Web Application in Java Servlet Page

Corso di Programmazione III
Prof. Angelo Ciaramella

Tutor
Domenico Capuano

Web Application

Le Web Application, sono dei software che risiedono su un server e sono raggiungibili e utilizzabili dai diversi utenti tramite un comune browser web.

Consentono una completa integrazione con gli applicativi aziendali preesistenti, fungendo, in alcuni casi, da contenitore e connettore tra questi.

Un'applicazione web può essere creata per una rete internet (raggiungibile da qualsiasi macchina avente un collegamento al web) o per una intranet (raggiungibile esclusivamente dai computer collegati alla rete aziendale).

Cosa cambia?

- Niente, dal punto di vista tecnologico
- Requisiti di Scalabilità, Sicurezza, etc.

Web Application

Principali Vantaggi

- **Facilità di installazione, distribuzione e aggiornamento**: tutte queste operazioni vengono svolte un'unica volta esclusivamente sul server che ospita l'applicazione e non sulle macchine di tutti coloro che la utilizzeranno

- **Accesso semplificato**: avvenendo tramite un qualsiasi browser, per l'accesso non sono richiesti hardware o software particolari

- **Visibilità**: le possibilità di utilizzo del web ampliano notevolmente la base di utenti a cui è possibile rivolgersi (es. social network, e-commerce, etc.)

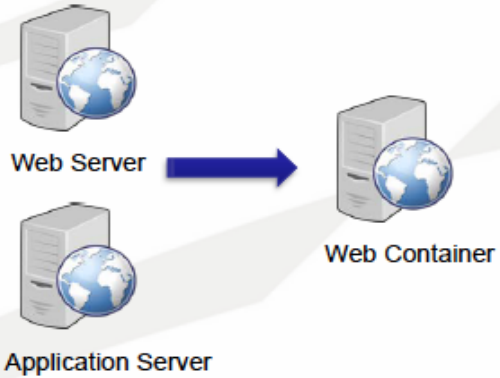
Web Application

Ambiti

Le applicazioni web costituiscono quindi un ottimo supporto per la gestione dei dati aziendali e possono spaziare negli ambiti più disparati, ad esempio:

- CMS e Portali Web;
- e-Commerce;
- Home Banking;
- Gestione dei processi commerciali e della forza vendite;
- ERP, CRM;
- Social Network;
- e Tanti altri ……

Web Application
(Server)



Il Web Server si occupa di predisporre e restiture le pagine web al client.
Può comprendere il supporto a linguaggi di scripting per consentire la generazione di pagine dinamiche

L' Applicaton Server si occupa di gestire la logica applicativa e interfacciarsi con altri moduli quali RDBMS e/o Applicazioni Legacy

Talvolta i ruoli sono fusi in unico oggetto, definito Web Container, nel caso Java ne è un esempio il Servlet Container Tomcat

Web Application
(Client)

Browser

Software Client in grado di tradurre pagine web descritte mediante HTML e CSS in oggetti grafici

Integra Engine Javascript per l'esecuzione di codice lato client, aumenta le capacità di interattività offerte da una normale pagina web

Il codice lato server può produrre pagine web costituite da un mix di HTML, CSS e Javascript.

Apache Tomcat

Può essere visto come l'insieme di tre componenti:

- **Catalina**: Il Servlet Container, implementa le specifiche Java per le Servlet e JSP
- **Connector:**
  - **HTTP**: consente a Tomcat di servire richieste mediante il protocollo http
  - **AJP:** consente di predisporre un Web Server che funga da Proxy per Tomcat con prestazioni migliori rispetto ad un Proxy HTTP standard
- **Jasper** : si occupa di compilare le pagine JSP e trasformarle in Servlet che saranno gestite da Catalina

Java Servlet / Java Server Pages (JSP)

- Sviluppato da Java Community Process
- Parte dello standard Java EE
- OO, platform-independent, efficiente, scalabile,…
- Consente la separazione tra il livello di presentazione (interfaccia) e la logica applicativa

Java Servlet

Vantaggi

Condividono tutti i vantaggi del sw scritto in Java:

- Portabilità, OO, riuso, supporto di librerie già esistenti, efficienza, sicurezza, etc.

