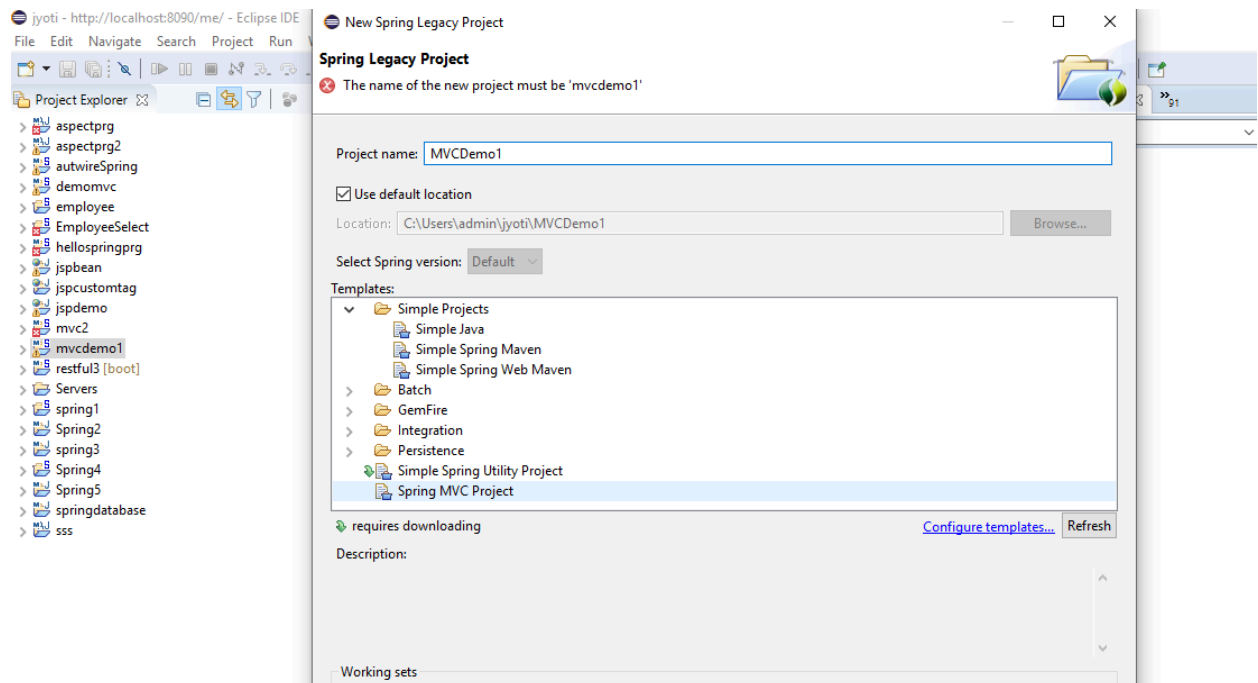
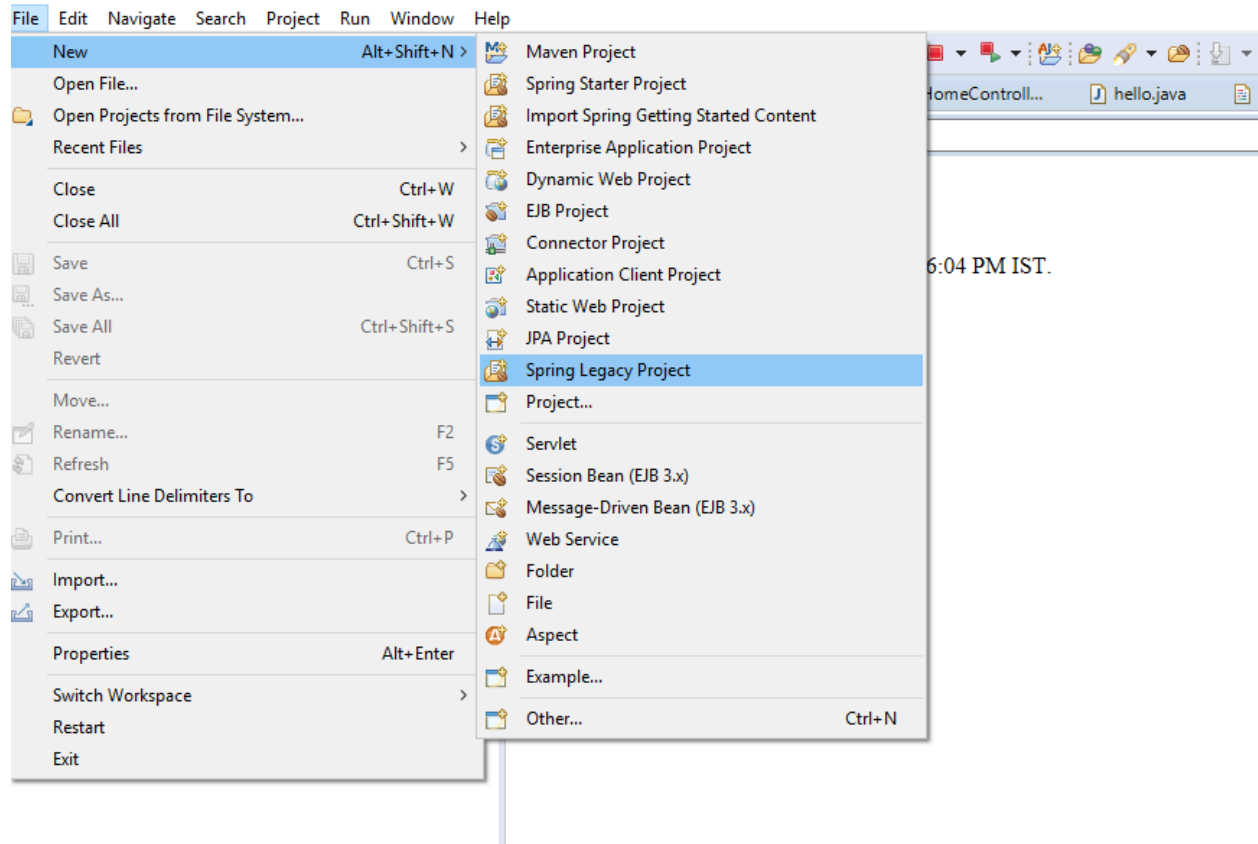


