Spring For Gold

Table of Contents

Dependency Injection

When we create 2 classes, [Student, Level] in the level, we can initialise the Student class by using Student student = new Student(); but when we have 1000s of classes, then this process become tedious to do. That's is where

inversion of controls comes into play: We give spring the authority to create the objects of the classes

One way of achieving this is through dependency injection, all the objects of the classes will be auto created as Beans into one spring container, so whenever you need a specific Bean, you will just call it from the container. No need to create any object, spring will do that for you

Spring Boot Intialiser

We need to initialise the application at https://start.spring.io/ then provide the configurations

Open the generated bootstrap code in intellij IDE

Spring Boot API

In the com.hydottech.Springboot.Tutorial, create a package and call it controller in the controller package, create a sample controller as a java class file, call it HelloController

```
package com.hydottech.Springboot.Tutorial.controller;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;
@RestController

public class HelloController {

@GetMapping("/")
   public String helloWorld(){
      return "Hello Money";
   }


}
```

Very Simple. You can run from the terminal using the command mvn spring-boot: run

Spring Boot DevTools

This dependency will help to auto run the spring application when changes have been detected

add this dependency to pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
</dependency>
```

Reload the maven project to add the dependency
Apply the changes in the IDEs

Spring Boot Database Connection(H2 MEMORY)

Add the following dependencies in your pom.xml

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
</dependency>
```

In the **application.properties**, add this configurations

```
spring.application.name=Spring-Boot-Tutorial
server.port=5000
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:dcbapp
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-
platform=org.hibernate.dialect.H2Dialect
```

Inside the browser, access it through this url
[http://localhost:5000/h2-console]

Input your password to make a connection

Spring Boot Components

Inside the com.hydottech.Springboot.Tutorial, create multiple packages like entity, service, repository, controller
Entity => Models [Annotations: `@Entity`]
Service => [Annotations: `@Service`]
Repository => [Annotations: `@Repository`]
Controller => For Http Methods [Annotations: `@RestController`]

Entity

In the entity package create a new Department model class file

```
package com.hydottech.Springboot.Tutorial.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long departmentId;
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
   //To generate the getters and setters, press command +
n, select getters and setters, highlight all the field we
will generate getters and setters for, press okay
//Use the same method for their constructors
//departmentId is a primary key and it will be auto
generated as a numeric value


   public Long getDepartmentId() {
      return departmentId;
   }
   public Department(Long departmentId, String
departmentName, String departmentAddress, String
departmentCode) {
      this.departmentId = departmentId;
      this.departmentName = departmentName;
      this.departmentAddress = departmentAddress;
      this.departmentCode = departmentCode;
   }
```

```
    public Department() {
    }
    @Override
    public String toString() {
        return "Department{" +
                "departmentId=" + departmentId +
                ", departmentName='" + departmentName + '\'' +
                ", departmentAddress='" + departmentAddress +
'\'' +
                ", departmentCode='" + departmentCode + '\'' +
                '}';
    }
    public void setDepartmentId(Long departmentId) {
        this.departmentId = departmentId;
    }
    public String getDepartmentName() {
        return departmentName;
    }
    public void setDepartmentName(String departmentName) {
        this.departmentName = departmentName;
    }
    public String getDepartmentAddress() {
        return departmentAddress;
    }
    public void setDepartmentAddress(String
departmentAddress) {
        this.departmentAddress = departmentAddress;
    }
    public String getDepartmentCode() {
        return departmentCode;
    }
    public void setDepartmentCode(String departmentCode) {
        this.departmentCode = departmentCode;
    }
}
```

Not defficult, command + N does the trick

Service

In the service create a standard approach, an interface and its implementation

create DepartmentService as an implementation as this

```
package com.hydottech.Springboot.Tutorial.service;
public interface DepartmentService {
}
```

Create DepartmentServiceImpl to implement the service as this

```
package com.hydottech.Springboot.Tutorial.service;
import org.springframework.stereotype.Service;
@Service
public class DepartmentServiceImpl implements
DepartmentService{
}
```

Repository

In the Repository folder, create a DepartmentRepository as an interface that extends all the functionalities in `JpaRepository`

```
package com.hydottech.Springboot.Tutorial.repository;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface DepartmentRepository extends
JpaRepository<Department,Long> {
}
```

