Documentazione tecnica “Societas”

Diego Tempesta, Rebecca Romilda Truden, Erika Saddi

Indice della Documentazione del Software

1. Introduzione

1.1 **Scopo del Software**
1.2 **Descrizione Generale del Software**
1.3 **Obiettivi della Documentazione**
1.4 **Panoramica del Documento**

2. Architettura del Software

2.1 **Descrizione Generale dell'Architettura**
2.2 **Componenti Principali**
2.3 **Flusso di Dati**
2.4 **Diagrammi UML** (Classi, Sequenze, Stato)

3. Analisi delle Proprietà

3.1 Robustezza
3.1.1 **Definizione di Robustezza nel Contesto del Software**
3.1.2 **Gestione degli Errori**
3.1.3 **Test di Robustezza**
3.1.4 **Resilienza del Software**
3.1.5 **Codice Documentato per la Gestione degli Errori**
3.2 Usabilità
3.2.1 **Definizione di Usabilità nel Contesto del Software**
3.2.2 **Interfaccia Utente (UI) e User Experience (UX)**
3.2.3 **Accessibilità**
3.2.4 **Feedback degli Utenti**
3.2.5 **Codice Documentato per la UI/UX**
3.3 Portabilità
3.3.1 **Definizione di Portabilità nel Contesto del Software**
3.3.2 **Compatibilità tra Piattaforme**
3.3.3 **Dipendenze Esterne e Gestione della Configurazione**
3.3.4 **Strategie di Testing Cross-Platform**
3.3.5 **Codice Documentato per la Portabilità**

4. Design del Software

4.1 **Principi di Design Adottati**
4.2 **Pattern di Design Utilizzati**
4.3 **Scelte Architetturali e Motivi**

5. Documentazione del Codice

5.1 **Struttura del Codice Sorgente**
5.2 **Classi e Metodi Principali**

5.3 **Commenti e Note Importanti nel Codice**
5.4 **Esempi di Utilizzo del Codice**
5.5 **Snippet di Codice per le Proprietà Analizzate** - **Codice per la Robustezza** - **Codice per l'Usabilità** - **Codice per la Portabilità**

6. Testing

6.1 **Strategia di Testing**
6.2 **Test Unitari**
6.3 **Test di Integrazione**
6.4 **Test di Performance e Stress**
6.5 **Test Cross-Platform**
6.6 **Risultati dei Test e Report**

7. Deployment e Portabilità

7.1 **Guida al Deployment**
7.2 **Ambienti di Produzione e Sviluppo**
7.3 **Esempi di Configurazione**
7.4 **Considerazioni sulla Portabilità durante il Deployment**

8. Manutenzione del Software

8.1 **Strategie di Manutenzione**
8.2 **Aggiornamenti del Software**
8.3 **Bug Fixing e Refactoring**
8.4 **Backup e Recupero dei Dati**

9. Conclusioni

9.1 **Sintesi delle Proprietà Analizzate**
9.2 **Suggerimenti Futuri per il Miglioramento**
9.3 **Lezioni Apprese durante lo Sviluppo**

10. Appendici

10.1 **Glossario**
10.2 **Riferimenti e Bibliografia**
10.3 **Link Utili e Risorse Esterne**

1. Introduzione

1.1 Scopo del Software

L'applicazione **Societas** è un social network minimalista che consente agli utenti di registrarsi, effettuare il login e condividere dei "post-it" di testo e immagini. Gli utenti possono interagire con i post mediante like, dislike, recensioni (voti a stelle) e commenti. Inoltre, è possibile caricare una foto profilo, che verrà visualizzata accanto ai post.

