Documento di migrazione

Versione: 0.1 ***Data:*** 18/06/2019
Autore: Pellizzari Luca ***Responsabile:*** Baradel Luca

Indice

Introduzione

L'applicazione web per la gestione delle incombenze legate al GDPR in via di sviluppo è stata scritta utilizzando principalmente quattro linguaggi di programmazione: HTML e CSS per gestire il lato front-end, PHP per gestire il lato back-end, quindi la comunicazione con la base di dati, e JavaScript per il passaggio dei dettagli relativi agli eventi alle librerie di FullCalendar (utilizzate per la visualizzazione del calendario all'interno delle varie schermate). Al momento della scelta iniziale su quale fosse il miglior modo per aiutare il cliente a risolvere una parte del problema che ci ha presentato si è deciso di sviluppare un'applicazione web con i quattro linguaggi sopra citati in quanto erano quelli più adatti per arrivare in tempi brevi a sviluppare un primo prototipo che potesse soddisfare le esigenze del cliente. In particolare, è stato scelto PHP perché i membri del team avevano già una certa familiarità con questo linguaggio, inoltre PHP permette di ottenere risultati concreti con uno sforzo richiesto relativamente basso.

Descrizione problematiche riscontrate

Il primo dei problemi riscontrato dal team di sviluppo riguarda l'architettura del sistema: il primo prototipo costruito dal team utilizzando PHP (versione 0.1 del codice consegnata al cliente) non aveva un'architettura ben definita. Il passaggio dei parametri fra il front-end ed il back-end avveniva tramite frammenti di codice PHP inserito all'interno dei file HTML che contenevano le varie schermate dell'applicazione. Il codice che gestiva le varie comunicazioni con il database (inserimento e modifica dei dettagli di un evento) veniva tenuto in file in cui oltre alle query (inserimento e modifica) erano presenti anche le varie validazioni (data inizio minore della data di fine, utente fra i partecipanti, eccetera) fatte con una serie di `if` a cascata. In questo modo il codice risultava caotico e non rispettando il Single Responsability Principle che stabilisce che ogni modulo, classe o funzione ha una responsabilità su una singola parte di una funzionalità del sistema software.

Il secondo problema riguarda la gestione delle interazioni fra PHP e FullCalendar. FullCalendar è una libreria scritta in JavaScript che mette a disposizione la possibilità di utilizzare un calendario Drag and Drop all'interno di un'applicazione. Questa libreria si aspetta di ricevere un array (JavaScript) di eventi dove ogni evento ha (almeno) tre campi: titolo, data inizio e data fine. Dal momento che la comunicazione con il database avviene utilizzando PHP, per passare gli eventi al file JavaScript che permette di fare il rendering del calendario abbiamo inserito all'interno di una pagina HTML, un frammento di codice PHP che andava a prendere i dettagli degli eventi e li salvava in un array che poi veniva letto da uno script JavaScript presente all'interno della stessa pagina HTML e poi convertito in un formato leggibile da FullCalendar. Anche in questo caso abbiamo diverse operazioni che dal punto di vista logico andrebbero separate, inoltre questo metodo di passaggio dei parametri è sconsigliato e andrebbe gestito in modo migliore, ad esempio utilizzando richieste Ajax.

Il terzo problema è parzialmente collegato al secondo e riguarda sempre il passaggio di parametri fra PHP e FullCalendar. Dal momento che il documento dei requisiti specifica che deve essere possibile modificare i dettagli relativi agli eventi e che il cliente ha chiesto la possibilità di accedere al form di modifica di un evento tramite click dal calendario le possibili soluzioni sono due: utilizzare un ascoltatore jQuery che al click della casella relativa all'evento sul calendario effettui una richiesta Ajax passando l'id dell'evento al form per la modifica oppure definire per ogni evento un url a cui andare quando si clicca sulla casella dell'evento sul calendario. In questo secondo caso bisognerebbe usare come url il path del file PHP contenente il form di modifica e a questo url andrebbe aggiunto in coda un parametro del tipo "&id" che permette al form di avere l'id dell'evento in modo da poter effettuare una query tramite PHP per precompilare il form con i suoi dettagli. Anche queste operazioni potrebbero essere gestite in modo migliore, soprattutto in un'ottica in cui in fasi avanzate dello sviluppo del prodotto vengano introdotti nuovi requisiti che potrebbero complicare la struttura (già complessa per un numero molto ridotto di funzionalità) del sistema.

