


[4.5.0.1 Code Inspection Report]

* A code inspection report is a document that details the results of a code inspection process conducted by a team of software developers. It is an important quality assurance technique that helps identify and address defects and potential issues in software code before it is deployed.
* The inspection process involves a team of developers carefully examining the code to identify any errors or defects, as well as opportunities for improvement. The team typically includes the code author, who presents the code to the group, as well as other developers who review the code and provide feedback.
* The inspection report documents the results of the inspection process, providing a record of the issues identified, along with recommended solutions and the actions taken to address them. The report may also include metrics such as defect density and the number of issues found per line of code, as well as an overall assessment of the code's quality.
* In addition to identifying and addressing issues, the code inspection process can also help promote knowledge sharing and collaboration among team members. By reviewing each other's code, developers can learn from each other and improve their skills and understanding of best practices.
* Overall, a code inspection report is a valuable tool for ensuring the quality and reliability of software code, and should be an important part of any software development process.
* Here are some basic reports which I created during my Knowledge Transfer:

1. [Analysis Report](https://docs.google.com/document/d/18O_S28SkihNbbfHsEbz8v8hli2tathUx/edit?usp=share_link&ouid=110149958339697313507&rtpof=true&sd=true)
2. [Optimization Report](https://docs.google.com/document/d/1IpEcMEcu93YxM1ToSFcz1vQRnsdpcFG8/edit?usp=share_link&ouid=110149958339697313507&rtpof=true&sd=true)
3. [Component Complete Report](https://docs.google.com/document/d/1O4oKZiT1L90c7TJdmAc0tr8rud--cel6/edit?usp=share_link&ouid=110149958339697313507&rtpof=true&sd=true)

### **Code Optimization Report:**



[4.6.0.1 Code Optimization Report]

* While similar in purpose to a code inspection report, a code optimization report focuses specifically on identifying opportunities for improving the performance and efficiency of software code.
* The code optimization process involves analyzing the code to identify any inefficient or resource-intensive operations, such as loops or memory allocations that may be slowing down the application. The report documents the findings of this analysis, as well as recommendations for optimizing the code to improve performance and reduce resource consumption.
* The report may also include metrics such as execution time and memory usage before and after optimization, as well as an assessment of the overall impact on the application's performance and user experience.
* Like a code inspection report, a code optimization report can help promote collaboration and knowledge sharing among team members, as well as improve the quality and reliability of the software code.
* Overall, a code optimization report is a valuable tool for ensuring that software applications are running at peak performance and delivering the best possible user experience. It should be considered an essential part of any software development process that prioritizes performance and efficiency.

# Research And Development of Custom Features

* **Research:** Research is a systematic and structured process of gathering and analysing information and data to gain new knowledge, insights, or understanding about a particular topic or issue.
* **Development of Custom Features:** Development of custom feature is a process of identifying and creating unique functionalities or capabilities that are tailored to meet the specific needs of a particular product or service. It involves researching, experimenting, and testing to create new features or modify existing ones.
* Research and development of custom features involves identifying and developing unique functionalities or capabilities that are tailored to meet the specific needs of a particular product or service. This process typically involves a combination of research, experimentation, and testing to create new features or modify existing ones.
* The first step in this process is to identify the specific needs or requirements of the product or service in question. This may involve conducting market research, analyzing user feedback, or consulting with experts in the relevant field. Once the requirements are established, the development team can begin exploring potential solutions.
* The research and development process may involve prototyping, testing, and refining the new feature until it meets the desired specifications. This may involve collaboration between different teams, such as software engineers, product designers, and user experience specialists. The end result is a custom feature that enhances the product or service and sets it apart from competitors.
* Custom features can be a key differentiator in competitive markets, as they offer unique benefits or capabilities that are not available in off-the-shelf solutions. However, the research and development process can be time-consuming and costly, requiring significant resources and expertise. As such, it is important for organizations to carefully weigh the potential benefits against the costs and risks before embarking on custom feature development.

