RELAZIONE TPSIT

Fabio Monas
4Binfo
2022/2023

SOCIAL POST-IT CODE

Abstract :

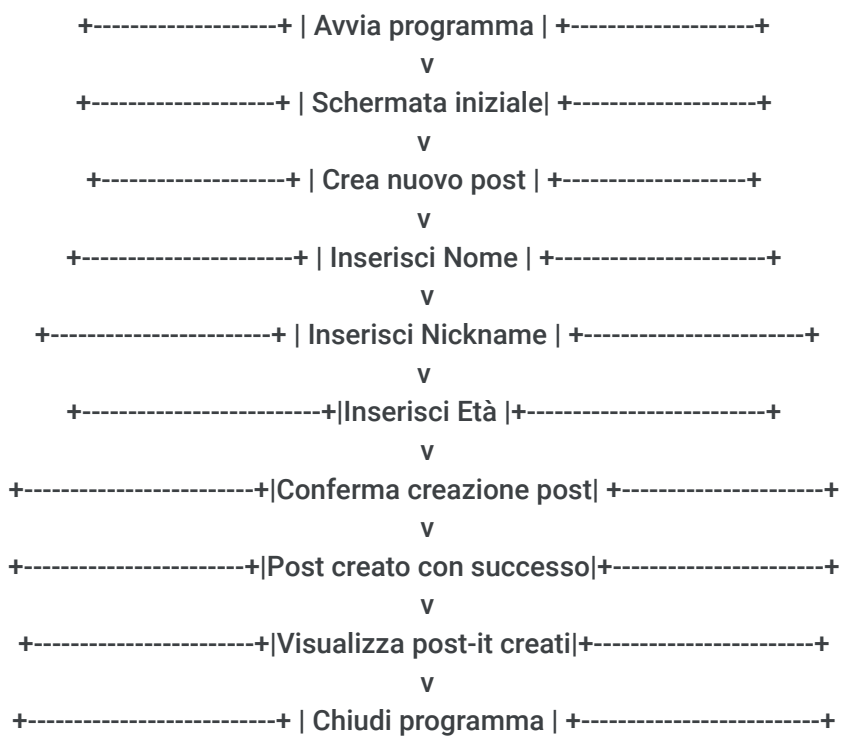
(ITA)
In questa relazione si parlerà di un progetto realizzato in classe nel quale si inseriranno dei dati anagrafici all'interno di un post-it e questo verrà salvato in una schermata apposita dove verranno visualizzati anche tutti gli altri post-it inseriti.
Inoltre,in questa relazione, verrà visualizzato il codice e spiegato il funzionamento di esso tramite dei commenti inseriti accanto alle righe di codice.
Ci saranno, successivamente, le problematiche che si sono riscontrate con relative soluzioni e saranno presenti delle eventuali migliorie che potrebbero aiutare il programma ad essere più efficiente.
Continuando si troveranno i riferimenti teorici e i riferimenti al Github personale, concludendo con il link ad una semplice ma efficace presentazione dove verrà esposto il progetto.

(ENG)
This report will talk about a project carried out in the classroom in which personal data will be entered in a post-it and this will be saved in a special screen where all the other post-its inserted will also be displayed.
Also, in this report, the code will be displayed and its operation explained through comments placed next to the lines of code.
Subsequently, there will be the problems that have been encountered with relative solutions and there will be any improvements that could help the program to be more efficient.
Continuing you will find the theoretical references and references to personal Github,
concluding with the link to a simple but effective presentation where the project will be exhibited.

Introduzione:

In questa relazione si parlerà di un progetto realizzato in classe nel quale ci sarà una tabella contenente i post-it con i relativi nomi e caratteristiche.

Flow diagram



Il programma inizia con la schermata iniziale, dove l'utente può scegliere di creare un nuovo post-it o visualizzare quelli già creati. Se l'utente sceglie di creare un nuovo post-it, il programma richiede di inserire un nome, un nickname e scegliere l'età per il post-it. Dopodiché, l'utente confermerà la creazione del post-it, e il programma mostrerà un messaggio di successo.L'utente può quindi scegliere di visualizzare tutti i post-it creati o chiudere il programma.

