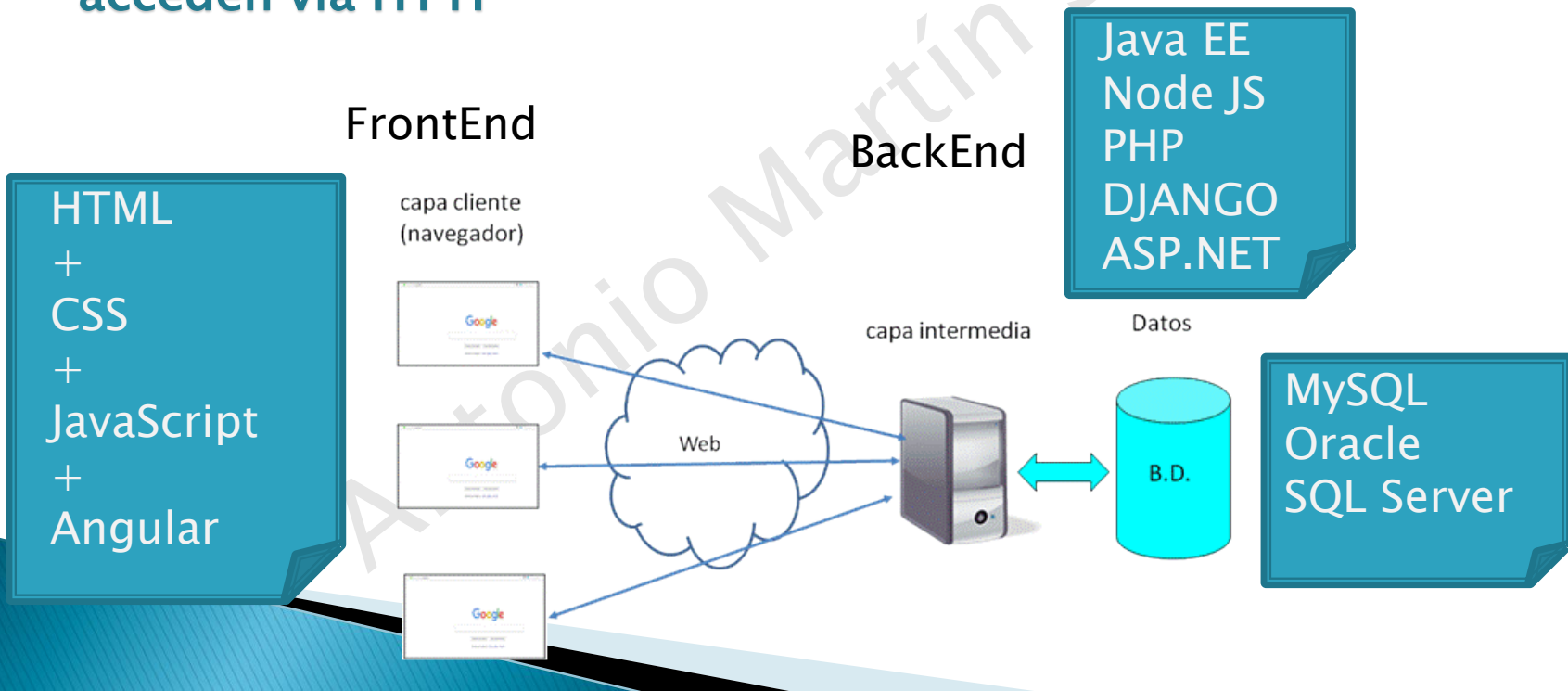


Aplicaciones Web con Spring



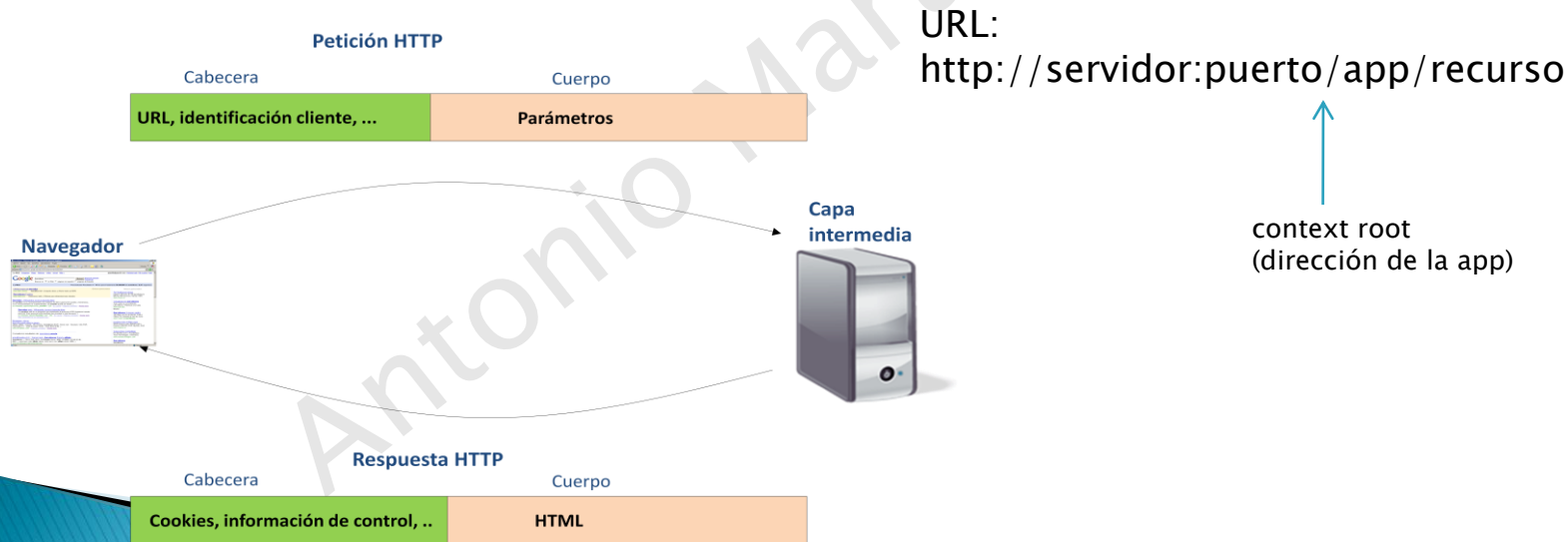
Arquitectura aplicaciones Web

➤ Se basan en estructura de tres capas, donde la app reside en un servidor (capa intermedia) al que los clientes navegadores acceden vía HTTP



