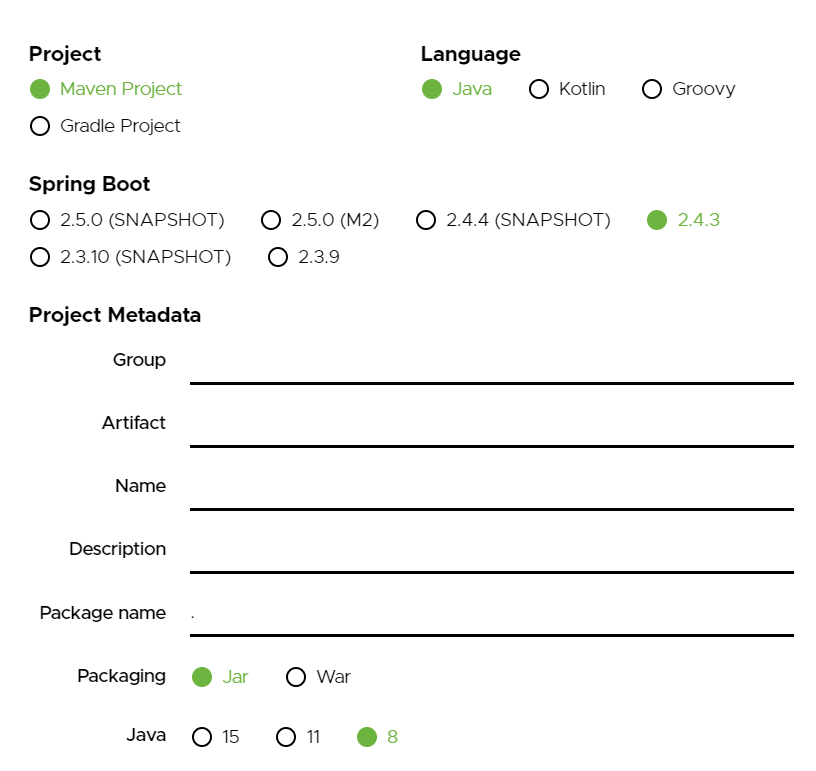
# **Hands-On Exercise 05: Creating Spring Boot with REST API**

## **Let’s get Started**

## **Starting with Spring Initializr**

For all Spring applications, you should start with the [Spring Initializr](https://start.spring.io/). The Initializr offers a fast way to pull in all the dependencies you need for an application and does a lot of the set up for you. This example needs the Spring Web and Spring Boot Actuator dependencies.



As shown in the image above, following steps have to be done

* Launch Spring Initializr and choose the following
  + Choose com.example as Group
  + Choose demo as Artifact
  + Choose following dependencies
    - Web
    - Actuator
    - DevTools
* Click Generate Project.
* Import the project into Eclipse/IntelliJ.
* If you want to understand all the files that are part of this project, you can go here.

## **1. HttpClient Configuration**

In HttpClientConfig class, we are configuring mainly two things –

1. [PoolingHttpClientConnectionManager](https://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/impl/conn/PoolingHttpClientConnectionManager.html) – As name suggests, its connection pool manager. Here, connections are pooled on a per route basis. A request for a route which already the manager has persistent connections for available in the pool will be services by leasing a connection from the pool rather than creating a brand new connection.  
   [ConnectionKeepAliveStrategy](https://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/conn/ConnectionKeepAliveStrategy.html) helps in setting time which decides how long a connection can remain idle before being reused.
2. And set a idleConnectionMonitor thread, which periodically checks all connections and free up which have not been used and idle time has elapsed.

The real http client to use is [CloseableHttpClient](https://hc.apache.org/httpcomponents-client-ga/httpclient/apidocs/org/apache/http/impl/client/CloseableHttpClient.html) bean. It is what RestTemplate will use to get the connection to API endpoints. Create ***HttpClientConfig.java*** under ***com/example.demo.springexamples/config*** :