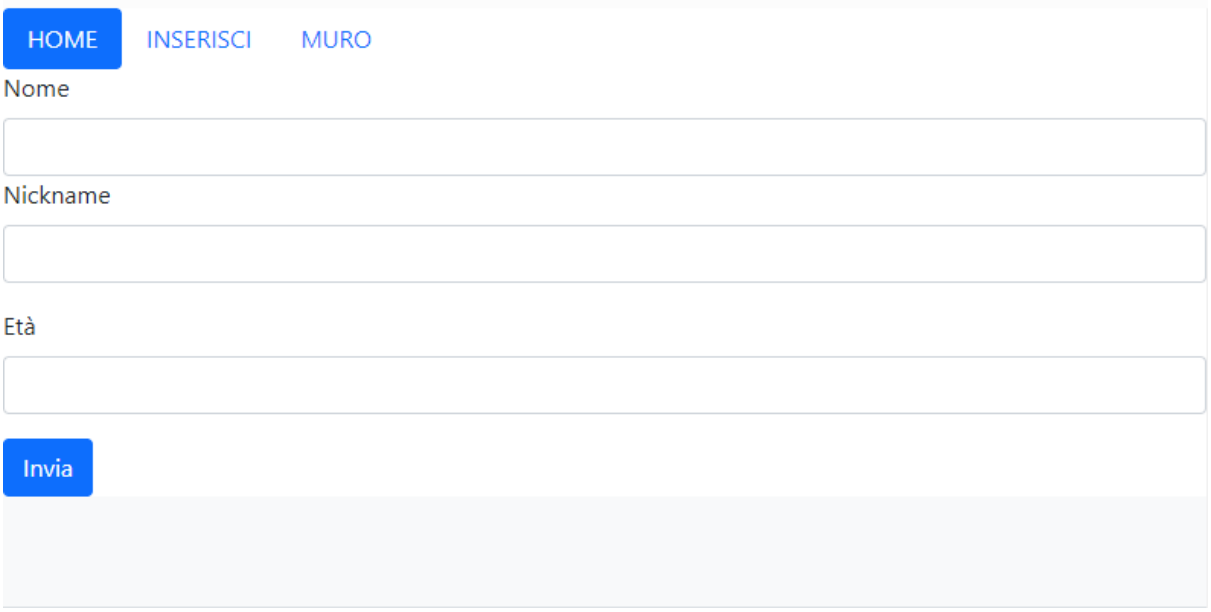
Caratteristiche e spiegazione parti del Progetto:

1. Schermata Home;

Questa è la pagina iniziale dove c'è una frase con Scritto "Benvenuto nella home clicca su Inserisci per aggiungere un post it".

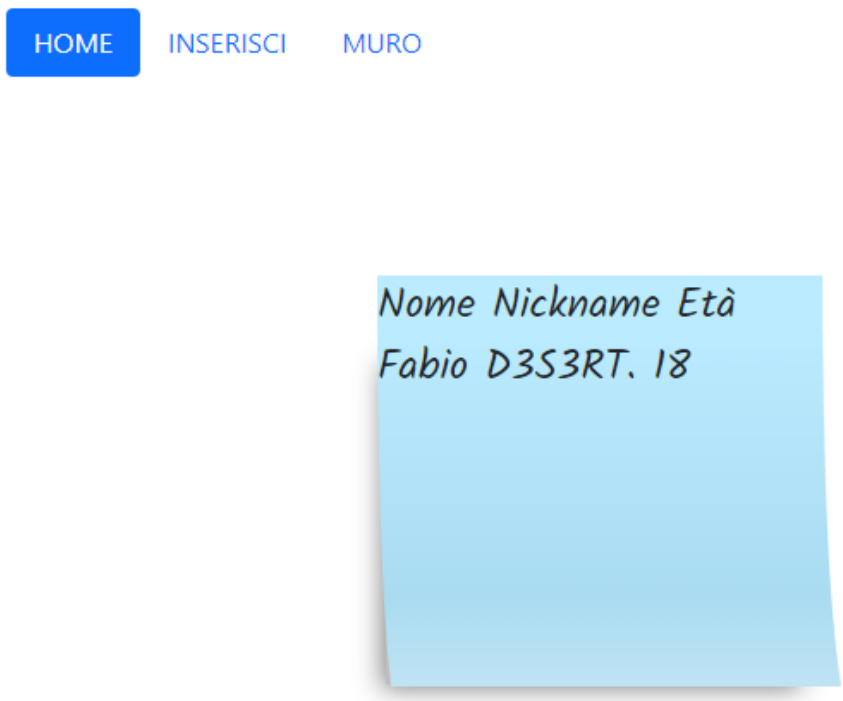
2. Inserimento post-it;

Cliccando sulla voce "Inserisci" Uscirà la seguente immagine questa schermata.