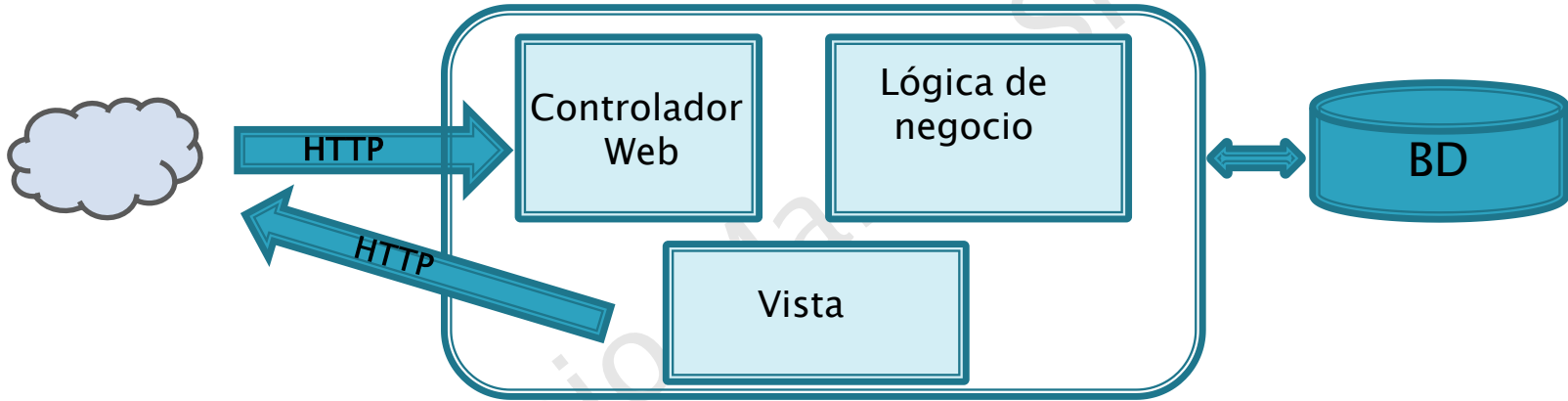
Interacción cliente-servidor

- En una arquitectura Web, el navegador cliente se comunica con la capa intermedia utilizando el protocolo HTTP. HTTP es un protocolo basado en un mecanismo petición-respuesta



Estructura de una app Web

Patrón MVC (backend)



➤ Patrón Modelo Vista Controlador:

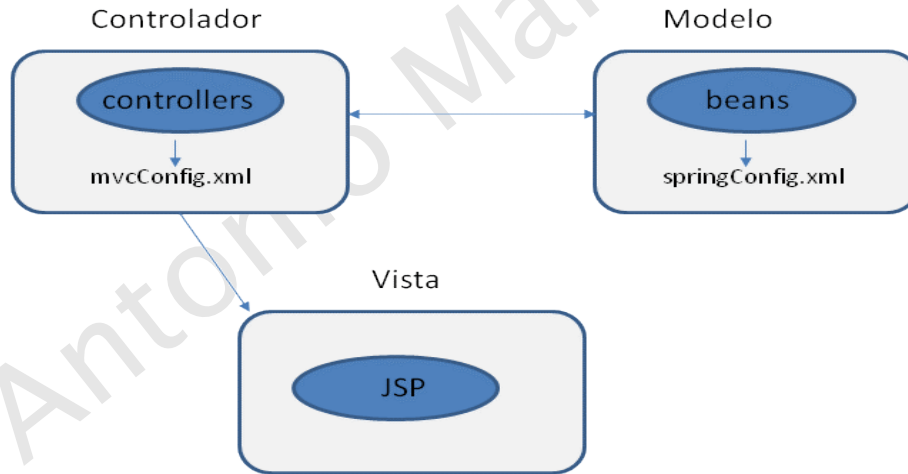
- Modelo. Lógica de negocio
- Controlador. Recepción y gestión de peticiones
- Vista. Generación de respuestas

Spring en aplicaciones Web

- Siguen el patrón Modelo Vista Controlador
- Proporciona utilidades que simplifican el proceso de creación de aplicaciones frente a Java EE
- Requiere las dependencias core, context, web y webmvc

Estructura general

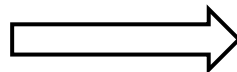
- Controlador y modelo mediante clases estándares Java
- Vistas mediante JSP



Configuración en Spring

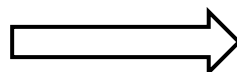
- Configuración específica de clases a través de anotaciones
- Configuración de datos de aplicación a través de archivos XML o clases de configuración
- Cada bloque suele tener su archivo/clase de configuración propia:

controlador



mvcConfig.xml
mvcConfig.java

modelo



springConfig.xml
springConfig.java

El modelo en Spring

Estructura general

- Se implementa en clases estándares anotadas con `@Service`.
- La interacción con el controlador se realiza mediante interfaz
- Las instancias son inyectadas en el controlador mediante `@Autowired`

Controlador

```
@Controller  
public class ClaseController {  
    @Autowired  
    InterfazService service;  
}
```

