CONTENTS

Spring Framework

Spring is a lightweight framework.
Spring provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.
The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

Advantages of Spring Framework

1. **Predefined Templates**: Spring provides templates for JDBC, Hibernate, JPA, etc., reducing the need for writing boilerplate code.

2. **Loose Coupling**: Dependency injection ensures loose coupling between components.

3. **Easy Testing**: Spring makes it easier to test applications without requiring a server.

4. **Lightweight**: Spring's POJO (Plain Old Java Object) implementation keeps it non-invasive and lightweight.

5. **Fast Development**: Dependency Injection and support for various frameworks facilitate rapid JavaEE application development.

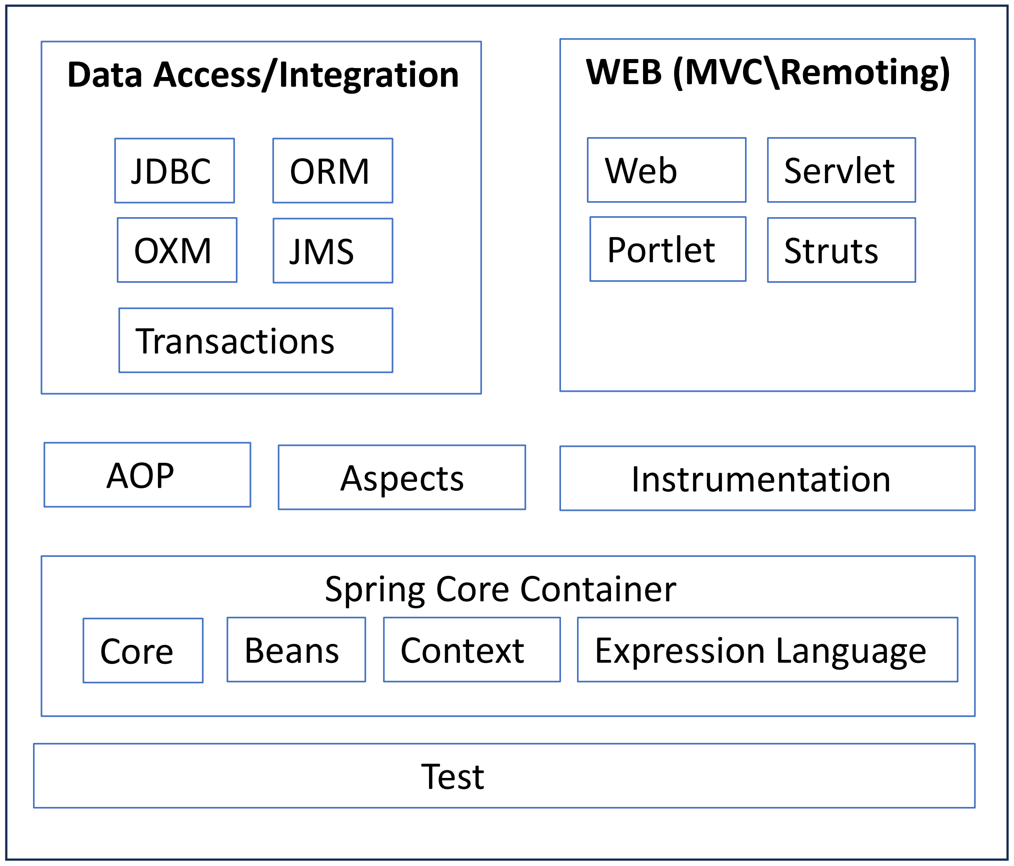
6. **Powerful Abstraction**: Spring abstracts JavaEE specifications like JMS, JDBC, JPA, and JTA[2345].

Spring Modules

The Spring framework comprises of many modules

- Core
- Beans
- Context
- Expression language
- AOP
- Aspects, Instrumentation
- JDBC
- ORM
- OXM
- JMS
- Transaction
- Web
- Servlet
- Struts

Spring Framework Runtime



Spring Boot

Spring Boot is a project that is built on the top of the Spring Framework.

It is easy to set up, configure and run both simple and web-based applications.

It is a Spring module that provides the feature to the Spring Framework. RAD (Rapid Application Development).

Spring Boot is the combination of Spring Framework and Embedded Servers.