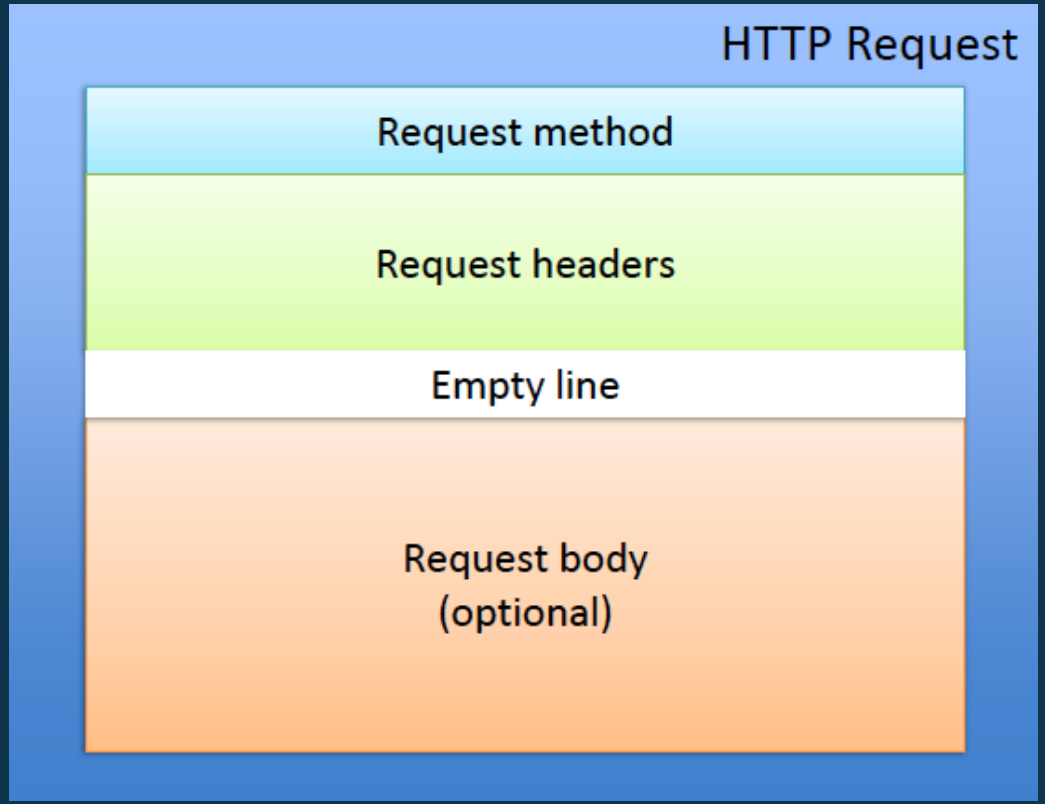
Si basano su una ben consolidata API specifica per il protocollo HTTP:

- processing delle richieste
- generazione delle risposte
- gestione delle sessioni e dei cookies

Cos’è una Java Servlet?

- È una normale classe Java che consente l’interazione richiesta/risposta con un’applicazione secondo il modello client/server
- Le **Servlets** sono progettate per gestire qualunque tipo di protocollo richiesta/risposta
- Tipicamente vengono usate per l’implementazione di applicazioni che interagiscono secondo il protocollo HTTP (richiesta/risposta via Web)

