Alma Mater Studiorum - Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA
in
TECNOLOGIE WEB T

Sviluppo di applicazioni web con Nuxt e TypeORM

Candidato:
Valerio Iacobucci

Relatore:
Chiar.mo Prof. Ing.
Paolo Bellavista

Sessione
Anno Accademico 2023/2024

Contents

Introduzione

Questo lavoro di tesi focalizza l'attenzione sul framework per la realizzazione di applicazioni web Nuxt e sulla libreria per *Object relational mapping* TypeORM, proponendo alcune soluzioni implementative intese a migliorare la qualità, la manutenibilità e la scalabilità del codice.

Chapter 1

Linee evolutive

1.1 Pagine statiche

Il primo modello del World Wide Web era inteso a facilitare la condivisione di documenti tra scienziati e ricercatori, permettendo ai lettori di navigare nei siti attraverso collegamenti non lineari tra pagine. Il WWW consisteva in una combinazione di applicazioni, di protocolli e di linguaggi di marcatura progettati e rilasciati presso il CERN, principalmente ad opera di Tim Berners Lee, a ridosso degli anni '90:

Browser e Server Con un browser web installato nel proprio sistema informatico, un utente può visualizzare pagine web e scegliere di seguire i collegamenti ipertestuali (hyperlinks) per accedere ad altre pagine. Il server web è responsabile di fornire le pagine web richieste dai client, come i browser.

Protocollo HTTP HTTP è il protocollo di comunicazione di livello applicativo (OSI 7) che permette la trasmissione di informazioni tra client e server web. Un browser contatta un server web inviando una richiesta HTTP ad un determinato URL e il server risponde con una risposta HTTP contenente i dati richiesti. HTTP è un protocollo stateless, non mantiene informazioni sullo stato della comunicazione. Sta all'applicazione gestire eventuali sessioni o autenticazioni.

Linguaggio HTML Il linguaggio HTML (HyperText Markup Language) è un linguaggio di marcatura utilizzato per la realizzazione di pagine web. HTML definisce la struttura e il contenuto di una pagina web attraverso l'uso di tag e attributi e consente di incorporare elementi multimediali. Il browser web interpreta il codice HTML e mostra la pagina web all'utente, con una resa grafica determinata da regole contenute in fogli di stile (CSS).

1.2 Pagine dinamiche

I primi browser web erano in grado di visualizzare solo pagine statiche, il che significa che il contenuto di una determinata pagina non cambiava in base all'interazione dell'utente, ed i primi server web erano in grado di fornire solo pagine statiche, cioè pagine il cui contenuto non cambiava nel tempo.

Tuttavia, tramite hyperlinks e successivamente tramite form (cioè dei moduli da compilare con opzioni selezionabili dall'utente, introdotte nel 1993) i browser potevano effettuare richieste al server inviando parametri tramite il protocollo HTTP. Il server poteva elaborare tali richieste e restituire una *nuova pagina HTML* in base ai parametri ricevuti. Questo processo era gestito da programmi detti **CGI** (Common Gateway Interface), da eseguire sul server per generare pagine dinamicamente, cioè al momento della richiesta.

I linguaggi di programmazione utilizzati all'epoca per scrivere programmi CGI erano principalmente:

Linguaggi di basso livello o di scripting

- Linaguaggi di basso livello come C o C++, erano di difficile gestione e manutenzione: sono performanti ma richiedono una solida conoscenza informatica.
- Linguaggi di scripting come Perl o Shell UNIX, erano più facili da utilizzare ma meno efficienti: comodi per la manipolazione di stringhe e file, ma non per la gestione di strutture dati complesse.

Linguaggi di templating Successivamente, dal 1995 in poi, emersero alcuni Linguaggi di templating che consentono di incorporare codice dinamico all'interno di pagine HTML.

