Università degli studi di Torino
Dipartimento di Informatica

Corso di laurea in informatica

Tesi di laurea

Strumenti web per la gestione e visualizzazione di grandi moli di dati georeferenziati

Relatore:
Prof. Torasso Pietro

Candidato:
Andrea Peretti

Sessione: Settembre 2014
a.a. 2013/2014

1 ottobre 2014

Indice

Capitolo 1

Introduzione

L'informatica è uno dei settori tecnologici che si evolvono con maggiore velocità. La Prima legge di Moore[1] asserisce che la potenza dei calcolatori raddoppia ogni diciotto mesi. Una simile progressione in un settore tecnologico assai più anziano come l'aeronautica, permetterebbe oggi di fare un giro completo della terra in una manciata di secondi.

Questa potenza di calcolo permette lo sviluppo di applicazioni impensabili fino ad una decina di anni fa, capaci di trattare una quantità di dati da elaborare estremamente grande. Un esempio significativo è l'ammontare di dati che viene generato dai sensori dell'acceleratore di particelle del CERN di Ginevra, sono circa 15 PetaByte l'anno (fonte: [Rus12]), i quali devono essere memorizzati e poi elaborati. Grandi siti web tutt'oggi generano dati analoghi, ed ogni giorno vengono memorizzati diversi terabyte di dati a seguito di ricerche, acquisti, pubblicazione di contenuti sui social network, eccetera. L'infografica nella figura 1.1 ci permette di vedere con immediatezza di quali quantità stiamo parlando, e possiamo evincere che per una tale mole di informazioni occorrono delle procedure di trattamento dell'informazione diverse da quelle che possono essere adottate da un programma per gestire una semplice rubrica. Stiamo parlando quindi di diversi GigaByte se non TeraByte di dati, che hanno bisogno di essere analizzati e processati molto velocemente. I tipi di elaborazione necessari possono essere il calcolo di statistiche, una suddivisione per categoria, un'interrogazione della base di dati, oppure una serie di calcoli che possono permettere ad un'azienda di capire, in base agli acquisti dei suoi clienti, i trend riguardo i suoi prodotti.

Le tecniche elaborate nel corso degli anni hanno portato alla nascita di una nuova branca dell'informatica, quella dei *Big Data*. Con questo termine non si intendono solo i dataset di grandi dimensioni, ma anche le tecniche utilizzate per *elaborarli* ed estrarre informazioni significative da esso.

[1] La prima legge di Moore è un'osservazione empirica di Gordon Moore, cofondatore di *Fairchild Semiconductor* e di *Intel*.

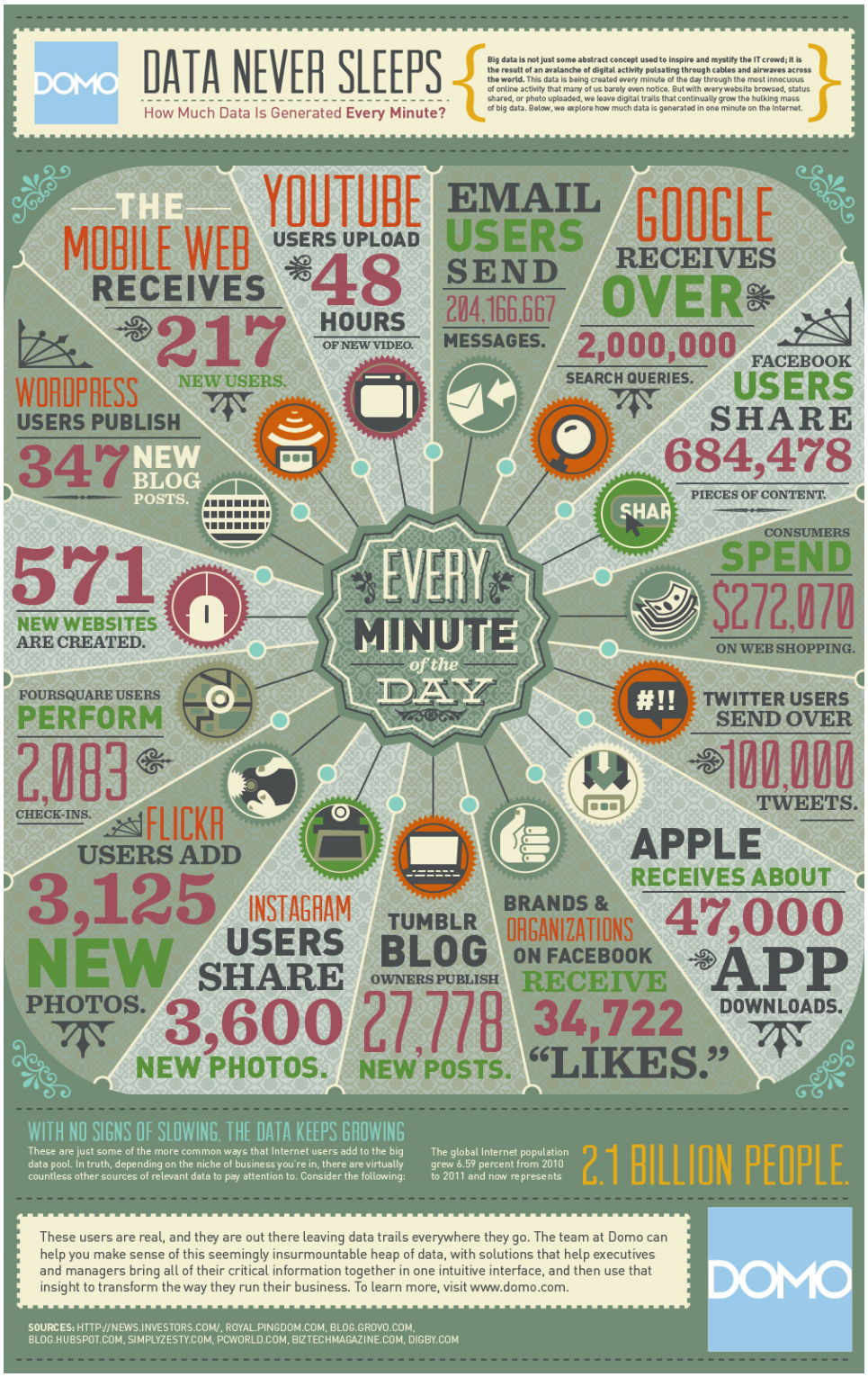


Figura 1.1: Quantità di dati al minuto generati dagli utenti di alcuni siti in rete. Fonte: [Jam12]

