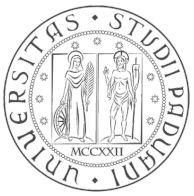


**Università degli Studi di Padova**

Facoltà di Ingegneria

*Corso di Laurea in Ingegneria Informatica*

# Utilizzo delle tecnologie web per la semplificazione burocratica nella Pubblica Amministrazione

*Relazione tirocinio*

*Laureando*

**Alan Giacomin**

*Relatore*

**Prof. Sergio Congiu**

Anno Accademico 2010/2011

Alan Giacomin:

*Utilizzo delle tecnologie web per la semplificazione burocratica nella Pubblica Amministrazione,* Relazione del tirocinio, 21 Luglio 2011.

E-MAIL:

alan.giacomin@gmail.com

# RINGRAZIAMENTI

Vorrei ringraziare, innanzitutto, il dirigente dell'Ufficio Immigrazione della Questura di Treviso, dott.ssa Elena Peruffo, e l'ispettore capo dott. Roberto Bassan per avermi seguito durante l'attività di tirocinio. Un immenso grazie va ai miei genitori ed a mio fratello, per il loro immancabile aiuto in ogni momento del mio percorso universitario. Ringrazio, infine, i miei parenti ed i miei amici per l'affetto che mi hanno sempre dimostrato.

# INTRODUZIONE

L'evoluzione tecnologica in ambito informatico, nel corso degli ultimi anni, si è sviluppata ad un ritmo esponenziale e questo ha portato a cambiamenti nella vita quotidiana delle persone. Si pensi all'*home banking* che permette di operare sul proprio conto bancario direttamente dal computer di casa o al frequente utilizzo della posta elettronica, alla quale si ricorre anche per una semplice comunicazione al collega di lavoro nella stanza a fianco. Una mancante o inadeguata informatizzazione dei sistemi o dei soli dati può portare a lavorare come in passato, "carta e penna", richiedendo quindi un maggior dispendio di energie per portare a termine i propri compiti. Questo comporta inevitabilmente un incremento della probabilità di commettere errori o che vengano adoperate differenti metodologie di lavoro, le quali, se incompatibili tra loro, rischiano di compromettere lo svolgimento di incarichi successivi, con ulteriori prolungamenti dei tempi di esecuzione.

Questo elaborato si pone l'obiettivo di analizzare i metodi di lavoro e le necessità di un ufficio della Pubblica Amministrazione al fine di fornire una valida miglioria per gli operatori, garantendo maggiore efficienza nel soddisfare le richieste degli utenti. Nel caso specifico, viene preso in considerazione l'Ufficio Immigrazione della Questura presso il quale è stato svolto il tirocinio.

# INDICE

# 1 LA SITUAZIONE ATTUALE

INDICE



## 1.1 L'UFFICIO IMMIGRAZIONE

L'Ufficio Immigrazione appartiene alla divisione amministrativa della Questura Italiana ed è competente per quanto riguarda la regolamentazione dei cittadini extracomunitari presenti nel territorio italiano.

Diviso in sezioni, tra i suoi compiti ci sono quelli di analisi e valutazione delle domande presentate dagli stranieri, eventuali rigetti delle stesse, se presentano irregolarità, o espulsioni, nel caso si verifichino gravi mancanze o reati commessi dal richiedente.

Da qui la necessità di avere informazioni aggiornate e condivise tra le varie sezioni, in modo da garantirne un miglior utilizzo e maggiore velocità nel reperimento in caso di richiesta da parte di altri uffici della Questura stessa o di altri enti, quali ad esempio i Carabinieri o la Guardia di Finanza, per una migliore cooperazione tra loro.

## 1.2 IL PERMESSO DI SOGGIORNO

Per ottenere il permesso di soggiorno è previsto, attualmente, che il cittadino extracomunitario si rechi presso un ufficio postale abilitato per presentare la domanda. Qui riceve dallo sportellista un appuntamento per presentarsi in Questura ed una ricevuta, da conservare assieme al passaporto, comprovante l'assoluzione di tutti gli obblighi di legge per poter soggiornare in Italia; la conferma definitiva rimane poi a carico dell'Ufficio Immigrazione il quale prenderà in esame la pratica acquisita.

In sede di convocazione presso la Questura, l'invitato deve consegnare tutti i documenti richiesti a seconda del tipo di permesso desiderato e deve essere disponibile nei confronti degli operatori a rilasciare le proprie impronte digitali, atto obbligatorio in quanto sono successivamente integrate nel permesso elettronico. Le domande acquisite sono poi analizzate per verificare l'idoneità del richiedente ed il possesso di tutti i requisiti necessari per il titolo di soggiorno richiesto. In caso di esito negativo, si procede con ulteriori convocazioni per eventuali documenti mancanti o errati, oppure con l'avvio di un procedimento di rigetto della domanda qualora si presentino carenze di più grave entità.

Tutte queste fasi operative sono manualmente annotate all'interno del fascicolo cartaceo, il quale poi viene passato a chi ne deve seguire l'iter della pratica, ma senza alcuna traccia di tutti questi spostamenti il rischio di non trovare le pratiche quando necessario è elevato.

Oltre alle procedure operative, è necessario tenere conto anche della legislazione corrente: nel caso specifico, ci si attiene a quanto presente nel D.Lgs. 25 luglio 1998, n. 286, in materia di "Testo Unico del-

le disposizioni concernenti la disciplina dell'immigrazione e norme sulla condizione dello straniero", integrato e modificato dalla L. 30 luglio 2002, n. 189, in materia di "Modifica alla normativa in materia di immigrazione e di asilo" ed in applicazione del suddetto testo gli operatori ricorrono spesso al rilascio di modelli compilati al computer inserendo i dati con procedure non efficienti.

Il lavoro dell'operatore viene reso ancora più complesso con l'utilizzo, oltre agli strumenti poco tecnologici sopra menzionati, di un portale nazionale per i permessi di soggiorno, il quale evidenzia nella globalità lo status di acquisizione/consegna dei documenti senza entrare in dettaglio nelle singole procedure operative delle quali non si fa alcuna annotazione. Si segnala, infine, anche l'impiego del portale messo a disposizione da Poste Italiane per un'automazione delle convocazioni degli stranieri per il ritiro della domanda o per la consegna finale del permesso di soggiorno.

Tutto questo, quindi, richiede ulteriore tempo per poter servire il cittadino extracomunitario ed ascoltare le richieste di quello successivo, oltre a prolungare la durata complessiva di evasione delle pratiche; attualmente, infatti, è previsto che i permessi siano consegnati nel termine di trenta giorni dalla richiesta, ma con fasi operative non efficienti si ottengono ritardi i quali portano a non rispettare le scadenze e ad un inevitabile accumulo di lavoro arretrato, difficile poi da recuperare.

# 2 L'AMBIENTE DI LAVORO

INDICE



## 2.1 SERVER WEB

Il progetto prevede l'utilizzo di un'architettura distribuita a due livelli: un fornitore del servizio (*server*) ed un utilizzatore (*client*). Per ottenere un'informazione, ad esempio l'anagrafica dello straniero interessato, il client spedisce la richiesta al server, il quale, è in grado di restituire l'informazione richiesta elaborando i dati presenti. Sviluppando un'applicazione web, il client sarà costituito dal browser web della macchina utente mentre per il server ci si affida alla ben collaudata struttura di Apache HTTP Server.

Il server Apache utilizza una particolare architettura, cioè presenta più MultiProcessing Modules (MPMs) i quali permettono di eseguirlo nella configurazione migliore per adattarsi alle infrastrutture richieste.

Un server web è in grado di gestire più siti ospitati grazie all'utilizzo degli host virtuali. Con questo meccanismo, opportunamente

configurato, è possibile prevedere una copia di prova del progetto in uso per effettuare test o implementazioni di nuove funzionalità.

Oltre alle funzioni base del *core*, Apache può essere esteso con moduli compilati, i quali hanno ambiti di lavoro differenti: sono in grado di interpretare particolari linguaggi di programmazione (PHP, Perl, etc.), gestire le autenticazioni degli utenti, svolgere funzioni di proxy, includere supporto SSL o concedere il rewriting dell'URL.

Avendo necessità di disporre all'utente pagine web con contenuti dinamici, per questo progetto è obbligatorio che il server sia configurato con il modulo *mod_php* caricato, in modo da concedere l'esecuzione degli script, assieme a *mod_auth_mysql*, per l'interazione con il database.

## 2.2 DATABASE

Dopo aver preparato la macchina che ospiterà l'applicazione è necessario pensare all'organizzazione dei dati; per fare questo ci si può affidare ad un DBMS (Database Management System), il quale consente la gestione in maniera efficiente di uno o più database[1].

Con una buona analisi delle informazioni interessate e ricorrendo ad un gestore si riduce la ridondanza dei dati, presente quando alcuni dati sono ripetuti in più archivi, limitando di fatto il rischio di incongruenze, cioè di avere elementi valorizzati in maniera differente, la qual cosa porterebbe ad avere un database inconsistente e quindi

1 Un database è costituito da uno o più grandi insiemi strutturati di dati persistenti, di solito associati con il software per aggiornare e interrogare i dati. Un semplice database potrebbe essere un singolo file contenente molti record, ognuno dei quali contiene lo stesso insieme di campi in cui ogni campo ha una lunghezza fissa.

non affidabile per le elaborazioni di cui si necessita.

La scelta del DBMS è stata effettuata analizzando le varie disponibilità per il mondo del web e relazionandole con le esigenze ricercate. Volendo gestire enormi quantità di dati con notevole affidabilità e sicurezza, un prodotto che ha grande successo ed un largo utilizzo è MySQL; con una dizione anglosassone viene indicato come "applicazione a livello Enterprise", con la quale si intende che il software è di livello tale da poter essere utilizzato da grandi aziende per la gestione dei dati inerenti la propria attività.

La licenza d'uso è un altro vantaggio in quanto il prodotto è a pagamento solo per usi commerciali, quindi per le aziende che ottengono un profitto con il suo utilizzo, rimanendo invece gratuita negli altri casi.

## 2.3 LINGUAGGI CLIENT-SIDE E SERVER-SIDE

Sviluppando un progetto web vengono utilizzati principalmente linguaggi appartenenti a due distinte categorie: client-side e server-side.

### Linguaggi client-side

I linguaggi che vengono interpretati dal browser (client-side) sono presenti all'interno delle pagine web sotto forma di script e permettono di realizzare effetti grafici o richiedere l'interazione dell'utente per eventi particolari, come ad esempio un semplice messaggio di avviso.

Il primo esempio (Codice 2.1) utilizza VBScript (Visual Basic Script),

proposto da Microsoft, ma questo può essere eseguito solamente all'interno di Internet Explorer; questa limitazione deve essere tenuta in considerazione in quanto il browser è il punto d'accesso tra l'utilizzatore e l'applicazione su tutti i terminali.

**Codice 2.1**: Esempio VBScript

```
<script language="vbscript">
document.write("Today is " & WeekdayName(Weekday(Date)))
</script>
```

Differente, invece, il secondo esempio (Codice 2.2) realizzato utilizzando JavaScript, nato ad opera di Netscape, perché può essere interpretato da qualsiasi browser, pur con implementazioni differenti.

**Codice 2.2**: Esempio JavaScript

```
<script language="javascript">
var d=new Date();
var weekday=new Array(7);
weekday[0]="Sunday";
weekday[1]="Monday";
weekday[2]="Tuesday";
weekday[3]="Wednesday";
weekday[4]="Thursday";
weekday[5]="Friday";
weekday[6]="Saturday";
document.write("Today is " + weekday[d.getDay()]);
</script>
```

Per facilitare il compito degli sviluppatori JavaScript, nel corso del tempo sono state realizzate librerie in grado di limare i difetti dell'ambiente dove viene eseguito. Tra tutte queste, jQuery è una di quelle di maggior pregio e di più ampia adozione alla quale si farà affidamento anche all'interno di questo progetto. Per comprendere l'enorme semplificazione si può vedere (Codice 2.3) come con una sola riga di codice si sia in grado di selezionare determinati elementi della pagina (tutti i tag * * con classe *invisible*), aggiungere a questi una

classe CSS (*popup*) e infine visualizzarli con un'animazione eseguita lentamente.