In questa Schermata bisogna inserire il Nome, Nickname e la propria età.
una volta completato il modulo schiacciare su “invia”.
Si ritornerà sulla schermata principale, se si vuole vedere la lista con tutti i post-it bisognerà schiacciare sulla voce “Muro”.

3. Muro dei post-it



Questa è la schermata dove ci saranno tutti i post-it che avremmo inserito.

Codice commentato e descritto :

INDEX.JS

```
1
2  const express = require('express')
3  const path = require('path')
4  const app = express()
5  const ejs = require('ejs')
6  const body = require("body-parser");
7  var utils = require("./mioMonasPellegrino.js");
8  var data;
9
10 app.use(body.urlencoded({ extended: true }))
11
12 app.set("view engine", 'ejs')
13 app.use('/', express.static(__dirname + "/views"));
14 app.set('views', path.join(__dirname, 'views'));
15
16
17 app.get('/', (req, res) => {
18   data = utils.leggiFile("./data/tabella.json");
19   res.render("index.ejs")
20 })
21
22
23 app.get('/classifica', (req, res) => {
24   data = utils.leggiFile("./data/tabella.json");
25
26   console.log(data);
27   res.render("score", { data: data })
28 })
29
30 app.get('/insert', (req, res) => {
31   res.render("insert")
32 })
33
34 app.post('/inserisci', (req, res) => {
35   console.log(data);
36   utils.appendArrayJSON("./data/tabella.json", data, req)
37   res.redirect("/");
38 })
39
40 var port = process.env.PORT || 8080;
41
42 app.listen(port, function() {
43   console.log("Listening on " + port);
44 });
45
```

Questo codice utilizza il framework Express.js per creare un server web che esegue alcune operazioni. In particolare, il codice utilizza il motore di template EJS per generare pagine web dinamiche. Il codice utilizza anche il modulo "body-parser" per analizzare i dati di richiesta.

Il server utilizza il metodo HTTP GET per gestire le richieste sulla radice del server ('/'), sulla pagina di classifica ('/classifica') e sulla pagina di inserimento dati ('/insert'). Quando viene

richiesta la pagina principale, il codice legge i dati di una tabella da un file JSON utilizzando una funzione definita in un modulo esterno. Quando viene richiesta la pagina di classifica, il codice legge nuovamente i dati dalla stessa fonte e li passa al motore di template per generare la pagina.

Quando viene richiesta la pagina di inserimento dati, il codice genera un form HTML che permette all'utente di inserire nuovi dati nella tabella. Quando l'utente invia il form, il server riceve i dati tramite una richiesta POST e utilizza una funzione definita in un modulo esterno per aggiungere i dati alla tabella.

Infine, il server viene avviato sulla porta 8080 (o sulla porta specificata dall'ambiente di esecuzione) e viene stampato un messaggio sulla console per indicare che il server è in ascolto.

Proseguiamo con il file “mioMonasPellegrino.js”:

```
var fs = require('fs');
var data = new Date();


function scriviFile(percorso, file) {
  fs.writeFile(percorso, file, function(err) {
    if (err) throw err;
    console.log("File scritto con successo");
  });
}


function leggiFile(percorsoFile) {
  var data;
  data = fs.readFileSync(percorsoFile, "utf8", (err, dati) => {
    if (err) {
      console.error(err);
      return;
    } else {
      return dati;
    }
  });
  return JSON.parse(data);
}


function appendArrayJSON(percorsoFile, fileJSON, req) {
  var file = "[";
  length = Object.keys(fileJSON).length;

  for (var i = 0; i < length; i++) {
    file += '{ "Nome": "' + fileJSON[i].Nome + '", "Nickname": "' + fileJSON[i].Nickname + '", "Età": "' +
fileJSON[i].Punti + '"},\n';
  }
  file += '{ "Nome": "' + req.body.Nome
    + '", "Nickname": "' + req.body.Nickname
    + '", "Età": "' + req.body.Punti + '"}\n';
  file += "]";
  scriviFile(percorsoFile, file);
}

module.exports = { scriviFile, leggiFile, appendArrayJSON }
```

