Chuleta Spring Framework *versión 30.11.2023*

Controlador y Servicio	
@Controller	Definición de controlador
@RequestMapping ("/rutaBase")	Ruta raíz de todos los mappings
@GetMapping("/ruta")*@PostMapping, @PutMapping,@DeleteMapping* public String showPage (Model model){ model.addAttribute ("city","Lugo"); return "vista"; }	Verbo HTTP al que responde el metodo Método que devuelve vista Pase de parámetros a la vista Vista html que muestra
@GetMapping({"/ruta1","/ruta2","/ruta3"})	Responde a varias rutas
@GetMapping("/ruta") public String showPage (@RequestParam String p, *Model model*){ *@RequestParam (required=false,defaultValue="X") String p* *@RequestParam Optional <String> p*	Recibe param query en URL: ?p=valor *Evitar error si vacío* *Evitar error si vacío: p.orElse("X")*
@GetMapping("/ruta/{p}") public String showPage (@PathVariable String p, *Model model*){ return "redirect:/rutaCompleta"; @Service public class MiServicioClase implements MiServicioInterfaz { *En el controlador:* @Autowired MiServicioInterfaz miServicioInterfaz;	Recibe param en path URL: ruta/p Redirige a otro controlador, no vista Definición de Clase de servicio Inyectamos la interfaz

Vistas + Thymeleaf	
<html xmlns:th="http://www.thymeleaf.org">	Etiqueta html para vistas con Thymeleaf
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet"> <script src="/webjars/bootstrap/js/bootstrap.bundle.min.js"></script>	Incluir Bootstrap con webjars
*	Variable con Thymeleaf
0}">. . . <span th:unless="${res>0}">. . .	Condicional con Thymeleaf
<div th:each="nombre:${listaNombres}"> <p th:text="${nombre}">*</p> 	For…each con Thymeleaf (texto)
<div th:each="producto:${listaProductos}"> <p th:text="${producto.nombre}">*</p>	For…each con Thymeleaf (objeto) *Necesita getters*

<head th:fragment="myHead">. . .</head> *(en templates/fragment.html)*
<head th:replace="~{/fragment.html::myHead}"></head>

Definición de fragmento con Thymeleaf Sustitución de etiqueta por fragmento

<a th:href="@{/ruta}">link <a th:href="@{/ruta (param=${variable}) }">link <a th:href="@{/ruta/*{param}* (param=${variable}) }">link	Enlace a ruta (URL local) Enlace a ruta con variable en query Enlace a ruta con variable en path

Formularios	
form action="#" method="post" th:action="@{/myForm/submit}" th:object="${formInfo}">	Etiqueta form con destino Y objeto con datos
<input type="text" id="nombre" th:field="*{nombre}">	Atributo de tipo texto vinculado a objeto

@GetMapping ("/myForm")
public String showForm (Model model) {
model.addAttribute ("formInfo", new FormInfo());
return "formView"; }
	Mapping presentación formulario

@PostMapping("/myForm/submit")
public String myformSubmit(FormInfo formInfo) {. . .
public String myformSubmit(@ModelAttribute FormInfo formInfo) {. . .

BindingResult br) {if (br.hasErrors()) {. . .
Recepción formulario
formInfo es el objeto con los datos recib. @ModelAtt…
los pasa a la vista directm.

Validación de datos recibidos

public String myformSubm(@Valid FormInfo formInfo,

Modelo	
@Getter, @Setter, @EqualsAndHashCode, @ToString = @Data	Anotaciones Lombok
@NoArgsConstructor, @AllArgsConstructor	Anotaciones Lombok
@Min(value=0), @NotEmpty, @Email, @AssertTrue, @Size	Validaciones en clase formulario
@Configuration,@Getter,@Setter @PropertySource("classpath:/fich.properties") public class MiClase { @Value("${iva}") private Double iva; }	Fichero de parámetros Ruta del archivo Clase y mapeo de cada campo
private List<MiClase> repositorio = new ArrayList<>();	Repositorio en Memoria en el servicio.

JPA	
@Entity @Id @notEmpty @Email @Min @Max	Entidad gestionada por JPA Clave de una entidad

@GeneratedValue	@Id es autogenerado por la BD.
@Data @AllArgsConstructor @NoArsgConstructor @EqualsAnsHashCode(of = “id”) @Entity	Para las clases, antes de class Objeto
public interface ProductoRepository extends JpaRepository <T, ID> {}	Repositorio JPA
List <T> findAll() *Optional <T>* findById (ID id) *void* delete (T t), *void* deleteById (ID id) *T* save (T)	Métodos de repositorio incluidos generados por defecto. Hay más…
List<Persona> findByEmail (String email); Empleado findTopByDepartamentoOrderBySalarioDesc (Departamento dep); List<Objeto> findByXXX(XXX declaracion) Ej: List<Jugador> findByEquipo (Equipo equipo) public List<Objeto> findByNombre (String nombre); List<Objeto> findByNombreAndEmail (String nombre, String email); List<Objeto> findByNombreEquals (String nombre); List<Objeto> findByEdad (Integer edad); List<Objeto> findBySalarioGreaterThanEqualOrderBySalario (double salario); (salario mayor o igual a X)	Metódos de repo. derivados por nombre

@Query("select e from Empleado e where e.salario>=?1") *List <Empleado>* getEmpleadoSalarioAlto (Double salar);
 	
@Query("SELECT j FROM jugador j WHERE j.equipo.id == ?1 ") //and j.edad > ?2 asi es para usar mas paramentros
List<Juagrod> buscarPorEquipo (Long idEquipo)
 	
@Query("SELECT j FROM jugador" + "j WHERE j.edad < ?1 " + "order by edad limit 3")
List<Juagrod> buscarPorMenores (Inreger edad)
 	
avg = media
count = size()
 	
`@Query ("select e from Empleado e " +`
`"where e.departamento.id = ?1 " +`
`"and e.salario = (select max(salario) from Empleado " +`
	Metodo de repo. con query.

