TEST DE INTEGRATION

//TESTEAR EL CONTROLADOR EN ESTE CASO DE EMPLOYEE//

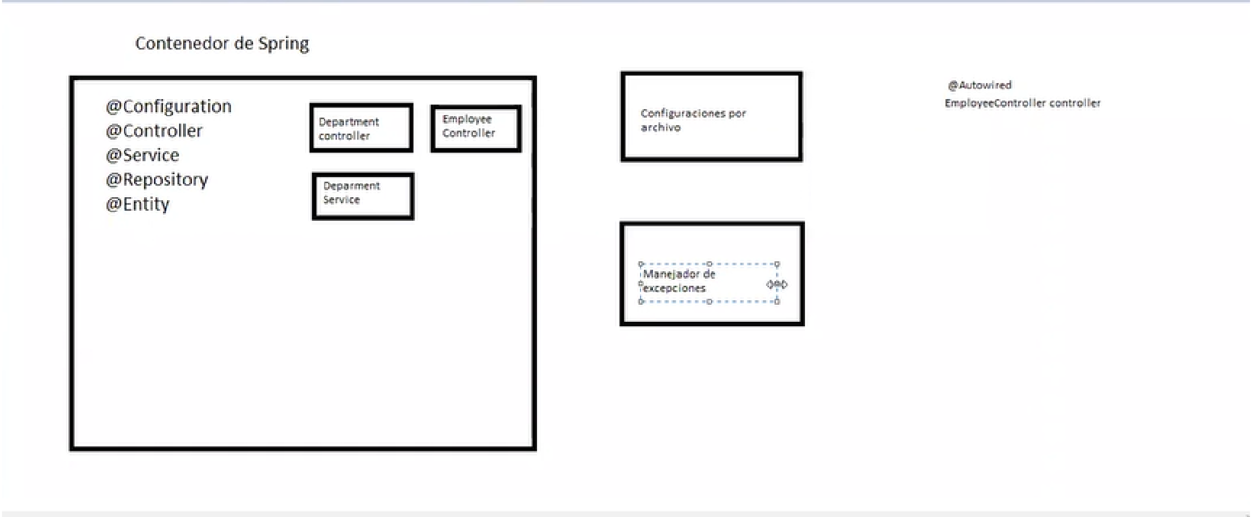
1- crear Carpeta integration. Dentro de el creamos una class en este caso EmployeeControllerIntegrationTest.

```
@SpringBootTest //Anotacion levanta el contexto de la aplicacion.
// Al contenedor de spring y a las aplicaciones le llamamos contexto . Aca queremos todo e
l contexto.
@AutoConfigureMockMvc //Levanta a las config, autoconfigure

public class EmployeeControllerIntegrationTest {

@Autowired
MockMvc mockMvc; // Es como la persona que manda el request.Clase que permite ejecutar las
consultas a un endpoint en particular

ObjectWriter writer; // inicializar un objeto
```



CONTEXTO

```
@BeforeEach
public void setupBeforeAll(){
    writer = new ObjectMapper()//Permite traducir un objeto nuestro a un string plano en J
SON.
            .registerModule(new JavaTimeModule())
            .writer();
}
```