1.2 Descrizione Generale del Software

Societas è un'applicazione web realizzata con Node.js e il framework Express.
Le view sono renderizzate con il motore di template EJS e lo stile è gestito tramite Bootstrap.
I dati (utenti e post) sono persistiti in file JSON (users.json e posts.json).
L'applicazione supporta il caricamento di file (immagini per i post e foto profilo) tramite il middleware multer.
Le funzionalità principali includono:

- Registrazione/login degli utenti
- Caricamento e aggiornamento della foto profilo
- Creazione di post (testo e/o immagini)
- Interazioni sui post: like, dislike, recensioni e commenti

1.3 Obiettivi della Documentazione

Questa documentazione ha lo scopo di:

- Fornire una panoramica completa dell'architettura e delle funzionalità dell'applicazione.
- Descrivere le proprietà qualitativi (robustezza, usabilità, portabilità) dell'app.
- Documentare il design, il codice sorgente e le strategie di testing.
- Fornire indicazioni per il deployment, la manutenzione e l'evoluzione futura del software.

1.4 Panoramica del Documento

Il documento è strutturato in 10 sezioni principali:

1. Introduzione
2. Architettura del Software
3. Analisi delle Proprietà
4. Design del Software
5. Documentazione del Codice
6. Testing
7. Deployment e Portabilità
8. Manutenzione del Software

9. Conclusioni

10. Appendici

2. Architettura del Software

2.1 Descrizione Generale dell'Architettura

L'applicazione segue un'architettura **MVC semplificata**:

- **Modello**: I dati sono salvati in file JSON (users.json e posts.json). Le funzioni `readUsers()`, `writeUsers()`, `readPosts()` e `writePosts()` gestiscono la persistenza.

- **Vista**: Le interfacce utente sono realizzate con EJS. I template sono organizzati nella cartella `views` con partial per la navbar e il footer.

- **Controller**: Le rotte di Express nel file `server.js` fungono da controller, gestendo le richieste HTTP, elaborando i dati e renderizzando le view.

2.2 Componenti Principali

- **Express**: Server web e routing.
- **EJS**: Motore di template per le view.

- **Body-parser**: Estrazione dei dati POST.

- **Express-session**: Gestione delle sessioni utente.

- **Multer**: Caricamento e gestione dei file (immagini e foto profilo).

- **File System (fs)**: Operazioni di lettura/scrittura su file JSON.

2.3 Flusso di Dati

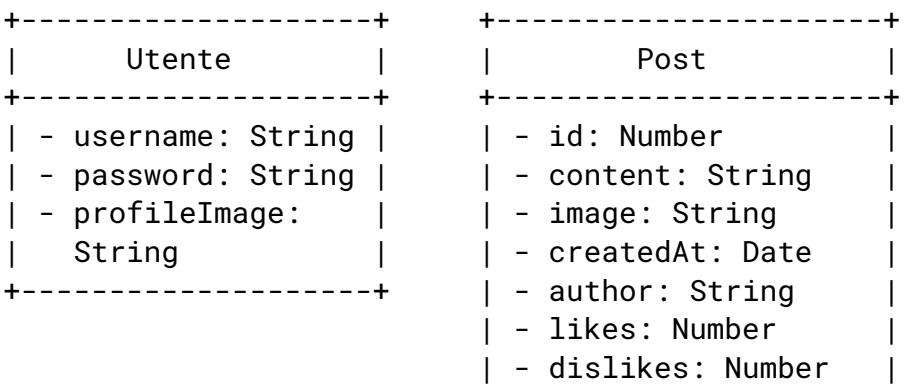
- Un utente interagisce con il front-end (view EJS) inviando dati via form.
- Le richieste vengono elaborate dalle rotte di Express.

- I dati vengono letti o scritti in file JSON tramite le funzioni `readUsers()` e `writeUsers()` per gli utenti e `readPosts()` e `writePosts()` per i post.

- Le view vengono renderizzate dinamicamente includendo dati come l'username, i post e le informazioni degli utenti.

2.4 Diagrammi UML

Diagramma delle Classi (Concettuale)



```
| - likedBy: [String]  |
| - dislikedBy:[String]|
| - rating: [Object]   |
|   { user, stars }    |
| - comments: [Object] |
|   { author, text }   |
+----------------------+
```

Diagramma di Sequenza (Login)

1. L'utente invia i dati di login dal form.

2. La rotta `/login` riceve i dati, chiama `readUsers()` e verifica le credenziali.

3. Se le credenziali sono corrette, il controller salva l'username nella sessione e reindirizza alla home.

3. Analisi delle Proprietà

3.1 Robustezza

3.1.1 Definizione di Robustezza nel Contesto del Software

La robustezza indica la capacità del software di gestire input non previsti o errori senza andare in crash o comportarsi in maniera anomala.

