Eserciziario Web Application

1. Crea una servlet che conta i visitatori:
   il numero di visitatori è riportato sulla pagina html restituita dal client.
   inserire un tasto reset per azzerare il contatore. (suggerimento: creare un form e dare un nome al pulsante submit, tipo `name='reset'`, oppure inserire un input con `type='hidden'`, `name='reset'` ed eventualmente con un valore preassegnato:

   ```
   response.getWriter().println("<form>"
         +"<input type=\"hidden\" name=\"reset\" value=\"ok\">"
         + "<input type=\"submit\" value=\"Reset\"></form>");
   ```

   Utilizzare il metodo `request.getParameter("nome_parametro")` per ricevere il valore del parametro inviato attraverso il `form`). Se il parametro non è presente nella richiesta del client, il metodo restituisce il valore `null`
   Per evitare che il contatore si resetti ogni volta che si fa un refresh della pagina dopo aver premuto il pulsante “reset”, “ripulire” l’url dalla query string mediante una redirect:

   ```
   response.sendRedirect("HelloServlet");
   ```

   oppure

   ```
   response.sendRedirect("/ProjectName/HelloServlet");
   ```

   Aggiungere alla pagina web anche la lista dei visitatori con i timestamp relativi alle richieste effettuate da ogni client (possibilmente all'interno di paragrafi separati (`<p></p>`) oppure all’interno di una `<table></table>`). L’indirizzo IP e il numero di porta del client si ottengono rispettivamente con i metodi `request.getRemoteAddr()` e `request.getRemotePort()`. Si consiglia di utilizzare un `ArrayList` per memorizzare le informazioni sugli accessi dei client. Il pulsante reset, oltre a resettare il contatore, svuota anche la lista dei visitatori (`list.clear()`).
2. Riscrivere la servlet in modo che la visualizzazione della pagina web venga ottenuta mediante una JSP (è consigliabile crearla all’interno della cartella `WEB-INF`). I dati da visualizzare verranno passati alla pagina JSP dalla Servlet e in seguito verrà effettuato il forward:

   ```
   request.setAttribute("count", count); // numero visitatori
   request.setAttribute("list", list);   // lista visitatori
   RequestDispatcher disp =
   request.getRequestDispatcher("/WEB-INF/conta.jsp");
   disp.forward(request, response);
   ```

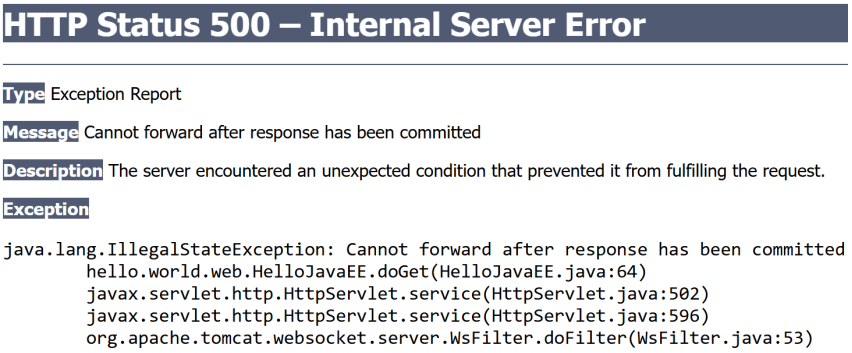
   Nella pagina JSP si utilizzerà l’*Expression Language* (EL) per visualizzare il valore del contatore:

   ```
   <h2>Welcome!</h2>
   <h4>You are the visitor number ${count}</h4>
   <form>
         <input type="hidden" name="reset" value="ok">
         <input type="submit" value="reset">
   </form>
   ```

Si può anche fare a meno dell'input hidden, semplicemente fornendo un nome per il pulsante submit:

```
<h2>Welcome!</h2>
<h4>You are the visitor number ${count}</h4>
<form>
      <input type="submit" name="btnreset" value="reset">
</form>
```

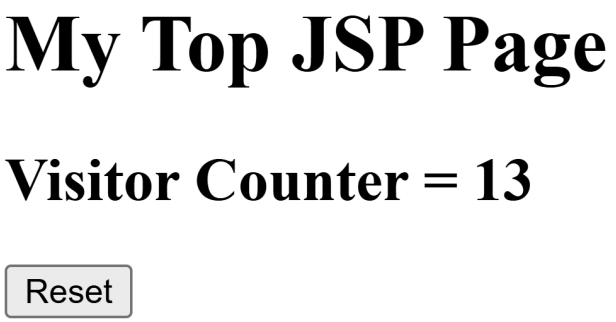
Fare attenzione all'operazione di `sendRedirect()`: assicurarsi che, in caso di redirezione, non venga effettuato anche il *dispatch* alla pagina JSP, poiché questo causerebbe un errore interno del server:



Ecco come gestire correttamente la *redirect* (come si nota, è stato aggiunto un `return` subito dopo la redirezione, per evitare che venga eseguita qualunque ulteriore istruzione nella servlet):

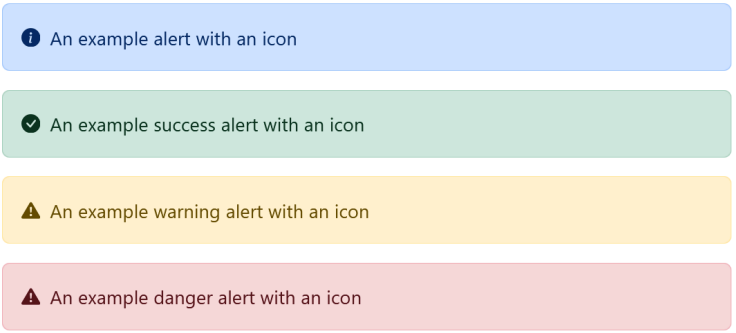
```
if(reset!=null) {
    count = 0;
    response.sendRedirect("HelloServlet");
    return;
}
```

Il risultato finale sarà qualcosa di simile a questo:



3. Creare una web application utilizzando una servlet come *controller* e una pagina JSP come *view*. La servlet si inizializza generando un numero pseudo casuale tra 1 e 1000. Il client deve cercare di **indovinare il numero** scrivendolo all'interno di un campo testuale e inviandolo alla servlet tramite un pulsante "submit". L'applicazione visualizza una pagina in cui scrive "Alto" se il numero inviato dal client è maggiore del numero segreto, "Basso" se è inferiore, "Hai vinto!" se ha indovinato. Se un client vince viene generato un nuovo numero da indovinare. Prevedere anche un pulsante

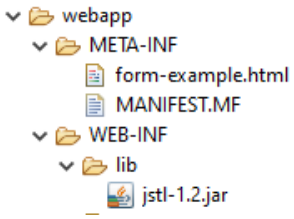
per forzare la generazione di un nuovo numero. Se possibile, visualizzare i messaggi utilizzando gli alert di Bootstrap 5:

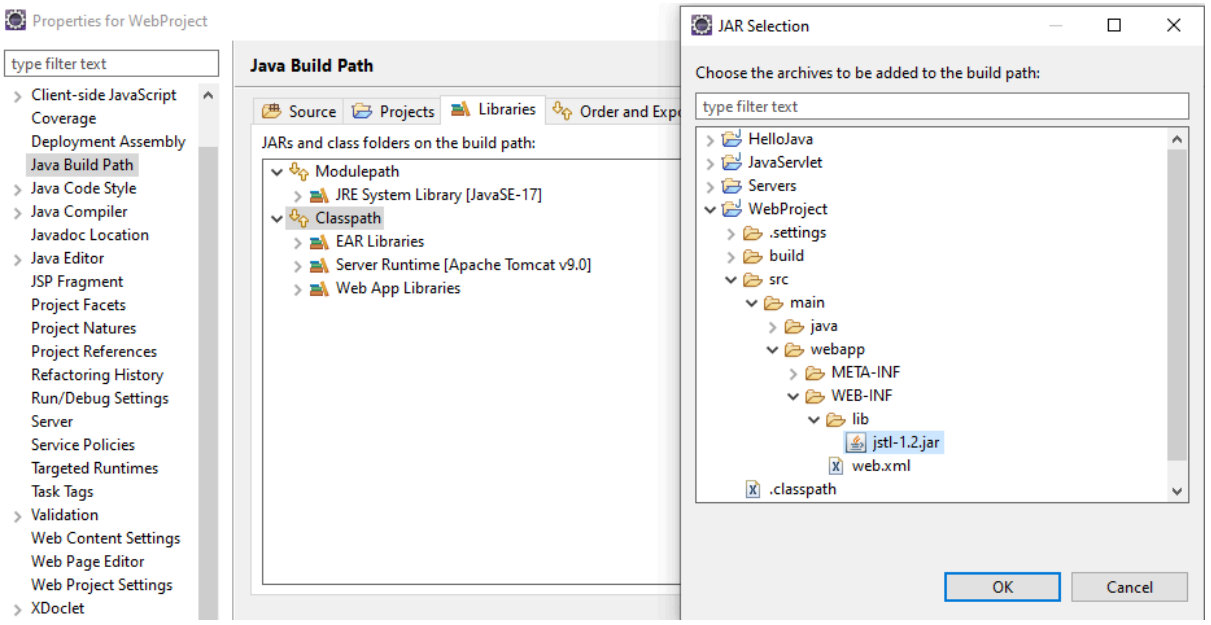


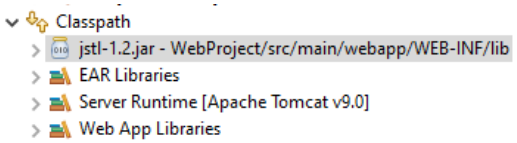
4. Creare una web application utilizzando una servlet come controller e una pagina JSP come view. La servlet deve gestire un **sistema di login** semplificato. La pagina JSP deve contenere un form con due campi testuali (username e password) e un pulsante "submit". La servlet verifica le credenziali inserite: se l'username è "admin" e la password è "Pippo1234", l'applicazione visualizza una pagina con il messaggio "Accesso riuscito". In caso contrario, visualizza il messaggio "Accesso negato" e permette di ritentare.
5. Creare una web application utilizzando una servlet come controller e una pagina JSP come view. La pagina JSP deve contenere un form con due campi numerici (numero1 e numero2) e un menu a tendina per selezionare l'**operazione aritmetica** (somma, sottrazione, moltiplicazione, divisione) e un pulsante "submit". La servlet riceve i numeri e l'operazione selezionata, esegue il calcolo e visualizza il risultato in una pagina JSP.
6. Creare una web application utilizzando una servlet come controller e una pagina JSP come view. La pagina JSP deve contenere un **form di registrazione** con campi per nome utente, password e email, oltre a un pulsante "submit". La servlet deve verificare che tutti i campi siano compilati correttamente e rispondere con un messaggio di successo ("Registrazione avvenuta con successo") o di errore ("Per favore, completa tutti i campi") a seconda del caso.
7. Creare una web application utilizzando una servlet come controller e una pagina JSP come view. La pagina JSP deve contenere un form con un campo numerico per l'importo, un menu a tendina per selezionare la valuta di origine (ad esempio, Euro) e un altro menu a tendina per selezionare la valuta di destinazione (ad esempio, Dollaro USA) e un pulsante "submit". La servlet deve **convertire l'importo inserito nella valuta di destinazione** utilizzando un tasso di cambio predefinito e visualizzare il risultato in una pagina JSP.
8. Creare una web application utilizzando una servlet come *controller* e una pagina JSP come *view*. Il client deve cercare di **indovinare una parola** segreta scrivendola all'interno di un campo testuale e inviandola alla servlet tramite un pulsante "submit". L'applicazione risponde visualizzando il messaggio "Hai indovinato!" se il client ha indovinato la parola, altrimenti visualizza il messaggio "Mi spiace, ritenta.." seguito da un suggerimento. I suggerimenti cambiano in modo random e sono "pescati" da un array di messaggi predisposto nell'applicazione (prevedere almeno una decina di suggerimenti differenti).
9. Creare una web application utilizzando una servlet come *controller* e una pagina JSP come *view*. La servlet **controlla la sicurezza di una password** inserita dall'utente

attraverso un campo testuale, e quantifica la sua robustezza con un punteggio da `0` a `9`. Creare un criterio di calcolo della robustezza personalizzato in base a:
    - lunghezza della password
    - presenza di caratteri maiuscoli e minuscoli
    - presenza di cifre
    - presenza di caratteri speciali
    - eventuali caratteri ripetuti più volte
    - presenza nella password di parole comuni che rendono possibile un attacco di tipo dizionario (come ad esempio “cielo”, “nuvola”, “sole”, nomi propri di persone, etc)

  La risposta della web app contiene anche le informazioni su come rendere la password più robusta (ad esempio: “aggiungi dei caratteri speciali”, oppure “la password è troppo corta”, oppure “ci sono troppi caratteri ripetuti”, o anche “non utilizzare parole di uso comune e nomi di persone nella password”)
10. Modificare l’esercizio sul contatore dei visitatori, facendo in modo che compaia anche la lista dei visitatori completa di indirizzo IP, numero di porta e timestamp per ogni visitatore. Per far questo è necessario passare alla pagina JSP un ArrayList (ad esempio) contenente tutte queste informazioni. Per visualizzare la lista all’interno della pagina JSP utilizzeremo la libreria *JSTL* (Java Standard Template Library). Aggiungere il *jar* della libreria *JSTL* alla pagina JSP creata al punto precedente, inserendo questo tag appena sotto il tag *page*:
    ```
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    ```
    Inserire il file **jstl-1.2.jar** all’interno della cartella **/WEB-INF/lib** e aggiungerlo al **Classpath** del progetto (tasto destro sul progetto -> properties -> Java Build Path -> Selezionare Classpath -> Add JARs…).



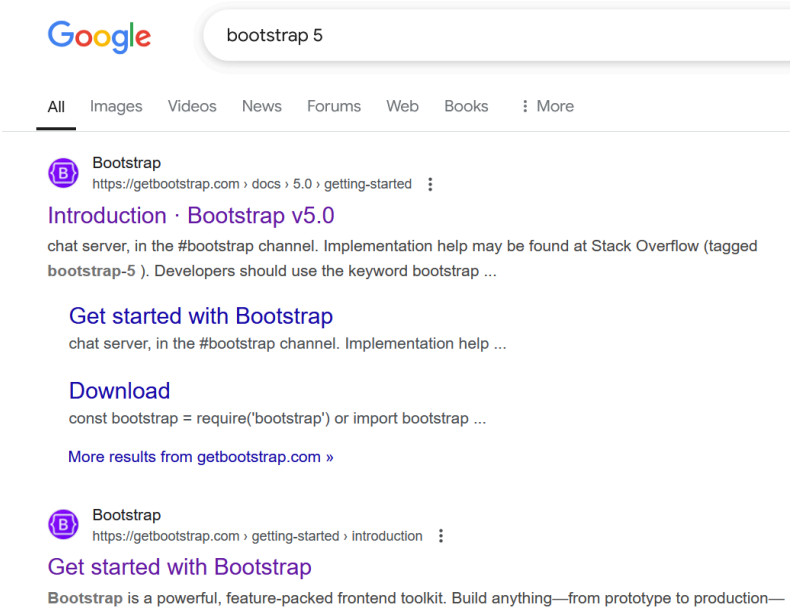




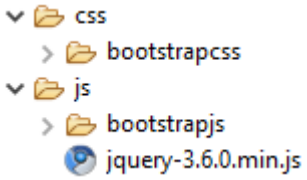
Nella pagina JSP, stampare la lista degli accessi all'interno di una tabella html utilizzando il *forEach* della libreria *jstl*:

```
<table>
   <c:forEach var = "visitor" items = "${list}">
   <tr>
     <td>
         <c:out value = "${visitor}" />
     </td>
   </tr>
   </c:forEach>
</table>
```

11. Integrare `Bootstrap` nella pagina JSP dell'esercizio precedente. Bootstrap è un framework Html/Css/JavaScript che rende più semplice la realizzazione di interfacce web.



Ecco i passaggi per incorporarlo nel progetto Eclipse:
Nel progetto web di Eclipse, creare le cartelle `js` e `css` all'interno di `webapp` (o `WebContent`, se utilizzate una versione precedente di Eclipse).
Scaricare la distribuzione di `bootstrap 5.0.2` e la libreria `jquery 3.6.0` dalla cartella condivisa su drive.
Estrarre l'archivio compresso di `bootstrap` sul proprio `Desktop`, rinominando la sottocartella `js` in `bootstrapjs` e la sottocartella `css` in `bootstrapcss` (per non confonderle con quelle create all'interno di `webapp` o `WebContent`).
Trascinare `bootstrapcss` nella cartella `css` di `webapp`, e `bootstrapjs` e `jquery-3.6.0.min.js` all'interno della cartella `js` di `webapp` (ex `WebContent`):



nella sezione `<head></head>` della pagina JSP, inserire i link a `bootstrap` e `jquery`:

```
<link href='css/bootstrapcss/bootstrap.min.css'
rel='stylesheet' type='text/css' />
<script src='js/jquery-3.6.0.min.js'
type='text/javascript'></script>
<script src='js/bootstrapjs/bootstrap.bundle.min.js'
type='text/javascript'></script>
```

In alternativa al metodo sopra descritto, l'installazione di `Bootstrap` può avvenire anche utilizzando la rete distribuita di server **cdn**, anziché installare i file di `Bootstrap` direttamente all'interno dell'applicazione web. Se si sceglie di utilizzare la rete cdn, il codice html da inserire nelle pagine JSP è il seguente (riferito all'attuale versione di `Bootstrap` all'ultima data di aggiornamento di questo esercizio - 01/02/2024).