Spring Framework + Embedded HTTP Servers(Tomcat, Jetty) – XML<bean> Configuration or @Configuration = Spring Boot

Prerequisite of Spring Boot Features

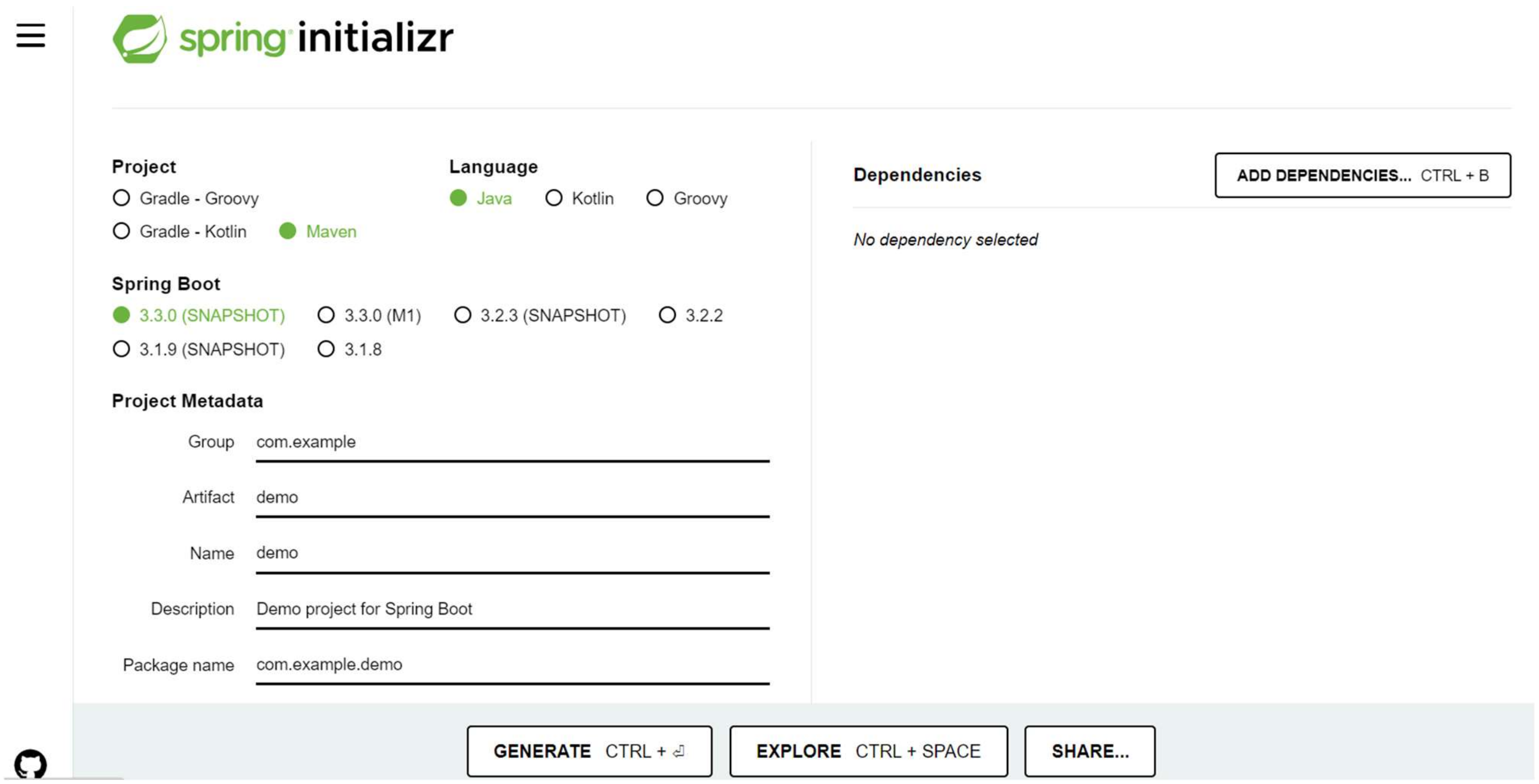
- Java 1.8
- Maven 3.0+
- Spring Framework 5.0.0.BUILD-SNAPSHOT
- An IDE (Spring Tool Suite) is recommended.