## Spring Boot MVC Application

ijyoti - http://localhost:8090/me/ - Eclipse IDE



jjyoti - demomvc/src/main/java/com/example/demo/HomeController.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



Project Explorer

- > aspectprg
- > aspectprg2
- > autwireSpring
- ▼ demomvc
  - > Deployment Descriptor: demomvc
  - > Spring Elements
  - > JAX-WS Web Services
  - ▼ Java Resources
    - ▼ src/main/java
      - ▼ com.example.demo
        - > HomeController.java
      - > src/main/resources
      - > src/test/java
      - > src/test/resources
      - > Libraries
    - > JavaScript Resources
    - > Deployed Resources
  - ▼ src
    - ▼ main
      - ▼ java
        - ▼ com
          - ▼ example
            - ▼ demo
              - ▼ HomeController.java
      - ▼ resources
        - ▼ META-INF
        - ▼ log4j.xml
      - ▼ webapp
        - ▼ resources
        - ▼ WEB-INF
          - ▼ classes
          - > spring
          - ▼ views
            - ▼ home.jsp

Home

web.xml

```
22- /**
23-  *
24-  */
25- @Re
26- pub
27-
28-
29-
30-
31-
32-
33-
34-
35-
36-
37- }
38-
39- }
```

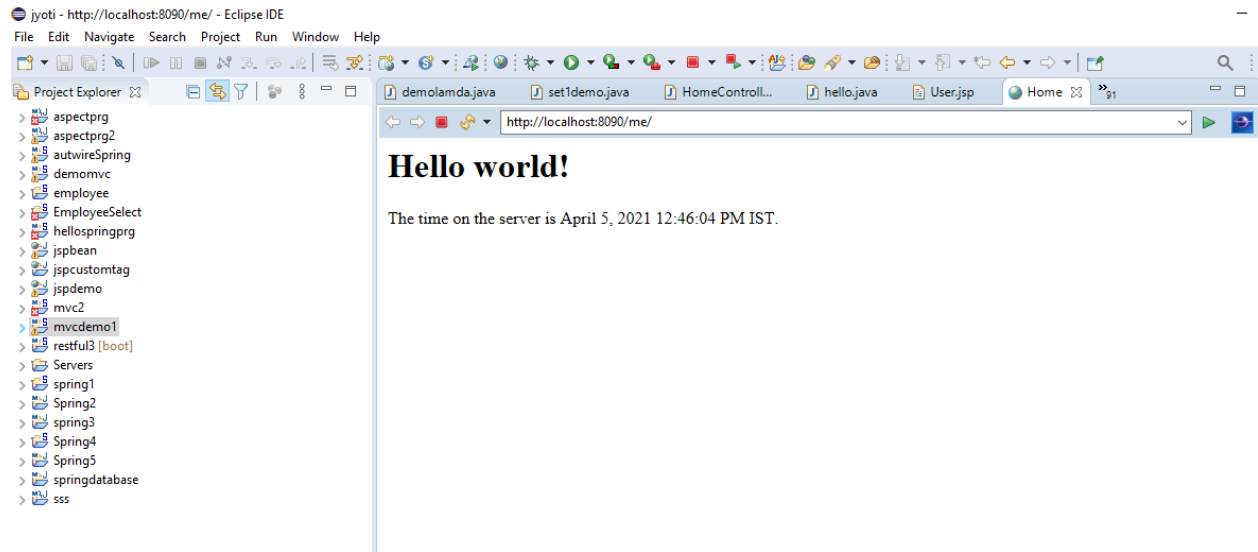
Markers

Properties

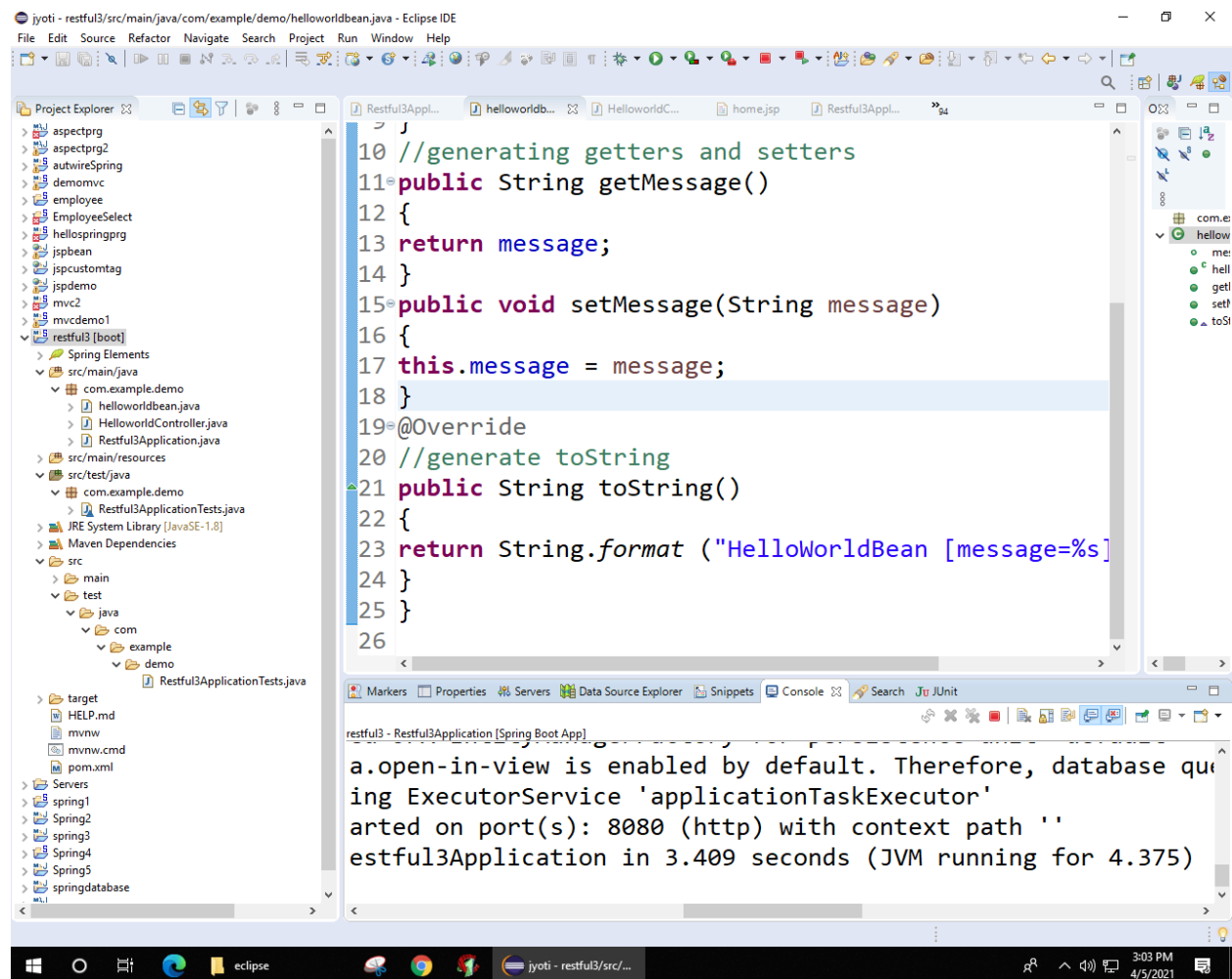
Component	Code
HomeContrller.java	<pre> package com.exampple.demo;  import java.text.DateFormat; import java.util.Date; import java.util.Locale;  import org.slf4j.Logger; import org.slf4j.LoggerFactory; import org.springframework.stereotype.Controller; import org.springframework.ui.Model; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RequestMethod; import org.springframework.validation.annotation.Validated;  /**  * Handles requests for the application home page.  */ @Controller public class HomeController {      private static final Logger logger = LoggerFactory.getLogger(HomeController.class);      /**      * Simply selects the home view to render by returning its name.      */     @RequestMapping(value = "/", method = RequestMethod.GET)     public String home(Locale locale, Model model) {         logger.info("Welcome home! The client locale is {}.\"", locale);          Date date = new Date(); </pre>

	<pre> DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);  String formattedDate = dateFormat.format(date);  model.addAttribute("serverTime", formattedDate );  return "home"; } </pre>
