Weekly Assessment-6

Theory Questions:

Q1. what is the @component and @Controller.?

Ans.) @Component is an annotation that allows Spring to automatically detect our custom beans.

In other words, without having to write any explicit code, Spring will:

- Scan our application for classes annotated with @Component
- Instantiate them and inject any specified dependencies into them
- Inject them wherever needed

@Controller annotation indicates that a particular class serves the role of a controller. There is no need to extend any controller base class or reference the Servlet API. The basic purpose of the @Controller annotation is to act as a stereotype for the annotated class, indicating its role. The dispatcher will scan such annotated classes for mapped methods, detecting @RequestMapping annotations

Q2. @RequestMapping, @Required, @Qualifier, @Autowired @Temporal, @Entity, @RequestBody, @RestController, @Query @PathVariable Annotation explain?

Ans.) @RequestMapping annotation is used to map web requests onto specific handler classes and/or handler methods. @RequestMapping can be applied to the controller class as well as methods.@Required annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time. Otherwise, the container throws a BeanInitializationException exception.

@Qualifier annotation is used to resolve the autowiring conflict, when there are multiple beans of same type. The @Qualifier annotation can be used on any class annotated with @Component or on methods annotated with @Bean . This annotation can also be applied on constructor arguments or method parameters.

@Autowired annotation in spring automatically injects the dependent beans into the associated references of a POJO class. This annotation will inject the dependent beans by matching the data-type (i.e. Works internally as Autowiring byType)

@Temporal annotation solves the one of the major issue of converting the date and time values from Java object to compatible database type and retrieving back to the application.

@Entity annotation specifies that the class is an entity and is mapped to a database table. The @Table annotation specifies the name of the database table to be used for mapping.

@RequestBody annotation maps the HttpRequest body to a transfer or domain object, enabling automatic deserialization of the inbound HttpRequest body onto a Java object. Spring automatically deserializes the JSON into a Java type, assuming an appropriate one is specified.

@RestController annotation is used in order to simplify the creation of RESTful web services. It's a convenient annotation that combines @Controller and @ResponseBody, which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.

@Query annotation is used for its value attribute contains the JPQL or SQL to execute. The @Query annotation takes precedence over named queries, which are annotated with @NamedQuery or defined in an orm.xml file.

@PathVariable annotation can be used to handle template variables in the request URI mapping, and set them as method parameters.

Q3. What are the Stereotype annotations and define MVC flow.?

Ans.) Spring Framework provides us with some special annotations. These annotations are used to create Spring beans automatically in the application context. @Component annotation is the main Stereotype Annotation. There are some Stereotype meta-annotations which is derived from @Component those are

1. @Service
2. @Repository
3. @Controller

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. ... MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

Q4.- Difference between application.properties & YML file.?

Ans.)

YAML(.yml)	.properties
Spec can be found here	It doesn’t really actually have a spec. The closest thing it has to a spec is actually the javadoc.
Human Readable (both do quite well in human readability)	Human Readable
Supports key/val, basically map, List and scalar types (int, string etc.)	Supports key/val, but doesn’t support values beyond the string
Its usage is quite prevalent in many languages like Python, Ruby, and Java	It is primarily used in java

YAML(.yml)	.properties
Hierarchical Structure	Non-Hierarchical Structure
Spring Framework doesn’t support @PropertySources with .yml files	supports @PropertySources with .properties file
If you are using spring profiles, you can have multiple profiles in one single .yml file	Each profile need one separate .properties file
While retrieving the values from .yml file we get the value as whatever the respective type (int, string etc.) is in the configuration	While in case of the .properties files we get strings regardless of what the actual value type is in the configuration

Q5. What are the scopes in Spring Framework?

Ans.) The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

Sr.No.	**Scope & Description**
1	**singleton** This scopes the bean definition to a single instance per Spring IoC container (default).
2	**prototype** This scopes a single bean definition to have any number of object instances.
3	**request** This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
4	**session** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
5	**global-session** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Q6. What is cascading and what are different types of cascading?