- Java Server Pages (JSP) è un'estensione di Java, quindi era possibile usare tutte le librerie di questo linguaggio molto popolare all'epoca.
- PHP è un linguaggio interpretato che ha avuto molto successo per oltre un decennio[1] grazie alla sua semplicità.

```
<?php
$username = $_POST["username"];
$password = $_POST["password"];

$query = sprintf( "SELECT * FROM Users WHERE username='%s' AND password='%
    s'", $username, $password);
```

[1] Indice TIOBE per PHP, si può vedere come il suo utilizzo sia diminuito a partire dal 2010.

```
$result = mysql_query($query, $conn);

if (mysql_num_rows($result) > 0){
?>
 <h1>Benvenuto <?php echo $username; ?></h1>
 ...
<?php
} else {
 ?>
 <h1>Accesso negato</h1>
 <p>Torna alla <a href="login">pagina di login</a></p>
<?php
}
?>
```

Un esempio di pagina di autenticazione in PHP, che riflette lo stile di programmazione tipico dell'epoca. È da notare come il codice HTML da inviare al browser sia inserito direttamente all'interno del codice da mantenere privato, rendendone difficile la manutenzione per via della *confusione tra logica di presentazione e logica di business*. Vengono poi adoperati 3 linguaggi diversi (PHP, HTML, SQL), soluzione non ottimale per la leggibilità, che si aggiunge ai problemi di sicurezza legati all'*interpolazione* di stringhe all'interno di query SQL.

1.3 Pagine attive con Javascript

Il dinamismo delle pagine web supportato da server CGI e linguaggi di scripting era comunque limitato per via del caricamento di nuove pagine ad ogni richiesta. Non era possibile aggiornare parzialmente la pagina, ma solo scaricarne una nuova. Nel 1995 il browser Netscape Navigator introdusse il supporto ad un nuovo linguaggio di scripting, Javascript, realizzato da Brendan Eich, per ovviare a questo problema.

Gestione di eventi e manipolazione del DOM Uno script Javascript, distribuito all'interno di una pagina HTML, può essere eseguito dal browser web in risposta a determinati eventi, come il click del mouse o la pressione di un tasto. Inizialmente il motore di esecuzione era sincrono, cioè bloccava l'esecuzione del codice fino al completamento dell'operazione, e la istruzioni per la risposta agli eventi erano inserite in funzioni dette di *callback*.

Javascript può manipolare il DOM (Document Object Model), cioè la rappresentazione ad albero della struttura della pagina, per aggiungere, rimuovere o modificare elementi HTML.

Richieste HTTP asincrone Le pagine web, erano diventate *attive*, ma tutti gli script da fornire agli utenti dovevano essere inseriti nella pagina inviata come prima risposta HTTP. Nel 1999, il browser Internet Explorer 5 introdusse una estensione del linguaggio Javascript, che disponeva di un oggetto chiamato *XMLHttpRequest*, in grado effettuare richieste HTTP asincrone al server e di ricevere risposte senza dover ricaricare l'intera pagina. Così si gettavano le basi per realizzare *Single Page Applications*.

Riveste una particolare importanza la libreria jQuery, rilasciata nel 2006, che semplifica la manipolazione del DOM e le richieste HTTP, fornendo un'interfaccia più semplice e omogenea rispetto ai diversi browser, che esponevano API diverse e non ancora standardizzate.

1.4 Node.js e Javascript lato server

La standardizzazione di Javascript seguitò attraverso le varie versioni di ECMAScript che definivano le nuove funzionalità del linguaggio e, di conseguenza, dei browser web. Nel 2008 fu rilasciata la prima versione di Google Chrome, un browser che, oltre ad includere caratteristiche appetibili per gli utenti finali, disponeva del motore di esecuzione Javascript V8. Questo *engine* apportò dei sostanziali miglioramenti di prestazioni[2] rispetto alla competizione e venne rilasciato come *Open source software*.

Nel 2009, Ryan Dahl iniziò a lavorare, basandosi sul codice di V8, a Node.js, un interprete di Javascript in modalità headless[3], al quale aggiunse la capacità di accedere al filesystem, di esporre servizi HTTP e di accettare connessioni in maniera *non bloccante*.

