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

Indice

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.

Capitolo 1

Linee evolutive

Il Web è la piattaforma software più estensiva al mondo, e la sua evoluzione è stata guidata da una serie di innovazioni tecnologiche che hanno permesso di realizzare applicazioni sempre più complesse e performanti. Questo capitolo ripercorre brevemente le linee evolutive del web, partendo dalle pagine statiche fino ad arrivare alle applicazioni web moderne.

1.1 Pagine statiche

Il primo modello del World Wide Web era orientato a facilitare la condivisione di documenti, permettendo ai lettori di esplorarli attraverso collegamenti tra le 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 e Robert Calliau, 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 per accedere ad altre pagine. Il server web è responsabile di fornire le pagine web richieste dai client, come i browser.

Protocollo 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, quindi sta all'applicazione gestire eventuali sessioni o autenticazioni.

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.

Nel 1993, il primo browser web grafico, Mosaic, introdusse il supporto per le immagini, per i form (dei moduli compilabili dall'utente con opzioni) e per i collegamenti ipertestuali, e successivamente Netscape Navigator 1.0 introdusse il supporto per CSS, il linguaggio che permise da subito agli sviluppatori di personalizzare la resa grafica della loro pagina. Queste innovazioni contribuirono a rendere il web più accessibile e visivamente attraente per un pubblico maggiore.

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[1].

Le prime realizzazioni di pagine dinamiche furono rese possibili grazie agli hyperlinks ed ai form, con i quali i browser potevano effettuare richieste al server inviando i parametri forniti dall'utente. 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

- Linguaggi di basso livello come C o C++, sono performanti ma di difficile progettazione e manutenzione.
- 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 dal lato server.

- 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[2] grazie alla sua semplicità.

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

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

if (mysql_num_rows($result) > 0){
?>
    <h1>Benvenuto <?php echo $username; ?></h1>
    ...
```

[1] Gli unici effetti che si potevano apprezzare immediatamente dopo un'interazione dell'utente erano quelli di CSS, ad esempio il cambio di colore di un link al passaggio del mouse.

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

```
13 <?php
14     } else {
15     ?>
16         <h1>Accesso negato</h1>
17         <p>Torna alla <a href="login">pagina di login</a></p>
18     <?php
19     }
20 ?>
```

Un esempio di pagina di autenticazione in PHP, che riflette lo stile di programmazione tipico dell'epoca[3]. È da notare come il codice HTML da inviare al browser sia inserito direttamente all'interno del codice da mantenere privato nel lato server, 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 Netscape Navigator 2.0 introdusse il supporto ad un nuovo linguaggio di scripting, che successivamente venne chiamato 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 dell'utente. Inizialmente il motore di esecuzione era sincrono, cioè bloccava l'esecuzione del codice fino al completamento dell'operazione, e le possibilità di Javascript si limitavano alla manipolazione a *runtime*[4] del DOM (Document Object Model), quindi ad aggiungere, rimuovere o modificare elementi HTML.

Richieste HTTP asincrone: Le pagine web, erano diventate *attive*, ma tutte le risorse da fornire agli utenti dovevano essere inserite nella pagina inviata come prima risposta HTTP. Nel 1999 però, 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 dunque ricevere risposte senza dover ricaricare l'intera pagina. Così si gettavano le basi per la realizzazione di *Single Page Applications*.

La libreria jQuery, rilasciata nel 2006, ha rivestito una particolare importanza perché semplificava 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.

[3] La libreria mysql di Micheal Widenius per PHP 2 risale al 1996.
[4] Il momento in cui la pagina è resa attiva con Javascript.

```
$('#update').click(function() {
    $.ajax({ // scaricamento asincrono
        url: 'data.json', type: 'GET', dataType: 'json',
        success: function(data) {
            var content = '';
            for (var i = 0; i < data.length; i++) {
                content += '<p>' + data[i].name + '</p>';
            }
            $('#content').html(content);
        },
    });
});
```

Nel frammento di codice jQuery, è messo in evidenza uno stile *imperativo* di definizione del comportamento dell'interfaccia, poco manutenibile per applicazioni complesse.

1.4 Node.js e Javascript lato server

La standardizzazione di Javascript procedette 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[5] 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[6], 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 ...
```

[5] 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.

