Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Analisi, progettazione e sviluppo del backend di un'applicazione web per la gestione di eventi

Tesi di laurea

Relatore
Prof.Davide Bresolin

Laureando
Alberto Lazari

Anno Accademico 2021-2022

Sommario

La tesi descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, presso la sede di Treviso di Moku S.r.l., il cui obiettivo era la reimplementazione del backend di una piattaforma di gestione di eventi, sfruttando gli strumenti tipicamente utilizzati nei progetti dell'azienda.

In particolare, i seguenti capitoli tratteranno del contesto lavorativo dell'azienda, dell'analisi svolta sullo stato della piattaforma ad inizio stage, della progettazione e successiva implementazione iniziale del nuovo backend, focalizzando l'attenzione sulle scelte stilistiche e architetturali perseguite.

Ringraziamenti

Voglio ringraziare il Prof. Davide Bresolin, per l'interesse, il supporto e l'aiuto fornito durante il periodo di stage e di stesura di questa tesi.

Ringrazio i miei genitori e tutti i miei famigliari per il supporto e l'affetto che mi hanno donato durante questi hanno di studio.

Ringrazio i colleghi di Moku che mi hanno accolto calorosamente tra loro durante la mia esperienza di stage, in particolare Riccardo e Nicolò, per avermi sempre fornito l'aiuto che cercavo durante il mio lavoro.

Ringrazio la Comunità Capi del Mestre 2, per avermi accompagnato fin qui, infondendo in me una maturità e una consapevolezza che mi hanno permesso di raggiungere questo traguardo.

Ringrazio i miei colleghi studenti, stagisti e gli amici dei gruppi di progetto, con cui ho condiviso le difficoltà e le soddisfazioni di quest'anno.

Infine ringrazio i miei amici, che mi sono sempre stati vicini e con cui ho condiviso esperienze indimenticabili.

Padova, Luglio 2022 Alberto Lazari

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

L'azienda

1.1 Descrizione generale

Descrizione dell'azienda: brevissima storia, divisione dei ruoli, spazi e luoghi di lavoro.

1.2 Modello di sviluppo

Modello agile, Scrum, organizzazione dei team.

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

Infinite Area, una piattaforma dell'innovazione, ha ideato e sviluppato un'applicazione web per l'organizzazione e la gestione di eventi e conferenze dal vivo, online e ibride, denominata Evvvents. L'applicazione si propone come un *software as a service*, configurabile nell'aspetto e nelle funzionalità, in base alle preferenze dell'utente. In questo modo ogni utente che possiede un piano di iscrizione all'applicazione, un *tenant*, gestisce la sua versione del software, rendendo Evvvents nel suo complesso un *software multi tenant*. I *tenant* si identificano all'interno dell'applicazione come "piattaforme"; ogni piattaforma gestisce più aziende, dette "organizzatori", perché dirette responsabili della creazione e organizzazione degli eventi, il fulcro del software. Sono presenti diverse funzionalità relative agli eventi:

Sistemi di videoconferenze Per ogni evento è possibile specificare l'integrazione con diversi sistemi di videoconferenze, tra cui:

- Zoom meeting e webinar
- Microsoft Teams
- GoToWebinar
- Webex
- Sistemi esterni di videoconferenze

Sistema di checkin Nelle pagine degli eventi vengono riportati i checkin effettuati in presenza e in remoto, per tenere traccia del numero di utenti partecipanti.

Comunicazione tra utenti È presente un sistema di messaggistica tra i partecipanti, online e in presenza. I partecipanti possono anche avere delle conversazioni con i relatori dell'evento.

Notifiche Ogni piattaforma può decidere di richiedere l'integrazione con il sistema di notifiche push e SMS per gli eventi. Le notifiche possono essere anche inviate via mail ai partecipanti e possono essere automatiche o programmate.

Sistema di crediti Le piattaforme possono anche decidere di utilizzare l'integrazione con il sistema dei crediti, in cui ogni evento permette agli utenti di guadagnare un numero specificato di crediti, in base alla durata della partecipazione all'evento.