Department is the model or entity
Long is the type of it primary key

CRUD Operation

Spring Boot Save Data

This is the application Flow

FrontEnd

Using FormData

departmentName Kitchen

departmentAddress Accra

departmentCode KT125

Entity

Department.java

```
package com.hydottech.Springboot.Tutorial.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long departmentId;
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
   //To generate the getters and setters, press command +
n, select getters and setters, highlight all the field we
will generate getters and setters for, press okay
//Use the same method for their constructors
//departmentId is a primary key and it will be auto
generated as a numeric value
   public Long getDepartmentId() {
      return departmentId;
   }
   public Department(Long departmentId, String
departmentName, String departmentAddress, String
departmentCode) {
      this.departmentId = departmentId;
      this.departmentName = departmentName;
      this.departmentAddress = departmentAddress;
      this.departmentCode = departmentCode;
   }
   public Department() {
   }
   @Override
   public String toString() {
```

```
      return "Department{" +
            "departmentId=" + departmentId +
            ", departmentName='" + departmentName + '\'' +
            ", departmentAddress='" + departmentAddress +
'\'' +
            ", departmentCode='" + departmentCode + '\'' +
            '}';
   }
   public void setDepartmentId(Long departmentId) {
      this.departmentId = departmentId;
   }
   public String getDepartmentName() {
      return departmentName;
   }
   public void setDepartmentName(String departmentName) {
      this.departmentName = departmentName;
   }
   public String getDepartmentAddress() {
      return departmentAddress;
   }
   public void setDepartmentAddress(String
departmentAddress) {
      this.departmentAddress = departmentAddress;
   }
   public String getDepartmentCode() {
      return departmentCode;
   }
   public void setDepartmentCode(String departmentCode) {
      this.departmentCode = departmentCode;
   }
}
```

Controller

DepartmentController.java

```
package com.hydottech.Springboot.Tutorial.controller;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
com.hydottech.Springboot.Tutorial.service.DepartmentService
;
import
com.hydottech.Springboot.Tutorial.service.DepartmentService
Impl;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
@RestController
```

```
public class DepartmentController {
   @Autowired
   private DepartmentService departmentService;
   @PostMapping("/departments")
   public Department saveDepartment(@ModelAttribute
Department department) {
      return departmentService.saveDepartment(department);
   }
/*
* Explanation
*
*  @Autowired
   private DepartmentService departmentService;
   This will bind the service with a global service stored
by spring
   *
  @PostMapping("/departments")
   public Department saveDepartment(@RequestBody Department
department){
      return departmentService.saveDepartment(department);
   };
1.  @PostMapping("/departments") this is a post method
2. public Department saveDepartment(@RequestBody Department
department),
 a public with the Entity type Department with a name,
saveDepartment and the JSON body of type Department
3.  return departmentService.saveDepartment(department);
inside the DepartmentService interface we are calling the
saveDepartment method in it,
When we highlight we can auto create the methods in the
interface as this
 public Department saveDepartment(Department department);
 This will generate an error in the DepartmentServiceImpl,
implementing the methods can auto generate like this
  @Override
   public Department saveDepartment(Department department)
{
      return null;
   }
*
*
*
*
*
* */
}
```

From the Controller the service method will be auto created

Service

DepartmentService.java as the interface

```
package com.hydottech.Springboot.Tutorial.service;
import com.hydottech.Springboot.Tutorial.entity.Department;
public interface DepartmentService {
  public Department saveDepartment(Department department);
}
```

From the DepartmentService.java the DepartmentServiceImpl.java method will be auto created

DepartmentServiceImpl.java as the implementation

```
package com.hydottech.Springboot.Tutorial.service;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
com.hydottech.Springboot.Tutorial.repository.DepartmentRepo
sitory;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class DepartmentServiceImpl implements
DepartmentService{
   @Autowired
   private DepartmentRepository departmentRepository;
   @Override
   public Department saveDepartment(Department department)
{
      return departmentRepository.save(department);
   }
}
```

Repository

```
package com.hydottech.Springboot.Tutorial.repository;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
```

```
public interface DepartmentRepository extends
JpaRepository<Department,Long> {
}
```

Spring Boot Get Data

Same Process, We will go from the controller to serviceInterface to serviceImplementation where the repository will be called

Controller

```
@GetMapping("/getDepartments")
public List<Department> fetchDepartments(){
   return departmentService.fetchDepartments();
}
```

Service Interface will be auto created as

```
public List<Department> fetchDepartments();
```

Service Implementation with data repository

```
@Override
public List<Department> fetchDepartments() {
   return departmentRepository.findAll();
}
```

Spring Boot Get Specific Data

From the Entity Model we defined the departmentId as the primaryKey, so it means that any findById will match with the departmentId.