[6] Cioè senza interfaccia grafica.

```
10          var login = "SELECT * FROM Users WHERE username = ? AND password = ?";
11          connection.query(login, [username, password], (err, rows) => {
12              if (rows.length > 0) {
13                  res.writeHead(200, {"Content-Type": "text/html"});
14                  res.end("<h1>Benvenuto " + username + "</h1>");
15              } else {
16                  res.writeHead(401, {"Content-Type": "text/html"});
17                  res.end("<h1>Accesso negato</h1>");
18              }
19          });
20      }
21  });
22  server.listen(80, () => { console.log("Server in ascolto alla 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 cablate all'interno di stringhe, rendendo il codice vulnerabile a errori di sintassi e di tipo.

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.

Tuttavia 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 JS che si occupano di popolare a runtime il DOM con 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 al server, essendo asincrone, potevano essere gestite in modo meno invasivo rispetto a prima: mentre la comunicazione avveniva in background, l'utente poteva continuare ad interagire con l'applicazione.

Il vantaggio di questo paradigma da parte degli sviluppatori 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 all'uso imperativo delle API DOM.

L'idea centrale delle nuove tendenze *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*[7]. Ad ogni componente sono 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 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. Per questi miglioramenti nella performace venne adottato moltissimo[8]. Da React in poi, lo sviluppo di pagine web ha riguardato un livello più astratto rispetto all'esecuzione del classico codice Javascript che manipola direttamente il DOM. Inteso così, il browser diventa l'interprete di un codice intermedio sul quale non si mette mano direttamente.

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.

```
<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>
```

[7] 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.

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

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 delle preoccupazioni, ed importati nel file principale come se fossero moduli Javascript. Ogni volta che compaiono in un template assumono un comportamento dettato dalla loro definizione (il loro template) e dal loro *stato* di istanza. Si noti come il componente HelloWorld sia parametrizzato con un attributo `msg`, che ne consente un utilizzo più flessibile.

```
<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. 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 iniziarono a diventare sempre più complesse quando anche i 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** in grado di rilevare errori di tipo in con analisi statica. Typescript permette anche di sfruttare le funzionalità delle nuove versioni di ECMAScript in modo *retrocompatibile*, cioè facendo *transpiling*[9] verso una specifica di ECMA inferiore, per usarle anche sui browser deprecati.

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

L'adozione di Typescript è stata pressoché immediata, per il motivo che la conversione di basi di codice a partire da Javascript vanilla[10] era 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 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.

L'evoluzione dei sistemi per fare query a basi di dati da Javascript è poi diramata in direzioni diverse, da quelli che usano protocolli applicativi binari (basati ad esempio su gRPC) a quelli che usano linguaggi di query specifici (come GraphQL), ad ORM che introducono nuovi linguaggi di definizione dei modelli (come Prisma).

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 *bundle* Javascript che vengono generati per le applicazioni CSR sono spesso onerosi in termini di risorse, e la loro esecuzione su dispositivi con capacità di calcolo limitate può risultare lenta e insoddisfacente per l'utente.

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[11].

Cumulative shift layout Per il motivo che gli aggiornamenti dell'interfaccia vengono vengono resi graficamente in maniera sequenziale nel browser, potrebbero esserci dei fastidiosi spostamenti di elementi visivi nell'interfaccia durante la fase di caricamento.

[10] 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.

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

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

Per questi motivi, a partire dal 2016, sono emerse delle nuove tendenze che hanno portato ad un ritorno al server side rendering, in combinazione con i sistemi basati su componenti, per unire i vantaggi di entrambi i modelli. Esempi di framework che supportano il SSR sono: Angular Universal, Next.js per React e Nuxt per Vue, che verrà illustrato nel capitolo 2, per la realizzazione di applicazioni web *fullstack*[12].