Ans.) Cascading is a phenomenon involving one object propagating to other objects via a relationship. It is transitive in nature and the cascade attribute in hibernate defines the relationship between the entities. The cascading types supported by the hibernate framework are as follow:

1. CascadeType.PERSIST: It means that the save() and persist() operations in the hibernate cascade to the related entities.
2. CascadeType.MERGE: It means that the related entities are joined when the owning entity is joined.
3. CascadeType.REMOVE: It means that the related entities are deleted when the owning entity is deleted.
4. CascadeType.DETACH: It detaches all the related entities if a manual detach occurs.
5. CascadeType.REFRESH: It works similar to the refresh() operation in the hibernate.
6. CascadeType.ALL: It is an alternative for performing all the above cascade operations in the hibernate framework

Q7. What is the IOC container and Autowiring?

Ans.) The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are: to instantiate the application class. to configure the object.

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values. It works with reference only.

Q8. What is difference between hibernate and JPA? and What is Hibernate cache.?

Ans.)

JPA	**Hibernate**
Java Persistence API (JPA) defines the management of relational data in the Java applications.	Hibernate is an Object-Relational Mapping (ORM) tool which is used to save the state of Java object into the database.
It is just a specification. Various ORM tools implement it for data persistence.	It is one of the most frequently used JPA implementation.
It is defined in **javax.persistence** package.	It is defined in **org.hibernate** package.
The **EntityManagerFactory** interface is used to interact with the entity manager factory for the persistence unit. Thus, it provides an entity manager.	It uses **SessionFactory** interface to create Session instances.

It uses **EntityManager** interface to create, read, and delete operations for instances of mapped entity classes. This interface interacts with the persistence context.	It uses **Session** interface to create, read, and delete operations for instances of mapped entity classes. It behaves as a runtime interface between a Java application and Hibernate.
It uses **Java Persistence Query Language** (JPQL) as an object-oriented query language to perform database operations.	It uses **Hibernate Query Language** (HQL) as an object-oriented query language to perform database operations.

Hibernate caching improves the performance of the application by pooling the object in the cache. It is useful when we have to fetch the same data multiple times.

There are mainly two types of caching:

- First Level Cache.
- Second Level Cache.

Q10.what is difference between Named Queries and Criteria Queries?

Ans.) Named queries are more optimal (they are parsed/prepared once). Criteria queries are dynamic, (they are not precompiled, although some JPA providers such as EclipseLink maintain a criteria prepare cache).

Practical Questions:

Q1. Please write a program to consume Employee record from service-1 to service-3?

Ans.)

Employee 1:

```
@SpringBootApplication
public class SpringBootSleuthApplication1Application {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootSleuthApplication1Application.class, args);
	}

	@Bean
	public RestTemplate restTemplate() {
		return new RestTemplate();
	}
```

```
        }
```

Controller 1:

```
@RestController
public class AppController1 {

        @Autowired
        RestTemplate restTemplate;

        private static final Logger LOG =
Logger.getLogger(AppController1.class.getName());

        @Bean
        public Sampler alwaysSampler() {
                return Sampler.ALWAYS_SAMPLE;
        }

        @RequestMapping("/application1")
        public String index() {
                LOG.log(Level.INFO, "Index API is calling");
                return "Welcome to server 1! 🎉";
        }

        @RequestMapping("/server2")
        public String getserver2() {
                HttpHeaders headers = new HttpHeaders();
                headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
                HttpEntity<String> entity = new HttpEntity<String>(null, headers);

                return restTemplate.exchange("http://localhost:8092/application2",
HttpMethod.GET, entity, String.class)
                                .getBody();
        }

        @RequestMapping("/server3")
        public String getserver3() {
                HttpHeaders headers = new HttpHeaders();
                headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
                HttpEntity<String> entity = new HttpEntity<String>(null, headers);

                return restTemplate.exchange("http://localhost:8093/application3",
HttpMethod.GET, entity, String.class)
                                .getBody();
        }

        @RequestMapping("/server3viaserver2")
        public String getserver3viaserver2() {
                HttpHeaders headers = new HttpHeaders();
                headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
                HttpEntity<String> entity = new HttpEntity<String>(null, headers);
```

```
		return restTemplate.exchange("http://localhost:8092/server3",
HttpMethod.GET, entity, String.class).getBody();
	}

}
```

Application.properties:

```
server.port=8091
spring.application.name=SleuthServer-1
```

Employee 2:

```
@SpringBootApplication
public class SpringBootSleuthApplication2Application {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootSleuthApplication2Application.class, args);
	}

	@Bean
	public RestTemplate restTemplate() {
		return new RestTemplate();
	}
}
```