HTTP: Richieste Client

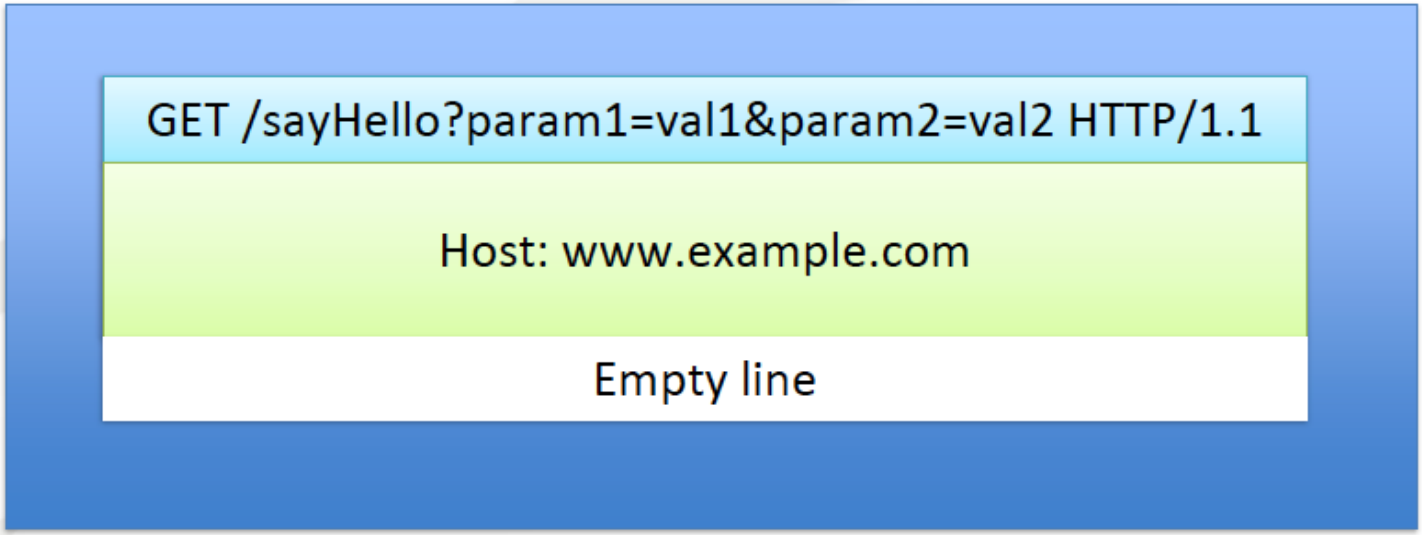


HTTP: Metodi di richiesta

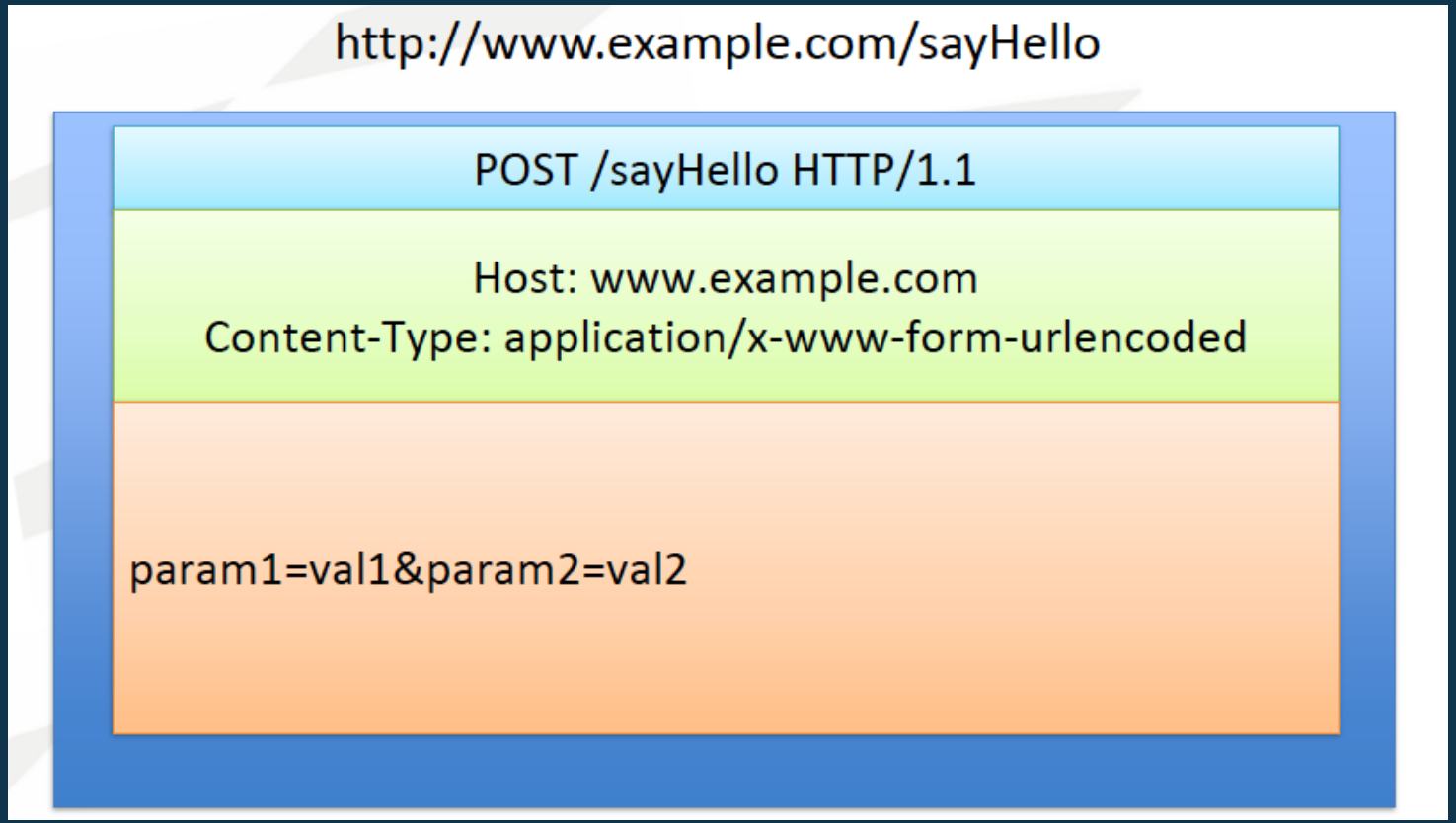
- HTTP definisce i seguenti metodi di richiesta:
- GET, POST, PUT, DELETE, HEAD
- GET e POST sono le più utilizzate
- Eventuali parametri della richiesta possono essere specificati:
  - nella query string (URL) se si usa il metodo GET http://example.com/sayHello?param1=val1&param2=val2
  - nel corpo della richiesta se si usa il metodo POST in combinazione con form HTML