- inserire il seguente codice nella sezione `<head></head>` della pagina:
  `<link`

```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">
```

- inserire il seguente codice appena prima della chiusura della sezione `<body>` della pagina:

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.min.js"
integrity="sha384-0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCsOGabYfZo0T0to5eqruptLy" crossorigin="anonymous"></script>
```

A questo punto, per verificare la corretta installazione di `Bootstrap`, modificare il sorgente della pagina JSP come segue:
inserire l'attributo `class="btn btn-primary"` all'interno del tag del pulsante di reset
inserire l'attributo `class="table"` (provare anche con i valori `table-striped` e `table-hover`) nel tag `table` della pagina JSP dell'esercitazione precedente, e verificare che cambi qualcosa nella visualizzazione dello stile della pagina web.
Esempio:

```
<table class="table table-striped table-hover">
```

Welcome!

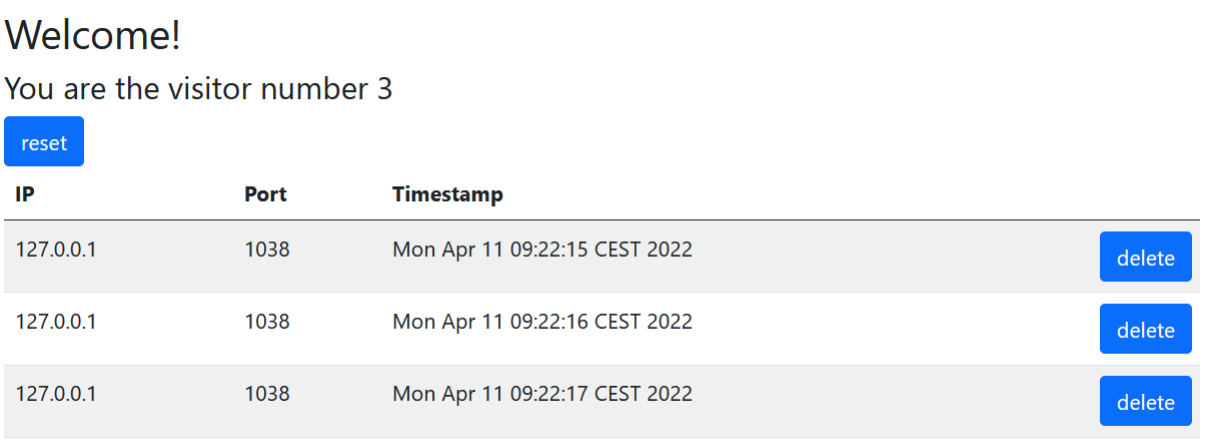
You are the visitor number 9

Lista accessi
127.0.0.1:28865 on Tue Apr 05 10:49:24 CEST 2022
127.0.0.1:28865 on Tue Apr 05 10:49:33 CEST 2022
127.0.0.1:28865 on Tue Apr 05 10:49:34 CEST 2022
127.0.0.1:28865 on Tue Apr 05 10:49:34 CEST 2022
127.0.0.1:28889 on Tue Apr 05 10:55:21 CEST 2022
127.0.0.1:28889 on Tue Apr 05 10:55:22 CEST 2022

12. Modificare l'esercizio precedente in modo tale che la tabella riporti in tre colonne separate:
    - indirizzo ip
    - numero di porta

- timestamp

Aggiungere una quarta colonna con un pulsante “delete” per cancellare la singola riga e un pulsante “reset” per effettuare il reset della tabella e del contatore accessi (devono essere vuoti/azzerati entrambi alla fine dell’operazione).
Per far questo è necessario creare una classe *wrapper* apposita da utilizzare come elemento della lista; a prescindere dal fatto che gli attributi della classe siano stati definiti come `public` o come `private`, per garantirne il corretto funzionamento nella pagina JSP, la classe deve implementare tutti i getter “standard” per gli attributi in essa definiti, rispettandone la convenzione sui nomi: se ad esempio un attributo della classe si chiama `pippo`, il getter si deve chiamare `getPippo()`. In questo modo, nella JSP si potrà richiamare il nome dell’attributo con l’espressione: `${item.pippo}`.



Per cancellare un singolo elemento dalla lista è necessario passare l’id di quell’elemento al controller mediante un form:

```
<c:forEach var = "visitor" items = "${list}">
  <tr>
    <td scope="col"><c:out value="${visitor.ip}" /></td>
    <td scope="col"><c:out value="${visitor.port}" /></td>
    <td scope="col"><c:out value="${visitor.timestamp}" /></td>
    <td scope="col">
      <form method='get' action=''>
        <input type='hidden' name='id' value='${visitor.id}' />
        <input class="btn btn-primary" type='submit'
value='delete'/>
      </form>
    </td>
  </tr>
</c:forEach>
```

13. Modificare l’esempio precedente aggiungendo una semplice gestione delle *route*. Le *route* rappresentano il sistema di navigazione dell’applicazione web, e fanno parte

della sua interfaccia. Per consentire la navigazione in diverse pagine, faremo in modo che il controller gestisca tutte le richieste il cui path termina con lo schema `"/App/*"`. Questo comportamento si ottiene mediante la seguente configurazione della servlet:

```
@WebServlet("/App/*")
```

Nel caso si voglia rimuovere il prefisso `"/App"` dall'url, cosicché il controller diventa la servlet di *default* (cioè risponde all'url vuoto `"/"` e `""`, e a tutti gli altri url non mappati in altre servlet dell'applicazione web), la configurazione è la seguente:

```
@WebServlet("/")
```

oppure:

```
@WebServlet("/*")
```

Importante: in caso si configuri la servlet come servlet di default, non sarà possibile servire il file css, js ed eventuali asset interni al progetto.
Nel nostro esercizio, l'applicazione web renderà disponibili solo due risorse:

- *home* (con path `"/App/home"`) , che rappresenta la homepage realizzata a piacere mediante la pagina `home.jsp` , sempre utilizzando `Bootstrap`
- *visitor* (con path `"/App/visitor"`), che rappresenta la solita pagina con il contatore e la lista dei visitatori, già implementata nelle esercitazioni precedenti

Inoltre, il server restituirà la pagina *home* anche per le richieste il cui path termina con `"/App/"` e `"/App"`, mentre per tutte le altre richieste non previste nei casi precedentemente elencati, il server risponde con una pagina di errore, realizzata mediante la pagina `error.jsp` (sempre utilizzando `Bootstrap`). Per effettuare il parsing del path della risorsa richiesta dal client, utilizzare il seguente frammento di codice:

```
String path = request.getRequestURL().toString();
String resource = path.substring(path.lastIndexOf('/') + 1);
```

La risorsa va poi analizzata per stabilire la corrispondente pagina da restituire al client:

```
/* Routing */
switch(resource.toLowerCase()) {
    case "home":
    case "":
        disp = request.getRequestDispatcher("/WEB-INF/home.jsp");
        disp.forward(request, response);
        break;
    case "visitor":
        request.setAttribute("counter", visit);
        request.setAttribute("list", list);
        disp = request.getRequestDispatcher("/WEB-INF/conta.jsp");
        disp.forward(request, response);
        break;
```

```
        default:
            disp = request.getRequestDispatcher("/WEB-INF/error.jsp");
            disp.forward(request, response);
            break;
    }
```

Inoltre, modificare l'interfaccia della web app aggiungendo una barra di navigazione. Ecco un esempio preso dalla documentazione di Bootstrap:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button"
       data-bs-toggle="collapse" data-bs-target="#navbarNav"
       aria-controls="navbarNav" aria-expanded="false"
       aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page"
                  href="home">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="paperino">Paperino</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="visitor">Visitor</a>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="#" tabindex="-1"
                  aria-disabled="true">Disabled</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

14. Modificare l'applicazione in modo tale che l'aggiunta dei client alla lista e l'incremento del contatore visitatori avvenga solo quando il client visita la home page dell'applicazione.

15. Aggiungere la procedura di autenticazione in modo che solo un utente autenticato possa visualizzare le pagine dell'applicazione. Per far questo è necessario utilizzare le sessioni. All'inizio del metodo `doGet()`, prendere il riferimento alla sessione del client con il seguente frammento di codice:

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(300);
// Overwrite session cookie to keep session even after the
browser is closed:
Cookie cookie = new Cookie("JSESSIONID", session.getId());
cookie.setMaxAge(Integer.MAX_VALUE);
response.addCookie(cookie);
```

L'override del cookie di sessione consente di mantenere la sessione attiva anche se il browser viene chiuso.
Inoltre, è necessario controllare se l'utente sta effettuando l'autenticazione attraverso il form di login e, nel caso, controllare le credenziali fornite. Se l'utente inserisce le credenziali corrette, verrà aggiunta una variabile `logged` in sessione, per marcare il client come autenticato e rilevarlo anche nelle successive richieste. Se il client non risulta autenticato, gli viene restituito il form di login, realizzato con apposita pagina JSP. Supponendo che, in questo esempio, le credenziali di accesso siano memorizzate "hard coded" in due variabili `user` e `password`:

```
// Check if user is trying to log in
// (and then check his credentials):
if(request.getParameter("user")!=null &&
request.getParameter("password")!=null) {
    if(request.getParameter("user").equals(user) &&
request.getParameter("password").equals(password)) {
        // Set this session variable to remember
        // that the user is authenticated:
        session.setAttribute("logged", true);
        System.out.println("Login successful:
"+request.getParameter("user")+",
"+request.getParameter("password"));
    }
    else
        System.out.println("Login failed:
"+request.getParameter("user")+",
"+request.getParameter("password"));
}
// Check if user is not logged (and show the login page):
```

```
if(session.getAttribute("logged")==null) {
    disp = request.getRequestDispatcher("/WEB-INF/login.jsp");
    disp.forward(request, response);
    return;
}
```