Let start the process

Controller

```
@GetMapping("/getSpecificDepartment/{Id}")
public Department fetchOneDepartment(@PathVariable("Id")
long Id){
   return departmentService.fetchOneDepartments(Id);
}
```

Service Interface

```
public Department fetchOneDepartments(long id);
```

Service Implementation

```
@Override
public Department fetchOneDepartments(long id) {
   return departmentRepository.findById(id).get();
}
```

Spring Boot Delete Specific Data

Controller

```
@DeleteMapping("/departments/{Id}")
public String deleteOneDepartment(@PathVariable("Id") long
Id){
   departmentService.deleteOneDepartments(Id);
   return "Department deleted successfully";
}
```

Service Interface

```
public void deleteOneDepartments(long id);
```

Service Implementation

```
@Override
public void deleteOneDepartments(long id) {
   departmentRepository.deleteById(id);
}
```

Spring Boot Update Specific Data

For Put, Instead of manually Updating it one by one, there is a shortcut

Controller

```
@PutMapping("/departments/{Id}")
public Department updateDepartment(@PathVariable("Id") long
Id, @ModelAttribute Department department){
   return departmentService.updateDepartment(Id,
department);
}
```

Service Interface

```
public Department updateDepartment(long id, Department
department);
```

Service Implementation

```
@Override
```

```
public Department updateDepartment(long id, Department
department) {
   Department db =
departmentRepository.findById(id).orElseThrow(() -> new
RuntimeException("Department not found"));
   // Get all the fields of the Department class
   Field[] fields = Department.class.getDeclaredFields();
   for (Field field : fields) {
      field.setAccessible(true);
      try {
         // Get the value of the current field from the
input department
         Object value = field.get(department);
         // Check if the value is not null and not an empty
string
         if (Objects.nonNull(value)
&& !"".equals(value.toString().trim())) {
            // Set the value of the current field in the db
department
            field.set(db, value);
         }
      } catch (IllegalAccessException e) {
         e.printStackTrace(); // Handle exception as
appropriate
      }
   }
   return departmentRepository.save(db);
}
```

Generic Api's

To Get the specific name of the department, the data repository does the trick

Let start with the **controller**

```
@GetMapping("/departmentsName/{Name}")
public Department
fetchDepartmentByName(@PathVariable("Name") String Name){
   return departmentService.fetchDepartmentByName(Name);
}
```

Service Interface

```
public Department fetchDepartmentByName(String name);
```

Service Implementation

```
@Override
```

```
public Department fetchDepartmentByName(String name) {
   return departmentRepository.findByDepartmentName(name);
}
```

Repository

The field name is `departmentName` so in the datarepository, type findBy+the fieldName in camel case. That all

```
public Department findByDepartmentName(String
departmentName);
we can even execute sql queries directly
@Query(value = "SELECT * FROM DEPARTMENT WHERE
DEPARTMENT_NAME ILIKE :departmentName", nativeQuery = true)
public Department
findByDepartmentNameIgnoreCaseKofi(@Param("departmentName")
String departmentName);
```

External Links

For More Information Visit the Following url

Docs: https://docs.spring.io/spring-data/jpa/reference/#jpa.query-methods

Spring Data JPA: **https://www.youtube.com/watch?v=XszpXoIl9Sg&t=0s**

Spring Security:**https://www.youtube.com/watch?v=tWcqSIQr6Ks&list=PLhfxuQVMs-nzbKxB2Zb7F9kjXntJhcP5k&index=7**

Microservices: **https://www.youtube.com/watch?v=BnknNTN8icw&t=0s**

Validation

In the pom.xml add the dependency for validation

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

In the entity, add the validation to the field

```
@NotBlank(message="Name is required")
private String departmentName;
```

In the controller add the @Valid annotation

```
@PostMapping("/departments")
public Department saveDepartment(@Valid @ModelAttribute
Department department) {
   return departmentService.saveDepartment(department);
}
```