Spesso non è vero che i dati di un dataset siano omogenei e che tutti rispettino la stessa struttura, perché possono essere il risultato dell'unione di più basi di dati, l'aggiunta di nuovi record può essere effettuata da entità diverse le quali non si preoccupano di seguire una forma standard, per questo motivo è necessario che siano prima *elaborati*. L'*obbiettivo* di queste tecniche è quindi quello di estrarre *informazioni significative*, per le quali intendiamo qualsiasi tipo di statistica che possa essere calcolata solamente prendendo in considerazione tutti i dati esistenti, impossibile da calcolare a mano o farsene un'idea "guardando" i dati. Tornando alla figura 1.1, il risultato dell'elaborazione dei dati è stato presentato in modo da avere un chiaro riscontro, dando la possibilità al lettore di trarre immediatamente le proprie conclusioni sulla globalità dei dati. Si può suddividere il processo che vede entrare in gioco i big data in tre parti:

1. Raccolta dei dati
2. Elaborazione
3. Visualizzazione

La raccolta dei dati è la prima parte del processo, e tratta la memorizzazione dei dati ed il loro mantenimento. Non è detto che memorizzazione avvenga su database, ma in qualunque modo, come file di testo o fogli elettronici. Spesso però questa fase è effettuata per altri scopi piuttosto che per effettuare delle elaborazioni tramite gli strumenti dei Big Data. Questo può comportare un dataset formato da dati provenienti da fonti diverse, che utilizzano tecnologie e standard diversi che devono essere aggregati, inoltre non tutti i record potrebbero essere completi, con campi mancanti, dati non consistenti eccetera. Anche questi problemi dovranno essere affrontati.

La parte di elaborazione prevede una serie di tecniche atte ad analizzare tutti i dati in un tempo accettabile, ed effettuare i calcoli. Questa parte prevede anche tutti quegli algoritmi statistici utili ad estrarre informazioni significative da campioni di dati. La parte di visualizzazione si occupa di presentare i risultati in modo che sia possibile interpretarli immediatamente e farsi un'idea delle proprietà emerse dall'elaborazione rispetto alla totalità dei dati.

Nel cap. 2 si parlerà di concetti alla base del funzionamento del progetto e degli strumenti utilizzati nello sviluppo, il capitolo 3 descrive il progetto Piedmont Heritage mentre il cap. 4 il suo funzionamento in dettaglio, con la descrizione delle porzioni di codice più significative. Infine nel cap. 5 si descriverà una sessione ipotetica di un utente in Piedmont Heritage.

Capitolo 2

Concetti e strumenti utilizzati

In questo capitolo verranno trattati alcuni concetti teorici la cui conoscenza è necessaria per fornire un quadro generale degli argomenti trattati. Nel Paragrafo 2.1 si parlerà dei *Big Data* e dei tre punti trattati nel capitolo 1 dando una descrizione più approfondita. Successivamente nel paragrafo 2.2 si parlerà del *pattern MVC*, un metodo di strutturazione del software molto utilizzato e potente. Infine, nei cap. 2.3 e 2.4 verranno descritti gli strumenti utilizzati per lo sviluppo del progetto.

2.1 Big Data

La definizione fornita da [Rez13] sui Big Data è che

> sono dati che superano i limiti degli strumenti di database tradizionali.

Essi sono una mole enorme di dati, spesso senza una struttura definita, difficili da trattare con i mezzi di computazione tradizionali. Con Big Data però, non si intendono solo i dati, ma anche le tecnologie utilizzate per manipolarli.

In questi paragrafi verranno discussi i tre punti già elencati nel Capitolo 1 a pagina 7, ossia *acquisizione*, *elaborazione* e *visualizzazione* rispettivamente nei Paragrafi 2.1.1, 2.1.2 ed 2.1.3.

2.1.1 Acquisizione

L'acquisizione dei dati è la parte iniziale del processo, in cui vengono memorizzati. La definizione di questa fase potrebbe finire qui, siccome non esistono altre "regole" per l'acquisizione dei dati nei Big Data: non viene imposta una struttura uguale per tutti i record, la completezza dei dati o che siano memorizzati tutti nello stesso luogo. Inoltre è anche possibile dire che molti dataset non sono pensati per un utilizzo futuro per l'estrazione di

informazione. Infatti nel mondo reale l'intuizione delle potenzialità dei dati avviene solo dopo alla memorizzazione, si pensi ai file di log dei sistemi unix o dei server web, che sono salvati su file di testo con centinaia di migliaia di righe. In altri casi i record vengono salvati in basi di dati da diverse organizzazioni, ognuna con i propri standard, comportando strutture dei dati diverse e probabilmente server dislocati in posizioni diverse.

Per questo motivo, enormi dataset possono trovarsi senza una struttura universalmente adottata da tutti i dati, con campi non compilati e provenienti da basi di dati diverse. Una fase tra l'acquisizione e l'elaborazione dei dati è quella della pulizia dove si tenta di riparare le tuple con campi mancanti ed in caso di fallimento non utilizzandola[1]. Inoltre si dovrà unire le diverse basi di dati se provenienti da diversi dataset[2] utilizzando una struttura standard.

- Individuare i campi che indicano la stessa informazione

  Dataset diversi possono individuare in modo diverso una stessa informazione, sarà compito del data analist di individuarli e creare un nuovo campo in cui memorizzare l'informazione, così da utilizzare solo quest'ultima durante la fase di elaborazione. In dataset di latte di vernici, un'azienda potrebbe utilizzare un campo chiamato *col* per indicare il colore della vernice contenuta nella latta, mentre un'altra potrebbe utilizzare un riferimento ad un record in un altra tabella contenente l'elenco dei colori. Occorre quindi adottare una soluzione per omogeneizzare la rappresentazione del colore.

- Standardizzare il dominio dei campi

  Le fonti dei dati possono essere istituzioni provenienti da parti diverse del mondo, dove vengono adottati standard diversi per la memorizzazione dell'informazione. Ad esempio, in Italia per misurare le distanze usiamo i Chilometri, mentre negli Stati Uniti vengono utilizzate le miglia. Al momento dell'unione di un dataset italiano ed uno statunitense, si dovrà decidere quale unità di misura adottare, e convertire tutte quelle che ne utilizzano una diversa.

- Scartare le tuple non valide

  Siccome i *Big Data* sono un insieme di dati eterogenei, quando si cerca di estrarre l'informazione da essi si devono individuare dei campi significativi, nel linguaggio dei database relazionali si direbbero *required*,

[1] Ovviamente senza compromettere il significato dei dati! Se abbiamo un database di latte di vernici, ed in una tupla non è memorizzato il nome del colore, potremmo risalire ad esso dal codice del colore presente in un altro campo della tupla. Nel caso in cui non sia presente nemmeno quello, non possiamo inventarci che la latta contiene la vernice gialla, la tupla andrà scartata!