Questo codice definisce tre funzioni che utilizzano il modulo "fs" di Node.js per leggere e scrivere dati su file.
La funzione "scriviFile" accetta due argomenti: il percorso del file da scrivere e il contenuto da scrivere. Viene utilizzata la funzione "fs.writeFile" per scrivere il contenuto del file e viene restituito un messaggio di conferma se l'operazione è riuscita.
La funzione "leggiFile" accetta il percorso del file da leggere e restituisce i dati del file come oggetto JSON. Viene utilizzata la funzione "fs.readFileSync" per leggere il contenuto del file e viene restituito un oggetto JSON.
La funzione "appendArrayJSON" accetta tre argomenti: il percorso del file JSON da modificare, l'oggetto JSON che rappresenta l'array da aggiornare e i dati della richiesta HTTP POST da aggiungere all'array. Viene utilizzato un ciclo for per generare una stringa JSON rappresentante l'array completo e viene aggiunta una nuova stringa JSON per rappresentare i nuovi dati. La funzione "scriviFile" viene chiamata per sovrascrivere il file originale con il nuovo contenuto.
Infine, le tre funzioni vengono esportate come oggetto per poter essere utilizzate in altri moduli.

si prosegue con le videate del programma:

base.ejs

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

    <!--  Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous"></script>

    <!-- Popper and Bootstrap JS -->
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js" integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js" integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF" crossorigin="anonymous"></script>

  </body>
</html>
```

Il codice rappresenta una pagina HTML 5 valida, che utilizza la libreria Bootstrap per aggiungere stili e funzionalità alla pagina. Il documento inizia con il tag <!doctype html>, che indica al browser che il documento è scritto in HTML5.

All'interno del tag <html> viene specificato l'attributo lang con il valore "en", che indica che la lingua utilizzata nel documento è l'inglese.

Il contenuto della pagina viene poi definito nel tag <head>, dove sono specificati i metadati necessari alla visualizzazione e all'interpretazione della pagina. In particolare, viene specificato l'attributo charset con il valore "utf-8", che indica che la codifica dei caratteri utilizzata è l'UTF-8.

Viene poi specificato il tag <meta> con l'attributo name impostato a "viewport" e l'attributo content impostato a "width=device-width, initial-scale=1", che definisce il viewport della pagina e ne permette una corretta visualizzazione su dispositivi mobili.

Segue poi la sezione in cui viene inclusa la libreria Bootstrap, tramite l'attributo href del tag <link>. L'attributo rel è impostato a "stylesheet", che indica che si sta includendo un file CSS. Successivamente, vengono inclusi i file JavaScript necessari alla funzionalità di Bootstrap. Il primo script, bootstrap.bundle.min.js, è il bundle completo di Bootstrap, che comprende la libreria Popper.js. Il secondo script, bootstrap.min.js, contiene solo il codice JavaScript di Bootstrap. Entrambi gli script vengono inclusi tramite l'attributo src del tag <script>.

Infine, il documento viene chiuso con il tag </body> e </html>.

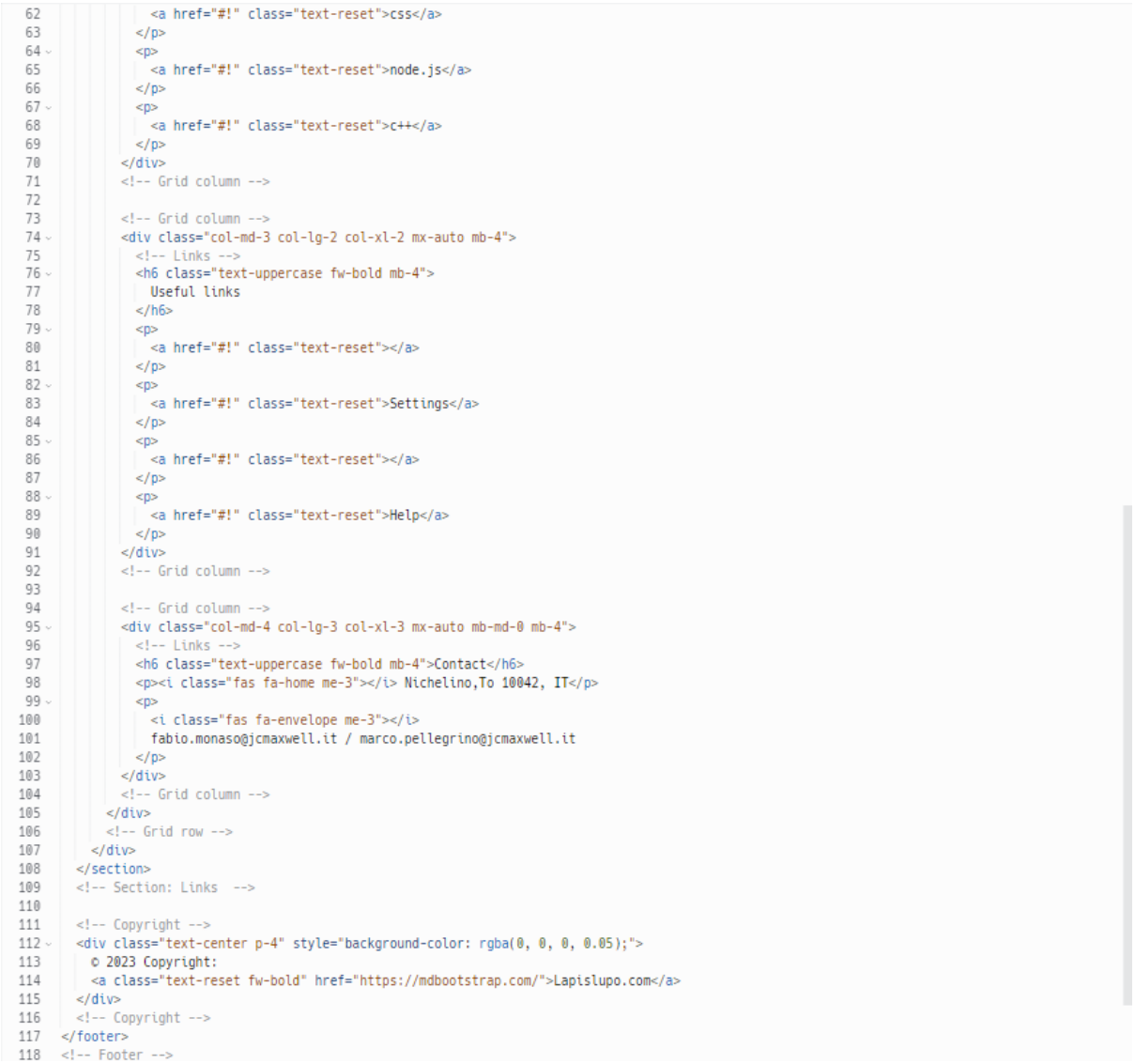
In conclusione, il codice fornito rappresenta una pagina HTML 5 valida, che utilizza la libreria Bootstrap per aggiungere stili e funzionalità alla pagina.

footer.ejs

