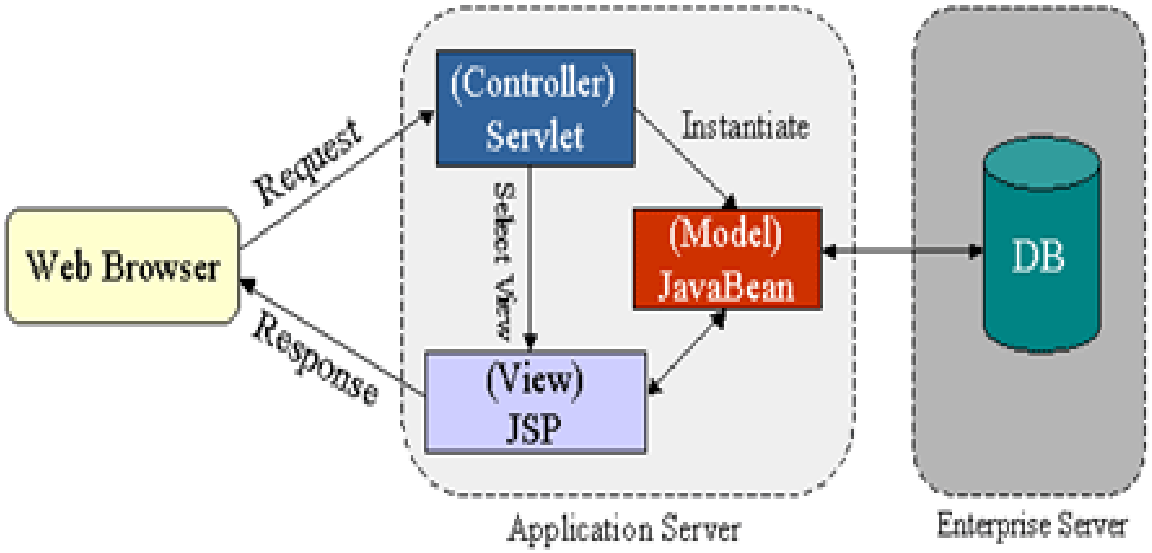
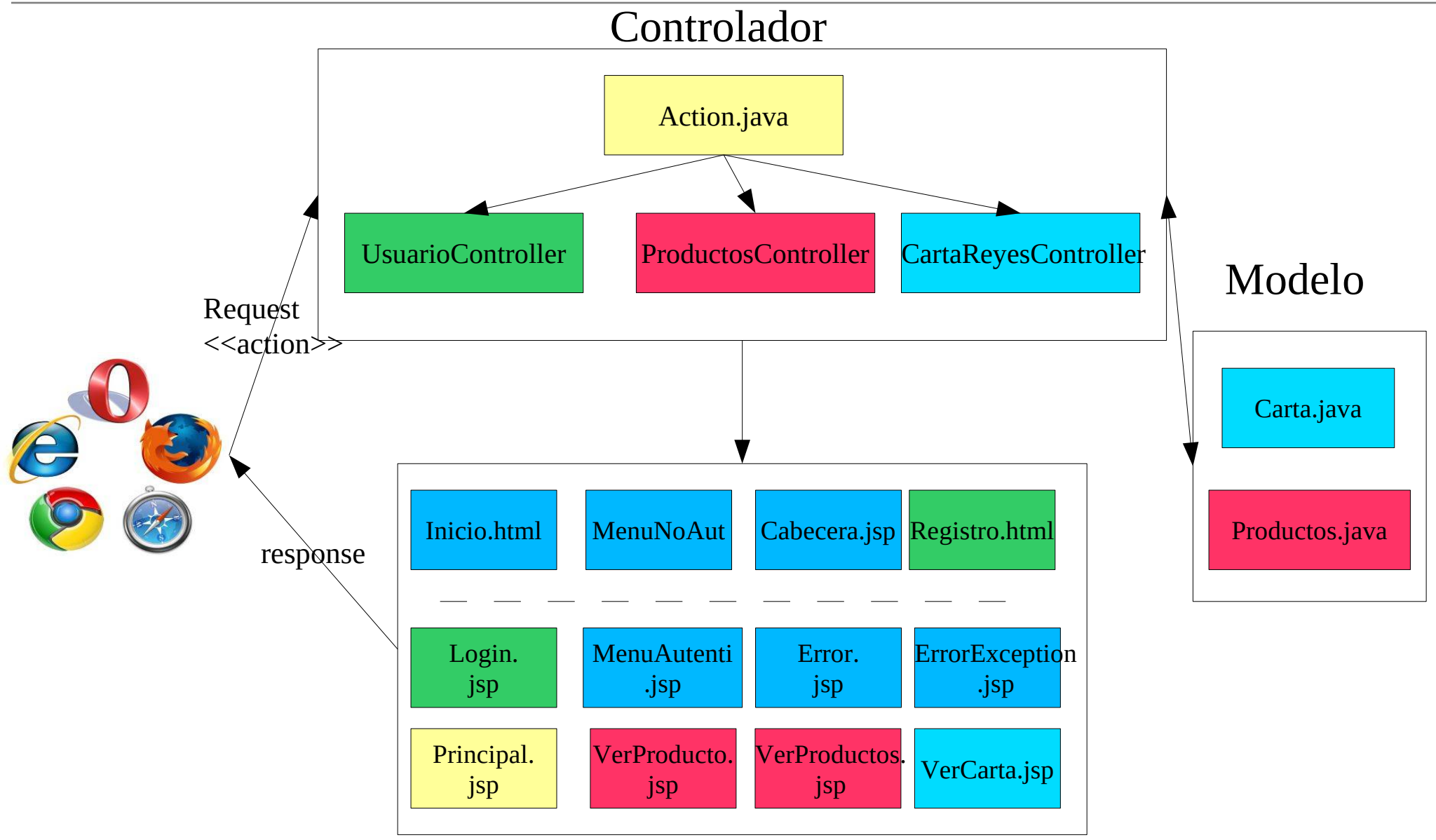
Práctica 10
Servlet+JSP+BD + Servicios