|  |
| --- |
| package com.example.springexamples.config;  import java.security.KeyManagementException;  import java.security.KeyStoreException;  import java.security.NoSuchAlgorithmException;  import java.util.concurrent.TimeUnit;  import org.apache.http.HeaderElement;  import org.apache.http.HeaderElementIterator;  import org.apache.http.HttpResponse;  import org.apache.http.client.config.RequestConfig;  import org.apache.http.config.Registry;  import org.apache.http.config.RegistryBuilder;  import org.apache.http.conn.ConnectionKeepAliveStrategy;  import org.apache.http.conn.socket.ConnectionSocketFactory;  import org.apache.http.conn.socket.PlainConnectionSocketFactory;  import org.apache.http.conn.ssl.SSLConnectionSocketFactory;  import org.apache.http.conn.ssl.TrustSelfSignedStrategy;  import org.apache.http.impl.client.CloseableHttpClient;  import org.apache.http.impl.client.HttpClients;  import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;  import org.apache.http.message.BasicHeaderElementIterator;  import org.apache.http.protocol.HTTP;  import org.apache.http.protocol.HttpContext;  import org.apache.http.ssl.SSLContextBuilder;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.scheduling.annotation.EnableScheduling;  import org.springframework.scheduling.annotation.Scheduled;  @Configuration  @EnableScheduling  public class HttpClientConfig {  private static final Logger *LOGGER* = LoggerFactory.*getLogger*(HttpClientConfig.class);  // Determines the timeout in milliseconds until a connection is established.  private static final int *CONNECT\_TIMEOUT* = 30000;    // The timeout when requesting a connection from the connection manager.  private static final int *REQUEST\_TIMEOUT* = 30000;    // The timeout for waiting for data  private static final int *SOCKET\_TIMEOUT* = 60000;  private static final int *MAX\_TOTAL\_CONNECTIONS* = 50;  private static final int *DEFAULT\_KEEP\_ALIVE\_TIME\_MILLIS* = 20 \* 1000;  private static final int *CLOSE\_IDLE\_CONNECTION\_WAIT\_TIME\_SECS* = 30;  @Bean  public PoolingHttpClientConnectionManager poolingConnectionManager() {  SSLContextBuilder builder = new SSLContextBuilder();  try {  builder.loadTrustMaterial(null, new TrustSelfSignedStrategy());  } catch (NoSuchAlgorithmException | KeyStoreException e) {  *LOGGER*.error("Pooling Connection Manager Initialisation failure because of " + e.getMessage(), e);  }  SSLConnectionSocketFactory sslsf = null;  try {  sslsf = new SSLConnectionSocketFactory(builder.build());  } catch (KeyManagementException | NoSuchAlgorithmException e) {  *LOGGER*.error("Pooling Connection Manager Initialisation failure because of " + e.getMessage(), e);  }  Registry<ConnectionSocketFactory> socketFactoryRegistry = RegistryBuilder  .<ConnectionSocketFactory>*create*().register("https", sslsf)  .register("http", new PlainConnectionSocketFactory())  .build();  PoolingHttpClientConnectionManager poolingConnectionManager = new PoolingHttpClientConnectionManager(socketFactoryRegistry);  poolingConnectionManager.setMaxTotal(*MAX\_TOTAL\_CONNECTIONS*);  return poolingConnectionManager;  }  @Bean  public ConnectionKeepAliveStrategy connectionKeepAliveStrategy() {  return new ConnectionKeepAliveStrategy() {  @Override  public long getKeepAliveDuration(HttpResponse response, HttpContext context) {  HeaderElementIterator it = new BasicHeaderElementIterator  (response.headerIterator(HTTP.*CONN\_KEEP\_ALIVE*));  while (it.hasNext()) {  HeaderElement he = it.nextElement();  String param = he.getName();  String value = he.getValue();  if (value != null && param.equalsIgnoreCase("timeout")) {  return Long.*parseLong*(value) \* 1000;  }  }  return *DEFAULT\_KEEP\_ALIVE\_TIME\_MILLIS*;  }  };  }  @Bean  public CloseableHttpClient httpClient() {  RequestConfig requestConfig = RequestConfig.*custom*()  .setConnectionRequestTimeout(*REQUEST\_TIMEOUT*)  .setConnectTimeout(*CONNECT\_TIMEOUT*)  .setSocketTimeout(*SOCKET\_TIMEOUT*).build();  return HttpClients.*custom*()  .setDefaultRequestConfig(requestConfig)  .setConnectionManager(poolingConnectionManager())  .setKeepAliveStrategy(connectionKeepAliveStrategy())  .build();  }    @Bean  public Runnable idleConnectionMonitor(final PoolingHttpClientConnectionManager connectionManager) {  return new Runnable() {  @Override  @Scheduled(fixedDelay = 10000)  public void run() {  try {  if (connectionManager != null) {  *LOGGER*.trace("run IdleConnectionMonitor - Closing expired and idle connections...");  connectionManager.closeExpiredConnections();  connectionManager.closeIdleConnections(*CLOSE\_IDLE\_CONNECTION\_WAIT\_TIME\_SECS*, TimeUnit.*SECONDS*);  } else {  *LOGGER*.trace("run IdleConnectionMonitor - Http Client Connection manager is not initialised");  }  } catch (Exception e) {  *LOGGER*.error("run IdleConnectionMonitor - Exception occurred. msg={}, e={}", e.getMessage(), e);  }  }  };  }  } |

