Documentazione tecnica “Social post-it”.

1. Introduzione

1.1 Scopo del Software

Il software "Social Postit" è un'applicazione web che consente agli utenti di pubblicare post con una durata temporale limitata o illimitata. L'obiettivo principale è creare un ambiente interattivo in cui gli utenti possano condividere messaggi che scadono automaticamente dopo un certo periodo di tempo.

1.2 Descrizione Generale del Software

Il software è sviluppato utilizzando Node.js con il framework Express e utilizza EJS per il rendering delle pagine. I dati degli utenti e dei post sono memorizzati in file JSON. Il sistema permette la registrazione e il login degli utenti, oltre alla pubblicazione e rimozione automatica dei post scaduti.

1.3 Obiettivi della Documentazione

Questa documentazione descrive l'architettura, il design e il funzionamento del software, fornendo dettagli tecnici e spiegazioni del codice.

1.4 Panoramica del Documento

Il documento copre i seguenti aspetti:

- Architettura del software
- Analisi delle proprietà (robustezza, usabilità, portabilità)
- Design del software
- Documentazione del codice
- Testing
- Deployment e manutenzione

---

2. Architettura del Software

2.1 Descrizione Generale dell'Architettura

L'architettura del software segue un modello MVC semplificato:

- **Model:** I dati vengono gestiti tramite file JSON.
- **View:** Le interfacce utente sono realizzate con EJS e Bootstrap.
- **Controller:** Le route Express gestiscono le richieste HTTP.

2.2 Componenti Principali

- **Express.js** per la gestione delle richieste HTTP.
- **EJS** per la generazione dinamica delle pagine HTML.
- **File JSON** per la memorizzazione di utenti e post.
- **bcryptjs** per la crittografia delle password.

2.3 Flusso di Dati

1. L'utente si registra o effettua il login.
2. Dopo il login, può pubblicare un post con una durata definita.
3. I post scaduti vengono rimossi automaticamente ogni 5 secondi.
4. Gli utenti possono visualizzare gli ultimi post pubblicati.

---

3. Analisi delle Proprietà

3.1 Robustezza

- **Gestione degli errori:** Il codice verifica l'esistenza di utenti duplicati in fase di registrazione.

- **Resilienza:** I post vengono caricati da file JSON e gestiti in memoria per efficienza.
- **Codice documentato per la gestione degli errori:** La verifica della password utilizza `bcrypt.compareSync()`.

3.2 Usabilità

- **Interfaccia utente chiara:** Utilizzo di Bootstrap per un design responsive.
- **Accessibilità:** Supporto al tema scuro/chiaro con localStorage.
- **Feedback degli utenti:** Messaggi di errore in caso di credenziali errate.

3.3 Portabilità

- **Compatibilità:** Funziona su qualsiasi server Node.js.
- **Gestione della configurazione:** Il codice può essere facilmente modificato per supportare database.

---

4. Design del Software

4.1 Principi di Design Adottati

- Separazione tra logica di business e presentazione.
- Uso di funzioni modulari per caricamento/salvataggio dati.

4.3 Scelte Architetturali e Motivi

- Scelta di JSON per la memorizzazione per semplicità e rapidità di sviluppo.

---

5. Documentazione del Codice

5.1 Struttura del Codice Sorgente

- `app.js` - Entry point del server.
- `views/` - Contiene i file EJS.
- `public/` - Contiene CSS, immagini e le due librerie di bootstrap.
- `users.json` - Archivio utenti.
- `posts.json` - Archivio post.

5.2 Classi e Metodi Principali

- `loadUsers()` - Carica gli utenti da JSON.
- `loadPosts()` - Carica i post da JSON.
- `cleanExpiredPosts()` - Rimuove i post scaduti.

