Guida all'Implementazione - Esame del 27 Giugno 2025

June 28, 2025

Contents

1 Introduzione

Questa guida fornisce una soluzione completa e dettagliata per tutte le richieste dell'esame del 27 Giugno 2025[cite: 1]. Verranno analizzati e implementati un server Node.js per la gestione dei dati e un'applicazione web front-end (HTML, CSS, JavaScript) per l'interazione con l'utente. Ogni sezione di codice è accompagnata da una spiegazione riga per riga per chiarire ogni passaggio.

2 Struttura del Progetto

Per una corretta organizzazione, il progetto sarà strutturato nella seguente maniera:

```
/progetto-esame
|-- /public
|   |-- index.html
|   |-- style.css
|   `-- script.js
|-- data.json
`-- server.js
```

- **public/**: Cartella che conterrà tutti i file statici del front-end.
- **data.json**: File JSON che funge da piccolo database per la nostra lista di attività[cite: 2].
- **server.js**: Il server Node.js che gestirà le richieste API e servirà i file del front-end[cite: 11].

2.1 Preparazione dell'Ambiente

Prima di iniziare, assicurati di avere Node.js installato. Crea la cartella del progetto e installa la libreria `express`, che semplifica la creazione di server web in Node.js.

```
# Crea e naviga nella cartella del progetto
mkdir progetto-esame
cd progetto-esame

# Inizializza un progetto Node.js
npm init -y

# Installa Express
npm install express

# Crea i file e le cartelle necessarie
mkdir public
touch public/index.html public/style.css public/script.js data.json server.js
```

3 File di Dati (data.json)

Questo file contiene la lista di attività come specificato nell'esame[cite: 2].

```
[
  { "id": 1, "testo": "Comprare il pane", "completato": false },
  { "id": 2, "testo": "Fare esercizio", "completato": true },
  { "id": 3, "testo": "Leggere un libro", "completato": false },
  { "id": 4, "testo": "Studiare per l'esame", "completato": true },
  { "id": 5, "testo": "Pulire la stanza", "completato": false },
  { "id": 6, "testo": "Scrivere il progetto", "completato": false }
]
```

Listing 1: data.json

Spiegazione:

- **Righe 1-7**: Definiscono un array di oggetti JSON[cite: 3, 4]. Ogni oggetto rappresenta un'attività con un `id` univoco, un `testo` descrittivo e uno stato booleano `completato`[cite: 5, 6, 7, 8, 9, 10].

4 Codice del Server (server.js)

Questo script crea un server Express che espone le API richieste e serve la pagina web.

```
1  const express = require('express');
2  const fs = require('fs');
3  const path = require('path');
4
5  const app = express();
6  const PORT = 3000;
7
8  // Middleware per servire i file statici dalla cartella 'public'
9  app.use(express.static(path.join(__dirname, 'public')));
10
11 // Endpoint GET /items
12 app.get('/items', (req, res) => {
13   fs.readFile(path.join(__dirname, 'data.json'), 'utf8', (err, data) => {
14     if (err) {
15       res.status(500).send("Errore nella lettura del file");
16       return;
17     }
18     res.json(JSON.parse(data));
19   });
20 });
21
22 // Endpoint GET /items-complete
23 app.get('/items-complete', (req, res) => {
24   fs.readFile(path.join(__dirname, 'data.json'), 'utf8', (err, data) => {
25     if (err) {
26       res.status(500).send("Errore nella lettura del file");
27       return;
28     }
29     const items = JSON.parse(data);
30     const completedItems = items.filter(item => item.completato === true);
31     res.json(completedItems);
32   });
33 });
34
35 // Avvio del server
36 app.listen(PORT, () => {
37   console.log(`Server in ascolto sulla porta ${PORT}`);
38 });
```