**Codice 2.3:** Esempio jQuery

```
$("div.invisible").addClass("popup").show("slow");
```

Sviluppare una cosa analoga con il puro codice JavaScript avrebbe richiesto un alto numero di righe di codice.

### Linguaggi server-side

Questa categoria di linguaggi viene eseguita sul server web e l'utente non è in grado di averne accesso, ma ne può solamente consultare il risultato all'interno del browser in sede di navigazione.

Con l'utilizzo di pagine web statiche, l'utente tramite il browser effettua una richiesta ed il server risponde inviando una pagina HTML pari a quella presente nei suoi dischi. Utilizzando invece un linguaggio server-side e di conseguenza pagine web dinamiche, alla richiesta dell'utente il server individua la risorsa interessata, ne esegue il codice ottenendo a runtime un risultato in formato HTML ed inviando questo al client in attesa. Tramite l'elaborazione prima di rispondere al client si può consentire ad esempio l'esecuzione di operazioni su file o l'accesso alla base dati.

La scelta del linguaggio, come per la categoria precedente, è stata prevalentemente pilotata dalla compatibilità con il sistema operativo: ASP (Active Server Pages) è utilizzabile solo in ambienti Windows e quindi si è preferito l'utilizzo di PHP (PHP: Hypertext Preprocessor) in quanto creato e adattato sia per sistemi Windows sia per quelli Unix-like.

Un semplice esempio evidenzia la differenza tra il file presente sul server (Codice 2.4) e quello che poi risulta a video nel browser. Il file php contiene un intreccio di tag HTML e linee di codice da elaborare; la pagina web risultante (Codice 2.5; Figura 1), oltre al titolo, visualizza una sequenza numerica da 1 a 5 con un valore per riga.

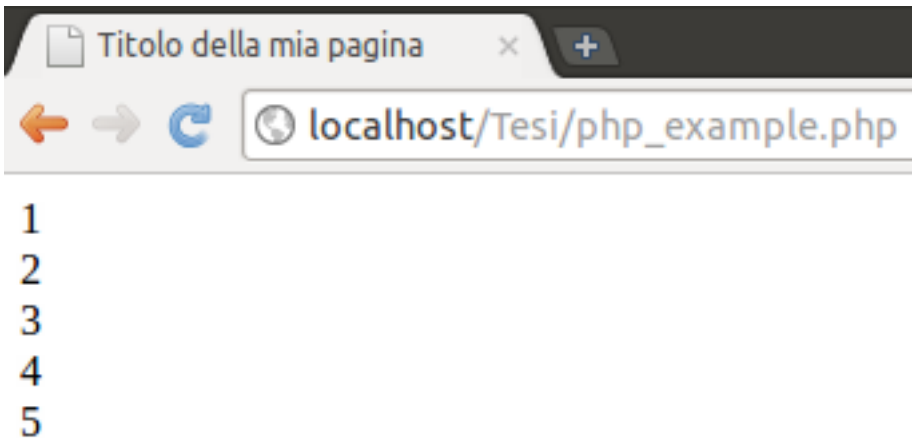
**Codice 2.4:** Esempio PHP

```
<?php
   $title = "Titolo della mia pagina";
?>
<html>
<head>
   <title><?php echo $title; ?></title>
</head>
<body>
<?php
   for ($i = 1; $i <= 5; ++$i)
   {
      echo "$i<br />";
   }
?>
</body>
</html>
```

**Codice 2.5:** Risultato HTML

```
<html>
<head>
   <title>Titolo della mia pagina</title>
</head>
<body>
   1<br />
   2<br />
   3<br />
   4<br />
   5<br />
</body>
</html>
```

**Figura 1:** L'esecuzione nel browser



# 3 LA REALIZZAZIONE

INDICE



## 3.1 AJAX

AJAX è un modo di indicare l'utilizzo di un insieme di tecnologie già esistenti singolarmente. Si differenzia quindi da quanto visto finora non essendo un oggetto reale o un linguaggio di programmazione d'ultima generazione.

In principio si trova l'HTML, evoluto con il passare degli anni, con a disposizione un Modello a Oggetti (DOM - Document Object Model) modificabile in qualsiasi momento nel browser tramite JavaScript. Da qui il termine DHTML (Dynamic HTML) in quanto i contenuti non rimangono statici dopo il loro caricamento e visualizzazione nel brow-

ser. Successivamente ha fatto comparsa l'XML che è diventato lo strumento per eccellenza per il trasporto dei dati nella programmazione moderna.

Questo insieme di tecnologie (DHTML, JavaScript e XML) ha portato alla costituzione di AJAX, acronimo per Asyncronous JavaScript and XML, cioè XML utilizzato da JavaScript in modo asincrono.

I flussi dati tra client e server passano sempre per i meccanismi di richiesta (*Request*) e risposta (*Response*), ove la seconda è l'intera pagina ritornata al richiedente. Ad una nuova richiesta segue poi una risposta contenente la pagina con i nuovi contenuti.

Si pensi ad un form di inserimento dati da validare prima che questi siano elaborati. Tramite l'utilizzo tradizionale l'utente compila il form e lo spedisce al server; questo lo analizza e se riscontra errori rimanda la stessa pagina indietro all'utente con indicazioni sui dati da correggere. Con l'utilizzo delle richieste asincrone, invece, è possibile interrogare il server in tempo reale durante la compilazione e permettere l'invio solo se la verifica ha avuto esito positivo. Questo consente una migliore interazione per l'utente ed un risparmio in termini di tempo e banda utilizzata.

In ambito tecnico tutto inizia con l'oggetto JavaScript *XMLHttpRequest*, il quale si occupa di realizzare le richieste compiute dal browser, utilizzando il protocollo standard HTML e restituendo l'informazione della risposta in formato XML.

Come accennato in precedenza, anche questo oggetto risente delle differenze implementative tra i vari browser e di conseguenza non è sufficiente istanziare normalmente l'oggetto (Codice 3.1), ma è necessario verificare a priori l'ambiente in cui è eseguito e comportarsi di conseguenza (Codice 3.2).

**Codice 3.1**: Istanza di XMLHttpRequest

```
var xmlHttpRequest = new XMLHttpRequest();
```

**Codice 3.2**: Istanza controllata di XMLHttpRequest

```
var xmlHttpHttpRequest = makeXMLHttpRequest();
if (!xmlHttpRequest)
{
   alert("Utilizzo di AJAX non supportato");
}
else
{
   // esecuzione richiesta con AJAX
   // [...]
}

/** Istanzia oggetto XMLHttpRequest **/
function make XMLHttpRequest()
{
   var request = undefined;
   if (window.XMLHttpRequest) // Mozilla, Safari, ...
   {
      request = new XMLHttpRequest();
      if request.overrideMimeType)
      {
         // per ovviare al bug di certe versioni di Mozilla
         request.overrideMimeType("text/xml");
      }
   }
   else if (window.ActiveXObject) // IE
   {
      try
      {
         request = new ActiveXObject("Msxml2.XMLHTTP");
      }
      catch (e)
      {
         // eventuale gestione eccezione
         // [..]
      }
   }
   return request;
}
```

XMLHttpRequest, essendo di natura un oggetto, porta con sé alcune proprietà e metodi come descritti nelle relative tabelle 1 e 2.

**Tabella 1**: Proprietà dell'oggetto XMLHttpRequest

| Proprietà | Descrizione |
|---|---|
| `onreadystatechange` | Gestore degli eventi (*event handler*) per ogni cambiamento di stato della richiesta |
| `readyState` | Valore intero che identifica lo stato della richiesta:<br>0 (unitialized) non è stata ancora invocata open()<br>1 (loading) deve ancora essere effettuata la richiesta con send()<br>2 (loaded) la risposta del server non è ancora disponibile<br>3 (interactive) la risposta è letta ancora parzialmente<br>4 (complete) la richiesta è completa e la risposta completamente |
| `responseText` | La risposta del server in formato stringa |
| `responseXML` | La risposta del server processata dal parser XML e convertita in un oggetto DOM |
| `status` | Valore numerico dello stato del server, per esempio 404 ("Not Found") per indicare che l'URL della richiesta non è stato trovato |
| `statusText` | Il messaggio in formato stringa associato allo stato del server |

**Tabella 2**: Metodi dell'oggetto XMLHttpRequest

| Metodi | Descrizione |
|---|---|
| `open(method, URL [, asyncFlag [, userName [, password]]])` | Assegna il metodo e l'URL della richiesta e, come parametri opzionali, l'asincronia (`true`) o sincronia (`false`) della richiesta, eventuali username e password |
| `send(content)` | Trasmette la richiesta, con argomento `null` se la richiesta è `GET`, facoltativamente con i parametri di `POST` come stringa o DOM |
| `abort()` | Arresta la richiesta corrente |
| `getAllResponseHeaders()` | Ritorna l'insieme completo degli header (nomi e valori) come una stringa |
| `getResponseHeader(headerLabel)` | Ritorna il valore di un header dato il suo nome |
| `setRequestHeader(label, value)` | Aggiunge all'header inviato con la richiesta una coppia nome/valore |

Senza entrare troppo in dettaglio, si può vedere come eseguire una richiesta al server in modalità GET (Codice 3.3), cioè con passaggio di parametri in forma pubblica tramite l'URL stesso.

**Codice 3.3**: Richiesta GET

```
xmlHttpRequest.open("GET","example.php?param1=aaa", true);
xmlHttpRequest.send(null);
```

Ad ogni richiesta segue una risposta e lo script deve essere in grado di interpretarla. A tal fine è necessario indicare le azioni da intraprendere ad ogni variazione di stato della comunicazione (Codice 3.4).

**Codice 3.4**: AJAX: funzione di Callback

```
xmlHttpRequest.onreadystatechange = ajaxCallback;

/** Callback delle richieste AJAX **/
function ajaxCallback()
{
   try
   {
      if (xmlHttpRequest.readystate == 4) // risposta completa
      {
         if (xmlHttpRequest.status == 200) // risposta positiva
         {
            var text = xmlHttpRequest.responseText;
            alert("Test di AJAX: " + text);
         }
         else if (xmlHttpRequest.status == 404) // errore 404
         {
            alert("404: Not found");
         }
         else if (xmlHttpRequest.status == 500) // errore 500
         {
            alert("500: Internal Server Error");
         }
         else  // altro errore
         {
            alert(xmlHttpRequest.status + ":"
                  + xmlHttpRequest.statusText);
         }
      }
```

```
      else
      {
         // risposta in esecuzione
         // [...]
      }
   }
   catch (e)
   {
      alert("Errore in callback function: " + e);
   }
}
```

In aggiunta agli aspetti base è possibile modificare gli script per aggiungere nuove funzionalità, quali ad esempio un timeout qualora l'elaborazione richiedesse troppo tempo.

Come è facile notare, la complessità degli script va crescendo sempre più con l'aggiunta di nuove implementazioni e le varie differenze dei browser. Per ovviare a questo problema è utile l'utilizzo di jQuery il quale fornisce, attraverso l'impiego di semplici metodi (Codice 3.5), l'intera gestione dell'oggetto XMLHttpRequest e conseguente comunicazione con il server.

**Codice 3.5:** jQuery AJAX

```
$.ajax({
   async: true, // true se chiamata asincrona, false se sincrona
   url: "example.php", //pagina richiamata
   type: "POST", // GET o POST
   data: "param1=aaa&param2=bbb", // elenco parametri da passare
   success: function(msg)
   {
      // da eseguire solo dopo l'esito positivo
      // [...]
   }
});
```

Sfruttando le potenzialità della libreria si ottengono gli stessi risultati di un'implementazione in JavaScript puro, ma a carico del programmatore rimane solo la gestione degli eventi a lui interessati e non

la parte tecnica di comunicazione client/server assieme alle differenze dei browser in quanto già presenti nella libreria stessa.