```ejs
  <%- include ("base.ejs")%>
<!-- Footer -->
<footer class="text-center text-lg-start bg-light text-muted">
  <!-- Section: Social media -->
  <section class="d-flex justify-content-center justify-content-lg-between p-4 border-bottom">
    <!-- Left -->
    <div class="me-5 d-none d-lg-block">
      <span>Get connected with us on social networks:</span>
    </div>
    <!-- Left -->
    <!-- Right -->
    <div>
      <a href="" class="me-4 text-reset">
        <i class="fab fa-facebook-f"></i>
      </a>
      <a href="" class="me-4 text-reset">
        <i class="fab fa-twitter"></i>
      </a>
      <a href="" class="me-4 text-reset">
        <i class="fab fa-google"></i>
      </a>
      <a href="" class="me-4 text-reset">
        <i class="fab fa-instagram"></i>
      </a>
      <a href="" class="me-4 text-reset">
        <i class="fab fa-linkedin"></i>
      </a>
      <a href="" class="me-4 text-reset">
        <i class="fab fa-github"></i>
      </a>
    </div>
    <!-- Right -->
  </section>
  <!-- Section: Social media -->

  <!-- Section: Links  -->
  <section class="">
    <div class="container text-center text-md-start mt-5">
      <!-- Grid row -->
      <div class="row mt-3">
        <!-- Grid column -->
        <div class="col-md-3 col-lg-4 col-xl-3 mx-auto mb-4">
          <!-- Content -->
          <h6 class="text-uppercase fw-bold mb-4">
            <i class="fas fa-gem me-3"></i>
          </h6>
          <p>
          </p>
        </div>
        <!-- Grid column -->

        <!-- Grid column -->
        <div class="col-md-2 col-lg-2 col-xl-2 mx-auto mb-4">
          <!-- Links -->
          <h6 class="text-uppercase fw-bold mb-4">
            Products
          </h6>
          <p>
            <a href="#!" class="text-reset">javascript</a>
          </p>
          <p>
```



```
          <a href="#!" class="text-reset">css</a>
        </p>
        <p>
          <a href="#!" class="text-reset">node.js</a>
        </p>
        <p>
          <a href="#!" class="text-reset">c++</a>
        </p>
      </div>
      <!-- Grid column -->

