Tesi

Marco Moroni

Luglio 2021

Abstract

Questa tesi è volta a presentare un'analisi approfondita dell' ambiente di sviluppo utilizzato e di tutti i servizi coinvolti per esternalizzare il lavoro e la sicurezza. Durante il tirocinio a Studio Indaco il mio ruolo è stato di backend developer di siti web utilizzando le tecnologie di sviluppo più adeguate e all'avanguardia. Oltre alla programmazione di singole web app, uno dei miei compiti principali è stato il contribuire e migliorare la crescita di CUBO, un sistema proprietario di gestione dei contenuti (CMS – Content Management System), con lo scopo futuro di renderlo open source. Un CMS è la scelta migliore per sviluppare un sito adatto alle esigenze del cliente, che poi gestirà automaticamente i contenuti attraverso un pannello grafico che semplifica l'integrazione tra utente e codice.

Indice

1 Introduzione

In questo documento di tesi si descrivono le attività svolte durante le trecentosessanta ore di tirocinio svolto presso Studio Indaco, un'azienda di Mantova specializzata in pianificazione strateica per il web, **realizzazione di siti internet**, campagne web e fotografia.

1.1 Descrizione dell'azienda

La struttura, composta da uno staff di 10 esperti suddivisi nelle varie aree aziendali, consente di organizzare le attività progettuali, curandone ogni singolo aspetto, dalla progettazione alla realizzazione finale. L'azienda si avvale di diversi servizi esterni di supporto: dall'hosting dei siti web, alle email, ai microservizi di supporto ai programmatori, che si stanno man mano diffondendo sempre di più nell'ambito dello sviluppo. Studio indaco è nata nel 2018 partendo da 2 soci fino ad arrivare a giugno 2021 ad avere 10 dipendenti.

1.2 Obiettivi ed esperienza aziendale

Lo scopo del tirocinio concordato con il tutor aziendale Marco Zolin, fondatore di Studio Indaco, è stato principalmente lo sviluppo del CMS proprietario dal nome CUBO, la base di ogni sito web creato dall'agenzia, nel quale ho investito la maggior parte del tempo del tirocinio. Indispensabile per svolgere tale compito, è stata la familiarità con la programmazione ed i linguaggi specifici appresi nei corsi di "Fondamenti di Informatica 1 - Python" e "Algoritmi e linguaggi di programmazione". Questi mi hanno aiutato nell'implementazione e la realizzazione di specifici software per il web.

Nell'arco delle trecentosessanta ore svolte, ho avuto una stretta collaborazione con il web designer, e co-fondatore dell'azienda, Marco Di Padova, figura addetta alla selezione e cura del layout grafico di ogni progetto, adattandolo perfettamente alle esigenze specifiche e particolari di ogni cliente. Durante il mio percorso non sono mancate relazioni fondamentali con il reparto commerciale, nel nome di Enrico che con diplomazia riescono a comunicare coi clienti riportando allo studio le loro le eventuali modifiche da apportare al progetto.

2 Aspetti teorici

In questo capitolo mi voglio focalizzare sugli aspetti teorici e strumenti necessari per poter realizzare l'obiettivo del tirocinio. Poiché i compiti principali a me assegnati riguardano la creazione e la gestione di siti internet, di seguito ho descritto come un sito web deve essere organizzato affinché sia accessibile e come l'usabilità del web sia fondamentale per venire incontro ai bisogni degli utenti finali.

2.1 Linguaggi utilizzati

In questa sezione si esaminano i linguaggi utilizzati durante lo sviluppo, i costrutti di programmazione e gli ausili che sono stati utilizzati.

Durante il periodo universitario ho appreso e approfondito il linguaggio Python, soprattutto nel corso di 'Informatica 1' tenuto dal Professor Marco Mamei. Hanno inoltre contribuito al mio miglioramento le ripetizioni private impartite agli studenti delle scuole superiori, che mi hanno dato modo di mettere alla prova le conoscenze apprese, consolidarle, ed approfondire nei dettagli le caratteristiche del linguaggio. Inoltre, sviluppando software con questo linguaggio in azienda, son riuscito ad imparare nuove tecniche di ottimizzazione del codice e di scrittura ordinata.