Listing 2: server.js

4.1 Spiegazione Dettagliata (server.js)

- **Riga 1-3**: Importa i moduli necessari: `express` per il server, `fs` (File System) per leggere il file `data.json`, e `path` per gestire i percorsi dei file.
- **Riga 5-6**: Crea un'istanza dell'applicazione Express e definisce la porta su cui girerà il server.
- **Riga 9**: `app.use(express.static(...))` è un middleware che dice a Express di servire i file statici (HTML, CSS, JS) presenti nella cartella `public`.
- **Riga 12-20**: Definisce l'endpoint `GET /items`[cite: 12].
  - **13**: Legge il file `data.json` in modo asincrono.
  - **14-17**: Gestisce eventuali errori durante la lettura del file.
  - **18**: Se la lettura ha successo, converte il contenuto del file (che è una stringa) in un oggetto JSON con `JSON.parse` e lo invia come risposta.
- **Riga 23-32**: Definisce l'endpoint `GET /items-complete`[cite: 13].
  - **24-28**: Legge il file `data.json`, gestendo gli errori.
  - **29**: Converte il contenuto testuale in un array JavaScript.
  - **30**: Usa il metodo `filter` per creare un nuovo array contenente solo gli elementi con la proprietà `completato` uguale a `true`[cite: 13].

  - **31**: Invia l'array filtrato come risposta JSON.

- **Riga 35-37**: Avvia il server sulla porta specificata (`3000`) e stampa un messaggio di conferma sulla console.

5 Codice HTML (public/index.html)

Questo è il file che definisce la struttura della pagina web.

```
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista Attivita</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Lista Attivita</h1>
        </header>

        <div class="theme-switcher">
            <a href="#" id="theme-light">Chiaro</a> | <a href="#" id="theme-dark">Scuro</a>
        </div>

        <div class="filters">
            <button id="btn-all">Mostra Tutti</button>
            <button id="btn-completed">Solo Completati</button>
        </div>

        <main id="task-list">
            </main>

        <footer>
            <p>&copy; 2024</p>
        </footer>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

Listing 3: public/index.html

5.1 Spiegazione Dettagliata (index.html)

- **Riga 1-9**: Intestazione standard di un file HTML5. Include la dichiarazione del doctype, la lingua, il charset, il viewport per il responsive design, il titolo della pagina e il link al foglio di stile CSS.

- **Riga 11**: `<div class="container">` è il contenitore principale che centra la pagina e ne definisce la larghezza[cite: 15].

- **Riga 12-14**: `<header>` contiene il titolo principale della pagina[cite: 16].

- **Riga 16-18**: Sezione per il cambio tema, con due link per 'Chiaro' e 'Scuro'[cite: 17, 18].

- **Riga 20-23**: Sezione per i filtri, con i due pulsanti 'Mostra Tutti' e 'Solo Completati'[cite: 19].

- **Riga 25-27**: L'elemento `<main>` con id `task-list` è il contenitore dove JavaScript inserirà dinamicamente le attività caricate dal server[cite: 23].

- **Riga 29-31**: `<footer>` con il testo del copyright[cite: 16].

- **Riga 34**: Inclusione del file JavaScript prima della chiusura del `<body>` per assicurarsi che il DOM sia completamente caricato prima dell'esecuzione dello script.

6 Codice CSS (public/style.css)

Questo file contiene tutte le regole di stile per l'applicazione, inclusi il layout, i temi e il design responsivo.

```css
/* Stili Generali e Tema Chiaro (default) */
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    color: #333;
    margin: 0;
    transition: background-color 0.3s, color 0.3s;
}

.container {
    min-width: 400px;
    max-width: 800px;
    margin: 20px auto;
    padding: 0 20px;
}

header, footer {
    height: 80px;
    background-color: #fff;
    border: 1px solid #alalal;
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 24px;
    font-weight: bold;
    transition: background-color 0.3s, border-color 0.3s;
}

.theme-switcher {
    text-align: center;
    padding: 10px 0;
}

.filters {
    text-align: center;
    margin-bottom: 20px;
}

.task-item {
    background-color: #fff;
    border: 1px solid #alalal;
    border-radius: 8px;
    padding: 15px;
    margin-bottom: 10px;
    display: flex;
    flex-direction: column; /* Mobile first: layout verticale */
    transition: background-color 0.3s, border-color 0.3s;
}

.task-item .task-text {
    font-size: 16px;
    margin-bottom: 10px; /* Spazio tra testo e bottone su mobile */
}