Dopo aver realizzato una prima versione prototipale di Evvvents, Infinite Area si è rivolta a Moku, chiedendo di apportare modifiche e miglioramenti all'applicazione. A seguito di un'attività di analisi effettuata da Moku, è risultato che il frontend sarebbe stato mantenuto e fatto evolvere, mentre il backend presentava grossi limiti, funzionali e strutturali. Mantenerlo e modificarlo per raggiungere uno standard di qualità adeguato sarebbe costato più tempo e risorse di una riscrittura completa, quindi è stata presa la decisione di svilupparlo da zero con le tecnologie utilizzate da Moku per prassi aziendale, per quanto possibile, puntando a ricreare una API quanto più simile a quella esistente, per limitare le modifiche necessarie sul frontend.

Lo scopo del progetto di stage è stata la realizzazione della nuova versione del backend, partendo da un'attività di analisi dettagliata della versione originale, passando per la progettazione della nuova versione.

2.2 Requisiti

I requisiti dello stage riportati nel piano di lavoro sono i seguenti, categorizzati per importanza:

Obbligatori Requisiti primari, necessari per una buona riuscita dello stage:

- gestione e pianificazione del progetto attraverso kanban board condivisa;
- analisi dei flussi attuali e delle API richieste;
- progettazione ed implementazione dei modelli e dei controller, a partire dai requisiti raccolti;
- analisi ed integrazione Zoom, GoToWebinar, Webex.

Desiderabili Non necessari, ma che contribuiscono alla completezza del prodotto, se rispettati:

- coordinamento con il cliente finale;
- integrazione team;
- integrazione stampante biglietti;
- suite di testing del software prodotto;
- documentazione completa.

Opzionali Che portano del valore aggiunto al progetto:

- ulteriori modifiche all'applicazione che esulano da quando riportato nel piano di lavoro.

Il conseguimento dei requisiti è stato in parte dipendente dalle decisioni di gestione del progetto da parte del *project manager* e dalle richieste o dalla disponibilità del committente. In particolare è stato chiesto di dedicare una quantità limitata di tempo al testing, perché non richiesto esplicitamente e non è stato possibile implementare le integrazioni, a causa di ritardi nelle risposte del committente, necessarie per definire alcuni aspetti.

2.3 Pianificazione

Il periodo di svolgimento dello stage era previsto tra il 26 aprile 2022 e il 1 luglio 2022, per una durata complessiva di 300 ore. Il periodo preventivato considera due settimane aggiuntive a quelle necessarie a raggiungere 300 ore, utili a coprire eventuali imprevisti. La tabella che segue mostra la pianificazione delle attività da svolgere per ogni settimana, considerando 8 ore di lavoro al giorno:

Settimana	Attività
1 27/04 - 29/04	Comprensione sistema e obiettivi Analisi dei requisiti
2 02/05 - 06/05	Progettazione
3 09/05 - 13/05	Progettazione Studio e setup ambiente di sviluppo
4 16/05 - 20/05	Implementazione
5 23/05 - 27/05	Implementazione
6 30/05 - 03/06	Implementazione
7 06/06 - 10/06	Implementazione Test e validazione
8 13/06 - 17/06	Test e validazione Documentazione

Tabella 2.1: Tabella della pianificazione del lavoro

2.4 Tecnologie utilizzate

Essendo il progetto un lavoro di riscrittura completa, è stato deciso di sfruttare quasi tutte le tecnologie comunemente utilizzate per convenzione aziendale nello sviluppo di backend:

2.4.1 Ruby

Ruby[1] è un linguaggio di *scripting* interpretato, a oggetti e dinamicamente tipizzato, che punta alla produttività, mantenendo una sintassi semplice ed elegante e una bassa curva di apprendimento. La sintassi prevede un uso limitato della punteggiatura e

[1] *Sito ufficiale di Ruby.* URL: `https://www.ruby-lang.org/`.

di elementi considerati superflui, preferendo uno stile spesso naturale e intuitivo, che migliora la leggibilità del codice:

```
user.updated_at = 1.hour.ago
```