Nel tempo libero ho intrapreso attività progettuali, da piccoli esperimenti personali fini a se stessi tramite l'ausilio di raspberry pi ed arduino, alla realizzazione e vendita di applicazioni web complesse. Questi lavori mi hanno dato l'occasione di fare esperienza con nuovi strumenti e tecniche di sviluppo con una base comune: Javascript. Javascript è un linguaggio di scripting lato client utilizzato comunemente per rendere interattive le pagine web; è un linguaggio di programmazione interpretato, ciò significa che quando si esegue lo script, viene letta, tradotta ed eseguita un'istruzione alla volta; a differenza dei linguaggi compilati (come c/c++, go, java) che prima di essere eseguiti vengoo appunto compliati, cioè tradotti in linguaggio macchina creando un file eseguibile. I linguaggi compilati vantano di performance superiori rispetto ai linguaggi interpretati, che hanno però solitamente una portabilità più alta e soprattutto l'immediatezza tra quello che viene scritto e l'esecuzione. Javascript, considerato all'inizio, un "linguaggio di serie B" perché non gode di una buona gestione dei tipi di dato, del casting (conversione), della gestione degli oggetti, ed altri svariati motivi, negli ultimi anni si è evoluto fino a diventare un linguaggio universale. La sua versatilità favorisce lo sviluppo sia front-end che back-end, oltre che essere utile per i test. Di conseguenza, si possono trovare molte librerie e framework JavaScript per ogni scopo.

Un framework (struttura), ha lo scopo di fornire strumenti per facilitare e velocizzare il lavoro di programmazione di chi lo adotta. Con front-end, nella programmazione, si intende la parte di codice visibile agli utenti finali, mentre con back-end si definisce la porzione di script nascosto che non è direttamente accessibile. In un'applicazione web, è molto consigliata la coesistenza di entrambi, dato che spesso ci si ritrova a lavorare con dati privati, che possono essere credenziali di database, chiavi api per pagamenti, o semplici configurazioni che non vogliono essere di pubblico dominio.

Tutti i framework lato client basati su Javascript, come React, Vue, o Angular, sono quindi categorizzati come front-end, mentre i framework lato server, sempre basati su Javascript, come Node o Deno, sono categorizzati come back-end. Per fare comunicare i due punti si utilizzano le API REST (o RESTful). Un'API REST è un'interfaccia di programmazione delle applicazioni conforme ai vincoli dello stile di architettura REST, che consente inoltre l'interazione con servizi web RESTful.

2.2 API

Le API (Application Programming Interface, ovvero Interfaccia di programmazione delle applicazioni) sono set di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi. Possono essere considerate come un contratto tra un fornitore di informazioni e l'utente destinatario di tali dati: l'API stabilisce il contenuto richiesto dal consumatore (la chiamata) e il contenuto richiesto dal produttore (la risposta). Ne è un esempio la struttura di un'API di un servizio meteorologico, in cui l'utente invia una richiesta contenente un codice postale al quale il produttore fornisce una risposta in due parti, dove la prima indica la temperatura massima e la seconda la minima.

Se, in altre parole, si desidera interagire con un computer o un sistema per recuperare informazioni o eseguire una funzione, un'API facilita la comunicazione con il sistema che può così comprendere e soddisfare la richiesta.

L'API funge quindi da elemento di intermediazione tra gli utenti o i clienti e le risorse o servizi web che questi intendono ottenere. È anche un mezzo con il quale un'organizzazione può condividere risorse e informazioni assicurando al contempo sicurezza, controllo e autenticazione, poiché stabilisce i criteri di accesso alle risorse.

L'utilizzo dell'API inoltre non impone all'utente di conoscere le specifiche con cui le risorse vengono recuperate o la loro provenienza.