Controller 2:

```
@RestController
public class AppController2 {

	@Autowired
	RestTemplate restTemplate;

	private static final Logger LOG =
Logger.getLogger(AppController2.class.getName());

	@Bean
	public Sampler alwaysSampler() {
		return Sampler.ALWAYS_SAMPLE;
	}

	@RequestMapping("/application2")
	public String index() {
		LOG.log(Level.INFO, "Index API is calling");
		return "Welcome to server 2! 💐";
	}

	@RequestMapping("/server1")
	public String getserver2() {
		HttpHeaders headers = new HttpHeaders();
		headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
		HttpEntity<String> entity = new HttpEntity<String>(null, headers);

		return restTemplate.exchange("http://localhost:8091/application1",
HttpMethod.GET, entity, String.class)
				.getBody();
	}

	@RequestMapping("/server3")
	public String getserver3() {
		HttpHeaders headers = new HttpHeaders();
		headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
		HttpEntity<String> entity = new HttpEntity<String>(null, headers);

		return restTemplate.exchange("http://localhost:8093/application3",
HttpMethod.GET, entity, String.class)
				.getBody();
	}

}
```

Application.properties:

```
server.port=8092

spring.application.name=SleuthServer-2
```

Employee 3:

```
@SpringBootApplication
public class SpringBootSleuthApplication3Application {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootSleuthApplication3Application.class, args);
	}

	@Bean
	public RestTemplate restTemplate() {
		return new RestTemplate();
	}

}
```

Controller 3:

```
@RestController
public class AppController3 {

	@Autowired
	RestTemplate restTemplate;
	private static final Logger LOG =
Logger.getLogger(AppController3.class.getName());

	@Bean
	public Sampler alwaysSampler() {
		return Sampler.ALWAYS_SAMPLE;
	}

	@RequestMapping("/application3")
	public String index() {
		LOG.log(Level.INFO, "Index API is calling");
		return "Welcome to server 3! 🎉";
	}

	@RequestMapping("/server1")
	public String getserver2() {
		HttpHeaders headers = new HttpHeaders();
		headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
		HttpEntity<String> entity = new HttpEntity<String>(null, headers);

		return restTemplate.exchange("http://localhost:8091/application1",
HttpMethod.GET, entity, String.class)
				.getBody();
	}

	@RequestMapping("/server2")
	public String getserver3() {
		HttpHeaders headers = new HttpHeaders();
		headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
		HttpEntity<String> entity = new HttpEntity<String>(null, headers);

		return restTemplate.exchange("http://localhost:8092/application2",
HttpMethod.GET, entity, String.class)
				.getBody();
	}

	@RequestMapping("/server1viaserver2")
	public String getserver3viaserver2() {
		HttpHeaders headers = new HttpHeaders();
		headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
		HttpEntity<String> entity = new HttpEntity<String>(null, headers);

		return restTemplate.exchange("http://localhost:8092/server1",
HttpMethod.GET, entity, String.class).getBody();
	}

}
```

Application.properties:

```
server.port=8093
spring.application.name=SleuthServer-3
```

Q2. Please Develop a restful webservice to perform CRUD operations. Entities should have Student, Courses and Teachers.?

Ans.)

Main Application:

```
@SpringBootApplication
public class SpringBootTaskApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootTaskApplication.class, args);
	}

}
```

Student Class:

```
@Entity

@Table(name = "student")

public class Student implements Serializable {


	/**

	 *

	 */

	private static final long serialVersionUID = 1064153100716867148L;


	@Id

	@GeneratedValue(strategy = GenerationType.AUTO)

	private Long rollNumber;


	@Column(name = "StudentName", nullable = true)

	private String name;


	@Column(name = "PhoneNumber", nullable = true)

	private Integer phoneNumber;


	@OneToOne(cascade = CascadeType.ALL)
```

```
private Courses course;

@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name ="student_roll_number")
private List<Teachers> teacher;

public Long getRollNumber() {
        return rollNumber;
}

public void setRollNumber(Long rollNumber) {
        this.rollNumber = rollNumber;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public Integer getPhoneNumber() {
        return phoneNumber;
}

public void setPhoneNumber(Integer phoneNumber) {
        this.phoneNumber = phoneNumber;
}

public Courses getCourse() {
		return course;
	}

	public void setCourse(Courses course) {
		this.course = course;
	}

	public List<Teachers> getTeachers() {
		return teacher;
	}

	public void setTeachers(List<Teachers> teachers) {
		this.teacher = teachers;
	}

	public static long getSerialversionuid() {
		return serialVersionUID;
	}

	@Override
	public String toString() {
		return "Student [rollNumber=" + rollNumber + ", name=" + name + ", phoneNumber=" + phoneNumber + ", course="
				+ course + ", teachers=" + teacher + "]";
	}

}
```