In Ruby ogni *statement* è un'espressione, che ritorna un risultato, infatti ogni blocco di codice ritorna il risultato dell'ultima istruzione. Nell'esempio seguente la funzione `get_image_from` ritorna l'immagine dall'array `attribute`, se presente; altrimenti ritorna `nil`, perché l'espressione non ha ritornato un risultato:

```
def get_image_from attribute do
  attribute[:image] if attribute.has_key? :image
end

user.image = get_image_from uploaded_files
```

Il linguaggio è multi-paradigma, ma fortemente orientato agli oggetti: perfino i tipi primiti e i metodi vengono trattati come oggetti. Aggiungendo la possibilità di estendere ogni classe con nuovi metodi, anche successivamente alla sua dichiarazione, si ottiene un linguaggio estremamente flessibile ed estendibile, che consente di adattare il comportamento del linguaggio stesso alle proprie necessità.

Da queste premesse si intuisce come Ruby sia predisposto per essere facilmente estensibile, infatti è presente un RubyGems, il *package manager* per gestire la distribuzione delle *gemme*, pacchetti che possono includere varie funzionalità per estendere il linguaggio, tra cui librerie o eseguibili. La configurazione delle gemme necessarie per un progetto, con le loro dipendenze e versioni, viene eseguita da Bundler, che si occupa di mantenere tutte queste informazioni nei *Gemfile*. Di seguito sono elencate le principali gemme utilizzate nel progetto di stage.

Rails Framework utilizzato, illustrato successivamente in dettaglio.

Devise Fornisce molti metodi e endpoint già configurati per l'autenticazione sicura degli utenti, insieme alla gemma DeviseTokenAuth, che fornisce l'autenticazione tramite token.

Pundit Permette di separare la gestione dei permessi degli utenti dai controller, riducendo anche il codice ripetuto per l'autorizzazione.

2.4.2 Rails

Alcune gemme di RubyGems sono molto complete, tanto da includere all'interno un intero framework. L'esempio principale è la gemma Rails, che fornisce il framework *Ruby on Rails*[2], o *Rails*. Si basa sul paradigma *convention over configuration*, quindi cerca di ridurre al minimo le decisioni che lo sviluppatore deve prendere, al fine di ridurre il codice *boilerplate* e favorire il rispetto del principio *don't repeat yourself* (DRY).

Rails fornisce tutti gli strumenti per sviluppare un'applicazione web basata sul pattern Model-View-Controller (MVC), in particolare le librerie:

[2] *Sito ufficiale di Ruby on Rails.* URL: `https://rubyonrails.org/`.

Active Record Implementa l'omonimo pattern architetturale, in cui si utilizza un database relazionale per memorizzare le informazioni relative ai modelli. Fornisce un'interfaccia per astrarre svariati *database management system* (DBMS), in particolare una contenuta nella gemma PG, che permette di utilizzare un database basato su PostgreSQL, comunemente utilizzato nei progetti di Moku.

Action Controller Fornisce metodi e classi per la definizione dei controller REST dell'applicazione.

Action View Fornisce metodi e classi per la definizione della parte di view dell'applicazione. Nei progetti di Moku non viene generalmente utilizzata, perché si utilizza Rails per sviluppare API backend, che comunicano con view separate. Nel caso del progetto di stage il frontend è sviluppato con Angular.

Active Storage Permette di memorizzare file, come immagini o documenti, su servizi cloud, invece che sul server o nel database. Sono disponibili diversi servizi, Moku utilizza AWS S3.

Active Admin Permette di creare velocemente e con semplicità interfacce per la gestione dei dati nel database. Non è stato usato nel progetto di stage, perché il frontend era già progettato per fornire le stesse funzionalità agli utenti super admin.

2.4.3 API REST

REST è uno stile architetturale per la progettazione di API *stateless*. Si basa su chiamate a degli *endpoint* (*routes* in Rails), coppie formate da un URL, che identifica univocamente una risorsa, e un metodo HTTP, normalmente uno tra GET, POST, PUT, PATCH o DELETE. Viene ampiamente utilizzato nello sviluppo di API ed è supportato nativamente da molti framework.