REST (Representational State Transfer) è un insieme di vincoli architetturali, non un protocollo né uno standard. Questo approccio architetturale

venne ideato per creare web API basandosi sul protocollo HTTP. Il REST è infatti un sistema di trasmissione dei dati che utilizza HTTP e che fa un grande uso delle funzioni di HTTP, ricorrendo ai comandi GET, POST, PUT, PATCH, DELETE e OPTIONS: grazie a questi si riesce ad identificare ben precise risorse presenti in tutto il vasto web. Chi sviluppa API può implementare i principi REST in diversi modi.

Quando una richiesta client viene inviata tramite un'API RESTful, questa trasferisce al richiedente o all'endpoint uno stato rappresentativo della risorsa. L'informazione, viene consegnata, solitamente in formato JSON (Javascript Object Notation). Il formato JSON è uno formati per lo camvio di dati tra client e sever più diffusi, perché è indipendente dal linguaggio e facilmente leggibile da persone e macchine, anche se basato su Javascript.

È importante tener presente anche altri aspetti, ad esempio intestazioni e parametri sono importanti nei metodi HTTP di una richiesta HTTP API RESTful API, perché forniscono informazioni per l'identificazione della richiesta quali metadata, autorizzazione, URI, caching, cookie e altro. Esistono intestazioni della richiesta e intestazioni della risposta, ognuna con le proprie informazioni di connessione HTTP e codici di stato.

Affinché un'API sia considerata RESTful, deve rispettare i seguenti criteri:

- un'architettura client-server composta da client, server e risorse, con richieste gestite tramite HTTP;
- una comunicazione client-server stateless, che quindi non prevede la memorizzazione delle informazioni del client tra le richieste GET; ogni richiesta è distinta e non connessa. A differenza del paradigma stateful che mantiene lo stato della connessione;
- dati memorizzabili nella cache che ottimizzano le interazioni client-server;
- un'interfaccia uniforme per i componenti, in modo che le informazioni vengano trasferite in una forma standard, ciò impone che:
  - le risorse richieste siano identificabili e separate dalle rappresentazioni inviate al client;
  - le risorse possano essere manipolate dal client tramite la rappresentazione che ricevono poiché questa contiene le informazioni sufficienti alla manipolazione;
  - i messaggi autodescrittivi restituiti a un client contengano le informazioni necessarie per descrivere come il client deve elaborare l'informazione;

  - le informazioni siano ipermediali e ipertestuali, ovvero accedendo alla risorsa il client deve poter individuare, attraverso hyperlink, tutte le altre azioni disponibili al momento.

- un sistema su più livelli che organizza ogni tipo di server (ad esempio quelli responsabili della sicurezza, del bilanciamento del carico, ecc.) che si occupa di recuperare le informazioni richieste in gerarchie, invisibile al client;

- codice on demand (facoltativo): la capacità di inviare codice eseguibile dal server al client quando richiesto, estendendo la funzionalità del client;

Sebbene l'API REST debba essere conforme a tutti questi criteri, il suo impiego è comunque considerato più semplice rispetto a quello di un protocollo prescrittivo come SOAP (Simple Object Access Protocol), che presenta requisiti specifici come la messaggistica XML e la conformità integrata di sicurezza e transazioni, che lo rendono più lento e pesante. Al contrario, REST è un insieme di linee guida applicabili quando necessario, il che rende le API REST più rapide, leggere e con una maggiore scalabilità, ottimali quindi per l'Internet of Things (IoT) e lo sviluppo di app mobili.

2.3 Strumenti utilizzati

Gli strumenti utilizzati durante il tirocinio sono stati predisposti dal tutor aziendale e prevedevano l'utilizzo di un MacBook e tutto il software necessario precedentemente installato. L'azienda, in particolare, ha scelto di utilizzare i seguenti software per lo sviluppo di applicazioni web.

