Domande teoriche - TPS 5IA/5IB
Anno scolastico 2024/2025

Prof. Gianluca Pironato

Domande teoriche

1. Cos'è Il World Wide Web Consortium?

Il World Wide Web Consortium, noto come W3C, è un'organizzazione internazionale che sviluppa standard e linee guida per il Web. È stato fondato da Tim Berners-Lee (l'inventore del Web) nel 1994 e ha sede al MIT (Massachusetts Institute of Technology) negli Stati Uniti, ma ha centri anche in Europa e in Asia.

Il suo obiettivo è garantire che il Web sia aperto, accessibile a tutti e funzioni in modo uniforme in tutto il mondo. I documenti e le raccomandazioni del W3C stabiliscono come devono funzionare le tecnologie e i protocolli del Web, come l'HTML, i fogli di stile CSS e molti altri standard fondamentali.

Riccardo Auriemma

2. Chi è Tim Berners Lee?

Tim Beernes Lee è un informatico britannico che nel 1991 pubblicò il primo sito web sul World Wide Web, la scelta deriva dal fatto che scienziati provenienti da tutto il mondo usavano sistemi informatici diversi e questo ostacolava la condivisione delle informazioni.

Nacque quindi l'idea di creare un sistema di gestione delle informazioni accessibile a tutti, basato su tre tecnologie fondamentali: l'hypertext, che permette di collegare documenti tramite link; il protocollo TCP/IP, per il trasporto dei dati; e il DNS, che assegna nomi ai computer collegati alla rete.

Il suo obiettivo principale era quello garantire accesso universale a una vasta gamma di documenti, purtroppo oggi, questa visione viene talvolta ostacolata da restrizioni imposte da alcuni governi.

Maggiori informazioni sono presenti in questo video in cui si parla di Tim Beernes Lee e della creazione del World Wide Web.

Federico Bonalumi

3. Cos'è un linguaggio di markup e quale ruolo ha HTML nella creazione di pagine web?

Un linguaggio di markup è un insieme di regole usato per indicare la struttura, la formattazione e il significato di un documento di testo. Diversamente dai linguaggi di programmazione un linguaggio di markup non esegue calcoli o algoritmi, ma definisce come deve essere interpretato e visualizzato un testo.

Alcuni esempi sono:
- HTML (HyperText Markup Language), usato per creare pagine web.

- XML (eXtensible Markup Language) usato per descrivere dati in modo flessibile e strutturato.

L'HTML è il linguaggio standard utilizzato per la creazione e la progettazione di siti web. Definisce la struttura e il layout del contenuto, incorporando elementi come testo, immagini, link e file multimediali. Consiste in una serie di elementi, ciascuno contenuto in tag, che istruiscono i browser su come interpretare e rappresentare il contenuto. Quando un utente accede a una pagina web, il browser recupera il codice HTML dal server web Il browser analizza quindi il documento HTML, interpretando i tag e gli elementi per determinare la struttura e il layout della pagina.

Matteo Camnasio

4. Illustra in modo preciso cosa avviene tra client e server quando l'utente visita una pagina web che contiene un'immagine.

Quando un utente visita una pagina web che contiene un'immagine, il processo tra client e server avviene secondo i seguenti passaggi

1. Risoluzione del dominio: l'utente inserisce l'URL nel browser (es. www.esempio.com). Il browser interroga un server DNS per ottenere l'indirizzo IP associato a quel dominio. Una volta ottenuto l'IP, inizia la comunicazione con il server web.

2. Richiesta HTTP della pagina HTML: il browser (client) apre una connessione TCP verso il server, solitamente sulla porta 80 (HTTP) o 443 (HTTPS) e invia una richiesta http per la risorsa.

3. Risposta del server con la pagina HTML: il codice HTML della pagina viene dato in risposta HTTP inviata dal server al client

4. Parsing del codice HTML: il browser inizia a leggere (fa parsing) il file HTML. Quando incontra il tag <img src="immagini/foto.jpg">, capisce che deve richiedere un'altra risorsa: l'immagine.

5. Richiesta HTTP dell'immagine: il browser invia una nuova richiesta HTTP (separata) al server per l'immagine.

6. Risposta del server con l'immagine: il server risponde con il file immagine (es. JPEG, PNG)

7. Rendering della pagina: il browser riceve l'immagine e la integra nel DOM (Document Object Model). Mostra la pagina completa all'utente, con l'immagine visualizzata al punto indicato nel codice HTML.

Leonardo Cazzaniga 2005, con il supporto di qualche LLM

5. Quale ruolo ha CSS e com'è sintatticamente organizzato?

CSS (Cascading Style Sheets) è un linguaggio utilizzato per descrivere l'aspetto e la formattazione di un documento scritto in HTML o XML. In pratica, CSS controlla

il layout, i colori, i font, gli spazi, le animazioni e altri aspetti visivi delle pagine web, separando il contenuto (HTML) dalla presentazione.

Alcune caratteristiche di CSS:
· Separazione tra contenuto e stile: l'HTML struttura le informazioni, il CSS le rende visivamente gradevoli e leggibili.

· Controllo centralizzato dello stile: un singolo file CSS può controllare lo stile di un intero sito web.

· Responsività: CSS permette di creare design che si adattano a schermi di dimensioni diverse (desktop, tablet, smartphone).

· Accessibilità e mantenibilità: modificando un solo file CSS si può aggiornare facilmente lo stile di tutto il sito.

