Frameworks para Aplicaciones Web
Spring

Frameworks

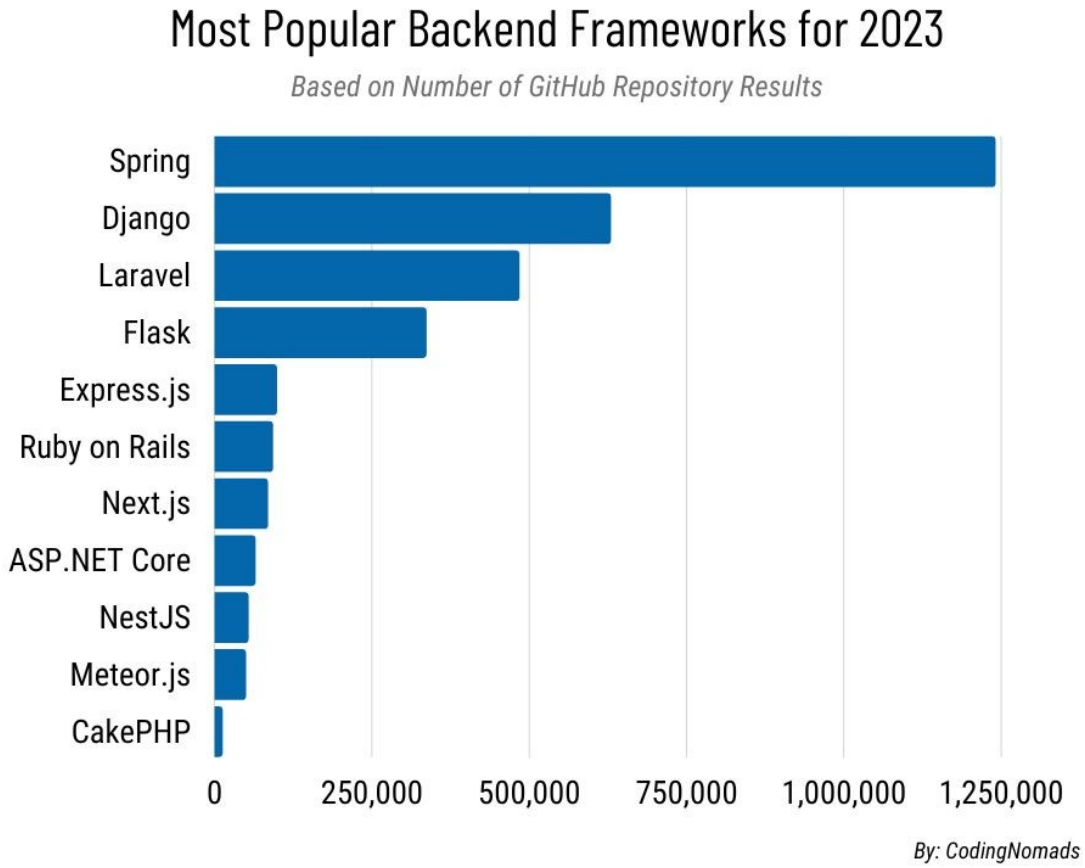
- “Entorno de trabajo”
  - Conjunto estandarizado de conceptos
  - Sirve como guía para la construcción de sistemas
  - Facilita la mantención y agregar nuevas funcionalidades a una aplicación
  - Provee separación de responsabilidades por capas
  - Proveen funcionalidades listas

Frameworks

- Backend, server-side
  - Permite crear la lógica y funcionalidades para la aplicación web
    - No es visible para el usuario final
  - Controla lo que hace la aplicación y cómo lo hace
  - Algunos beneficios:
    - Gestión de autenticación y autorización
    - Recopila, recupera, organiza, almacena y analiza datos
    - Define que vista y datos se debe presentar al usuario final
    - Gestiona los estados de la aplicación (ejemplo: carro de compras)
    - Procesa pagos de forma segura
    - Gestión segura de datos de usuario