Per consentire all'utente di effettuare manualmente il logout, è necessario aggiungere un link "Logout" alla barra di navigazione delle pagine:

```
<li class="nav-item">
  <a class="nav-link" href="logout">Logout</a>
</li>
```

Chiaramente è necessario anche gestire la richiesta di logout all'interno del controller:

```
/* Routing */
switch(resource.toLowerCase()) {
    case "logout":
        session.invalidate();
        response.sendRedirect("home");
        break;
    ...
```

Il form di login è preso dagli esempi di Bootstrap:

```
<form>
  <div class="mb-3">
    <label for="email" class="form-label">Email address</label>
    <input name="user" type="email" class="form-control" id="email"
aria-describedby="emailHelp" required >
    <div id="emailHelp" class="form-text">We'll never share your
email with anyone else.</div>
  </div>
  <div class="mb-3">
    <label for="password" class="form-label">Password</label>
    <input name="password" type="password" class="form-control"
id="password" required >
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
```

```
</form>
```

16. Modificare l'applicazione precedente in modo che il controllo delle credenziali venga effettuato mediante accesso a un database. In particolare:
    utilizzare `MySQL`/`MariaDB` (incorporato in Xampp) eseguito in locale
    il nome del database è `userlogin` e il nome della tabella contenente le credenziali è `account`
    il nome dell'utente del database è `login` e la password è `pippo`
    gli *username* degli utenti dell'applicazione web sono indirizzi email (campo `email` nella tabella)
    le password degli utenti dell'applicazione web sono memorizzate in `hash MD5` (campo `pwd` nella tabella)
    la tabella degli utenti contiene anche i campi `name` e `surname`
    Di seguito è riportato il codice `SQL` per creare il database delle credenziali e l'utente del database:

```sql
CREATE DATABASE userlogin;
USE userlogin;
CREATE TABLE account (
          id int(11) NOT NULL AUTO_INCREMENT,
          name varchar(100) NOT NULL,
          surname varchar(100) NOT NULL,
          email varchar(100) NOT NULL,
          pwd varchar(100) NOT NULL,
          PRIMARY KEY (id)
);
INSERT INTO account (name, surname, email, pwd) VALUES
('Pippo','De Pippis','pippo@supersballo.gov', MD5('pippo')),
('Pluto','De Plutis','pluto@ultrafico.yes',  MD5('pippo')),
('Paperino','De Paperis','paperino@supersballo.gov',
MD5('pippo'));

CREATE USER 'login'@'%' IDENTIFIED BY 'pippo';
GRANT SELECT, UPDATE, INSERT, DELETE
 ON userlogin.* TO 'login'@'%';
CREATE USER 'login'@'localhost' IDENTIFIED BY 'pippo';
GRANT SELECT, UPDATE, INSERT, DELETE ON userlogin.* TO
'login'@'localhost';
```

    Il secondo utente `'login'@'localhost'` è utilizzato per effettuare la connessione dallo stesso host in cui gira il DBMS, mentre il primo, `'login'@'%'`, può connettersi da qualunque altro host della rete. Agli utenti del database verranno forniti solo i privilegi strettamente necessari che garantiscono il funzionamento della web application (in questo caso solo la possibilità di effettuare delle query).

Eventualmente, per modificare la password dell'utente del database, utilizzare il comando:

```
ALTER USER 'login'@'localhost' IDENTIFIED BY 'newPassword';
```

Ecco il contenuto della tabella `account`:

id	name	surname	email	pwd
1	Pippo	De Pippis	pippo@supersballo.gov	0c88028bf3aa6a6a143ed846f2be1ea4
2	Pluto	De Plutis	pluto@ultrafico.yes	0c88028bf3aa6a6a143ed846f2be1ea4
3	Paperino	De Paperis	paperino@supersballo.gov	0c88028bf3aa6a6a143ed846f2be1ea4

Per effettuare una connessione al DB `MySQL` da un programma Java è necessario il componente `JDBC` driver, che deve essere incluso nel progetto effettuando le seguenti operazioni:

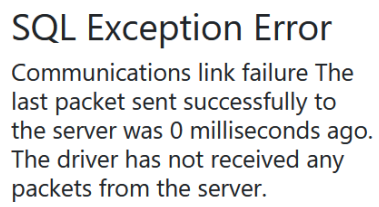
- Copiare il file jar della libreria (mysql-connector-java-8.0.28-bin.jar) nella directory `webapp\WEB-INF\lib`
- Aggiungere la libreria jar al **Build Path** del progetto: seleziona la cartella del progetto dal Package Explorer di Eclipse->tasto destro->**Properties**->**Java Build Path**->**Libraries**->**Add JARs..** e seleziona il file jar all'interno della cartella `webapp\WEB-INF\lib`

Creare una classe separata `DbHelper` per la gestione delle operazioni sul database:

```
public class DbHelper {
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DBurl =
             "jdbc:mysql://localhost:3306/userlogin";
    private static final String user = "login";
    private static final String pwd = "pippo";
    private Connection con;

    public DbHelper() {
      con = null;
      try {
      Class.forName(DRIVER);  // check if library is loaded
      System.out.println("Driver Connector/J trovato!");
      }
      catch (ClassNotFoundException e) {
      System.out.println("Driver Connector/J NON trovato!");
      }
    }
    /* CONNESSIONE DB */
    public void connect() throws SQLException {
        con = DriverManager.getConnection(DBurl, user, pwd);

    }
    public void disconnect() throws SQLException {
        if(con != null)
             con.close();
    }
```

```
    /* DB API */
    public boolean logon(String email, String pwd) throws SQLException {
        String query = "SELECT * FROM account WHERE
              email='"+email+"' AND pwd=md5('"+pwd+"')";
        Statement sql = con.createStatement();
        ResultSet res = sql.executeQuery(query);
        if(res.next())  // if query result contains a row
              return true;
        return false;
    }
}
```

In caso di eccezione generata da un'operazione sul database, mostrare all'utente una nuova pagina JSP di errore con il messaggio relativo all'eccezione (`e.getMessage()`), del tipo:



17. Aggiungere le seguenti funzionalità all'applicazione.
    - Creare una pagina di registrazione per nuovi utenti. Aggiungere quindi nella classe `DbHelper` il metodo per inserire un nuovo utente nel database. Ecco un esempio di codice Java per effettuare un inserimento:

      ```
      String cmd = "INSERT INTO accounts (name, surname, email,
      password)"+
      "VALUES ('"+name+"', '"+surname+"', '"+email+"'+"',
      md5('"+password+"'))";
      Statement sql = con.createStatement();
      sql.execute(cmd);
      ```

    - Quando si tenta di registrare un nuovo utente, controllare che l'email (lo username) non sia già presente nella tabella degli account. Se è già presente, l'applicazione mostra un errore. Inoltre, un utente che effettua la registrazione con successo è automaticamente autenticato con le credenziali appena registrate (non deve quindi effettuare il login subito dopo la registrazione). Per consentire ad un utente non autenticato di accedere alla pagina di registrazione, è necessario apportare una modifica al frammento di codice che si occupa di deviare l'utente sulla pagina di login:

      ```
      // Check if user user is logged:
      if(session.getAttribute("logged")==null) {
        if(resource.toLowerCase().equals("register"))
           disp =
      request.getRequestDispatcher("/WEB-INF/regis.jsp");
      ```

```
  else
     disp =
request.getRequestDispatcher("/WEB-INF/login.jsp");
  disp.forward(request, response);
  return;
}
```

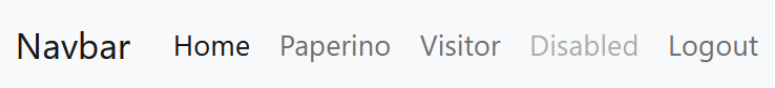
- Aggiungere anche una pagina `/users` in cui vengono riportati in una tabella tutti gli utenti dell'applicazione, specificandone nome, cognome e email. Ecco un esempio di come è possibile accedere, riga per riga, ai campi di una tabella risultante da una query:

```
String query = "SELECT * FROM mytable";
Statement sql = con.createStatement();
ResultSet res = sql.executeQuery(query);
while(res.next()) { // field index starts from 1
      String firstField = res.getInt(1);
      String secondField = res.getString(2);
      ...
}
```

  I campi di ogni record vengono recuperati con i metodi `getInt(index)`, `getLong(index)`, `getDouble(index)`, `getString(index)`, `getDate(index)`, etc.. L'index corrisponde alla posizione del campo rispetto alla tabella prodotta dalla query. Il primo campo della tabella ha indice `1`.

- Nella tabella html degli utenti, inserire per ogni riga un pulsante "delete" che elimina il singolo utente dal database.

