

Documentazione POSTIT

1. Registrazione Utente

- **Attori coinvolti:** Utente non registrato
 - **Descrizione:** L'utente inserisce nome utente, password (e altri dati se richiesti) in un modulo di registrazione.
 - **Flusso principale:**
 - L'utente compila il form.
 - Il server controlla che l'utente non esista già.
 - I dati vengono salvati (probabilmente in un file JSON o simile).
 - **Eccezioni:**
 - Username già esistente.
 - Campi mancanti o invalidi.
-

2. Login

- **Attori coinvolti:** Utente registrato
 - **Descrizione:** L'utente inserisce le credenziali nel modulo di login.
 - **Flusso principale:**
 - Il server confronta le credenziali con quelle salvate nel file.
 - Se corrette, crea una sessione (o un token).
 - **Eccezioni:**
 - Username inesistente.
 - Password errata.
-

3. Accesso all'Area Personale

- **Attori coinvolti:** Utente autenticato

- **Descrizione:** L'utente può accedere ad una pagina protetta che mostra i suoi dati personali.
 - **Flusso principale:**
 - Verifica della sessione/token.
 - Lettura dei dati utente dal file.
 - Visualizzazione dati nell'interfaccia.
 - **Eccezioni:**
 - Accesso non autorizzato (sessione scaduta o inesistente).
-



4. Scrittura nella Community

- **Attori coinvolti:** Utente autenticato
 - **Descrizione:** L'utente scrive un messaggio visibile ad altri nella sezione community.
 - **Flusso principale:**
 - Verifica sessione.
 - Salvataggio del messaggio in un file (`conversazioni.json`).
 - Visualizzazione aggiornata per tutti.
 - **Eccezioni:**
 - Tentativo di scrittura senza login.
 - Dati del messaggio vuoti.
-



5. Logout

- **Attori coinvolti:** Utente autenticato
- **Descrizione:** L'utente termina la propria sessione.
- **Flusso principale:**
 - Rimozione della sessione.
 - Redirect alla pagina iniziale.

- **Eccezioni:**
 - Nessuna rilevante (è un'azione sicura).
-

6. Gestione dei Dati sul Server

(non visibile all'utente ma importante da documentare)

- **Attori coinvolti:** Server
- **Descrizione:** Tutti i dati (utenti, messaggi, ecc.) vengono scritti su file.
- **Dettagli:**
 - Registrazione crea o aggiorna `credenziali.json` .
 - Scrittura community aggiorna `conversazioni.json` .
 - Ogni operazione scrive/legge da file in modo asincrono (presumibilmente con i metodi `fs`).

Diagramma dei casi uso ←link

Gestione file

Link alla gestione dei file

Cartelle principali:

- **Server**

File server principale (es. `server.js` o `app.js`) che avvia l'applicazione Express, gestisce le rotte, importa moduli, e connette i file `.ejs` alle view.
- **Credenziali.json**

File contenente dati sensibili come utenti, password o API key. Sperando che tu lo tenga fuori da GitHub, perché sai... privacy.

- **Conversazioni.json**

Probabilmente contiene dati strutturati per gestire conversazioni (chat, messaggi, ecc.). Usato dal server per mostrare o salvare dialoghi.

Cartella View (template .ejs)

- **index.ejs**

Pagina iniziale o homepage dell'applicazione. Potrebbe visualizzare un'introduzione o dashboard base.

- **login.ejs**

Pagina di login, con form per l'inserimento delle credenziali.

- **profilo_log.ejs**

Pagina del profilo quando l'utente ha già effettuato l'accesso. Mostra dati personali.

- **profilo.ejs**

Possibile versione pubblica del profilo, visibile ad altri utenti o prima del login.

- **registrazione.ejs**

Pagina per la creazione di un nuovo account. Contiene un form di registrazione.

Cartella partials

- **navbar.ejs**

è un "partial" (incluso con `<%- include() %>` dentro altri `.ejs`) per riutilizzare la stessa barra di navigazione in più pagine senza duplicare il codice.

Cartella public (file statici)

- **login.css**

Foglio di stile per le pagine di login e registrazione. Quindi sì: “server per la gestione grafica del login e registrazione” è corretto in spirito, anche se il server lo serve, non lo crea.