3.1.2 Gestione degli Errori

- Verifica dell'esistenza dei file JSON prima di leggere i dati.
- Gestione di casi di errore nelle rotte (es. se un post non viene trovato).
- Uso di condizioni di controllo nelle rotte per assicurarsi che l'utente sia autenticato.

3.1.3 Test di Robustezza

Sono stati eseguiti test per verificare:

- Comportamento in caso di file JSON non esistenti.
- Gestione delle richieste con dati mancanti (ad es. post senza contenuto o immagine).
- Gestione delle rotte protette (richiesta di azioni da parte di utenti non autenticati).

3.1.4 Resilienza del Software

Il software continua a funzionare anche in presenza di errori controllati, grazie a:

- Controlli condizionali nelle rotte.
- Messaggi di errore informativi (es. "Post non trovato").

3.1.5 Codice Documentato per la Gestione degli Errori

I blocchi condizionali nelle rotte (ad es. verifica se `!post`) sono stati commentati per descrivere la logica di gestione degli errori.

3.2 Usabilità

3.2.1 Definizione di Usabilità nel Contesto del Software

L'usabilità definisce la facilità con cui un utente può imparare a utilizzare il sistema, completare compiti e ottenere una buona esperienza.

3.2.2 Interfaccia Utente (UI) e User Experience (UX)

- L'interfaccia è basata su Bootstrap, garantendo un design reattivo ed estetico.
- Le view sono strutturate con layout chiari: header (navbar), container centrale e footer.
- I form sono ben strutturati e forniscono placeholder ed etichette chiare.

3.2.3 Accessibilità

- Utilizzo di tag semantici HTML.
- Campi con etichette (label) e attributi required per guidare l'utente.

3.2.4 Feedback degli Utenti
- Messaggi di successo e di errore (es. "Registrazione effettuata con successo", "Credenziali non valide").

3.2.5 Codice Documentato per la UI/UX
Nei file EJS sono stati aggiunti commenti per indicare la logica di visualizzazione (es. se l'utente è loggato, mostra form, ecc.) e suggerimenti per migliorare l'esperienza utente.

3.3 Portabilità
3.3.1 Definizione di Portabilità nel Contesto del Software
La portabilità esprime la capacità del software di funzionare su più piattaforme e ambienti senza modifiche sostanziali.

3.3.2 Compatibilità tra Piattaforme
- L'app è scritta in Node.js, garantendo compatibilità cross-platform (Windows, macOS, Linux).
- L'uso di Express e middleware standard assicura un ambiente stabile.

3.3.3 Dipendenze Esterne e Gestione della Configurazione
- Le dipendenze (express, multer, ecc.) sono gestite via npm.
- La configurazione (porta, secret della sessione) è centralizzata nel file principale.

3.3.4 Strategie di Testing Cross-Platform
- Test manuali e, se necessario, script di test in ambienti diversi per verificare la corretta lettura/scrittura dei file e il corretto funzionamento delle rotte.

3.3.5 Codice Documentato per la Portabilità
I moduli e le funzioni utilizzati sono standard e commentati per facilitare l'adattamento su altri ambienti.

4. Design del Software
4.1 Principi di Design Adottati
- **Modularità**: Il codice è suddiviso in moduli (rotte, funzioni di lettura/scrittura) per facilitare la manutenzione.
- **Separation of Concerns**: Le funzioni per la persistenza dei dati sono separate dalla logica del controllo HTTP.
- **Semplicità**: L'app sfrutta file JSON per la persistenza, rendendo l'implementazione semplice e facile da estendere.