- **Django**, è un web framework con licenza open source per lo sviluppo di applicazioni web, scritto in linguaggio Python, seguendo il paradigma "Model View Template". Il progetto è sviluppato dalla "Django Software Foundation" (DSF), un'organizzazione indipendente senza scopo di lucro. Venne concepito inizialmente per gestire diversi siti di notizie per la World Company di Lawrence (Kansas), e distribuito con una licenza BSD a luglio 2005;

- **Vue.js**, è un framework JavaScript open-source in configurazione "Model View Viewmodel" per la creazione di interfacce utente e single-page applications. È stato creato da Evan You ed è gestito da lui stesso e i membri attivi del core team, provenienti da varie società come Netlify e Netguru;

- **Visual Studio Code** è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Include il supporto per debugging, un controllo per Git integrato, Syntax highlighting, IntelliSense, Snippet e refactoring del codice. Visual Studio Code è basato su Electron, un framework con cui è possibile sviluppare applicazioni Node.js;

- **PostgreSQL** è un completo DBMS (DataBase Management System) ad oggetti rilasciato con licenza libera. Spesso abbreviato come "Postgres", sebbene questo sia un vecchio nome dello stesso progetto. É una reale alternativa sia rispetto ad altri prodotti liberi come MySQL, Firebird SQL e MaxDB, sia a quelli a codice chiuso come Oracle, IBM Informix o DB2, ed offre caratteristiche uniche nel suo genere che lo pongono per alcuni aspetti all'avanguardia nel settore delle basi di dati, ad esempio postgres permette agli utenti di definire nuovi tipi basati sui normali tipi di dato SQLe PostgreSQL, inoltre, permette l'ereditarietà dei tipi;

- **Tower** è un client Git molto potente che permette agli sviluppatori di accedere ai propri repositories tramite interfaccia utente. È possibile utilizzare qualsiasi comando proveniente dal protocollo Git. È un'applicazione estremamente utile e fondamentale quando si lavora in team su uno stesso progetto e si è costretti a ricercare errori o modifiche fatte nei precedenti commit.

3 Sviluppo

In questo capitolo si vanno a descrivere e approfondire tutti i passaggi dello sviluppo di CUBO.

3.1 Browser

Agli esordi di internet, i siti web erano pagine molto semplici, statici e con pochi componenti. Nel 2021, sempre più spesso, si parla di vere e proprie applicazioni web. L'informatica è la scienza, che più di tutte, si sviluppa velocemente ed in questi decenni sono stati inventati e sviluppati modi diversi, migliori e più efficienti per sviluppare siti web.

Avere siti web sempre più impegnativi e con sempre più funzionalità significa avere anche dei 'motori' sempre più performanti e adeguati al contenuto. Questi 'motori' sono i browser. Con il termine browser s'intende un software, proprio come qualsiasi altro programma installato sul computer, pensato in maniera specifica per poter recuperare, presentare e navigare determinate risorse su internet; ad esempio testi, video, canzoni, immagini e svariate altre tipologie di contenuti che vengono identificati attraverso un apposito URL (Uniform Resource Locator), ovvero un nome che identifica univocamente una risorsa all'interno della rete. Esistono svariati tipi di browser, chi più performante, chi più veloce, chi più rispettoso della privacy dell'utente, chi ti offre la possibilità di navigare tramite rete Tor e chi permette perfino di gestire i portafogli di cryptovalute dalla dashboard del browser.

Per lo sviluppo si utilizza Google Chrome, il browser più diffuso con circa il 64% di a livello mondiale.

3.2 Conoscenze preliminari

Per sviluppare un sito web è necessario conoscere i tre linguaggi che un browser è in grado di interpretare: html, css, e javascript.

- **Html** (Hypertext Markup Language) è un linguaggio di markup, cioè definisce un insieme di regole che descrivono i meccanismi di rappresentazione (strutturali, semantici, presentazionali) ed il layout di un testo;
- **Css** (Cascading Style Sheets) è un linguaggio usato per definire la formattazione e lo stile di documenti HTML;
- **Javascript** — discusso precedentemente