## 3.2 SICUREZZA

Negli uffici della Pubblica Amministrazione vengono in generale trattati dati sensibili dei cittadini. Nel caso specifico dell'Ufficio Immigrazione, sono presenti archiviazioni di dati anagrafici, informazioni sulla posizione del cittadino extracomunitario e documenti ufficiali il cui principale interessato è il richiedente stesso. Da qui la necessità di proteggere le informazioni da occhi indiscreti o da manipolazioni errate dovute a malfunzionamenti dell'applicazione oppure mancata o inadeguata formazione dell'operatore.

Per quanto riguarda quest'ultimo punto, è necessario fornire guide informative sugli strumenti disponibili ed effettuare qualche ora di formazione interna sull'utilizzo dell'applicazione.

La registrazione di eventuali errati utilizzi e l'occultamento di certi dati o funzionalità, invece, possono essere garantiti implementando un sistema di login: questo permette all'applicazione di riconoscere l'operatore ed abilitargli o meno determinate funzioni a seconda dei permessi a lui concessi.

Per l'implementazione del login si sceglie di utilizzare il database per archiviare gli utenti loggati sfruttando un id univoco memorizzato anche nella sessione del browser. La realizzazione del login si appoggia invece su AJAX e jQuery visti in precedenza.

Riducendo il sistema al minimo necessario, è obbligatorio per ciascun operatore (Tabella 3) possedere un identificatore univoco, il suo

cognome e nome ed una password di accesso; questa sarà memorizzata utilizzando una funzione di hash unidirezionale in modo da non poter essere rivelata a terzi.

**Tabella 3:** tblOperatore

| Campo | Tipo | Descrizione |
|---|---|---|
| userID | Numerico | Identificatore univoco dell'operatore |
| surname | Alfabetico | Cognome dell'operatore |
| name | Alfabetico | Nome dell'operatore |
| password | Alfanumerico | Password di accesso al sistema |
| Altri eventuali dati... | | |
| Chiave primaria: userID | | |

Il collegamento tra l'operatore e l'avvenuto login risiede in una seconda tabella (Tabella 4) dove si trovano il codice operatore, la sessione ed un timestamp comprovante l'istante in cui è stata effettuata l'ultima operazione. Memorizzare data e ora serve per essere in grado di riconoscere i lunghi periodi d'inattività e quindi richiedere una nuova autenticazione prima di proseguire oltre.

**Tabella 4:** tblSessione

| Campo | Tipo | Descrizione |
|---|---|---|
| sessionID | Alfanumerico | Identificatore univoco della sessione |
| userID | Numerico | Identificatore univoco dell'operatore |
| timestamp | Numerico | Data e ora di registrazione della sessione |
| Altri eventuali dati... | | |
| Chiave primaria: sessionID, userID<br>Chiave esterna: userID(tblOperatore) | | |

L'operatore, per iniziare a lavorare, inserisce codice e password ed effettua il login, eseguito in maniera asincrona con il server in modo da ricaricare l'intera pagina solo quando si ottiene l'esito positivo e non per ogni tentativo fallito di effettuare il login.

Un semplice form di autenticazione consiste nell'insieme di due campi testo per codice e password ed un pulsante per l'invio. La

gestione degli eventi è a carico di JavaScript con il quale si controlla l'esistenza di entrambi i campi compilati ed effettua la chiamata al server per il controllo dell'autenticazione (Codici 3.6, 3.7, 3.8).

**Codice 3.6**: Form login

```
<?php session_start(); ?>
<html>
<head>
   <script src="jquery.js" type="text/javascript"></script>
   <script src="login.js" type="text/javascript"></script>
</head>
<body>

<form onsubmit="login(); return false;">
   User ID: <input id="userid" type="text" /><br />
   Password: <input id="password" type="password" /><br />
   <input type="submit" value="Login!" />
</form>
<div id="loginError" style="display:none"></div>

</body>
</html>
```

**Codice 3.7**: login.js

```
// [...]

function login()
{
   var user = $.trim($("#userid").val());
   var pass = $.trim($("#password").val());
   if (user == "" || pass == "")
   {
      $("#loginError").html("userID/password non presenti").show();
   }
   else
   {
      $("#loginError").hide();
      $.ajax({
         type: "POST",
         url: "login.php",
         data: "userid=" + user + "&password=" + pass,
         success: function(msg){
            if (msg == "")
            {
               location = "index.php";
```

```
                exit();
            }
            else
            {
                $("#loginError").html(msg).show();
            }
        }
    });
}
// [...]
```

**Codice 3.8**: login.php

```
<?php

session_start();

// connessione al database
// [...]

$sql = "
      SELECT *
      FROM tblOperatore
      WHERE userID = {$_POST['userid']}
         AND password = 'sha1({$_POST['password']})'";
$res = mysql_query($sql);

if (mysql_num_rows($res) == 0)
   echo "Login fallito";
else
{
   // si suppone sessionID fisso in quanto
   // non oggetto di studio in questo esempio
   $_SESSION['sessionID'] = 'uvs76vsd';
   $user = mysql_fetch_assoc($res);
   $sql = "
      REPLACE INTO tblSessione(sessionID, userID, timestamp)
      VALUES('{$_SESSION['sessionID']}', {$user['userID']}, time())";
   mysql_query($sql);
}

// chiusura connessione
// [...]

?>
```

L'utilizzo di jQuery e di file script esterni consente di avere un codice più pulito e di più facile manutenzione. Si noti la presenza del tag <*div*> inizialmente nascosto ed attivato solo se la risposta di autenticazione contiene un esito negativo.

La verifica dei dati viene effettuata lato server e l'esito viene ritornato alla pagina chiamante. Se l'autenticazione è riuscita, il server provvede a registrare l'utente e ne traccia le attività fino all'azione di logout o prolungata inattività.

All'interno del browser lo script attende la risposta ed all'occorrenza segnala l'errore all'operatore o consente l'accesso alla pagina principale. Quest'ultima, come tutte le pagine protette, deve garantire che l'autenticazione sia stata effettuata e di conseguenza è necessario un controllo prima di visualizzarne i contenuti. Il riscontro tra i dati inseriti dall'operatore e quelli memorizzati nel database consente l'accesso all'applicazione ed il timestamp viene aggiornato (Codici 3.9, 3.10).

**Codice 3.9**: Home page

```
<?php session_start(); ?>
<html>
<head>
   <script src="jquery.js" type="text/javascript"></script>
   <script src="login.js" type="text/javascript"></script>
</head>
<body>

<script language="javascript">
   $.ajax({
      url: "loginCheck.php",
      async: false,
      success: function(msg){
         if (msg != "Logged")
         {
            location = "formlogin.php";
            exit();
         }
      }
   });
```

```
</script>

Qui la mia home page.


</body>
</html>
```

**Codice 3.10:** loginCheck.php

```
<?php

session_start();

// connessione al database
// [...]

$timeLimit = time() - 60*30; // 30 minuti
$sql = "
      SELECT *
      FROM tblSessione
      WHERE sessionID = '{$_SESSION['sessionID']}'
         AND timestamp > {$timeLimit}";
$res = mysql_query($sql);

if (mysql_num_rows($res) > 0)
   echo "Logged";

// chiusura connessione
// [...]

?>
```

Un'ulteriore precauzione in ambito di sicurezza è la suddivisione delle sezioni accessibili e la gestione delle operazioni permesse al singolo operatore; per ottenere ciò si memorizzano in una tabella tutti i permessi con un nome univoco ed un codice numerico costituito da una potenza di due. Questo meccanismo permette di tenere una variabile numerica nei dati dell'operatore la quale, tramite una semplice decodifica, permette di risalire a tutti i permessi posseduti. Oltre a questa gestione è previsto che per utenti selezionati non ci siano re-

strizioni di alcun genere e quindi che questi vengano qualificati come amministratori (Tabelle 5, 6; Codici 3.11, 3.12).

**Tabella 5:** tblOperatore aggiornata con permessi

| Campo | Tipo | Descrizione |
|---|---|---|
| userID | Numerico | Identificatore univoco dell'operatore |
| surname | Alfabetico | Cognome dell'operatore |
| name | Alfabetico | Nome dell'operatore |
| password | Alfanumerico | Password di accesso al sistema |
| perms | Numerico | Tutti i permessi posseduti |
| admin | Booleano | Indicatore del livello di amministrazione |
| Altri eventuali dati... | | |
| Chiave primaria: sessionID, userID | | |
| Chiave esterna: userID(tblOperatore) | | |

**Tabella 6:** tblPermesso

| Campo | Tipo | Descrizione |
|---|---|---|
| name | Alfabetico | Identificatore univoco del permesso |
| value | Numerico | Valore del permesso |
| Altri eventuali dati... | | |
| Chiave primaria: name | | |

**Codice 3.11:** Utilizzo dei permessi

```
<?php

$userA = 1; // si suppone sia un operatore qualsiasi
$userB = 2; // si suppone sia un operatore amministratore

echo "Utente A<br />";
echo "Permesso xxx: ";
if (checkUserPerm($userA, 'xxx'))
   echo "abilitato";
else
   echo "non abilitato";

echo "<hr />";

echo "Utente B<br />";
echo "Permesso xxx: ";
if (checkUserPerm($userB, 'xxx'))
   echo "abilitato";
```

```
else
   echo "non abilitato";

?>
```

**Codice 3.12**: Controllo permesso abilitato

```
<?php

// si suppone sia presente una connessione attiva al database

function checkUserPerm($user, $perm)
{
   $sql = "SELECT * FROM tblOperatore WHERE userID={$user}";
   $res = mysql_query($sql);
   if ($res !== true)
      return false;

   $us = mysql_fetch_assoc($res);
   if ($us['admin'])
      return true;
   $usperm = $us['perms'];

   $sql = "SELECT * FROM tblPermesso WHERE Name='{$perm}'";
   $res = mysql_query($sql);
   if ($res !== true)
      return false;
   $p = mysql_fetch_assoc($res);
   for ($i = 32; $usperm > 0; --$i)
   {
      $pow = pow(2, $i)
      if ($pow <= $usperm)
      {
         if ($pow == p['value'])
            return true;
         $usperm -= $pow;
      }
   }
   return false;
}

?>
```

Per garantire un livello minimo di sicurezza ed un'attività di supervisione in caso di contestazioni operative, è buona norma avere a disposizione un sistema di log delle attività ritenute più delicate op-

pure, qualora si desideri, registrare qualsiasi tipo di azione intrapresa all'interno dell'applicazione.

Il raggiungimento di questo scopo si ottiene salvando nel database i dati necessari per una traccia delle attività svolte. Questi dati, oltre a quelli essenziali come operatore, azione, giorno e ora, possono essere scelti a seconda del tipo di livello d'informazione che si vuole salvare, ad esempio memorizzare anche l'indirizzo IP della macchina sulla quale opera l'utente (Tabella 7).

**Tabella 7:** tblLog

| Campo | Tipo | Descrizione |
|---|---|---|
| timestamp | Numerico | Data e ora di registrazione della sessione |
| userID | Numerico | Identificatore univoco dell'operatore |
| action | Alfanumerico | Messaggio dell'evento accaduto |
| IPaddress | Alfanumerico | Indirizzo IP della macchina esecutrice |
| Altri eventuali dati... | | |
| Chiave primaria: timestamp, userID<br>Chiave esterna: userID(tblOperatore) | | |

Come per le altre operazioni viste in precedenza, anche i log possono essere salvati in un processo parallelo a quello operativo, cioè facendo comunicazioni asincrone al server e di conseguenza senza gravare sui tempi di risposta dell'applicazione all'operatore, il quale può continuare il suo lavoro ignaro di quanto viene svolto in secondo piano.

A seconda del tipo di log è possibile scegliere se registrare i singoli accessi alla pagina richiesta (Codice 3.13) oppure su determinate azioni, come l'eliminazione di un record (Codice 3.14). Si evidenziano anche i file aggiuntivi lato client (Codice 3.15) e lato server (Codice 3.16).