In questo modo si poterono realizzare non solo applicazioni **frontend** ma anche **backend** con Javascript, abilitando sempre più novizi alla creazione di siti web completi. Attorno a Node crebbe una comunità di sviluppatori che contribuirono, secondo i principi dell'Open source, alla creazione di un ecosistema di librerie, che potevano essere installate tramite il gestore di pacchetti NPM.

```
const http = require("http");
const mysql = require("mysql");

const connection = mysql.createConnection({ /* ... */ })

const server = http.createServer((req, res) => {
    if (req.method === 'POST' && req.url === '/login') {

  var username, password;
  // unmarshalling della query string ...
```

[2] Google Chrome announcement , in questo video si può vedere come l'esecuzione di Javascript su Chrome sia di circa 60 volte più veloce che su Internet Explorer 8.

[3] Cioè senza interfaccia grafica.

```
  var login = "SELECT * FROM Users WHERE username = ? AND password = ?";
  connection.query(login, [username, password], (err, rows) => {
   if (rows.length > 0) {
    res.writeHead(200, {"Content-Type": "text/html"});
    res.end("<h1>Benvenuto " + username + "</h1>");
   } else {
    res.writeHead(401, {"Content-Type": "text/html"});
    res.end("<h1>Accesso negato</h1>");
   }
  }
 }
}

server.listen(80, () => { console.log("Server in ascolto sulla porta
    8080"); });
```

In questo frammento di codice è mostrato l'utilizzo della libreria "http" fornita di default da Node e della libreria "mysql" di Felix Geisendörfer, una delle prime per l'accesso a database da Node. È da notare l'architettura a callback, che permette di gestire in maniera asincrona le richieste HTTP e le query al database.

In questo esempio le query SQL sono parametrizzate, facendo uso di *Prepared statements*, per evitare attacchi di tipo injection, ma rimangono camblate all'interno di stringhe, rendendo il codice vulnerabile a errori di sintassi e di tipo.

Nonostante, per brevità, venga inviato l'HTML in maniera diretta, è stato fin da subito possibile utilizzare le capacità di lettura asincona di files di Node per servire pagine statiche.

1.5 Applicazioni web orientate a componenti

Anche con Node fu possibile realizzare applicazioni web *monolitiche*, parimenti al templating PHP, usando librerie come EJS di TJ Holowaychuk.

Le tendenze di quel periodo (circa 2010) si discostarono dal modo tradizionale di scrivere applicazioni web, basate su pagine generate lato server, per passare a un modello di **client-side rendering**. Secondo questo modello il server invia al browser una pagina HTML con un DOM minimo, corredato di script Javascript che si occupano di popolare il DOM dei contenuti e di gestire le logiche di presentazione.

Le applicazioni renderizzate lato cliente potevano beneficiare di una maggiore reattività e di una migliore esperienza utente, essendo basate su una pagina unica che veniva

aggiornata in maniera incrementale, aggirando i caricamenti di nuove pagine da richiedere al server. Le richieste, essendo asincrone, potevano essere gestite in modo meno invasivo rispetto a prima: mentre la comunicazione client-server avveniva in background, l'utente poteva continuare ad interagire con l'applicazione.

Il vantaggio da parte degli sviluppatori di questo paradigma era la possibilità di scrivere la logica di presentazione interamente in Javascript, sfruttando il sistema di oggetti e la modularità del linguaggio in maniera più espressiva rispetto al templating o alle API DOM. Scrivere applicazioni con jQuery, ad esempio, portava ad assumere uno stile troppo imperativo e poco manutenibile per applicazioni complesse o che dovevano essere sviluppate da più persone.

L'idea centrale della tendenza *CSR* era quella di progettare l'interfaccia utente partendo da parti più piccole, chiamate **componenti**, e riutilizzabili all'interno dell'intera applicazione. Lo stile assunto era *dichiarativo*[4] Ad ogni componente erano associati:

- un template HTML, più piccolo e gestibile rispetto ad una pagina intera.
- un foglio CSS, per la stilizzazione.
- il codice Javascript che ne definisce la logica di interazione.

Tra gli esempi più noti di sistemi di componenti:

Angular.js Uno dei primi framework a proporre un modello di componenti, sviluppato in Google e rilasciato nel 2010. Angular.js introduceva il concetto di *Two-way data binding*, cioè la possibilità di sincronizzare automaticamente i dati tra il modello e la vista.

React.js La libreria di componenti più popolare[5], sviluppata da un team interno di Facebook e rilasciata nel 2013. React introduceva il concetto di *Virtual DOM*, una rappresentazione in memoria del DOM reale, che permetteva di calcolare in maniera efficiente le differenze tra due stati del DOM e di applicare solo le modifiche necessarie.