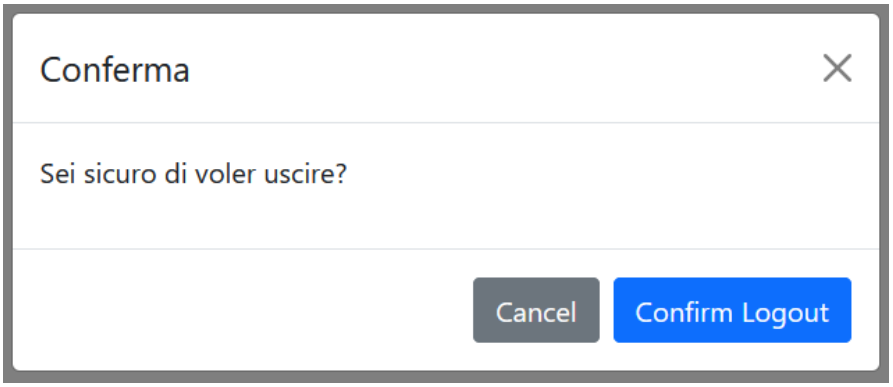
18. Fare in modo che il nome dell'utente autenticato compaia nella barra di navigazione dell'applicazione, al posto della scritta `Navbar`:



   Il nome utente non deve essere cliccabile (utilizzare la classe css `disabled`):

```
<a class="nav-link disabled" href="#">${username}</a>
```

19. Modificare l'esercizio precedente in modo che ogni volta che si clicca sul link "`Logout`" della barra di navigazione, compaia una **finestra modale** che richiede di confermare dell'operazione. Utilizzare questo esempio come guida:



Nell'esempio html fornito, la finestra è attivata dal pulsante "Logout" mediante uno script. La definizione delle variabili per accedere agli elementi html e la registrazione dei callback degli eventi viene effettuata solo dopo che l'intera pagina html viene caricata nel DOM; questo comportamento si ottiene attraverso la funzione `ready()` di jQuery:

```
$(document).ready(function(){ … });
```

In generale `$()` è una funzione jQuery utilizzata per trovare uno o più elementi nella pagina. È importante inserire questo script nella sezione `<body>`, poiché l'istruzione `$(document).ready()` va eseguita senza uno specifico trigger.
Per utilizzare il modale è necessario prendere il riferimento all'elemento corrispondente:

```
myModalLogout = new
bootstrap.Modal(document.getElementById('modalLogout'), {
        backdrop: 'static', // don't close if click outside
        keyboard: false   // don't close if press esc
    })
```

In questo caso, i callback vengono registrati utilizzando la funzione `on()` di jQuery:

```
$(document).on("click", "#logout", function (event) {
      event.preventDefault();
      myModalLogout.show();
});
```

Il metodo `preventDefault()` impedisce al browser di eseguire l'azione di default prevista dall'evento stesso. Se si tratta ad esempio del click su un collegamento ipertestuale, il browser non effettuerà la richiesta per la nuova pagina specificata nel link. Questo viene fatto perché non vogliamo che l'utente esca prima di aver confermato tramite finestra modale.
Per implementare il modale relativo al logout nel progetto web, seguire i seguenti passaggi:

- creare una cartella `shared/` all'interno di `WEB-INF/`, e inserire al suo interno una nuova pagina JSP nominata come `modal-logout.jsp` ; la cartella `shared/` sarà impiegata per ospitare tutti i frammenti di codice che saranno riutilizzati e incorporati in più pagine JSP, come appunto la finestra modale per il logout
- inserire il seguente codice nella nuova pagina JSP appena creata, eliminando tutto l'html precedentemente esistente:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<script type="text/javascript">
$(document).ready(function(){
    // Show dialog to confirm logout:
    modalLogout = new
bootstrap.Modal(document.getElementById('modalLogout'), {
        backdrop: 'static', //don’t close modal if user click
outside
        keyboard: false // can’t even press esc from keyboard
    })
    formLogout = document.getElementById("logout-form");
    // Register a callback to trigger when user clicks “Logout”:
    $(document).on("click", "#logout", function(event) {
        event.preventDefault(); // prevent default action
        modalLogout.show(); // show modal window
    });
    // Register a callback to trigger when user clicks “Confirm ..”:
    $(document).on("click", "#logout-form-submit", function () {
        formLogout.submit(); // submit form
        modalLogout.dismiss(); // hide modal window
    });


});

</script>

<!-- Modal with form: confirm to logout -->
<div id="modalLogout" class="modal" tabindex="-1"
data-bs-backdrop="static" data-bs-keyboard="false" >
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Confirm Logout</h5>
        <button type="button" class="btn-close"
        data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
```

```
        <p id="modal-message">Are you sure you want to exit?</p>
        <form id="logout-form" method="post" action="logout">
          <input hidden type="text" name="id" id="id"
value="${variable}"/>
          <!--button id="your-id">submit</button-->
        </form>

      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary"
        data-bs-dismiss="modal">Cancel</button>
        <button type="button" class="btn btn-primary"
        data-bs-dismiss="modal" id="logout-form-submit" >
        Confirm Logout</button>
      </div>
    </div>
  </div>
</div>
```

- All'interno di ogni pagina JSP dell'applicazione, inserire il link alla libreria jQuery (poichè viene utilizzata nei nostri script) e includere la pagina JSP contenente la finestra modale. Per far questo basta inserire il seguente frammento di codice in tutte le pagine JSP dell'applicazione, appena prima della chiusura del tag `<body>`:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/j
query.min.js"></script>
<jsp:include page="shared/modal-logout.jsp"/>
```

- Infine, nella barra di navigazione di tutte le pagine JSP, è necessario modificare il link "`Logout`" aggiungendo l'attributo `id` e impostandolo al valore `"logout"`, che è lo stesso utilizzato nella parte JavaScript per la registrazione del callback all'evento "`click`":

```
<a class="nav-link" id="logout" href="logout">Logout</a>
```

- …

20. Fare in modo che compaia un modale di conferma anche quando l'utente clicca sul tasto "`Reset`" per cancellare la lista degli accessi e azzerarne il contatore.
21. Mostrare un modale di conferma per la cancellazione di ogni singola riga della tabella degli accessi. Il funzionamento è analogo a quello implementato per il tasto "`Reset`", ma in questo caso è necessario passare alla finestra modale anche l'`id`

dell'elemento da eliminare mediante uno script JavaScript. L'esempio completo si trova qui. Notare come la proprietà `data-id` del pulsante "`delete`" della tabella venga utilizzata per memorizzare l'id dell'elemento da cancellare; questo valore viene poi passato all'input hidden del form della finestra modale mediante istruzione JavaScript:

```
$('#myModalDelete .modal-body #id').val($(this).data('id'));
```

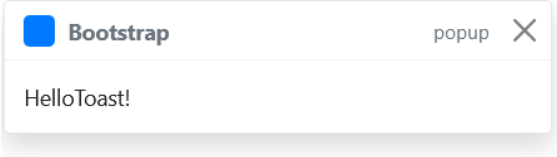
Gli attributi di tipo `data-*` vengono utilizzati per memorizzare informazioni all'interno degli elementi html, senza che vi sia alcuna visualizzazione grafica di tali informazioni nell'interfaccia utente.
Si richiede di utilizzare la stessa finestra modale sia per l'operazione di reset dell'intera lista, sia per la cancellazione della singola riga. Fare in modo che il messaggio mostrato all'utente nella modale sia coerente con l'operazione da eseguire: se si tratta del reset completo, verrà mostrato un messaggio del tipo `"Vuoi cancellare l'intera lista?"`; se invece si tratta dell'eliminazione di una singola riga, verrà visualizzato un messaggio del tipo `"Vuoi eliminare la riga con id=...?"`. Utilizzare il metodo `html()` di jQuery per impostare il contenuto del messaggio.

22. Modificare l'esercizio precedente in modo tale che ogni volta che la lista è vuota compaia un messaggio di popup (`toast`) con il testo `"Non ci sono visitatori!"`. Si può mostrare un opportuno messaggio di conferma anche a seguito della cancellazione di una riga dalla tabella. Per far questo, è sufficiente passare alla pagina JSP un attributo `"toastmessage"` contenente la stringa del messaggio da visualizzare nel popup. Dopo aver inserito il codice html del Toast nella pagina JSP, inserire `${toastmessage}` come contenuto testuale del `div toast-body`. Il seguente codice JavaScript (da eseguire a caricamento pagina completato) consente la visualizzazione automatica del Toast quando è presente un messaggio all'interno del `toast-body`:

```
if($('.toast .toast-body').html().trim()!="")
// if element toast-body contains a non empty string
        $('.toast').toast('show');  // then show the toast
```

Per cambiare il comportamento del Toast, modificare gli attributi dell'elemento `<div class="toast">`. In particolare:

- utilizzare l'attributo `data-bs-delay` per impostare la durata del Toast (espressa in ms), ad esempio `data-bs-delay="5000"`
- se si vuole invece disabilitare la scomparsa automatica del Toast utilizzare l'attributo `data-bs-autohide="false"`



Utilizzare questo esempio come guida. Ecco i vari passaggi da realizzare per implementare il toast nel progetto:

- Creare una nuova JSP nominata come `toast.jsp` all'interno della cartella `shared/`
- Inserire all'interno della JSP il seguente codice:

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<script type="text/javascript">
    $(document).ready(function() {
        if ($('.toast .toast-body').html().trim() != "")
       // if element toast-body contains a string
            $('.toast').toast('show'); // show the toast
    });
</script>