**Codice 3.13**: Log accesso pagina

```
<html>
<head>
   <script src="jquery.js" type="text/javascript"></script>
   <script src="log.js" type="text/javascript"></script>
</head>
<body onload="log('page');">

Contenuto della pagina

</body>
</html>
```

**Codice 3.14**: Log singola azione

```
<html>
<head>
   <script src="jquery.js" type="text/javascript"></script>
   <script src="log.js" type="text/javascript"></script>
   <script language="javascript">
      function delRecord()
      {
         if (confirm("Sicuro?") == true)
         {
            log('deleted');
         }
      }
   </script>
</head>
<body>

Contenuto della pagina

<input type="button" value="Cancella!" onclick="delRecord()" />

</body>
</html>
```

**Codice 3.15:** log.js

```
// [...]

function log(action)
{
   $.ajax({
      type: "POST",
      url: "logRegister.php",
      data: "action=" + action,
      success: function(msg){
      }
   });
}

// [...]
```

**Codice 3.16:** logRegister.php

```
<?php

session_start();

// connessione database
// [...]

$time = time();
$action = trim($_POST['action'];
$sql = "INSERT INTO tblLog(timestamp, userID, action, IPaddress)
      VALUES({$time}, {$_SESSION['user']}, '{$action}',
             '{$_SERVER['HTTP_HOST']}')";
$mysql_query($sql);

//chiusura connessione database
// [...]

?>
```

In questo capitolo non si analizzano i dati dei log raccolti, ma l'elaborazione e ricerche sugli stessi viene lasciata per una futura estensione dell'applicazione.

## 3.3 LA STRUTTURA

Viste le basi della tecnologia AJAX ed un suo possibile utilizzo in ambito di sicurezza, si vogliono utilizzare le sue potenzialità per rendere più veloce l'elaborazione dei dati.

A tal fine è necessario che la pagina sia strutturata in modo da consentire aggiornamenti localizzati dei contenuti suddividendo il corpo della pagina in quattro aree distinte (Figura 2):

**Figura 2:** Suddivisione corpo pagina


**HEADER** è l'intestazione della pagina dove si trovano il titolo ed un orologio digitale realizzato con javascript (Codice 3.17)

**Codice 3.17:** main.php: header

```
<div id="header">
   <div id="headTime"></div>
   <script type="text/javascript">
      updateHeadTime();
   </script>
   <div id="headTitle">
```

```
      <h1>{$page['headTitle']}</h1>
    </div>
  </div>
```

**NAVIGATION** è l'area dove sono presenti i link di navigazione per accedere alle varie sezioni del progetto (Codice 3.18)

**Codice 3.18:** main.php: navigation

```
  <div id="navigation">
    <ul>
      <li class="link" onclick="navGoto('home')">
        <a href="#" onclick="return false;">Home</a>
      </li>
      <li class="link" onclick="navGoto('foreigners')">
        <a href="#" onclick="return false;">Stranieri</a>
      </li>
      <li class="link" onclick="navGoto('help')">
        <a href="#" onclick="return false;">Aiuto</a>
      </li>
EOF;
if (isset($_SESSION['sessionID']) && $_SESSION['sessionID'] != "")
  echo <<<EOF
      <li class="link" onclick="logout()">
        <a href="#" onclick="return false;">Logout</a>
      </li>
EOF;
echo <<<EOF
    </ul>
    <div style="clear:both"></div>
  </div>
```

**CONTENT** parte centrale della pagina dove sono visualizzati i contenuti della sezione scelta nel pannello di navigazione (Codice 3.19)

**Codice 3.19:** main.php: content

```
  <div id="content">
EOF;
```

```
require_once("template/{$page['content']}.php");
echo <<<EOF
   </div>
```

**FOOTER** chiusura della pagina con indicazione dell'eventuale copyright (Codice 3.20)

**Codice 3.20:** main.php: footer

```
   <div id="footer">
      v 1.0 &copy; 2011 Developed by Alan Giacomin
   </div>
</body>
</html>
EOF;
?>
```

Si definiscono, inoltre, nell'intestazione del codice HTML (Codice 3.21) il titolo della pagina ed i percorsi dei file dei fogli di stile e degli script; si noti la presenza del file *jquery.js* il quale consente l'utilizzo dell'omonima libreria.

**Codice 3.21:** main.php: head

```
<?php
echo <<<EOF
<html>
<head>
   <title>{$page['metaTitle']}</title>
   <link rel="stylesheet" type="text/css"
         href="template/css/style.css" />
   <script src="template/js/jquery.js"
           type="text/javascript"></script>
   <script src="template/js/utils.js"
           type="text/javascript"></script>
   <script src="template/js/login.js"
           type="text/javascript"></script>
   <script src="template/js/foreigners.js"
           type="text/javascript"></script>
</head>
<body>
```

Per garantire una migliore organizzazione dei sorgenti, le funzioni javascript per l'orologio (Codice 3.22) e la navigazione (Codice 3.23) sono inserite nel file esterno *utils.js*.

**Codice 3.22**: Gestione orologio

```
var month = Array("Gennaio", "Febbraio", "Marzo", "Aprile",
      "Maggio", "Giugno", "Luglio", "Agosto",
      "Settembre", "Ottobre", "Novembre", "Dicembre");
var day = Array("Domenica", "Luned&igrave;", "Marted&igrave;",
                "Mercoled&igrave;", "Gioved&igrave;",
                "Venerd&igrave;", "Sabato");
function updateHeadTime()
{
   var data = new Date();
   var h = data.getHours() < 10 ? "0"
         + data.getHours() : data.getHours();
   var m = data.getMinutes() < 10 ? "0"
         + data.getMinutes() : data.getMinutes();
   var s = data.getSeconds() < 10 ? "0"
         + data.getSeconds() : data.getSeconds();
   var out = day[data.getDay()] + " ";
   out += data.getDate() + " ";
   out += month[data.getMonth()];
   out += " " + data.getFullYear();
   out += "      ";
   out += h + ":" + m + ":" + s;
   $("#headTime").html(out);
   setTimeout("updateHeadTime()", 1000);
}
```

**Codice 3.23**: Gestione navigazione

```
function navGoto(link)
{
   if (link == "home")
   {
      location = "index.php";
   }
   else
   {
      $.ajax({
         type: "POST",
         url: "index.php?ind="+link+"&ajax=1",
         success: function(msg){
            $('#content').html(msg);
         }
```

```
15        });
    }
}
```

Realizzata la base grafica è necessaria la creazione del punto d'accesso dell'applicazione web, nel caso specifico il file *index.php*, nel quale si specificano le basi comuni per l'applicazione ed è presente la logica per restituire al chiamante i contenuti richiesti.

Si vedono ora in dettaglio le varie funzioni presenti in detto file.

Come prima cosa è necessario includere il file delle impostazioni, cioè variabili per il settaggio del progetto, come ad esempio i parametri per la connessione al database o l'url di base del progetto (Codici 3.24, 3.25).

**Codice 3.24:** index.php: impostazioni

```
<?php

// File inclusion
require_once("settings.php");
```

**Codice 3.25:** settings.php

```
<?php

$baseurl = "http://localhost/Tesi";
$root = "C:\\xampp\htdocs\Tesi";
5
$db_server = "localhost";
$db_user = "root";
$db_pass = "";
$db_name = "Tesi";
10 $db_type = "mysql";

?>
```

Successivamente viene effettuata la connessione al database, la quale consente l'accesso ai dati e di conseguenza la loro elaborazione (Codice 3.26). La classe *Database* è riportata nell'appendice A.1.

**Codice 3.26:** index.php: connessione al database

```
// Database connection
require_once("include/db{$db_type}.php");
$db = new Database($db_server, $db_user, $db_pass, $db_name);
$db->connect();
```

Si prosegue poi con la creazione di un contenitore per i dati della pagina e memorizzando in esso il titolo della finestra del browser e quello dell'intestazione della pagina (Codice 3.27).

**Codice 3.27:** index.php: dati pagina

```
// Page data
$page = array();
$page['metaTitle'] = "Progetto Tesi";
$page['headTitle'] = "Tesi";
$page['content'] = "";
```

Nella sezione 3.2 è stato implementato un sistema di login ed il relativo controllo della sessione per concedere la navigazione all'interno del sito. In questa fase, ne viene preso il funzionamento base analizzato in precedenza ed adattato alla struttura generale del progetto, verificando nel file index l'eventuale sessione presente consentendo quindi la prosecuzione nella navigazione o reindirizzando l'operatore alla pagina di login (Codice 3.28).

**Codice 3.28:** index.php: controllo sessione valida

```
$ind = isset($_GET['ind']) ? trim($_GET['ind']) : "";

// Session checking
$timeLimit = time() - 60*30; // 30 minutes
if (!isset($_SESSION['userID']) || $_SESSION['userID'] <= 0)
   $_SESSION['userID'] = -1;
if (!isset($_SESSION['sessionID']) || $_SESSION['sessionID'] == "")
   $_SESSION['sessionID'] = "";
$sql = "DELETE
        FROM tblSessione
        WHERE timestamp < {$timeLimit}";
$res = $db->query($sql);
```

```
$sql = "SELECT *
        FROM tblSessione
        WHERE sessionID = '{$_SESSION['sessionID']}'
           AND userID = {$_SESSION['userID']}";
$res = $db->query($sql);
if (!in_array($ind, array("login", "loginCheck", "logout"))
    && $db->num_rows($res) == 0)
{
   $_SESSION = null;
   session_destroy();
   if (isset($_GET['ajax']) && $_GET['ajax'] == 1)
   {
      echo "<script language=\"javascript\">
            location.href='index.php';
            </script>
            <noscript>
            <meta http-equiv=\"refresh\" content=\"0; url=\"index.php\">
            </noscript>
      ";
      exit();
   }
   else
   {
      session_start();
      $ind = "login";
   }
}
else
{
   $res = $db->qInsert("UPDATE", "tblSessione", array(
         "timeStamp:int" => time(),
         ),
         "sessionID='{$_SESSION['sessionID']}'
          AND userID={$_SESSION['userID']}");
}
```

Come ultima impostazione viene analizzato l'eventuale parametro *ind* passato in modalità GET per determinare la pagina richiesta (Codice 3.29).

**Codice 3.29:** index.php: selezione contenuto pagina

```
// Content evaluation
switch($ind)
{
   case "":
   case "home":
      $page['content'] = "home"; break;
```

```
    // login
    case "login":
       $page['content'] = "login"; break;
    case "loginCheck":
       $page['content'] = "loginCheck"; break;
    case "logout":
       $page['content'] = "logout"; break;
    // foreigners
    case "foreigners":
       $page['content'] = "foreigners"; break;
    case "foreignersNew":
       $page['content'] = "foreignersNew"; break;
    case "foreignersEdit":
       $page['content'] = "foreignersEdit"; break;
    case "foreignersSearch":
       $page['content'] = "foreignersSearch"; break;
    case "foreignersDocument":
       $page['content'] = "foreignersDocument"; break;
    // help (non implementato in questo elaborato)
    case "help":
       $page['content'] = "help"; break;
    default:
       $page['content'] = "error404";
}
if (!file_exists("source/{$page['content']}.php"))
    $page['content'] = "error404";
```

A questo punto la pagina richiesta, o quella d'errore qualora fosse inesistente, viene mandata in esecuzione e visualizzato il template associato, ponendo attenzione al fatto che la pagina sia quella principale o sia stata invocata da una chiamata ajax (Codice 3.30).

**Codice 3.30:** index.php: esecuzione pagina

```
// Page execution
require_once("source/{$page['content']}.php");
if (!isset($_GET['ajax']) || $_GET['ajax'] != 1)
    require_once("template/main.php");
else if (file_exists("template/{$page['content']}.php"))
    require_once("template/{$page['content']}.php");
else
{
    $page['error404']['ind'] = "template/{$page['content']}";
    require_once("template/error404.php");
}
?>
```

Siccome l'applicazione consiste di molti file, ma solamente l'index è quello direttamente richiamato dall'operatore, è necessario imporre un blocco all'interno del codice, oltre ad una buona configurazione del server web ospitante per la determinazione dei permessi d'accesso alle cartelle. Per ottenere ciò, è sufficiente definire una costante nell'index (Codice 3.31) e controllarne la presenza in tutti gli altri file (Codice 3.32).

