**Spring Boot**

* *ResponseEntity* **represents the whole HTTP response: status code, headers, and body**. Because of it, we can use it to fully configure the HTTP response.
* @GetMapping(**"/age"**)
* ResponseEntity<String> age(
* @RequestParam(**"yearOfBirth"**) **int** yearOfBirth) {
* **if** (isInFuture(yearOfBirth)) {
* **return** **new** ResponseEntity<>(
* **"Year of birth cannot be in the future"**,
* HttpStatus.BAD\_REQUEST);
* }
* **return** **new** ResponseEntity<>(
* **"Your age is "** + calculateAge(yearOfBirth),
* HttpStatus.OK);
* **Additonally we can set Http Headers**
* @GetMapping(**"/customHeader"**)
* ResponseEntity<String> customHeader() {
* HttpHeaders headers = **new** HttpHeaders();
* headers.add(**"Custom-Header"**, **"foo"**);
* **return** **new** ResponseEntity<>(
* **"Custom header set"**, headers, HttpStatus.OK);
* }
* ResponseEntity<T> - Reference
* <https://stackoverflow.com/questions/26549379/when-use-responseentityt-and-restcontroller-for-spring-restful-applications>
* **Rest Service call from one service to another service**
* <https://howtodoinjava.com/spring-boot2/resttemplate/spring-restful-client-resttemplate-example/>
* <https://www.tutorialspoint.com/spring_boot/spring_boot_rest_template.htm>
* <https://www.baeldung.com/rest-template>
* Rest Template is used to create applications that consume RESTful Web Services.
* You can **use** the **exchange()** method to **consume the web services** for all HTTP methods.
* @GetMapping("/service1")
* public ResponseEntity<ResponseModel> showResult(){
* ResponseModel response = new ResponseModel();
* //response.setMessage("Welcomee");
* **HttpHeaders headers = new HttpHeaders();**
* **headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));**
* **HttpEntity <ResponseModel> entity = new HttpEntity<ResponseModel>(headers);**
* **response = restTemplate.exchange("http://localhost:8081/api/employees/service2", HttpMethod.GET, entity, ResponseModel.class).getBody();**
* return new ResponseEntity<ResponseModel>(response, HttpStatus.OK);
* }
* Where Use **HttpEntity** to **wrap the request object.**

**H2 Database in Spring Boot**

**Step 1:-** Add H2 dependency in pom file

**Step 2:-** Update application.properties file

* spring.h2.console.enabled=true
* spring.datasource.platform=h2
* spring.datasource.url=jdbc:h2:mem:testDB

**Step 3: -** We can access the h2 console from browser

* [**http://localhost:8080/h2-console**](http://localhost:8080/h2-console)
* **In case of in memory H2 data base data are not permanent.**
* **When we run the application then only it will active.**
* **Spring Data JPA – Query Method**
* Query method in Spring Data JPA is used to write custom query.
* findById() is the in build method
* Suppose if we want to fetch record by any property name except Id the in Repo interface that custom name should be start with findByTech(String tech) -> where Tech in findByTech is the property name in entity.
* Example :-
* public interface AlienRepo extends CrudRepository<Alien,Integer>{
* Alien findByAname(String aname);
* List<Alien> findByAidGreaterThan(int aid);
* }
* CrudRepository returns the list of obejects in Iterable format
* But JpaRepository returns list of ojects in LIst format or in JSON format....JpaRepository has extra features apart from CRUD methods I.e. pagination etc.
* @Query("from Student where address = ?1 order by sname")
* List<Student> findByAddressSorted(String address);
* **Spring Boot – Security**
* **Step 1 :-** Add dependency for Spring Security
* **Step 2:- Configuration – Create a class for configuration**
* **Example 1:- In memory User Credential**
* @Configuration
* @EnableWebSecurity
* public class AppSecurityConfig extends WebSecurityConfigurerAdapter{
* @Bean
* @Override
* protected UserDetailsService userDetailsService() {
* List<UserDetails> users = new ArrayList<>();
* users.add(User.withDefaultPasswordEncoder().username("Riya").roles("USER").password("1234").build());
* return new InMemoryUserDetailsManager(users);
* }

* }
* Spring Boot Security – MySQL
* **Step 1: -** Add dependency for Spring Data JPA and MYSQL Connector
* **Step 2: -** Update application.properties file with Data base URL , Username , Password
* Update Configuration Class
* Create Entity Class