Frameworks

- Backend



Spring Framework

- https://spring.io/

- https://spring.io/why-spring

Spring MVC

- Framework que es parte de Spring
- Diseñado con DispatcherServlet
  - Envía las solicitudes a controladores
  - Configurable handler mappings
  - Resolución de vistas
  - Locale...etc
- Los handler están basados en anotaciones:
  - @Controller
  - @RequestMapping

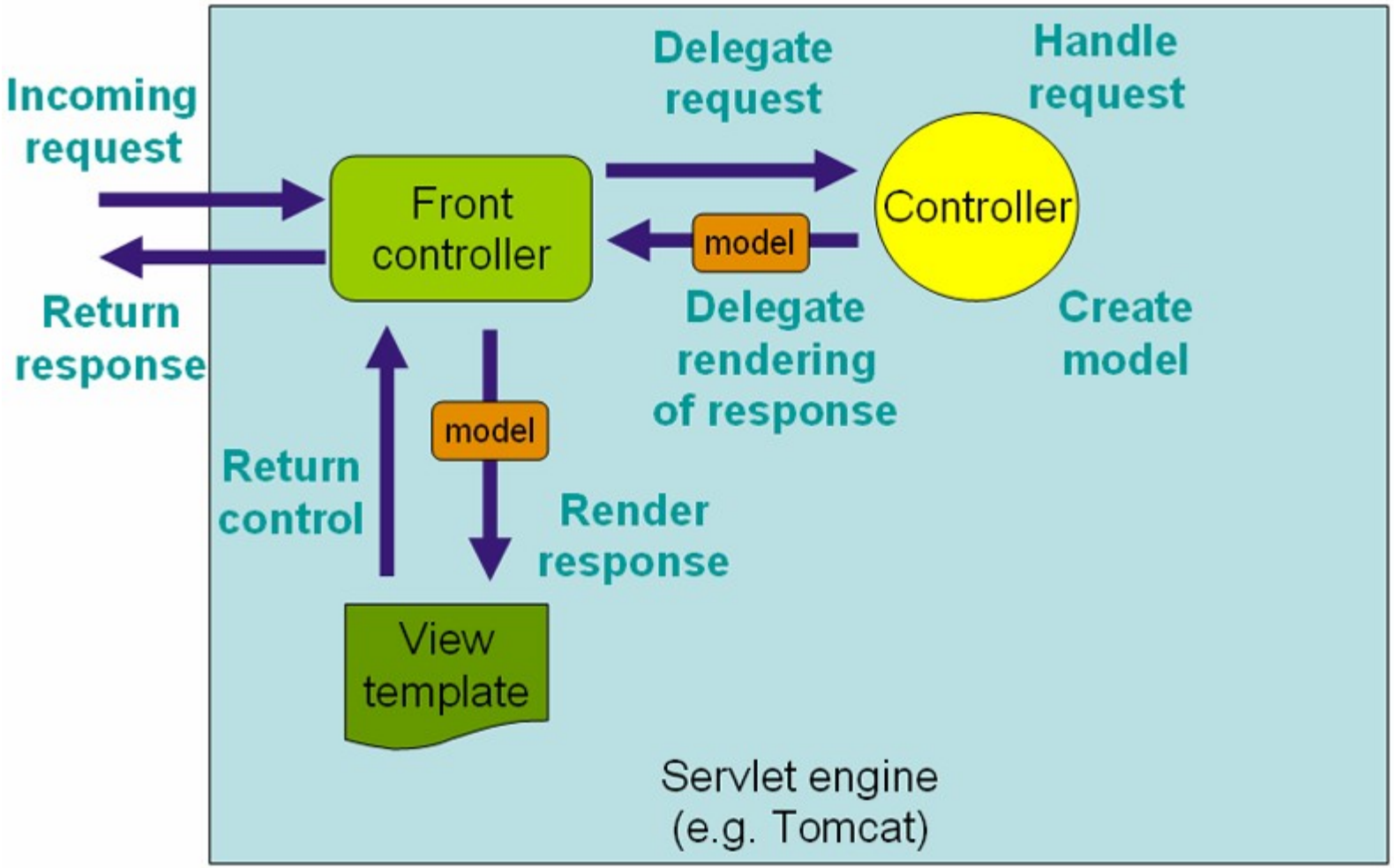
Spring MVC

- Características de Spring MVC
  - Clara separación de roles:
    - controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver
  - Potente y sencilla configuración del framework y de las clases como JavaBeans
  - Adaptable y flexible: define las firmas de métodos del controlador como se necesita, usando alguna anotación para algún escenario
    - @RequestParam, @RequestHeader, @PathVariable...