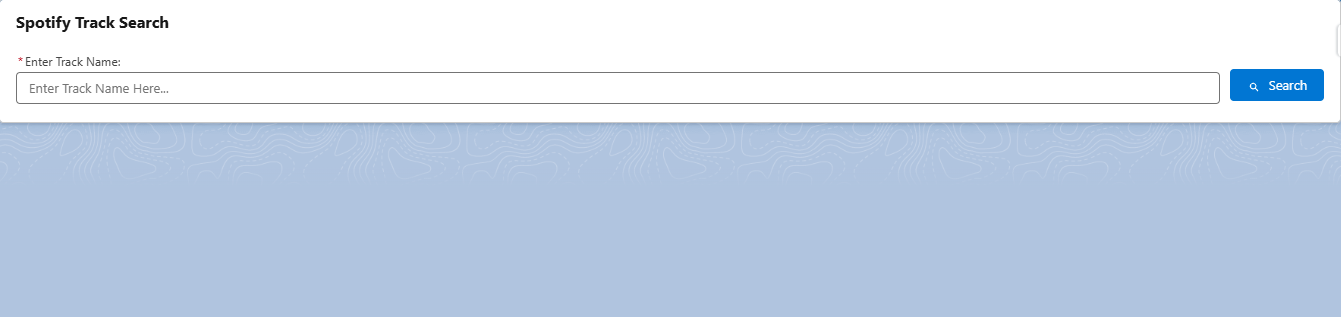
**Research I done during my internship:**

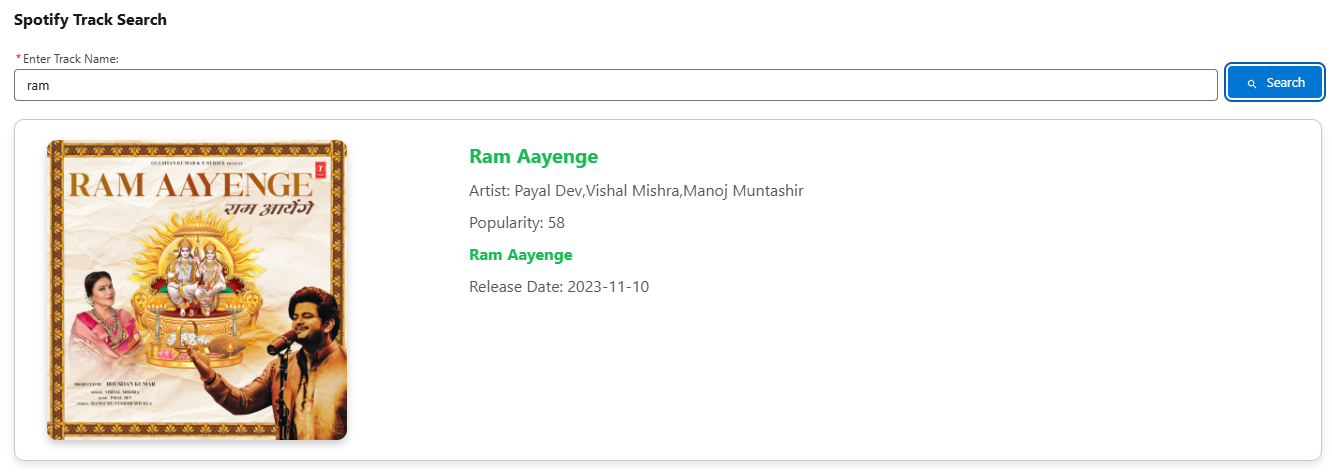
## **Integration:**

During my internship, I researched and implemented an integration strategy with third-party services. Key points of my research include:

* **Spotify Integration:** Utilized Spotify's API to search for and retrieve track details.
* **Zendesk Ticket Creation:** Automated the creation of a Zendesk ticket when a Salesforce case is opened.
* **Data Synchronization:** Ensured that relevant Zendesk ticket details are updated on the Salesforce Case object.
* **Secure API Communication:** Employed secure authentication methods (OAuth/API keys) for reliable data exchange.
* **Automation & Error Handling:** Implemented automation processes with error logging to maintain integration reliability.

This research helped me understand how to seamlessly connect external systems with Salesforce, enhancing both functionality and user experience.