      <!-- Grid column -->
      <div class="col-md-3 col-lg-2 col-xl-2 mx-auto mb-4">
        <!-- Links -->
        <h6 class="text-uppercase fw-bold mb-4">
          Useful links
        </h6>
        <p>
          <a href="#!" class="text-reset"></a>
        </p>
        <p>
          <a href="#!" class="text-reset">Settings</a>
        </p>
        <p>
          <a href="#!" class="text-reset"></a>
        </p>
        <p>
          <a href="#!" class="text-reset">Help</a>
        </p>
      </div>
      <!-- Grid column -->

      <!-- Grid column -->
      <div class="col-md-4 col-lg-3 col-xl-3 mx-auto mb-md-0 mb-4">
        <!-- Links -->
        <h6 class="text-uppercase fw-bold mb-4">Contact</h6>
        <p><i class="fas fa-home me-3"></i> Nichelino,To 10042, IT</p>
        <p>
          <i class="fas fa-envelope me-3"></i>
          fabio.monaso@jcmaxwell.it / marco.pellegrino@jcmaxwell.it
        </p>
      </div>
      <!-- Grid column -->
    </div>
    <!-- Grid row -->
  </div>
</section>
<!-- Section: Links  -->

<!-- Copyright -->
<div class="text-center p-4" style="background-color: rgba(0, 0, 0, 0.05);">
  © 2023 Copyright:
  <a class="text-reset fw-bold" href="https://mdbootstrap.com/">Lapislupo.com</a>
</div>
<!-- Copyright -->
</footer>
<!-- Footer -->
```

Il codice definisce la struttura del footer di una pagina web.

La sezione "Social media" contiene un div con una span che mostra un messaggio ai visitatori del sito web e un altro div con icone che rappresentano i link ai social media del sito web.

La sezione "Links" contiene quattro div, ognuno dei quali rappresenta una colonna di link utili per gli utenti. In particolare, il primo div contiene solo un'icona, mentre gli altri tre contengono titoli e link a diverse pagine web.

La sezione "Copyright" contiene un div con il testo del copyright e un link al sito web proprietario.

Inoltre, il codice utilizza classi Bootstrap per la formattazione e la disposizione degli elementi all'interno del footer.

header.ejs

```
<%- include ("base.ejs")%>
<!--test -->
<!doctype html>
<html lang="en">



<!-- navbar -->
<ul class="nav nav-pills">
 <li class="nav-item">
   <a class="nav-link active"  href="/">HOME</a>
  </li>
  <li class="nav-item">
   <a class="nav-link" href="insert">INSERISCI</a>
  </li>
   <li class="nav-item">
   <a class="nav-link" href="classifica">MURO</a>
  </li>

</ul>
```

Questo codice è una sezione di codice HTML che include un file "base.ejs" e definisce una struttura di navigazione per un sito web.
La navigazione consiste in una lista di collegamenti a tre pagine web: "HOME", "INSERISCI" e "MURO".
Il collegamento "HOME" è evidenziato come attivo attraverso la classe "active" della sua voce di navigazione.
La struttura di navigazione è implementata come una lista non ordinata (<ul>) e ogni voce di navigazione è un elemento di lista (<li>) con un collegamento (<a>) al rispettivo indirizzo della pagina web.

index.ejs

```
<%- include ("partiejs/header.ejs")%>
<html>
  <br>
  <center><strong> Benvenuto nella home clicca su Inserisci per aggiungere un post-it </strong>
</center>

 </html>

 <%- include ("partiejs/footer")%>
```

Il codice fornito è una pagina HTML che utilizza il motore di template EJS per includere un header e un footer in parti separate del documento.

In particolare, la linea <%- include ("partiejs/header.ejs")%> utilizza la sintassi di EJS per includere il file "header.ejs" nella pagina HTML. Questo suggerisce che il file "header.ejs" contenga codice che verrà inserito nella parte superiore della pagina.

In modo simile, la linea <%- include ("partiejs/footer")%> utilizza la sintassi di EJS per includere il file "footer.ejs" nella pagina HTML. Questo suggerisce che il file "footer.ejs" contenga codice che verrà inserito nella parte inferiore della pagina.

Il resto del codice HTML include un tag
 per inserire una riga vuota, seguito da un tag per centrare il testo che contiene un messaggio di benvenuto per gli utenti. Infine, il tag </html> chiude il documento HTML.