[2] Questa operazione, ovviamente, è possibile solo se i dataset contengono informazioni sulla stessa tipologia di dati.

che devono essere presenti al fine di creare un'informazione significativa. Nel caso in cui un campo necessario non sia settato, è opportuno non prendere in considerazione quel record, siccome non ci fornisce informazione. Una similitudine con la lingua italiana potrebbe essere la mancanza di uno tra soggetto, complemento oggetto e verbo in una frase. *Piero ha una ferramenta* ha un significato ben preciso, mentre *Piero ha* oppure *una ferramenta* hanno ben poco da dirci, perché non sappiamo cosa possiede Piero nel primo caso, mentre nel secondo caso non sappiamo se una ferramenta è posseduta da qualcuno, oppure se è aperta, costosa, redditizia eccetera.

Un record contenente informazioni inconcludenti di questo tipo non ci fornisce quindi informazioni. Potrebbe sembrare incorretto non prenderlo in considerazione, siccome è nel dataset per qualche motivo è stato inserito. Potrebbe essere stato corrotto da una chiusura improvvisa del programma che stava effettuando la memorizzazione, oppure il supporto su cui è memorizzato contiene degli errori. Dopotutto però, siccome non possiamo estrarre l'informazione dal record non ci resta che scartarlo, ed al massimo, tenere in considerazione che abbiamo incontrato un record corrotto.

- Selezionare solo i campi utilizzati

  Se nell'analisi dei dati alcuni campi non hanno importanza per la statistica, è possibile non prenderli in considerazione. Questa operazione corrisponde ad una *proiezione* o *select* in un database relazionale.

Quando effettuare l'unione di due basi di dati distinte

Teniamo sempre in mente che stiamo parlando di una grande mole di dati. Le operazioni per l'unione di due dataset descritti prima sono molto costose computazionalmente ed in termini di spazio: occorrerebbe analizzare tutti i dataset almeno una volta, e creare un nuovo dataset contenente il risultato. Questo non è sempre possibile, per problemi di spazio oppure anche solo per i diritti sulla proprietà dei dati, molto probabilmente appartenenti alle rispettive istituzioni. Nel caso in cui questo non fosse un problema, ed ignorando completamente il problema del mantenere aggiornato il nuovo dataset, l'unico vantaggio sarebbe la velocità di accesso ai record siccome il dataset è nello stesso luogo. Visti i pro e contro, è molto conveniente effettuare tutti i controlli e le conversioni al momento dell'analisi dei dati durante la fase di lettura, oppure come spiegato più tardi, nella fase di Map (Cap. 2.1.2).

Strumenti di memorizzazione

I dataset devono essere capaci di gestire grandi quantità di dati, quindi devono essere molto performanti. Un database è utile nel caso di molti

dati, ma nel caso dei Big Data stiamo parlando di una quantità ancora più grande, quindi anche il migliore dei DBMS potrebbe avere difficoltà ad effettuare le query in tempo utile. Inoltre i database impongono una rigidità sulla struttura dei dati non necessaria secondo la definizione dei Big Data. Un semplice file di testo può adempiere al compito senza causare particolari problemi, a patto che la struttura utilizzata per la memorizzazione sia valido. Esistono moltissimi metodi per salvare dati su file largamente utilizzati, da campi separati da tabulazioni e record separati da "a capo", a più sofisticate come il formato JSON o XML oppure, fogli elettronici in formato CSV o XLS. Poco importa quale formato venga scelto, l'importante è che esista un modo per recuperare dai file le informazioni ed utilizzarle.

Database non relazionali

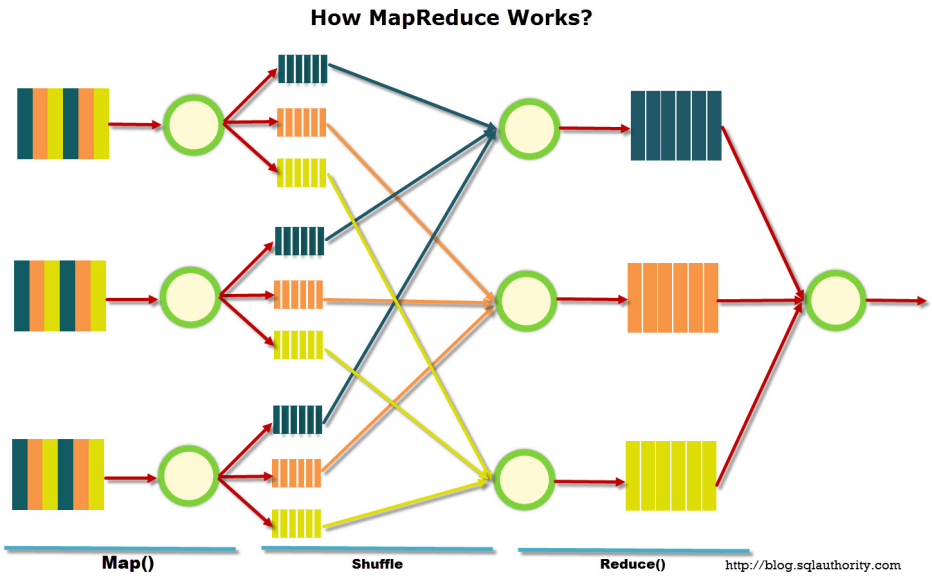
I database non relazionali, anche detti *NoSQL*, sono una tipologia di database nati in seguito allo sviluppo spropositato del web e delle grandi quantità di dati con cui esso ha a che fare. Ben presto gli scienziati si sono accorti che i database relazionali sono molto limitati per quanto riguarda la scalabilità e la parallelizzazione, inoltre sono molto costosi. La necessità di porre rimedio a queste problematiche ha portato alla nascita di nuovi database più flessibili, essi infatti non seguono il livello relazionale, rendendo possibile il salvataggio di dati senza una struttura predefinita. Il fatto che non utilizzino il linguaggio SQL per interrogare il DBMS è una conseguenza del non adottare il modello relazionale, da qui il nome. Per maggiori informazioni, consultare [Vai13]. Nel capitolo 2.3.2 si parlerà di *Mongo DB*, un database non relazionale che utilizza *JavaScript* come linguaggio di interrogazione.