Spring Boot Features

- Web Development
- Spring Application
- Application events and listeners
- Admin features
- Externalized Configuration
- Properties Files
- YAML Support
- Type-safe Configuration
- Logging
- Security

Spring Boot Initializr

In any web browser if we type https://start.spring.io/, we will get user interface of spring initializr



Project
- Gradle - Groovy
- Gradle - Kotlin
- [✓ Maven]

Language
- [✓ Java]
- Kotlin
- Groovy

Spring Boot
- [✓ 3.3.0 (SNAPSHOT)]
- 3.3.0 (M1)
- 3.2.3 (SNAPSHOT)
- 3.2.2
- 3.1.9 (SNAPSHOT)
- 3.1.8

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo

Dependencies

No dependency selected

Hibernate

Hibernate is an Object-Relational Mapping (ORM) tool which bridges gap between object oriented features of Java and relational database.

Hibernate maps Java objects to database tables and vice versa.

Advantages of Hibernate Framework

1. Open Source and Lightweight
2. Fast Performance
3. Database Independent Query
4. Automatic Table Creation
5. Simplifies Complex Joins.
6. Query Statistics and Database Status.

How Does Hibernate Work?

- Hibernate abstracts away many low-level details, such as establishing database connections, writing CRUD (Create, Read, Update, Delete) operations, and handling transactions.
- It uses **annotations** or **XML configuration** to define the mapping between Java classes and database tables.
- When you perform operations on Java objects (such as saving, updating, or retrieving data), Hibernate translates these actions into SQL queries and manages the database interactions behind the scenes

JPA (Java Persistence API)

JPA (Java Persistence API)

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools.
The javax.persistence package contains the JPA classes and interfaces.

JpaRepository

JpaRepository is a JPA (Java Persistence API) specific extension of Repository. It contains the full API of CrudRepository and PagingAndSortingRepository. So it contains API for basic CRUD operations and also API for pagination and sorting.

Methods of JpaRepository

a) save() → Saves the object in the database table

b) findAll() → fetches all the records of table

c) findById() → fetch a record by id

d) deleteById() → delete a record by id

Restful API

A RESTful API, or simply a REST API, is a web service that follows the principles of Representational State Transfer (REST) architecture.

Features of Rest API

a) Rest API is stateless, which means that the server does not store any information about the client's state. In the request itself all the information is included and server does not need to store any thing of client’s information.

b) Rest API is that it uses HTTP methods, such as GET, POST, PUT and DELETE, to indicate the type of operation the client requests.

GET request retrieves information from the server.

POST request is used to submit information to the server.

c) REST API is that it is platform-independent, which means that it can be used with any programming language or framework that supports HTTP.

API Method

To create API methods using Java, our computer system should have following prerequisites

- Java Development Kit (JDK) 8 or later installed on your system.
- An Integrated Development Environment (IDE) like Eclipse or IntelliJ IDEA.
- Familiarity with Java programming, Spring and HTTP concepts.

Steps to create API methods

1. Set up the project.

The project can be set up by spring boot initialzr.

In any web browser if we type https://start.spring.io/, we will get user interface of spring initializr.

In the spring initializr select java language and maven project.

2. Define the model

The object model is a system or interface which is basically used to visualize elements in terms of objects in a software application.

```
@Entity
public class Employee
{
            long empId,
            string empName,
            double salary
}
```

Inside the above class programmer need to write the setter and getter of properties of Employee class.

3. Implementing the repository

```
public interface EmployeeRepository extends JpaRepository<Employee, long>{}
```

4. Building the Service Layer

Service layer acts as a bridge between data access layer and controller layer.

```
public interface EmployeeService
{
        void updateEmployee(Employee employee);
}

public class EmployeeServiceImpl implements EmployeeService
{
        private final EmployeeRepository empRepository;
        @Override
        public void updateEmployee(Employee employee)
        {
                empRepository.save(employee);
        }
}
```

5. Creating Controller

```
@RestController
@RequestMapping("/api/employee")
public class EmployeeController
{
            private final EmployeeService employeeService;
            @PostMapping
            public void saveEmployee(@RequestBody Employee employee)
            {
                        employeeService.save(employee);
            }
}
```

