Java 8 features:

1. Lambda Expressions:
2. Streams API:
3. Optional Class
4. Default and Static Methods in Interfaces
5. New Date and Time API
6. Nashorn JavaScript Engine:
7. Improved performance and other enhancements:

OOPS concepts:

Object,classes,inheritance,polymphism,encapsulation and abstraction.

Creating the webapplication:

1. Develop the front-end with Angular 13:
   * Create components, services, and routing modules for your application's features.
   * Use Angular Material or another UI library for styling and reusable UI components.
   * Implement HttpClient services to communicate with the back-end API.
   * Utilize Angular's reactive forms and validation for user input.
   * Implement state management and authentication, if needed.
2. Develop the back-end with Spring Boot:
   * Create the necessary packages for your application (e.g., models, repositories, services, and controllers).
   * Define your models/entities and annotate them with JPA annotations.
   * Create repositories for data access using Spring Data JPA.
   * Implement services to handle the business logic.
   * Create controllers to define RESTful API endpoints.
   * Configure the database connection and other settings in **application.properties** or **application.yml**.
   * Implement exception handling and validation as needed.
3. Testing and debugging:
   * Write unit tests for your Java and Angular code using testing frameworks like JUnit and Jasmine.
   * Test API endpoints with tools like Postman.
   * Use your IDE's debugger or browser DevTools to debug issues.
4. Deployment:
   * Package your Spring Boot application into a JAR file using **./mvnw package** or **./gradlew build**.
   * Deploy the JAR file to a server or platform-as-a-service (PaaS) like Heroku, AWS Elastic Beanstalk, or Google App Engine.
   * Build your Angular app for production using **ng build --prod**. Deploy the generated **dist** folder to a web server or hosting service like AWS S3, Netlify, or Firebase Hosting.

restful web service based application

Creating a RESTful web service-based application involves designing and implementing a server-side API that follows the REST (Representational State Transfer) principles. Here's a step-by-step guide for building a RESTful web service-based application using Java and Spring Boot:

1. Plan your application:
   * Define the application's purpose, target audience, and key features.
   * Identify the required API endpoints and their corresponding HTTP methods (GET, POST, PUT, DELETE).
2. Set up your development environment:
   * Install Java Development Kit (JDK) and a Java IDE (e.g., IntelliJ IDEA or Eclipse).
   * Install and set up a version control system (e.g., Git).
3. Initialize the project:
   * Create a new Java project using the Spring Initializr (<https://start.spring.io/>). Select the necessary dependencies, such as Spring Web and any database-related dependencies if needed. Download the generated project and import it into your IDE.
4. Develop the web service:
   * Organize your code into packages (e.g., models, repositories, services, and controllers).
   * Define your data models (if necessary) and annotate them with JPA annotations (if using a database).
   * Create repositories for data access using Spring Data JPA (if using a database).
   * Implement services to handle the business logic.
   * Create controllers to define RESTful API endpoints. Use proper HTTP methods, and ensure endpoints are designed following REST principles.
   * Configure CORS (Cross-Origin Resource Sharing) settings if needed.
   * Implement exception handling and input validation.
5. Testing and debugging:
   * Write unit and integration tests for your Java code using testing frameworks like JUnit and Mockito.
   * Test API endpoints with tools like Postman, Insomnia, or curl.
   * Use your IDE's debugger to debug issues.
6. Deployment:
   * Package your Spring Boot application into a JAR file using **./mvnw package** or **./gradlew build**.
   * Deploy the JAR file to a server or platform-as-a-service (PaaS) like Heroku, AWS Elastic Beanstalk, or Google App Engine.

By following these steps, you will create a RESTful web service-based application using Java and Spring Boot. The resulting API can be consumed by various clients, such as web applications, mobile apps, or other services.

Spring framework:

The Spring Framework is a popular Java-based framework for building enterprise-level applications. It is an open-source project that provides a comprehensive programming and configuration model for modern Java-based applications. Spring simplifies the development of Java applications by offering a lightweight and flexible infrastructure that promotes modularity, testability, and ease of integration.

Some key features of the Spring Framework include:

Dependency injection

Aspect-Oriented Programming (AOP):

Data Access and Integration:

Web Application Development:

Spring Boot:

Microservices Support

Testing Support:

Most recently asked questions about oracle and mysql?

Oracle Database:

1. What is the difference between Oracle and other relational databases?
2. Explain the Oracle architecture and its main components.
3. What are the different types of tablespaces in Oracle?
4. Explain the differences between primary key, foreign key, and unique key.
5. What is normalization, and why is it important in database design?
6. What is an index in Oracle, and what are the different types of indexes?
7. What are the differences between an explicit cursor and an implicit cursor in Oracle?
8. Explain the difference between TRUNCATE and DELETE commands.
9. What is a transaction in Oracle, and how do you control transactions using COMMIT, ROLLBACK, and SAVEPOINT?
10. What are the differences between stored procedures, functions, and triggers in Oracle?

MySQL Database:

1. What is MySQL, and why is it so popular?
2. Explain the MySQL architecture and its main components.
3. What are the different storage engines in MySQL, and how do they differ?
4. What are the differences between InnoDB and MyISAM storage engines?
5. What is the difference between CHAR and VARCHAR data types in MySQL?
6. Explain the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN in MySQL.
7. How do you create and use indexes in MySQL to optimize query performance?
8. What is the difference between AUTO\_INCREMENT and LAST\_INSERT\_ID() in MySQL?
9. What are the various backup methods available in MySQL?
10. How do you secure a MySQL database, and what are some best practices for MySQL security?

Cassandra:

Apache Cassandra is a highly scalable, distributed, and fault-tolerant NoSQL database management system designed to handle large amounts of data across many commodity servers. It provides high availability and performance without compromising on scalability. Cassandra was originally developed by Facebook and later open-sourced under the Apache Foundation.

Docker Containers:

1. What is Docker, and why is it useful?
2. Explain the difference between a Docker image and a Docker container.
3. What is Dockerfile, and what are some common instructions used in a Dockerfile?
4. What are the main components of the Docker architecture (e.g., Docker Daemon, Docker Client, Docker Registry)?
5. How do you create and manage Docker containers using Docker CLI commands?
6. What is Docker Compose, and how is it used to manage multi-container applications?
7. Explain Docker Networking and the different types of network drivers available in Docker.
8. How do you share data between containers using Docker Volumes and Bind Mounts?
9. What is the difference between CMD and ENTRYPOINT instructions in a Dockerfile?
10. Explain best practices for creating Docker images and optimizing container performance.

OpenShift:

1. What is OpenShift, and how does it differ from Kubernetes?
2. Describe the main components of the OpenShift architecture (e.g., Master, Nodes, etcd, API Server).
3. Explain the role of Projects and Namespaces in OpenShift.
4. How do you deploy an application in OpenShift using Source-to-Image (S2I)?
5. What are OpenShift Routes, and how do they differ from Kubernetes Ingress?
6. Describe the role of BuildConfigs and DeploymentConfigs in OpenShift.
7. Explain the concept of OpenShift Templates and their use in deploying applications.
8. How do you manage application secrets and configuration data using ConfigMaps and Secrets in OpenShift?
9. Describe the role of OpenShift Operators and the Operator Framework.
10. Explain best practices for managing applications and resources in an OpenShift cluster.