Course Class:

```
@Entity
public class Courses implements Serializable {

	/**
```

```
	 *
	 */
	private static final long serialVersionUID = -1772978403863902891L;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long courseId;

	@Column(name = "CourseName", nullable = true)
	private String courseName;

	public Long getCourseId() {
		return courseId;
	}

	public void setCourseId(Long courseId) {
		this.courseId = courseId;
	}

	public String getCourseName() {
		return courseName;
	}

	public void setCourseName(String courseName) {
		this.courseName = courseName;
	}

	public static long getSerialversionuid() {
		return serialVersionUID;
	}

	@Override
	public String toString() {
		return "Courses [courseId=" + courseId + ", courseName=" + courseName +
"]";
	}

}
```

Teacher Class:

```
@Entity

@Table(name = "Teachers")

public class Teachers implements Serializable {


	/**

	 *

	 */
```

```java
private static final long serialVersionUID = 3751001576456959207L;

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long teacherId;

@Column(name = "TeacherName", nullable = true)
private String name;

@Column(name = "PhoneNumber", nullable = true)
private Integer phoneNumber;

@ManyToOne(cascade = CascadeType.MERGE)
@JoinColumn(name ="student_roll_number")
private Student student;

public Long gettId() {
        return teacherId;
}

public void settId(Long tId) {
        this.teacherId = tId;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}
```

```
	public Integer getPhoneNumber() {
		return phoneNumber;
	}

	public void setPhoneNumber(Integer phoneNumber) {
		this.phoneNumber = phoneNumber;
	}

	public Student getStudent() {
		return student;
	}

	public void setStudent(Student student) {
		this.student = student;
	}

	public static long getSerialversionuid() {
		return serialVersionUID;
	}

	@Override
	public String toString() {
		return "Teachers [tId=" + teacherId + ", name=" + name + ", phoneNumber="
+ phoneNumber + ", student=" + student
				+ "]";
	}

}
```

Student Repository:

```
@Repository
```

```
public interface StudentsRepository extends JpaRepository<Student, Long>{

}
```

Course Repository:

```
@Repository
public interface CoursesRepository extends JpaRepository<Courses, Long>{

}
```

Teacher Repository:

```
@Repository
public interface TeachersRepository extends JpaRepository<Teachers, Long>{

}
```

Student RestController:

```
@RestController
@RequestMapping("/student")
public class StudentsRestController {

	@Autowired
	private StudentsRepository studentsRepository;

	@PostMapping("/create")
	public Student createStudent(@RequestBody Student student) {
		return studentsRepository.save(student);
	}

	@GetMapping("/list")
	public List<Student> listStudents(){
		return studentsRepository.findAll();
	}

	@PutMapping("/update")
	public Student updateStudent(@RequestBody Student student) {
		return studentsRepository.save(student);
	}

	@DeleteMapping("/delete")
	public String deleteStudent(@RequestBody Student student) {
		studentsRepository.delete(student);
		return "Deleted Student Record";
	}
}
```

Course RestController:

```
@RestController
@RequestMapping("/course")
public class CoursesRestController {
	@Autowired
	private CoursesRepository coursesRepository;

	@PostMapping("/create")
	public Courses createCourse(@RequestBody Courses course) {
		return coursesRepository.save(course);
	}

	@GetMapping("/list")
	public List<Courses> listCourses(){
		return coursesRepository.findAll();
	}

	@PutMapping("/update")
	public Courses updateCourses(@RequestBody Courses course) {
		return coursesRepository.save(course);
	}

	@DeleteMapping("/delete")
	public String deleteCourse(@RequestBody Courses course) {
		coursesRepository.delete(course);
		return "Deleted Course Record";
	}
}
```