Vue.js Partito come progetto personale di Evan You e rilasciato nel 2014, Vue si proponeva come un'alternativa più flessibile e meno verbosa rispetto ad Angular e React, dai quali riprende il binding bidirezionale e il Virtual DOM. È la libreria di componenti usata da Nuxt.

[4] In questo contesto, uno stile dichiarativo è riferito ad un approccio alla programmazione in cui si descrive cosa il programma deve fare piuttosto che come farlo. Con jQuery si dovevano specificare esplicitamente i passaggi per manipolare il DOM, mentre i componenti permettono di definire il comportamento dell'interfaccia attraverso delle dichiarazioni più astratte e concise.

[5] Github - React - la più popolare in base numero di stelle su Github.

```
<script setup>
 import { RouterView } from "vue-router";
 import HelloWorld from "./components/HelloWorld.vue";
</script>

<template>
 <HelloWorld msg="Ciao mondo" />

 <RouterView />
</template>

<style scoped>
 background-color: #f0f0f0;</style
>
```

Un esempio moderno di applicazione Vue 3, che mostra l'utilizzo di un componente ”HelloWorld” all'interno di un template principale, e di un RouterView per la navigazione tra le pagine. Questi componenti sono definiti in file distinti, per favorire la separazione, ed importati nel file principale come se fossero moduli Javascript. Agli effetti lo sono, ed ogni volta che compaiono in un template assumono un comportamento dettato dalla loro definizione (il loro template) e dal loro *stato* interno. Si noti come il componente HelloWorld sia parametrizzato con un attributo ”msg”, che verrà visualizzato all'interno del componente.

```
<html lang="en">
 <head>
  <title>Vite App</title>
 </head>
 <body>
  <div id="app"></div>
  <script type="module" src="/src/main.js"></script>
 </body>
</html>
```

Il DOM minimo che viene distribuito da una applicazione Vue 3 servita con lo strumento *Vite*. La pagina viene assemblata lato client, a partire dal così detto *entry point*: l'elemento con id ”app” in cui viene montata l'applicazione. Si noti che nel caso che Javascript non sia abilitato nel client il contenuto dell'applicazione non verrà visualizzato affatto.

1.6 Typescript e ORM

Le basi di codice Javascript, che fino a quel momento erano rimaste in dimensioni ridotte, iniziarono a diventare sempre più complesse quando anche team di sviluppatori di grandi compagnie iniziarono ad adottare i framework a componenti. A partire dal 2014 anche applicazioni web come Instagram, Netflix e Airbnb incorporarono React nei loro stack tecnologici per realizzare interamente l'interfaccia utente.

Javascript, essendo un linguaggio interpretato e debolmente tipizzato, non era in grado di garantire la correttezza del codice e i test di unità erano fatti in modo *Behavior driven*, cioè basati sul comportamento dell'applicazione e non sulla tipizzazione dei dati. Questo spesse volte portava ad errori, difficili da individuare e correggere, soprattutto in applicazioni di grandi dimensioni.

Nel 2012 Anders Hejlsberg ed il suo team interno a Microsoft iniziarono a lavorare al linguaggio Typescript, una estensione di Javascript, realizzando un **compilatore** con un sistema di tipi robusto in grado di rilevare errori di questa specie. Typescript permette anche di sfruttare le funzionalità delle nuove versioni di ECMAScript in modo *retrocompatibile*, cioè facendo *transpiling*[6] verso una specifica di ECMA inferiore, per usarle anche sui browser deprecati.

L'adozione di Typescript è stata pressoché immediata, per il motivo che la conversione di basi di codice a partire da Javascript vanilla[7] erano a costo zero: ogni sorgente Javascript è valido Typescript. Typescript ha avuto successo non solo lato client, ma anche lato server. Sono comparse infatti alcune librerie di supporto all'accesso a database che basate sul pattern **ORM**, Object-relational mapping, quindi capaci di mappare il modello dei dati presente nel database a strutture dati proprie di Typescript. Librerie notevoli di questo tipo sono:

Sequelize È stata una delle prime, il progetto è iniziato nel 2010 quindi funzionava con Javascript vanilla, ma si è evoluta fino a supportare le migliorie di Typescript ed una moltitudine di DBMS.