6. Setting database configuration in application.properties file.

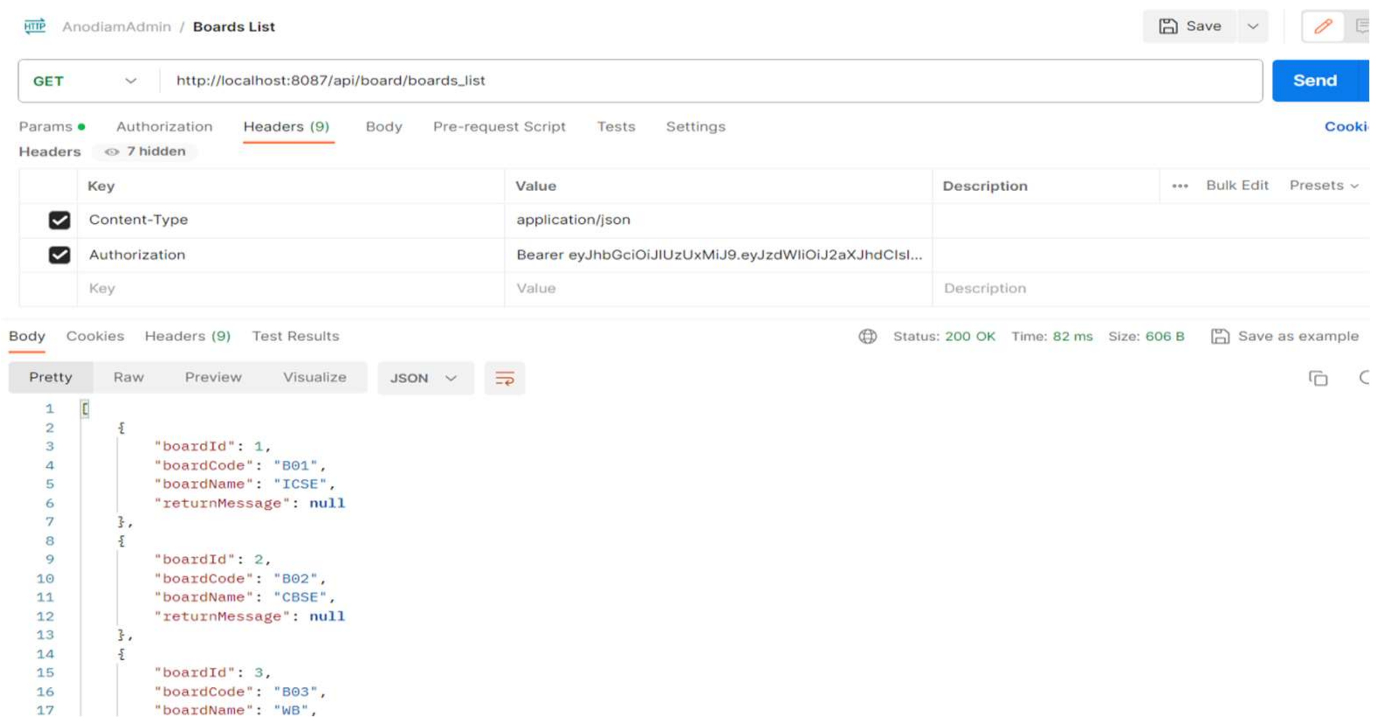
```
# Database Configuration
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.datasource.url=jdbc:mysql://localhost:3306/employee_db?useSSL=false&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=dfT$89*#@

# Hibernate Properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Testing API Method

Postman

Postman is an API platform for building and using APIs.

Through postman API methods are tested.



http://localhost:8087/api/board/boards_list

	Key	Value	Description
✓	Content-Type	application/json	
✓	Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ2aXJhdClsl...	
	Key	Value	Description

Status: 200 OK Time: 82 ms Size: 606 B