```
<%- include ("partiejs/header.ejs")%>
 <form action="/inserisci" method="post">
 <label for="Nome" class="form-label">Nome</label>
          <input type="text" class="form-control" id="Nome" aria-describedby="emailHelp"
name="Nome">
        </div>
        <div class="mb-3">
          <label for="Nickname" class="form-label">Nickname</label>
          <input type="text" class="form-control" id="Nickname" name="Nickname">
        </div>
  <div class="mb-3">
          <label for="Punti" class="form-label">Età</label>
          <input type="number" class="form-control" id="Punti" name="Punti">
        </div>
<button type="submit" class="btn btn-primary">Invia</button>
</form>

  <%- include ("partiejs/footer")%>
```

Questo codice rappresenta una pagina HTML con un form che consente agli utenti di inserire il proprio nome, nickname e età. Il form contiene tre campi di input, corrispondenti alle informazioni richieste: nome (input di tipo testo), nickname (input di tipo testo) e età (input di tipo numerico).Il form è stato configurato in modo che, quando l'utente preme il pulsante "Invia", i dati inseriti siano inviati al server tramite una richiesta HTTP POST, con destinazione "/inserisci".

Questa destinazione indica che, probabilmente, lato server esiste una rotta associata a "/inserisci" che si occuperà di ricevere e gestire i dati inviati dal form. Inoltre, la pagina HTML include due file EJS ("partiejs/header.ejs" e "partiejs/footer.ejs") che verranno inclusi nella pagina renderizzata dal server. Questo suggerisce che la pagina HTML sia stata creata con l'intenzione di essere utilizzata in un framework web che supporta EJS.

score.ejs (MURO):

```ejs
<%- include ("partiejs/header.ejs")%>

  <div class="container">
   <%data.forEach(function(dato) {%>
    <div class="container-inner">
   <div class="sticky-container">
     <div class="sticky-outer">
       <div class="sticky">
         <svg width="0" height="0">
           <defs>
             <clipPath id="stickyClip" clipPathUnits="objectBoundingBox">
               <path
                     d="M 0 0 Q 0 0.69, 0.03 0.96 0.03 0.96, 1 0.96 Q 0.96 0.69, 0.96 0 0.96 0, 0 0"
                     stroke-linejoin="round"
                     stroke-linecap="square"
                     />
             </clipPath>
           </defs>
         </svg>
         <div class="sticky-content">
             <thead>
    <tr>
      <th scope="col">Nome</th>
      <th scope="col">Nickname</th>
      <th scope="col">Età</th>
    </tr>
  </thead>
             <tbody>
           <tr>
             <th scope="row" ><%=dato.Nome%></th>
             <td scope="row"><%=dato.Nickname%></td>
             <td scope="row"><%=dato.Punti%></td><br>
           </tr>
               </div>
         </div>
       </div>
     </div>
   </div>

   </tbody>


</div>
    <%}%>

<style>
  @import url('https://fonts.googleapis.com/css2?family=Kalam&display=swap');

/* Some positioning and ratios */
.sticky-container {
  max-width: 270px;
  position: relative;
}

.sticky-outer {
  display: flex;
  padding-top: 92.5925926%;
  position: relative;

  width: 100%;
}
```

```ejs
.sticky {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
}

/* Shadow behind the sticky note */
.sticky:before {
  box-shadow: -2px 2px 15px 0 rgba(0, 0, 0, 0.5);
  background-color: rgba(0, 0, 0, 0.25);
  content: '';
  width: 90%;
  left: 5px;
  height: 75%;
  position: absolute;
  top: 30%;
}

/* The sticky note itself */
.sticky-content {
  background: linear-gradient(
    180deg,
    rgba(187, 235, 255, 1) 0%,
    rgba(187, 235, 255, 1) 12%,
    rgba(170, 220, 241, 1) 75%,
    rgba(195, 229, 244, 1) 100%
  );
  width: 100%;
  height: 100%;

  display: flex;

  font-family: 'Kalam', cursive;
  font-size: 1.25rem;

  clip-path: url(#stickyClip);
}


/* Position the sticky nicely, so it looks better */
html,
body {
  height: 100%;
  margin: 5px;
  font-size: 16px;
}

.container {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 100%;
  height: 100%;
}
.container-inner {
  width: 50%;
  margin: 25px;
}
```

```
}

/* Add responsiveness */
@media screen and (min-width: 640px) {
  .sticky-content {
    font-size: 1.5rem;
  }
  .container-inner {
    width: 50%;
  }
}