home.jsp	<pre> &lt;%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %&gt; &lt;%@ page session="false" %&gt; &lt;html&gt; &lt;head&gt;     &lt;title&gt;Home&lt;/title&gt; &lt;/head&gt; &lt;body&gt; &lt;h1&gt;     Hello world! &lt;/h1&gt;  &lt;P&gt; The time on the server is \${serverTime}. &lt;/P&gt; &lt;/body&gt; &lt;/html&gt; </pre>

Output



## Springboot RESTFull API



```
package com.example.demo;
public class helloworldbean
{
    public String message;
    //constructor of HelloWorldBean
    public helloworldbean(String message)
    {
        this.message=message;
    }
    //generating getters and setters
    public String getMessage()
    {
        return message;
    }
    public void setMessage(String message)
    {
        this.message = message;
    }
    @Override
    //generate toString
    public String toString()
    {
        return String.format ("HelloWorldBean
[message=%s]", message);
    }
}
```

HelloWorldController.java

```
package com.example.demo;

import
org.springframework.web.bind.annotation.Ge
tMapping;
import
org.springframework.web.bind.annotation.Re
stController;
//Controller
@RestController
public class HelloWorldController
{
    //using get method and hello-world as URI
    @GetMapping(path="/hello-world")
    public String helloWorld()
    {
        return "Hello World";
    }
    @GetMapping(path="/hello-world-bean")
    public helloworldbean helloWorldBean()
    {
        return new helloworldbean("Hello World");
    }
    //constructor of HelloWorldBean
}
```

Restful3Application.java

```
package com.example.demo;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class Restful3Application {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Restful3Application.class, args);
```

```
    }
```

```
}
```

Restful3ApplicationTests.java

```
package com.example.demo;
```

```
import org.junit.jupiter.api.Test;
```

```
import org.springframework.boot.test.context.SpringBootTest;
```

```
@SpringBootTest
```

```
class Restful3ApplicationTests {
```

```
    @Test
```



.s.d.r.c.RepositoryConfigurationDelegate :  
Bootstrapping Spring Data JPA repositories  
in DEFAULT mode.