Cartella Nav

- **navBar.css**


Stili grafici legati alla navbar. Probabilmente associato al `navbar.ejs` incluso nelle varie pagine.

In sintesi:

File/Cartella	Utilizzo
<code>Server</code>	Entry point dell'app, gestisce routing e rendering
<code>Credenziali.json</code>	Contiene dati di accesso o config sensibili
<code>Conversazioni.json</code>	Dati strutturati di conversazioni (chatbot? utenti?)
<code>View/*.ejs</code>	Template delle varie pagine renderizzate dal server
<code>partials/navbar.ejs</code>	Blocco riutilizzabile di codice per navbar
<code>public/login.css</code>	Stili del login e della registrazione
<code>Nav/navbar.css</code>	Stili per la navbar (separati, perché tu ami la confusione)

1. Visualizzazione della Home Page (CU: Home Page)

Codice Server

 [codice completo](#)

```
const express = require('express');
const app = express();
const path = require('path');
const fs = require('fs');
const ll = require('./libreria_login');
const session = require('express-session');
var sessione_attiva = false;
```

Prima di iniziare la parte che chiedi, vediamo brevemente queste righe, per avere il quadro completo:

- **express** : È il framework che stiamo usando per creare il nostro server. Facilita la gestione delle richieste HTTP, le route e la gestione dei middleware.
- **app** : È l'istanza di Express che viene utilizzata per definire le route e configurare il server.
- **path** : Modulo integrato di Node.js che permette di lavorare con i percorsi dei file e delle directory in modo sicuro e cross-platform.
- **fs** : Modulo integrato di Node.js per lavorare con il filesystem (lettura e scrittura di file).
- **ll** : Importazione del modulo personalizzato `libreria_login.js` , probabilmente per la gestione degli utenti (registrazione, login, etc.).
- **session** : Middleware che gestisce le sessioni degli utenti, utile per memorizzare informazioni temporanee come il login.

```
// Impostiamo il body-parser per gestire i dati dei form
app.use(express.urlencoded({ extended: true }));
```

Questa riga di codice è un **middleware** di Express che permette al server di leggere i dati inviati tramite i **moduli HTML** (forms). La funzione `express.urlencoded()` viene utilizzata per decodificare i dati che arrivano in una richiesta **POST** con il tipo di contenuto `application/x-www-form-urlencoded` .

- **extended: true** : Specifica che puoi inviare oggetti e array annidati nei dati del modulo. Se fosse **false**, solo i dati primitivi (stringhe e numeri) sarebbero decodificabili.

```
// Impostiamo EJS come motore di template
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
```

Questa parte riguarda la configurazione del motore di template **EJS** (Embedded JavaScript), che è usato per generare dinamicamente HTML lato server.

app.set('view engine', 'ejs');

- **view engine** : Imposta il motore di template che Express utilizzerà per rendere le viste (le pagine HTML). In questo caso, abbiamo impostato **EJS** come motore di template.
- EJS permette di mescolare codice JavaScript dentro il HTML, in modo che tu possa visualizzare dinamicamente variabili, cicli, condizionali, ecc. direttamente nelle tue pagine.

```
js
CopiaModifica
app.get('/', (req, res) => {
  fs.readFile('conversazioni.json', 'utf8', (err, data) => {
    if (err) {
      console.log('Errore nella lettura del file:', err);
      return res.status(500).send('Errore nel caricamento delle conversazioni.');
```

```

    } catch (e) {
      console.log('File JSON vuoto o malformato, inizializzo un array vuoto.');
```

```

    }

    res.render('index', {
      conversazioni: conversazioni,
      avviso: ""
    });
  });
});
});

```

Cosa fa questa route?

1. Legge il file `conversazioni.json` per recuperare tutte le conversazioni salvate.
2. Se c'è un errore nella lettura del file (ad esempio se il file non esiste), viene restituito un errore HTTP 500.
3. Se il file è valido, **parsa** il contenuto JSON e lo rende disponibile per il rendering della pagina EJS.
4. Rende la **pagina** `index.ejs` passandole:
 - `conversazioni` : L'array di conversazioni letto dal file.
 - `avviso` : Un messaggio che può essere vuoto.