Teacher RestController:

```
@RestController
@RequestMapping("/teacher")
public class TeachersRestController {

	@Autowired
	private TeachersRepository teachersRepository;

	@PostMapping("/create")
	public Teachers createTeacher(@RequestBody Teachers teacher) {
		return teachersRepository.save(teacher);
	}

	@GetMapping("/list")
	public List<Teachers> listTeachers(){
		return teachersRepository.findAll();
	}

	@PutMapping("/update")
	public Teachers updateTeacher(@RequestBody Teachers teacher) {
		return teachersRepository.save(teacher);
	}

	@DeleteMapping("/delete")
	public String deleteTeacher(@RequestBody Teachers teacher) {
		teachersRepository.delete(teacher);
		return "Deleted Teacher Record";
	}
}
```

Application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/task?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false

spring.datasource.username=root

spring.datasource.password=krishna@1998S

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.generate-ddl=true

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

server.port=8086
```

Q3. Develop wrapper restful webservice to consume downstream restful service using Feign Client. Use service in Step 1 as downstream service.?

Ans.)

Producer Side:

Book Class:

```
@Data

@NoArgsConstructor

@AllArgsConstructor

public class Book {


	private Integer bookId;

	private String bookName;

	private Double bookCost;


	public Integer getBookId() {

		return bookId;

	}
	public void setBookId(Integer bookId) {
		this.bookId = bookId;
	}

	public String getBookName() {
		return bookName;
	}

	public void setBookName(String bookName) {
		this.bookName = bookName;
	}

	public Double getBookCost() {
		return bookCost;
	}

	public void setBookCost(Double bookCost) {
		this.bookCost = bookCost;
	}

	public Book(Integer bookId, String bookName, Double bookCost) {
		super();
		this.bookId = bookId;
		this.bookName = bookName;
		this.bookCost = bookCost;
	}

	@Override
	public String toString() {
		return "Book [bookId=" + bookId + ", bookName=" + bookName + ", 
bookCost=" + bookCost + "]";
```

```
	}

}
```

Book RestController:

```
@RestController
@RequestMapping("/book")
public class EurekaRestController {

	@Autowired
	Environment environment;

	@GetMapping("/data")
	public String getBookData() {

		return "data of BOOK-SERVICE, Running on port: " +
environment.getProperty("local.server.port");
	}

	@GetMapping("/{id}")
	public Book getBookById(@PathVariable Integer id) {
		return new Book(id, "Head First Java", 500.75);
	}

	@GetMapping("/all")
	public List<Book> getAll() {
		return List.of(new Book(501, "Head First Java", 439.75), new Book(502,
"Spring in Action", 340.75),
					new Book(503, "Hibernate in Action", 355.75));
	}

	@GetMapping("/entity")
	public ResponseEntity<String> getEntityData() {
```

```
                return new ResponseEntity<String>("Hello from BookRestController", 
HttpStatus.OK);
        }
}
```

Main Application:

```
@SpringBootApplication
@EnableEurekaClient
public class SpringBootBookProducerApplication {

        public static void main(String[] args) {
                SpringApplication.run(SpringBootBookProducerApplication.class, args);
        }

}
```

Application.properties:

```
server.port=8090
spring.application.name=BOOK-SERVICE
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

Consumer Side:

Book Class:

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Book {

        private Integer bookId;
        private String bookName;
        private Double bookCost;

        public Integer getBookId() {
                return bookId;
```

```
}

public void setBookId(Integer bookId) {
	this.bookId = bookId;
}

public String getBookName() {
	return bookName;
}

public void setBookName(String bookName) {
	this.bookName = bookName;
}

public Double getBookCost() {
	return bookCost;
}

public void setBookCost(Double bookCost) {
	this.bookCost = bookCost;
}

public Book(Integer bookId, String bookName, Double bookCost) {
	super();
	this.bookId = bookId;
	this.bookName = bookName;
	this.bookCost = bookCost;
}

@Override
public String toString() {
```

```
                return "Book [bookId=" + bookId + ", bookName=" + bookName + ",
bookCost=" + bookCost + "]";
        }