<!-- Toast -->
<div style="position: relative;">
    <!-- Flexbox container for aligning the toasts:
position-fixed class do the overlay -->
    <div aria-live="polite" aria-atomic="true"
        class="d-flex justify-content-center
            align-items-center w-100 position-fixed">
        <div class="toast-container">
            <div class="toast" data-bs-delay="5000"
                  data-autohide="false"
                  role="alert" aria-live="assertive"
                  aria-atomic="true">
              <div class="toast-header">
               <svg class="bd-placeholder-img rounded me-2"
                    width="20" height="20"
                    xmlns="http://www.w3.org/2000/svg"
                    aria-hidden="true"
                    preserveAspectRatio="xMidYMid slice"
                    focusable="false">
               <rect width="100%" height="100%"
                     fill="#007aff"></rect></svg>
               <strong class="me-auto">Bootstrap</strong>
               <small>popup message</small>
               <button type="button" class="btn-close"
                     data-bs-dismiss="toast"
                     aria-label="Close"></button>
            </div>
            <div class="toast-body">${toastmessage}</div>
            </div>
```

```
            </div>
        </div>
    </div>
```

- Nelle pagine in cui si desidera visualizzare un toast, inserire il seguente frammento di codice prima della chiusura del tag `<body>`:

```
<jsp:include page="shared/toast.jsp">
    <jsp:param name="toastmessage"
    value="${toastmessage}"/>
</jsp:include>
```

- Per triggerare il toast dal controller, è necessario passare alla pagina JSP un parametro contenente il messaggio da visualizzare nel toast:

```
request.setAttribute("toastmessage", "Message to
show here!");
```

Oltre a implementare il messaggio `"Non ci sono visitatori!"` quando la lista è vuota, fare in modo che compaia un *toast* ogni volta che l'utente effettua con successo il login (messaggio `"Login successful!"`), ogni volta che il login fallisce (messaggio `"Login failed!"`) e ogni volta che viene richiesta una pagina inesistente (messaggio `"Page not found!"`). Utilizzare la sessione per memorizzare quando è appena avvenuto un login oppure quando è fallito.
Suggerimento: per gestire il toast di login conviene memorizzare una variabile in sessione ogni volta che si rileva un tentativo di autenticazione da parte dell'utente:

```
session.setAttribute("login-attempt", true);
```

In seguito, controllando se questa variabile di sessione è presente, si può mostrare all'utente il messaggio corretto nei diversi casi, ricordandosi di eliminare la variabile alla fine del controllo.
A titolo di esempio, il seguente frammento di codice del *controller* gestisce gli utenti non autenticati mostrando loro la pagina di login; si può notare come, in caso sia appena avvenuto un tentativo di login dell'utente (evidentemente non andato a buon fine), venga mostrato il *toast* con il messaggio `"Login failed!"`:

```
// Check if user is not logged:
if(session.getAttribute("logged")==null) {
    if(session.getAttribute("login-attempt")!=null) {
        request.setAttribute("toastmessage", "Login failed!");
        session.removeAttribute("login-attempt");
    }
    disp = request.getRequestDispatcher("/WEB-INF/login.jsp");
    disp.forward(request, response);
    return;
}
```

Analogamente, si può gestire il login andato a buon fine all'interno della logica che

realizza il *routing*. In questo caso, nella pagina *home* comparirà il *toast* con il messaggio `"Login successful!"`:

```
...
case "home":
case "app":
case "":
    if(session.getAttribute("login-attempt")!=null) {
        request.setAttribute("toastmessage",
                             "Login successful!");
        session.removeAttribute("login-attempt");
    }
    ...
```

23. Creare una pagina per l'autenticazione utente con username e password.
L'applicazione consiste in un'unica servlet centrale (controller) e in due pagine JSP (`login.jsp` e `logout.jsp`). La pagina di `login`, con relativo form, compare a qualunque utente che non abbia effettuato l'autenticazione. Una volta autenticato, il server se ne ricorderà anche per le richieste successive, mostrando sempre la pagina di `logout` al posto di quella di `login`, finché la sessione non scade. In particolare:
l'inserimento dei campi è obbligatorio (`required`)
la password è oscurata
le credenziali di accesso fisse (*hardcoded*) sono `pippo` (username) e `pippo123` (password)
Cliccando sul pulsante *accedi* viene mostrata una seconda pagina, che chiameremo pagina di `logout`, con un titolo `<h1>` contenente il testo `"Welcome <username>"` e un un pulsante *esci* per effettuare il logout
nella pagina di `logout`, in caso di login corretto compare anche un `toast` con messaggio `"login effettuato con successo"`
nella pagina di `login`, in caso di login errato, compare un `toast` con messaggio `"credenziali errate"`, specificando eventualmente se è errato lo username o la password (`"username errato"` o `"password errata"`)
il tempo massimo di inattività della sessione è 2 minuti
la sessione sopravvive anche se il browser viene chiuso
(extra) premendo il pulsante `logout` compare un `modal` per chiedere conferma di uscita dall'applicazione
(extra) nella pagina di `login`, è possibile visualizzare la password spuntando un'apposita checkbox *mostra password* nel form di inserimento
Ecco il frammento di codice da aggiungere all'inizio del metodo `doGet()` per utilizzare la sessione:

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(300);
```

```
// Overwrite session cookie to keep session even after the
browser is closed:
Cookie cookie = new Cookie("JSESSIONID", session.getId());
cookie.setMaxAge(Integer.MAX_VALUE);
response.addCookie(cookie);
```

24. Modificare l'applicazione precedente in modo che il controllo delle credenziali venga effettuato mediante accesso a un database. In particolare:
    utilizzare MySQL/MariaDB (incorporato in Xampp) eseguito in locale
    il nome del database è `userlogin` e il nome della tabella contenente le credenziali è `account`
    il nome dell'utente del database è `login` e la password è `pippo`
    gli *username* degli utenti dell'applicazione web sono indirizzi email (campo `email` nella tabella)
    le password degli utenti dell'applicazione web sono memorizzate in `hash MD5` (campo `pwd` nella tabella)
    la tabella degli utenti contiene anche i campi `name` e `surname`
    Di seguito è riportato il codice SQL per creare il database delle credenziali e l'utente del database:

```
CREATE DATABASE userlogin;
USE userlogin;
CREATE TABLE account (
          id int(11) NOT NULL AUTO_INCREMENT,
          name varchar(100) NOT NULL,
          surname varchar(100) NOT NULL,
          email varchar(100) NOT NULL,
          pwd varchar(100) NOT NULL,
          PRIMARY KEY (id)
);
INSERT INTO account (name, surname, email, pwd) VALUES
('Pippo','De Pippis','pippo@supersballo.gov', MD5('pippo')),
('Pluto','De Plutis','pluto@ultrafico.yes',  MD5('pippo')),
('Paperino','De Paperis','paperino@supersballo.gov',
MD5('pippo'));

CREATE USER 'login'@'%' IDENTIFIED BY 'pippo';
GRANT SELECT, UPDATE, INSERT ON userlogin.* TO 'login'@'%';
CREATE USER 'login'@'localhost' IDENTIFIED BY 'pippo';
GRANT SELECT, UPDATE, INSERT ON userlogin.* TO
'login'@'localhost';
```

    Il secondo utente `'login'@'localhost'` è utilizzato per effettuare la connessione

dallo stesso host in cui gira il DBMS, mentre il primo, `'login'@'%'`, può connettersi da qualunque altro host della rete. Agli utenti del database verranno forniti solo i privilegi strettamente necessari che garantiscono il funzionamento della web application (in questo caso solo la possibilità di effettuare delle query).
Eventualmente, per modificare la password dell'utente del database, utilizzare il comando:

```
ALTER USER 'login'@'localhost' IDENTIFIED BY 'newPassword';
```

Ecco il contenuto della tabella `account`:

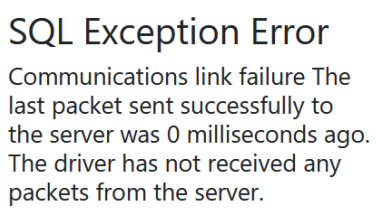
id	name	surname	email	pwd
1	Pippo	De Pippis	pippo@supersballo.gov	0c88028bf3aa6a6a143ed846f2be1ea4
2	Pluto	De Plutis	pluto@ultrafico.yes	0c88028bf3aa6a6a143ed846f2be1ea4
3	Paperino	De Paperis	paperino@supersballo.gov	0c88028bf3aa6a6a143ed846f2be1ea4

Per effettuare una connessione al DB `MySQL` da un programma Java è necessario il componente `JDBC` driver, che deve essere incluso nel progetto effettuando le seguenti operazioni:
Copiare il file jar della libreria (mysql-connector-java-8.0.28-bin.jar) nella directory webapp\WEB-INF\lib
Aggiungere la libreria jar al Build Path del progetto: seleziona la cartella del progetto dal Package Explorer di Eclipse->tasto destro->Properties->Java Build Path->Libraries->Add JARs.. e seleziona il file jar all'interno della cartella webapp\WEB-INF\lib
Creare una classe separata `DbHelper` per la gestione delle operazioni sul database:

```
public class DbHelper {
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DBurl =
            "jdbc:mysql://localhost:3306/userlogin";
    private static final String user = "login";
    private static final String pwd = "pippo";
    private Connection con;

    public DbHelper() {
      con = null;
      try {
       Class.forName(DRIVER);  // check if library is loaded
       System.out.println("Driver Connector/J trovato!");
       }
      catch (ClassNotFoundException e) {
       System.out.println("Driver Connector/J NON trovato!");
       }
    }
    /* CONNESSIONE DB */
    public void connect() throws SQLException {
        con = DriverManager.getConnection(DBurl, user, pwd);

    }
```

```
    public void disconnect() throws SQLException {
        if(con != null)
            con.close();
    }
    /* DB API */
    public boolean logon(String email, String pwd) throws SQLException {
        String query = "SELECT * FROM account WHERE
            email='"+email+"' AND pwd=md5('"+pwd+"')";
        Statement sql = con.createStatement();
        ResultSet res = sql.executeQuery(query);
        if(res.next())  // if query result contains a row
            return true;
        return false;
    }
}
```

In caso di eccezione generata da un'operazione sul database, mostrare all'utente una nuova pagina JSP di errore con il messaggio relativo all'eccezione (`e.getMessage()`), del tipo:



Suggerimento: per generare l'eccezione `SQL` nell'applicazione e verificare il funzionamento della pagina di errore basta spegnere `MySQL`..