4.2 Pattern di Design Utilizzati
- **MVC semplificato**: Anche se i dati sono gestiti direttamente tramite file JSON, la divisione in view (EJS), controller (rotte Express) e modelli (funzioni di persistenza) è evidente.
- **Middleware Pattern**: Viene utilizzato per la gestione delle sessioni e per rendere disponibili variabili globali nelle view.

4.3 Scelte Architetturali e Motivi
- **Persistenza tramite JSON**: Adatto per un prototipo o ambienti a bassa scala, semplice da configurare.
- **Uso di Express e EJS**: Permette uno sviluppo rapido e semplice di applicazioni web dinamiche.
- **Caricamento file tramite multer**: Necessario per gestire upload di immagini in modo sicuro e gestito.

5. Documentazione del Codice
5.1 Struttura del Codice Sorgente
- **app.js (o server.js)**: File principale contenente la logica dell'applicazione, rotte,

configurazioni e funzioni helper.
- **views/**: Cartella contenente i template EJS (index.ejs, login.ejs, signup.ejs, update-profile.ejs).
- **partials/**: Sottocartella in `views` contenente parti comuni alle view, ad esempio navbar e footer.
- **public/**: Cartella per i file statici, inclusi CSS, immagini e file caricati (uploads).

5.2 Classi e Metodi Principali

- **Funzioni di persistenza**:
  - `readUsers()` / `writeUsers()`: Leggono e scrivono nel file `users.json`.
  - `readPosts()` / `writePosts()`: Leggono e scrivono nel file `posts.json`.
- **Rotte Express**: Gestiscono login, registrazione, aggiornamento profilo, creazione e interazione con i post (commenti, like, dislike, rating).

5.3 Commenti e Note Importanti nel Codice

- Sono presenti commenti dettagliati (come in questo documento) che spiegano lo scopo di ogni sezione.
- Note relative alla gestione degli errori (verifica dell'esistenza dei file JSON, controllo delle sessioni).

5.4 Esempi di Utilizzo del Codice

- **Registrazione**: Un utente visita `/signup`, compila il form (incluso il caricamento della foto profilo) e viene salvato in `users.json`.
- **Login**: L'utente invia username e password a `/login` e, se validi, viene impostata la sessione.
- **Post**: Un utente loggato invia un post (testo e/o immagine) a `/post` e il post viene salvato in `posts.json`.
- **Interazioni**: L'utente può commentare (`/comment/:id`), votare o cambiare voto (rotte `/like/:id`, `/dislike/:id` e `/rate/:id`).

5.5 Snippet di Codice per le Proprietà Analizzate

Robustezza:

```
// Esempio di controllo: verifica se il post esiste prima di
aggiungere un commento
if (!post) {
    return res.status(404).send('Post non trovato');
}
```

Usabilità:

```
<!-- Visualizzazione condizionale: mostra il form per il post solo
se loggato -->
<% if (username) { %>
    <form method="post" action="/post"
enctype="multipart/form-data">...</form>
<% } else { %>
    <p>Accedi per poter creare un nuovo post.</p>
<% } %>
```

Portabilità:

```
// Le funzioni read/write usano fs ed operano su file JSON, rendendo
il sistema indipendente dalla piattaforma
```

```
function readUsers() {
    if (!fs.existsSync(usersFile)) return [];
    return JSON.parse(fs.readFileSync(usersFile));
}
```

6. Testing

6.1 Strategia di Testing

- Esecuzione di test manuali sulle rotte principali.
- Verifica della corretta gestione delle sessioni e dei file JSON.
- Uso di console.log e strumenti di debug per tracciare i flussi.

6.2 Test Unitari

- Test delle funzioni di lettura/scrittura (readUsers, writePosts, ecc.) per garantire la persistenza dei dati.

6.3 Test di Integrazione

- Simulazione di richieste POST e GET (per login, post, commenti, ecc.) e verifica della corretta risposta e aggiornamento dei file JSON.

6.4 Test di Performance e Stress

- Verifica del comportamento con numerosi post e utenti.
- Controllo del tempo di risposta e gestione degli errori in caso di input massivo.