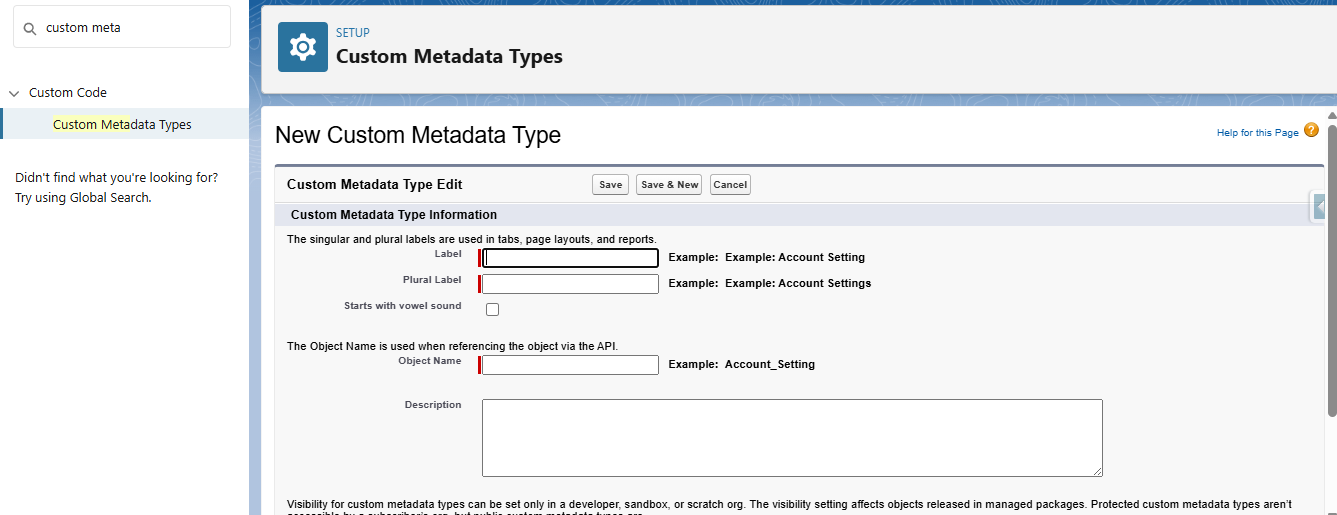


[5.1.0.1 Spotify Search Bar]

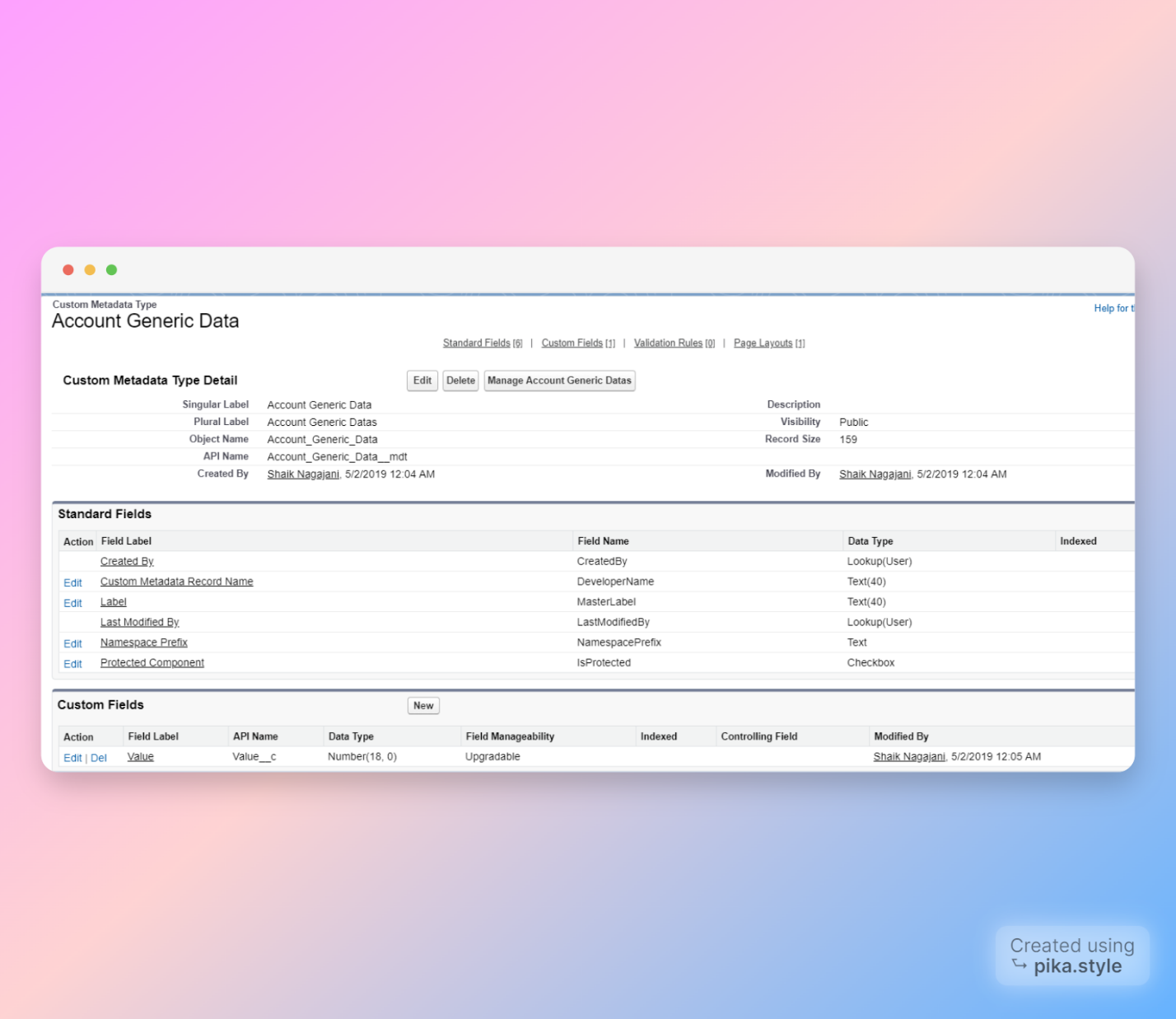
[5.1.0.2 Spotify Result ]