2.1.2 Elaborazione

La parte di elaborazione è quella più costosa computazionalmente, bisogna quindi prestare attenzione alla struttura dei processi che elaborano i dati. Tutto deve essere calibrato per i grandi numeri che comportano i Big Data, prestando attenzione al tipo di calcoli da effettuare per ogni record, perché potrebbe influire moltissimo sul tempo di computazione totale. Un pattern molto utilizzato, introdotto da Google, è il *Map Reduce*. Esso permette di analizzare parallelamente[3] i record di uno o più dataset nella fase di *Map* ed unire i risultati nella fase di *Reduce*. La Figura 2.1 mostra il principio di funzionamento di un programma che applica Map Reduce. I dati vengono prelevati dai rispettivi dataset e suddivisi tra più macchine che effettuano il Map, in cui viene eseguito del codice per ogni singolo record. In questa fase vengono effettuati tutti i controlli descritti nel paragrafo 2.1.1. In output si ottengono zero o più coppie chiave valore. Successivamente se sono presenti più unità di reduce i risultati vengono smistati tra di essi. Ogni unità di

[3] Ovviamente solo se l'hardware supporta il parallelismo

Figura 2.1: Pattern Map Reduce



reduce si occupa di processare le coppie secondo la funzione di reduce, ed effettuano l'effettivo calcolo della statistica desiderata.

2.1.3 Visualizzazione

La parte di visualizzazione è la parte più creativa ed interessante, parte principale in questa pubblicazione. Una volta effettuate le elaborazioni bisogna presentare i risultati, e si possono utilizzare tabelle o grafici. Con grafici si intendono diagrammi a barre, istogrammi, diagrammi a torte eccetera, quello che meglio può presentare ad un lettore il risultato elaborazioni. Una presentazione che permette di capire immediatamente il contenuto dei dati, utilizzando icone e simboli è sicuramente migliore di un'altra che presenta un grafico difficile da interpretare. Inoltre, è compito del data scientist decidere cosa mostrare, scegliendo cosa è interessante e cosa non lo è per raggiungere l'obbiettivo che si è preposto.

Gli strumenti a disposizione degli sviluppatori sono molteplici, da fogli elettronici come Excel o Calc di OpenOffice, programmi matematici come Matlab, fino alle librerie disponibili per i vari linguaggi di programmazione, sicuramente più complesse ma molto più potenti in fatto di personalizzazione.

L'evoluzione del World Wide Web ha permesso la creazione di pagine interattive che supportano animazioni. Nel capitolo 2.4.1 vedremo uno strumento web chiamato D3.js, il quale ci permette non solo di disegnare grafici in una pagina web, ma di gestire molti componenti grafici come rettangoli, cerchi, poligoni, *mappe* e le loro animazioni.

2.2 Pattern MVC

> Un design pattern descrive un problema che si ripete più e più volte nel nostro ambiente, descrive poi il nucleo della soluzione del problema, in modo tale che si possa usare la soluzione un milione di volte, senza mai applicarla alla stessa maniera

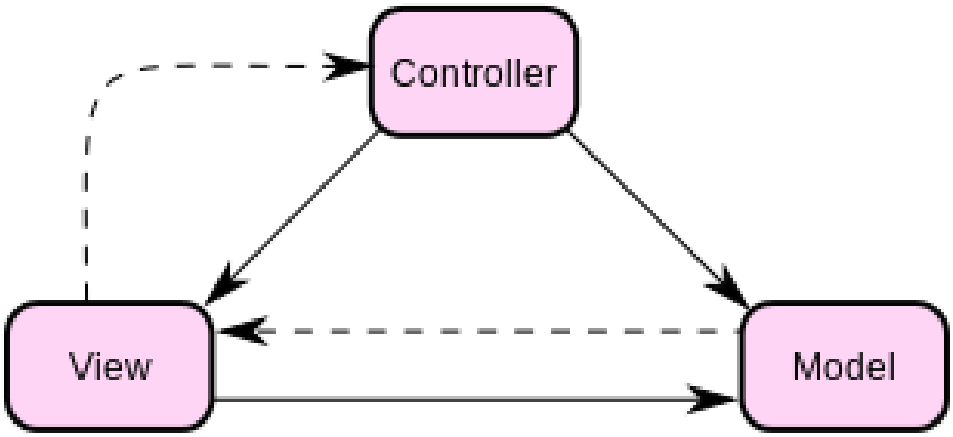
Questa definizione di *Design Pattern* viene data da [Gam02, p. 2,3], l'autore cita a sua volta la definizione di pattern architettonici, mostrando la similitudine con quelli informatici.

Il pattern MVC si occupa di definire una struttura per il software che divide e rende autonome le parti di visualizzazione, elaborazione e memorizzazione dei dati (Fig. 2.2).

Questa struttura offre notevoli vantaggi in termini di pulizia, estensione e mantenimento del codice. Secondo [Gam02, p. 4]

> MVC consta di tre tipologie di oggetti. Il Model è l'oggetto dell'applicazione, il View è la sua rappresentazione a video, e il Controller definisce le reazioni dell'interfaccia grafica all'input dell'utente.

Figura 2.2: Pattern MVC



Il programma sarà più pulito siccome in ogni sorgente sarà presente solamente il codice che si occupa di uno specifico compito. Non si presenterà quindi una situazione in cui nella stessa parte di codice oltre ad elaborare i dati ci dobbiamo preoccupare di formattarli per la presentazione su schermo, perché sarà compito della *View*. L'aggiunta do nuove funzioni al software è molto semplice, per effettuare delle modifiche è spesso necessario aggiungere poche righe di codice in ogni parte del programma, senza il bisogno di

stravolgerne la struttura. Il mantenimento del software risulta più semplice grazie alla suddivisione del codice in base al ruolo ed alla conseguente leggibilità.

2.2.1 Model

Il model, o *modello* è la parte che si occupa della memorizzazione dei dati dell'applicazione, fornendo delle funzioni per salvare, leggere e modificare i dati dalla base dei dati. Le funzioni fornite sono l'*interfaccia* attraverso cui il resto del programma può interagire con i dati dell'applicazione, non è possibile farlo direttamente sul database oppure in altri modi: rappresenta quindi uno strato al di sopra della base di dati, che ha il compito di fornire metodi ad alto livello per l'accesso ai dati, e di proteggere gli stessi da errori di programmazione comportati dall'accesso diretto.