5.3 Struttura di ogni codice e spiegazione riga per riga

5.3.1 app.js:

```
const express = require('express');
const fs = require('fs');
const path = require('path');
const bcrypt = require('bcryptjs');
const bodyParser = require('body-parser');
```

express: framework per la creazione di applicazioni web in Node.js.
fs: modulo per la gestione dei file sul filesystem.
path: modulo per la gestione dei percorsi dei file.
bcryptjs: libreria per la crittografia delle password.
bodyParser: middleware per elaborare i dati delle richieste HTTP POST.

```
const app = express();
const PORT = 3000;
```

app: oggetto principale che rappresenta l'applicazione Express.
PORT: numero della porta su cui il server ascolterà.

```
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));
app.set('view engine', 'ejs');
```

bodyParser.urlencoded({ extended: false }): consente di gestire i dati inviati tramite form.
express.static(path.join(__dirname, 'public')): serve i file statici dalla cartella public.
app.set('view engine', 'ejs'): imposta **EJS** come motore di template.

```
function loadUsers() {
    return JSON.parse(fs.readFileSync(path.join(__dirname, 'users.json')));
}

function loadPosts() {
    return JSON.parse(fs.readFileSync(path.join(__dirname, 'posts.json')));
}

function savePosts(posts) {
    fs.writeFileSync(path.join(__dirname, 'posts.json'), JSON.stringify(posts, null, 2));
}
```

loadUsers(): carica gli utenti da users.json.
loadPosts(): carica i post da posts.json.
savePosts(posts): salva i post nel file posts.json.

```
// Rimuovi post scaduti
function cleanExpiredPosts() {
    let posts = loadPosts();
    const now = Date.now();
    posts = posts.filter(post => post.expiryTime === null || post.expiryTime > now);
    savePosts(posts);
}

setInterval(cleanExpiredPosts, 5000); // Pulisce i post scaduti ogni 5 secondi
```

cleanExpiredPosts(): rimuove i post con **expiryTime** passato.

setInterval(cleanExpiredPosts, 5000): chiama questa funzione ogni **5 secondi**.

```
app.get('/', (req, res) => {
    cleanExpiredPosts();
    const posts = loadPosts();
    res.render('index', { posts });
});
```

/: carica i post (dopo aver rimosso quelli scaduti) e rende la vista index.ejs.

```
app.get('/login', (req, res) => {
    res.render('login');
});

app.get('/register', (req, res) => {
    res.render('register');
});
```

/login: rende la vista login.ejs.

/register: rende la vista register.ejs.

```
app.post('/register', (req, res) => {
    const { username, password } = req.body;
    const users = loadUsers();

    if (users.find(user => user.username === username)) {
        return res.redirect('/register'); // Username già esistente
    }

    const hashedPassword = bcrypt.hashSync(password, 8);
    users.push({ username, password: hashedPassword });
    fs.writeFileSync(path.join(__dirname, 'users.json'), JSON.stringify(users, null, 2));

    res.redirect('/login');
});
```

Controlla se l'username è già esistente.

Cripta la password con bcrypt.hashSync(password, 8).

Salva il nuovo utente in users.json.

Reindirizza l'utente alla pagina di login.

```
app.post('/login', (req, res) => {
    const { username, password } = req.body;
    const users = loadUsers();
    const user = users.find(user => user.username === username);

    if (user && bcrypt.compareSync(password, user.password)) {
        res.redirect('/');
    } else {
        res.redirect('/login'); // Credenziali errate
    }
});
```

Verifica se l'utente esiste.

Confronta la password inserita con quella salvata (cripta).

Se corretta, reindirizza alla home, altrimenti rimanda al login.

```
app.post('/post', (req, res) => {
    const { username, content, timer } = req.body;
    const expiryTime = timer === "unlimited" ? null : Date.now() + parseInt(timer);
    const posts = loadPosts();
    posts.push({ username, content, date: new Date().toISOString(), expiryTime });
    savePosts(posts);
    res.redirect('/');
});
```

Ottiene i dati del post dal form.

Calcola l'expiryTime in millisecondi (o null se illimitato).

Salva il post in posts.json.

Reindirizza alla home.

```
app.listen(PORT, () => {
    console.log(`Server in esecuzione su http://localhost:${PORT}`);
});
```

Avvia il server sulla **porta 3000**.