## **2. Spring RestTemplate Configuration**

Here we are configuring RestTemplate bean which we will finally use to invoke REST APIs. As mentioned above, it uses CloseableHttpClient bean instance to build [ClientHttpRequestFactory](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/client/ClientHttpRequestFactory.html), which is used to create RestTemplate.

1. [HttpComponentsClientHttpRequestFactory](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/client/HttpComponentsClientHttpRequestFactory.html) is ClientHttpRequestFactory implementation that uses *Apache HttpComponents HttpClient* to create requests.
2. We have used @Scheduled annotation in httpClient configuration. To support this, we have to add support of scheduled execution of thread. For that, we have used bean ThreadPoolTaskScheduler which internally utilizes [ScheduledThreadPoolExecutor](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html) to schedule commands to run after a given delay, or to execute periodically.

Create ***RestTemplateConfig.java*** under ***com/example.demo.springexamples/config*** :

|  |
| --- |
| package com.example.springexamples.config;  import org.apache.http.impl.client.CloseableHttpClient;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.http.client.HttpComponentsClientHttpRequestFactory;  import org.springframework.scheduling.TaskScheduler;  import org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler;  import org.springframework.web.client.RestTemplate;  public class RestTemplateConfig {  @Autowired  CloseableHttpClient httpClient;  @Bean  public RestTemplate restTemplate() {  RestTemplate restTemplate = new RestTemplate(clientHttpRequestFactory());  return restTemplate;  }  @Bean  public HttpComponentsClientHttpRequestFactory clientHttpRequestFactory() {  HttpComponentsClientHttpRequestFactory clientHttpRequestFactory = new HttpComponentsClientHttpRequestFactory();  clientHttpRequestFactory.setHttpClient(httpClient);  return clientHttpRequestFactory;  }  @Bean  public TaskScheduler taskScheduler() {  ThreadPoolTaskScheduler scheduler = new ThreadPoolTaskScheduler();  scheduler.setThreadNamePrefix("poolScheduler");  scheduler.setPoolSize(50);  return scheduler;  }  } |

## Now let’s create files named as ***EmployeeListVO.java*** and ***EmployeeVO.java*** under ***com/example.demo.springexamples/model*** :

## ***EmployeeListVO.java:***

|  |
| --- |
| package com.example.springexamples.model;  import java.util.ArrayList;  import java.util.List;  import javax.xml.bind.annotation.XmlRootElement;  @XmlRootElement (name="employees")  public class EmployeeListVO  {  private List<EmployeeVO> employees = new ArrayList<EmployeeVO>();  public List<EmployeeVO> getEmployees() {  return employees;  }  public void setEmployees(List<EmployeeVO> employees) {  this.employees = employees;  }  } |

## 

## ***EmployeeVO.java:***

|  |
| --- |
| package com.example.springexamples.model;  import java.io.Serializable;  import javax.xml.bind.annotation.XmlAccessType;  import javax.xml.bind.annotation.XmlAccessorType;  import javax.xml.bind.annotation.XmlAttribute;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;  @XmlRootElement (name = "employee")  @XmlAccessorType(XmlAccessType.*NONE*)  public class EmployeeVO implements Serializable  {  private static final long *serialVersionUID* = 1L;  @XmlAttribute  private Integer id;    @XmlElement  private String firstName;    @XmlElement  private String lastName;    @XmlElement  private String email;    public EmployeeVO(Integer id, String firstName, String lastName, String email) {  super();  this.id = id;  this.firstName = firstName;  this.lastName = lastName;  this.email = email;  }  public EmployeeVO(){  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getFirstName() {  return firstName;  }  public void setFirstName(String firstName) {  this.firstName = firstName;  }  public String getLastName() {  return lastName;  }  public void setLastName(String lastName) {  this.lastName = lastName;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  @Override  public String toString() {  return "EmployeeVO [id=" + id + ", firstName=" + firstName  + ", lastName=" + lastName + ", email=" + email + "]";  }  } |