## **Custom Metadata:**

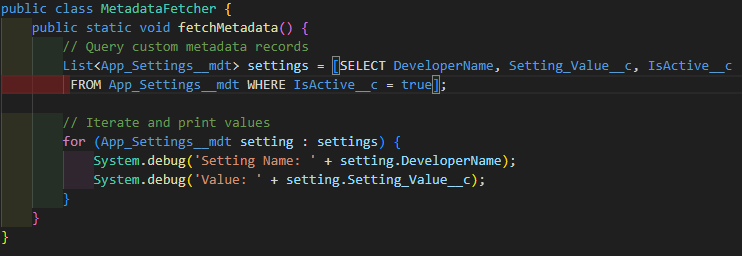
* Custom metadata is a feature in Salesforce that allows businesses to create and manage custom data records that can be used to configure and customize their Salesforce applications. Custom metadata records are similar to custom objects or custom settings, but they offer several unique advantages.
* One key advantage of custom metadata is that it is deployable between different Salesforce environments, such as sandboxes or production orgs. This means that businesses can develop and test their customizations in a separate environment before deploying them to their live applications, reducing the risk of errors or conflicts.
* Another advantage of custom metadata is that it can be accessed and modified through the Salesforce Metadata API, making it easier to automate and integrate with other systems or tools. Custom metadata records can also be used in formulas, workflows, or Apex code, allowing businesses to create more flexible and dynamic applications.
* Custom metadata records can be created using a variety of data types, such as picklists, text, numbers, or even Apex code. They can also be organized into groups or hierarchies, making it easier to manage large or complex sets of records.
* Overall, custom metadata is a powerful and flexible feature in Salesforce that can help businesses customize their applications and streamline their development processes.



[5.2.0.1 Create Metadata]



[5.2.0.2 Manage Metadata]



[5.2.0.3 Get Custom Metadata]