Essendo uno stile architetturale, REST non è una vera e propria tecnologia, ma in questo caso si contrappone a GraphQL, un linguaggio di query per API, che viene normalmente utilizzato nei progetti di Moku. Per il progetto di stage si è optato per sviluppare una API REST per limitare le modifiche necessarie da fare sul frontend, essendo che questo era già pronto e si basava sulla API precedente, sviluppata secondo lo stile REST. Sarebbe stato troppo costoso modificarne il funzionamento perché potesse funzionare con GraphQL.

Capitolo 3

Analisi e refactor dei modelli

3.1 Introduzione

Spiegazione del lavoro svolto in questa fase.

3.2 Modifiche effettuate

Decisioni significative prese durante l'attività di refactor dei modelli.

3.3 Diagramma ER completo

Diagramma ER del nuovo backend.

Capitolo 4

Progettazione della API

4.1 Introduzione

Spiegazione del lavoro svolto in questa fase.

4.2 Notazione adottata

Spiegazione convenzioni adottate nella descrizione degli endpoint.

4.3 Descrizione delle funzionalità esposte

Descrizione degli endpoint esposti dalla API, in generale per ogni modello e nello specifico per le eccezioni.

4.3.1 Lista delle risorse

Route index, attributi mostrati per ogni modello implementato.

4.3.2 Dettagli di una risorsa

Route show, attributi mostrati per ogni modello implementato.

4.3.3 Creazione di una risorsa

Route create.

4.3.4 Modifica di una risorsa

Route update.

4.3.5 Eliminazione di una risorsa

Route delete.

4.4 Gestione dei permessi

Permessi per le categorie di utenti per ogni controller.

Capitolo 5

Codifica

5.1 Modelli

5.1.1 Migrazioni del database

Comando `rails generate` *e migrazione prodotta.*

5.1.2 Associazioni a modelli e file

Associazioni di Active Record e Active Storage.

5.1.3 Validazioni

Validazioni sugli attributi del modello e le associazioni.

5.2 Controller

5.2.1 APIController

Descrizione dei metodi di utilità ereditati dai controller dell'API.

5.2.2 Implementazione delle action

Descrizione ed esempio di action tipiche dei controller.

5.3 Gestione dei permessi

Funzionamento e uso della gemma "Pundit" per la gestione dei permessi relativi agli endpoint dell'API, scope e metodi relativi alle action, esempio di gestione della gerarchia che andrà rivisto

5.4 Test di unità

Descrizione della gemma "RSpec", che fornisce strumenti per lo sviluppo guidato dal comportamento (behaviour-driven development), esempi di modelli testati.

Capitolo 6

Conclusioni

6.1 Raggiungimento dei requisiti

Tabella con stato di completamento dei requisiti, con commento (dove necessario)

6.2 Valutazione personale

Messe alla prova le competenze fornite dal corso di laurea, verificata l'efficacia dei corsi e dei progetti svolti, imparato un nuovo linguaggio e framework con filosofia di sviluppo a me nuova, scoperto ambiente lavorativo aziendale con i ruoli e le dinamiche interne.

Bibliografia

Siti web consultati

Documentazione di Ruby. URL: https://ruby-doc.org/.

Doyle, Kerry. *Programming in Ruby: A critical look at the pros and cons*. URL: https://www.techtarget.com/searchapparchitecture/tip/Programming-in-Ruby-A-critical-look-at-the-pros-and-cons.

Guide ufficiali di Ruby on Rails. URL: https://guides.rubyonrails.org/.

Hartl, Michael. *Ruby on Rails Tutorial: Learn Web Development with Rails*. URL: https://www.railstutorial.org/book.

Sito ufficiale di Ruby. URL: https://www.ruby-lang.org/.

Sito ufficiale di Ruby on Rails. URL: https://rubyonrails.org/.