Desarrollo de la práctica

- Modelo Vista Controlador
  - Modelo: se encarga de representar los objetos del dominio y contener la lógica de negocio (Beans)
  - Vista: se encarga de presentar los datos al usuario (JSP)
  - Controlador: acepta las acciones del usuario, las convierte en manipulaciones sobre el modelo y selecciona vistas adecuadas para representar la información relacionada con las acciones del usuario (Servlet)



Desarrollo de la práctica. MVC



2. Desarrollo basado en Servicios

INCONVENIENTE

- En la implementación anterior se tenía:

```
public void doLogin(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
  String usuario = req.getParameter("user");
  String passRecibido = req.getParameter("password");
  String passBD = null;

  if (s.getAttribute("id")!=null){
      ResultSet resultados = null;
      try {
              synchronized (sentencia) {
              resultados = sentencia
              .executeQuery("SELECT * FROM usuarios where
              username='"+ usuario + "'");
    }
    if (rs.next()){
 ….
```

Mezcla de: capa lógica + capa de datos + capa de control

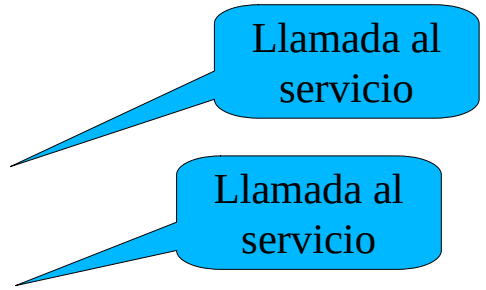
2. Desarrollo basado en Servicios

- **INCONVENIENTE**
  - Mezclando: capa lógica + capa de datos + capa de control
  - Tenemos clases denominadas UsuarioController, ProductosController y Cartas Controller donde se mezcla acceso a la BD, con lógica de la aplicación, etc
- Modelo-vista-controlador (MVC) con capa de servicio (4 capas)
  - **Capa de presentación o vista**
    - ✓ Compuesto por las páginas .jsp
  - **Capa de control: Los distintos servlets.**
    - ✓ Se encargan de capturar las peticiones y reenviar las respuesta
  - **Capa de modelo:**
    - ✓ Se encarga de representar los datos obtenidos
  - **Capa de servicio o lógica**
    - ✓ Se encarga de recuperar y manipular los datos

2. Desarrollo basado en Servicios

- UsuarioController (1/2)
  - UsuariosController: Únicamente debe preocuparse de realizar los controles de sesión y a continuación utilizar los servicios.

```
public void doLogin(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
  String username = req.getParameter("user");
  String passRecibido = req.getParameter("password");
  try {
    UsuarioService userService = new UsuarioServiceBD();
    boolean result = userService.authenticate(username, passRecibido);
    if (result == true) {
              Usuario user = userService.getUserByUsername(username);
             HttpSession session = req.getSession(true);
             session.setAttribute("id", String.valueOf(user.getId()));
             session.setAttribute("nombre", user.getNombre());
             session.setAttribute("username", user.getUsername());
             req.setAttribute("mensaje", "Autenticacion correcta");
             req.getRequestDispatcher("WEB-INF/Principal.jsp").forward(req, res);
   } else { res.sendRedirect("Error.jsp?error=Error en Password");       return;  }
  } catch (Exception e2) {res.sendRedirect("Error.jsp?error=ERROR:Fallo en SQL");  }
}
```