File `.ejs` coinvolto

- `index.ejs` : La pagina che visualizza la home, che contiene le conversazioni lette dal file. Viene renderizzata quando viene chiamata la route `/`.

Comportamento in `index.ejs`

- Se ci sono conversazioni, vengono visualizzate nel formato:

```

html
CopiaModifica

```



```
<p><strong><%= conv.name %>:</strong> <%= conv.txt %></p>
```

- Se non ci sono conversazioni, viene mostrato il messaggio "Nessun messaggio presente."

File JSON

- **conversazioni.json**: Questo file JSON contiene le conversazioni in formato array di oggetti. Ogni oggetto ha:
 - **name**: Nome dell'autore del messaggio.
 - **txt**: Contenuto del messaggio.

CSS

- **navbar/style.css**: Gestisce lo stile della barra di navigazione (parte condivisa tra tutte le pagine).

2. Login (CU: Login)

Codice Server

```
js
CopiaModifica
app.post('/login', (req, res) => {
  const { email, psw } = req.body;
  const utenti = ll.leggiUtenti(); // Funzione per leggere gli utenti registrati
  const utente = utenti.find(user => user.email === email && user.password === psw);

  if (utente) {
    sessione_attiva = true;
    req.session.user = utente;
    res.redirect('/profilo');
  } else {
    res.render('login.ejs', {
```

```
    avviso: "Credenziali sbagliate"
  });
}
});
```

Cosa fa questa route?

1. **Riceve i dati di login** tramite POST (email e password) dal form di login.
2. Utilizza la funzione `leggiUtenti()` della libreria `libreria_login.js` per ottenere tutti gli utenti registrati.
3. **Trova l'utente** che ha l'email e la password corrispondenti.
4. Se l'utente esiste, crea una **sessione** (`req.session.user`) e reindirizza a `/profilo` .
5. Se non viene trovato nessun utente corrispondente, ritorna la pagina `login.ejs` con un messaggio di errore.

File `.ejs` coinvolto

- `login.ejs` : La pagina di login, che contiene il form per inserire email e password.
- **Messaggio di avviso**: Se le credenziali sono errate, viene visualizzato il messaggio di errore `avviso` .

Comportamento in `login.ejs`

- Contiene un modulo con due campi (email e password) per il login e un messaggio di errore che può essere visualizzato sotto il modulo in caso di credenziali sbagliate.
- Ha anche un link per la registrazione (reindirizza a `/registra`).

File JSON

- `credenziali.json` : Non direttamente usato in questa route, ma contiene i dati degli utenti che possono essere letti dalla funzione `leggiUtenti()` .

3. Registrazione Nuovo Utente (CU: Registrazione)

Codice Server

```
js
CopiaModifica
app.post('/salva', (req, res) => {
  const pw = req.body.psw;
  const email = req.body.email;
  const nome = req.body.nome;

  const user = { nome, email, password: pw };

  fs.readFile('credenziali.json', 'utf8', (err, data) => {
    if (err) {
      return res.status(500).send('Errore nel salvataggio delle credenziali.');
```

```
    }

    let users = [];
    try {
      users = JSON.parse(data);
    } catch (e) {
      console.log('File vuoto o non valido, creando un nuovo array.');
```

```
    }

    const emailExists = users.some(user => user.email === email);
```

```
    if (emailExists) {
      res.render('registrazione.ejs', { avviso: "Email già presente" });
    } else {
      users.push(user);
      fs.writeFile('credenziali.json', JSON.stringify(users, null, 2), (err) => {
        if (err) {
          return res.status(500).send('Errore nel salvataggio delle credenziali.');
```

```
        }
        res.render('registrazione.ejs', { avviso: "Email salvata con successo" });
      });
    }
  });
}
```

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

Cosa fa questa route?

1. Riceve i dati della registrazione (nome, email, password) tramite POST.
2. Legge il file `credenziali.json` per ottenere gli utenti già registrati.
3. **Verifica se l'email è già presente.** Se sì, ritorna alla pagina di registrazione con un messaggio di errore.
4. Se l'email non è presente, **aggiunge il nuovo utente** e salva i dati nel file JSON `credenziali.json`.