La sintassi base di CSS è composta da selettori e dichiarazioni:

```
selettore {
    proprietà: valore;
    proprietà2: valore2;
    }
```

Dove si scrive il CSS: inline (dentro un tag HTML), interno (nel <head>) oppure esterno (in un file .css).

Leonardo Cazzaniga 2006

6. Qual è lo scopo di XML? Qual è lo scopo di JSON? Quali sono le differenze?

XML(eXtensible Markup Language) è un linguaggio di markup simile a HTML, designato per salvare e trasportare dati. Inoltre è anche auto-descrittivo, ossia che i dati sono accompagnati da tag che spiegano cosa rappresentano; quindi, è possibile capire il contenuto anche senza documentazione esterna. (è un insieme di informazioni descritto da tag)

```
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

Anche JSON (JavaScript Object Notation) è un formato per salvare e trasportare i dati, spesso utilizzato quando i dati vengono trasmessi da un server a una pagina web. La sintassi di JSON deriva dalla notazione degli oggetti di JavaScript, ma il formato JSON è solo testo. Il codice per leggere e generare dati JSON può essere scritto in qualsiasi linguaggio di programmazione.

```
{"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
 ]
}
```

JSON e XML sono entrambi formati utilizzati per rappresentare e scambiare dati strutturati, ma differiscono per sintassi, leggibilità, flessibilità e uso comune. Ecco le differenze principali:
- JSON usa una sintassi più semplice e compatta, simile a quella degli oggetti JavaScript
- XML usa tag di apertura e chiusura, più prolissi
- JSON è più facile da leggere e scrivere per gli esseri umani
- XML può risultare più complesso e prolisso
- JSON ha tipi di dato nativi: stringhe, numeri, booleani, array, oggetti
- XML tratta tutto come testo; i tipi vanno interpretati a parte
- JSON è ideale per rappresentare oggetti e array (dati strutturati gerarchici)
- XML è più adatto a documenti con metadati, attributi e contenuti misti

Luca Conti

7. Qual è la differenza tra tag semantici e non semantici in HTML? Quando arrivano i tag semantici?

I tag semantici sono elementi di codice che contribuiscono a rendere la pagina chiara e leggibile: indicano al browser utilizzato dall'utente come leggere la pagina, stabilendone quindi la struttura. Indicano il significato del contenuto che racchiudono: ad esempio, il tag ¡header¿ racchiude solitamente l'intestazione della pagina o la prima sezione. Il codice diventa così più leggibile e organizzato.

Invece, i tag non semantici non forniscono informazioni riguardo al contenuto che racchiudono. Un esempio è il tag ¡div¿, infatti non ci da nessuna informazione riguardo a ciò che può contenere, è un semplice contenitore di blocco per raggruppare una parte di codice.

I tag semantici sono arrivati con HTML5, per migliorare la struttura e la leggibilità del codice, sia da parte di un programmatore che da parte del browser.

Alessandro D'Abrusco

8. A cosa serve l'attributo `action` di un form HTML? Che problemi può dare se i dati del form devono essere acquisiti da uno script JS?

L'attributo action di un tag <form> serve a specificare l'endpoint al quale devono essere inviati i dati del form quando l'utente preme il pulsante di invio. Di solito questo URL punta a uno script lato server (come action_page.php) che riceve, elabora e gestisce i dati del form.

```
<form action="action_page.php" method="post">
 <input type="text" name="nome">
 <input type="submit" value="Invia">
</form>
```

Problemi con JS: principalmente, se viene premuto il tasto di invio, il browser manda subito i dati al server e cambia pagina. Questo può bloccare lo script JavaScript e così non avrà il tempo di fare nulla.

Marco Donadoni

9. Che differenza c'è tra GET e POST nella trasmissione dei dati mediante HTTP? Illustra in dettaglio dove vengono scritti i dati da inviare

Nella trasmissione dei dati mediante HTTP, la differenza tra il metodo GET e il metodo POST è la seguente:

GET: Viene utilizzato per ottenere risorse dal server, come ad esempio pagine web o risultati di una ricerca. I dati da inviare (come parametri) vengono scritti nell'URL sotto forma di query string, ad esempio:
"https://esempio.com/cerca?categoria=libri&autore=verga"

- I dati sono visibili nella barra degli indirizzi del browser.

- Possono essere memorizzati nella cronologia del browser e nei log del server.

POST: Viene utilizzato per inviare dati al server, in particolare quando si vogliono creare o modificare risorse, come nel caso dell'invio di un modulo di contatto o la registrazione a un sito. I dati vengono scritti nel corpo (body) della richiesta HTTP, e quindi:

- Non sono visibili nell'URL.

- Non vengono salvati nella cronologia del browser.

- È più sicuro del GET per l'invio di dati riservati (es. password, email).

- Non è possibile salvare o condividere facilmente una richiesta POST, perché non è codificata nell'URL.

Riccardo Franceschini

10. Che differenze possono esserci tra form per login e per registrazione? Quale metodo HTTP è consigliabile e perché?

La differenza principale tra un form di login e un form per la registrazione è principalmente la struttura dei form, perché in form per login sono presenti due entry, uno per la e-mail e uno per la password, nella registrazione invece, esistono molti più campi, come il Nome, il cognome, l' e-mail, la password, la conferma password, ecc.

Un'altra differenza è quella della validazione dei dati, ad esempio nel caso del login si verificano che le credenziali siano corrette; invece, nella registrazione avviene la verifica che i dati siano validi e non già usati.

Ci sono altre differenze tra login e registrazione: come le azioni eseguite sul database, perché in caso del login viene eseguita una select e una verifica della correttezza dei dati inseriti; nel caso di una registrazione invece, funziona diversamente, perché in questo contesto avviene una insert nel database dei dati dell'utente.

Pietro Gallo

11. Sviluppa brevemente il seguente spunto interdisciplinare di area tecnica: come potresti mettere in sicurezza un form di login?

Un esempio base di form di login è:

```
<form action="/login" method="post">
    <label for="username">Username:</label>
```

```
    <input type="text" id="username" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <input type="submit" value="Login">
</form>
```

Per migliorare la sicurezza:

1. Usare method=”POST” per evitare che i dati appaiano nell’URL (più sicuro di GET).

2. Utilizzare HTTPS è fondamentale per far sì che il form e i dati in transito siano protetti.

3. Campi con required per obbligare l’utente a compilare i campi (un semplice controllo lato client).

4. Protezione lato server: i dati vanno validati e sanificati anche sul server (es. password devono essere hashate e non salvate in chiaro).

Lorenzo Lissoni

12. Cos’è un layout responsive?

Il layout responsive è una tecnica di progettazione e sviluppo web che consente a un sito internet o un’applicazione di adattarsi automaticamente alla dimensione e alla risoluzione dello schermo del dispositivo da cui viene visualizzato. Che si tratti di uno smartphone, un tablet, un laptop o uno schermo ultra-wide, il contenuto del sito si ridistribuisce e si ridimensiona per offrire la migliore esperienza utente possibile.

Esempio: l’uso di griglie a larghezza fluida (in percentuali) permette agli elementi della pagina di ridimensionarsi in proporzione allo schermo. Ad esempio, invece di fissare una larghezza in pixel, si usa:

```
.container {
    width: 90%;
}
```

In questo modo, il contenitore si adatta automaticamente allo schermo, occupando il 90% della larghezza disponibile.

I vantaggi sono:

1. Accessibilità universale: il sito funziona ovunque, su ogni dispositivo.

2. Miglior usabilità: l’utente trova ciò che cerca più facilmente.

3. SEO-friendly: Google premia i siti responsive nel posizionamento.

4. Risparmio: non servono versioni separate del sito per mobile e desktop.

5. Flessibilità futura: il sito è già pronto per nuovi dispositivi con risoluzioni diverse.

Riccardo Lombardi

13. Cosa sono le media queries?

Le media query sono funzionalità CSS che definiscono il layout di pagine web in base alla dimensione di esse.

Permettono di definire al loro interno i parametri CSS da applicare in base alla condizione specificata nella media query, ovvero la risoluzione dello schermo o la larghezza del browser.

È così possibile impostare il layout della propria pagina web in modo che si adatti automaticamente alla larghezza possibile, cioè al dispositivo (smartphone, tablet, desktop).

Vanno inserite nel tag style della pagina html o direttamente nel foglio CSS.

```
@media (max-width: 768px) {
        .menu {
            display: none;
            flex-direction: column;
            position: absolute;
            top: 50px;
            right: 10px;
            background-color: rgb(10, 18, 248);
            padding: 10px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            border-radius: 5px;
        }
}
@media screen and (max-width: 992px) {
  body {
    background-color: blue;
  }
}
```

Stefano Motto

14. Quali sono i vantaggi dell'utilizzo di un framework CSS come Bootstrap rispetto alla scrittura da zero di fogli di stile?

Usare un framework CSS come Bootstrap offre numerosi vantaggi rispetto a scrivere tutto da zero. Innanzitutto, consente un grande risparmio di tempo, grazie a componenti predefiniti pronti all'uso come pulsanti, moduli e menu.

Garantisce inoltre coerenza nello stile, utile soprattutto in progetti complessi o sviluppati in gruppo, e un design responsive grazie al sistema a griglia mobile-first.

Bootstrap è anche compatibile con i principali browser e facilmente personalizzabile, permettendo di adattarlo alle esigenze del progetto.

Francesco Mutti

15. Cos'è JavaScript e quale ruolo ha in una pagina web?

JavaScript è un linguaggio di programmazione che gli sviluppatori utilizzano per realizzare pagine Web interattive.

Il codice JavaScript viene interpretato, ossia tradotto direttamente nel codice del linguaggio macchina sottostante da un motore JavaScript.

Con "JavaScript lato client" ci si riferisce al modo in cui JavaScript funziona nel browser. In questo caso, il motore JavaScript si trova all'interno del codice del browser.

Tutti i principali browser Web sono dotati di motori JavaScript integrati.

Con "JavaScript lato server" ci si riferisce all'uso del linguaggio di codice nella logica del server di back-end. In questo caso, il motore JavaScript si trova direttamente sul server. Una funzione JavaScript lato server può accedere al database, eseguire diverse operazioni logiche e rispondere a vari eventi attivati dal sistema operativo del server.

In passato, le pagine Web erano statiche, simili alle pagine di un libro. Una pagina statica mostrava principalmente informazioni in un layout fisso e non funzionava come oggi ci aspettiamo da un sito Web moderno.

JavaScript è nato come tecnologia lato browser per rendere più dinamiche le applicazioni Web.

Stefano Origgi

16. Nel modello client-server, dove e quando avviene l'esecuzione di script JS?

Nel modello client-server, gli script JavaScript vengono eseguiti sul lato client, cioè direttamente sul dispositivo dell'utente (come un computer, uno smartphone o un tablet), una volta che la pagina web è stata scaricata e interpretata dal browser.

Quando si visita un sito web, il browser riceve la pagina dal server e poi esegue il codice JavaScript direttamente sul dispositivo dell'utente, ad esempio per gestire interazioni come menu a tendina o controlli sui moduli.

Tommaso Perego

17. Spiega il paradigma ad eventi in JavaScript con degli esempi.

È un approccio che permette di definire come reagire a determinate azioni o eventi che accadono all'interno di una pagina web o in un'applicazione. Grazie a ciò quest'ultime diventano più interattive e dinamiche, modificandosi contemporaneamente alle azioni dell'utente.

Gli elementi fondamentali sono:

· Eventi: azioni che si verificano in una pagina web, come un click su un bottone, il caricamento di una pagina, la pressione di un tasto, etc.

· Gestori di eventi: sono funzioni o blocchi di codice che vengono eseguiti quando un determinato evento si verifica.

· Ascoltatori: intercettano il verificarsi di eventi e mandano in esecuzione i gestori a questi associati.