6.5 Test Cross-Platform

- Test su diversi sistemi operativi (Windows, macOS, Linux) per verificare la portabilità del codice.

6.6 Risultati dei Test e Report

- I test manuali hanno confermato il corretto funzionamento nelle funzionalità base.
- I risultati sono documentati tramite log e feedback visuale nelle view.

7. Deployment e Portabilità

7.1 Guida al Deployment

- L'app può essere avviata con `node app.js` (o `npm start`) su un ambiente di sviluppo.
- Assicurarsi di avere Node.js installato.

7.2 Ambienti di Produzione e Sviluppo

- **Sviluppo**: Utilizzo di file JSON per la persistenza, semplice ed immediato.
- **Produzione**: In caso di necessità, considerare il passaggio a un database (ad esempio, MongoDB) e un sistema di gestione delle sessioni distribuito.

7.3 Esempi di Configurazione

- Configurazione della porta, secret della sessione e percorsi dei file sono definiti nel file principale e possono essere resi variabili d'ambiente.

7.4 Considerazioni sulla Portabilità durante il Deployment

- L'app utilizza moduli standard e file JSON, garantendo che possa essere eseguita su qualunque piattaforma che supporti Node.js.
- Le dipendenze sono gestite tramite npm, facilitando l'installazione su ogni ambiente.

8. Manutenzione del Software

8.1 Strategie di Manutenzione

- Mantenere il codice ben documentato e commentato.
- Suddividere il codice in moduli qualora il progetto cresca, per migliorarne la leggibilità e manutenibilità.

8.2 Aggiornamenti del Software
- Documentare ogni nuova funzionalità e modifiche apportate al codice.
- Utilizzare il versionamento (ad es. Git) per tracciare le modifiche.

8.3 Bug Fixing e Refactoring
- Utilizzare strumenti di debugging e test automatici per individuare e correggere bug.
- Eseguire refactoring periodico per migliorare l'architettura del codice.

8.4 Backup e Recupero dei Dati
- Conservare copie di backup dei file JSON (o del database) per prevenire la perdita di dati.
- Automatizzare il backup se l'app viene distribuita in un ambiente di produzione.

9. Conclusioni
9.1 Sintesi delle Proprietà Analizzate
- **Robustezza**: L'app gestisce correttamente input non validi e offre messaggi di errore appropriati.
- **Usabilità**: L'interfaccia è semplice e reattiva grazie a Bootstrap ed EJS.
- **Portabilità**: L'app è sviluppata in Node.js ed è indipendente dalla piattaforma grazie all'uso di moduli standard.

9.2 Suggerimenti Futuri per il Miglioramento
- Migrare la persistenza dei dati su un database (MongoDB, MySQL, ecc.) per scalabilità.
- Integrare test automatici (unitari e di integrazione).
- Migliorare la UI/UX con framework frontend più avanzati (React, Vue, ecc.).
- Implementare funzionalità avanzate (notifiche in tempo reale, ricerca, filtri).

9.3 Lezioni Apprese durante lo Sviluppo
- L'importanza della documentazione e dei commenti nel codice.
  L'utilizzo di middleware per gestire sessioni e dati globali.
- La necessità di testare e validare ogni funzionalità dell'app per garantire robustezza e portabilità.

10. Appendici
10.1 Glossario
- **EJS**: Embedded JavaScript templating, utilizzato per generare le view HTML dinamiche.
- **Express**: Framework per Node.js per creare applicazioni web.
- **Multer**: Middleware per la gestione dell'upload di file.
- **Sessione**: Meccanismo per mantenere lo stato tra richieste HTTP.
- **JSON**: Formato di testo per lo scambio di dati, usato qui per la persistenza dei dati.

10.2 Riferimenti e Bibliografia
- Documentazione Express
- Documentazione EJS
- Documentazione Multer
- Node.js Documentation

10.3 Link Utili e Risorse Esterne
- Bootstrap 5 Documentation
- Guide e Tutorial Node.js
- GitHub