```
[
    {
        "boardId": 1,
        "boardCode": "B01",
        "boardName": "ICSE",
        "returnMessage": null
    },
    {
        "boardId": 2,
        "boardCode": "B02",
        "boardName": "CBSE",
        "returnMessage": null
    },
    {
        "boardId": 3,
        "boardCode": "B03",
        "boardName": "WB",
```

JWT (JSON Web Token)

JWT is an open standard for securely sharing JSON data between parties.

The data is encoded and digitally signed, which ensures its authenticity.

JWT is widely used in API authentication and authorization workflows, as well as for data transfer between clients and servers.

JWT provides secure way of transferring statements referred to as “claims” about an entity (typically the user) between two parties.

Claims in JWT are key-value pairs that are represented with either JWS (JSON Web Signature) or JWE (JSON Web Encryption).

JWS and JWE are standards for digitally signing and encrypting JWTs, which helps ensure their authenticity and integrity.

Structure of JWT

A JWT has three sections: a header, a payload, and a signature.

Each section is a Base64-encoded string, and the sections are separated by periods.

A typical JWT looks like this, where the X’s represent the header, the Y’s represents the payload, and the Z’s represents the signature:

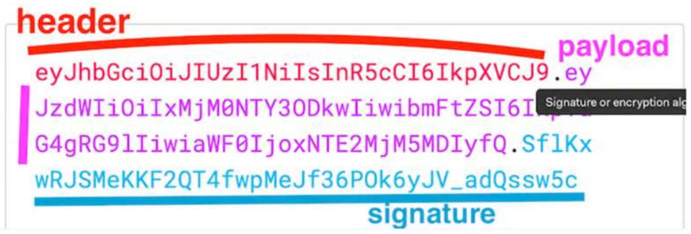
xxxxxx.yyyyyy.zzzzzz

xxxxxx → Header

yyyyyy → Payload

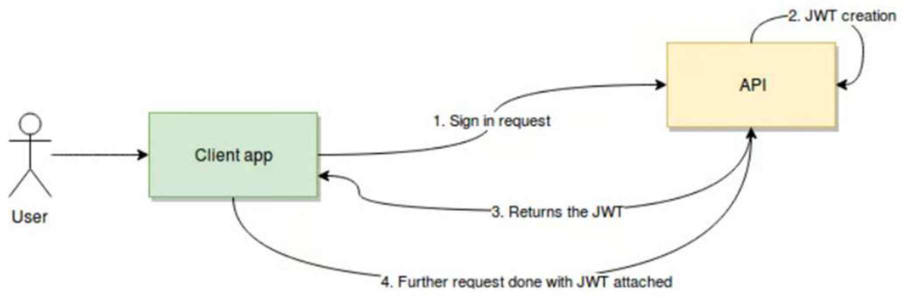
zzzzzz → Signature

JWT will look this



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6I
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

How does JWT gets created ?



As shown above user sends login request to the server from the client application.

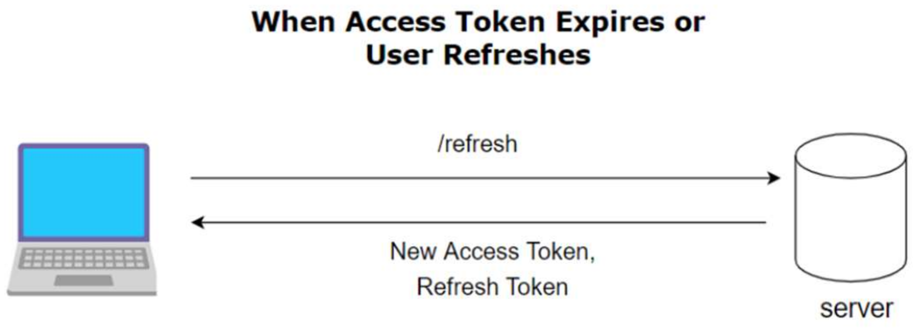
API on the server end creates JWT.

JWT is returned to the client application from the server end.

From the client’s end the further request is done with JWT attached.

Refresh JWT (JSON Web Token)

Refresh token is a unique token that is used to obtain additional access tokens and it is obtained without having to collect credentials every time one expires.



In the application.properties file we need to write properties for generation of token and refresh token

```
anodiam.app.jwtSecret= mySecretKey-123456jsfwwrwsdfdssgsgeeeybvdgh
anodiam.app.jwtExpirationMs= 86400
anodiam.app.refreshExpirationDateInMs=9000000
anodiam.app.jwtRefreshExpirationMs=86400000
```

Spring boot – Eureka Server & Client

In order to communicate between services, the following components are required

- microservices architectures
- service registration
- discovery

Eureka Server is a service registry that plays a central role in the automatic detection of devices and services on a network. It acts as the heart of your microservices ecosystem, allowing service instances to register themselves and facilitating service discovery. Key aspects of Eureka Server include:

- Client Registration: Instances of microservices automatically register themselves with Eureka Server.
- Service Discovery: Eureka Server maintains a registry of all client applications running on different ports and IP addresses.

Eureka Server operates on a simple "Register, Lookup, Connect" principle, making it an excellent choice for managing microservices in a Spring Boot environment. Here are some compelling reasons to use Eureka Server:

1. Centralized Service Registry: Eureka Server knows about all client applications and their locations. This centralization simplifies service discovery.
2. Automatic Registration: Microservices automatically register themselves with Eureka Server, reducing manual configuration efforts.
3. Load Balancing: Eureka Server can help implement load balancing among service instances.
4. Health Checks: Eureka Server can perform health checks on registered services, ensuring robustness and reliability.
5. Integration with Spring Cloud: Eureka Server seamlessly integrates with the Spring Cloud ecosystem, enabling easy scaling and deployment.

In the java project's pom.xml on server and client side