**Codice 3.31:** Definizione della costante di sicurezza

```
<?php
   define("TESI", 1);
   // [...]
?>
```

**Codice 3.32:** Controllo della costante di sicurezza

```
<?php
   if (!defined("TESI"))
      die("Hacking...");
   // [...]
?>
```

## 3.4 GESTIONE DATI

Volendo applicare queste tecnologie alla gestione dei cittadini extracomunitari è necessario prima analizzare i dati che si vogliono trattate.

Trattandosi di persone, è facile immaginare che siano indispensabili i dati anagrafici dell'interessato, quali il cognome, il nome, la data di nascita e la nazionalità, oltre al numero del passaporto, di cui è obbligatorio il possesso da parte sua, la data di rilascio e di scadenza. A

questi si possono poi aggiungere quelli relativi alla domanda per il permesso di soggiorno e quindi si memorizzano anche il numero dell'assicurata, ovvero l'identificativo presente sulla ricevuta postale in suo possesso, la tipologia di permesso desiderato e la data di richiesta. Trattandosi di persone straniere, non tutte sono in possesso di codice fiscale, ritenuto univoco per ciascuna persona, ed i soli dati di cognome, nome e data di nascita non sono sufficienti per l'identificazione certa di uno straniero senza indurre in errore e quindi si necessita l'introduzione di un identificatore numerico autoincrementante per garantire l'univocità della persona in questione (Tabella 8).

**Tabella 8:** tblStraniero

| Campo | Tipo | Descrizione |
|---|---|---|
| foreignID | Numerico | Identificato univoco della persona |
| surname | Alfanumerico | Cognome della persona |
| name | Alfanumerico | Nome della persona |
| dateBirth | Numerico | Data di nascita della persona |
| nation | Alfanumerico | Nazione di nascita della persona |
| passportNum | Alfanumerico | Numero del passaporto |
| passportDate | Numerico | Data di rilascio del passaporto |
| passportExpire | Numerico | Data di scadenza del passaporto |
| assicurata | Alfanumerico | Identificatore ricevuta postale |
| permitTypeID | Numerico | Tipo permesso di soggiorno richiesto |
| permitReasonID | Numerico | Motivo del permesso di soggiorno |
| dateRequest | Numerico | Data richiesta permesso di soggiorno |
| Altri eventuali dati... | | |
| Chiave primaria: foreignID<br>Chiave esterna: permitTypeID(tblPermTipo), permitReasonID(tblPermMotivo) | | |

Per una migliore gestione dei dati ed in previsione di una futura implementazione di un pannello di amministrazione, si utilizzano due tabelle accessorie per il tipo di permesso richiesto (Tabella 9) e la motivazione (Tabella 10).

Definiti i dati d'interesse, si vedono le operazioni da effettuare con essi.

**Tabella 9:** tblPermTipo

| Campo | Tipo | Descrizione |
|---|---|---|
| permitTypeID | Numerico | Identificatore univoco del tipo di permesso di soggiorno |
| description | Alfanumerico | Descrizione del tipo di permesso di soggiorno |
| Altri eventuali dati... | | |
| Chiave primaria: permitTypeID | | |

**Tabella 10:** tblPermMotivo

| Campo | Tipo | Descrizione |
|---|---|---|
| permitReasonID | Numerico | Identificatore univoco del motivo del permesso di soggiorno |
| description | Alfanumerico | Descrizione del motivo del permesso di soggiorno |
| Altri eventuali dati... | | |
| Chiave primaria: permitReasonID | | |

Trattandosi di anagrafiche, è lecito supporre operazioni di inserimento, di modifica e di ricerca delle stesse, con conseguente realizzazione delle pagine grafiche per l'operatore, le quali comprendono anche quella di semplice visualizzazione dei dati o di una lista di risultati ottenuti dalla ricerca.

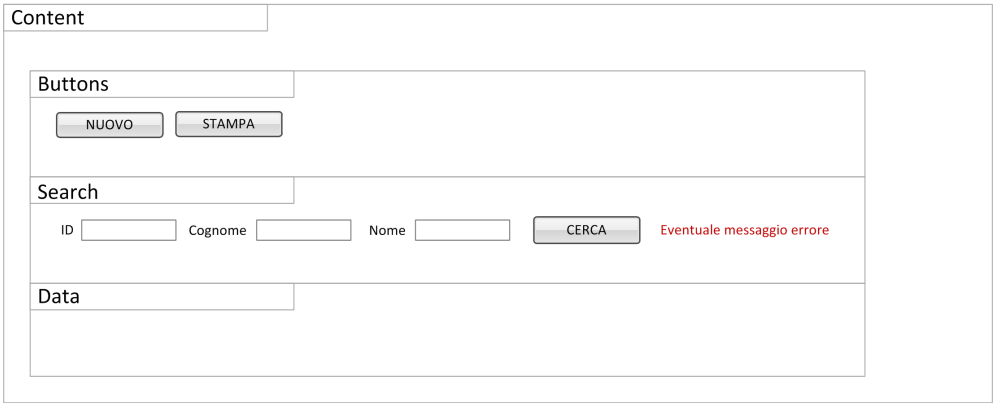
La pagina per la gestione dei dati, come già realizzato per la struttura principale, viene suddivisa in aree per consentire minime modifiche alla stessa a seconda degli eventi.

In particolare, è prevista una sezione per funzioni generali, come la richiesta di creazione di una nuova anagrafica o la stampa della pagina[1], una per i campi della ricerca ed una terza per visualizzare all'occorrenza quanto necessario per svolgere le proprie operazioni (Figura 3; Codice 3.33).

Inizialmente la sezione dei dati è vuota in quanto non è stata ancora intrapresa alcuna azione.

1 Non oggetto di questo elaborato.

**Figura 3**: Corpo per pagina stranieri




**Codice 3.33**: Corpo per pagina stranieri

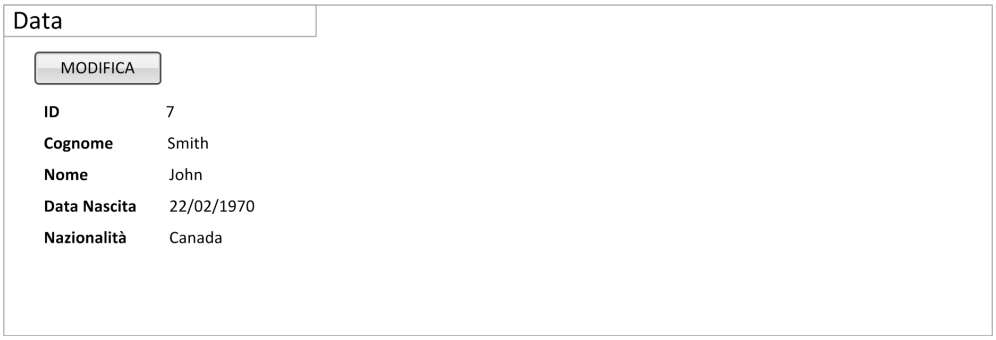
```
<?php
echo <<<EOF
      <h3>Stranieri</h3>
      <div id="foreignButtons">
         <input type="button" class="button" value="NUOVO"
                id="newForeigner" onclick="newForeigner()" />
         <input type="button" class="button" value="STAMPA"
                onclick="printForeigner()" />
      </div>
      <div id="foreignSearch">
         <form action="#" method="post"
               onsubmit="searchForeigners(); return false;">
            ID <input type="text" id="fID" />
            Cognome <input type="text" id="fSurname" />
            Nome <input type="text" id="fName" />
            <input type="submit" value="Cerca" />
            <span id="searchError"></span>
         </form>
         <script language="javascript">
            $("#fID").focus();
         </script>
      </div>
      <div id="foreignData">
      </div>
EOF;
?>
```

I contenuti saranno determinati dagli eventi scatenati dai controlli presenti nella pagina. In dettaglio, la creazione di una nuova anagrafica per lo straniero deve presentare una scheda vuota e abilitata all'inserimento dei valori nei singoli campi. Tramite appositi pulsanti è possibile confermare e salvare i dati[2] oppure annullare l'operazione in corso (Figura 4).

**Figura 4:** Creazione nuova anagrafica


La conferma spedisce i dati al server e se l'esito ritornato all'operatore è positivo viene visualizzata la pagina definitiva con i dati appena salvati (Figura 5).

**Figura 5:** Visualizzazione anagrafica




La modifica dei dati ripropone all'utente una maschera uguale a quella di nuova creazione con la differenza che questa è precompilata

2 Per non dilungare nel codice, gli esempi si limitano ai soli cognome, nome, data di nascita, nazionalità e identificatore della persona.

con i dati già presenti nel database. Per non realizzare molteplici strutture simili si ricorre ancora all'uso di jQuery definendo un'unica pagina per i dati gestita poi dal browser del client (Codice 3.34).

**Codice 3.34:** Anagrafica stranieri

```
<?php
echo <<<EOF
         <form action="#" method="post"
               onsubmit="saveForeigner(); return false;">
            <input type="button" class="button view" value="MODIFICA"
                   onclick="editForeigner()" />
            <input type="button" class="button edit" value="ANNULLA"
                   onclick="cancelForeigner()" />
            <input type="submit" class="button edit" value="SALVA" />
            <div id="foreignError">
            </div>
            <table>
               <tr id="fIDrow">
                  <td class="bold">ID</td>
                  <td>
                     <span id="editID">
                        {$page['foreignData']['ID']}
                     </span>
                  </td>
               </tr>
               <tr>
                  <td class="bold">Cognome</td>
                  <td>
                     <span class="view">
                        {$page['foreignData']['Surname']}
                     </span>
                     <span class="edit">
                        <input type="text" id="editSurname"
                            value="{$page['foreignData']['Surname']}" />
                        <span id="editSurnameError"></span>
                     </span>
                  </td>
               </tr>
               <tr>
                  <td class="bold">Nome</td>
                  <td>
                     <span class="view">
                        {$page['foreignData']['Name']}
                     </span>
                     <span class="edit">
                        <input type="text" id="editName"
                               value="{$page['foreignData']['Name']}" />
```

```
                        <span id="editNameError"></span>
                     </span>
                  </td>
               </tr>
               <tr>
                  <td class="bold">Data nascita</td>
                  <td>
                     <span class="view">
                        {$page['foreignData']['DateBirth']}
                     </span>
                     <span class="edit">
                        <input type="text" id="editDateBirth"
                          value="{$page['foreignData']['DateBirth']}" />
                        <span id="editDateBirthError"></span>
                     </span>
                  </td>
               </tr>
               <tr>
                  <td class="bold">Nazionalit&agrave;</td>
                  <td>
                     <span class="view">
                        {$page['foreignData']['Nation']}
                     </span>
                     <span class="edit">
                        <input type="text" id="editNation"
                             value="{$page['foreignData']['Nation']}" />
                        <span id="editNationError"></span>
                     </span>
                  </td>
               </tr>
            </table>
         <form>
EOF;
?>
```

All'interno del file esterno *foreigners.js* sono presenti le funzioni per la gestione dei dati dei cittadini extracomunitari.

La modifica, come previsto, deve solo trasformare la pagina visualizzando i campi editabili al posto di quelli fissi (Codice 3.35).

**Codice 3.35:** Script: modifica anagrafica straniero

```
function editForeigner()
{
   $(".view").hide(); $(".edit").show();
}
```

La creazione, invece, va effettuata richiedendo la pagina base al server ed adattandola poi localmente per consentirne l'inserimento dei dati (Codice 3.36).

**Codice 3.36:** Script: nuova anagrafica straniero

```
function newForeigner()
{
   $.ajax({
      type: "POST",
      url: "index.php?ind=foreignersNew&ajax=1",
      data: "exec=0",
      success: function(msg){
         $('#foreignData').html(msg);
         isNewForeigner = true;
         $(".view").hide();
         $(".edit").show();
         $("#editSurname").focus();
      }
   });
}
```

L'annullamento dell'operazione si comporta in due modi differenti: se si esegue un inserimento cancella la sezione dei dati, mentre in sede di modifica restituisce i dati come in origine (Codice 3.37).