Loggers

In the resources folder create the logback.xml and add this configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <!-- Define the file appender -->
   <appender name="FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
      <file>logs/application.log</file>
      <!-- Define a rolling policy to manage log file
rotation -->
      <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
         <!-- rollover daily -->
         <fileNamePattern>logs/application.%d{yyyy-MM-
dd}.log</fileNamePattern>
         <!-- keep 30 days worth of history -->
         <maxHistory>30</maxHistory>
      </rollingPolicy>
      <encoder>
         <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level
%logger{36} - %msg%n</pattern>
      </encoder>
   </appender>
   <!-- Log all messages at INFO level or above -->
   <root level="INFO">
      <appender-ref ref="FILE" />
   </root>
</configuration>
```

In the Controller Define the Logger Property

```
private final Logger LOG =
LoggerFactory.getLogger(DepartmentController.class);
```

Now use it in your methods like this

```
LOG.info("Save Department Initiated");
```

Lombok

In the pom.xml, add the dependencies for lombok

```
<dependency>
   <groupId>org.projectlombok</groupId>
   <artifactId>lombok</artifactId>
   <optional>true</optional>
</dependency>

<build>
   <plugins>
     <plugin>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-maven-plugin</artifactId>
       <configuration>
         <excludes>
           <exclude>
             <groupId>org.projectlombok</groupId>
             <artifactId>lombok</artifactId>
           </exclude>
         </excludes>
       </configuration>
     </plugin>
   </plugins>
</build>
```

Install the lombok plugin in the ide

Now in the entity, Clear all the getters, setters, constructors and toString. Add the lombok @Data annotation

```
package com.hydottech.Springboot.Tutorial.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.validation.constraints.NotBlank;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
```

```
@Builder
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long departmentId;
   @NotBlank(message="Name is required")
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
}
```

Exception Handling

Step 1.

Create an error package and add your custom DepartmentNotFound class

```
package com.hydottech.Springboot.Tutorial.error;
public class DepartmentNotFoundException extends Exception{
public DepartmentNotFoundException(String message){
   super(message);
}
   public DepartmentNotFoundException(String message,
Throwable cause){
      super(message,cause);
   }
   public DepartmentNotFoundException(Throwable cause){
      super(cause);
   }
}
```

Step 2.

In the DepartmentServiceImpl, modify the fetchOneDepartment method

```
@Override
public Department fetchOneDepartments(long id) throws
DepartmentNotFoundException {
Optional<Department> department =
departmentRepository.findById(id);
if(!department.isPresent()){
   throw new DepartmentNotFoundException("Department not
available");
}
return department.get();
}
```

In the Service and Controller, right click to *add the exception to method signature*

Step 3

In the entity package, create Error message class

```
package com.hydottech.Springboot.Tutorial.entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.http.HttpStatus;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorMessage {
   private HttpStatus status;
   private String message;
}
```

Step 4

In the error package, create `RestResponseEntityExceptionHandler class like this`

```
package com.hydottech.Springboot.Tutorial.error;
import
com.hydottech.Springboot.Tutorial.entity.ErrorMessage;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.ControllerAdvice;
import
org.springframework.web.bind.annotation.ExceptionHandler;
import
org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.Respo
nseEntityExceptionHandler;
@ControllerAdvice
@ResponseStatus
public class RestResponseEntityExceptionHandler extends
ResponseEntityExceptionHandler {
   @ExceptionHandler(DepartmentNotFoundException.class)
```

```
   public ResponseEntity<ErrorMessage>
departmentNotFoundException(DepartmentNotFoundException
exception, WebRequest request){
      ErrorMessage message = new
ErrorMessage(HttpStatus.NOT_FOUND, exception.getMessage());
      return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);
   }
}
```

MySQL

In the pom.xml add the mysql driver

```
<dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
```

In the application.properties, add the mysql configuration

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/
LearnSpring
spring.datasource.driver-class-
name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=HydotTech
spring.jpa.show-sql=true
```

Unit Testing (Service Layer)

From the DepartmentServiceImpl, right click to generate a test for fetchDepartmentByName. This will create a DepartmentServiceTest file in the test package of the com.hydottech.LearnSpring package