Negli ultimi decenni non è più sufficiente la conoscenza base di questi tre linguaggi, ma ad affiancare i programmatori si sono aggiunti i framework (definiti nel capitolo precedente), che forniscono una base già sviluppata di classi e metodi comuni e riutilizzabili nella maggior parte dei progetti. Esistono framework pressochè per ogni linguaggio, ognuno con uno scopo e con delle funzionalità differenti. Oltre ad essi, che sono a supporto degli sviluppatori, hanno preso piede anche i CMS (Content Management System) a supporto dei gestori di siti web, permettendo loro di creare pagine, modificarle, inserire immagini e contenuti rimanendo però esenti (o quasi) dalla programmazione. Un esempio di un cms diffuso è Wordpress, che copre circa il 40% dei siti web attualmente in circolazione, che si stima siano in totale 1,7 bilioni. Grazie a Wordpress chiunque può creare il proprio spazio su internet e rendersi visibile a tutti. L'obiettivo, come già ribadito, di questo tirocinio è creare un cms proprietario, chiamato CUBO, che racchiuda le funzionalità base per rendere indipendenti i clienti, ma che allo stesso tempo sia totalmente personalizzabile dall'agenzia in base alle esigenze di questi ultimi.

Lo sviluppo è partito dallo studio di Django, il framework su cui è basato Cubo. Django, libreria scritta nel linguaggio Python, racchiude al suo interno un set di classi, e metodi che permettono di velocizzare lo sviluppo iniziale di un progetto web. Django è basato sul paradigma model-view-template:

1. creo il **modello**: un insieme di classi per la rappresentazione dei dati. Queste classi forniscono una rappresentazione logica per la crazione delle tabelle sul database e consente di utilizzare gli oggetti per effettuare le operazioni CRUD sui dati presenti sul database;

2. imposto le **viste**, funzioni che gestiscono il flusso dell'applicazione. Grazie a queste è possibile definire il comportamento delle pagine all'interno dell'applicazione in funzione dell'utente. In una vista è possibile prendere una serie di dati dal modello, elaborarli e decidere quale template utilizzare per mostrare il risultato all'utente finale;

3. infine, si definiscono i **template**, file html che descrivono la presentazione dei dati cooperando con la **vista** del pattern ed utlizzando i dati provenienti da essa.

Questa suddivisione consente a diverse figure professionali, di lavorare in modo indipendente alle parti dell'applicazione di propria competenza: gli sviluppatori si dedicheranno al modello dei dati e alle viste, i web designer ai template.

Per quanto riguarda le richieste di dati al database, django effettua una mappatura automatica tra i modelli e le tabelle logiche sul database, evitando

così ai programmatori di effettuare le classiche richieste in linguaggio SQL, ma permettendo di utilizzare oggetti, rendendo la scrittura del codice più ordinata e pulita.

Questa tecnica, chiamata **ORM** (Object Relational Mapping), rende il codice più sicuro dato che impedisce, di natura, qualsiasi attacco di SQL injection: una tecnica di iniezione di codice, usata per infettare applicazioni che gestiscono dati attraverso database relazionali sfruttando il linguaggio SQL. Il mancato controllo dell'input dell'utente permette di inserire artificiosamente delle stringhe di codice SQL che saranno eseguite dall'applicazione server: grazie a questo meccanismo è possibile far eseguire comandi SQL anche molto complessi, dall'alterazione dei dati, alla creazione di nuovi utenti, al download completo dei contenuti nel database.

3.3 Ideazione

Per la creazione del gestore di contenuti ci si è prima concentrati sull'ideazione e sulle caratteristiche principali del cms. Si definiscono applicazioni tutti i moduli aggiuntivi ed opzionali relativi ad un sito web, nello specifico sono:

> chi siamo, banner, blog, catalogo, documenti, homepage, offerte, portfolio, camere, luoghi, recensioni, e-commerce, negozi.