2.2.2 Controller

Questa sezione si occupa di interfacciarsi con le altre due. Un controller in genere riceve le richieste dell'utente sotto forma di chiamate a funzioni e si occupa di eseguirle, preoccupandosi della validazione e della coerenza dei dati. Esso può essere un tramite tra *View* e *Model*, oppure operare solo con uno di loro:

- Aggiunta di nuovi dati o modifica di dati esistenti

  Quando l'utente inserisce alcuni dati in un form e desidera che vengano memorizzati sul database, la View incapsula i dati inseriti in opportune strutture dati che verranno passate al controller, il quale effettuerà i controlli necessari per verificare che i dati non contengano errori, assicurarsi che non confliggano con quelli memorizzati, per poi inoltrare una richiesta di memorizzazione al Model, il quale eseguirà in modo del tutto trasparente al Controller l'operazione. Infine, il Controller invierà una conferma o l'errore alla View, in modo di informare l'utente su cosa ha fatto il sistema.

- Visualizzazione di dati

  In questo caso, la view invia una richiesta contenente le informazioni riguardanti a quali dati l'utente desidera visualizzare al Controller, quest'ultimo si occupa di verificare se l'utente è autorizzato ad accedere a quei dati restituendo errore nel caso in cui non sia autorizzato, oppure, se il Model supporta un sistema di gestione dei permessi ad utenti, inoltra la richiesta al model aggiungendo le informazioni sull'identità dell'utente.Il Model a questo punto provvederà a recuperare i dati e restituirli al Controller, effettuare alcune operazioni su di loro se necessario, per poi inviarle alla View.

I Controller sono quindi quelle parti del programma che si trovano tra l'interfaccia utente e la base di dati, che si occupano di coordinare le altre due componenti. Il loro compito è eseguire le richieste degli utenti, utilizzando le interfacce fornite da View e Model per portarle a termine, ma preoccupandosi anche dei controlli sui dati che transitano in essi.

2.2.3 View

La View è la parte che si occupa della *visualizzazione* dei dati forniti dal controller. Essa utilizza nel sorgente dei *placeholder* che verranno sostituiti a runtime dai dati. Un esempio in campo web, potrebbe essere una semplice pagina che saluta il cliente e visualizza un elenco di prodotti in vendita:

```
#View
Ciao {{nome}}!
Oggi in vendita abbiamo: {{for frutto in frutti}}
```

Le parti tra le doppie parentesi graffe[4] sono i placeholder, i quali possono contenere anche semplici istruzioni di ciclo o di confronto. Nel momento dell'esecuzione queste verranno sostituite con i dati indicati. Questo è necessario perché al momento della scrittura del programma le variabili "nome" e "frutti" non sono ancora stati calcolati! Se i dati forniti dal controller sono questi:

```
#Controller
nome="Andrea"
frutti=['pere', 'mele', 'arance']
```

Il risultato dell'esecuzione sarà quindi:

```
Ciao Andrea!
Oggi in vendita abbiamo: pere mele arance
```

2.3 Software lato server

Come ogni sito web, per visualizzare una pagina sono necessari un browser internet ed un server che risponda alle richieste HTTP. Il browser si connetterà al server e quest'ultimo invierà la pagina, infine il browser si occuperà di visualizzarla interpretando il codice HTML ed eseguendo gli script lato client in esso contenuto.

[4] La sintassi di questo esempio è ispirata ad un *Template Engine* chiamato Mustache (http://mustache.github.io)

2.3.1 Turbogears

Turbo Gears 2 è un framework web scritto in *Python*[5], esso permette la creazione di siti web in modo veloce e semplice applicando il pattern MVC. Fornisce la possibilità di scegliere quale Model utilizzare, fornendo una scelta fra database relazionali e non. Anche per gli altri componenti è possibile effettuare delle scelte, ma per lo sviluppo di questo progetto sono stati utilizzati quelli di default. Turbo Gears può anche essere utilizzato come server web, siccome offre un modo molto semplice per avviarne uno. TG analizza le richieste ricavando informazioni dall'url richiesto dal client. Un url come quello del listato 2.1 significa:

```
1 http://www.tg2site.com/controller/function
```

Listing 2.1: Url per la richiesta di una pagina

http://www.tg2site.com Nome dell'host

controller Indica quale controller utilizzare. Questo può essere omesso, in tal caso verrà utilizzato il *root controller*.

function Indica quale funzione del controller si desidera richiamare.

In TG la *view* è implementata utilizzando un linguaggio di markup simile all'html con l'aggiunta di *marker* o *segnaposto* che permettono di utilizzare un meccanismo simile a quello mostrato nel paragrafo 2.2.3. Il modulo che si occupa della sostituzione del segnaposto con le informazioni si chiama Genshi.

Il *model* invece è implementato utilizzando una libreria chiamata *Ming*[6], la quale permette di definire delle classi di oggetti che avranno la caratteristica di rimanere sincronizzate con Mongo DB. Esso appartiene a quella classe di software che viene definito un *ORM* (Object Relational Mapping). Tutti gli ORM forniscono la possibilità di associare una classe di un programma con una tabella del database, assicurando la *consistenza* nei dati che si raggiunge con tecniche che permettono di creare un oggetto a partire da un record di un database, modificarlo e di occuparsi di memorizzare le modifiche, oppure creare direttamente un nuovo record. Il tutto avviene senza scrivere una riga del linguaggio di interrogazione del database, ci pensa ORM.

A questo punto è lecito chiedersi perché la libertà offerta dalla natura *NOSQL* di Mongo, ossia la mancanza della necessità di avere una struttura dati ben definita, viene eliminata utilizzando una libreria come Ming. Le

[5] Tutte le informazioni possono essere trovate sul sito ufficiale: http://turbogears.org/.

[6] merciless.sourceforge.net/tour.html

ragioni dietro a questa scelta sono che questa proprietà di mongo è molto comoda nel momento dello sviluppo, durante il quale il database viene rigenerato più volte al giorno, ma diventa un problema quando si ha bisogno di una certa garanzia della tipologia di dati memorizzati in una certa *collezione*[7], inoltre l'utilizzo di un ORM aumenta la portabilità del progetto.

Ming è un'estensione adottata da TG per permettere l'utilizzo di Mongo, mentre SQLAlchemy è quello utilizzato per la controparte SQL.

La struttura tipo di un progetto TurboGears 2 è rappresentata nella figura 2.3.

A seguire verranno descritte alcune delle cartelle più importanti.

model contiene le classi necessarie a Ming per definire una struttura per i dati da salvare sul database.

controllers contiene i controller. `root.py` è il *root controller*, `error.py`, creato da TG, si occupa della gestione degli errori.

templates contiene le pagine html del progetto.

public contiene tutte le risorse accessibili via web, come immagini, librerie JavaScript e fogli di stile.

config e lib contengono impostazioni e funzioni utili, utilizzate da TG e liberamente utilizzabili dal programmatore.

