1) S05_s1 - Material 01_KMSRPW.pdf — *Introducción al Back-End con Java y Spring Boot*

Este PDF es la clase básica sobre qué hace el backend, por qué usar Java + Spring Boot y trae ejercicios prácticos (crear proyecto, endpoints, cambiar puerto, ejemplo de sumador y registro).

S05_s1 - Material 01_KMSRPW

Qué es (en palabras simples)

- **Backend** = "la parte que corre en el servidor" que procesa datos, guarda en base de datos y responde a peticiones del navegador o apps.
- **Java** = lenguaje de programación usado en servidores.
- **Spring Boot** = marco que facilita crear aplicaciones Java para web (APIs, páginas) sin complicarte con mucha configuración.

Por qué usarlo

- Es robusto para aplicaciones empresariales.
- Spring Boot **automático**: arranca servidor embebido, detecta dependencias y reduce configuración.
- Gran ecosistema (seguridad, datos, etc.).

S05_s1 - Material 01_KMSRPW

Cuándo usarlo

- Cuando necesitas una aplicación estable, escalable o integrarte con sistemas empresariales.
- Cuando tu equipo ya usa Java o cuando quieres APIs REST robustas.

Cómo empezar (pasos concretos, nivel 0)

1. **Instala Java JDK** (versión 11 o superior; si puedes, JDK 17).
2. **IDE recomendado**: NetBeans o Eclipse (el PDF usa NetBeans).
3. Crear proyecto: usa *Spring Initializr* (desde el IDE o web): elige Grupo/Artifact, y agrega dependencia **Spring Web** (y Thymeleaf si quieres páginas).

S05_s1 - Material 01_KMSRPW

4. Ejecutar:
    - Desde IDE: run (la clase con public static void main que Spring Boot crea).
    - Desde terminal con Maven wrapper: ./mvnw spring-boot:run (Windows: mvnw.cmd spring-boot:run) o mvn spring-boot:run si tienes Maven.
5. Probar en el navegador: http://localhost:8080 (o el puerto que configures).

Ejemplos mínimos (copia y pega)

Clase principal (generada por Spring Initializr):

```
@SpringBootApplication
public class HolaSpringApplication {
  public static void main(String[] args) {
    SpringApplication.run(HolaSpringApplication.class, args);
  }
}
```

Endpoint simple (/mensaje):

```
@RestController
public class MensajeController {
  @GetMapping("/mensaje")
  public String mensaje() {
    return "¡Bienvenido al back-end!";
  }
}
```

- Prueba: curl http://localhost:8080/mensaje → devuelve el texto.

Sumador (/calculadora/sumar?a=5&b=3):

```
@RestController
@RequestMapping("/calculadora")
public class CalculadoraController {
 @GetMapping("/sumar")
 public int sumar(@RequestParam int a, @RequestParam int b) {
  return a + b;
 }
}
```

POST JSON /registro:

```
@RestController
public class RegistroController {
 @PostMapping("/registro")
 public ResponseEntity<Usuario> registrar(@RequestBody Usuario usuario) {
  // procesar (guardar etc.)
  return ResponseEntity.ok(usuario);
 }
}
class Usuario {
 public String nombre;
 public String email;
```

// getters/setters o usar lombok

}

Estos ejercicios vienen directamente en el PDF como práctica.

S05_s1 - Material 01_KMSRPW

Cambiar puerto (application.properties)

Pon en src/main/resources/application.properties:

server.port=8888

(Ejercicio del PDF pedía cambiar el puerto a 8888).

S05_s1 - Material 01_KMSRPW

Errores comunes y soluciones rápidas

- **500 / ClassNotFound**: falta dependencia en pom.xml.
- **404 para endpoint**: verifica la URL y que la clase tenga @RestController y @GetMapping.
- **Puerto en uso**: cambia server.port.
- **JSON mal formado** en POST → check Content-Type: application/json.

Mini plan de práctica (sigue esto)

1. Instala JDK + IDE.
2. Crea proyecto con Spring Initializr (añade Spring Web).
3. Implementa /mensaje.
4. Implementa /calculadora/sumar.
5. Implementa POST /registro y prueba con curl o Postman.
   (El PDF te guía para estas actividades).

---

2) Semana 06.pptx — *Spring Web: definiciones, usos, ventajas y aplicaciones*

Este PPT explica qué es **Spring Web**, las anotaciones principales (@Controller, @RestController, @RequestMapping), usos (APIs, controladores), y por qué elegirlo.