5.3.2 index.ejs:

codice html:

```html
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Social Postit</title>
    <link href="/bootstrap/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="/darkmode.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light" id="navbar">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">Postit</a>
            <ul class="navbar-nav ms-auto">
                <li class="nav-item"><a class="nav-link" href="/login">Login</a></li>
                <li class="nav-item"><a class="nav-link" href="/register">Registrati</a></li>
                <li class="nav-item">
                    <button class="btn btn-outline-secondary" id="theme-toggle">🌙 / 🌞</button>
                </li>
            </ul>
        </div>
    </nav>

    <div class="container">
        <div class="text-center"><img src="/img/postit2.png" class="img-fluid"></div>
        <form action="/post" method="POST" class="">
            <textarea class="form-control" name="content" rows="3" placeholder="Scrivi un post..." required></textarea>
            <label for="timer">Seleziona durata:</label>
            <select class="form-control" name="timer" required>
                <option value="unlimited">Illimitato</option>
                <option value="300000">5 minuti</option>
                <option value="1800000">30 minuti</option>
                <option value="3600000">1 ora</option>
            </select>
            <button type="submit" class="btn btn-primary mt-3">Pubblica</button>
        </form>

        <div class="mt-5">
            <h3>Ultimi Post</h3>
            <ul class="list-group">
                <% posts.forEach(post => { %>
                    <li class="list-group-item">
                        <strong><%= post.username %></strong> <small><%= new Date(post.date).toLocaleString() %></small>
                        <p><%= post.content %></p>
                    </li>
                <% }) %>
            </ul>
        </div>
    </div>

    <footer class="text-center py-3">
        <p>&copy; 2025 Social Postit. All rights reserved.</p>
    </footer>
```

Struttura Generale

Il documento HTML definisce la pagina principale di *Social Postit*. Include Bootstrap per la grafica e un file CSS personalizzato per la gestione del tema scuro/chiaro.

Navbar

Contiene il logo (che rimanda alla home), due link per login e registrazione e un pulsante per cambiare il tema tra chiaro e scuro.

Contenitore Principale

- Un logo al centro.
- Un form per scrivere un post, con una *textarea* e un menu a tendina per selezionare la durata del post (illimitato, 5 min, 30 min, 1 ora).
- Un pulsante per pubblicare il post.

Lista degli ultimi post

Questa è la parte dinamica gestita da **EJS**:

```
<% posts.forEach(post => { %>
  <li class="list-group-item">
    <strong><%= post.username %></strong>
    <small><%= new Date(post.date).toLocaleString() %></small>
    <p><%= post.content %></p>
  </li>
<% }) %>
```

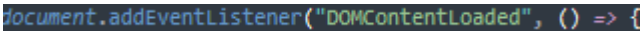
- *posts* è un array che contiene i post.
- Il codice **forEach** scorre ogni post e lo inserisce nella lista.
- **<%= %>** stampa i dati (username, data e contenuto).

Footer

Contiene un semplice copyright con l'anno 2025.

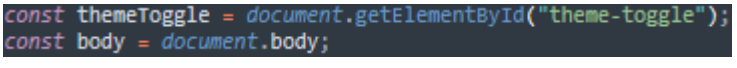
```
<script>
    document.addEventListener("DOMContentLoaded", () => {
        const themeToggle = document.getElementById("theme-toggle");
        const body = document.body;

        if (localStorage.getItem("theme") === "dark") {
            body.classList.add("dark-mode");
        }

        themeToggle.addEventListener("click", () => {
            body.classList.toggle("dark-mode");
            localStorage.setItem("theme", body.classList.contains("dark-mode") ? "dark" : "light");
        });
    });
</script>
```

NB QUESTA PARTE DI SCRIPT E' UGUALE ANCHE NEL FILE LOGIN.EJS E REGISTER.EJS QUINDI EVITERO' DI RISPIEGARLO.