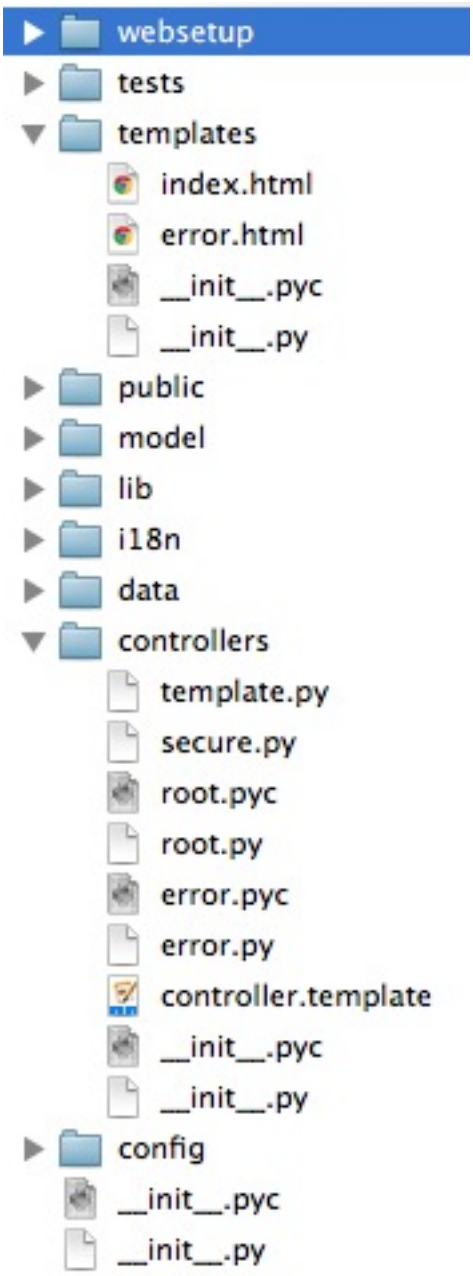
Il flusso di lavoro di TG potrebbe essere riassunto in:

1. Analizza l'url
2. Richiama l'opportuna funzione dell'opportuno controller
3. La funzione potrebbe interfacciarsi con il database, memorizzare o estrarre dati.

   In essa è sempre definito un *decoratore* riconoscibile con la direttiva `expose`, con cui si indica alla View quale templare utilizzare. La funzione di un controller viene richiamata nel caso in cui TG riceve una richiesta del tipo `hostname/func/param1/param2/...`, i valori `param` sono i parametri della funzione, i dati da passare a Genshi devono essere memorizzati in un oggetto `dictionary` di python e restituiti dalla funzione.
4. La view sostituisce i marker con le relative informazioni, infine genera la pagina
5. La pagina generata viene inviata al client che ha effettuato la richiesta.

[7] L'equivalente di una tabella, in Mongo, come verrà spiegato nel paragrafo 2.3.2.

Figura 2.3: Struttura di un progetto TG2.



2.3.2 Mongo DB

Come spiegato nel capitolo 2.1.1, *Mongo DB* è il DBMS utilizzato per il progetto, in cui vengono memorizzati i dati utilizzati dall'applicazione. Il sito ufficiale è `www.mongodb.org`, in cui oltre al programma si può trovare il

manuale ed un tutorial interattivo.

Mongo come linguaggio di interrogazione non utilizza SQL a causa della sua natura NOSQL, ma utilizza JavaScript. Importantissimi sono il concetto di *documento* e *collezione*, in quando a differenza del modello relazionale, in Mongo non esistono tabelle e relazioni, ma collezioni di documenti. Molto facilmente si potrebbe dire che una collezione è l'equivalente di una tabella nel modello relazionale ed un documento ad un record memorizzato in una tabella, ma le differenze si estendono alle chiavi primarie ed alle relazioni, siccome esse non esistono. Il tutto è molto semplificato rispetto al modello relazionale: per identificare univocamente ogni record si utilizza un campo *id* come descritto più avanti, e per riferirsi a documenti di altre collezioni si memorizza un puntatore al documento stesso. Inoltre, ogni documento è un oggetto JavaScript, che può contenere come attributi liste e riferimenti ad altri oggetti. Nel database vengono memorizzati gli oggetti JavaScript tramite una funzione `save()`, ed è possibile estrarli utilizzando la funzione `find()`. Nel momento della creazione di un nuovo documento, Mongo aggiunge un campo `_id` contenente un oggetto che rappresenta un identificativo univoco nella collezione, equivalente ad una chiave primaria di una tabella. Per riferirsi ad una collezione si utilizza l'oggetto `database.collezione` ossia, la collezione `collezione` è un attributo dell'oggetto `database`. Nell'oggetto collezione sono definiti i metodi per il salvataggio, la ricerca e l'eliminazione dei documenti prima citati. In particolare, per la funzione di ricerca, il parametro passato è un oggetto JavaScript contenente i parametri di ricerca, i quali valgono per i campi del documento: si può definire una ricerca per valori *uguali*, *maggiori*, *minori* o *diversi* da valori specificati.

Il valore di ritorno è un iteratore, il quale è un oggetto che applica il pattern *iterator*, il quale, brevemente, permette di definire un'interfaccia per scorrere una struttura dati, in modo da renderla virtualmente una lista di elementi accessibili sequenzialmente. In questo modo ad esempio, se si vuole percorrere un albero, una funzione *successivo* di un iteratore che visita l'albero in *profondità* restituirà la radice, il figlio della radice, poi il figlio del figlio della radice. Nel caso si voglia visitare l'albero in ampiezza, basterà istanziare un iteratore di quel tipo e si otterrà l'effetto desiderato, il tutto senza cambiare il codice della funzione che riceve gli elementi. Questo oggetto fornisce il metodo `next()`, che restituisce il documento successivo nella lista, ed offre la comodità di poter essere utilizzato in un ciclo `for` perché supportato nativamente da JavaScript.

2.4 Librerie

2.4.1 d3.js

D3.js è una libreria JavaScript che offre strumenti molto potenti per la visualizzazione di dati, il sito ufficiale è raggiungibile alla pagina http:

//d3js.org/. La descrizione reperibile da esso, spiega che

> D3.js è un libreria JavaScript per la manipolazione di documenti basati sui dati. D3 fornisce gli strumenti per rappresentare i dati utilizzando HTML, SVG e CSS. D3 si focalizza sugli standard web offrendo tutta la potenza dei browser attuali senza essere dipendenti da un framework proprietario, combinando strumenti di visualizzazione con un approccio *data-driven* per la manipolazione del DOM.