}
```

BookRestConsumer:

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Book {

        private Integer bookId;
        private String bookName;
        private Double bookCost;

        public Integer getBookId() {
                return bookId;
        }

        public void setBookId(Integer bookId) {
                this.bookId = bookId;
        }

        public String getBookName() {
                return bookName;
        }

        public void setBookName(String bookName) {
                this.bookName = bookName;
        }
```

```
	public Double getBookCost() {
		return bookCost;
	}

	public void setBookCost(Double bookCost) {
		this.bookCost = bookCost;
	}

	public Book(Integer bookId, String bookName, Double bookCost) {
		super();
		this.bookId = bookId;
		this.bookName = bookName;
		this.bookCost = bookCost;
	}

	@Override
	public String toString() {
		return "Book [bookId=" + bookId + ", bookName=" + bookName + ", 
bookCost=" + bookCost + "]";
	}

}
```

Student RestController:

```
@RestController
@RequestMapping("/student")
public class StudentRestController {

	@Autowired
	private BookRestConsumer consumer;

	@GetMapping("/data")
```

```
        public String getStudentInfo() {
                System.out.println(consumer.getClass().getName()); // prints as a proxy class
                return "Accessing from STUDENT-SERVICE ==> " +
consumer.getBookData();
        }

        @GetMapping("/allBooks")
        public String getBooksInfo() {
                return "Accessing from STUDENT-SERVICE ==> " +
consumer.getAllBooks();
        }

        @GetMapping("/getOneBook/{id}")
        public String getOneBookForStd(@PathVariable Integer id) {
                return "Accessing from STUDENT-SERVICE ==> " +
consumer.getBookById(id);
        }

        @GetMapping("/entityData")
        public String printEntityData() {
                ResponseEntity<String> resp = consumer.getEntityData();
                return "Accessing from STUDENT-SERVICE ==> " + resp.getBody() + " ,
status is:" + resp.getStatusCode();
        }
}
```

Main Application:

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class SpringBootBookConsumerApplication {

        public static void main(String[] args) {
                SpringApplication.run(SpringBootBookConsumerApplication.class, args);
```

```
	}


}
```

Application.properties:

```
server.port=8091

spring.application.name=STUDENT-SERVICE

eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

Q4 Develop Springbatch to read the records(Student, Course, Teachers entities) from csv to
insert into main tables.
Development should cover below
YAML Configuration
Swagger 3.0
Spring JPA
SpringData
Hikari Datasource
Timeouts(DB transaction should be timeout in case of transaction takes more than 5
sec)
Retries(Retry 2 times in case of any failure while connecting to DB)
Circuit Breaker
Rate limiters
Unit Testing
Lombook & Sleuth
Acutator(Metrics & health check URL configuration)
Profiles(Dev, QA,Prod)
Spring Security Basic authentication

Ans.)

Main Application:

```
@SpringBootApplication
@EnableSwagger2
public class SpringBootTaskApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootTaskApplication.class, args);
	}

}
```

Student Class:

```
@Entity
@Table(name = "student")
public class Student implements Serializable {

        /**
         *
         */
        private static final long serialVersionUID = 1064153100716867148L;

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private Long rollNumber;

        @Column(name = "StudentName", nullable = true)
        private String name;

        @Column(name = "PhoneNumber", nullable = true)
        private Integer phoneNumber;

        @OneToOne(cascade = CascadeType.ALL)
        private Courses course;

        @OneToMany(cascade = CascadeType.ALL)
        @JoinColumn(name ="student_roll_number")
        private List<Teachers> teacher;

        public Long getRollNumber() {
                return rollNumber;
        }
```

```
public void setRollNumber(Long rollNumber) {
        this.rollNumber = rollNumber;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public Integer getPhoneNumber() {
        return phoneNumber;
}

public void setPhoneNumber(Integer phoneNumber) {
        this.phoneNumber = phoneNumber;
}

public Courses getCourse() {
        return course;
}

public void setCourse(Courses course) {
        this.course = course;
}

public List<Teachers> getTeachers() {
        return teacher;
}
```

```
	public void setTeachers(List<Teachers> teachers) {

		this.teacher = teachers;

	}


	public static long getSerialversionuid() {

		return serialVersionUID;

	}


	@Override

	public String toString() {

		return "Student [rollNumber=" + rollNumber + ", name=" + name + ", 
phoneNumber=" + phoneNumber + ", course="

				+ course + ", teachers=" + teacher + "]";

	}


}
```