2021-04-05 15:02:27.852 INFO 7280 --- [main]

.s.d.r.c.RepositoryConfigurationDelegate :  
Finished Spring Data repository scanning  
in 5 ms. Found 0 JPA repository  
interfaces.

2021-04-05 15:02:28.283 INFO 7280 --- [main]

o.s.b.w.embedded.tomcat.TomcatWebServer :  
Tomcat initialized with port(s): 8080  
(http)

2021-04-05 15:02:28.291 INFO 7280 --- [main]

o.apache.catalina.core.StandardService :  
Starting service [Tomcat]

2021-04-05 15:02:28.291 INFO 7280 --- [main]

org.apache.catalina.core.StandardEngine :  
Starting Servlet engine: [Apache  
Tomcat/9.0.41]

2021-04-05 15:02:28.378 INFO 7280 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/]

: Initializing Spring embedded  
WebApplicationContext

```
2021-04-05 15:02:28.378 INFO 7280 --- [
main]
w.s.c.ServletWebServerApplicationContext :
Root WebApplicationContext: initialization
completed in 1409 ms
2021-04-05 15:02:28.568 INFO 7280 --- [
main] com.zaxxer.hikari.HikariDataSource
: HikariPool-1 - Starting...
2021-04-05 15:02:28.700 INFO 7280 --- [
main] com.zaxxer.hikari.HikariDataSource
: HikariPool-1 - Start completed.
2021-04-05 15:02:28.752 INFO 7280 --- [
main]
o.hibernate.jpa.internal.util.LogHelper :
HHH000204: Processing PersistenceUnitInfo
[name: default]
2021-04-05 15:02:28.811 INFO 7280 --- [
main] org.hibernate.Version
: HHH000412: Hibernate ORM core version
5.4.27.Final
2021-04-05 15:02:28.989 INFO 7280 --- [
main]
o.hibernate.annotations.common.Version :
HCANN000001: Hibernate Commons Annotations
{5.1.2.Final}
2021-04-05 15:02:29.157 INFO 7280 --- [
main] org.hibernate.dialect.Dialect
```

```
: HHH000400: Using dialect:
org.hibernate.dialect.H2Dialect
2021-04-05 15:02:29.319 INFO 7280 --- [
main] o.h.e.t.j.p.i.JtaPlatformInitiator
: HHH000490: Using JtaPlatform
implementation:
[org.hibernate.engine.transaction.jta.plat
form.internal.NoJtaPlatform]
2021-04-05 15:02:29.332 INFO 7280 --- [
main]
j.LocalContainerEntityManagerFactoryBean :
Initialized JPA EntityManagerFactory for
persistence unit 'default'
2021-04-05 15:02:29.423 WARN 7280 --- [
main]
JpaBaseConfiguration$JpaWebConfiguration :
spring.jpa.open-in-view is enabled by
default. Therefore, database queries may
be performed during view rendering.
Explicitly configure
spring.jpa.open-in-view to disable this
warning
2021-04-05 15:02:29.574 INFO 7280 --- [
main]
o.s.s.concurrent.ThreadPoolTaskExecutor :
Initializing ExecutorService
'applicationTaskExecutor'
```

```
2021-04-05 15:02:29.802 INFO 7280 --- [
main]
o.s.b.w.embedded.tomcat.TomcatWebServer :
Tomcat started on port(s): 8080 (http)
with context path ''
2021-04-05 15:02:29.810 INFO 7280 --- [
main] com.example.demo.Restful3Application
: Started Restful3Application in 3.409
seconds (JVM running for 4.375)
```

we will see the steps to set up a Spring Boot application with PostgreSQL. We will have a simple CRUD operation in Postgres Database by exposing the application via Rest API. We will use POSTMAN to test the application.

## Setting up Postgres Server

- Download the Postgres server from the link:  
<https://www.postgresql.org/download/>
- Run the installer. It will also ask the password for the superuser: *postgres*
- Click on pgAdmin4.exe located inside the PostgreSQL folder inside Program Files.