In pratica D3 permette di modificare il DOM di una pagina web in base ai dati collegati all'applicazione, utilizzando i tag *SVG* di *HTML*, acronimo di *Scalable Vector Graphics*. *Protovis*, realizzato dal professor Jeff Heer, Vadim Ogievetsky e Mike Bostock della Stanford University, era una libreria JavaScript capace di creare grafica SVG a partire dai dati. Nel 2011 il suo sviluppo venne interrotto per spostare l'attenzione al suo successore, D3.js, creato dagli stessi tre sviluppatori ed in particolar modo da Mike Bostock, che oltre ad essere più performante si attiene ai più recenti standard web.[8].

Gli standard web sono una serie di regole e *Best Practices* non proprietarie, che definiscono e descrivono gli aspetti del *World Wide Web*. Uno degli enti che si occupa degli standard web è il World Wide Web Consortium o *W3C*, che descrive gli elementi di HTML, CSS, JavaScript e non solo, fornendo spiegazioni esaustive di cono siano, come funzionano ed anche di come si usano. Il sito di W3C è http://www.w3.org/.

Gli strumenti che offre D3.js comprendono:

- Metodi di selezione

  Il framework rispetta le *W3C Selectors API*, uno standard alternativo alle API di manipolazione del DOM tradizionali. Con queste è possibile selezionare uno o più elementi nella pagina utilizzando diversi criteri, come il nome del *tag*, l'*id* o se un attributo è uguale ad un certo valore, ottenendo un array contenente tutti gli elementi conformi al criterio di selezione.

- Binding dei dati

  Dato un array in JavaScript, D3 permette di associarlo ad una funzione che si occuperà di eseguire del codice per ogni singolo elemento, con la possibilità di specificare cosa fare nel caso in cui questo array cambi, fornendo una funzione che viene richiamata quando degli elementi vengono aggiunti o eliminati. Quindi, dato un array contenente degli oggetti che rappresentano dei locali, lo si può associare ad un elemento Mappa che si occuperà di visualizzare in un canvas SVG un puntino per ogni locale nella giusta posizione su di una mappa.

[8] Fonte: http://en.wikipedia.org/wiki/D3.js#Context

- Transizioni

  Questa parte si occupa di fornire le primitive per animare le parti di una pagina. Definendo il tempo della transizione e lo stato finale di alcuni attributi, al momento dell'esecuzione del codice D3 si occupa di modificare in modo graduale gli attributi specificati fino da diventare come definiti nello stato finale. In questo modo si può modificare la posizione di un oggetto del DOM in modo che se si verifica un preciso evento, gradualmente si sposti verso l'esterno della pagina fino a scomparire, oppure per cambiarne il colore in modo graduale.

D3 permette lo sviluppo di grafici molto vari: diagrammi a torte, grafi e mappe interattive, il solo limite è la fantasia del programmatore. Sul sito principale si possono vedere molti esempi interessanti.

2.4.2 Polymaps

Polymaps (http://polymaps.org/), scritta dallo stesso sviluppatore di D3.js Mike Bostock, è una libreria JavaScript che permette la creazione di mappe dinamiche ed è basata su D3. Offre numerose funzioni per la manipolazione di coordinate geospaziali, di mappe vettoriali o di mappe basate su immagini. Queste ultime possono essere ottenute tramite le funzioni fornite dalla libreria di siti come *OpenStreetMaps* o *Bing*.

Capitolo 3

Analisi del progetto

3.1 Descrizione

Piedmont Heritage è il progetto finale sviluppato durante il corso *Big Dive 2*[1] dal sottoscritto insieme con *Maximilien Rzepka* e *Marco Sors*. È possibile visualizzare il progetto online all'indirizzo http://piedmontheritage.bigdive.eu, la cui versione è attualmente quella precedente alle modifiche descritte in questo testo.

Il sito mostra una serie di "slide" in cui vengono visualizzate informazioni su locali ed oggetti storici presenti in Piemonte. Partendo dall'alto è presente una mappa che visualizza l'ubicazione dei locali ed a seguire una serie di infografiche rappresentanti le categorie dei locali presenti, il periodo in cui sono nati, ed infine un grafico sulle tipologie di oggetti, il inframmezzato da citazioni e detti popolari inerenti a Torino ed al Piemonte.

3.2 Contesto in cui il progetto è stato realizzato

Piedmont Heritage è stato sviluppato durante il corso di Big Dive, come prova finale in cui si sarebbero utilizzati tutti i concetti e gli strumenti presentati durante il corso. Dal sito del corso,

> BIG DIVE è un programma di formazione volto a promuovere una nuova generazione di sviluppatori. Una palestra combattimento da strada, dove dataset di alto valore sono la materia prima nelle mani di un gruppo di appassionati seguiti da esperti in tre aree chiave: Development, Data Science and Visualization.

La nascita di questo progetto è stata dettata dallo spirito del corso: scegliere dei compagni di lavoro, scegliere un dataset tra quelli messi a disposizione, capire che cosa offre il dataset ed elaborare dei dati da esso, utilizzando la creatività del gruppo.

[1] Il sito di Big Dive è http://www.bigdive.eu

Durante il corso sono stati affrontati argomenti inerenti ai Big Data, dal calcolo di statistiche basate sulle basi di dati alla visualizzazione di quest'ultime in modo *bello*, *immediato* ed *esaustivo*. Sarà quest'ultima parte l'argomento di questo testo, dove si affronterà il problema di *come visualizzare i dati su una mappa*.

3.3 Il dataset

Il dataset utilizzato è il **Guarini Dataset**, il quale è un progetto della Regione Piemonte nato ai fini di

> migliorare la definizione dei programmi di intervento, salvaguardia e valorizzazione.[2]

Il Sistema Informativo Guarini è il software fornito dalla Regione Piemonte utilizzato per il censimento sancito dall'art. 2 della L.R. 34/95[3] e L.R 35/95[4]. I locali storici presenti nel dataset sono selezionati dai consigli comunali appartenenti al luogo in cui si trovano, e vengono censiti solo se meritevoli. L'aggiunta dei dati al dataset è caratterizzata da una grande libertà nel tipo di informazioni inseribili, inoltre ogni record è inserito da persone diverse, con la conseguenza di una frammentazione dei dati presenti nel database, con campi non compilati e campi ad-hoc per alcune entry.