1. Realizamos los test:

```
@Test
public void saveEmployee() throws Exception {
 // Arrange

 EmployeeDTO newEmployee = EmployeeDTOFactory.getTinchoDTO(); // Param necesario

 CrudDTO expected = CrudDTOFactory.crudDTOCreationWithId3(); // La devolucion, Lo que
espero, seria el BodyExcepted



 //REQUEST CON MockHttpServletRequestBuilder & MockMvcRequestBuilders (librerias)

 MockHttpServletRequestBuilder request = MockMvcRequestBuilders  //Declaramos la requ
est que vamos a llamar o hacer
         .post("/employee/save") // POST porque hacemos un save nos fijamos en el con
troller
         .contentType(MediaType.APPLICATION_JSON) // aca aviso que estoy enviando un
 JSON
         .content(writer.writeValueAsString(newEmployee)); // y aca hay que usar el w
ritter . Es el empleado convertido en json


 //Los 3 EXPECTED con ResultMatcher & MockMvcResultMatchers

 ResultMatcher statusExpected = MockMvcResultMatchers.status().isOk(); //StatusExpect
ed el isOk puede varias badrequest ejemplo.


 ResultMatcher bodyExpected = MockMvcResultMatchers.content().json( //BodyExpected
         writer.writeValueAsString(expected)
 );


 ResultMatcher contentTypeExpected = MockMvcResultMatchers.content().contentType(Medi
aType.APPLICATION_JSON); //ContentTypeExpected



 // act & assert con mockmvc
 mockMvc.perform(request)// le paso la request
         .andDo(MockMvcResultHandlers.print()) //Devuelve el request de manera gráfic
a
         .andExpect(statusExpected)
         .andExpect(bodyExpected)
         .andExpect(contentTypeExpected);
}
```

```
@Test
public void findOneEmployee() throws Exception {
// arrange
Integer id = EmployeeDTOFactory.getJeanDTO().getId();//PARAM REQUERIDO
EmployeeDTO expected = EmployeeDTOFactory.getJeanDTO();// ESPERO UN EMPLOYEEDTO

//REQUEST CON MockHttpServletRequestBuilder & MockMvcRequestBuilders (librerias)
          //Declaramos la request que vamos a llamar o hacer
    MockHttpServletRequestBuilder request = MockMvcRequestBuilders.get("/employee/get/"+i
d);


    //Los 3 EXPECTED con ResultMatcher & MockMvcResultMatchers
         //StatusExpected
    ResultMatcher statusExpected = MockMvcResultMatchers.status().isOk();

         //BodyExpected
    ResultMatcher bodyExpected = MockMvcResultMatchers.content().json(
            writer.writeValueAsString(expected)
    );

          //ContentTypeExpected
    ResultMatcher contentTypeExpected = MockMvcResultMatchers.content().contentType(MediaT
ype.APPLICATION_JSON);


    // act & assert con mockmvc
    mockMvc.perform(request)
            .andDo(MockMvcResultHandlers.print()) //Devuelve el request de manera gráfica
            .andExpect(statusExpected)
            .andExpect(bodyExpected)
            .andExpect(contentTypeExpected);
}
```

```
@Test
public void deleteEmploee() throws Exception {
//arrange
    Integer id = 1;
    CrudDTO expected = CrudDTOFactory.crudDTODeletationWithId1();

    //REQUEST CON MockHttpServletRequestBuilder & MockMvcRequestBuilders (librerias)
       //Declaramos la request que vamos a llamar o hacer
    MockHttpServletRequestBuilder request = MockMvcRequestBuilders.delete("/employee/delet
e/{id}",id);

    //Los 3 EXPECTED con ResultMatcher & MockMvcResultMatchers
        //StatusExpected
    ResultMatcher statusExpected = MockMvcResultMatchers.status().isOk();

        //BodyExpected
    ResultMatcher bodyExpected = MockMvcResultMatchers.content().json(
            writer.writeValueAsString(expected)
    );

       //ContentTypeExpected
    ResultMatcher contentTypeExpected = MockMvcResultMatchers.content().contentType(MediaT
ype.APPLICATION_JSON);

    // act & assert con mockmvc
    mockMvc.perform(request)
            .andDo(MockMvcResultHandlers.print()) //Devuelve el request de manera gráfica
            .andExpect(statusExpected)
            .andExpect(bodyExpected)
            .andExpect(contentTypeExpected);
}
```

}

1. EN CrudDTOFactory:

```
public static CrudDTO crudDTOCreationWithId1(){
return CrudDTO.builder()
.message("Se creo el empleado con id: " + 0)
.action(CrudEnum.CREATION)
.build();
}

public static CrudDTO crudDTODeletationWithId1(){
```

```java
    return CrudDTO.builder()
            .message("Se elimino el empleado con id " + 1)
            .action(CrudEnum.DELETATION)
            .build();
}

public static CrudDTO crudDTOCreationWithId2(){
    return CrudDTO.builder()
            .message("Se creo el departamento con id: " + 2)
            .action(CrudEnum.CREATION)
            .build();
}

public static CrudDTO crudDTODeletationWithId2(){
    return CrudDTO.builder()
            .message("Se elimino el departamento con id " + 2)
            .action(CrudEnum.DELETATION)
            .build();
}

public static CrudDTO crudDTOReadingNotFoundWithId7(){
    return CrudDTO.builder()
            .message("No se pudo encontrar el empleado con id " + 7)
            .action(CrudEnum.READING)
            .build();
}
```