Semana 06

Qué es (nivel 0)

- **Spring Web** es el módulo que maneja las peticiones HTTP en aplicaciones Spring. Convierte URLs en métodos Java (controladores) y devuelve respuestas (HTML o JSON).

Conceptos clave (muy simple)

- **DispatcherServlet**: es como el recepcionista que recibe la petición y la manda al controlador correcto.
- **Controlador**: clase con métodos que responden a rutas (/usuarios, /productos).
- **@Controller**: para devolver vistas (HTML).
- **@RestController**: para devolver datos (JSON/text).
- **@GetMapping / @PostMapping**: atajos para manejar GET o POST.

Ejemplo de controlador (REST)

```
@RestController
@RequestMapping("/api")
public class ApiController {
 @GetMapping("/hola")
 public String hola() { return "hola"; }
```

```
 @PostMapping("/echo")
 public Map<String,String> echo(@RequestBody Map<String,String> body) {
   return body;
 }
}
```

Flujo resumido (qué pasa cuando abres una URL)

1. Navegador pide GET /productos.
2. **DispatcherServlet** recibe la petición.
3. Busca qué controlador y método manejan /productos.
4. Ejecuta el método, que puede pedir datos a servicios/repositorios.
5. Devuelve respuesta (JSON o una vista HTML).

Semana 06

Cuándo usar Spring Web

- Para crear **APIs REST** que consumen clientes móviles o SPAs.
- Para aplicaciones web que requieren lógica en servidor (formularios, autenticación).

Semana 06

Ventajas destacadas (del PPT)

- Fácil configuración con anotaciones.
- Integración con otros módulos (Security, Data, etc.).
- Escalable y con buena comunidad.

Semana 06

Errores comunes (puntuales de Spring Web)

- Usar @Controller esperando JSON (usa @RestController o @ResponseBody).
- No mapear bien parámetros (@RequestParam, @PathVariable).
- Intentar devolver una vista sin tener la dependencia del motor de plantillas (Thymeleaf) o plantillas en la carpeta correcta.

---

3) Semana 07.pptx — *Thymeleaf: definiciones, usos, ventajas, aplicaciones*

El PPT explica Thymeleaf, un motor de plantillas para Java (ideal para páginas HTML generadas en el servidor) y sus ventajas (natural templating, integración con Spring).

Semana 07

¿Qué es Thymeleaf? (nivel 0)

- Es un motor que **mezcla HTML con expresiones** para insertar datos desde Java.
- "Natural templating": los archivos .html se ven como HTML normal y se pueden abrir en navegador sin servidor (útil para diseño).

Por qué usar Thymeleaf

- Fácil de editar por diseñadores (plantilla sigue siendo HTML).
- Integración nativa con Spring MVC.
- Buen soporte para formularios y enlaces.

Semana 07

Cuándo usar Thymeleaf

- Aplicaciones donde quieres **renderizar HTML en el servidor** (web tradicional) y necesitas SEO o páginas ya cargadas desde el servidor.
- Prototipos o apps con poca interactividad en cliente (no SPA).

Cómo usarlo (pasos concretos)

1. Añade dependencia en pom.xml:

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2. Crea plantillas en src/main/resources/templates/ (ej. index.html).
3. Controlador devuelve nombre de vista:

```
@Controller
public class HomeController {
 @GetMapping("/")
 public String home(Model model) {
  model.addAttribute("mensaje", "Hola desde Thymeleaf");
  return "index"; // index.html en templates
 }
}
```

4. En index.html usa namespaces de Thymeleaf:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head><title>Inicio</title></head>
<body>
 <h1 th:text="${mensaje}">Mensaje por defecto</h1>
```

</body>

</html>

(Este flujo está descrito en el PPT).

Semana 07

Funciones comunes (ejemplos sencillos)

- th:text="${var}" → reemplaza texto.
- th:each="item : ${lista}" → iterar sobre colecciones.
- th:if="${condicion}" → condicional.
- th:href="@{/ruta}" → construir enlaces.
- Formularios con th:object y th:field:

```
<form th:action="@{/registro}" th:object="${usuario}" method="post">
  <input th:field="*{nombre}" />
  <input th:field="*{email}" />
  <button type="submit">Enviar</button>
</form>
```

Fragmentos / reusar plantillas

- Puedes definir header y footer como fragmentos y reutilizarlos con th:replace o th:insert (útil para no repetir código).

Semana 07

Errores comunes Thymeleaf

- TemplateInputException: plantilla no encontrada → revisa ruta templates/.
- Variables null → muestra valores por defecto o comprobaciones th:if.

- Cache en desarrollo: cambia spring.thymeleaf.cache=false en application.properties para ver cambios sin reiniciar.