Oltre a questi problemi i membri del team di sviluppo, che nel frattempo hanno utilizzato il framework Rails in altri progetti, hanno notato che la gestione dell'applicazione potrebbe essere semplificata di molto sfruttando le funzionalità messe a disposizione dal framework.

Vantaggi dell'utilizzo del framework Rails

Quando ci si trova a dover sviluppare un'applicazione, una scelta può essere quella di appoggiarsi ad un framework: un framework è un'infrastruttura che fornisce un certo livello di supporto durante lo sviluppo di un sistema software; in questo modo i programmatori possono astrarre da alcuni dettagli (ad esempio come costruire un'architettura per l'applicazione) e sfruttare le funzionalità messe a disposizione dal framework. Nel caso specifico di Rails, l'architettura su cui è costruito il framework è un'architettura MVC (Model, View, Controller) dove il Model è la componente che si occupa della gestione delle comunicazioni con la base di dati, il Controller si occupa del passaggio dei dati fra il Model e le viste, mentre le View sono appunto le viste ovvero le schermate dell'applicazione. Queste tre componenti permettono di ottenere un'architettura stabile e mantenibile in cui ogni componente ha le sue responsabilità; il framework funziona bene utilizzando questa architettura quindi è sufficiente che il programmatore rispetti le convenzioni che sono alla base dell'utilizzo del framework per ottenere un'applicazione con una struttura solida e mantenibile. Ad esempio, una di queste convenzioni, per quello che abbiamo appena detto sull'architettura MVC, è quella di evitare di inserire troppi frammenti di codice all'interno delle viste in quanto queste dovrebbero solo utilizzare i dati ricevuti dal Controller o dall'utente e non dovrebbero contenere al loro interno una parte della logica dell'applicazione.

Di seguito elenchiamo alcuni dei vantaggi che potrebbero derivare dall'utilizzo del framework Ruby on Rails:

1) Possibilità di avere validazioni a livello dei modelli: ad ogni modello è possibile associare un set di validazioni che verranno effettuate prima dell'operazione di salvataggio di un singolo record. Ad esempio, al modello Event è possibile aggiungere un set di validazioni che controllino la presenza di alcuni campi, oppure è possibile effettuare delle validazioni più specifiche definendo delle funzioni custom per validare il valore di un singolo campo. In questo modo oltre alle validazioni lato front-end fatte dal form (inserendo ad esempio il campo "required" nelle select o negli input), si hanno a disposizione delle validazioni più forti che permettono di evitare che record con dati errati vengano salvati nella base di dati;
2) Gestione comoda del path: il framework Rails mette a disposizione degli helper che possono essere chiamati per creare i vari link fra le pagine all'interno dell'applicazione, in questo modo il programmatore non deve sporcarsi le mani per gestire i vari redirect ad esempio dopo la creazione di un nuovo record o dopo la sua modifica ma può sfruttare gli helper forniti dal framework; questa logica è racchiusa all'interno del router, che traduce un set definito di URI leggibili all'utente in percorsi leggibili dall'applicazione, gli URI presentati all'utente hanno la caratteristica di essere semplici, leggibili e danno un'immediata e chiara idea della struttura dell'applicazione. Il router inoltre può essere responsabile del passaggio di dati, specificandone le nomenclature, attraverso le richieste HTTP, inoltre sono possibili elaborazioni più avanzate non previste nel progetto.
3) Gestione semplificata di FullCalendar tramite una gemma apposita: il framework Rails permette di integrare all'interno dell'applicazione delle librerie ausiliarie tramite l'installazione di specifiche "gemme". Ad esempio, per utilizzare FullCalendar all'interno di un'applicazione Rails è sufficiente installare la gemma "fullcalendar-rails" che permette di utilizzare tutte le funzioni di FullCalendar all'interno dell'applicazione senza la necessità di dover integrare manualmente la libreria e di interagire con FullCalendar usando JavaScript. Per quanto riguarda gli obiettivi del progetto corrente è richiesta l'installazione di un certo numero di gemme oltre a quella appena citata, come ad esempio:
    a. "devise" che permette una comoda gestione del login; anche in questo caso sarà sufficiente chiamare alcune funzioni ausiliarie come ad esempio "user_signed_in?" per vedere se