Course Class:

```
@Entity
public class Courses implements Serializable {

	/**
	 * 
	 */
	private static final long serialVersionUID = -1772978403863902891L;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long courseId;

	@Column(name = "CourseName", nullable = true)
	private String courseName;

	public Long getCourseId() {
		return courseId;
	}

	public void setCourseId(Long courseId) {
		this.courseId = courseId;
	}

	public String getCourseName() {
		return courseName;
	}

	public void setCourseName(String courseName) {
		this.courseName = courseName;
	}

	public static long getSerialversionuid() {
		return serialVersionUID;
	}

	@Override
	public String toString() {
		return "Courses [courseId=" + courseId + ", courseName=" + courseName +
"]";
	}

}
```

Teacher Class:

```
@Entity

@Table(name = "Teachers")

public class Teachers implements Serializable {


	/**

	 *

	 */

	private static final long serialVersionUID = 3751001576456959207L;


	@Id

	@GeneratedValue(strategy = GenerationType.AUTO)

	private Long teacherId;


	@Column(name = "TeacherName", nullable = true)

	private String name;


	@Column(name = "PhoneNumber", nullable = true)
```

```
}

private Integer phoneNumber;

@ManyToOne(cascade = CascadeType.MERGE)
@JoinColumn(name ="student_roll_number")
private Student student;

public Long gettId() {
	return teacherId;
}

public void settId(Long tId) {
	this.teacherId = tId;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public Integer getPhoneNumber() {
	return phoneNumber;
}

public void setPhoneNumber(Integer phoneNumber) {
	this.phoneNumber = phoneNumber;
}

public Student getStudent() {
        return student;
    }

    public void setStudent(Student student) {
        this.student = student;
    }

    public static long getSerialversionuid() {
        return serialVersionUID;
    }

    @Override
    public String toString() {
        return "Teachers [tId=" + teacherId + ", name=" + name + ", phoneNumber="
+ phoneNumber + ", student=" + student
                + "]";
    }

}
```

Student Repository:

```
@Repository
public interface StudentsRepository extends JpaRepository<Student, Long>{

}
```

Course Repository:

```
@Repository
public interface CoursesRepository extends JpaRepository<Courses, Long>{

}
```

Teacher Repository:

```
@Repository
public interface TeachersRepository extends JpaRepository<Teachers, Long>{

}
```

Student RestController:

```
@RestController
@RequestMapping("/student")
public class StudentsRestController {

	@Autowired
	private StudentsRepository studentsRepository;

	@PostMapping("/create")
	public Student createStudent(@RequestBody Student student) {
		return studentsRepository.save(student);
	}

	@GetMapping("/list")
	public List<Student> listStudents(){
		return studentsRepository.findAll();
	}

	@PutMapping("/update")
	public Student updateStudent(@RequestBody Student student) {
		return studentsRepository.save(student);
	}

	@DeleteMapping("/delete")
	public String deleteStudent(@RequestBody Student student) {
		studentsRepository.delete(student);
		return "Deleted Student Record";
	}
}
```

Course RestController:

```
@RestController
@RequestMapping("/course")
public class CoursesRestController {

	@Autowired
	private CoursesRepository coursesRepository;

	@PostMapping("/create")
	public Courses createCourse(@RequestBody Courses course) {
		return coursesRepository.save(course);
	}

	@GetMapping("/list")
	public List<Courses> listCourses(){
		return coursesRepository.findAll();
	}

	@PutMapping("/update")
	public Courses updateCourses(@RequestBody Courses course) {
		return coursesRepository.save(course);
	}

	@DeleteMapping("/delete")
	public String deleteCourse(@RequestBody Courses course) {
		coursesRepository.delete(course);
		return "Deleted Course Record";
	}
}
```

Teacher RestController:

```
@RestController
@RequestMapping("/teacher")
public class TeachersRestController {

	@Autowired
	private TeachersRepository teachersRepository;

	@PostMapping("/create")
	public Teachers createTeacher(@RequestBody Teachers teacher) {
		return teachersRepository.save(teacher);
	}

	@GetMapping("/list")
	public List<Teachers> listTeachers(){
		return teachersRepository.findAll();
	}

	@PutMapping("/update")
	public Teachers updateTeacher(@RequestBody Teachers teacher) {
		return teachersRepository.save(teacher);
	}

	@DeleteMapping("/delete")
	public String deleteTeacher(@RequestBody Teachers teacher) {
		teachersRepository.delete(teacher);
		return "Deleted Teacher Record";
	}
}
```