25. Creare una pagina di cambio password, in cui l'utente autenticato inserisce la vecchia password, la nuova password e la conferma della nuova password. Il sistema accetta il cambio password solo se la vecchia password è corretta e la nuova password rispecchia i seguenti requisiti:
    è diversa dalla vecchia password
    è lunga almeno 8 caratteri
    contiene almeno una lettera maiuscola
    contiene almeno una cifra
    contiene almeno un carattere tra i seguenti: `@#$%&`
    Utilizzare dei messaggi `toast` come conferma o avviso di errore nella procedura di cambio password. La validazione della password può essere effettuata lato server (dal controller) oppure lato client in `JavaScript`. In particolare, in `jQuery` esiste il metodo `val()` che restituisce il valore di un input. Inoltre è possibile definire una funzione per controllare i requisiti della password:

```
function checkPassword(password){
    //regular expression for password
    var pattern =
/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%&]).*$/;
```

```
    if(pattern.test(password)){
        return true;
    }else{
        return false;
    }
}
```

Di seguito è riportato il link alla spiegazione dell'espressione regolare da StackOverflow, un tool online per l'interpretazione del significato e il test delle espressioni regolari e un articolo che spiega la procedura per controllare la password. Un esempio (incompleto) di validazione password in JavaScript è il seguente (questa è la versione con il pulsante mostra password). **Il codice va opportunamente integrato e corretto** in modo che mostri all'utente quali caratteristiche mancano nella password inserita. Inoltre il sistema di controllo della complessità della password non funziona in modo ottimale (si nota ad esempio che le password molto lunghe vengono considerate comunque deboli). A questo proposito può essere utile verificare la robustezza delle password inserite in password monster; in questo modo si possono anche tarare meglio le espressioni regolari per il controllo.

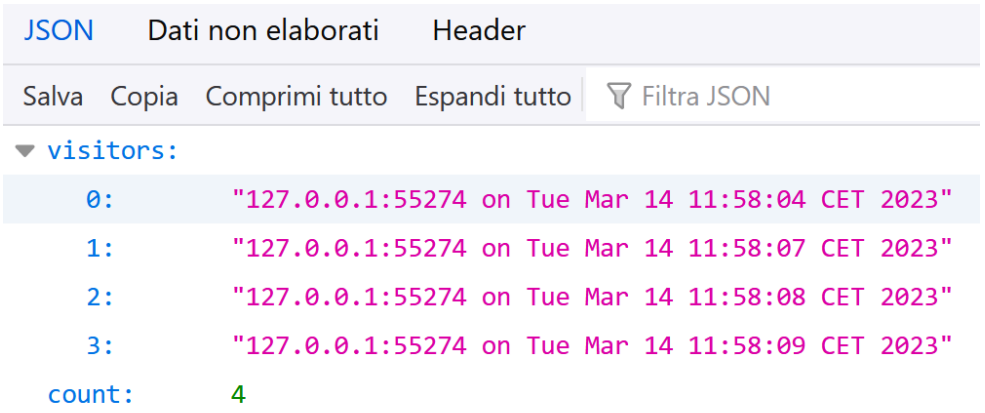
N.B.: Nelle regex utilizzate, l'espressione `(?=)` rappresenta un *look ahead positive*, dove si controlla se una certa espressione è seguita dall'espressione specificata dopo il simbolo uguale. Se ci sono ulteriori espressioni *look ahead*, la valutazione della stringa riparte sempre dal punto precedente all'espressione *look ahead*. In questo modo si controlla ad esempio se la stringa contiene un carattere cifra, poi si controlla se la stessa stringa contiene un carattere "lettera maiuscola", poi si controlla se la stessa stringa contiene almeno un carattere "lettera minuscola", e così via (come se ci fosse un `AND` tra le singole espressioni *look ahead*).

26. Creare una web app che si interfaccia a un database MySQL. In particolare:
    l'applicazione web che si collega al database "azienda" e visualizza l'intera tabella "dipendente" in una `<table>` html.
    aggiungere un form per inserire un nuovo record nella tabella
    aggiungere la voce "elimina" accanto a ogni record (nella table html) della tabella per poterlo eliminare dal database.
    fare in modo che venga visualizzato un `toast` con messaggio di conferma per ogni inserimento di un nuovo record
    fare in modo che ad ogni cancellazione di record venga mostrato un `modal` di conferma prima di effettuare la cancellazione
    Per effettuare una connessione al DB `MySQL` da un programma Java è necessario `JDBC` driver, che deve essere incluso nel progetto effettuando entrambe le seguenti operazioni:
    1. Copiare il file jar della libreria (mysql-connector-java-8.0.28-bin.jar) nella directory WebContent\WEB-INF\lib
    2. Aggiungere la libreria jar al Build Path del progetto: seleziona la cartella del progetto dal Package Explorer di Eclipse->tasto destro->Properties->Java Build Path->Libraries->Add JARs.. e seleziona il file jar all'interno della cartella WebContent\WEB-INF\lib

27. Creare una servlet che restituisca i dati dell'esercizio della lista visitatori in formato JSON. Il contatore e la lista vengono azzerati se viene inviato il parametro "reset". La servlet fa parte del progetto `WebProject` e il suo path è `"/json"`; modificare quindi l'annotazione `@WebServlet` come segue:

```
@WebServlet("/json")
```

Per inviare dati in formato json, utilizzare la seguente libreria, da incorporare nel progetto come è stato fatto per la JSTL. La servlet restituirà un output di questo tipo:



In questo caso non è necessario l'utilizzo di pagine JSP. Il content-type sarà impostato a "application/json":

```
JSONObject json=new JSONObject();
...
String data = json.toString();
response.setContentType("application/json");
response.getWriter().println(data);
```

Ecco dei frammenti di codice di riferimento per la creazione del json:

```
import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONArray;

// Creating JSON object:
JSONObject auth=new JSONObject();
auth.put("username","admin");
auth.put("password", "pwd1234");
String message = auth.toString();

// Creating JSON array:
JSONArray array = new JSONArray();
array.add("element_1");
array.add("element_2");
array.add("element_3");

// Creating JSON array from ArrayList of Strings:
ArrayList<String> list = new ArrayList<String>();
list.add("foo");
list.add("baar");
```

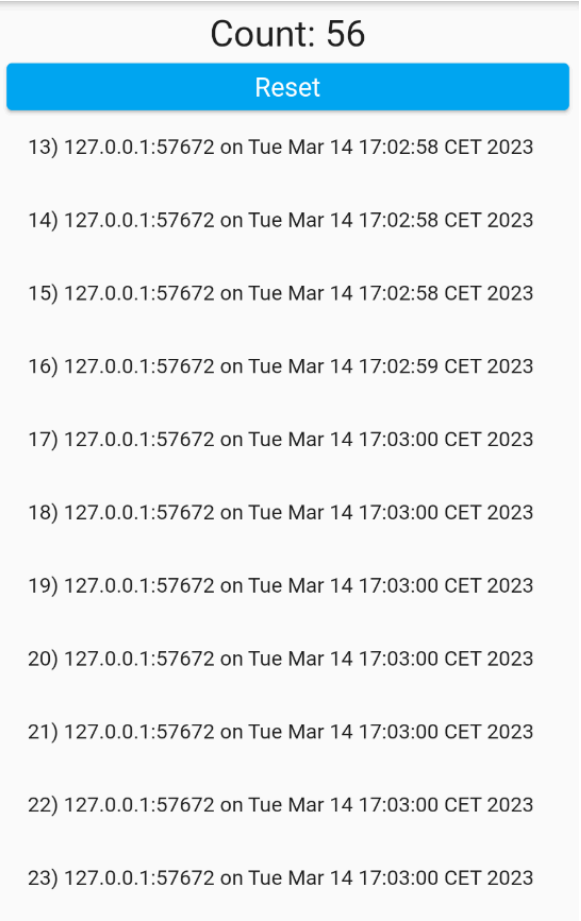
```
JSONArray jsArray = new JSONArray(list);

// Creating JSON array from ArrayList of objects:
ArrayList<Person> list = new ArrayList<Person>();
... // filling the list with objects..
JSONArray jsonarr = new JSONArray(list);
// As an alternative, you can even
// manually put objects into the array:
for(Person item : list) {
      JSONObject jsonobj = new JSONObject();
      jsonobj.put("firstName", item.getName());
      jsonobj.put("lastName", item.getSurname());
      jsonarr.put(jsonobj);
}

// Parsing JSON:
JSONObject jObj = new JSONObject(jsonString);
String totalItems= jObj.getString("totalItems");
```