```
<div id="myDiv"> Passa sopra qui </div>


<div id="elenco">
```

```
    <div>Elemento 1</div>
    <div>Elemento 2</div>
    <div>Elemento 3</div>
    <div>Elemento 4</div>
</div>

<script>
    div=document.getElementById("div")

    div.addEventListener("mouseover", function() {
        const elenco = document.getElementById("elenco");
        for (let i = 0; i < elenco.children.length; i++) {
            elenco.children[i].style.display = "none";
        }
    });


    div.addEventListener("mouseout", function() {
        const elenco = document.getElementById("elenco");
        for (let i = 0; i < elenco.children.length; i++) {
            elenco.children[i].style.display = "list-item";
        }
    });
</script>
```

Simone Ranieri

18. Che differenza c'è tra una funzione sincrona e una asincrona in JavaScript?

In JavaScript, tutto il codice è sincrono per impostazione predefinita, salvo che tu non utilizzi esplicitamente tecniche asincrone (setTimeout, async, await). Una funzione sincrona è una funzione che viene eseguita in modo sequenziale, riga per riga. Ogni istruzione deve completarsi prima che venga eseguita quella successiva. Questo significa che blocca l'esecuzione del codice finché non ha terminato.

```
function saluta() {
  console.log("Ciao");
  console.log("Come stai?");
 }
 saluta();
 console.log("Fine");

Ciao
Come stai?
Fine
```

Funzioni sincrone si usano quando l'operazione è istantanea (es. calcoli matematici, concatenazione stringhe), non dipendi da risorse esterne (es. server, file, timer).

- Vantaggi: il flusso del codice è facile da leggere e seguire, sai esattamente cosa succede prima e dopo e puoi gestire gli errori in modo immediato.
- Svantaggi: se una funzione richiede tempo (es. calcolo pesante o attesa di dati), tutto il resto si ferma. In un browser, un'operazione sincrona lunga può bloccare l'interfaccia utente, rendendola non responsiva. Per operazioni più complesse o potenzialmente lente, è importante sapere quando passare all'asincronia per evitare problemi.

Una funzione asincrona consente invece di non bloccare l'esecuzione del codice... è una funzione che non blocca l'esecuzione del programma mentre svolge un'operazione lunga o esterna. Il programma può continuare a eseguire altre istruzioni mentre la funzione aspetta il risultato, come nel caso di: chiamate a server API, accesso a database o file, timer, caricamento di risorse. Questo è reso possibile dal modello event loop di JavaScript, che permette di delegare operazioni lunghe e riprendere l'esecuzione quando sono completate.

```
async function miaFunzione() {
 let risultato = await operazioneLunga();
 console.log("Risultato:", risultato);
 }
```

- Vantaggi: non bloccano il thread principale: l'interfaccia utente rimane reattiva.
- Svantaggi: se non usi Promise, async/await non serve. In alcune situazione, può essere difficile ragionare sull'ordine di esecuzione.

Andrea Ravasi

19. Cos'è Node.js e come differisce da JavaScript lato client?

Node.js è un ambiente di esecuzione per JavaScript, che permette di eseguire codice JavaScript al di fuori del browser, quindi lato server. Mentre il JavaScript tradizionalmente veniva eseguito solo nel browser per gestire l'interazione con l'interfaccia utente e dinamiche della pagina web, Node.js consente di utilizzare JavaScript per sviluppare applicazioni server-side, come ad esempio server web, API, strumenti a riga di comando e molto altro. Una delle principali differenze tra Node.js e JavaScript lato client è quindi il contesto di esecuzione. Il codice lato client gira nel browser e ha accesso al DOM (Document Object Model), che consente di manipolare la struttura HTML di una pagina, ma ha accessi limitati per motivi di sicurezza (non può, ad esempio, accedere al file system del computer dell'utente). Node.js invece gira sul server, non ha accesso al DOM ma può interagire con il sistema operativo, leggere e scrivere file, gestire connessioni di rete, e così via. Inoltre, Node.js è particolarmente adatto per applicazioni in tempo reale o che richiedono una gestione efficiente delle operazioni asincrone, grazie alla sua natura event-driven e non bloccante. In sintesi, Node.js amplia le potenzialità del linguaggio JavaScript permettendone l'uso anche lato server, offrendo strumenti e librerie pensati per lo sviluppo back-end, in contrasto con il JavaScript lato client che resta focalizzato sull'interazione con l'utente e il contenuto delle pagine web.

Andrea Rigamonti

20. Cos'è Express.js e cos'è un middleware?

Express.js è uno dei framework più popolari e utilizzati per quanto riguarda Node.js: questo framework offre molte funzionalità per la creazione di siti e applicazioni web.

Per la sua installazione è necessario utilizzare il gestore di pacchetti "npm" tramite la shell digitando il seguente comando:

```
npm install express
```

Una di queste funzionalità che semplifica molto l'utilizzo di Node.js è quello di poter gestire le richieste HTTP di GET, POST, DELETE, e PUT in maniera semplice, programmando direttamente gli endpoints.

I middleware sono delle funzioni che hanno accesso a req (oggetto richiesta), res (oggetto risposta) e next (per passare al middleware successivo) queste funzioni servono per gestire/modificare il flusso di richieste e risposte del server.

```
const express = require('express');
const app = express();

app.use((req, res, next) => {
  console.log('Richiesta ricevuta:', req.method, req.url);
  next();
});

app.use((req, res, next) => {
  req.utente = 'Mario';
  next();
});

app.get('/', (req, res) => {
  res.send(`Ciao, ${req.utente}!`);
});

app.listen(3000);
```

I middleware sono per esempio quelle funzioni passate a app.use(); e il primo serve per stampare a console il tipo di metodo HTTP utilizzato e l'URL e grazie a next passa al secondo middleware, che serve ad aggiungere la proprietà "utente" all'oggetto req e gli assegna il nome "Mario" e infine, sempre dopo next, viene eseguito app.get()che stamperà sulla porta 3000 la stringa "Ciao, Mario".

Daniele Rigamonti

21. Cos'è una rotta in Express e come si definisce?

In Express.js, una rotta rappresenta il modo in cui un'applicazione web risponde a una richiesta HTTP fatta a un determinato percorso URL (come /home, /utenti/1, ecc.). Ogni rotta può rispondere a diversi metodi HTTP, come:

· GET – per ottenere dati dal server

· POST – per inviare nuovi dati al server

· PUT – per aggiornare dati esistenti

· DELETE – per eliminare dati

Una rotta è quindi un "filtro" che intercetta richieste con determinate caratteristiche e risponde eseguendo del codice.

```
const express = require('express');
const app = express();
// Rotta GET per la homepage
app.get('/', (req, res) => {
    res.send('Benvenuto nella homepage!');
});
```

Quando un utente apre una pagina o un'applicazione, Express cerca tra le sue rotte quella che corrisponde a quella URL e quel metodo HTTP. Se la trova, esegue il callback associato: cioè una funzione che può leggere i dati della richiesta (req) e inviare una risposta (res).

```
    app.get('/profilo', (req, res) => {
        res.send('Questa è la pagina del profilo');
    });
```

Marco Rossetti

22. Cos'è EJS e a cosa serve?

EJS, acronimo di Embedded JavaScript, è un linguaggio di templating che permette di generare markup HTML utilizzando JavaScript. È progettato per essere facile da usare e da imparare, in quanto si basa su una sintassi familiare a chi conosce HTML, integrando JavaScript per la gestione di contenuto dinamico.

In sostanza, EJS crea una pagina web HTML utilizzando JavaScript, rendendo più semplice la creazione di contenuti dinamici senza dover scrivere codice JavaScript puro.

Con EJS si possono inserire tag nell'HTML che vengono interpretati lato server, viene prodotto un HTML che viene poi mandato al client; si riesce quindi lato server a prendere dei dati.

Come installarlo:

```
npm install express ejs --save
```

Come utilizzarlo:

```
const express = require('express');
const app = express();
app.set('view engine', 'ejs');
app.get('all', (req, res) => {
  mongo.connect(url).then((db) => {
    dbo = db.db('agenda')
```