**Codice 3.37:** Script: annullo operazione

```
function cancelForeigner()
{
   if (isNewForeigner)
   {
      isNewForeigner = false;
      $('#foreignData').html("");
   }
   else
   {
      $("#fIDrow").show();
      $(".view").show();
      $(".edit").hide();
   }
}
```

Un caso particolare si verifica in fase di salvataggio dei dati. Viene controllato localmente che i dati siano corretti[3] altrimenti viene segnalato l'errore e non si procede oltre (Codice 3.38).

**Codice 3.38:** Script: controllo dati

```
function saveForeigner()
{
   fSurname = $.trim($("#editSurname").val());
   fName = $.trim($("#editName").val());
   fDateBirth = $.trim($("#editDateBirth").val());
   fNation = $.trim($("#editNation").val());
   if (checkFData())
      // [...]
}
function checkFData()
{
   var fOK = true;
   if (fSurname == "")
   {
      $("#editSurnameError").html("Errore").show();
      $("#editSurname").focus();
      fOK = false;
   }
   else
   {
      $("#editSurnameError").hide();
   }
   if (fName == "")
   {
      $("#editNameError").html("Errore").show();
      if (fOK)
         $("#editName").focus();
      fOK = false;
   }
   else
   {
      $("#editNameError").hide();
   }
   if (fDateBirth == "")
   {
      $("#editDateBirthError").html("Errore").show();
      if (fOK)
         $("#editDateBirth").focus();
      fOK = false;
   }
```

3 Per semplicità viene controllato solamente che i campi siano tutti compilati.

```
    else
    {
       $("#editDateBirthError").hide();
    }
    if (fNation == "")
    {
       $("#editNationError").html("Errore").show();
       if (fOK)
          $("#editNation").focus();
       fOK = false;
    }
    else
    {
       $("#editNationError").hide();
    }
    return fOK;
}
```

Questo permette un risparmio di tempo dovuto al minor numero di connessioni effettuate. In assenza di errori si procede con due connessioni: la prima con i dati per il salvataggio e la seconda per visualizzare la pagina finale se l'operazione ha avuto successo. È necessario quindi attendere il termine della prima, effettuando una chiamata sincrona, per poi poter effettuare la seconda (Codice 3.39).

**Codice 3.39**: Script: salvataggio dati

```
function saveForeigner()
{
   saveError = true;
   saveMsg = "";
   saveID = 0;
   fSurname = $.trim($("#editSurname").val());
   fName = $.trim($("#editName").val());
   fDateBirth = $.trim($("#editDateBirth").val());
   fNation = $.trim($("#editNation").val());
   if (checkFData())
   {
      if (isNewForeigner)
      {
         $.ajax({
            type: "POST",
            url: "index.php?ind=foreignersNew&ajax=1",
            data: "exec=1&fSurname=" + fSurname + "&fName=" + fName
                  + "&fDateBirth=" + fDateBirth + "&fNation=" + fNation,
```

```
            async: false,
            success: function(msg){
               saveMsg = msg;
               if (msg > 0 && msg < 999999)
               {
                  saveError = false;
                  saveID = msg;
                  isNewForeigner = false;
               }
            }
         });
      }
      else
      {
         fID = $("#editID").html().trim();
         $.ajax({
            type: "POST",
            url: "index.php?ind=foreignersEdit&ajax=1",
            data: "exec=1&fID=" + fID + "&fSurname=" + fSurname
                  + "&fName=" + fName + "&fDateBirth=" + fDateBirth
                  + "&fNation=" + fNation,
            async: false,
            success: function(msg){
               saveMsg = msg;
               if (msg == "OK")
               {
                  saveError = false;
                  saveID = fID;
               }
            }
         });
      }
      if (!saveError)
      {
         $.ajax({
            type: "POST",
            url: "index.php?ind=foreignersSearch&ajax=1",
            data: "type=byID&fID=" + saveID,
            success: function(msg2){
               $('#foreignData').html(msg2);
               $(".view").show();
               $(".edit").hide();
            }
         });
      }
      else
         $('#foreignError').html(saveMsg);
   }
}
```

Il server, da parte sua, alla richiesta di salvataggio esegue un inserimento dei dati nell'archivio (Codice 3.40) oppure un aggiornamento di quelli già presenti (Codice 3.41).

**Codice 3.40**: Inserimento dati

```
<?php

   $foreignersNew['exec'] = trim($_POST['exec']);
   if ($foreignersNew['exec'] == 1)
   {
      $fSurname = trim($_POST['fSurname']);
      $fName = trim($_POST['fName']);
      $fDateBirth = trim($_POST['$fDateBirth']);
      $fNation = trim($_POST['$fNation']);

      $res = $db->qInsert("", "tblStraniero", array(
            "Surname:string" => $fSurname,
            "Name:string" => $fName,
            "DateBirth:int" => mktime(0, 0, 0,
                                substr($fDateBirth, 3, 2),
                                substr($fDateBirth, 0, 2),
                                substr($fDateBirth, 6, 4)),
            "Nation:string" => $fNation,
      ));
      $foreignersNew['fID'] = ($res !== false) ? $db->get_id() : 0;
   }

?>
```

**Codice 3.41**: Aggiornamento dati

```
<?php

   $foreignersEdit['exec'] = trim($_POST['exec']);

   if ($foreignersEdit['exec'] == 1)
   {
      $fID = trim($_POST['fID']);
      $fSurname = trim($_POST['fSurname']);
      $fName = trim($_POST['fName']);
      $fDateBirth = trim($_POST['$fDateBirth']);
      $fNation = trim($_POST['$fNation']);

      $res = $db->qInsert("UPDATE", "tblStraniero", array(
            "Surname:string" => $fSurname,
            "Name:string" => $fName,
```

```
            "DateBirth:int" => mktime(0, 0, 0,
                                substr($fDateBirth, 3, 2),
                                substr($fDateBirth, 0, 2),
                                substr($fDateBirth, 6, 4)),
            "Nation:string" => $fNation,
            ),
            "foreignID=$fID");

        $foreignersEdit['fOK'] = ($res !== false) ? "OK" = "";
    }

?>
```

Visti i dettagli della pagina anagrafica, si procede con l'analisi del funzionamento della ricerca. Si vuole consentire la ricerca per ID in modo da offrire l'accesso diretto alla pagina interessata oppure fornendo cognome e nome; per ridurre il numero di caratteri digitati, fonte di errori, si fa in modo che solo il cognome sia obbligatorio e la ricerca avviene per similitudine, cioè è restituita la lista delle persone che hanno solo la parte iniziale del cognome e l'eventuale nome congruenti con quanto immesso nei campi di ricerca. Nel caso si volesse effettuare una ricerca senza valorizzare i parametri, un avviso avverte l'operatore della mancanza dei dati, mentre la presenza contemporanea dell'ID e del cognome comporta una ricerca per ID essendo questa prioritaria.

La gestione della priorità della ricerca e dell'eventuale errore è interamente demandata ad uno script locale, il quale, in assenza di errori, effettua in seguito una chiamata asincrona per l'esecuzione della ricerca sul server (Codice 3.42).

**Codice 3.42:** Ricerca straniero (script)

```
function searchForeigners()
{
    fID = $.trim($("#fID").val());
    fSurname = $.trim($("#fSurname").val());
    fName = $.trim($("#fName").val());
```

```
    if (fID != "")
    {
       $("#searchError").text("");
       $.ajax({
          type: "POST",
          url: "index.php?ind=foreignersSearch&ajax=1",
          data: "type=byID&fID=" + fID,
          success: function(msg){
             $('#foreignData').html(msg);
             $("#foreignData").show();
             $(".view").show();
             $(".edit").hide();
          }
       });
    }
    else if (fSurname != "")
    {
       $("#searchError").text("");
       $.ajax({
          type: "POST",
          url: "index.php?ind=foreignersSearch&ajax=1",
          data: "type=byName&fSurname=" + fSurname
                + "&fName=" + fName,
          success: function(msg){
             $('#foreignData').html(msg);
             $("#foreignData").show();
             $(".view").show();
             $(".edit").hide();
          }
       });
    }
    else
    {
       $("#searchError").text("Inserire ID o Cognome");
    }
}
```

Quando il server riceve una richiesta di ricerca, recupera i parametri passati in modalità POST ed effettua un'interrogazione al database per ottenere i dati. A seconda del numero di record compatibili con i parametri di ricerca, vengono preparati i dati per la pagina della singola anagrafica oppure per l'elenco da visualizzare all'operatore (Codice 3.43).

**Codice 3.43:** Ricerca straniero (server)

```
<?php
$type = isset($_POST['type']) ? trim($_POST['type']) : "";
$fID = isset($_POST['fID']) ? trim($_POST['fID']) : "";
$fSurname = isset($_POST['fSurname']) ? trim($_POST['fSurname']) : "";
$fName = isset($_POST['fName']) ? trim($_POST['fName']) : "";


switch ($type)
{
   case "byID":
   {
      $sql = "SELECT *
               FROM tblStraniero
               WHERE foreignID = {$fID}";
      $res = $db->query($sql);
      $page['foreignersList']['count'] = $db->num_rows($res);
      if ($page['foreignersList']['count'])
      {
         oneForeign($res);
      }
      else
      {
         $page['foreignersList']['list'] = array();
         $page['content'] = "foreignersList";
      }

   } break;
   case "byName":
   {
      $sql = "SELECT *
               FROM tblStraniero
               WHERE surname Like '$fSurname%'
                  AND name Like '$fName%'";
      $res = $db->query($sql);
      $page['foreignersList']['count'] = $db->num_rows($res);
      if ($page['foreignersList']['count'] == 1)
      {
         oneForeign($res);
      }
      else
      {
         $page['foreignersList']['list'] = array();
         while ($f = $db->fetch($res))
         {
            $page['foreignersList']['list'][] = array(
                  'foreignID' => $f['foreignID'],
                  'surname' => $f['surname'],
                  'name' => $f['name'],
```

```
            );
        }
        $page['content'] = "foreignersList";
      }
   } break;
}
function oneForeign($res)
{
   global $db, $page;

   $fData = $db->fetch($res);
   $page['foreignData']['ID'] = $fData['foreignID'];
   $page['foreignData']['Surname'] = $fData['surname'];
   $page['foreignData']['Name'] = $fData['name'];
   $page['foreignData']['DateBirth'] = date("d/m/Y",
                                           $fData['dateBirth']);
   $page['foreignData']['Nation'] = $fData['nation'];
   $page['content'] = "foreignData";
}
?>
```

Nel caso della lista per le molteplici corrispondenze, all'operatore viene proposta una tabella con i risultati ottenuti e la possibilità di accedere alla pagina interessata tramite un pulsante (Figura 6; Codice 3.44).

**Figura 6**: Lista anagrafiche trovate


**Codice 3.44**: Lista anagrafiche trovate

```
<?php
echo <<<EOF
      <p>Trovati: {$page['foreignersList']['count']}</p>
      <table>
```

```
         <tr>
            <th></th>
            <th>ID</th>
            <th>Cognome</th>
            <th>Nome</th>
         </tr>
EOF;
foreach ($page['foreignersList']['list'] as $f)
{
   echo <<<EOF
         <tr>
            <td>
               <input type="button" value="Apri"
                      onclick="openForeigner({$f['foreignID']})" />
            </td>
            <td>{$f['foreignID']}</td>
            <td>{$f['surname']}</td>
            <td>{$f['name']}</td>
         </tr>
EOF;
}
echo <<<EOF
      </table>
EOF;
?>
```

Anche in questo caso la selezione comporta una nuova connessione al server simulando una ricerca per ID sapendo a priori che un esito esiste sicuramente (Codice 3.45).