Le caratteristiche principali di cui CUBO deve godere sono:

- **flessibilità** nel decidere di utilizzare moduli diversi in progetti diversi. Per esempio per ogni sito web è possibile poter scegliere se utilizzare o meno le applicazioni (come catalogo o e-commerce) e, in caso di necessità, personalizzare ognuno di essi per il progetto in specifico;
- **soft-coded**, strutturare il codice più possibile dinamico, in modo da apportare modifiche alle singole applicazioni senza dover compromettere l'integrità del resto del codice;
- **modularità**: la possibilità di aggiungere funzionalità recenti ad applicazioni più obsolete senza causare errori;
- **semplice** da gestire per il cliente;
- **aggiornato** alle grafiche moderne per i visitatori

Essendoci una gran mole di applicazioni, si è pensato di suddividere il progetto in classi e sottoclassi, in modo da seguire la filosofia del "code reuse" (riuso del codice), e non dover quindi riscrivere lo stesso script in più punti

diversi del progetto. Con riuso di codice si intende la pratica, estremamente comune nella programmazione, di richiamare o invocare parti di codice precedentemente già scritte ogni qualvolta risulta necessario, senza doverle riscrivere, e nel caso in futuro si dovessero apportare modifiche, applicarle una volta sola.

3.4 Workflow

Il work-flow segue il paradigma MVT (Model-View-Template) precedentemente discusso. Per questa sezione e le successive si prende come esempio il modello *Post* presente nell'applicazione *Blog*.

Ogni modello del CUBO estende la classe *StandardPageMetaModel* che racchiude al suo interno gli attributi di base comuni ad ogni sotto modello, (come titolo, sottotitolo, data di creazione, slug, immagine...). Il modello *Post* comprende, oltre agli attributi dello *StandardPageMetaModel*, anche una categoria, il contenuto del post, i tag e i post correlati. Una volta creato il modello è necessario creare i file di migrazione per il database. Le migrazioni sono il modo tramite il quale i cambiamenti che vengono effettuati nei file dei modelli, vengono effettivamente riflessi anche nelle tabelle 2sul database. Ogni volta che si effettuano modifiche ai file dei modelli, è necessario generare anche i file di migrazione. Django include i comandi per autogenerare i file di migrazioni e per riflettere le modifiche sul database per rendere effettivi i cambiamenti:

```
python manage.py makemigrations
python manage.py migrate
```

La vista è una funzione che viene invocata quando l'utente apre una qualsiasi pagina sul sito web (anche la homepage), e restituisce tutte le informazioni necessarie per strutturare la pagina. Per esempio, da come si può dedurre dall'immagine xx, si tratta della la funzione per recuperare i post appartenenti ad una certa categoria e creare una lista da 12 post, che verrà poi passata al template.

Lo scopo dei template è quello di trasformare le informazioni logiche in elementi di markup html. Come si può notare dall'immagine xxx i dettagli del post (immagine, titolo, e sottotitolo) non sono hard-coded ma sono ricavate dalle variabili recuperate dalla vista. I template sono semplici file html, con una sintassi adattata per le variabili di Django; è inoltre possibile eseguire funzioni direttamente all'interno del codice in caso di necessità.

Quando viene effettuata una richiesta, il server di django effettua tutte le operazioni attraverso le viste e restituisce all'utente una pagina html pura senza variabili o dati sensibili. Questo perché, a differenza di altri framework

basati, django effettua un server-side rendering, cioè, come si anticipato poco fa, ad ogni richiesta, il server effettua tutte le operazioni necessarie, e poi restituisce al client solamente il codice html. Altri framework, come React o Vue che son basati su javascript, effettuano invece tutte le operazioni a livello client attraverso il browser, dinamicizzando così il sito web.

Entrambe le opzioni sono valide, la scelta dipende dal contesto:

- **server-side** rendering: gode di migliori performance, di sicurezza dei dati, ma le pagine web restano statiche;
- **client-side** rendering: gode di peggiori performance (variabili in base a prestazioni del dispositivo e scelta del browser), ma lavorando con javascript, si riesce a rendere le pagine dinamiche senza ricaricarle.

Scegliere una delle due soluzioni non esclude l'altra, ma è possibile combinarle per costruire una struttura sicura attraverso l'isolamento dei dati di Django e dinamica tramite javascript. Per questo la soluzione migliore è l'innesto di script Vue.js nei template django html.