```
    dbo.collection("events").find({}).toArray().then((result) => {
      res.render("all", { data: { events: result } })
      db.close()
    })
  })
})
```

Margherita Sironi

23. In che senso le API fanno astrazione?

L'astrazione è il processo con cui in informatica si intende il processo di nascondere le informazioni superflue mostrando solo quelle essenziali, essenzialmente nascondendo tutta la parte "complessa", le API in quest'ottica si occupano della chiamata mostrando all'utente solo le informazioni da lui richieste senza preoccuparsi delle query utilizzate per arrivare all'obbiettivo.

Matteo Spinelli

24. In che senso le API aiutano nel riutilizzo del codice?

Le API (Application Programming Interface) facilitano il riutilizzo del codice fornendo interfacce standardizzate e ben documentate che permettono di accedere a funzionalità già sviluppate. In questo modo, è possibile sfruttare una singola implementazione in contesti diversi, senza doverla riscrivere da zero, anche da parte di gruppi o sviluppatori differenti.

Come le API contribuiscono al riutilizzo del codice:

· Modularità: le API incapsulano funzionalità complesse in moduli riutilizzabili, che possono essere richiamati facilmente da diverse applicazioni o componenti.

· Standardizzazione: offrono un metodo uniforme e coerente per accedere a servizi o risorse, semplificando l'integrazione tra sistemi diversi.

· Manutenzione centralizzata: Le modifiche o gli aggiornamenti al codice interno dell'API possono essere gestiti in un unico punto, senza impattare i software che la utilizzano.

· Interoperabilità: Consentono a gruppi diversi di lavorare su componenti separati che comunicano tramite API, senza dover conoscere i dettagli dell'implementazione degli altri.

Grazie alle API, funzionalità avanzate possono essere rese disponibili attraverso interfacce semplici da usare, favorendo il riutilizzo del codice, riducendo i tempi di sviluppo, migliorando la qualità complessiva del software e rendendo più efficiente la creazione di nuove soluzioni.

Alessandro Yuan

25. Che differenza c'è tra una API e un Web Service? Fai esempi di entrambi

La risposta di Simone Balestreri non arrivò mai

26. Cosa prevede, in breve, il protocollo SOAP per lo sviluppo di Web Service?

SOAP (Simple Object Access Protocol) è un tipo di architettura usata per creare servizi web, cioè per far comunicare applicazioni diverse attraverso internet. Si basa sul protocollo HTTP, ma impacchetta le informazioni in un formato più rigido e strutturato. Questo lo rende più lento rispetto a REST, che è più leggero e veloce. SOAP però oAre tante funzionalità avanzate, come la gestione della sicurezza e delle transazioni. Tuttavia, proprio per questa sua complessità, non è adatto per applicazioni web usate direttamente da un browser. In generale, SOAP è più complicato da usare rispetto ad altre soluzioni moderne, ma rimane molto potente in contesti aziendali dove è necessaria una comunicazione aAidabile e strutturata.

Pietro Barachetti

27. Cosa prevede, in breve, l'architettura REST per lo sviluppo di Web Service?

REST, acronimo di Representational State Transfer, è uno stile architetturale per la progettazione di Web Services. Si basa sull'utilizzo del protocollo HTTP per la gestione delle richieste e delle risposte, risultando molto più semplice e leggero rispetto allo standard SOAP.

Nel modello REST, ogni risorsa è univocamente identificata da un URI (Uniform Resource Identifier). Una "risorsa" può essere qualsiasi tipo di dato, servizio o entità accessibile via web e identificabile tramite un URI.

REST si fonda su quattro principi fondamentali:

1. Uso esplicito dei metodi HTTP: ad esempio, 'GET' per leggere dati, 'POST' per crearli, 'PUT' per aggiornarli, e 'DELETE' per eliminarli.
2. Statelessness (assenza di stato lato server): ogni richiesta contiene tutte le informazioni necessarie per essere elaborata, senza fare affidamento su uno stato mantenuto dal server tra le chiamate.
3. Esposizione di URI gerarchici: le risorse sono organizzate in maniera gerarchica, simile alla struttura di una directory.
4. Trasferimento dei dati in formato JSON o XML: i dati tra client e server vengono generalmente scambiati in formato JSON (più leggero) o XML.

Nicolò Bignotti

28. Fai un esempio di richiesta HTTP diretta a un endpoint di Web Service non RESTful.

Un esempio è:

```
async function fetchComuni() {
  try {
    const response = await fetch("https://example.it/comuni?province=LC");
    comuni = await response.json();
  } catch (error) {
    console.error(error);
  }
}
```

Poiché l'uso di query string va contro i principi architetturali RESTful.

Federico Cappi

29. Cosa significa architettura orientata ai servizi (SOA)?

L'architettura orientata ai servizi, conosciuta come SOA è un modello di progettazione software in cui le funzionalità di un sistema sono suddivise in servizi indipendenti che possono essere utilizzati e combinati tra loro per costruire applicazioni più complesse. Ogni servizio rappresenta una specifica funzione o processo che può comunicare con altri servizi tramite interfacce standardizzate come API o protocolli web; questo approccio permette una maggiore flessibilità e modularità nel sistema facilitando la manutenzione l'aggiornamento e la scalabilità delle applicazioni, inoltre consente ai servizi di essere riutilizzati in contesti diversi migliorando l'efficienza dello sviluppo software e l'interoperabilità tra piattaforme diverse.

Lorenzo Cardellicchio

30. Riassumi brevemente i benefici della distribuzione.

La risposta di Leonardo Caruso dev'essersi persa nei meandri del web

31. Riassumi brevemente le sfide della distribuzione.

Poiché i sistemi distribuiti coinvolgono molte componenti che operano simultaneamente, può risultare complesso gestire correttamente tutti i computer che lavorano in parallelo.

La presenza di diversi elementi aumenta la possibilità di errori, inclusi malfunzionamenti parziali (partial failures).

I sistemi distribuiti vengono utilizzati per raggiungere determinate prestazioni, ma ottenere questi risultati può essere difficile.

Simone Caspani

32. Sviluppa il seguente spunto interdisciplinare di area tecnica: come collegheresti il cloud computing a quanto svolto durante l'anno?

Spiegazione di cosa è il cloud computing: il cloud computing è l'insieme di tecnologie che permettono l'utilizzo di risorse virtualizzate e distribuite in remoto in un'architettura Client/server.

Spiegazione, vantaggi e vari tipi di cloud computing (sistemi e reti): il cloud computing richiede all'utilizzatore di pagare solamente per le risorse virtualizzate che utilizza, abbattendo quindi i costi e la complessità dei sistemi informativi;

I principali tipi di cloud computing sono:

- Private: interno all'azienda virtualizzando le risorse, i servizi e standardizzandone la gestione

- Public: fornitore di servizi (provider) esterno all'azienda

- Community: servizi comuni a più organizzazioni ma non disponibili pubblicamente

- Hybrid: unisce i vantaggi delle altre (non l'abbiamo approfondito)

I modelli di servizi cloud sono:

- SaaS: utilizzo programmi installati su un server remoto

- DaaS: disponibilità online di dati a cui si può accedere

- HaaS: l'utente invia i dati in remoto che vengono elaborati e poi ritornati

- PaaS: piattaforma eseguita in remoto su cui runnano diversi programmi

- IaaS: risorse hardware usate in remoto

Lorenzo Colombo

33. Su quale modello di organizzazione dei dati poggiano MySQL e MariaDB? Riassumine i punti cardine

Seguono il modello relazionale, i dati sono fortemente strutturati e sono organizzati in tabelle e le relazioni sono implementate attraverso le tabelle tramite le chiavi esterne.

Proprietà dei dbms relazionali – proprietà ACID:

- Atomicità: un'operazione o va tutta a buon fine o viene rifiutata, non viene accettata l'esecuzione a metà di un'operazione.

- Consistenza: i dati sono coerenti rispetto all'ultima modifica fatta alla struttura.

- Isolamento: se ho due transazioni che provano a essere svolte contemporaneamente c'è il problema della concorrenza, vengono eseguite in serie.

- Durabilità: le modifiche devono permanere nel tempo.

Punti deboli dei dbms relazionali:

- Hanno dei limiti in contesti di applicazioni distribuite perché c'è il problema della distribuzione di dati strutturati in tabelle. Distribuire i dati diventa un problema soprattutto nei casi di big data; in questi casi può essere d'aiuto il cloud ma possono esserci dei disservizi.

- Scalano bene in verticale (se c'è un grande server) ma non in orizzontale (tra più server).

Manuela Colombo

34. Quali sono i limiti del modello relazionale nel contesto distribuito?

Christian Cutuli inviò una risposta, ma non a questa domanda

35. Quali sono i punti di forza del modello relazionale?

Il modello relazionale organizza i dati in tabelle, ciascuna corrispondente a un'entità, e collega le entità mediante chiavi.

I punti di forza sono noti e tra questi troviamo il rispetto delle proprietà ACID:

· Atomicità: le transazioni devono essere completate interamente o fallire.

· Consistenza: il database rimane in uno stato valido dopo ogni transazione.

· Isolamento: le transazioni concorrenti non interferiscono tra loro (prima si esegue una, poi l'altra).

· Durabilità: i dati confermati rimangono memorizzati anche in caso di guasti.

Alessandro Del Tredici

36. Cos'è un database NoSQL e quali modelli può seguire? Cos'è MongoDB?

Un Database NoSQL è un database che segue un modello differente da quello dei database relazionali infatti questo consente alte prestazioni, scalabilità ed è privo

di una struttura rigida, ma al contrario ne possiede una molto permissiva (non si basa su tabelle o schemi fissi). Il Modello che questo tipo di database segue è il cosiddetto "BASE":

- Basically Available: disponibilità garantita anche in caso di guasti

- Soft State: non è garantita la consistenza dei dati

- Eventual Consistency: il sistema converge a uno stato consistente (cosa vuol dire sta roba? Vuol dire che dopo un tot di tempo verrà aggiornato il database con i dati eventualmente nuovi)

MongoDB è un database non relazionale che organizza i documenti in collezioni dello stesso tipo. Questi dati sono salvati come un Json binario e ogni qualvolta si tenta di recuperare questi dati, viene usata una tabella di hash, garantendone così la sicurezza.

Gabriele Fumagalli

37. Devi operare delle azioni analoghe alle `CREATE TABLE` su MongoDB? Perché?

Mongodb non ha il comando createtable e non ne necessita perché è un database non relazionale. I database non relazionali non salvano i dati in delle tabelle essendo che sono schema less. Ciò gli consente di essere flessibili quindi permettono modifiche dinamiche della struttura dei dati, possono crescere distribuendo il carico su più server e sono ottimizzati per operazioni di lettura/scrittura veloci, perfetti per real-time analysis.

Andrea Goldonetto

38. Fai esempi di operazioni CRUD su MongoDB.

1. Operazione di Lettura (Read):

```
dbo.collection("eventi").find(
{date: req.params.anno + '-' + req.params.mese + '-' + req.params.giorno})
```

2. Operazione di Creazione (Create):

```
const myobj = { title: titolo, date: data,
description: descrizione, time: ora };
dbo.collection("eventi").insertOne(myobj)
```

3. Operazione di aggiornamento (Update):