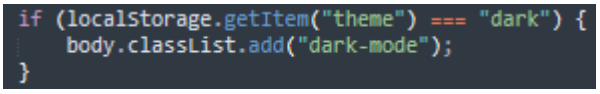
```
document.addEventListener("DOMContentLoaded", () => {
```

Aspetta che l'intero documento HTML sia caricato prima di eseguire il codice JavaScript. Evita problemi in cui lo script viene eseguito prima che gli elementi HTML siano disponibili.



```
const themeToggle = document.getElementById("theme-toggle");
const body = document.body;
```

themeToggle: Rappresenta il **bottone** che cambia il tema.
body: Seleziona l'intero **body** della pagina, perché qui vogliamo applicare o rimuovere la classe "dark-mode".



```
if (localStorage.getItem("theme") === "dark") {
    body.classList.add("dark-mode");
}
```

localStorage.getItem("theme") recupera il valore memorizzato nel browser.

Se il valore è "dark", allora viene **aggiunta la classe "dark-mode"**, attivando subito il tema scuro.

```
    themeToggle.addEventListener("click", () => {
        body.classList.toggle("dark-mode");
        localStorage.setItem("theme", body.classList.contains("dark-mode") ? "dark" : "light");
    });
});
```

- **Quando l'utente clicca sul bottone (themeToggle)**:
    1. body.classList.toggle("dark-mode") → Attiva o disattiva il tema scuro.
    2. localStorage.setItem("theme", body.classList.contains("dark-mode") ? "dark" : "light");
        - Salva la preferenza dell'utente nel localStorage, quindi il tema scelto rimane anche dopo un refresh della pagina.

5.3.3 login.ejs:

```
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Social Postit</title>
    <link href="/bootstrap/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="/darkmode.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light" id="navbar">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">Torna alla home</a>
            <ul class="navbar-nav ms-auto">
                <li class="nav-item">
                    <button class="btn btn-outline-secondary" id="theme-toggle">🌙 / 🌞</button>
                </li>
            </ul>
        </div>
    </nav>

    <div class="container">
        <img src="/img/Login.png" class="img-fluid">
        <form action="/login" method="POST">
            <div class="mb-3">
                <label for="username" class="form-label">Nome Utente</label>
                <input type="text" class="form-control" id="username" name="username" required>
            </div>
            <div class="mb-3">
                <label for="password" class="form-label">Password</label>
                <input type="password" class="form-control" id="password" name="password" required>
            </div>
            <button type="submit" class="btn btn-primary">Accedi</button>
        </form>
        <p class="mt-3">Non hai un account? <a href="/register">Registrati</a></p>
    </div>
```

(lo script è uguale al file index.ejs)

Struttura Generale

Il file **login.ejs** è una pagina HTML che permette agli utenti di accedere al sito *Social Postit*. Come nel precedente file, è presente l'integrazione con Bootstrap per lo stile e un file CSS personalizzato per il tema scuro/chiaro.

Navbar

La navbar ha:

- Un link che riporta alla home, intitolato *"Torna alla home"*.
- Un pulsante per il cambio del tema scuro/chiaro.

Contenitore Principale

- Un'immagine centrale che rappresenta la pagina di login.
- Un form con due campi:
    - **Nome utente**: Campo di testo per inserire il nome utente.
    - **Password**: Campo per inserire la password, con il tipo "password" per mascherare i caratteri.
- Un pulsante per inviare i dati e accedere.
- Un link per chi non ha un account, che rimanda alla pagina di registrazione.

CSS e Bootstrap

- Il layout è stilizzato utilizzando il framework Bootstrap.
- La struttura di base per i campi del form e il pulsante di invio segue lo stile fornito da Bootstrap.

Interazione con il Backend (EJS)

Non ci sono parti dinamiche con **EJS** in questa pagina specifica, poiché si tratta di una semplice interfaccia di login.