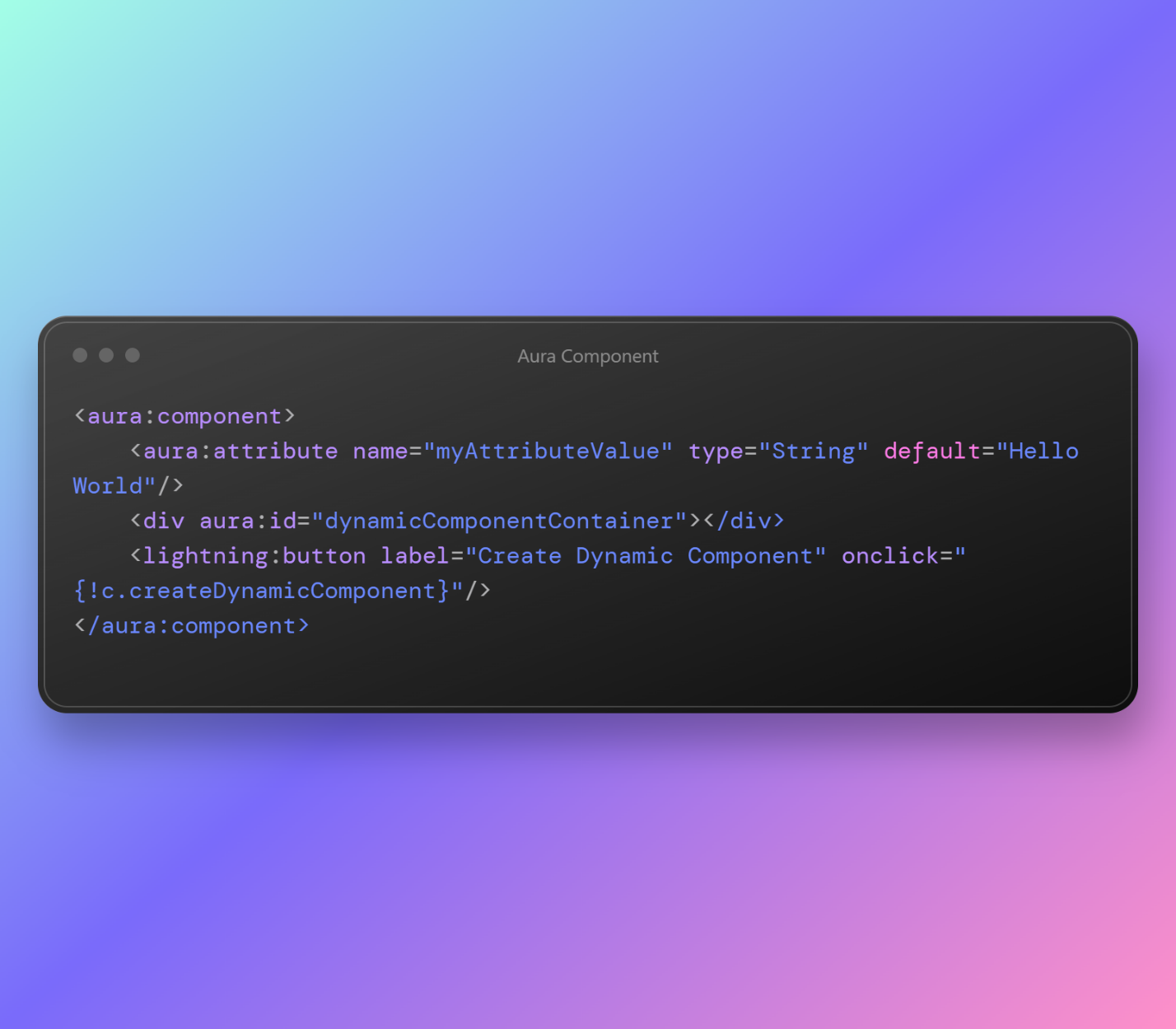
## **Dynamic Components:**

* In Aura, Dynamic Components are components that are created at runtime, rather than being declared in markup. They are useful when you need to create components dynamically based on user input or other conditions.
* Here is an example of how to create a Dynamic Component in Aura:
* Create a new component, let's call it "DynamicComponentExample". In this component, we will define the markup and controller for the dynamic component.
* In the component's controller, define a method that creates the dynamic component. For example:



[5.3.0.1 Dynamic Component Hepler.js]

* This method uses the $A. createComponent function to create a new instance of a custom component called "MyDynamicComponent". It passes in a set of attributes for the component, including an "aura:id" attribute that can be used to reference the dynamic component later.
* The method also defines a callback function that runs after the component is created. This function checks the status of the component creation and adds the new component to the "dynamicComponentContainer" in the main component's markup.
* In the main component's markup, define a container for the dynamic component. For example:



[5.3.0.2 Dynamic Component Component.cmp]

* This markup defines a container div with an "aura:id" attribute of "dynamicComponentContainer". It also includes a button that triggers the "createDynamicComponent" method in the controller.
* When the button is clicked, the "createDynamicComponent" method creates a new instance of the "MyDynamicComponent" component with the attribute value from the main component's "myAttributeValue" attribute. It then adds the new component to the "dynamicComponentContainer" div in the main component's markup, creating a dynamic component on the fly.

# My Skills & Work

## **Skills:**

* **Flight Management System Development:**

1. Full-Stack Web Development a. Angular Framework b. Component Architecture c. Responsive Design Principles d. State Management e. UI/UX Implementation f. Dynamic Form Handling
2. Backend Development a. Spring Boot Application Development b. RESTful API Design and Implementation c. Authentication & Authorization d. Database Integration e. Exception Handling f. Service Layer Implementation g. Java 8+ Features
3. Database Management a. MySQL Database Design b. SQL Query Optimization c. Entity Relationship Modeling d. Data Normalization e. Transaction Management
4. Development Workflow a. Git Version Control b. Branch Management
5. System Integration a. Payment Gateway Integration b. Email Notification Services c. Flight Data API Integration d. Map Services Integration