2. Desarrollo basado en Servicios

- UsuarioController (2/2)
  - UsuariosController: Únicamente debe preocuparse de realizar los controles de sesión y a continuación utilizar los servicios.

```
public void darAlta(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {

  String n = req.getParameter("nombre");
  String apel = req.getParameter("apellidos");
  String email = req.getParameter("email");
  String username = req.getParameter("username");
  String pass = req.getParameter("password");
  Usuario usuario1 = new Usuario(-1, n, apel, email, username, pass);
  try {
     UsuarioService userService = new UsuarioServiceBD();
     boolean result = userService.register(usuario1);
     if (result) {
              req.setAttribute("mensaje", "Dado de alta correctamente");
              req.getRequestDispatcher("Login.jsp").forward(req, res);
             return;
   } else {res.sendRedirect("Error.jsp?error=El usuario ya existe");return;}
   } catch (Exception e2) { res.sendRedirect("Error.jsp?error=ERROR:Fallo en SQL");}
}
```

Servicio basado en BD

Llamada al servicio

2. Desarrollo basado en Servicios

- UsuarioServices.java (1/2)
  - Definido como interface, de este modo se pueden tener distintas implementaciones
  - Operaciones (**CRUD**)
    - ✓ Creación (Create)
    - ✓ Lectura (Read)
    - ✓ Actualización (Update)
    - ✓ Borrado (Delete)
    - ✓ Listados (getters)

```
public interface UsuarioService {
  public boolean register(Usuario user);          //Create
  public boolean delete(String userId);           //Delete
  public boolean authenticate(String username, String password); //Read
  public Usuario getUserByUsername(String username);    //Read
  public Usuario getUserByUserId(String userId);        //Read
  public List<Usuario> getListOfUsers();                //gettter
}
```

2. Desarrollo basado en Servicios

- UsuarioServicesBD.java (1/2)
  - Implementación basada en BD

```
public boolean authenticate(String username, String password) {
  Usuario user = getUserByUsername(username);
   if (user != null && user.getUsername().equals(username)&&
        user.getPassword().equals(password)) { return true;
   } else {return false;}
}

public Usuario getUserByUsername(String username) {
  Usuario user = null;
  ResultSet resultados = null;
  Statement sentencia= ConexionUtil.openStatement();
   synchronized (sentencia) {// Cogemos todos los datos de los usuarios
      resultados = sentencia.executeQuery("SELECT * FROM usuarios where
                              username='"+ username + "'");
    }
    if (resultados.next() == false) {return null;
    } else {
              user = new Usuario(……);return user;
    }
return user;
}
```

UsuarioServicesBD.java (2/2)

Implementación basada en BD

```
public Usuario getUserByUserId(String userId) {
  Statement sentencia= ConexionUtil.openStatement();
  ResultSet resultados = sentencia .executeQuery
              ("SELECT * FROM usuarios where userId='" + userId + "'");
  if (resultados.next() == false) {return null;
  } else {
     Usuario user = new Usuario(Integer.parseInt(resultados.getString("id")),
              resultados.getString("Nombre"),resultados.getString("Apellidos"),
              resultados.getString("email"), resultados.getString("username"),
              resultados.getString("password"));
     return user;
  }
  } catch (Exception e) {...}
  return user;
}
public List<Usuario> getListOfUsers() {
   List<Usuario> l = new ArrayList<Usuario>();
   Statement sentencia= ConexionUtil.openStatement();
   ResultSet resultados = sentencia.executeQuery("SELECT * FROM usuarios");
   while (resultados.next()) {
              Usuario user = new Usuario(<<Datos recuperados>>);
               l.add(user);
   }
   } catch (Exception e) {….}
   return l;
}
```

2. Desarrollo basado en Servicios

- UsuarioServicesMemory.java (1/2)
  - Implementación basada en Memoria

```
public class UsuarioServiceMemory implements UsuarioService {
  private static HashMap<Integer, Usuario> lUsuario = null;
  private void buildMap() {
      if (lUsuario == null) {
        lUsuario= new HashMap<Integer, Usuario>();
      }
  }
  public boolean authenticate(String username, String password) {
    Usuario user = getUserByUsername(username);
    if (user != null && user.getUsername().equals(username)&&
        user.getPassword().equals(password)) {return true;
     } else { return false;}
  }
  public Usuario getUserByUsername(String username) {
   if (lUsuario == null) buildMap();
   synchronized (lUsuario) {
     Iterator it = lUsuario.entrySet().iterator();
     while (it.hasNext()) {
        Map.Entry e = (Map.Entry) it.next();
        Usuario user = (Usuario) e.getValue();
        if (user.getUsername().equals(username))
                return user;
        }
     }
     return user;
  }
```

Estático.
Por qué???