## After this create service named as ***UserService.java*** under ***com/example.demo.springexamples/service*** :

|  |
| --- |
| package com.example.springexamples.service;  public interface UserService {    public String testUserService();  } |

## After this create impl as ***UserServiceImpl.java*** under ***com/example.demo.springexamples/service/impl***:

|  |
| --- |
| package com.example.springexamples.service.impl;  import com.example.demo.springexamples.service.UserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.http.HttpEntity;  import org.springframework.http.HttpHeaders;  import org.springframework.stereotype.Service;  import org.springframework.web.client.RestTemplate;  @Service  public class UserServiceImpl implements UserService {  @Autowired  RestTemplate restTemplate;    @Override  public String testUserService()  {  final String uri = "http://localhost:8080/users";    HttpHeaders headers = new HttpHeaders();  headers.set("Header", "value");  headers.set("Other-Header", "othervalue");    HttpEntity<String> entity = new HttpEntity<String>(headers);    String result = restTemplate.getForObject(uri, String.class);  return result;  }  } |

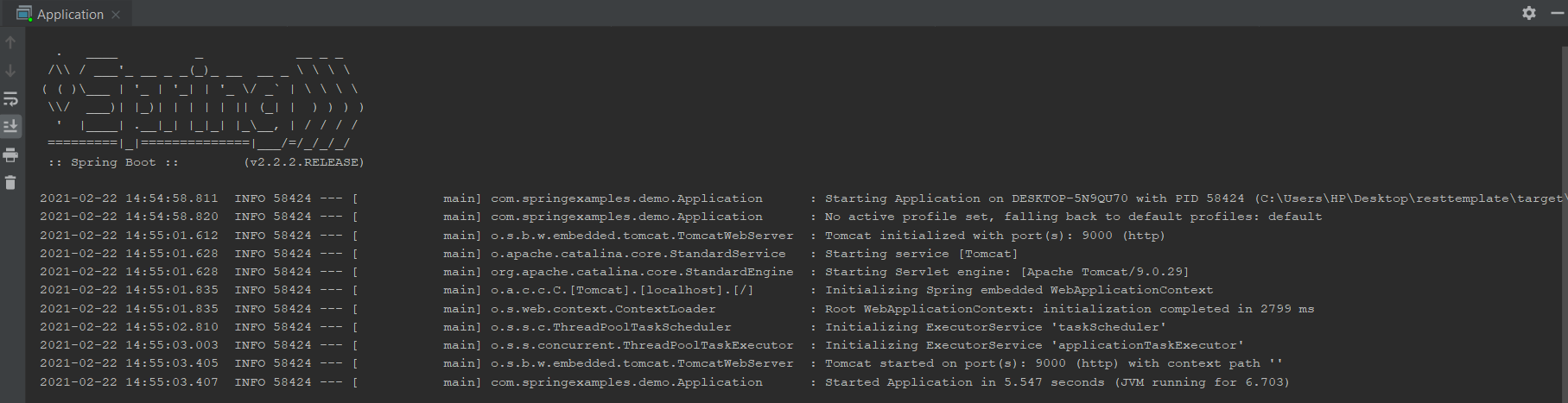
## Now let’s create web controller named as ***EmployeeController.java*** under ***com/example.demo.springexamples/web*** :