5.3.4 register.ejs:

```html
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registrazione - Social Postit</title>
    <link href="/bootstrap/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="/darkmode.css">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light" id="navbar">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">Torna alla home</a>
            <ul class="navbar-nav ms-auto">
                <li class="nav-item">
                    <button class="btn btn-outline-secondary" id="theme-toggle">🌙 / 🟠</button>
                </li>
            </ul>
        </div>
    </nav>
    <div class="container">
        <img src="/img/Registrati.png" class="img-fluid">
        <form action="/register" method="POST">
            <div class="mb-3">
                <label for="username" class="form-label">Nome Utente</label>
                <input type="text" class="form-control" id="username" name="username" required>
            </div>
            <div class="mb-3">
                <label for="password" class="form-label">Password</label>
                <input type="password" class="form-control" id="password" name="password" required>
            </div>
            <button type="submit" class="btn btn-primary">Registrati</button>
        </form>
        <p class="mt-3">Hai già un account? <a href="/login">Accedi</a></p>
    </div>
```

Struttura Generale

Il file **register.ejs** è una pagina HTML che permette agli utenti di registrarsi su *Social Postit*. Come per le altre pagine, viene utilizzato Bootstrap per lo stile e un file CSS personalizzato per il tema scuro/chiaro.

Navbar

La navbar contiene:

- Un link che riporta alla home, con la dicitura *"Torna alla home"*.
- Un pulsante che consente agli utenti di alternare tra il tema scuro e quello chiaro.

Contenitore Principale

- Un'immagine centrale che rappresenta la pagina di registrazione.
- Un form con due campi:
    - **Nome utente**: Un campo di testo per inserire il nome utente desiderato.
    - **Password**: Un campo di tipo "password" per inserire la password in modo sicuro (i caratteri saranno nascosti).
- Un pulsante per inviare i dati e completare la registrazione.
- Un link che reindirizza gli utenti alla pagina di login nel caso in cui abbiano già un account.

CSS e Bootstrap

- Il layout è stilizzato grazie a **Bootstrap**, che fornisce una struttura responsive e ben organizzata per i campi del form e i pulsanti.

Interazione con il Backend (EJS)

Anche in questa pagina non ci sono elementi dinamici specifici gestiti con **EJS**. La pagina si limita a ricevere i dati dal form di registrazione e inviarli al server per la creazione di un nuovo account.

5.3.5 darkmode.css:

```css
/* darkmode.css */
.dark-mode {
    background-color: #121212;
    color: #ffffff;
}

.dark-mode .navbar {
    background-color: #1f1f1f !important;
}

.dark-mode .list-group-item {
    background-color: #222;
    border-color: #444;
    color: #ddd;
}

.dark-mode a {
    color: #bb86fc;
}

.dark-mode .btn-primary {
    background-color: #6200ea;
    border-color: #3700b3;
}

.dark-mode .btn-primary:hover {
    background-color: #3700b3;
}
```

Il file **darkmode.css** serve a stilizzare l'aspetto della pagina web quando il tema scuro è attivo. Ecco cosa fa ogni parte del codice:

Classe .dark-mode

Questa classe applica uno stile di colore scuro al sito:

- **Sfondo scuro** (background-color: #121212) e **testo chiaro** (color: #ffffff), che rendono il sito facilmente leggibile in ambienti con poca luce.

Navbar in Dark Mode

La navbar, quando il tema scuro è attivo, cambia colore:

- Lo sfondo diventa un grigio scuro (background-color: #1f1f1f) per adattarsi al tema scuro.

Elementi della lista

Gli elementi della lista (usati per visualizzare i post) vengono stilizzati in modo da integrarsi con il tema scuro:

- Lo sfondo dei singoli item diventa ancora più scuro (background-color: #222) e il colore del testo si fa più chiaro (color: #ddd) per migliorare la leggibilità.

Link in Dark Mode

I link diventano di un colore viola chiaro (color: #bb86fc) che spicca su uno sfondo scuro, rendendoli facili da individuare.