File `.ejs` coinvolto

- `registrazione.ejs` : La pagina di registrazione che permette agli utenti di inserire nome, email e password. Se l'email è già presente, visualizza il messaggio di errore. Se la registrazione ha successo, visualizza una conferma.

Comportamento in `registrazione.ejs`

- Form di registrazione con:
 - Nome
 - Email
 - Password
 - Un pulsante di invio.
- **Messaggi di avviso:**
 - Se l'email è già registrata, viene mostrato `"Email già presente"`.
 - Se la registrazione è avvenuta correttamente, viene mostrato `"Email salvata con successo"`.

File JSON

- **credenziali.json** : Questo file JSON contiene le informazioni sugli utenti registrati, che vengono letti e aggiornati ogni volta che un nuovo utente si registra.

4. Logout (CU: Logout)

Codice Server

```
js
CopiaModifica
app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) {
      console.error('Errore nel logout:', err);
      return res.redirect('/profilo');
    }
    res.redirect('/login');
  });
});
```

Cosa fa questa route?

1. **Distrugge la sessione** dell'utente corrente.
2. Se c'è un errore nel logout, l'utente viene reindirizzato alla pagina del profilo.
3. In caso di successo, l'utente viene reindirizzato alla pagina di login (**/login**).

5. Profilo Utente (CU: Profilo)

Codice Server

```
js
CopiaModifica
app.get('/profilo', (req, res) => {
  if (req.session.user) {
    const user = req.session.user;
```

```
res.render('profilo_log.ejs', { n: user.nome, e :user.email });
} else {
  res.render('profilo.ejs');
}
});
```

Cosa fa questa route?

1. Se l'utente è **loggato** (verificato tramite la sessione), visualizza la pagina `profilo_log.ejs` con il nome e l'email dell'utente.
2. Se l'utente non è loggato, reindirizza alla pagina `profilo.ejs` (pagina di accesso).

File `.ejs` coinvolto

- `profilo_log.ejs` : Mostra le informazioni del profilo dell'utente, come nome e email, e fornisce un'opzione per il logout.
- `profilo.ejs` : Viene visualizzato se l'utente non è loggato, mostrando solo l'opzione per accedere.

Analisi di tutti i codici ejs

1. `login.ejs`

</> codice completo

```
html
CopiaModifica
<!DOCTYPE html>
<html lang="it">
<head>
```

```

<meta charset="UTF-8">
<title>Login</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH" crossorigin="anonymous">
<link rel="stylesheet" href="/public/login.css">
<script src="/script.js" defer></script> <!-- Carica lo script →
</head>
<body>
<%- include('partials/navbar') %>

```

1. `<!DOCTYPE html>` : Indica al browser che la pagina è un documento HTML5.
2. `<html lang="it">` : Imposta la lingua della pagina come italiano, migliorando l'accessibilità per motori di ricerca e lettori di schermo.
3. `<meta charset="UTF-8">` : Imposta la codifica dei caratteri per la pagina su UTF-8, consentendo la visualizzazione di caratteri speciali e simboli.
4. `<title>Login</title>` : Imposta il titolo della pagina nel tab del browser, che qui è "Login".
5. `<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">` : Carica il file CSS di Bootstrap tramite CDN per la stilizzazione della pagina.
6. `<link rel="stylesheet" href="/public/login.css">` : Carica un file CSS personalizzato per la pagina di login.
7. `<script src="/script.js" defer></script>` : Collega un file JavaScript esterno che verrà eseguito dopo il caricamento della pagina, per aggiungere interattività.

html

CopiaModifica

```

<div class="container d-flex justify-content-center align-items-center">
  <div class="login-container">
    <h3 class="mb-4">Accedi al tuo account</h3>
    <form name="formLogin" method="post" action="/login">

```

```

        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" class="form-control" id="email" name="email" required>
        </div>
        <div class="form-group">
            <label for="psw">Password:</label>
            <input type="password" class="form-control" id="psw" name="password" required>
        </div>
        <button type="submit" class="btn btn-primary">Accedi</button>
    </form>
    <br>
    <div class="alert alert-danger" role="alert">
        <%= avviso %>
    </div>
</div>
</body>
</html>

