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

Storia del progetto prima del mio arrivo, azienda che ha commissionato il progetto, descrizione dello scopo della piattaforma e del suo funzionamento, motivazioni alla base della scelta di riscrittura del backend.

2.2 Requisiti

Requisiti obbligatori, desiderabili e opzionali previsti.

2.3 Pianificazione

Divisione settimanale del lavoro dal piano di lavoro, incluse correzioni.

2.4 Tecnologie utilizzate

Descrizione della configurazione del framework Ruby on Rails utilizzata: librerie utilizzate, Postgres, AWS, API REST .

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

La codifica dei modelli passa per tre fasi successive:

1. creazione della entità del modello nel database e della classe, utilizzando le migrazioni;
2. l'associazione del modello con altri modelli o elementi di *storage*;
3. le validazioni sugli attributi e sulle associazioni dichiarate.

5.1.1 Migrazioni del database

Basandosi su quanto definito nella fase di progettazione dei modelli, descritta nel capitolo 3, questi sono stati generati utilizzando da linea di comando il generatore automatico `rails generate model` o, in versione ridotta, `rails g model`. Il comando accetta come argomenti:

- il nome del modello, al singolare e in *CamelCase*;
- gli attributi che deve avere il modello;
- per ogni attributo: il suo tipo, che rispecchia, ad alto livello, i tipi comunemente disponibili per le colonne nei DBMS SQL. Normalmente è uno dei seguenti tipi nativi delle migrazioni di Rails[1], agnostici rispetto all'implementazione del database:
  - `primary_key`,
  - `string`,
  - `text`,
  - `integer`,
  - `bigint`,
  - `float`,
  - `decimal`,

[1] *Documentazione ufficiale di Ruby on Rails - metodo* ***`add_column`***. URL: `https://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html#method-i-add_column`.

  - `datetime`,
  - `timestamp`,
  - `time`,
  - `date`,
  - `binary`,
  - `blob`,
  - `boolean`,
  - `references`

- per ogni attributo: l'identificatore `uniq`, che imposta un indice su quella colonna del database, che ne specifica l'unicità nell'entità.

Di conseguenza, la sintassi generale è la seguente:

```
rails g model ModelName attr_1:type:[uniq] attr_2:type:[uniq] ...
```

Portando un esempio reale, per la generazione del modello degli organizzatori è stato usato il comando seguente:

```
rails g model Organizer name:string vat:string:uniq sdi:string:uniq
↪  address:string city:string country:string zip_code:string status:integer
↪  email:string email_pec:string phone:string finished_webhook:string
↪  add_participant_webhook:string remove_participant_webhook:string
↪  platform:references creator_id:bigint
```

che ha prodotto la seguente migration:

```ruby
# db/migrate/{timestamp}_create_organizers.rb

class CreateOrganizers < ActiveRecord::Migration[7.0]
  def change
  create_table :organizers do |t|
    t.string :name
    t.string :vat
    t.string :sdi
    t.string :address
    t.string :city
    t.string :country
    t.string :zip_code
    t.integer :status
    t.string :email
    t.string :email_pec
    t.string :phone
    t.string :finished_webhook
    t.string :add_participant_webhook
    t.string :remove_participant_webhook
    t.references :platform, foreign_key: true
    t.bigint :creator_id

    t.timestamps
  end
```

```
  add_index :organizers, :vat, unique: true
  add_index :organizers, :sdi, unique: true
  end
end
```

Successivamente questa è stata modificata, aggiungendo i vincoli `NOT NULL`, la chiave esterna verso l'utente creatore e il valore di default per il ruolo, prima di eseguire effettivamente la migrazione.

Si noti come non sia necessario specificare la chiave primaria. Il comportamento di *default* di Active Record è introdurre automaticamente un identificativo progressivo, chiamato `id`, di tipo `bigint`. Inoltre `timestamps` genera automaticamente degli attributi gestiti dalla gemma, per tracciare l'istante di creazione e ultima modifica dei record.