Application.yml:

```
endpoints:

  beans:

    enabled: true

    id: springbeans

    sensitive: false

  health:

    sensitive: false
```

```
info:
  app:
    description: This is a spring boot Task in which Swagger, Actuator and Kafka are
implemented
    name: Spring Boot Task
    version: 1.0.0
kafka:
  bootstrapAddress: 127.0.0.1
management:
  address: 127.0.0.1
  endpoint:
    health:
      group:
        custom:
          include: diskSpace,ping
          show-components: always
          show-details: always
          status:
            http-mapping:
              up: 207
    shutdown:
      enabled: true
  endpoints:
    web:
      exposure:
        exclude: loggers
        include: '*'
  port: 8081
  security:
    enabled: true
    role: SUPERUSER
myname: krishnaReddy
```

```
security:
  user:
    name: admin
    password: secret
server:
  error:
    whitelabel:
      enabled: true
  port: 9090
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: krishna@1998S
    url:
jdbc:mysql://localhost:3306/task?useSSL=false&serverTimezone=UTC&useLegacyDatetime
Code=false
    username: root
  jpa:
    database-platform: org.hibernate.dialect.MySQL8Dialect
    generate-ddl: true
    hibernate:
      show-sql: true
  kafka:
    bootstrap-servers: 127.0.0.1:9092
    consumer:
      group-id: mygroup
```

Q5. Deploy the above application in Docker container?

Ans.)

Q6. How Rest API Call/consume happens in SpringBoot? RestTemplate, Feign Client?

Ans.) 1.) Create Spring Boot Project.

2.) Create Rest Controllers and map API requests.

3.) Build and run the Project.

4.) Make a call to external API services and test it.

Q7. What is Spring Profile?

Ans.) Spring Profiles helps segregating your application configurations, and make them available only in certain environments. An application run on many different environments. For example, Dev, QA, Test, Stage, Production etc.

Q8. Demonstrated Springboot actuator?

Ans.) In essence, Actuator brings production-ready features to our application.

Monitoring our app, gathering metrics, understanding traffic, or the state of our database become trivial with this dependency.

The main benefit of this library is that we can get production-grade tools without having to actually implement these features ourselves.

Actuator is mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.

Once this dependency is on the classpath, several endpoints are available for us out of the box. As with most Spring modules, we can easily configure or extend it in many ways.

To enable Spring Boot Actuator, we just need to add the *spring-boot-actuator* dependency to our package manager.

/auditevents lists security audit-related events such as user login/logout. Also, we can filter by principal or type among other fields.

/beans returns all available beans in our BeanFactory. Unlike /auditevents, it doesn't support filtering.

/conditions, formerly known as /autoconfig, builds a report of conditions around autoconfiguration.

/configprops allows us to fetch all @ConfigurationProperties beans.

/env returns the current environment properties. Additionally, we can retrieve single properties.

/flyway provides details about our Flyway database migrations.

/health summarizes the health status of our application.

/heapdump builds and returns a heap dump from the JVM used by our application.

/info returns general information. It might be custom data, build information or details about the latest commit.

/liquibase behaves like /flyway but for Liquibase.

/logfile returns ordinary application logs.

/loggers enables us to query and modify the logging level of our application.

/metrics details metrics of our application. This might include generic metrics as well as custom ones.

/prometheus returns metrics like the previous one, but formatted to work with a Prometheus server.

/scheduledtasks provides details about every scheduled task within our application.

/sessions lists HTTP sessions given we are using Spring Session.

/shutdown performs a graceful shutdown of the application.

/threaddump dumps the thread information of the underlying JVM.

Q9. What is Junit and annotations used in junit? write all test cases for above given example

Ans.)

Q10.List Annotations from springboot?

Ans.) Basic Setup:

- @SpringBootApplication
- @Configuration
- @ComponentScan
- @EnableAutoConfiguration

Request Responses:

- @GetMapping
- @RequestMapping
- @RequestParam

Component Types:

- @Component
- @Service
- @Repository
- @Controller
- @RestController

Testing:

- @SpringBootTest

@MockBean

@Validated

Misc:

@Bean

@ConditionalOnJava