Simulamos
Patrón Singleton

Patrón Singleton

2. Desarrollo basado en Servicios

- UsuarioServicesMemory.java (2/2)
  - Implementación basada en memoria

```
public Usuario getUserByUserId(String userId) {
  if (lUsuario == null) buildMap();
  return  lUsuario.get(Integer.parseInt(userId));
}
public List<Usuario> getListOfUsers() {
  if (lUsuario == null) buildMap();
   List<Usuario> l = new ArrayList<Usuario>();
   synchronized (lUsuario) {
        Iterator it = lUsuario.entrySet().iterator();
        while (it.hasNext()) {
              Map.Entry e = (Map.Entry) it.next();
              l.add( (Usuario) e.getValue());
        }
  }
  return l;
}
public boolean register(Usuario user) {
    if (lUsuario == null) buildMap();
    if (getUserByUsername(user.getUsername()) != null)  return false;
   synchronized (lUsuario) {
              user.setId(lUsuario.size() + 1);
              lUsuario.put(lUsuario.size() + 1, user);
  }
return true;
}
```

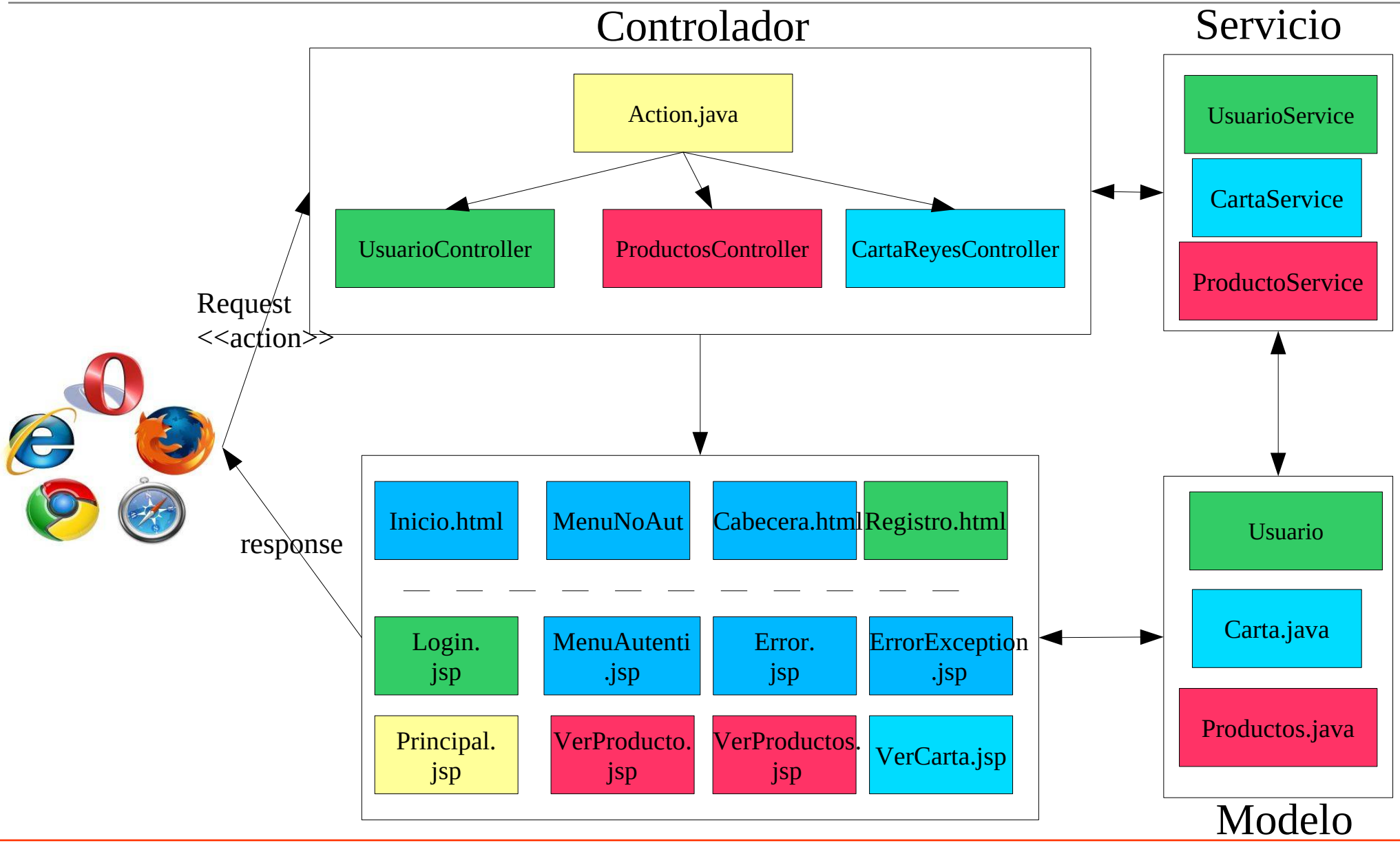
Patrón Singleton

Patrón Singleton

2. Desarrollo basado en Servicios

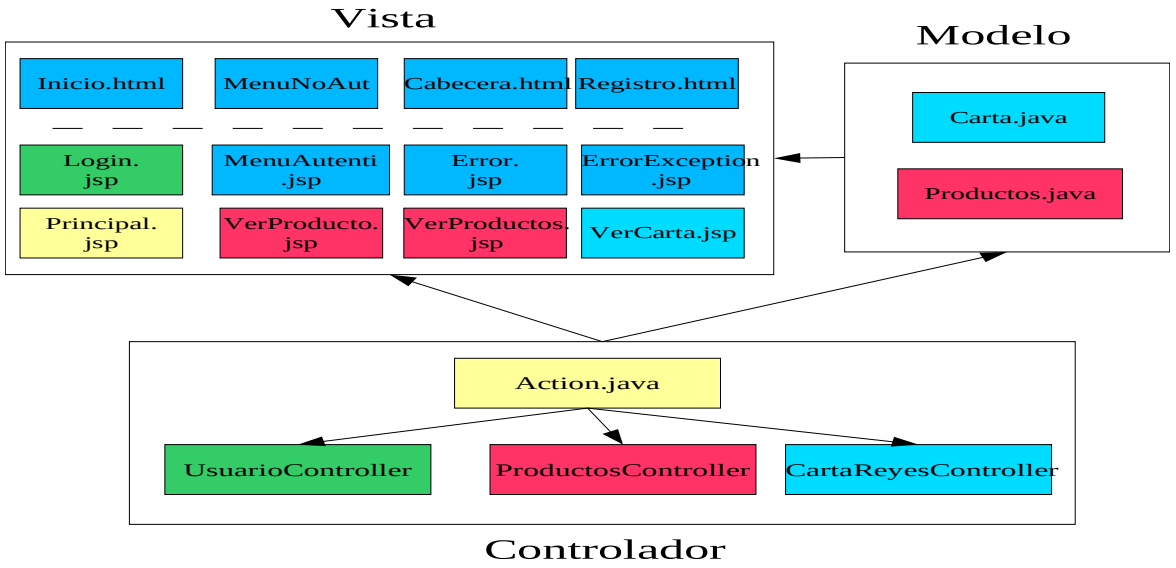
- Nótese que se podría realizar todo el proyecto también basado en memoria
  - La base de datos de productos se cargaría de un fichero csv cuando se inicializa el servlet