## Setting up Spring Boot Application

### Prerequisite:

Have JDK 1.8 installed

- Download a sample Spring Boot project from <https://start.spring.io/>
- Update the pom.xml as below:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
6     http://maven.apache.org/xsd/maven-4.0.0.xsd">
7
8   <modelVersion>4.0.0</modelVersion>
9
10  <parent>
11
12    <groupId>org.springframework.boot</groupId>
13
14    <artifactId>spring-boot-starter-parent</artifactId>
```

	8
<version>2.1.1.RELEASE</version>	9
<relativePath /> <!-- lookup parent from repository -->	10
</parent>	11
<groupId>com.sample</groupId>	12
<artifactId>postgress</artifactId>	13
<version>0.0.1-SNAPSHOT</version>	14
<name>postgress</name>	15
<description>Demo project for Spring Boot</description>	16
	17
<properties>	18
<java.version>1.8</java.version>	19
</properties>	20
	21
<dependencies>	22
<dependency>	23
<groupId>org.springframework.boot</groupId>	24
<artifactId>spring-boot-starter</artifactId>	25
</dependency>	26
<dependency>	27
<groupId>org.springframework.boot</groupId>	28
<artifactId>spring-boot-starter-web</artifactId>	29
</dependency>	30
<dependency>	31
<groupId>org.springframework.boot</groupId>	32
<artifactId>spring-boot-starter-jdbc</artifactId>	33
</dependency>	34
<dependency>	35
<groupId>org.postgresql</groupId>	36
<artifactId>postgresql</artifactId>	37

<code>&lt;scope&gt;runtime&lt;/scope&gt;</code>	38
<code>&lt;/dependency&gt;</code>	39
<code>&lt;dependency&gt;</code>	40
<code>&lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;</code>	41
<code>&lt;artifactId&gt;spring-boot-starter-test&lt;/artifactId&gt;</code>	42
<code>&lt;scope&gt;test&lt;/scope&gt;</code>	43
<code>&lt;/dependency&gt;</code>	44
<code>&lt;/dependencies&gt;</code>	45
	46
<code>&lt;build&gt;</code>	47
<code>&lt;plugins&gt;</code>	48
<code>&lt;plugin&gt;</code>	49
<code>&lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;</code>	50
<code>&lt;artifactId&gt;spring-boot-maven-plugin&lt;/artifactId&gt;</code>	51
<code>&lt;/plugin&gt;</code>	52
<code>&lt;/plugins&gt;</code>	53
<code>&lt;/build&gt;</code>	54
	55
<code>&lt;/project&gt;</code>	

`spring-boot-starter-jdbc` artifact will give all the spring jdbc related jars

`org.postgresql.postgresql` will have the dependency of postgres jdbc driver in runtime.

- Create a *schema.sql* in resource folder. An employee table will be created in server startup. This can be ignored if you don't want the initial database to be configured during server start. Generally, for building a production-ready application, this step can be ignored as tables will be created with scrip directly in the DB.

<code>CREATE TABLE employee</code>	1
<code>(</code>	2
<code>employeeName varchar(100) NOT NULL,</code>	3
	4

employeeId varchar(11) NOT NULL ,	5
employeeAddress varchar(100) DEFAULT NULL,	6
employeeEmail varchar(100) DEFAULT NULL,	7
PRIMARY KEY (employeeId)	8
);	

- Create *data.sql* in resource folder for loading the first set of employee during startup. Can be skipped otherwise:

```
insert into employee(employeeId, employeeName ,
employeeAddress,employeeEmail) values('1','Jack','USA','jack@gmail.com');
```

- Changes in application.properties to configure the data source with URL, username, and password of the Postgres DB. 5432 is the default port of Postgres. Hibernate will automatically pick up the postgresSQLDialect.

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect	1
spring.jpa.hibernate.ddl-auto=none	2
spring.jpa.hibernate.show-sql=true	3
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres	4
spring.datasource.username=postgres	5
spring.datasource.password=admin	6
	7
	8
	9
spring.datasource.initialization-mode=always	10
spring.datasource.initialize=true	11
spring.datasource.schema=classpath:/schema.sql	12
spring.datasource.continue-on-error=true	

**spring.jpa.hibernate.ddl-auto** will turn off the hibernate auto-creation of the tables from the entity objects. Generally, Hibernate runs it if there is an Entity defined. But we will be using a native SQL query with JdbcTemplate, hence, we can turn this off as we will not be creating an Entity.

**spring.datasource.initialization-mode** is marked as always as we want initialization of the database to happen on every startup. This is optional and made for this sample purpose.

`spring.datasource.initialize=true` will mark the initialization to be true.

`spring.datasource.continue-on-error=true` will continue application startup in spite of any errors in data initialization.

`spring.datasource.schema` is the schema path that needs to be initialized.

`spring.datasource.url` URL of the Postgres DB. It can be a remote DB as well.

`spring.datasource.username` username for the database.

`spring.datasource.password` password for the database.

- Create a dao interface and dao implementation.