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

Capitolo 2

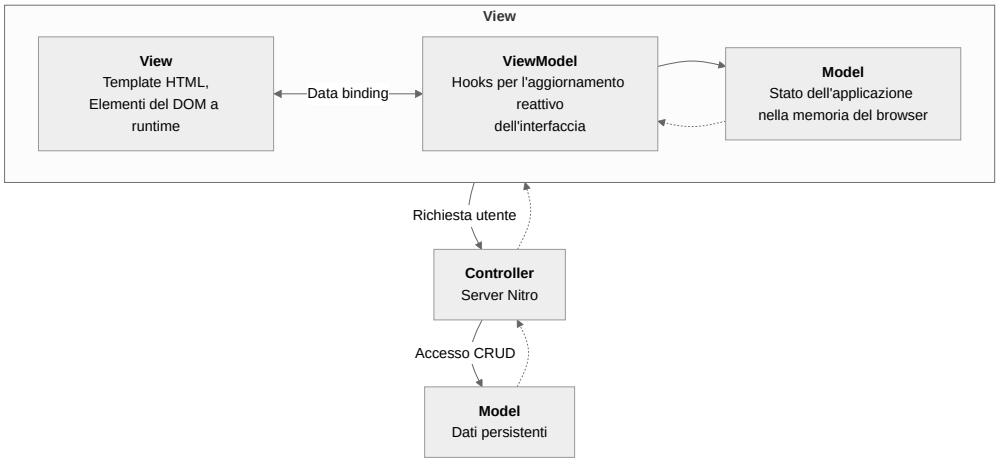
Descrizione delle tecnologie

In questo capitolo si illustrano due particolari tecnologie: Nuxt e Typeorm. Sono state scelte tra le molte alternative disponibili per il loro uso diffuso e consolidato nel settore dello sviluppo web perché esemplificano una naturale continuazione delle linee evolutive descritte nel capitolo precedente fornendo una soluzione alle problematiche affrontate, e per altre ragioni che saranno discusse in seguito.

2.1 Nuxt

Nuxt è un framework per la realizzazione di applicazioni web, avviato come progetto Open source da Alexandre Chopin e Pooya Parsa nel 2016, che continua ad essere mantenuto attivamente su Github da un team di sviluppatori che accettano contributi, all'indirizzo github.com/nuxt/nuxt.

Nuxt si propone di risolvere i problemi di performance, di ottimizzazione e di accessibilità che sono stati mostrati nel capitolo 1 con il suo sistema di frontend, ma anche di fornire un ambiente di sviluppo flessibile, per facilitare la scalabilità e la manutenibilità del codice backend. Si possono infatti realizzare applicazioni **fullstack** secondo il pattern MVC, in cui la view è implementata con Vue ed il controller con *Nitro*, un server http fatto su misura per Nuxt.



L'architettura generale di una applicazione Nuxt. Si noti che il frontend Vue adotta il pattern *MVVM* quindi si hanno due modelli con interfacce potenzialmente distinte. Infatti nel modo tradizionale di usare Vue, backend e frontend potrebbero essere viste come due applicazioni a bassa coesione (basti pensare a come potrebbero essere realizzate in due linguaggi di programmaizone differenti) ed alto accoppiamento (nel senso che un cambiamento da un lato potrebbe richiederebbe un altro cambiamento dall'altro lato del sistema, per mantenere la coerenza). Nuxt si occupa appunto di gestire la comunicazione tra i due models: il model dei dati persistenti ed il model dell'applicazione che esegue nel browser, in modo da ottenere *loose coupling* e *high cohesion*.

Lo slogan di Nuxt è "The Intuitive Vue Framework", che è in accordo con il suo obiettivo di semplificare la creazione di applicazioni web fornendo un'infrastruttura preconfigurata e pronta all'uso. In questo modo lo sviluppatore può concentrarsi da subito sulla logica dell'applicazione, piuttosto che sulla configurazione del progetto. È quindi ricalcato il punto di vista di David Heinemeier Hansson su Rails, il framework per applicazioni web per Ruby che ideò nel luglio 2004, per il quale sosteneva il principio "convention over configuration"[1].

2.1.1 Convenzioni di progetto

Già dalla creazione di un nuovo progetto Nuxt, si vede come siano proposte alcune *sensible defaults*[2], pur lasciando la possibilità di personalizzare il progetto in base alle esigenze specifiche. È consigliato avviare un nuovo progetto con la *command line interface* di Nuxt, che guida lo sviluppatore nella scelta delle opzioni di configurazione.

Command line interface

L'ecosistema Nuxt fa uso di un programma invocabile da linea di comando chiamato *nuxi*. È installabile globalmente su un sistema operativo con Node eseguendo `npm i -g @nuxt/cli`, e dispone di vari sotto-comandi per la gestione del progetto.

`nuxi init <nome-progetto>` È il comando per avviare un nuovo progetto nella directory `./<nome-progetto>`. Eseguendolo si dovrà scegliere il sistema di gestione dei pacchetti, che riguarderà il modo con il quale Nuxt ed anche gli agli altri pacchetti di terze parti saranno installati, e può essere tra:

- **Npm**: Il classico package manager di Node, solitamente installato assieme ad esso scegliendo il pacchetto `node` nelle repository delle maggiori distribuzioni Linux, e disponibile di default nelle immagini Docker ufficiali di Node. È intesa come la sensible default.
- **Pnpm**: Un package manager alternativo a npm, progettato per migliorare le performance e ottimizzare l'utilizzo dello spazio su disco rispetto a npm, preferito per lo sviluppo locale.
- **Yarn**: Un altro package manager alternativo a npm, sviluppato in Facebook nel 2016.
- **Bun**: Con questa opzione si sceglie di usare una runtime diversa da Node: Bun, più efficiente in alcune operazioni di I/O, compatibile con le API Node e i suoi pacchetti di terze parti.