```

1. `<div class="container d-flex justify-content-center align-items-center">` : Crea un contenitore **Flexbox** di Bootstrap, centrando orizzontalmente e verticalmente il contenuto al suo interno.
2. `<div class="login-container">` : Un div che contiene il modulo di login, con una classe CSS personalizzata per la stilizzazione.
3. `<h3 class="mb-4">Accedi al tuo account</h3>` : Un titolo che introduce la pagina di login. La classe `mb-4` aggiunge un margine inferiore per separarlo dal resto del contenuto.
4. `<form name="formLogin" method="post" action="/login">` : Un modulo di login che invia una richiesta POST al percorso `/login`.
5. `<div class="form-group">` : Ogni campo di input è avvolto in un **gruppo di modulo** per una stilizzazione più ordinata.

6. `<label for="email">Email:</label>` : Un'etichetta che precede il campo di input per l'email.
 7. `<input type="email" class="form-control" id="email" name="email" required>` : Un campo di input per l'email dell'utente, con validazione HTML integrata.
 8. `<label for="psw">Password:</label>` : Un'etichetta che precede il campo di input per la password.
 9. `<input type="password" class="form-control" id="psw" name="psw" required>` : Un campo di input per la password, mascherato per privacy.
 10. `<button type="submit" class="btn btn-primary">Accedi</button>` : Un pulsante per inviare il modulo. Utilizza una classe di **Bootstrap** per la stilizzazione.
 11. `<div class="alert alert-danger" role="alert">` : Un **div** che visualizza eventuali avvisi, come errori di login. La variabile `<%= avviso %>` è una variabile che viene passata dal server per mostrare messaggi di errore, come credenziali sbagliate.
-

2. `index.ejs`

`</>` codice completo

```
html
CopiaModifica
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Home</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <%- include('partials/navbar') %>
```

1. `<!DOCTYPE html>` : Documento HTML5.

2. `<html lang="it">` : Lingua italiana.
3. `<meta charset="UTF-8">` : Codifica UTF-8.
4. `<title>Home</title>` : Titolo della pagina.
5. `<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">` : Bootstrap tramite CDN per la stilizzazione.

html

CopiaModifica

```
<div class="container">
  <h1>Benvenuto nella tua area personale</h1>
  <p>Questa è la tua home page.</p>
  <form name="formTesto" method="post" action="/index">
    <div class="form-group">
      <label for="inputTesto">Inserisci un messaggio:</label>
      <input type="text" class="form-control" id="inputTesto" name="inputTesto" required>
    </div>
    <button type="submit" class="btn btn-primary">Invia</button>
  </form>
  <div>
    <ul>
      <% conversazioni.forEach(function(convo) { %>
        <li><%= convo.name %>: <%= convo.txt %></li>
      <% }); %>
    </ul>
  </div>
</div>
</body>
</html>
```

1. `<div class="container">` : Un contenitore principale per il contenuto della pagina.
2. `<h1>Benvenuto nella tua area personale</h1>` : Titolo di benvenuto per l'utente.

3. `<form name="formTesto" method="post" action="/index">` : Modulo per l'invio di un messaggio.
 4. `<input type="text" class="form-control" id="inputTesto" name="inputTesto" required>` : Un campo di input per il messaggio da inviare.
 5. `<button type="submit" class="btn btn-primary">Invia</button>` : Pulsante per inviare il modulo.
 6. `` : Una lista non ordinata per mostrare le conversazioni precedenti.
 7. `<% conversazioni.forEach(function(convo) { %>` : Cicla attraverso tutte le conversazioni e le visualizza.
 8. `<%= convo.name %>: <%= convo.txt %>` : Mostra il nome e il messaggio di ogni conversazione.
-

3. `profilo_log.ejs`

`</>` codice completo

```
html
CopiaModifica
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Profilo</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <%- include('partials/navbar') %>
```

1. `<!DOCTYPE html>` : Documento HTML5.
2. `<html lang="it">` : Lingua italiana.