```
1 package com.sample.postgress.dao;
2
3
4 import java.util.List;
5
6 import com.sample.postgress.entity.Employee;
7
8 public interface EmployeeDao {
9
10     List<Employee> findAll();
11
12     void insertEmployee(Employee emp);
13
14     void updateEmployee(Employee emp);
15
16     void executeUpdateEmployee(Employee emp);
17
18     public void deleteEmployee(Employee emp);
19 }
20
21 package com.sample.postgress.dao;
22
```

	3
import java.sql.PreparedStatement;	4
import java.sql.SQLException;	5
import java.util.HashMap;	6
import java.util.List;	7
import java.util.Map;	8
	9
import org.springframework.dao.DataAccessException;	10
import org.springframework.jdbc.core.PreparedStatementCallback;	11
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;	12
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;	13
import org.springframework.jdbc.core.namedparam.SqlParameterSource;	14
import org.springframework.jdbc.support.GeneratedKeyHolder;	15
import org.springframework.jdbc.support.KeyHolder;	16
import org.springframework.stereotype.Repository;	17
	18
import com.sample.postgress.entity.Employee;	19
import com.sample.postgress.mapper.EmployeeRowMapper;	20
@Repository	21
public class EmployeeDaoImpl implements EmployeeDao{	22
	23
public EmployeeDaoImpl(NamedParameterJdbcTemplate template) {	24
this.template = template;	25
}	26
NamedParameterJdbcTemplate template;	27
	28
@Override	29
public List<Employee> findAll() {	30
return template.query("select * from employee", new EmployeeRowMapper());	31
}	32

@Override	
	33
public void insertEmployee(Employee emp) {	
	34
final String sql = "insert into employee(employeeId, employeeName , employeeAddress,employeeEmail) values(:employeeId,:employeeName,:employeeEmail,:employeeAddress)";	
	35
	36
KeyHolder holder = new GeneratedKeyHolder();	
	37
SqlParameterSource param = new MapSqlParameterSource()	
	38
.addValue("employeeId", emp.getEmployeeId())	
	39
.addValue("employeeName", emp.getEmployeeName())	
	40
.addValue("employeeEmail", emp.getEmployeeEmail())	
	41
.addValue("employeeAddress", emp.getEmployeeAddress());	
	42
template.update(sql,param, holder);	
	43
	44
}	
	45
	46
@Override	
	47
public void updateEmployee(Employee emp) {	
	48
final String sql = "update employee set employeeName=:employeeName, employeeAddress=:employeeAddress, employeeEmail=:employeeEmail where employeeId=:employeeId";	
	49
	50
KeyHolder holder = new GeneratedKeyHolder();	
	51
SqlParameterSource param = new MapSqlParameterSource()	
	52
.addValue("employeeId", emp.getEmployeeId())	
	53
.addValue("employeeName", emp.getEmployeeName())	
	54
.addValue("employeeEmail", emp.getEmployeeEmail())	
	55
.addValue("employeeAddress", emp.getEmployeeAddress());	
	56
template.update(sql,param, holder);	
	57
	58
}	
	59

@Override	60
	61
public void executeUpdateEmployee(Employee emp) {	62
final String sql = "update employee set employeeName=:employeeName, employeeAddress=:employeeAddress, employeeEmail=:employeeEmail where employeeId=:employeeId";	63
	64
	65
Map<String,Object> map=new HashMap<String,Object>();	66
map.put("employeeId", emp.getEmployeeId());	67