`"where departamento.id = ?1)"`
 	
`)`
`Empleado obtenerEmpleadoMaxSalarioPorDepartamento(Long idDepartamento);`	

@ManyToOne @OnDelete (action = OnDeleteAction.CASCADE) private Departamento departamento; @ToString.Exclude	Relación muchos a uno (n empleados – 1 departamento) -> Para no generar bucles
Entidad NM: {...@ManyToOne N + @ManyToOne M + AtribExtra} Entidad N: @OneToMany mappedBy=”n” List <NM> nm = new... Entidad M: @OneToMany mappedBy=”m” List <NM> nm= new ..	Relac. muchos a muchos N-M con atrib.extra. …esta línea solo si bidirecc …esta línea solo si bidirecc

public List<Empleado> getEmpleadosPaginados(Integer pageNum) { Pageable paging = PageRequest.of(pageNum, 10, Sort.by("nombre").ascending()); Page<Empleado>pagedResult = empleadoRepository.findAll(paging); if (pagedResult.hasContent()) return pagedResult.getContent(); else return null; }
pagedResult.getTotalPages()
 	
public int getTotalPaginas() {
Pageable paging = PageRequest.of(0, pageSize, Sort.by("nombre"));
Page<Empleado> pagedResult = empleadoRepository.findAll(paging);
return pagedResult.getTotalPages();
}
 	
Falta los botones en view y el controlador
	Resultados paginados, de 10 en 10, ordenador por nombre.
	
	Devuelve la página pasada como
	
	parámetro Total páginas
	
	Total páginas

1) Añadir al repositorio de empleados el método derivado por nombre:
`List <Empleado> findByDepartamento(Departamento departamento);`
y luego en el servicio:
`Long cantEmpleadosDepto = (long) empleadoRepository.findByDepartamento( de partamento).size();`
`if (cantEmpleadosDepto == 0) departamentoRepository.delete(departam en to);`
 	
2) Añadir al repositorio de empleados el método derivado por nombre pero con countBy en vez de findBy:
`Long countByDepartamento (Departamento departamento);`
con lo que el servicio ya no haría falta el size().
 	
3) Añadir al repositorio de empleados el método @Query con la misma consulta que el anterior.
`@Query("select count (e) from Empleado e " + "where e.departamento.id = ?1") Long cantidadEmpleadosDepto(Long idDepto);`
y en el servicio emplear este método.
	Borrado sin cascada

`List <Empleado> findByOrderByEmailDesc();`
 	
`empleadoRepository.findAll(Sort.by(Sor t. Direction.DESC, "email"));`
	Ordenar datos

pom.xml
<artifactId> <name> Nombre del proyecto
<dependencies>web, thymeleaf, devtools,test, Dependencias básicas
<dependencies> webjars-bootstrap,webjars-locator Dependencia para BootStrap con Maven
<dependencies> lombok, data-jpa, validation, h2, modelmapper Dependencia para acceso a datos
<dependencies> springfox-boot-starter, springfox-swagger-ui Incorporar Swagger
<dependencies> spring-boot-starter-webflux WebClient

<dependencies> spring-boot-starter-security, thymeleaf-extras-sprigsecurity5
<dependencies> Control de acceso

Seguridad MVC	
@Bean public SecurityFilterChain filterChain(HttpSecurity http) throws Exception { http.authorizeRequests().antMatchers(*rutas*).*permisos* .anyRequest().authenticated().and() .formLogin().permitAll().and().logout().permitAll().and() .exceptionHandling().accessDeniedPage("/accessError"); return http.build(); }	Control de acceso: Permisos: permitAll(), hasRole (rol), hasAnyRole (rol1,rol2), denyAll()

application.properties
server.port=9000 Puerto del servidor (defecto 8080)

spring.thymeleaf.cache=false Refresh automático plantilla

spring.datasource.url=jdbc:h2:mem:nombreBD spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa spring.datasource.password= spring.jpa.database-platform=org.hibernate.dialect.H2Dial ect spring.jpa.show-sql=true spring.h2.console.enabled=true	**JPA:** nombre base de datos, Driver del SGBD usuario/pass de la BD. Mostrar instruc. SQL en consola Habilitar Consola
server.error.include-message=always server.error.include-stacktrace=never	Configuración de seguridad
spring.mvc.pathmatch.matching-strategy=ant-path-matcher	Para Swagger

Otros

@Bean CommandLineRunner metodo(argumentos) {
return args -> { /*código*/ }; }
@Bean CommandLineRunner initData(MiServicio miServicio) { return args -> { miServicio.add(new Persona("pepe",10)); }; } Ej para borrar un jugador por edad:

void borrarJugadorPorEdad(intteger edad){

List<Jugador> borrables = jugadorRepository.findByEdad(edad);
for(Jugador j: borrables){

jugadorRepository.delete(j);

}
CommandLineRunner (en clase con main) para código inicial.
Ejemplo: añadir al repo desde servicio