HTTP GET

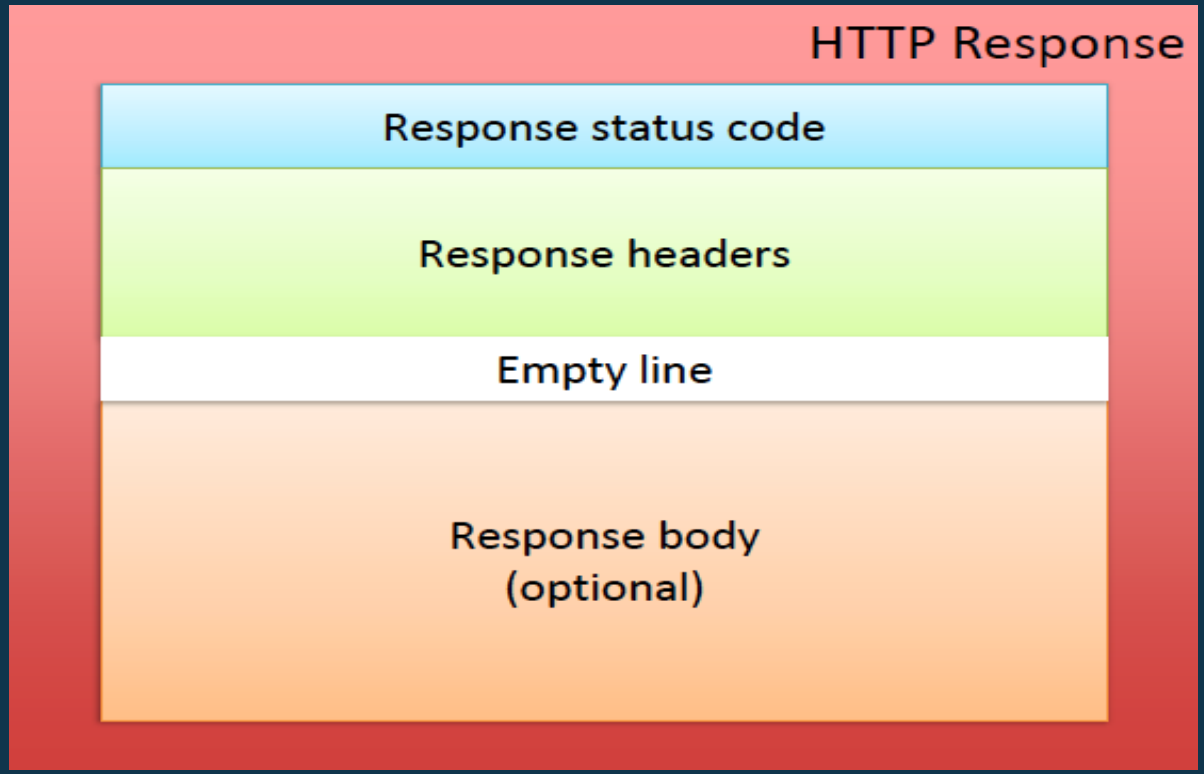
http://www.example.com/sayHello?param1=val1&param2=val2



HTTP POST


http://www.example.com/sayHello
POST /sayHello HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
param1=val1&param2=val2

HTTP: Risposta Server



HTTP: Codici di risposta

- 2xx = Successo
  - 200 OK → la richiesta ha avuto successo
    - GET: l'entità corrispondente alla richiesta viene inviata nella risposta
    - POST: l'entità corrispondente al risultato dell'azione richiesta viene inviata nella risposta
- 3xx = Redirezione
- 4xx = Errore Client
  - 401 Bad Request, 404 Not Found, etc.
- 5xx = Errore Server
  - 500 Internal Server Error, 503 Service Unavailable, etc.

I Metodi: doGet e doPost

Per rispondere a richieste HTTP GET e POST occorre implementare o fare l'override del metodo dei metodi doGet e doPost

doGet

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    out.print("Hello ");
    out.println(super.getInitParameter("name"));
}
```

doPost