Add this configuration to it

```
package com.hydottech.Springboot.Tutorial.service;
import com.hydottech.Springboot.Tutorial.entity.Department;
import
com.hydottech.Springboot.Tutorial.repository.DepartmentRepo
sitory;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
```

```
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import static org.junit.jupiter.api.Assertions.*;
@SpringBootTest
class DepartmentServiceTest {
   @Autowired
   private DepartmentService departmentService;
   @MockBean
   private DepartmentRepository departmentRepository;
   @BeforeEach
   void setUp() {
      Department department = Department.builder().
                        departmentName("Kitchen1").
                        departmentAddress("Accra").
                        departmentCode("KT-123").
                        departmentId(1L).
                        build();

Mockito.when(departmentRepository.findByDepartmentNameIgnor
eCaseKofi("Kitchen1"))
             .thenReturn(department);
   }

   @Test
   @DisplayName("Get Department Name")
   public void whenDepartnameExist_thenDepartmentIsFound(){
      String departmentName = "Kitchen1";
      Department found =
departmentService.fetchDepartmentByName(departmentName);
      assertEquals(departmentName,
found.getDepartmentName());
   }
}
```

Properties Configuration

In the application.properties, add this configuration

```
welcome.message = Welcome to Hydot Tech;


@Value("${welcome.message}")
private String message;
@GetMapping("/")
public String helloWorld(){
```

```
    return message;
}
```

Adding application.yml

```
spring:
  application:
    name: Spring-Boot-Tutorial
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
  datasource:
    url: jdbc:mysql://localhost:3306/LearnSpring
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: HydotTech
server:
  port: 5000
welcome:
  message: Welcome to Hydot Tech;
```

Profiles

```
# Default Configuration
server:
  port: 5000
spring:
  profiles:
    active: dev
# Dev Profile
---
spring:
  config:
    activate:
      on-profile: dev
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
  datasource:
    url: jdbc:mysql://localhost:3306/LearnSpring
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: HydotTech
```

```
welcome:
  message: Welcome to Hydot Tech (Development);
# Test Profile
---
spring:
  config:
    activate:
      on-profile: test
  jpa:
    hibernate:
      ddl-auto: create-drop
    show-sql: true
  datasource:
    url: jdbc:mysql://localhost:3306/LearnSpringTest
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: TestPassword
welcome:
  message: Welcome to Hydot Tech (Testing);
# Prod Profile
---
spring:
  config:
    activate:
      on-profile: prod
  jpa:
    hibernate:
      ddl-auto: validate
    show-sql: false
  datasource:
    url: jdbc:mysql://prod-db-server:3306/LearnSpringProd
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: prod_user
    password: ProdSecurePassword
welcome:
  message: Welcome to Hydot Tech (Production);
```

Production

In the Pom.xml, change the version of the application

```
<version>1.0.0</version>
```

In the terminal, run this command [mvn clean install]

To run the application, run this [java -jar Spring-Boot-Tutorial-1.0.0.jar]

Actuator

Add the dependency in pom.xml

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the configuration in application.yml

```
# Default Configuration
server:
  port: 5000
spring:
  profiles:
    active: dev
# Dev Profile
---
spring:
  config:
    activate:
      on-profile: dev
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
  datasource:
    url: jdbc:mysql://localhost:3306/LearnSpring
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: HydotTech
welcome:
  message: Welcome to Hydot Tech (Development);
management:
  endpoints:
    web:
      exposure:
        include: "*"  # Include all endpoints
         exclude: "env, beans,mappings"

# Test Profile
---
spring:
  config:
    activate:
      on-profile: test
  jpa:
    hibernate:
```

```
    ddl-auto: create-drop
   show-sql: true
  datasource:
   url: jdbc:mysql://localhost:3306/LearnSpringTest
   driver-class-name: com.mysql.cj.jdbc.Driver
   username: root
   password: TestPassword
welcome:
  message: Welcome to Hydot Tech (Testing);
management:
  endpoints:
   web:
    exposure:
     include: "*"  # Include all endpoints
    cors:
     allowed-origins: "*"  # Allow CORS for all origins
     allowed-methods: "GET,POST,PUT,DELETE"
# Prod Profile
---
spring:
  config:
   activate:
    on-profile: prod
  jpa:
   hibernate:
    ddl-auto: validate
   show-sql: false
  datasource:
   url: jdbc:mysql://prod-db-server:3306/LearnSpringProd
   driver-class-name: com.mysql.cj.jdbc.Driver
   username: prod_user
   password: ProdSecurePassword
welcome:
  message: Welcome to Hydot Tech (Production);
management:
  endpoints:
   web:
    exposure:
     include: "*"  # Include all endpoints
    cors:
     allowed-origins: "*"  # Allow CORS for all origins
     allowed-methods: "GET,POST,PUT,DELETE"
```