**Codice 3.45**: Apertura anagrafica scelta

```
function openForeigner(id)
{
   $.ajax({
      type: "POST",
      url: "index.php?ind=foreignersSearch&ajax=1",
      data: "type=byID&fID=" + id,
      success: function(msg){
         $('#foreignData').html(msg);
         $(".view").show();
         $(".edit").hide();
      }
   });
}
```

A questo punto si è ottenuta un'applicazione funzionante comprensiva di una gestione anagrafica dei cittadini extracomunitari.

Le migliorie introdotte, come le numerose elaborazioni script effettuate in locale e le ridotte connessioni con limitate dimensioni dei dati trasmessi, consentono quindi una migliore resa nel lavoro dell'operatore ed un servizio funzionale con la soddisfazione dell'utente.

## 3.5 GESTIONE DOCUMENTI

Gli uffici della Pubblica Amministrazione compilano molti documenti, sia da rilasciare agli utenti, sia per semplice uso interno. I modelli sono ricercati tra le varie tipologie disponibili, adattati dagli operatori al caso del bisogno e compilati con i dati richiesti. Nascono quindi archivi di modelli personali differenti tra gli operatori portando così all'interno dell'ufficio diversi tipi di documenti per lo stesso uso.

Una possibile soluzione è l'impiego di sistemi automatizzati per la compilazione dei documenti. A tal fine si utilizzano documenti in formato RTF ed un parser per la loro compilazione.

### 3.5.1 Rich Text Format

I documenti Rich Text Format (spesso abbreviato con RTF) sono file di testo i quali sono riconosciuti anche da editor avanzati come MS Word. Per una migliore compatibilità, il formato RTF utilizza la codifica ASCII a 7 bit, mentre gli altri caratteri sono espressi indicando

il loro valore unicode con due cifre esadecimali (tipo \'*ef* per i-dieresi).

I documenti RTF sono composti di comandi e raggruppamenti, oltre al testo semplice. Il corretto utilizzo di questi elementi consente di creare documenti con una formattazione pari a quella di Microsoft Word, ma privi di oggetti come le macro.

I comandi sono particolari sequenze di caratteri precedute dal carattere backslash e servono principalmente per formattare la pagina o il testo. I comandi hanno validità dal momento della loro dichiarazione fino alla fine del gruppo in cui sono inseriti. Fanno eccezione i comandi istantanei, come *page* con il quale s'impone il passaggio ad una nuova pagina.

Un gruppo è una porzione di documento al cui interno vi si possono trovare altri gruppi, comandi o testo. È delimitato da parentesi graffe e nell'intero documento devono essere presenti tante parentesi chiuse quante ne sono state inserite di aperte, altrimenti il documento non è valido e risulta impossibile da leggere.

Un semplice documento RTF è considerato valido quando è composto di un gruppo che racchiude tutto il contenuto del file. Il primo comando, inoltre, deve necessariamente essere *rtf1* ad indicare il formato del documento e la relativa versione di utilizzo.

### 3.5.2 Parser

Un parser è un programma che analizza una sequenza di dati, digitati alla tastiera o provenienti da un file, per ottenerne la struttura e salvarla per eventuali ispezioni successive. Sono state anche realizzate varianti che lavorano su determinati *token* per spezzare l'input in più parti consentendone un'analisi separata.

Il parser utilizzato in questo elaborato sfrutta l'idea delle varianti, ma elabora solamente il testo che corrisponde a determinati *token*. Considerato l'alfabeto dei caratteri ammessi per i comandi RTF, i *token* saranno costituiti da parole chiave in maiuscolo ed il carattere underscore racchiuse da parentesi quadre, come *[COGNOME]* o *[DATA_NASCITA]*.

Il compito del parser, oltre alle inizializzazioni di base, è quello di leggere l'intero documento, effettuare le sostituzioni dei *token* e salvare il documento finale risultante. Si analizza la funzione di creazione del documento (Codice 3.46), rimandando all'appendice A.2 per la classe intera.

**Codice 3.46**: Creazione documento

```
<?php

class Parser
{
   // [...]

   function createDocument()
   {
      if (!file_exists($this->fileInput))
         return self::NO_INPUT_FILE;

      $this->fileInputText = file_get_contents($this->fileInput);
      $this->fileOutputText = preg_replace_callback(
            "/\[([^\[\]]*)\]/",
            create_function(
               '$matches',
               '
                  $ret = "";
                  for ($i = 0; $i < strlen($matches[1]); ++$i)
                  {
                     $c = $matches[1]{$i};
                     if ($c == "_" || ($c >= "A" && $c <= "Z"))
                        $ret .= $c;
                  }
                  return "[$ret]";
               '
            ),
            $this->fileInputText);
```

```
30        foreach ($this->replaceArray as $key => $value)
          {
             $value = preg_replace("/\r\n|\n\r|\r|\n|<br \/>|<br>/i",
                                   "\par ", $value);
             $this->fileOutputText = preg_replace("/\[$key\]/",
35                                             $value, $this->fileOutputText);
          }
          file_put_contents($this->fileOutput, $this->fileOutputText);

          return self::NO_ERROR;
40     }
   }
   ?>
```

La funzione *file_get_contents* recupera l'intero contenuto del file RTF in una variabile. Le parole chiave potrebbero essere nitide nel documento come anche interrotte da comandi RTF per la formattazione e quindi irriconoscibili. Si rende quindi necessaria la pulizia dei *token* considerando validi solo i caratteri definiti per le parole chiave ignorando, di fatto, quelli intermedi. A questo punto è possibile operare la sostituzione ponendo attenzione alle varie forme possibili per andare a capo riga le quali devono essere sostituite con il corrispondente comando *\par*. Il file viene infine salvato a lavoro ultimato.

### 3.5.3 Un documento: il biglietto d'invito

Lo straniero, in sede di convocazione presso l'Ufficio Immigrazione, è tenuto a consegnare tutti i documenti richiesti. Quelli presentati, a volte, sono errati o non sufficienti e per questo motivo l'operatore è costretto a fissare un secondo appuntamento rilasciando un biglietto d'invito quale atto formale. Un esempio di tale documento è disponibile in figura 7.

**Figura 7:** Biglietto d'invito (Compilato)

# INTESTAZIONE UFFICIO

indirizzo ufficio

Prot. xxx\yyy\zzz Luogo, 11/01/2009

OGGETTO: Lettera invito

Con la presente si invita il/la cittadino/a extracomunitario **John Smith** nato il 22/02/1970 in Canada a presentarsi presso l'ufficio in intestazione il giorno **18/01/2009** alle ore **10.30** per le seguenti motivazioni:

portare copia contratto assunzione

Il dirigente
*Nome dirigente*

Partendo da questo, è necessario estrarne il modello base adattabile a tutte le necessità. Sono presenti i dati anagrafici dello straniero, il giorno e l'ora del nuovo appuntamento e le relative motivazioni. Questi dati devono essere rappresentati da *token* secondo la sintassi esposta in precedenza. Il modello risultante è come da figura 8.

**Figura 8:** Biglietto d'invito (Modello)

# INTESTAZIONE UFFICIO

indirizzo ufficio

Prot. xxx\yyy\zzz Luogo, 11/01/2009

OGGETTO: Lettera invito

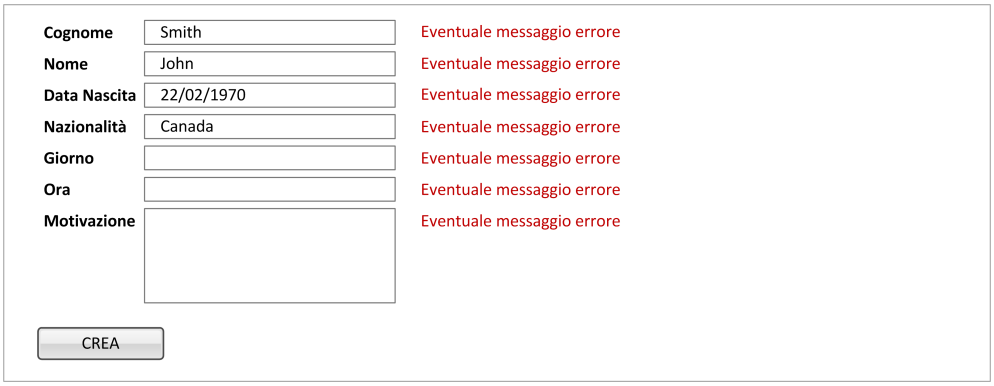
Con la presente si invita il/la cittadino/a extracomunitario **[COGNOME] [NOME]** nato il [DATA_NASCITA] in [NAZIONALITA] a presentarsi presso l'ufficio in intestazione il giorno **[GIORNO]** alle ore **[ORA]** per le seguenti motivazioni:

[MOTIVAZIONI]

Il dirigente
*Nome dirigente*

Dopo aver definito tutti i *token*, si realizza nell'applicazione una sezione per la compilazione di dati. Alcuni di questi sono legati a informazioni già in possesso e quindi vengono precompilati risparmiando tempo all'operatore (Figura 9; Codice 3.47).

**Figura 9:** Form dati per il documento




**Codice 3.47:** Form dati per il documento

```
<?php
echo <<<EOF
         <div class="view">
            <input type="button" class="button" value="BIGL. INVITO"
                   onclick="$('#appointment').show();" />
         </div>

         <div id="appointment" style="display:none">
            <h2>Biglietto d'invito</h2>
            <form class="view" action="#" method="post"
                  onsubmit="createDocument(); return false;">
               <input type="hidden" id="docID"
                      value="{$page['foreignData']['ID']}" />
               <input type="hidden" id="docType" value="invito" />
               <table>
                  <tr>
                     <td class="bold">Cognome</td>
                     <td>
                        <input type="text" id="docSurname"
                          value="{$page['foreignData']['Surname']}" />
                     </td>
                  </tr>
                  <tr>
                     <td class="bold">Nome</td>
                     <td>
                        <input type="text" id="docName"
                             value="{$page['foreignData']['Name']}" />
                     </td>
                  </tr>
                  <tr>
                     <td class="bold">Data di nascita</td>
                     <td>
```

```
                        <input type="text" id="docDateBirth"
                        value="{$page['foreignData']['DateBirth']}" />
                     </td>
                  </tr>
                  <tr>
                     <td class="bold">Nazionalit&agrave;</td>
                     <td>
                        <input type="text" id="docNation"
                           value="{$page['foreignData']['Nation']}" />
                     </td>
                  </tr>
                  <tr>
                     <td class="bold">Giorno</td>
                     <td>
                        <input type="text" id="docDay" />
                     </td>
                  </tr>
                  <tr>
                     <td class="bold">Ora</td>
                     <td>
                        <input type="text" id="docHour" />
                     </td>
                  </tr>
                  <tr>
                     <td class="bold">Motivazioni</td>
                     <td>
                        <textarea id="docReason"
                                  rows="7" cols="40"></textarea>
                     </td>
                  </tr>
                  <tr>
                     <td colspan="2">
                        <input type="submit" value="CREA" />
                     </td>
                  </tr>
               </table>
            </form>
         </div>
EOF;

?>
```

Al termine della compilazione, la conferma dei dati da parte dell'operatore pone inizio alla creazione del documento in due fasi. La prima viene eseguita localmente dal browser tramite uno script, il quale recupera tutti i dati e li invia al server sfruttando ancora le connessioni asincrone (Codice 3.48).

**Codice 3.48**: Chiamata creazione documento

```
function createDocument()
{
   fID = $.trim($("#docID").val());
   fSurname =$.trim( $("#docSurname").val());
   fName = $.trim($("#docName").val());
   fDateBirth = $.trim($("#docDateBirth").val());
   fNation = $.trim($("#docNation").val());
   dDay = $.trim($("#docDay").val());
   dHour = $.trim($("#docHour").val());
   dReason = $.trim($("#docReason").val());
   dType = $.trim($("#docType").val());
   $.ajax({
      type: "POST",
      url: "index.php?ind=foreignersDocument&ajax=1",
      data: "exec=1&dType=" + dType + "&fID=" + fID
            + "&fSurname=" + fSurname + "&fName=" + fName
            + "&fDateBirth=" + fDateBirth + "&fNation=" + fNation
            + "&dDay=" + dDay + "&dHour=" + dHour
            + "&dReason=" + dReason,
      success: function(msg){
         if (msg.substr(0, 4) == "http")
            var w = window.open(msg);
         else
            $('#foreignData').html(msg);
      }
   });
}
```

La seconda fase viene eseguita dal server il quale ha il compito di mandare in esecuzione il parser per il documento richiesto con i dati ricevuti dalla macchina dell'operatore. Terminata l'elaborazione, viene ritornato al richiedente un puntatore sotto forma di link al file creato oppure un messaggio in caso si presentassero errori (Codice 3.49).