  - Código de lógica de negocio reusable

Spring MVC

- Características de Spring MVC (2):
  - Validaciones e interpretación de datos personalizable
  - Controlador de mapeos y resolución de vistas personalizables
  - Modelo de transferencia flexible
    - Usa Map nombre/valor

Spring MVC


Incoming request
Delegate request
Handle request
Front controller
Controller
model
Return response
Delegate rendering of response
Create model
model
Return control
Render response
View template
Servlet engine (e.g. Tomcat)

Spring MVC

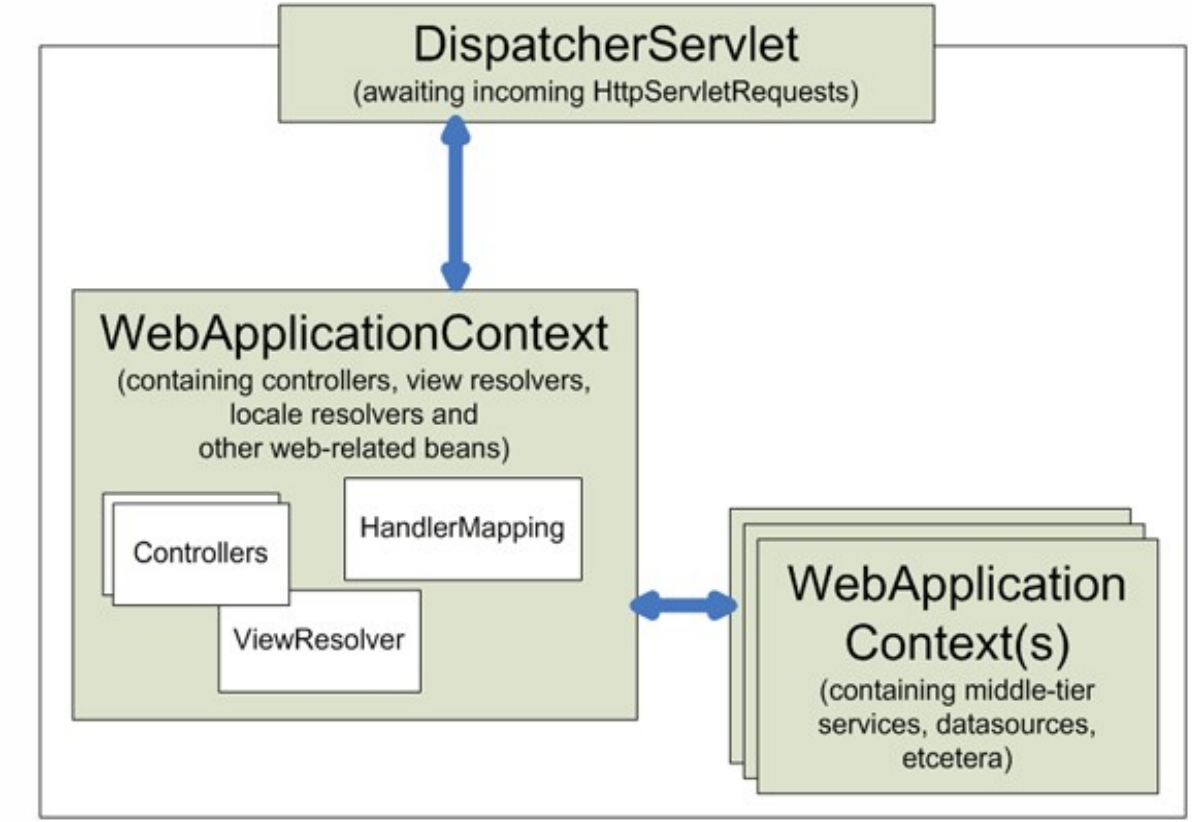
- DispatcherServlet
  - Corresponde al “Front Controller”
  - Es un servlet, hereda de HttpServlet
  - Se debe declarar en el web.xml de la aplicación
  - Se debe asociar las URL que debe manejar en el mismo deployment descriptor de la aplicación