```
/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
    doGet(request, response);
}
```

Gestione della Richiesta

Le Servlet possono gestire le richieste

- Direttamente: risposta diretta al client
- Indirettamente:
  - Lato Server:
    - tramite l'inclusione lato server del contenuto di un'altra risorsa (**include**)
    - tramite l'inoltro lato server della richiesta ad un'altra risorsa (**forward**)
  - Lato Client: tramite l'inoltro della richiesta ad un'altra risorsa previa comunicazione con il client (redirect)

Gestione Indiretta della Richiesta

Lato Server

- Lo **stesso oggetto** che rappresenta la richiesta viene trasferito ad un'altra risorsa server (ad es. una **Servlet**)
- Il **forwarding** della richiesta è interamente eseguito **lato server** dal **container** ed è **trasparente** e nascosto al **client** (ecco perché l'URL nel browser non cambia!)
- Per invocare un'altra risorsa la **Servlet** deve ottenere un oggetto **RequestDispatcher** con il metodo della richiesta

getRequestDispatcher(String url)

Include vs. Forward

- Con il metodo **include**, RequestDispatcher aggiunge il contenuto della risorsa alla Servlet “chiamante”
- Con il metodo **forward**, RequestDispatcher chiude il buffer di output della Servlet chiamante e passa totalmente la gestione della richiesta alla risorsa chiamata

Gestione Indiretta della Richiesta

Lato Client

- Alternativa al forwarding gestito lato server
- Il forwarding della richiesta ad un'altra risorsa passa prima nuovamente al client
- Il container istruisce il browser sulla nuova richiesta da eseguire tramite header HTTP
- Il client inizia una nuova comunicazione con il server all'URL specificato (l'URL stavolta cambia nel browser!)
- Per invocare un'altra risorsa la Servlet deve chiamare il seguente metodo sull'oggetto risposta

sendRedirect(String url)

Forward vs. Redirect

Forward (lato server):

- la richiesta originale del client non viene modificata e passa da una risorsa all'altra
- Possibile inoltrare la richiesta solo a componenti server interni allo stesso dominio dell'applicazione

Redirect (lato client):

- la richiesta originale viene "persa" e il server indica al browser di effettuare una nuova richiesta verso la risorsa alla quale vogliamo inoltrare il controllo
- Possibile inoltrare la richiesta a risorse esterne al dominio dell'applicazione

Preferibile perché più "leggero" se la richiesta può essere interamente gestita dallo stesso dominio

Inevitabile quando è necessario trasferire il controllo a componenti che risiedono su altri domini (es. broker di pagamento on line)

Passaggio dei Dati: Request Scope

- Necessario quando la gestione della singola richiesta è affidata a più Servlets
- Le Servlets condividono i dati della richiesta nella cosiddetta **request scope storage**
- **ServletRequest** definisce i seguenti metodi per i dati condivisi nel request scope:

void setAttribute(String k, Object v)

Object getAttribute(String k)

void removeAttribute(String k)

I dati sono disponibili per l'intera durata della richiesta e vengono eliminati una volta che questa è servita

Passaggio dei Dati: Session Scope

- HTTP è un protocollo **stateless**
- Spesso necessario **mantenere informazioni di stato** del cliente attraverso più richieste HTTP (**session**)
- Java EE servers offrono questa possibilità in modo semplice per lo sviluppatore dell'applicazione

- La prima volta che se ne fa richiesta, il server crea un oggetto "**session**" e gli assegna un ID univoco che invia la client.
- Ad ogni richiesta successiva, il client invia **sessionID** sottoforma di cookie o di parametro

Http Session

- Per accedere all'oggetto che identifica la sessione utente, invocare il metodo **getSession()** della classe **HttpServletRequest**
- Per salvare dati in sessione invocare il metodo **session.setAttribute(key, value)**
- Per recuperare/rimuovere un dato dalla sessione invocare

session.getAttribute(key)

session.removeAttribute(key)

- Le chiavi sono di tipo String.
- Gli oggetti HttpSession sono distrutti al momento dell'invocazione di invalidate()
- Le sessioni che non sono utilizzate oltre un certo **timeout** vengono automaticamente invalidate

JSP (Java Server Pages)

- E' una tecnologia che permette di creare pagine dinamiche lato server.
- Permette di separare la sezione di gestione delle operazioni e di produzione dei contenuti, da quello di visualizzazione vera e propria.
- Normalmente una pagina JSP ha una struttura molto simile a quella di un documento HTML.
- Essa si compone di:
    - Porzioni di codice HTML, JavaScript, CSS ecc... , comunemente definiti blocchi di codice statico.
    - Porzioni di codice Java definito codice dinamico

Di fatto una pagina JSP è una Servlet!

JSP: Compilazione

La compilazione da JSP a Servlet avviene in 2 passi:

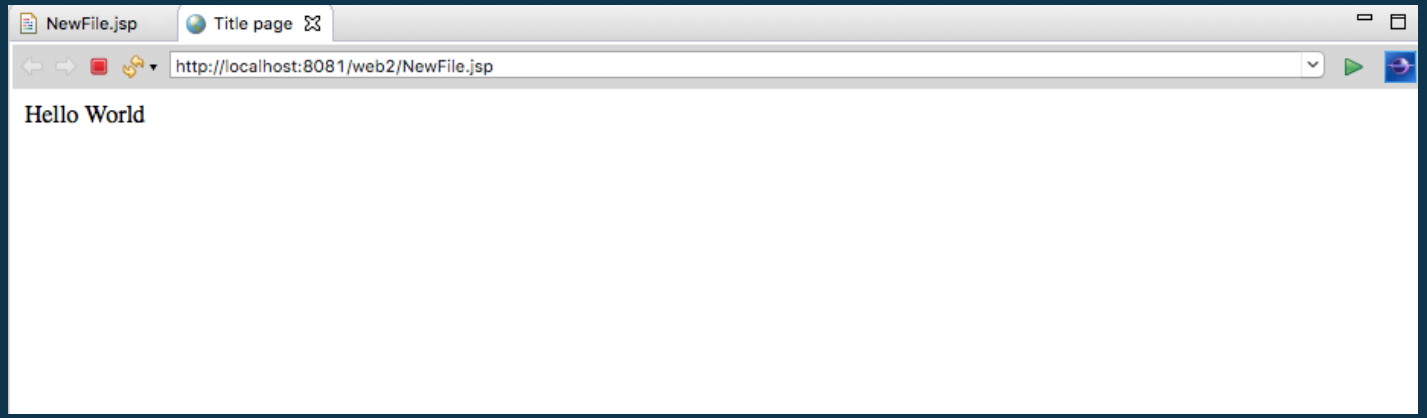
1. Il file .jsp viene compilato in un file .java dal compilatore del container
2. Il file .java generato dal compilatore viene a sua volta compilato in un file .class dal compilatore Java standard (javac)

   Ecco perché il Servlet container necessita dell'intero JDK e non della sola piattaforma JRE

La conversione .jsp → .java → .class avviene soltanto una volta a fronte di una modifica del file .jsp

JSP: Esempio

```
1
2  <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6      <meta charset="UTF-8">
7      <title>Title page</title>
8  </head>
9  <body>
10     Hello World
11 </body>
12 </html>
```



Componenti di una JSP

Una pagina JSP può essere composta da:

- Direttive
- Script
- Oggetti impliciti
- Azioni

Direttive

Non producono output, ma forniscono informazioni globali su un'intera pagina JSP.

La sintassi delle direttive è la seguente