.task-item.completed .task-text {
    text-decoration: line-through;
    color: #888;
}

.task-item button {
    padding: 8px 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    cursor: pointer;
    align-self: flex-end; /* Allinea il bottone a destra nel layout a colonna */
}

.task-item button:disabled {
    background-color: #e0e0e0;
```

```
70      cursor: not-allowed;
71  }
72
73  /* Responsive Design per schermi piu grandi */
74  @media (min-width: 601px) {
75      .task-item {
76          flex-direction: row; /* Layout orizzontale */
77          justify-content: space-between;
78          align-items: center;
79      }
80
81      .task-item .task-text {
82          margin-bottom: 0; /* Rimuove il margine inferiore */
83      }
84
85      .task-item button {
86          align-self: auto; /* Resetta l'allineamento */
87      }
88  }
89
90  /* Tema Scuro */
91  body.dark-theme {
92      background-color: #222;
93      color: #f0f0f0;
94  }
95
96  body.dark-theme .container {
97      /* Nessun cambiamento specifico richiesto per il container */
98  }
99
100 body.dark-theme header,
101 body.dark-theme footer {
102     background-color: #111;
103     border-color: #666;
104 }
105
106 body.dark-theme .task-item {
107     background-color: #444;
108     border-color: #666;
109 }
110
111 body.dark-theme .task-item.completed .task-text {
112     color: #aaa;
113 }
```

Listing 4: public/style.css

6.1 Spiegazione Dettagliata (style.css)

- **Righe 2-9**: Stili di base per il `<body>`. Imposta il font Arial, i colori di sfondo e testo per il tema chiaro[cite: 16, 21, 30], e una transizione per un cambio tema più fluido.
- **Righe 11-16**: Stili per `.container`. Imposta la larghezza minima/massima e i margini per centrare il layout[cite: 15].
- **Righe 18-28**: Stili per `header` e `footer`. Definisce altezza di 80px, sfondo, bordo e utilizza Flexbox per centrare il testo verticalmente e orizzontalmente[cite: 16, 17].
- **Righe 30-38**: Stili per i contenitori del selettore di tema e dei filtri, centrando il loro contenuto. [cite: 18, 19]
- **Righe 40-52**: Stili per ogni attività (`.task-item`) in modalità mobile-first[cite: 24, 25]. Ha uno sfondo bianco, bordo, padding e margine[cite: 20]. `display: flex` e `flex-direction: column` creano un layout verticale.
- **Righe 54-57**: Barra il testo dell'attività se essa è completata (`.task-item.completed`)[cite: 21].
- **Righe 59-69**: Stili per il pulsante 'Completa', incluso lo stato disabilitato[cite: 22].

- **Righe 72-85**: Media Query per schermi con larghezza superiore a 600px[cite: 26]. Cambia la `flex-direction` di `.task-item` in `row` per un layout orizzontale e allinea gli elementi.
- **Righe 88-109**: Definisce gli stili per il tema scuro[cite: 29]. Quando il `<body>` ha la classe `dark-theme`, questi stili sovrascrivono quelli di default per cambiare i colori di sfondo, testo, header, footer, box e bordi come richiesto[cite: 30].

7 Codice JavaScript (public/script.js)

Questo script gestisce la logica del front-end: caricamento dei dati, aggiornamento dinamico della lista e cambio tema.

```
document.addEventListener('DOMContentLoaded', () => {
    const taskList = document.getElementById('task-list');
    const btnAll = document.getElementById('btn-all');
    const btnCompleted = document.getElementById('btn-completed');
    const btnThemeLight = document.getElementById('theme-light');
    const btnThemeDark = document.getElementById('theme-dark');

    // Funzione per renderizzare le attivita nella pagina
    const renderTasks = (tasks) => {
        taskList.innerHTML = ''; // Pulisce la lista corrente
        if (tasks.length === 0) {
            taskList.innerHTML = '<p>Nessuna attivita da mostrare.</p>';
            return;
        }

        tasks.forEach(task => {
            const item = document.createElement('div');
            item.classList.add('task-item');
            if (task.completato) {
                item.classList.add('completed');
            }

            const text = document.createElement('span');
            text.classList.add('task-text');
            text.textContent = task.testo;

            const button = document.createElement('button');
            button.textContent = 'Completa';
            if (task.completato) {
                button.disabled = true;
            }

            item.appendChild(text);
            item.appendChild(button);
            taskList.appendChild(item);
        });
    };