|  |
| --- |
| package com.example.springexamples.web;  import java.net.URI;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.http.HttpStatus;  import org.springframework.http.MediaType;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.DeleteMapping;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.PutMapping;  import org.springframework.web.bind.annotation.RequestHeader;  import org.springframework.web.bind.annotation.RestController;  import org.springframework.web.servlet.support.ServletUriComponentsBuilder;  import com.example.demo.springexamples.model.EmployeeListVO;  import com.example.demo.springexamples.model.EmployeeVO;  @RestController  public class EmployeeController  {  private static Logger *LOGGER* = LoggerFactory.*getLogger*(EmployeeController.class);    @GetMapping(value = "/employees",  produces = {MediaType.*APPLICATION\_XML\_VALUE*, MediaType.*APPLICATION\_JSON\_VALUE*})  public EmployeeListVO getAllEmployees(  @RequestHeader(name = "X-COM-PERSIST", required = true) String headerPersist,  @RequestHeader(name = "X-COM-LOCATION", defaultValue = "ASIA") String headerLocation)  {  *LOGGER*.info("Header X-COM-PERSIST :: " + headerPersist);  *LOGGER*.info("Header X-COM-LOCATION :: " + headerLocation);    EmployeeListVO employees = getEmployeeList();  return employees;  }    @GetMapping(value = "/employees/{id}",  produces = {MediaType.*APPLICATION\_XML\_VALUE*, MediaType.*APPLICATION\_JSON\_VALUE*})  public ResponseEntity<EmployeeVO> getEmployeeById (@PathVariable("id") Integer id)  {  *LOGGER*.info("Requested employee id :: " + id);    if (id != null && id > 0) {  //*TODO: Fetch the employee and return from here*  EmployeeVO employee = new EmployeeVO(id, "Lokesh","Gupta", "example@gmail.com");  return new ResponseEntity<EmployeeVO>(employee, HttpStatus.*OK*);  }  return new ResponseEntity<EmployeeVO>(HttpStatus.*NOT\_FOUND*);  }    @PostMapping(value = "/employees")  public ResponseEntity<String> createEmployee(EmployeeVO employee)  {  //*TODO: Save employee details which will generate the employee id*  employee.setId(111);    //Build URI  URI location = ServletUriComponentsBuilder.*fromCurrentRequest*()  .path("/{id}")  .buildAndExpand(employee.getId())  .toUri();  return ResponseEntity.*created*(location).build();  }    @PutMapping(value = "/employees/{id}")  public ResponseEntity<EmployeeVO> updateEmployee(@PathVariable("id") int id  ,EmployeeVO employee)  {  //*TODO: Save employee details*  return new ResponseEntity<EmployeeVO>(employee, HttpStatus.*OK*);  }    @DeleteMapping(value = "/employees/{id}")  public ResponseEntity<String> deleteEmployee(@PathVariable("id") int id)  {  //*TODO: Delete the employee record*  return new ResponseEntity<String>(HttpStatus.*OK*);  }      //*TODO: Making it static to persist data - for demo purpose*  static EmployeeListVO *employees* = new EmployeeListVO();    //*TODO: Hardcoded data to replace*  private EmployeeListVO getEmployeeList()  {  EmployeeListVO employees = new EmployeeListVO();    EmployeeVO empOne = new EmployeeVO(1,"Lokesh","Gupta","LokeshGupta@gmail.com");  EmployeeVO empTwo = new EmployeeVO(2,"Amit","Singhal","asinghal@yahoo.com");  EmployeeVO empThree = new EmployeeVO(3,"Kirti","Mishra","kmishra@gmail.com");    employees.getEmployees().add(empOne);  employees.getEmployees().add(empTwo);  employees.getEmployees().add(empThree);    return employees;  }  } |

## After this create web Application named as ***Application.java*** under ***com/example.demo.springexamples/web*** :

|  |
| --- |
| package com.example.springexamples;  import com.example.demo.springexamples.config.HttpClientConfig;  import com.example.demo.springexamples.config.RestTemplateConfig;  import com.example.demo.springexamples.service.UserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.CommandLineRunner;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.context.annotation.Import;  @SpringBootApplication  @Import({ HttpClientConfig.class, RestTemplateConfig.class })  public class Application implements CommandLineRunner {    @Autowired  private UserService userService;  public static void main(String[] args) {  SpringApplication.*run*(Application.class, args);  }  @Override  public void run(String... args) throws Exception {  //userService.testUserService();  }  } |

## After this click on the **Run** button present on the top of the page.



Now go to the browser and type **localhost:9000/employee/{id}** here **{id}** is **1, 2** or **3**.



**Voila!!** We have successfully completed this exercise.