```
const dbo = db.db("agenda");
dbo.collection("eventi").updateOne(
{ _id: eventId },
{ $set: updateData }
)
```

Matteo Isacchi

39. Come definiresti UML?

UML (Unified Model Language), é un linguaggio di modellazione visuale che aiuta a capire, descrivere e documentare le caratteristiche strutturali, funzionali e dinamiche di sistemi software complessi.

UML è un linguaggio di modellazione utile per: la progettazione del software, la verifica dello stato dei lavori di sviluppo, la verifica del funzionamento del software, la documentazione del software.

I formalismi grafici di UML seguono una notazione semplice e priva di ambiguità.

UML definisce, per diverse fasi del processo di sviluppo (analisi, progetto, codifica), tredici diagrammi suddivisi in due categorie principali: Structure diagrams, forniscono le viste Logical, Development e Physical (tutto ciò che 'e statico); Behavior diagrams, forniscono le viste Process e Use case (ovvero, la parte dinamica)

Matteo Maggioni

40. Perché e quando nasce UML?

Questo linguaggio di modellazione cominciò ad essere sviluppato a metà degli anni '90 quando i suoi sviluppatori ebbero l'idea di unificare diversi approcci alla progettazione di software object oriented. La sua versione 2.0 fu pubblicata nel 2005.

Andrea Magni

41. Quali sono le principali viste fornite da UML e in cosa consistono?

Le principali viste fornite da UML sono 3:

Vista strutturale: Mostra la struttura statica del sistema: classi, attributi, metodi e relazioni -¿ Class Diagram Vista funzionale: Descrive cosa deve fare il sistema dal punto di vista dell'utente -¿ Use-case Diagram Vista comportamentale: Rappresenta il comportamento dinamico e le interazioni nel tempo -¿ Sequence Diagram, Activity Diagram, State Diagram

Ogni vista mostra il sistema da un'angolazione diversa.

Amal Pecoraro

42. Cosa rappresenta un use-case diagram?

Uno use case diagram rappresenta le funzionalità e i casi d'uso di un sistema descrivendo gli attori e le interazioni tra attori e sistema descrivendo

- Attori
- Casi d'uso
- Sistema
- Relazioni

Mattia Redaelli

43. Qual è la differenza tra attori primari e attori secondari nei diagrammi dei casi d'uso?

Attori primari: inizializzano le azioni, prendono atto per primi. Collocati sulla sinistra

Attori secondari: rispondo alle azioni degli attori primari, prendono atto per secondi. Collocati sulla destra

Vittorio Rezzano

44. A cosa serve un class diagram?

Un class diagram serve a modellare le classi di software e le loro relazioni; quindi, lo scopo principale è quello di esplicitare la composizione delle varie classi (metodi e attributi) e le relazioni tra le classi; il class diagram sfrutta la vista strutturale che permette di visualizzare i componenti statici e le loro relazioni.

Ogni class diagram è composto da classi (rettangoli) composte da attributi e metodi, le classi sono in relazioni tra di loro con diversi tipi di relazioni.

Loris Ripamonti

45. Quali sono le affinità tra class diagram in UML e modello Entità-Relazione?

1. Concetti fondamentali simili

· Entità (ER) e Classi (UML) In ER, si modellano le entità che rappresentano "cose" o "oggetti" nel dominio. In UML, le classi rappresentano tipi di oggetti con attributi e comportamenti.

· Attributi (ER e UML) Entrambi i modelli permettono di definire attributi per descrivere le proprietà delle entità o classi.

· Relazioni (ER) e Associazioni (UML) In ER, le relazioni rappresentano collegamenti tra entità. In UML, le associazioni rappresentano collegamenti tra classi.

2. Rappresentazione grafica

· Entrambi sono rappresentati con diagrammi grafici che mostrano entità/classi come rettangoli e relazioni/associazioni come linee o collegamenti tra questi rettangoli.

3. Cardinalità e molteplicità

· Entrambi specificano la cardinalità/multiplicità delle relazioni:

o ER usa la cardinalità (1:1, 1:N, N:M) per indicare quanti elementi di una entità sono collegati a quanti elementi di un'altra entità.

o UML usa la molteplicità (0..1, 1.., 0..) nelle associazioni per esprimere lo stesso concetto.

4. Vincoli

· Entrambi possono definire vincoli sulle relazioni o sulle entità (ad esempio vincoli di unicità, chiavi primarie in ER, e vincoli di associazione o vincoli OCL in UML).

5. Concetto di ereditarietà / generalizzazione

· Nel modello ER si può rappresentare la generalizzazione/specializzazione tra entità (es. entità padre e entità figlio).

· In UML, la generalizzazione è un concetto centrale per rappresentare ereditarietà tra classi.

Differenze principali:

· UML Class Diagram include metodi (comportamenti), mentre ER si concentra solo sui dati.

· ER è più orientato alla progettazione di database, UML è più orientato alla modellazione del software a livello di oggetti.

Giacomo Rossi

46. Che tipi di relazione è possibile modellare in un class diagram?

Le relazioni modellabili tramite class diagram sono:

Associazione: Definisce una relazione semplice tra 2 classi. Le classi non sono dipendenti l'una dall'altra.

Dipendenza: Relazioni in cui una classe ha influenza sulla seconda. (es. se varia il prezzo di un prodotto anche l'importo da pagare del carrello aumenta)

Aggregazione: Definisce un'entità e le sue parti. Le 2 classi possono esistere indipendentemente

Composizione: Forma più forte di aggregazione. Le 2 classi non possono esistere da sole.

Ereditarietà: Definisce una relazione tra classe genitore (generica) e figlia (specializzata). I figli ereditano attributi e metodi del genitore.

Realizzazione: Relazione tra un'interfaccia e le classi che la implementano.

Gabriel Alessandro Salducco

47. Quando può tornare utile un sequence diagram?

Mai pervenuta la risposta di Matteo Salvi

48. Qual è la differenza tra attori e oggetti in un diagramma di sequenza?

Tra gli elementi base presenti all'interno di un Sequence Diagram ci sono: Attori e Oggetti. Gli Attori sono entità esterne che interagiscono con il sistema e sono rappresentati tramite l'utilizzo di figure umane stilizzate; Gli Oggetti sono le componenti interne del sistema, sono rappresentati con dei rettangoli.

Pietro Sesana