@media screen and (min-width: 768px) {
  .sticky-content {
    font-size: 1.5rem;
  }
  .container-inner {
    width: 50%;
  }
}

@media screen and (min-width: 1024px) {
  .sticky-content {
    font-size: 1.875rem;
  }
  .container-inner {
    width: 25%;
  }
}
</style>
 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous"></script>









<%- include ("partiejs/footer")%>
```

Il codice fornito è un template EJS per generare una lista di dati in formato tabellare, dove ogni riga corrisponde a un oggetto del dataset fornito tramite la variabile “data”.Per ogni oggetto in” data”, viene generato un div con la classe “container-inner” che contiene un elemento “sticky-container,” che a sua volta contiene un elemento” sticky-outer” e un elemento” sticky.”

L'elemento sticky rappresenta una nota adesiva con un background gradiente, e contiene un elemento “sticky-content” che mostra i dati dell'oggetto corrente, rappresentati da “ dato.Nome, dato.Nickname,” e “dato.Punti.”

Infine, il codice include stili CSS per il layout e la formattazione della nota adesiva e delle tabelle, e utilizza Bootstrap 5 per la gestione della responsività.

Criticità problematiche e possibili ottimizzazioni del programma:

Criticità:

La disposizione uniforme dei post-it può rivelarsi cruciale per garantire la chiarezza e l'accessibilità delle informazioni ivi contenute. Un'ubicazione casuale dei post-it può generare difficoltà nella lettura e nell'organizzazione dei dati, specialmente se si tratta di un gran numero di annotazioni. In aggiunta, una disposizione omogenea dei post-it può contribuire ad un aspetto accattivante dell'ambiente lavorativo, promuovendo una sensazione di ordine e pulizia. Per ottenere una disposizione uniforme dei post-it, è consigliabile utilizzare degli strumenti di misurazione per garantire la distanza adeguata tra le annotazioni, oppure creare un modello di posizionamento per avere un punto di riferimento. Inoltre, si possono usare colori differenti per distinguere le diverse categorie di note e facilitare la lettura e l'organizzazione. Una disposizione uniforme dei post-it può quindi migliorare la produttività, facilitare l'organizzazione e creare un ambiente di lavoro più piacevole e ordinato.

Ottimizzazione:

Una strategia di ottimizzazione che può avere un impatto positivo sulla qualità e l'usabilità dell'applicazione è l'implementazione di un'interfaccia utente intuitiva e facile da utilizzare. Una buona interfaccia utente può migliorare significativamente l'esperienza dell'utente, rendendo l'applicazione più piacevole e semplice da utilizzare. Inoltre, una buona progettazione dell'interfaccia utente può ridurre gli errori degli utenti e migliorare l'efficienza dell'uso dell'applicazione. Per realizzare un'interfaccia utente di alta qualità, è necessario considerare aspetti come la disposizione degli elementi dell'interfaccia, la scelta dei colori, la dimensione dei caratteri e la navigazione. Inoltre, è importante testare l'interfaccia utente con utenti reali per valutare la sua usabilità e apportare eventuali miglioramenti. Implementare un'interfaccia utente di qualità superiore può quindi aumentare la soddisfazione degli utenti, migliorare la reputazione dell'applicazione e migliorare l'efficienza complessiva del sistema.

Riferimenti teorici:

1. Il progetto Social Post-it 1.0 può essere analizzato da molteplici prospettive teoriche.
2. La teoria dell'interazione sociale si concentra sulla condivisione di informazioni personali attraverso l'utilizzo di post-it. In questo modo, gli utenti hanno l'opportunità di interagire tra di loro, commentare e condividere informazioni.
3. La teoria della comunicazione analizza invece come gli utenti codificano e decodificano le informazioni sui post-it e come questo influenza la comprensione e la condivisione di tali informazioni.
4. La teoria dei social network si concentra sulla creazione di una rete sociale informale tra gli utenti che condividono informazioni sui post-it. In questo contesto, la struttura della rete sociale può influenzare la diffusione delle informazioni e la formazione di gruppi di interesse.
5. https:// La teoria dell'usabilità si concentra sull'interazione degli utenti con i post-it e su come questo influisce sulla facilità d'uso del sistema. Infine, la teoria del design dell'interazione analizza il processo di design dell'interfaccia utente e come questo influisce sull'esperienza utente complessiva.

RIferimenti del Progetto:

Link della Presentazione:
Presentazione

Link del GitHub del progetto:
Progetto

Link dell'articolo del progetto:
Articolo