TypeORM Offre un supporto a Typescript nativamente. È illustrata con dettaglio nel capitolo 2.

[6] Wikipedia - Source-to-source compiler - il transpiling è il processo di traduzione automatica di codice sorgente da un linguaggio ad un altro.

[7] Con "vanilla" ci riferisce a Javascript senza estensioni, quindi al codice che può eseguire nativamente sui browser conformi alle specifiche ECMA. Typescript invece è un *superset*, quindi ha un insieme di espressioni sintattiche più grande ma che comprende interamente quello di Javascript.

1.7 Ritorno al server side rendering

Dal lancio di React sempre più applicazioni web hanno fatto uso della tecnica CSR per via della migliorata esperienza utente e di sviluppo. Questo approccio ha portato però una serie di nuovi problemi e limitazioni legate al meccanismo di rendering.

Performance su dispositivi lenti I dispositivi con capacità di calcolo inferiore possono subire rallentamenti all'esecuzione di *bundle* Javascript di dimensione elevata, ed uno dei problemi delle applicazioni CSR è quello della spedizione di script spesso ridondanti all'utente. C'è uno spreco di risorse.

Search engine optimization I siti web che fanno uso di CSR sono più difficilmente indicizzabili dai *crawler* dei motori di ricerca, questo può portare a problemi di esposizione e di traffico ridotti.

First contentful paint È il tempo che intercorre tra la cattura della risposta HTTP del server e il momento in cui viene visualizzato a schermo dal browser il primo elemento di contenuto significativo per l'utente. Nelle applicazioni CSR questa durata spesso eccede quella massima suggerita da Google[8].

Cumulative shift layout Per il motivo che gli aggiornamenti dell'interfaccia vengono eseguiti nell'engine del browser in maniera sequenziale, potrebbero esserci dei fastidiosi spostamenti di elementi visivi nell'interfaccia.

Accessibility Per gli stessi motivi che portano a CSL, ci potrebbero essere degli impedimenti di accessibilità per chi usa metodi di input alternativi o per gli screen-reader che aiutano nella fruizione delle pagine web le persone non vedenti.

Attraverso i seguenti capitoli saranno illustrate alcune soluzioni ibride tra Client side rendering e Server side rendering attraverso il framework Nuxt, che mira a risolvere questi ed altri problemi, in combinazione con TypeORM per realizzare applicazioni web *fullstack*[9].

[8] Google developers - Core web vitals - Al 10 maggio 2023, la durata massima ammissibile per il FCP è di 2.5s.

[9] Si occupano sia di frontend che di backend.

Chapter 2

Descrizione delle tecnologie

2.1 Nuxt

ciao

2.2 Typeorm

Chapter 3

Soluzioni di design

Conclusioni

Bibliografia

1. CERN - A short history of the Web - un articolo che percorre velocemente la storia delle origini di WWW.
2. ECMA - 262 - lo standard ufficiale di Javascript.
3. Honeypot - Node.js: The Documentary - un documentario sulla storia di Node, di NPM, e dei loro ideatori.
4. Honeypot - How a small team of developers created React at Facebook - un documentario sulla storia di React, i primi progetti che lo hanno adottato e la comunità di sviluppatori Open source che lo supporta.
5. Honeypot - Vue.js: The Documentary - un documentario sulla libreria Vue e sulle esigenze che hanno portato alla sua creazione.
6. OfferZen - Typescript Origins: The Documentary - un documentario sulla storia di Typescript, del suo team di sviluppo e selle politiche Open source di Microsoft.
7. Google developers - Core web vitals - una guida di Google sulle metriche di performance web.
8. Nuxt official documentation - la documentazione ufficiale del framework Nuxt.
9. TypeORM official documentation - la documentazione ufficiale della libreria TypeORM.
10. Fireship - il canale YouTube di Jeff Delaney, omnicomprensivo per lo sviluppo web, sul quale sono pubblicati tutorial, introduzioni a nuove tecnologie e notizie di attualità sul settore informatico.