l'utente è loggato, "current_user" per accedere ai dettagli dell'utente corrente e altre funzioni di questo tipo;
   b. "bootstrap" che permette di utilizzare tutte le librerie e gli stili di bootstrap all'interno dell'applicazione;
   c. "jquery-rails" che permette di utilizzare le funzioni di jquery (utili anche per il corretto funzionamento dei componenti bootstrap);
   d. "faker" per la generazione di dati casuali per i test;
   e. "yard" per la generazione automatica di documentazione all'interno del codice, ad esempio, dopo aver eseguito il comando per generare la documentazione, nel codice di un modello possiamo trovare (sotto forma di commento) la struttura della tabella della base di dati con cui il modello comunica. In questo modo quando dobbiamo scrivere le varie validazioni o funzioni ausiliarie per il modello, nello stesso file abbiamo i nomi di tutti i campi della tabella quindi non si perde tempo ad andare a cercare i nomi dei vari campi (su cui ad esempio dobbiamo fare dei controlli) nella base di dati;
   f. Libreria di supporto alla programmazione e allo sviluppo come "better-errors", "awesome-print" ,"annotate", e altre librerie che rendono più semplice e meno tedioso il lavoro.
4) Generazione automatica di codice: tramite i comandi "rails generate scaffold nome_modello campo1:tipo1, campo2:tipo2, …" è possibile generare Model, Views e Controller per il modello di cui abbiamo specificato il nome. Questo comando genera anche una tabella nella base di dati con i campi che abbiamo specificato precedentemente, quindi con una riga di codice abbiamo la tabella, le varie schermate che mi permettono di visualizzare i record presenti nella tabella e di modificarli (form di creazione a modifica) e il controller che effettua i reindirizzamenti fra le varie schermate. Ovviamente gli stili di queste schermate sono molto semplici ma sono completamenti personalizzabili ad esempio utilizzando bootstrap. La generazione automatica di codice richiede un minimo di conoscenza del framework da parte del programmatore, che anche se non deve generare tutto il codice manualmente deve comunque sapere quello che sta facendo;
5) Test Driven Development: il framework Rails di default permette di generare alcuni set di test in modo automatico: chiaramente questi test sono molto di base ma comunque sono una buona base di partenza e sono facilmente modificabili in modo che possano essere mantenuti aggiornati man mano che vengono effettuate delle modifiche alle tabelle della base di dati;
6) Convention over Configuration, è il principio fondamentale del framework e consiste nel fornire allo sviluppatore un "golden path" da seguire per effettuare la maggior parte delle operazioni, seguendo queste convenzioni prestabilite la quantità di codice scritto diminuirà drasticamente, riducendo anche la quantità di errori e il tempo per risolverli (riducendo la quantità di codice da controllare). Grazie a questo principio è possibile creare uno scheletro di applicazione funzionante rispettando tutti requisiti di business logic (non grafici o di user experience) più semplici in un tempo molto ridotto;

Uno degli svantaggi che potrebbe derivare dall'utilizzo di un framework è lo sforzo richiesto per imparare ad utilizzare le sue funzionalità (almeno quelle principali) e questo è stato il motivo per cui il team ha scelto inizialmente di utilizzare degli strumenti che già conosceva. Poi quando i vari membri del team hanno acquisito una certa familiarità con Rails si è deciso di effettuare la migrazione.

Analisi di impatto

Nelle precedenti sezioni del documento abbiamo descritto i motivi che ci hanno portato a decidere di effettuare la migrazione e i possibili vantaggi che potrebbero derivare da essa. In questa sezione del documento riportiamo una stima indicativa di quale sarebbe lo sforzo in termini di tempo per ricostruire il

sistema attualmente in via di sviluppo (versione del codice 0.1) utilizzando il framework Rails. I membri del team hanno sostenuto una riunione in cui hanno parlato di questo e qui riportiamo solamente le decisioni prese e le stime fatte, non l'intero discorso fatto durante la riunione perché non avrebbe una grande utilità per lo scopo del documento. Nel corso della riunione sono state identificate alcune possibili problematiche da parte dei membri del team che potrebbero riguardare la migrazione ma dopo una breve fase di ricerche online si è visto che i problemi identificati erano tutti facilmente risolvibili.