La migrazione definitiva è stata la seguente:

```ruby
# db/migrate/{timestamp}_create_organizers.rb

class CreateOrganizers < ActiveRecord::Migration[7.0]
  def change
    create_table :organizers do |t|
      t.string :name, null: false
      t.string :vat, null: false
      t.string :sdi, null: false
      t.string :address, null: false
      t.string :city, null: false
      t.string :country, null: false
      t.string :zip_code, null: false
      t.integer :status, null: false, default: Organizer.statuses[:active]
      t.string :email
      t.string :email_pec
      t.string :phone
      t.string :finished_webhook
      t.string :add_participant_webhook
      t.string :remove_participant_webhook
      t.references :platform, null: false, foreign_key: true
      t.bigint :creator_id, null: false

      t.timestamps
    end
    add_foreign_key :organizers, :users, column: :creator_id
    add_index :organizers, :vat, unique: true
    add_index :organizers, :sdi, unique: true
  end
end
```

Utilizzando il metodo `change`, le migrazioni possono modificare la struttura del database secondo quando specificato nella migrazione, senza necessità di ricorrere a *downtime* del server e di eseguire il rollback alla versione dello schema del database precedente, se fosse necessario.

Il generatore produce altri due file, oltre alla migrazione:

```ruby
# app/models/organizer.rb
class Organizer < ActiveRecord::Base
  belongs_to :platform
end

# spec/models/organizer_spec.rb
require 'rails_helper'

RSpec.describe Organizer, type: :model do
  pending "add some examples to (or delete) #{__FILE__}"
end
```

Il primo contiene la definizione della classe, in cui andranno inseriti i metodi, le validazioni e le associazioni sul modello, descritte in §5.1.2 e §5.1.3. Nel secondo andranno definiti i test di unità per il modello, descritti in §5.4.

5.1.2 Associazioni a modelli e file

Una volta generata la struttura del modello attraverso le migrazioni del database, è stato necessario associare tra loro i modelli, al livello dell'applicazione, secondo le relazioni espresse nel diagramma ER prodotto durante la fase di analisi e refactor (§3). Per farlo, sono stati utilizzati i metodi forniti da `ActiveRecord::Base`, classe ereditata da tutti i modelli. Nel progetto, in realtà, tutti i modelli ereditano da `ApplicationRecord`, che a sua volta eredita da `ActiveRecord::Base`, ma viene utilizzato per aggiunge metodi di utilità a tutti i modelli implementati.

Rails incentiva l'implementazione di associazioni bidirezionali, attraverso l'utilizzo dei metodi:

- `belongs_to`: utilizzato per specificare l'associazione con il modello di cui la classe memorizza la chiave esterna;
- `has_one`: specifica l'associazione con un record di un modello che memorizza la chiave esterna alla classe;
- `has_many`: specifica l'associazione con più record di un modello che memorizza la chiave esterna alla classe;
- `has_and_belongs_to_many`: permette di specificare associazioni del tipo "molti a molti", utilizzando una tabella, creata manualmente, che possiede le chiavi esterne ad entrambi i modelli coinvolti.

Oltre alle associazioni con i modelli sono state specificate le associazioni con i file. Queste vengono gestite con la gemma Active Storage, che fornisce i metodi per eseguire l'associazione (*attach*) dei file, chiamati *attachments*: `has_one_attached` e `has_many_attached`.

Proseguendo con l'esempio dell'implementazione degli organizzatori, il file della classe `Organizer` con le associazioni dichiarate risulta essere il seguente:

```ruby
class Organizer < ApplicationRecord
  belongs_to :platform
  belongs_to :creator, class_name: 'User', inverse_of: :created_organizers,
  ↪  optional: true
```

```
  has_many :users, dependent: :nullify
  has_one :managed_platform, class_name: 'Platform', inverse_of:
  ↪  :main_organizer, dependent: :nullify
  has_many :locations, dependent: :nullify

  has_one_attached :logo
end
```

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 ufficiale di Ruby on Rails - metodo `add_column`. URL: https://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html#method-i-add_column.