[1] Wikipedia - Convention over configuration

[2] Cioè delle impostazioni scelte in base all'uso che è stato rilevato come il più comune, in base alle discussioni degli sviluppatori nei forum, si veda [^convention-over-configuration].

- **Deno**: Un'altra runtime JavaScript che offre supporto nativo a Typescript, ma non è del tutto compatibile con alcuni pacchetti npm.

Subito dopo c'è la scelta **Initialize git repository**, che eseguirà `git init` se selezionata.

`nuxi add` È il comando per aggiungere un nuovo **template** al progetto, cioè un insieme di file e di configurazioni predefinite che possono essere usate per creare una nuova funzionalità.

- **page**:
- **component**:
- **layout**:
- **middleware**:
- **api**:
- **composable**:
- **plugin**:

`nuxi dev`

`nuxi devtools` Abilita o disabilita l'iniezione degli script Devtools, cioè un set di strumenti il debugging di applicazioni Nuxt, aggiuntivi a quelli già presenti nei browser moderni[3].

`nuxi test` Esegue i test di unità e di integrazione

`nuxi build`

`nuxi module`

Directories

```
assets/                 # Risorse statiche come immagini, media e font
components/
composables/
layouts/
middleware/
node_modules/           # Dipendenze del progetto
pages/                  # Pagine del sito
    # PARLARE DI COME page[nome_parametro] rende disponibile
    ↪ $route.params.nome_parametro
plugins/
public/
server/
app.vue                 # Il componente principale montato sulla radice
```

[3]Come quelli di Firefox, dei derivati di Chromium, di Safari ed di Edge.

```
nuxt.config.ts
package.json
tsconfig.json
```

tooling e Typescript "out of the box"

tsconfig.json
: vite in alternativa a webpack e a configurazione manuale
: tutto questo manualmente!

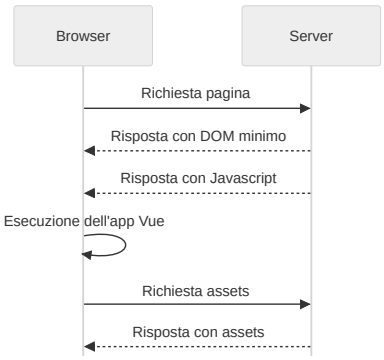
Configurazione

.nuxt.config.ts

2.1.2 Modalità di rendering del frontend

Durante la fase di progettazione, diversi tipi di applicazione suggeriscono diverse esigenze, e Nuxt si dimostra versatile a partire dalle modalità di rendering che offre.

In questo contesto, con rendering di una pagina web non si intende il processo di disegno dei pixel sullo schermo, del quale generalmente si occuperà il browser web delegando al sistema operativo la gestione dell'hardware. Qui con rendering si intende il processo di generazione del codice HTML, CSS e Javascript che costituisce la pagina web, e che viene inviato al client per essere visualizzato.



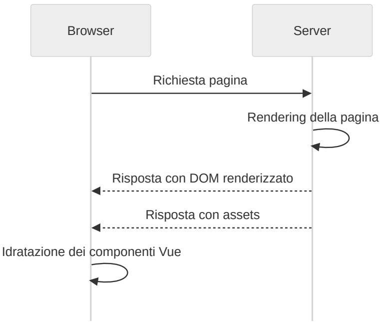
Client Side Rendering

Nuxt supporta la stessa modalità di rendering discussa nel capitolo 1, in cui il codice dell'applicazione Vue viene eseguito interamente sul browser.

Si può attivare globalmente nel file `nuxt.config.ts` con:

```
export default defineNuxtConfig({
  ssr: false
})
```

Il beneficio che si ottiene nello sviluppare in maniera CSR con Nuxt è la disponibilità del classico oggetto `window` negli script Vue, oltre ad una riduzione del costo infrastrutturale, perché il server non deve eseguire il codice Javascript per generare la pagina. Tuttavia rimangono i problemi di performance, di accessibilità e di SEO che sono stati discussi precedentemente.



Universal rendering

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec pur

Static Site Generation

md -> html

2.1.3 Server Nitro

Routes tipizzate

COME FANNO AD ESSERCI DELLE ROUTES TIPIZZATE?? controllare

Modalità di sviluppo

Build per la produzione

2.1.4 Repository e contributi

Nel particolare la repository è strutturata secondo il modello di *monorepo*, quindi include pacchetti funzionanti in maniera disaccoppiata, ma che sono usati tutti in maniera coesa all'interno del sistema Nuxt.