Di seguito riportiamo i passi da eseguire per ricostruire il sistema software utilizzando Rails:

1) Terminale: "rails new nome_progetto": questo comando genera tutti i file necessari al framework per gestire un nuovo progetto;
2) Uno dei file generato dal comando precedente si chiama Gemfile e contiene la lista di tutte le gemme che saranno utilizzate nel progetto, a questo file, che di defautl già contiene un certo numero di gemme, ne vanno aggiunte altre 9 che i membri del team hanno identificato durante la riunione. Dopo averle aggiunte è sufficiente lanciare il comando "bundle install" affinché le nuove gemme vengano integrate nel progetto;
3) Ora il progetto è già funzionante, se lanciamo il server veniamo reindirizzati ad una home page di benvenuto. Il prossimo passo è quello di utilizzare il comando "rails generate scaffold Event …" e successivamente "rails generate scaffold Event_Typology …"; con questi due comandi il framework genera Model, Views e Controller per gli eventi e per le tipologie di evento. Dopo aver lanciato questi due comandi abbiamo già un'applicazione web funzionante che ci permette di inserire eventi e tipologie e se abbiamo già specificato la chiave esterna fra i due modelli ci impedisce di inserire un evento se non specifichiamo la sua tipologia;
4) Al punto 2 abbiamo già installato la gemma "devise" che gestisce il login e la gemma "fullcalendar-rails" che ci permette di utilizzare la libreria fullCalendar all'interno del nostro progetto, ora dobbiamo effettuare alcune azioni di configurazione per queste due gemme. Il tempo totale per queste due azioni è circa 15 minuti, link ai tutorial:
    a. https://github.com/plataformatec/devise ;
    b. https://medium.com/@a01700666/fullcalendar-in-ruby-on-rails-f98816950039 ;
5) Come prossimo passo dobbiamo definire una struttura per la navigazione: dopo aver effettuato il login l'utente verrà reindirizzato ad una home page che mostrerà il calendario e le notifiche, da qui ci saranno dei link per andare alle pagine di visualizzazione eventi/tipologie e dei link (per adesso fittizi) che manderanno alla pagina con un form per effettuare una segnalazione di un data breach e alla pagina di un form per segnalare una richiesta di esercizio dei diritti;
6) Integrare la libreria per la gestione della visualizzazione delle notifiche a popup, per questo libreria utilizziamo un cdn (link all'interno della home che va a cercare online la libreria e le sue funzioni), la logica per l'utilizzo di queste funzioni era già stata quasi completamente implementata nella versione del codice 0.1 ed essendo file JavaScript è possibile importarli nella nuova applicazione senza problemi;
7) Migliorare le grafiche utilizzando bootstrap, questa azione richiede un po' di tempo in quanto nell'applicazione precedente non abbiamo utilizzato bootstrap ma i membri del team già conoscono questa libreria e quindi non sarà un problema.

Arrivati a questo punto abbiamo ottenuto un'applicazione web funzionante che svolge esattamente gli stessi compiti di quella che corrisponde alla versione 0.1 del codice di questo progetto; inoltre abbiamo anche la possibilità di modificare/cancellare gli eventi, funzionalità che nella versione precedente non era ancora stata implementata. Ora mancano alcuni passi per coprire tutti i requisiti che sono stati richiesti dal cliente:

8) Scrivere un file di seed che popoli la base di dati con un certo numero di eventi/tipologie per poter fare i test;

9) Terminare l'implementazione della visualizzazione delle notifiche;
10) Inserire il concetto di utente amministratore (può vedere tutti gli eventi, non solo quelli di cui è un partecipante).

Per implementare i passi 1-7 descritti precedentemente il tempo di lavoro stimato è di 2 ore di Pair Programming; questo perché sono necessarie le competenze di tutti i membri del team per configurare correttamente un nuovo progetto in Rails.