    // Funzione per caricare i dati da un URL e renderizzarli
    const fetchAndRenderTasks = async (url) => {
        try {
            const response = await fetch(url);
            const tasks = await response.json();
            renderTasks(tasks);
        } catch (error) {
            console.error('Errore nel caricamento delle attivita:', error);
            taskList.innerHTML = '<p>Impossibile caricare le attivita.</p>';
        }
    };

    // Gestori degli eventi per i filtri
    btnAll.addEventListener('click', () => fetchAndRenderTasks('/items'));
    btnCompleted.addEventListener('click', () => fetchAndRenderTasks('/items-complete'))
    ;

    // Gestori degli eventi per il cambio tema
    btnThemeLight.addEventListener('click', (e) => {
        e.preventDefault();
        document.body.classList.remove('dark-theme');
```

```
59      });
60
61      btnThemeDark.addEventListener('click', (e) => {
62          e.preventDefault();
63          document.body.classList.add('dark-theme');
64      });
65
66      // Caricamento iniziale delle attivita
67      fetchAndRenderTasks('/items');
68  });
```

Listing 5: public/script.js

7.1 Spiegazione Dettagliata (script.js)

- **Riga 1**: L'evento `DOMContentLoaded` assicura che lo script venga eseguito solo dopo che l'intera pagina HTML è stata caricata e analizzata.
- **Righe 2-6**: Seleziona e memorizza in costanti gli elementi del DOM con cui interagiremo.
- **Riga 9-36**: La funzione `renderTasks` si occupa di visualizzare la lista di attività.
  - **10**: Svuota il contenuto precedente di `taskList`.
  - **11-14**: Mostra un messaggio se non ci sono attività da visualizzare.
  - **16-35**: Itera su ogni attività (`task`) nell'array ricevuto[cite: 23]. Per ognuna, crea dinamicamente gli elementi HTML (`<div>`, `<span>`, `<button>`), assegna le classi CSS appropriate (inclusa `completed` se l'attività è completata [cite: 21]), imposta il testo e disabilita il pulsante se necessario[cite: 22]. Infine, aggiunge l'elemento completo alla lista nel DOM.
- **Riga 39-48**: La funzione `fetchAndRenderTasks` è una funzione asincrona che gestisce le chiamate API.
  - **41**: Esegue una chiamata `fetch` all'URL specificato (es. `/items`).
  - **42**: Converte la risposta in formato JSON.
  - **43**: Chiama `renderTasks` per mostrare i dati ottenuti.
  - **44-47**: Gestisce eventuali errori che potrebbero verificarsi durante la chiamata di rete.
- **Riga 51**: Aggiunge un event listener al pulsante 'Mostra Tutti'. Al click, richiama la funzione per caricare e renderizzare i dati dall'endpoint `/items`[cite: 28].
- **Riga 52**: Aggiunge un event listener al pulsante 'Solo Completati'. Al click, richiama la funzione per caricare e renderizzare i dati dall'endpoint `/items-complete`[cite: 27].
- **Riga 55-63**: Aggiunge gli event listener ai link per il cambio tema[cite: 29].
  - `e.preventDefault()` previene il comportamento predefinito del link (navigare a '').
  - A seconda del link cliccato, aggiunge o rimuove la classe `dark-theme` dal `<body>`, attivando le regole CSS corrispondenti[cite: 29].
- **Riga 66**: Esegue un caricamento iniziale di tutte le attività chiamando `fetchAndRenderTasks('/items')` non appena lo script viene eseguito[cite: 23].

8 Esecuzione del Progetto

Per avviare l'applicazione completa, esegui il seguente comando dalla radice del progetto:

```
node server.js
```

Apri il tuo browser e naviga all'indirizzo `http://localhost:3000` per visualizzare e interagire con l'applicazione.