28. Modificare l'applicazione della lista dei visitatori in modo che il contatore e i dati sugli accessi vengano richiesti in modo asincrono con richiesta di tipo AJAX. Utilizzare a questo proposito la servlet creata al punto precedente, che restituisce i dati richiesti in formato json. Ecco un'esempio di richiesta AJAX utilizzando la libreria jQuery:

```
$.get('ContaJson',  function(response){
      var visitors = response['visitors']; // get access data
      var table = $("#table-body-id"); // find table body
      table.empty();  // clear table content
      $.each(visitors, function(index, visitor){ // fill table
        table.append("<tr><td>"+visitor+"</td></tr>");
      });
});
```

Notare che non è necessario eseguire il parsing della response, che viene già fornita come oggetto JavaScript. Lo script deve essere eseguito sia al caricamento iniziale della pagina (evento ready), sia ogni volta che viene premuto il pulsante "refresh" (da inserire). Modificare opportunamente sia il back end che il front end in modo tale che nella tabella vengono visualizzati ip, porta e timestamp in colonne separate. Aggiungere anche le funzionalità di reset e cancellazione della singola riga.

29. Creare il front-end Flutter per l'applicazione che fornisce il contatore e la lista dei visitatori in formato json. La app presenta un'unica pagina da cui si visualizza il numero dei visitatori e la lista scollabile. Inoltre è presente anche il tasto "Reset":



È necessario installare il modulo http per effettuare le richieste al back end (servlet). Il package va installato da terminale con il comando:

```
> flutter pub add http
```

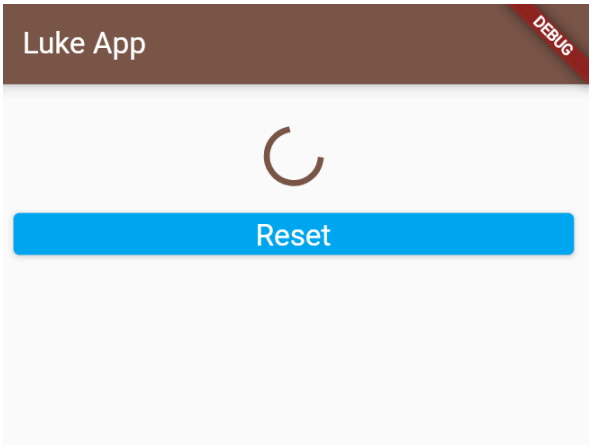
Includere il seguente package per poter effettuare le http request:

```
import 'package:http/http.dart' as http;
```

Per poter inserire la `ListView` all'interno del widget `Column` (che contiene anche il `Text` e `ElevatedButton`), è necessario inserire la `ListView` stessa in un widget `Expanded`:

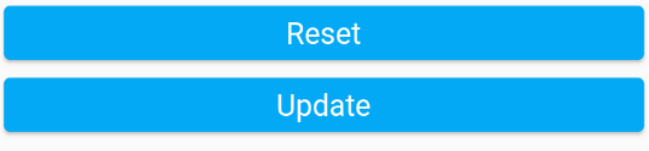
```
Expanded(
   child: ListView.builder(
      itemCount: _list.length,
       itemBuilder: (context, index) {
           return ListTile(
               title: Text("${_list[index]}"),
               onTap: () {
               },
           );
       },
   ),
),
```

30. Se la connessione al server non fosse disponibile, visualizzare una barra di progressione circolare al posto del valore del contatore, utilizzando il widget `CircularProgressIndicator`:

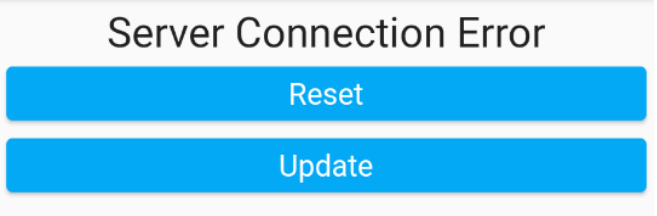



```
if (_count == null)
  Container(
    margin: const EdgeInsets.all(20.0),
    child: CircularProgressIndicator(),
  ),
if (_count != null && _count! >= 0)
  Text(
    "Count: $_count",
    style: TextStyle(fontSize: 28),
    textAlign: TextAlign.center,
  ),
```

Aggiungere un pulsante “Update” per aggiornare la lista degli accessi:



Visualizzare il messaggio “errore di connessione al server” al posto della rotellina di caricamento in caso di server spento, non raggiungibile o connessione rifiutata:



Utilizzare il costrutto `try-catch` per intercettare gli errori relativi alla richiesta http:

```
void doGet() async {
```

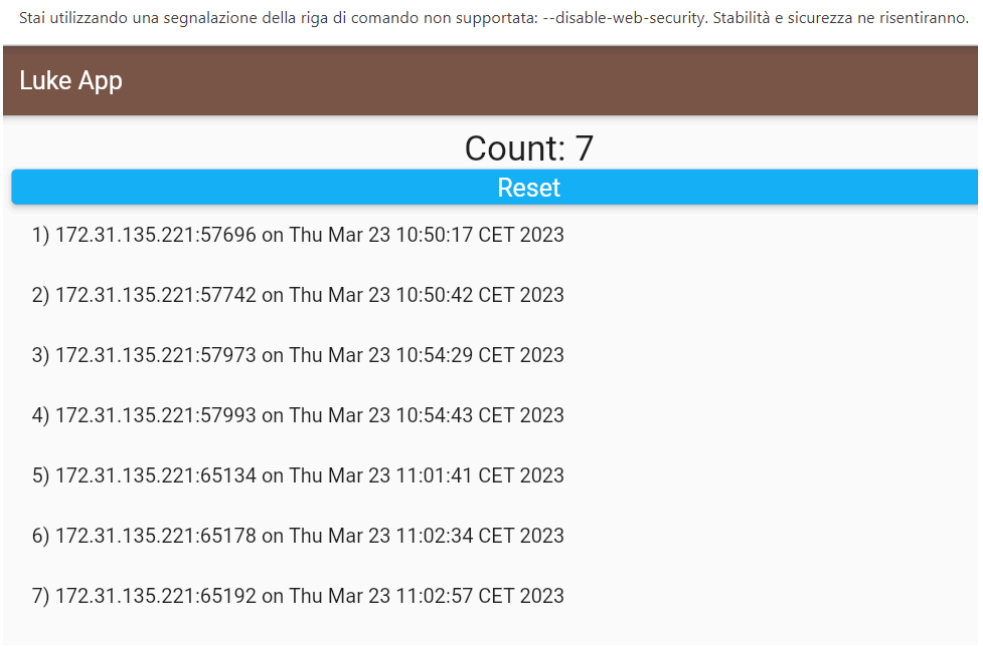
```
        try {
            var response = await http.get(Uri.parse(url));
            setState(() { ... });
        }
        catch(e) {  // Connection Refused
            print(e);
            setState(() { ... });
        }
    }
```

31. Provare a realizzare anche la versione web dell'applicazione precedente. Per evitare il problema delle CORS (Cross-Origin Resource Sharing), inserire questo frammento di codice nella servlet in modo da soddisfare le "preflight" request del browser:

```
response.addHeader("Access-Control-Allow-Origin", "*");
response.addHeader("Access-Control-Allow-Methods", "POST, GET,
OPTIONS, PUT, DELETE, HEAD");
response.addHeader("Access-Control-Allow-Headers", "X-PINGOTHER,
Origin, X-Requested-With, Content-Type, Accept");
response.addHeader("Access-Control-Max-Age", "1728000");
```

In alternativa è possibile disabilitare il controllo delle CORS da parte del browser, a discapito ovviamente della sicurezza. Per avviare Chrome disabilitando la sicurezza sulle CORS è necessario utilizzare il terminale:

```
>chrome.exe --disable-web-security
```

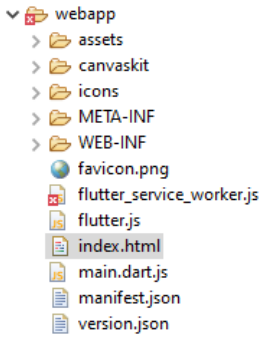


Per evitare il problema delle CORS senza abilitarle sulla servlet e senza disabilitare la sicurezza sul browser, è necessario che il front end e il back end vengano integrati

nello stesso server (Apache Tomcat in questo caso). Per far questo, creare la build del progetto web di Flutter con il comando:

```
flutter build web --release --web-renderer html
```

Copiare tutti i file presenti dentro la cartella `build/web` nella cartella `webapp` del progetto `WebProject` di Eclipse:



Aprire da Eclipse il file `index.html` e modificare l'attributo `href` dell'elemento `<base>` come segue (supponendo che il progetto web si chiami WebProject):

```
<base href="/WebProject/">
```

Verificare che tutto funzioni correttamente inserendo questo url nel browser:

```
http://localhost:8080/WebProject/
```