Spring MVC

```
<web-app>

    <servlet>
        <servlet-name>example</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>*.form</url-pattern>
    </servlet-mapping>

</web-app>
```

Spring MVC

- En el framework MVC, cada DispatcherServlet tiene su propio WebApplicationContext



Spring MVC

- Beans Especiales en WebApplicationContext

Bean type	Explanation
controllers	Form the c part of the MVC.
handler mappings	Handle the execution of a list of pre-processors and post-processors and controllers that will be executed if they match certain criteria (for example, a matching URL specified with the controller).
view resolvers	Resolves view names to views.
locale resolver	A locale resolver is a component capable of resolving the locale a client is using, in order to be able to offer internationalized views
Theme resolver	A theme resolver is capable of resolving themes your web application can use, for example, to offer personalized layouts
multipart file resolver	Contains functionality to process file uploads from HTML forms.
handler exception resolvers	Contains functionality to map exceptions to views or implement other more complex exception handling code.

Spring MVC

- Controladores
  - Provee acceso a la aplicación
  - Interpreta los datos que envía el usuario y los transforma en un “modelo”
    - Este modelo es presentado al usuario por la “vista”
  - Modelo de programación con anotaciones
    - Agregado en Spring 2.5
    - @RequestMapping, @RequestParam, @ModelAttribute

Spring MVC

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("helloWorld");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}
```

Spring MVC

- La anotación @Controller indica que esta clase cumple el rol de controlador
  - El DispatcherServlet busca en estas clases por métodos con la anotación @RequestMapping
- Se pueden definir controladores directamente en el contexto del dispatcher (XML)
- Habilitar la auto-detección de estos controladores
  - Agregar un “component-scan” a la configuración

Spring MVC

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="org.springframework.samples.petclinic.web"/>

    // ...

</beans>
```

Spring MVC

- Mapping request con @RequestMapping
  - Se puede usar a nivel de clase o para un método en particular
  - Si se usa a nivel de clase, todos los métodos son relativos a esa URL
  - No es obligatorio definirlo a nivel de clase

Spring MVC

```java
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {

    private final AppointmentBook appointmentBook;

    @Autowired
    public AppointmentsController(AppointmentBook appointmentBook) {
        this.appointmentBook = appointmentBook;
    }

    @RequestMapping(method = RequestMethod.GET)
    public Map<String, Appointment> get() {
        return appointmentBook.getAppointmentsForToday();
    }

    @RequestMapping(value="/{day}", method = RequestMethod.GET)
    public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Mod
        return appointmentBook.getAppointmentsForDay(day);
    }

    @RequestMapping(value="/new", method = RequestMethod.GET)
    public AppointmentForm getNewForm() {
        return new AppointmentForm();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String add(@Valid AppointmentForm appointment, BindingResult result) {
        if (result.hasErrors()) {
            return "appointments/new";
        }
        appointmentBook.addAppointment(appointment);
        return "redirect:/appointments";
    }
}
```

Spring MVC

```java
@Controller
public class ClinicController {

    private final Clinic clinic;

    @Autowired
    public ClinicController(Clinic clinic) {
        this.clinic = clinic;
    }

    @RequestMapping("/")
    public void welcomeHandler() {
    }

    @RequestMapping("/vets")
    public ModelMap vetsHandler() {
        return new ModelMap(this.clinic.getVets());
    }

}
```