map.put("employeeName", emp.getEmployeeName());	68
map.put("employeeEmail", emp.getEmployeeEmail());	69
map.put("employeeAddress", emp.getEmployeeAddress());	70
	71
template.execute(sql,map,new PreparedStatementCallback<Object>() {	72
@Override	73
public Object doInPreparedStatement(PreparedStatement ps)	74
throws SQLException, DataAccessException {	75
return ps.executeUpdate();	76
}	77
});	78
	79
	80
}	81
	82
@Override	83
public void deleteEmployee(Employee emp) {	84
final String sql = "delete from employee where employeeId=:employeeId";	85
	86
	87
Map<String,Object> map=new HashMap<String,Object>();	88

map.put("employeeId", emp.getEmployeeId());	89
	90
template.execute(sql,map,new PreparedStatementCallback<Object>() {	91
@Override	92
public Object doInPreparedStatement(PreparedStatement ps)	93
throws SQLException, DataAccessException {	94
return ps.executeUpdate();	95
}	96
});	97
	98
	99
}	100
	101
}	

- findAll() retrieves all the employee and then map the resultset to a Employee Object using RowMapper described below .
- insertEmployee() will insert an employee using **template.update(sql,param,holder)** where param is the SqlParameterSource, which will map the values dynamically in the query marked with a colon. GeneratedKeyHolder will return an auto-generated value when data is inserted.
- executeUpdateEmployee() will update the employee using **template.execute**

	1
template.execute(sql,map,new PreparedStatementCallback<Object>() {	2
@Override	3
public Object doInPreparedStatement(PreparedStatement ps)	4
throws SQLException, DataAccessException {	5
return ps.executeUpdate();	6
}	7
});	

- EmployeeRowMapper to map the result set retrieved from the *select* query with the POJO.

package com.sample.postgress.mapper;	1
	2
	3
import java.sql.ResultSet;	4
import java.sql.SQLException;	5
	6
import org.springframework.jdbc.core.RowMapper;	7
	8
import com.sample.postgress.entity.Employee;	9
	10
public class EmployeeRowMapper implements RowMapper<Employee> {	11
	12
@Override	13
public Employee mapRow(ResultSet rs, int arg1) throws SQLException {	14
Employee emp = new Employee();	15
emp.setEmployeeId(rs.getString("employeeId"));	16
emp.setEmployeeName(rs.getString("employeeName"));	17
emp.setEmployeeEmail(rs.getString("employeeEmail"));	18
	19
return emp;	20
}	21
	22
	23
}	

- You can create a controller and a service class as follows:

package com.sample.postgress.controller;	1
	2
	3
import java.util.List;	4

import javax.annotation.Resource;	5
	6
	7
import org.springframework.web.bind.annotation.DeleteMapping;	8
import org.springframework.web.bind.annotation.GetMapping;	9
import org.springframework.web.bind.annotation.PostMapping;	10
import org.springframework.web.bind.annotation.PutMapping;	11
import org.springframework.web.bind.annotation.RequestBody;	12
import org.springframework.web.bind.annotation.RequestMapping;	13
import org.springframework.web.bind.annotation.RestController;	14
	15
import com.sample.postgress.entity.Employee;	16
import com.sample.postgress.service.EmployeeService;	17
	18
@RestController	19
@RequestMapping("/postgressApp")	20
public class ApplicationController {	21
	22
@Resource	23