```xml
<!-- For Eureka Server -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>

<!-- For Eureka Client -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

To enable Eureka functionality, the programmer needs to enable annotation

@EnableEurekaServer: Use this annotation to designate a Spring Boot application as the central instance where other services will get registered.

@EnableEurekaClient: Services use this annotation to enable service registration and discovery with the central Eureka Server.

For marking a Spring Boot Application as an Eureka Server

```
@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {
     public static void main(String[] args) {
        SpringApplication.run(ServiceRegistryApplication.class, args);
    }
}
```

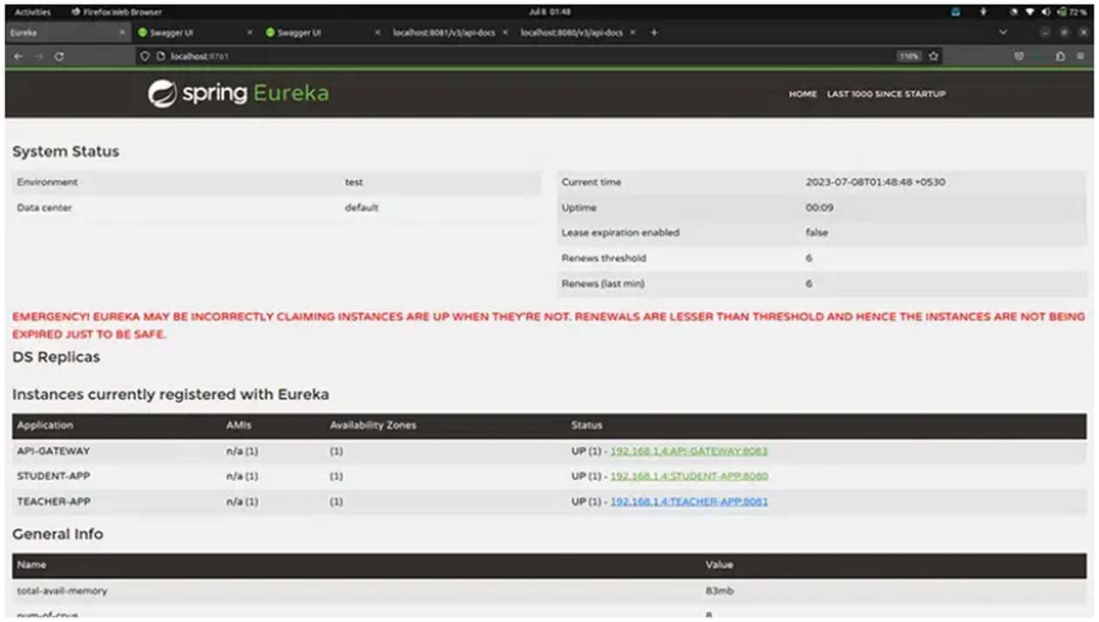
For marking a Spring Boot Application as an Eureka Client to be used up by the server

```
@SpringBootApplication
@EnableEurekaClient
public class StudentApplication {
    public static void main(String[] args) {
        SpringApplication.run(StudentApplication.class, args);
    }
}
```

Eureka Dashboard

When the Spring Boot application is up and running, just hit the default URL – http://localhost:8761.

Dashboard will display the list of micro services



System Status

Environment	test
Data center	default

Current time	2023-07-08T01:48:48 +0530
Uptime	00:09
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	6

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.1.4:API-GATEWAY:8083
STUDENT-APP	n/a (1)	(1)	UP (1) - 192.168.1.4:STUDENT-APP:8080
TEACHER-APP	n/a (1)	(1)	UP (1) - 192.168.1.4:TEACHER-APP:8081

General Info

Name	Value
total-avail-memory	83mb
num-of-cpus	8

Dashboard of Eureka discovery client