- `packages/nuxt` è il core del framework.
- `packages/nuxi` è lo strumento da linea di comando per la creazione di nuovi progetti, ora spostato su `github.com/nuxt/cli`.
- `packages/schema` contiene le definizioni dei tipi di dati utilizzati.
- `packages/kit` è un toolkit per la creazione di moduli aggiuntivi.
- `packages/test-utils` contiene degli script per il testing di unità.

- `packages/vite` è una fork di Vite, un bundler per gli script di frontend, usato di default da Nuxt.
- `packages/webpack` è una fork di Webpack, un'altro bundler per gli script di frontend che si può scegliere in alternativa a Vite.
- `docs` è la documentazione ufficiale, scritta sotto forma di sito web staico, usando Nuxt stesso.

I contributi sono proposti su Github e l'iter consigliato varia in base al tipo di modifica:

- Per proporre un **Bugfix** si apre un *issue*[4] per discutere il problema, e poi si apre una *pull request* che risolva l'issue.
- Per proporre una **Nuova funzionalità** si apre una *discussion*, e poi di aprire una *pull request* che implementi la funzionalità.

I contributi poi vengono sottoposti a test automatici prima di essere passati ad una revisione da parte del team di sviluppo, in modo da conformare lo stile del codice, della documentazione ed anche del messaggio di commit. Le etichette fornite nelle *PR* più comunemente sono: `enhancement`, `nice-to-have`, `bug`, `discussion`, `documentation`, `performance` e `refactor`.

Al Novembre 2024, sono stati aperti circa 15'000 issues, sono stati avanzati circa 7'000 commi da più di 700 contributori. I progetti Open source su Github che usano Nuxt sono circa 350'000 e questi numeri sono in costante crescita.

Moduli

Oltre a modificare la monorepo, gli sviluppatori Open source sono invitati a creare **moduli** per estendere le Nuxt con funzionalità non essenziali, ma idonee per l'interoperabilità con altri software. Questi moduli possono essere pubblicati su Npm come pacchetti, con `@nuxt/kit` come dipendenza, ed al Dicembre 2024 se ne contano più di 200[5].

Nel capitolo 3 si illustrerà un modulo che permette di usare Nuxt in combinazione con Typeorm.

2.2 Typeorm

DESIGN PATTERNS COME FA A FUNZIONARE?

2.2.1 Dbms supportati

2.2.2 Rappresentazione di entità e relazioni in Typescript

[4] Si tratta di un thread aggiunto alla sezione "Issues", che funziona come un forum specifico per ogni progetto, accessibile a tutti gli utenti registrati di Github.

[5] Moduli supportati ufficialmente da Nuxt

Capitolo 3

Soluzioni di design

In questo capitolo si illustrano alcune soluzioni di design per la realizzazione di applicazioni web con Nuxt in combinazione con TypeORM.

3.1 Architettura

3.2 Sicurezza

3.3 Scalabilità

Capitolo 4

Bibliografia

4.1 Testi di riferimento

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. Mozilla developer network web docs - la documentazione e le linee guida ufficiali di Mozilla per lo sviluppo web.
4. A brief history of CSS until 2016 - un articolo che celebra i 20 anni di CSS, spiegando la sua storia e le sue evoluzioni.
5. Honeypot - Node.js: The Documentary - un documentario sulla storia di Node, di NPM, e dei loro ideatori.
6. 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.
7. Honeypot - Vue.js: The Documentary - un documentario sulla libreria Vue e sulle esigenze che hanno portato alla sua creazione.
8. OfferZen - Typescript Origins: The Documentary - un documentario sulla storia di Typescript, del suo team di sviluppo e selle politiche Open source di Microsoft.
9. Google developers - Core web vitals - una guida di Google sulle metriche di performance web.
10. Nuxt official documentation - la documentazione ufficiale del framework Nuxt.
11. TypeORM official documentation - la documentazione ufficiale della libreria TypeORM.
12. 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.

4.2 Ringraziamenti

Ringrazio i miei genitori, la mia famiglia ed i miei amici per l'incoraggiamento e il supporto che mi hanno dato durante la stesura di questo lavoro.

4.3 Strumenti Open source

- **Code** - la build open source di Visual Studio Code, utilizzata per la scrittura e la correzione del testo.
- **Markdown** - il formato di scrittura utilizzato per la stesura del testo.
- **Latex** - per la formattazione del testo.
- **Pandoc** - per la conversione dei file Markdown in Latex.
- **Minted** - per la colorazione della sintassi dei codici sorgente.
- **Panflute** - per la creazione di filtri personalizzati per Pandoc.
- **Mermaid** - per la creazione di diagrammi di sequenza e di flusso.