Modelo


```
public interface InterfazService {  
  
}
```

```
@Service  
public class ClaseService  
implements InterfazService {  
  
}
```

onio Martín Sierra

Configuración del modelo

Paquete con clases a
instanciar

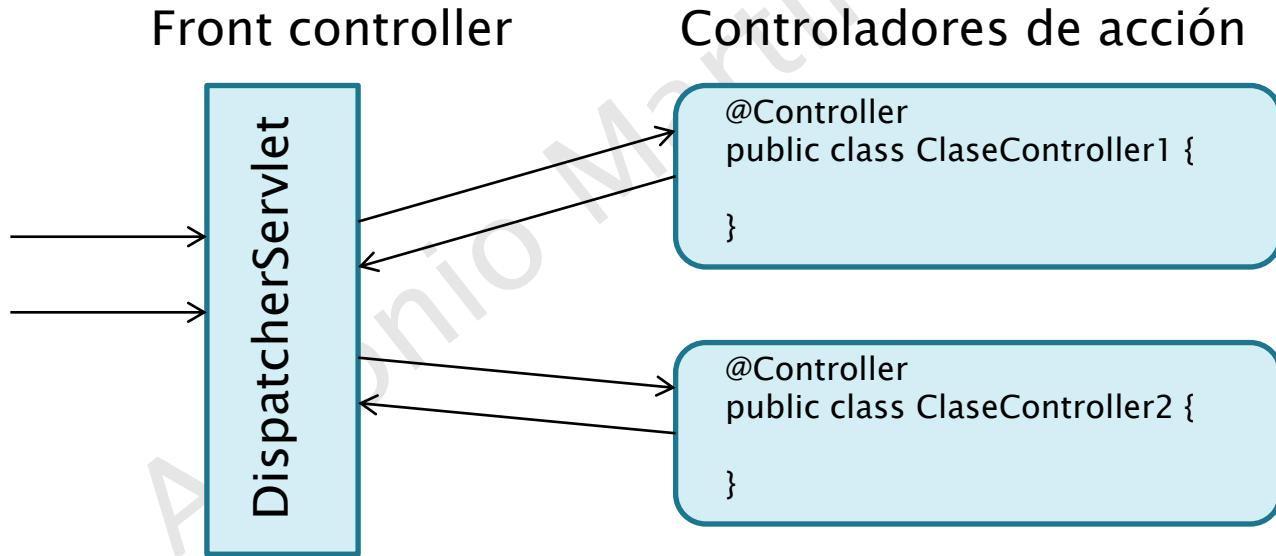


```
@ComponentScan(basePackages = {"com.formacion.service"})
@Configuration
public class SpringConfig {
    //definición de métodos @Bean para instanciación
    //de objetos
}
```

El controlador en Spring

Estructura general

- Además de Front controller, DispatcherServlet inicia el framework Spring.



Métodos controladores de acción

```
@Controller
public class ClaseController1 {
    @GetMapping(value="direccion")
    public String metodo(..){
        ...
        return "pagina";
    }
}
```

Dirección asociada al controlador

El método puede recibir diferentes tipos y variedad de parámetros

Nombre de la página JSP de respuesta

Configuración del controlador

Paquete con clases a instanciar

```
@ComponentScan(basePackages= {"com.formacion.controller"})
@EnableWebMvc
@Configuration
public class MvcConfig implements WebMvcConfigurer{
    @Bean
    public InternalResourceViewResolver getInternalResourceViewResolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/");
        resolver.setSuffix(".jsp");
        return resolver;
    }
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/volver").setViewName("datos");
    }
}
```