3. `<meta charset="UTF-8">` : Codifica UTF-8.
4. `<title>Profilo</title>` : Titolo della pagina.
5. `<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">` : Bootstrap per la stilizzazione.

html

CopiaModifica

```
<div class="container">
  <h1>Benvenuto <%= n %></h1>
  <p>Email: <%= e %></p>
  <form name="formLogout" method="get" action="/logout">
    <button type="submit" class="btn btn-danger">Logout</button>
  </form>
</div>
</body>
</html>
```

1. `<h1>Benvenuto <%= n %></h1>` : Saluto personalizzato all'utente, visualizzando il nome `n`.
2. `<p>Email: <%= e %></p>` : Visualizza l'email dell'utente `e`.
3. `<form name="formLogout" method="get" action="/logout">` : Modulo per effettuare il logout.
4. `<button type="submit" class="btn btn-danger">Logout</button>` : Pulsante di logout.

4. `profilo.ejs`

`</>` codice completo

html

CopiaModifica

```
<!DOCTYPE html>
<html lang="it">
<head>
```

```

<meta charset="UTF-8">
<title>Profilo</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <%- include('partials/navbar') %>

```

1. `<!DOCTYPE html>` : Documento HTML5.
2. `<html lang="it">` : Lingua italiana.
3. `<meta charset="UTF-8">` : Codifica UTF-8.
4. `<title>Profilo</title>` : Titolo della pagina.
5. `<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">` : Bootstrap per la stilizzazione.

html

CopiaModifica

```

<div class="container">
  <h1>Benvenuto, <%= nome %></h1>
  <p>La tua email è: <%= email %></p>
</div>
</body>
</html>

```

1. `<h1>Benvenuto, <%= nome %></h1>` : Visualizza il nome dell'utente `nome` .
2. `<p>La tua email è: <%= email %></p>` : Visualizza l'email dell'utente `email` .

5. `registrazione.ejs`

`</>` [codice completo](#)

```

html
CopiaModifica
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Registrazione</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <%- include('partials/navbar') %>

```

1. `<!DOCTYPE html>` : Documento HTML5.
2. `<html lang="it">` : Lingua italiana.
3. `<meta charset="UTF-8">` : Codifica UTF-8.
4. `<title>Registrazione</title>` : Titolo della pagina.
5. `<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">` : Bootstrap per la stilizzazione.

```

html
CopiaModifica
<div class="container">
  <h1>Crea un nuovo account</h1>
  <form method="post" action="/registrazione">
    <div class="form-group">
      <label for="nome">Nome:</label>
      <input type="text" class="form-control" id="nome" name="nome" required>
    </div>
    <div class="form-group">
      <label for="email">Email:</label>

```

```

        <input type="email" class="form-control" id="email" name="email"
required>
    </div>
    <div class="form-group">
        <label for="psw">Password:</label>
        <input type="password" class="form-control" id="psw" name="ps
w" required>
    </div>
    <button type="submit" class="btn btn-success">Registrati</button>
</form>
</div>
</body>
</html>

```

1. `<form method="post" action="/registrazione">` : Modulo di registrazione che invia i dati via POST.
2. `<input type="text" class="form-control" id="nome" name="nome" required>` : Campo per il nome dell'utente.
3. `<input type="email" class="form-control" id="email" name="email" required>` : Campo per l'email dell'utente.
4. `<input type="password" class="form-control" id="psw" name="psw" required>` : Campo per la password dell'utente.
5. `<button type="submit" class="btn btn-success">Registrati</button>` : Pulsante per inviare il modulo di registrazione.

6. `navbar.ejs` (nella cartella `partials`)

`</>` codice completo

```

html
CopiaModifica
<nav class="navbar navbar-expand-lg navbar-light bg-light">

```

```

<a class="navbar-brand" href="/">Logo</a>
<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
  <span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarNav">
  <ul class="navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="/">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/login">Login</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/registrazione">Registrazione</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/profilo">Profilo</a>
    </li>
  </ul>
</div>
</nav>

```

1. `<nav class="navbar navbar-expand-lg navbar-light bg-light">` : Un **navbar** utilizzando Bootstrap, che si espande su dispositivi più grandi e ha uno sfondo chiaro.
2. `Logo` : Il logo che rimanda alla home page.
3. `<button class="navbar-toggler" type="button" ...>` : Il pulsante che permette di espandere o comprimere il menu di navigazione sui dispositivi mobili.
4. `<ul class="navbar-nav">` : Una lista di link di navigazione.
5. `<li class="nav-item">` : Ogni voce del menu di navigazione, che punta a una delle pagine principali.