Pulsanti (btn-primary)

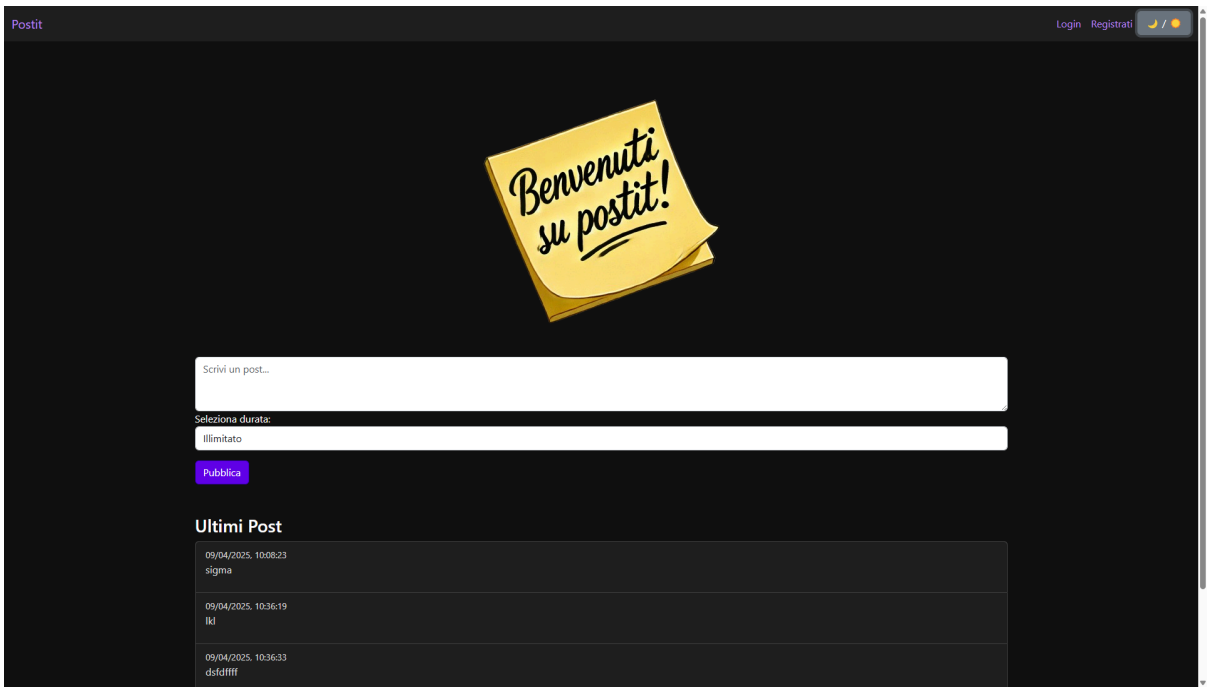
I pulsanti principali cambiano aspetto:

- Il **colore di sfondo** e il **bordo** diventano viola scuro (background-color: #6200ea e border-color: #3700b3).

- Quando si passa sopra un pulsante (hover), il colore di sfondo diventa ancora più scuro (background-color: #3700b3), creando un effetto visivo di interazione.

---

6. Videate



---

7. Testing

7.1 Strategia di Testing

Sono stati effettuati test manuali sulle principali funzionalità.

7.3 Test di Integrazione

Test manuali effettuati su diverse configurazioni di browser e dispositivi.

7.4 Test di Performance e Stress

Verificato il corretto funzionamento con centinaia di post caricati.

---

8. Deployment e Portabilità

8.1 Guida al Deployment

1. Installare Node.js
2. Clonare il repository
3. Eseguire `npm install`
4. Avviare il server con `node app.js`

---

9. Manutenzione del Software

9.1 Strategie di Manutenzione

- Controllo periodico della sicurezza degli utenti.
- Ottimizzazione delle prestazioni per grandi volumi di dati.

9.2 Aggiornamenti del Software

Aggiornamenti previsti per supportare database relazionali.

9.3 Bug Fixing e Refactoring

Monitoraggio dei log per identificare problemi di scalabilità.

---

10. Conclusioni

10.1 Sintesi delle Proprietà Analizzate

Il software è robusto, usabile e portabile, con un'architettura semplice ma efficace.

10.2 Suggerimenti Futuri per il Miglioramento

- Implementare autenticazione con sessioni.
- Integrare un database SQL o NoSQL.

10.3 Lezioni Apprese durante lo Sviluppo

L'uso di JSON semplifica la gestione dei dati, ma potrebbe limitare la scalabilità a lungo termine.

link al progetto: clicca qui