  - Para simular el concepto de Carta
    - ✓ Se crearía una clase ItemData (idproducto, cantidad)
    - ✓ Se crearía una clase CartaData (lista de ItemData)
    - ✓ En CartaServicios se tendría un mapa estático definido como patrón Singleton constituido por la clave del usuario y su lista de productos

Desarrollo de la práctica. MVC



Desarrollo de la práctica. Un paso más alla

MVC
Spring, Struts,
JSTL, JSF, JQuery



Clase Java, Spring

Lógica (Servicio)

DAO, Hibernate,
OpenJPA, IBATIS

Capa de Datos

Desarrollo de la práctica

- Usando la implementación proporcionada de CartaReyesMagos se pide añadir la siguiente operabilidad:
  - CartaServicesBD y VerCarta.jsp
  - ProductosServicesBD y VerProductos.jsp
  - AñadirProducto
    - ✓ Se debe añadir una nueva entrada en el menú izquierdo (action=anadirProductos)
    - ✓ Se debe añadir una nueva action en ProductosController
    - ✓ Debe recibir los parámetros por parámetros e insertar en la base de datos (la imagen se copiará manualmente en img)
  - InfoUsuario
    - ✓ Proporcione información del usuario
    - ✓ Se debe añadir un nueva action en UsuarioController
    - ✓ Coger la información de la BD y reenviar a un JSP que lo muestre