**Codice 3.49**: Utilizzo del parser

```
<?php
require_once("include/parser.php");

$foreignersDocument['exec'] = trim($_POST['exec']);
if ($foreignersDocument['exec'] == 1)
{
```

```
    $fID = trim($_POST['fID']);
    $dType = trim($_POST['dType']);
    $fSurname = trim($_POST['fSurname']);
    $fName = trim($_POST['fName']);
    $fDateBirth = trim($_POST['fDateBirth']);
    $fNation = trim($_POST['fNation']);
    $dDay = trim($_POST['dDay']);
    $dHour = trim($_POST['dHour']);
    $dReason = trim($_POST['dReason']);

    $parser = new Parser($baseurl, $root, "docs");
    $parser->setFileInput($dType);
    $parser->setReplacings(array(
       "COGNOME" => strtoupper($fSurname),
       "NOME" => $fName,
       "DATA_NASCITA" => $fDateBirth,
       "NAZIONALITA" => $fNation,
       "GIORNO" => $dDay,
       "ORA" => $dHour,
       "MOTIVAZIONI" => $dReason,
    ));
    $res = $parser->createDocument();
    if ($res != Parser::NO_ERROR)
    {
       $foreignersDocument['link'] = "Parser error: " . $parser->
           getErrorDescription($res);
    }
    else
    {
       $foreignersDocument['link'] = $parser->getFileOutput();
    }
}

?>
```

Analogamente alle altre pagine dell'applicazione, è presente un file per la visualizzazione delle informazioni da ritornare all'operatore (Codice 3.50).

**Codice 3.50**: Risultato del parser

```
<?php

if ($foreignersDocument['exec'] == 1)
   echo $foreignersDocument['link'];
?>
```

Il browser, alla ricezione della risposta, ne analizza il contenuto ed all'occorrenza visualizza un messaggio di errore all'operatore oppure apre il documento appena creato. L'apertura del documento può avere comportamenti differenti a seconda delle impostazioni del browser nel quale viene eseguita l'applicazione in quanto non sono modificabili tramite script per motivi di sicurezza. Questo comporta che per alcuni operatori i documenti siano visibili all'interno del browser stesso, per altri si apre automaticamente l'editor associato a quel tipo di documenti oppure ad altri ancora può esser consentito solo il salvataggio di tale file e poi l'apertura è a carico dell'operatore stesso.

# 4 CONCLUSIONI

Con il presente elaborato si sono descritte le procedure per l'ottenimento del permesso di soggiorno da parte dei cittadini extracomunitari. Tali procedure prevedono l'analisi di fascicoli e la gestione delle pratiche; queste fasi all'interno degli uffici della Pubblica Amministrazione sono spesso inefficienti. Attraverso l'analisi degli strumenti in uso e delle metodologie di lavoro, ritenute non ottimali, si è voluto offrire un'innovazione tecnologica per ovviare a questi problemi. Si sono quindi studiate le varie tecnologie del mondo web ed attraverso alcuni esempi è stata realizzata un'applicazione web esemplificativa di quanto è possibile creare. Nella realizzazione si è dovuto tenere conto delle differenze tra i vari browser, le quali hanno presentato ostacoli in fase d'implementazione degli script; tali difficoltà sono state tenute sotto controllo attraverso l'utilizzo di librerie adattative e lasciando quindi al programmatore l'incarico di effettuare dei test per verificare che alcuni controlli non siano rimasti non funzionanti.

Il progetto implementato presso l'Ufficio Immigrazione della Questura ove è stato svolto il tirocinio è stato realizzato con l'impiego delle nozioni descritte nei capitoli precedenti ed è attualmente operativo dal mese di Febbraio 2010. Il tirocinio si è concluso con grande soddisfazione dell'Ufficio per gli enormi benefici cui fruisce attraverso la nuova applicazione, come anche del laureando per aver fornito un così valido strumento di lavoro.

# A APPENDICE

## A.1 CLASSE DATABASE

**Codice A.1:** include/dbmysql.php

```
<?php
if (!defined("TESI"))
   die("Hacking...");

class Database
{
   var $conn;
   var $server;
   var $user;
   var $pass;
   var $name;
   var $query_count;
   var $debug;

   function Database($server, $user, $pass, $name)
   {
      $this->server = $server;
      $this->user = $user;
      $this->pass = $pass;
      $this->name = $name;
      $this->query_count = 0;
      $this->debug = array();
   }

   function connect()
   {
      $this->conn = mysql_connect($this->server,
                                  $this->user, $this->pass);
      if (!$this->conn)
         die ("No connection available");
      $this->debug[] = "connected: {$this->user}@{$this->server}" .
                       "(password:" .
                       ($this->pass == "" ? "no" : "yes") . ")";
```

```
}

function close()
{
   mysql_close($this->conn);
}

function query($sql)
{
   $conndb = mysql_select_db($this->name, $this->conn);
   if (!$conndb)
      die ("No database selected");
   $this->debug[] = "selected: {$this->name}";

   $ret = mysql_query($sql, $this->conn);
   ++$this->query_count;
   $this->debug[] = "sql: $sql";
   if (mysql_errno())
   {
      die ($this->qError($sql));
   }
   return $ret;
}

function qInsert($qtype, $table, $data, $qwhere = "")
{
   if ($qtype == "")
      $qtype = "INSERT";
   if ($qtype == "INSERT" || $qtype == "REPLACE")
   {
      $sql = "$qtype INTO $table(";
      $fields = array();
      $values = array();
      foreach($data as $k => $v)
      {
         list($f, $t) = explode(":", $k);
         $fields[] = $f;
         $values[] = $t == "string" ? "'$v'" : $v;
      }
      $sql .= implode(", ", $fields);
      $sql .= ") VALUES (";
      $sql .= implode(", ", $values);
      $sql .= ")";
   }
   else if ($qtype == "UPDATE")
   {
      $sql = "UPDATE $table SET ";
      $values = array();
      foreach($data as $k => $v)
      {
```

```
            list($f, $t) = explode(":", $k);
            $values[] = "$f=" . ($t == "string" ? "'$v'" : $v);
         }
         $sql .= implode(", ", $values);
      }
      if ($qwhere != "")
         $sql .= " WHERE $qwhere";
      return $this->query($sql);
   }

   function fetch($res)
   {
      return mysql_fetch_assoc($res);
   }

   function num_rows($res)
   {
      return mysql_num_rows($res);
   }

   function get($res, $row, $field)
   {
      return mysql_result($res, $row, $field);
   }

   function get_query_count()
   {
      return $this->query_count;
   }

   function get_id()
   {
      return mysql_insert_id();
   }

   function escape($str)
   {
      return mysql_real_escape_string(trim($str));
   }

   function getDebug()
   {
      print_array($this->debug);
   }

   function qError($sql)
   {
      $err = mysql_error();
      $errno = mysql_errno();
      preg_match("/'([^']*)'/", $err, $matches);
```

```
        $m = str_replace("(", "\(", $matches[1]);
        $m = str_replace(")", "\)", $m);
        $sql = preg_replace("/($m)/", '<u>${1}</u>', $sql);
        $back = debug_backtrace();
        array_shift($back);
        $ret = "
              <br /><b>MYSQL ERROR</b><br /><b>query</b>: $sql
              <br />
              <b>error</b>: $err<br />
              <b>file</b>: " .
                 substr($back[0]['file'], strlen($_SERVER["DOCUMENT_ROOT
                     "])) . "<br />
              <b>line</b>: " . $back[0]['line'];

        $ret .= <<<EOF
              <br /><br />
              <script language="javascript">
                 function qshow()
                 {
                    if ($('#qerror').css('display') == "none")
                    {
                       $('#qerror').css('display', '');
                       $('#qdetails').attr('value', 'Nascondi');
                    }
                    else
                    {
                       $('#qerror').css('display', 'none');
                       $('#qdetails').attr('value', 'Dettagli');
                    }
                 }
              </script>
              <input type="button" class="button" value="Dettagli"
                     id="qdetails" onclick="qshow()" />
              <div id="qerror" style="display:none; width: 100%;">
              <pre>
EOF;
        $ret .= print_r($back, true);
        $ret .= <<<EOF
              </pre>
              </div>
EOF;

        return $ret;
     }
}
?>
```

## A.2 CLASSE PARSER

**Codice A.2:** include/parser.php

```
<?php
if (!defined("TESI"))
   die("Hacking...");

class Parser
{
   var $baseurl;
   var $root;
   var $docFolder;
   var $docModels;
   var $docOutput;
   var $linkOutput;

   var $fileInput;
   var $fileInputText;
   var $fileOutput;
   var $fileOutputText;
   var $replaceArray;

   const NO_ERROR = 0;
   const NO_INPUT_FILE = 1;

   function Parser($baseurl, $root, $docFolder)
   {
      $this->baseurl = $baseurl;
      $this->root = $root;
      $this->docFolder = $docFolder;
      $this->docModels = "$root/{$docFolder}/models";
      $this->docOutput = "$root/{$docFolder}/output";
      $this->linkOutput = "$baseurl/{$docFolder}/output";

      $this->fileInput = "";
      $this->fileInputText = "";
      $this->fileOutput = "";
      $this->fileOutputText = "";
      $this->replaceArray = array();
   }

   function setFileInput($file)
   {
      $this->fileInput = $this->docModels . "/{$file}.rtf";
      $filename = $file . time() . ".rtf";
      $this->fileOutput = $this->docOutput . "/{$filename}";
```

```
      $this->linkOutput = "{$this->baseurl}/{$this->docFolder}
                           /output/{$filename}";
   }

   function getFileOutput($link = true)
   {
      if ($link)
         return $this->linkOutput;
      else
         return $this->fileOutput;
   }

   function setReplacings($replace)
   {
      $this->replaceArray = $replace;
   }

   function createDocument()
   {
      if (!file_exists($this->fileInput))
         return self::NO_INPUT_FILE;

      $this->fileInputText = file_get_contents($this->fileInput);
      $this->fileOutputText = preg_replace_callback(
            "/\[([^\[\]]*)\]/",
            create_function(
               // single quotes are essential here,
               // or alternative escape all $ as \$
               '$matches',
               '
                  $ret = "";
                  for ($i = 0; $i < strlen($matches[1]); ++$i)
                  {
                     $c = $matches[1]{$i};
                     if ($c == "_" || ($c >= "A" && $c <= "Z"))
                        $ret .= $c;
                  }
                  return "[$ret]";
               '
            ),
            $this->fileInputText);

      foreach ($this->replaceArray as $key => $value)
      {
         $value = preg_replace("/\r\n|\n\r|\r|\n|<br \/>|<br>/i",
                               "\par ", $value);
         $this->fileOutputText = preg_replace("/\[$key\]/", $value,
                                              $this->fileOutputText);
      }
```

```
        file_put_contents($this->fileOutput, $this->fileOutputText);

        return self::NO_ERROR;
    }

    function getErrorDescription($error)
    {
        switch($error)
        {
            case self::NO_ERROR:
                return "No error"; break;
            case self::NO_INPUT_FILE:
                return "No input file"; break;
            default:
                return "Undefined error";
        }
    }

}

?>
```

# B BIBLIOGRAFIA

**MASSIMO CANDUCCI** PHP 5

ed. Apogeo

**SAVERIO RUBINI** JavaScript

ed. Apogeo

**A. ROMAGNOLI, P. SALERNO, A. GUIDI** AJAX per applicazioni web

ed. Apogeo

http://www.w3schools.com

http://jquery.com

http://php.net