```
<%@ direttiva attributo="valore" %>
```

Ad esempio volendo indicare che il linguaggio da utilizzare nelle Pagine JSP deve essere Java, è possibile utilizzare la riga di codice seguente:

```
<%@ page language="java" %>
```

La Direttiva **Include** serve per inserire testo e codice all'interno di una Pagina JSP al momento della sua traduzione.

La sua sintassi è la sequente:

```
<%@ include file="<..url..>" %>
```

Script JSP

Gli script JSP consentono di includere porzioni di codice direttamente all'interno di pagine HTML.

Esistono tre elementi coinvolti negli script JSP, ciascuno dei quali ha una posizione ben definita all'interno della Servlet generato:

- Scriptlet
- Espressioni
- Dichiarazioni

Scriptlets

```
<% String name = request.getParameter("name");
   if (name==null || name.trim().equals("")) {
 %>
      <form action="">
       <h2>Quale il tuo nome?</h2><br/>
       Inseriscilo qui:
       <input type="text" name="name"/>
       <input type="submit" value="Invia"/>
      </form>
  <% }else{ %>
      <h1>Ciao!   <%=name%> </h1>
  <% } %>
```

Expressions

```
<% String name = request.getParameter("name");
   if (name==null || name.trim().equals("")) {
 %>
      <form action="">
       <h2>Quale il tuo nome?</h2><br/>
       Inseriscilo qui:
       <input type="text" name="name"/>
       <input type="submit" value="Invia"/>
      </form>
 <% }else{ %>
      <h1>Ciao!   <%=name%> </h1>
 <% } %>
```

Servlet vs JSP

Sia le **servlet** che le **jsp** possono essere impiegate per la produzione di **contenuti web dinamici**.

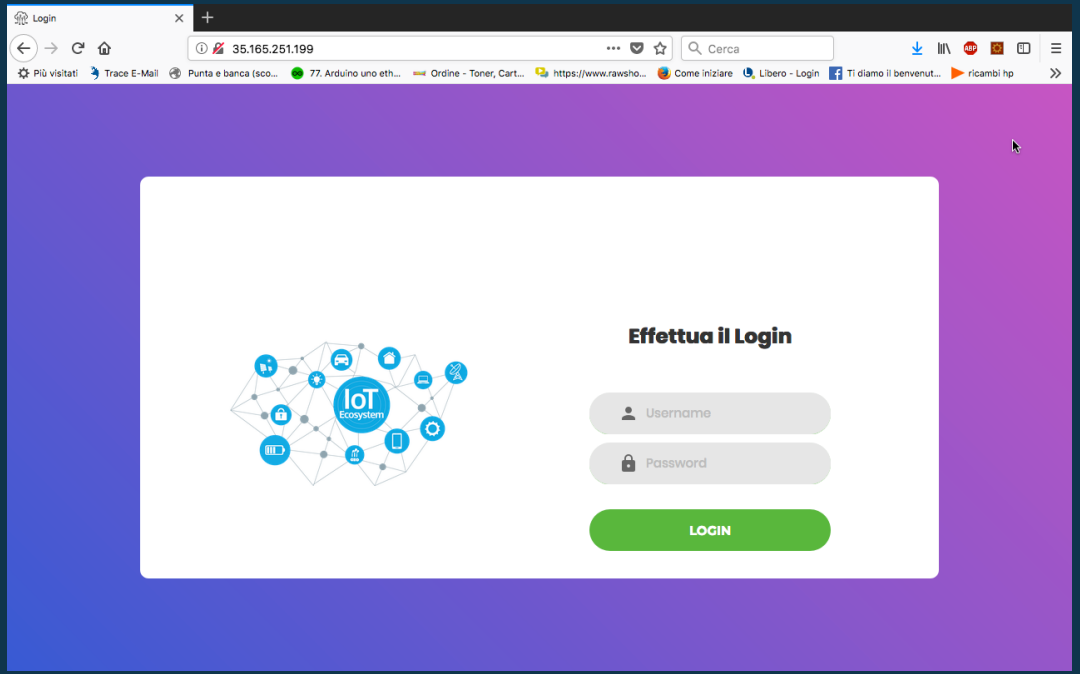
Come e quando conviene utilizzarle?