* **Web Development:**

1. HTML5
2. CSS3/SCSS
3. TypeScript
4. JavaScript ES6+
5. Bootstrap/Material Design

* **Soft Skills:**

1. Communication
2. Teamwork
3. Time Management
4. Problem Solving
5. Critical Thinking
6. Adaptability
7. Attention to Detail
8. Work Ethic

## **Work:**

"During my internship, I contributed significantly to the development of the JetWayz flight management system, focusing on both frontend and backend components. My primary responsibilities included implementing the flight booking module with interactive seat selection capabilities and developing the user authentication system with role-based access control for customers, airline staff, and administrators. I created a comprehensive reservation management system that handled the entire booking lifecycle, from search to confirmation, and integrated secure payment processing with multiple gateway options. Additionally, I designed and implemented the admin dashboard for monitoring system metrics and managing flight operations, along with a responsive customer profile management interface. On the research side, I investigated industry best practices for airline API integration, secure payment processing workflows, and efficient flight data caching strategies to enhance system performance. During deployment phases, I configured the Spring Boot application for various environments, documented API endpoints, conducted thorough cross-browser compatibility testing, and implemented automated testing for critical system components. Throughout the project, I adhered to agile methodologies and collaborated closely with senior developers to ensure code quality and timely delivery of features."

## **Tasks Completed During Internship:**

* **Development:**

1. Flight booking module implementation
2. User authentication and role-based access control
3. Reservation management system development
4. Payment processing integration
5. Booking confirmation and e-ticket generation
6. Flight search and filtering functionality
7. Admin dashboard implementation

* **Research:**

1. Research on airline API integration best practices
2. Research on secure payment processing workflows
3. Research on booking cancellation and refund policies
4. Research on flight data caching strategies
5. Research on performance optimization for search results

* **Deployment:**

1. Deployment of Angular components and services
2. Configuration of Spring Boot application for different environments
3. Database schema migration and management
4. Documentation of API endpoints and system architecture

# Conclusion

* After completing my internship at OPL Pvt. Ltd., I am pleased to conclude that my experience working on the JetWayz flight management system has been incredibly valuable and enriching. This comprehensive project has provided me with practical insights into enterprise software development that no classroom experience could match.
* Throughout my time working on JetWayz, I had the opportunity to develop and refine a diverse set of technical skills across the full development stack. From implementing complex flight booking modules to integrating secure payment processing systems, each challenge expanded my capabilities as a developer. The experience of building a responsive, user-friendly interface for multiple stakeholders—from administrators to airline staff to passengers—taught me the importance of understanding diverse user needs when designing digital solutions.
* I was fortunate to work alongside experienced professionals who offered guidance and constructive feedback, helping me improve both my coding practices and problem-solving approaches. Their mentorship significantly enhanced my understanding of software architecture and best practices in the industry. OPL's collaborative environment encouraged open communication and idea sharing, making it possible to overcome technical hurdles efficiently and learn continuously from my peers.
* The agile methodology adopted by OPL allowed me to experience real-world software development cycles, from planning and development to testing and deployment. This practical exposure to project management has given me valuable insights into how teams coordinate to deliver complex systems on schedule.
* In conclusion, my internship experience working on the JetWayz project at OPL Pvt. Ltd. has been transformative in my journey as a software engineer. The technical expertise, collaborative skills, and industry knowledge I've gained have prepared me well for future challenges in the field. I am confident that these experiences will serve as a strong foundation as I embark on my professional career in software development.

# References

* **Documents:**
* Angular Documentation
* Spring Boot Documentation
* MySQL Documentation
* RESTful API Design Guidelines
* **Online articles:**
* Baeldung: <https://www.baeldung.com/>
* Medium - Angular In Depth: <https://medium.com/angular-in-depth>
* Spring Framework Guru: <https://springframework.guru/>
* Stack Overflow Developer Survey 2024
* **Websites:**
* OPL Pvt. Ltd. (n.d.). About Us, Mission, Vision, Logo, etc... Retrieved from
* <https://www.oplinnovate.com/>
* <https://angular.io/docs>
* <https://docs.spring.io/spring-boot/docs/current/reference/html/>
* <https://dev.mysql.com/doc/>
* <https://s3-ap-southeast-1.amazonaws.com/gtusitecirculars/uploads/Internship___Project_Report_Guidelines__3180701__661242.pdf>

# Mentor’s Feedback (Company Feedback)