objeto para resolver
las vistas

Navegaciones
estáticas

Configuración Web

➤ Se debe definir una clase de configuración para Java EE que cargue el front controller y registre las clases de configuración de Spring:

```
//configuración para Java EE
public class InitConfig implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext container) {
        // Registra la clase de configuración del modelo
        AnnotationConfigWebApplicationContext rootContext = new AnnotationConfigWebApplicationContext();
        rootContext.register(ServiceConfig.class);
        // Registra la clase de configuración del controlador
        AnnotationConfigWebApplicationContext dispatcherContext = new AnnotationConfigWebApplicationContext();
        dispatcherContext.register(ControllerConfig.class);
        // Gestiona el ciclo de vida del contexto de aplicación
        container.addListener(new ContextLoaderListener(rootContext));
        // Crea y registra el DispatcherServlet
        ServletRegistration.Dynamic dispatcher =
            container.addServlet("dispatcher", new DispatcherServlet(dispatcherContext));
        dispatcher.setLoadOnStartup(1);
        dispatcher.addMapping("/");
    }
}
```

Vistas con Spring

Java Server Pages(JSP)

- Componente que forma parte de una aplicación Web JavaEE
- Archivo de texto en el que se combinan bloques HTML con código Java (scriptlet) que se ejecuta en el servidor

.jsp



- Adecuado para la generación de respuestas

Código Java en JSP

➤ Se delimita por `<% y %>`. Pueden aparecer en cualquier parte de la página

➤ Puede ser:

- Scriptlet. Bloque Java que realiza alguna tarea
- Expresión. Devuelve un resultado a la página

scriptlet

```
<body>
<center>
<%for(int i=1;i<=6;i++){ %>
<h<%=i%>> Bienvendio a mi página</h<%=i%>>
<%} %>
</center>
</body>
```

expresión

Bienvendio a mi página

Bienvendio a mi página

Bienvendio a mi página

Bienvendio a mi página

Bienvendio a mi página

Bienvendio a mi página

Objetos implícitos JSP

➤ Dentro de un scriptlet podemos hacer uso de una serie de objetos implícitos:

- request. Instancia de HttpServletRequest
- response. Instancia de HttpServletResponse
- session. Instancia de HttpSession
- application. Instancia de ServletContext
- exception. Instancia de excepción creada
- out. Objeto PrintWriter de salida Http

Thymeleaf

- Basado en la utilización de páginas HTML
- Mejora el rendimiento respecto a JSP
- El contenido dinámico se genera mediante tags especiales que se utilizan como atributos de etiquetas HTML:

```
<html lang="es" xmlns:th="http://www.thymeleaf.org">  
  <div th:each="it:${items}">  
    <h3>  
      <a th:href="${it.url}" th:text="${it.url}"></a><br>
```

Namespace thymeleaf

tags

Dependencias thymeleaf

➤ Para utilizar thymeleaf se requieren las siguientes dependencias:

```
<dependency>  
  <groupId>org.thymeleaf</groupId>  
  <artifactId>thymeleaf</artifactId>  
  <version>${thymeleaf.version}</version>  
</dependency>  
<dependency>  
  <groupId>org.thymeleaf</groupId>  
  <artifactId>thymeleaf-spring5</artifactId>  
  <version>${thymeleaf.version}</version>  
</dependency>
```

➤ En Spring Boot se utiliza el starter:

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

Configuración thymeleaf

➤ En aplicaciones sin Spring Boot:

```
@Bean
public SpringResourceTemplateResolver templateResolver(){
    SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();
    templateResolver.setApplicationContext(this.applicationContext);
    templateResolver.setPrefix("/");
    templateResolver.setSuffix(".html");
    // HTML es la plantilla por defecto, se indica por claridad.
    templateResolver.setTemplateMode(TemplateMode.HTML);
    return templateResolver;
}

@Bean
public SpringTemplateEngine templateEngine(){
    SpringTemplateEngine templateEngine = new SpringTemplateEngine();
    templateEngine.setTemplateResolver(templateResolver());
    templateEngine.setEnableSpringELCompiler(true);
    return templateEngine;
}

@Bean
public ThymeleafViewResolver viewResolver(){
    ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
    viewResolver.setTemplateEngine(templateEngine());
    return viewResolver;
}
```

➤ En Spring Boot configuración por defecto