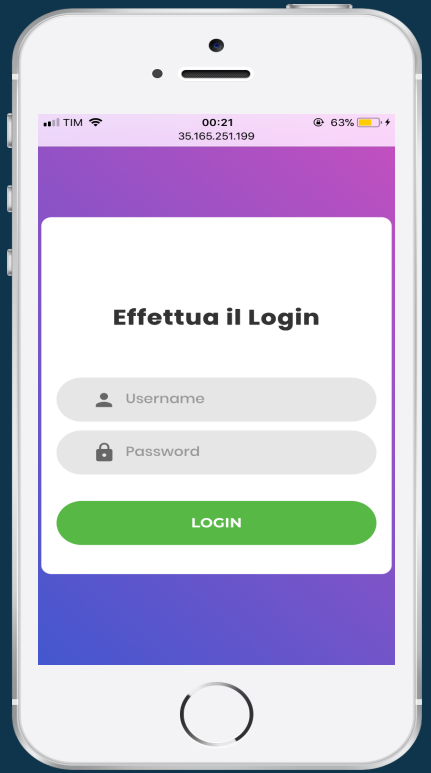
Anche se usate per gli stessi scopi servlet e jsp presentano sostanziali differenze che suggeriscono un uso ottimale di entrambe.

- Le servlets sono classi Java per cui andrebbero usate per le elaborazioni richieste da un'applicazione.
- Le JSP sono prevalentemente formate da codice HTML per cui ha senso utilizzarle per disegnare le pagine web dinamiche.

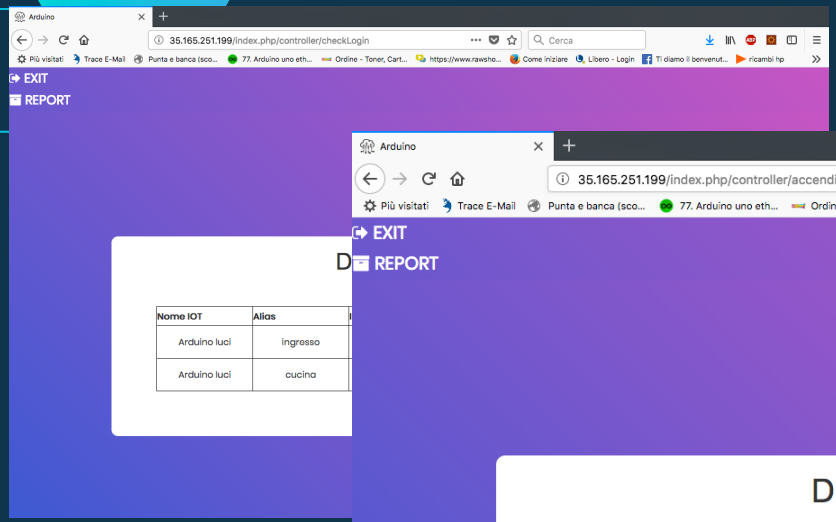
Web app: Example

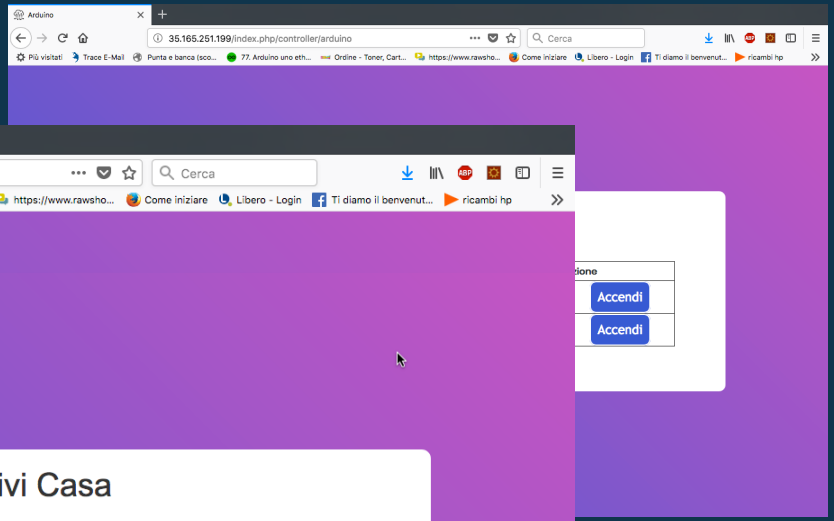
The bootstrap framework was used to make the theme responsive and facilitate the use of the smartphone app

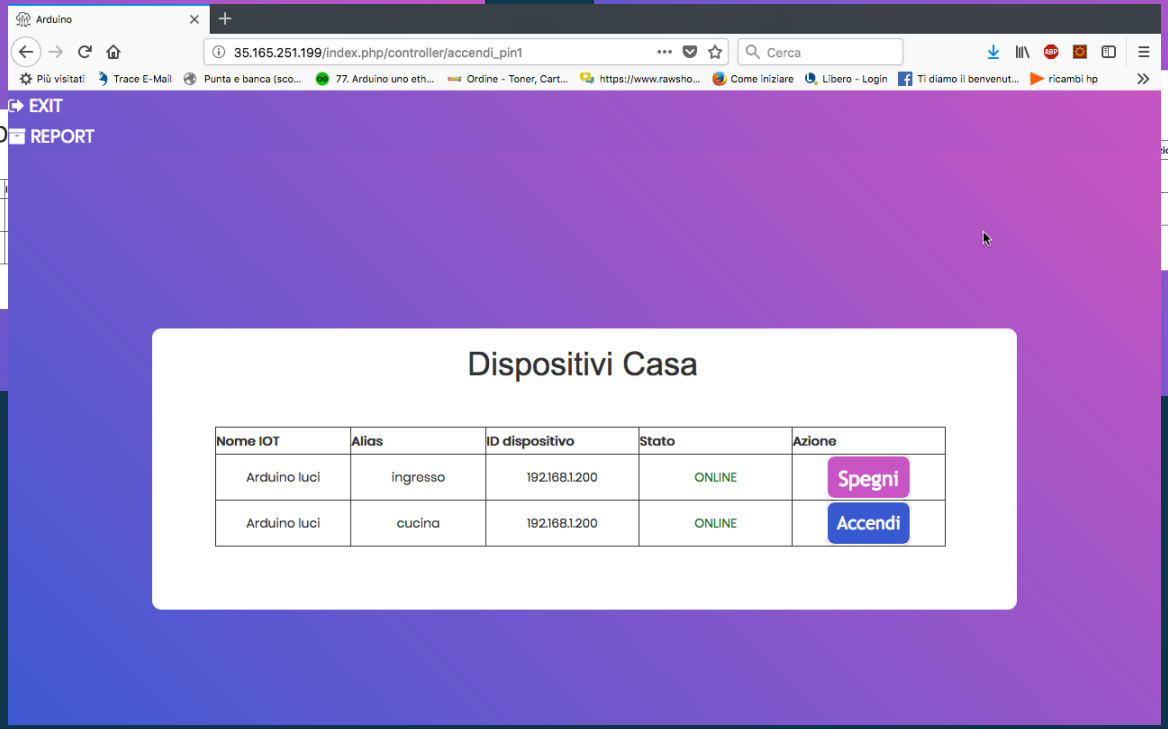




Home page system

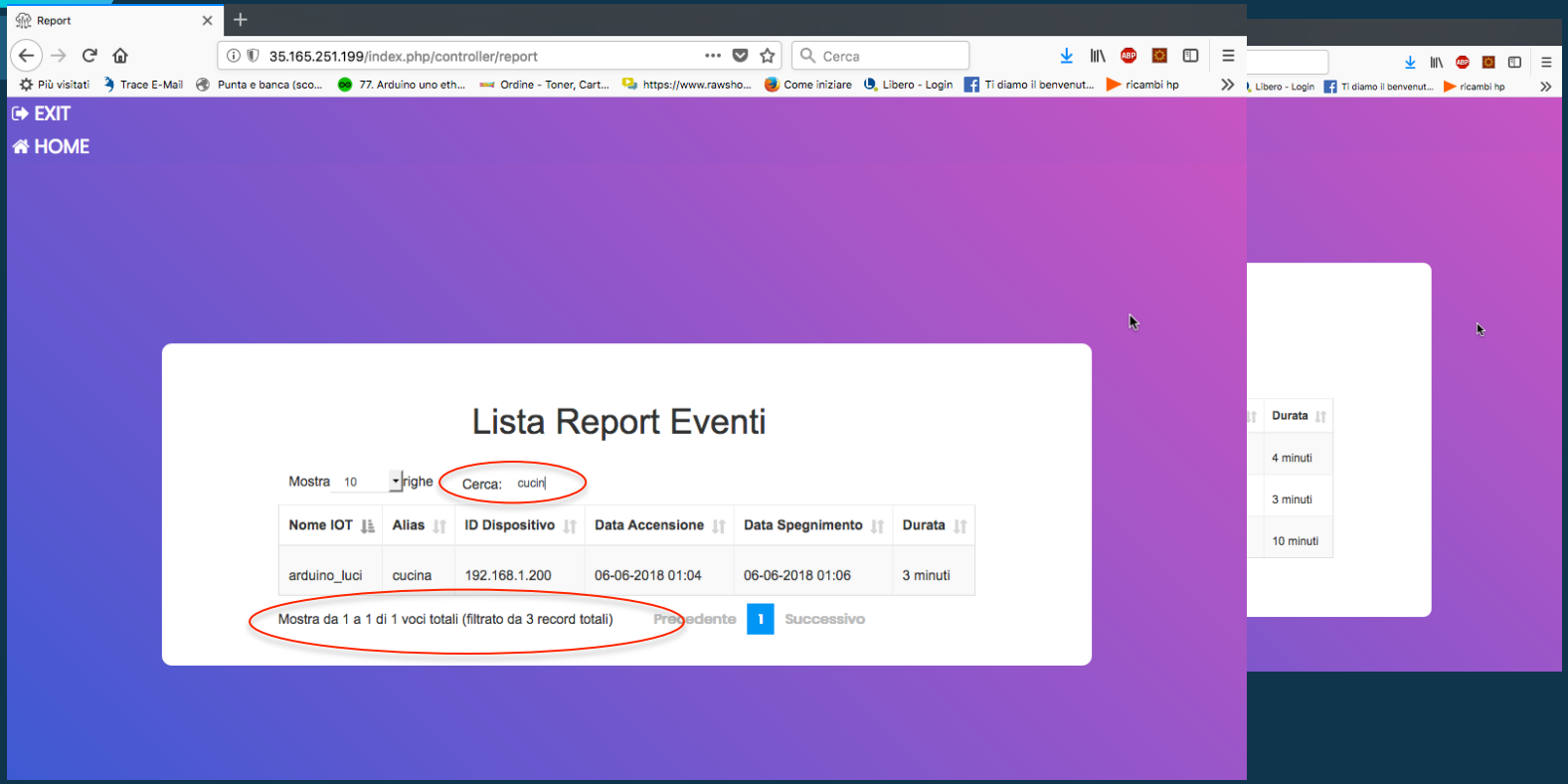






List Report

Search events from the list of all actions performed on the various devices



Video TEST

Thanks!