EmployeeService employeeService;	24
	25
@GetMapping(value = "/employeeList")	26
public List<Employee> getEmployees() {	27
return employeeService.findAll();	28
	29
}	30
	31
@PostMapping(value = "/createEmp")	32
public void createEmployee(@RequestBody Employee emp) {	33
employeeService.insertEmployee(emp);	34

	35
}	36
@PutMapping(value = "/updateEmp")	37
public void updateEmployee(@RequestBody Employee emp) {	38
employeeService.updateEmployee(emp);	39
	40
}	41
@PutMapping(value = "/executeUpdateEmp")	42
public void executeUpdateEmployee(@RequestBody Employee emp) {	43
employeeService.executeUpdateEmployee(emp);	44
	45
}	46
	47
@DeleteMapping(value = "/deleteEmpById")	48
public void deleteEmployee(@RequestBody Employee emp) {	49
employeeService.deleteEmployee(emp);	50
	51
}	52
	53
	54
}	

Now, let's use POSTMAN to validate the changes:

Test 1: Get the list of employees

<http://localhost:8080/postgressApp/employeeList>

GET

http://localhost:8080/postgressApp/employeeList

Params

Send

Authorization

Headers

Body

Pre-request Script

Tests

Type

No Auth

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

Pretty

Raw

Preview

JSON

1

2

3

4

5

6

7

8

[

{

"employeeId": "1",

"employeeName": "Jack",

"employeeEmail": "jack@gmail.com",

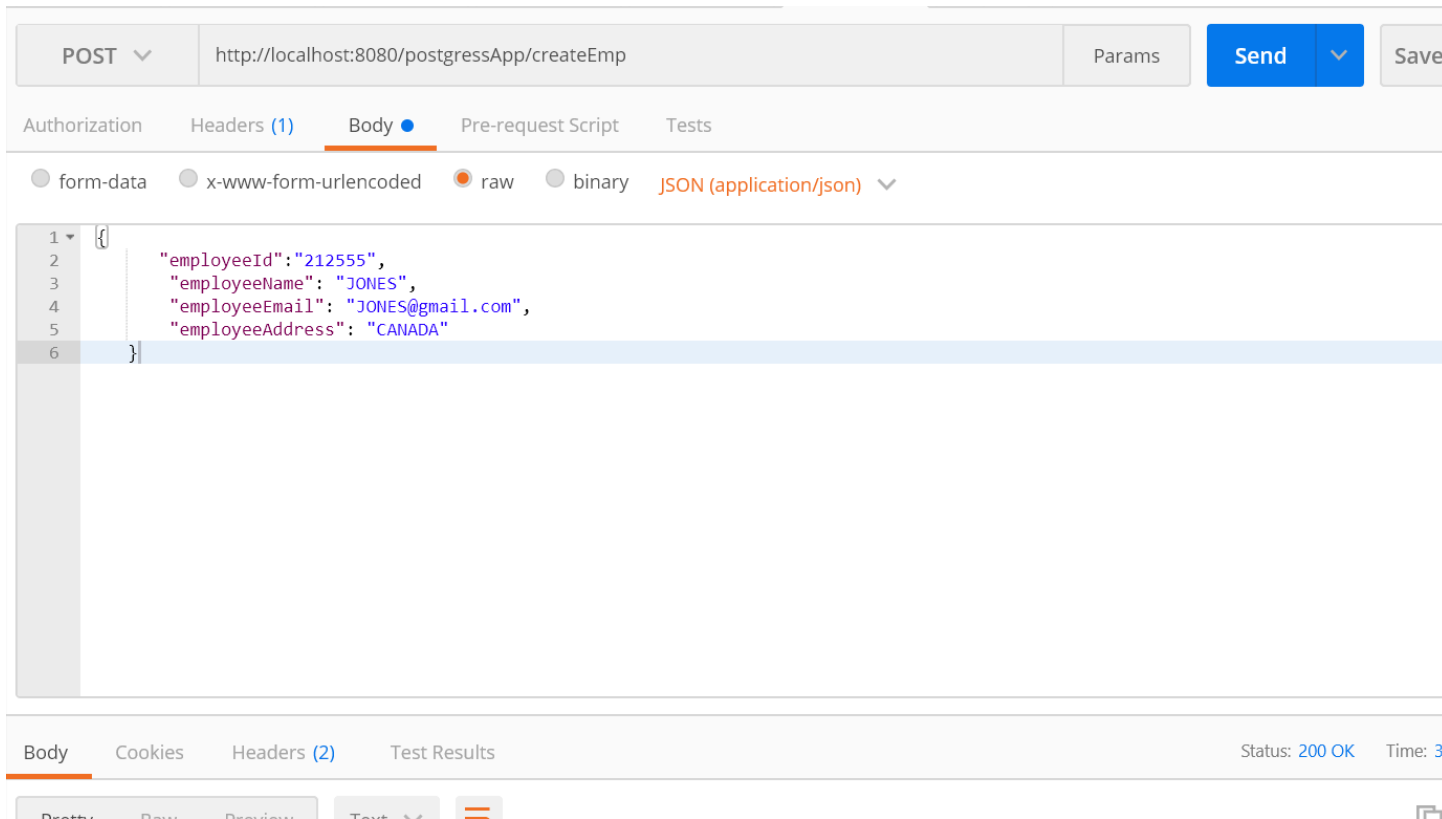
"employeeAddress": null

}

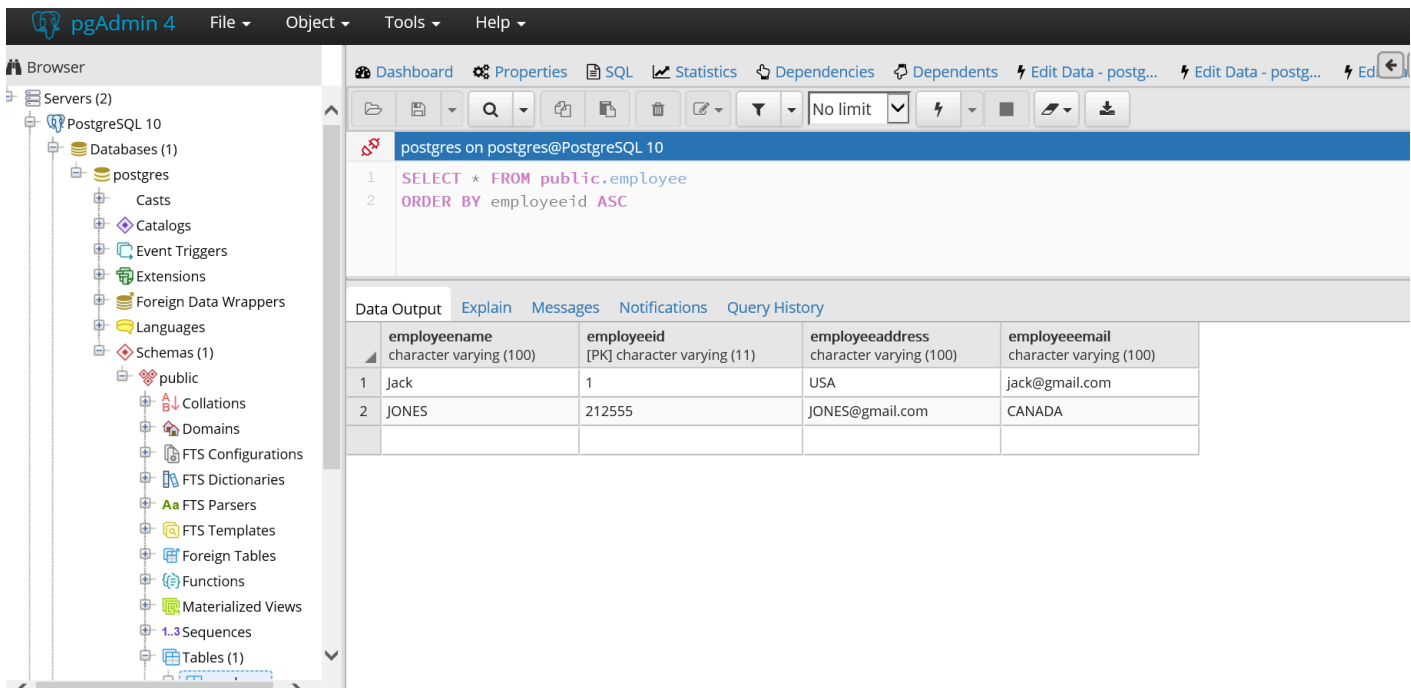
]

## Test 2: Create An employee

http://localhost:8080/postgressApp/createEmp



We see an entry got inserted with JONES.



Test 3: Update an Employee

http://localhost:8080/postgressApp/executeUpdateEmp

PUT

http://localhost:8080/postgressApp/executeUpdateEmp

Params

Send

Save

Authorization

Headers (1)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

1

2

3

4

5

6

{

"employeeId": "212555",

"employeeName": "jones",

"employeeEmail": "jones@gmail.com",

"employeeAddress": "USA"

}

Data Output				
Explain				
Messages				
Notifications				
Query History				
	employeename character varying (100)	employeeid [PK] character varying (11)	employeeaddress character varying (100)	employeeemail character varying (100)
1	Jack	1	USA	jack@gmail.com
2	jones	212555	USA	jones@gmail.com

## Test 4: Delete Employee

http://localhost:8080/postgressApp/deleteEmpById

DELETE ▼ http://localhost:8080/postgressApp/deleteEmpById Params Send ▼

Authorization Headers (1) **Body** ● Pre-request Script Tests

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) ▼

```
1 {  
2   "employeeId": "212555"  
3 }  
4
```

Body Cookies Headers (2) Test Results Status: 200 OK

Pretty Raw Preview Text ↺

Dashboard Properties SQL Statistics Dependencies Dependents Edit Data - postg... Edit

postgres on postgres@PostgreSQL 10

```
1 SELECT * FROM public.employee  
2 ORDER BY employeeid ASC
```

Data Output Explain Messages Notifications Query History

	employeename character varying (100)	employeeid [PK] character varying (11)	employeeaddress character varying (100)	employeeemail character varying (100)
1	Jack	1	USA	jack@gmail.com

## Conclusion

We have learned how to set up a Spring Boot application with Postgres and how to do a CRUD operation.