Una nota a proposito del database

Anche se il corso tratta di *Big Data* il dataset è molto ristretto, conta infatti circa 1200 entry, che dopo essere state selezionate e corrette sono scese a circa 900. Alla fine del corso per questo motivo il progetto non era ottimizzato nel caso in cui il numero di entry fosse aumentato, cosa a cui si è provveduto successivamente a scopo didattico. Inoltre, con un numero così basso di elementi non è stato necessario utilizzare i metodi studiati per l'elaborazione di grandi moli di dati visti nel corso, per questo motivo nei prossimi capitoli se ne accennerà solamente.

3.4 La mappa

La parte che verrà analizzata è quella della prima slide, la *mappa dei locali*. Durante lo sviluppo l'obbiettivo è stato trovare ed utilizzare uno strumento che permettesse di *visualizzare dei segnalini* su di una mappa in corrispondenza dei locali memorizzati nel dataset, *filtrare* quali risultati visualizzare

[2] Tratto dal sito ufficiale della Regione: http://www.regione.piemonte.it/cultura/guarinipat/index.htm

[3] http://arianna.consiglioregionale.piemonte.it/base/leggi/l1995034.html

[4] http://arianna.consiglioregionale.piemonte.it/base/leggi/l1995035.html

e mostrare i *dettagli del locale* il cui segnalino è stato cliccato. Per farlo sono stati utilizzati alcuni strumenti presentati nel corso, mentre altri sono stati scelti liberamente.

3.4.1 Miglioramenti dopo Big Dive

Durante l'A.A. successivo a quello dello stage, sono stati effettuati dei miglioramenti alla mappa della prima slide di Piedmont Heritage da parte dell'autore, permettendo una maggiore libertà per il filtraggio dei risultati, migliorando la navigazione tra di essi, implementando un sistema per ottenere dati dal server in modo asincrono con il caricamento della pagina e risolvendo un bug. Il sito inoltre è ora servito tramite un framework presentato durante lo stage: queste modifiche sono state effettuate per rendere il progetto utilizzabile per basi di dati più grosse, come descritto nei prossimi capitoli, ma anche a scopo didattico in modo da utilizzare altri strumenti presentati nel corso.

Capitolo 4

Architettura del sistema e sue funzionalità

Le problematiche affrontate, tenendo conto delle richieste dei fornitori del dataset, sono state quelle di mostrare i dati nel modo più esplicativo possibile senza fornire la possibilità di accedere direttamente al dataset, utilizzando solo l'interfaccia front-end sviluppata. Il programma carica una sola volta la pagina web ed effettua delle richieste asincrone al server per per interrogare il database, ottenendo risultati tali da contenere solo i dati necessari alla visualizzazione corretta sulla mappa dei punti di interesse.

Altri obbiettivi da raggiungere sono stati:

- Rendere possibile una crescita del progetto in futuro

  Il progetto è stato strutturato in modo da rendere possibile in futuro l'aggiunta di funzioni ed aggiungere nuovi locali, tenendo ben presente che i pochi record attuali sono molto semplici da gestire, mentre se dovessero aumentare di molto il programma è comunque capace di gestire i nuovi record senza appesantirsi troppo. L'utilizzo di framework web, oltre a velocizzare enormemente lo sviluppo, offre di strutturare il progetto modularmente e permettendo così di estenderlo in futuro se sono necessarie altre features.

- Utilizzare un database

  Siccome il dataset iniziale era un semplice file CSV che necessitava di molte correzioni, l'utilizzo di un database non relazionale come Mongo DB (Capitolo 2.3.2 a pag. 18) avrebbe semplificato di molto la gestione dei record, permettendo di effettuare con la programmazione una più semplice pulizia dei dati. La natura del database permette di memorizzare i record anche se alcuni non contengono informazione in tutti i campi, oppure se alcuni dei campi non sono presenti. Tramite l'utilizzo di funzioni scritte appositamente per interrogare il DB

ed ottenere parti precise di dati, si risolve il problema della non distribuzione del dataset. La struttura del progetto è quella standard di Turbogears ed è stata discussa nel paragrafo 2.3.1 ed in particolare si faccia riferimento all'immagine 2.3.

Nel progetto i linguaggi di programmazione usati prevalentemente sono stati Python per la parte lato server e Javascript lato client. Non mancano ovviamente i linguaggi web come HTML e CSS.

4.1 Lato client

Una volta scaricata la pagina, al posto della mappa sarà presente un tag `div` con id="map" vuoto, che sarà completato dalla chiamata a funzione `drawMap()` a fondo pagina[1] che si occuperà di creare le strutture dati necessarie a mostrarla. `drawMap` è una funzione javascript definita nel file `public/js/map.js` che esegue queste operazioni:

1. seleziona nel DOM il tag `div` prima citato
2. crea l'oggetto della mappa, settando la posizione nella pagina usando la variabile del punto 1, le coordinate geografiche a cui puntare e lo zoom.
3. assegna all'oggetto mappa la sorgente delle immagini raffiguranti la mappa vera e propria.
4. aggiungono i pulsanti per regolare lo zoom, per navigare nei risultati, ed il riquadro che visualizzerà i dettagli dei locali.
5. aggiunge l'oggetto mappa all'interno del tag `div` con id="map", visualizzandolo effettivamente sulla mappa.

La mappa è composta da immagini prelevate dinamicamente da CloudMade[2]. Abbiamo scelto quest'ultimo perché è possibile applicare un tema alla mappa che rispecchiava quello del sito.

I locali però non sono ancora stati caricati a questo punto, ma nemmeno le città ed i tipi selezionabili. La funzione `json()` di D3 ha la capacità di reperire dati in formato JSON dall'url passato come parametro. Il JSON, acronimo di JavaScript Object Notation, è un formato adatto per lo scambio dei dati in applicazioni client-server[3] e consiste nel linguaggio utilizzato per definire oggetti in javascript. Tramite due chiamate alla funzione `json()`

[1] Una buona tecnica per evitare che il caricamento della pagina rimanga bloccato a causa del download degli script, è quello di spostare i tag `script` alla fine del documento.

[2] http://cloudmade.com/products/web-maps-api

[3] cit. Wikipedia http://it.wikipedia.org/wiki/JSON

si reperiscono dal server l'elenco delle città e dei tipi, successivamente si inserisce ogni opzione nelle liste sopra la mappa.

Infine, si devono visualizzare i segnalini dei locali sulla mappa. La funzione `updateViz()` si occupa di controllare quali opzioni sono selezionate nei menu a tendina delle città e dei tipi, ed effettuare una richiesta al server, tramite la funzione `json()` di D3. I risultati ottenuti rappresentano alla prima chiamata tutti i locali presenti nel database, ed è necessario creare un puntino blu per ognuno di essi sulla mappa.

D3 permette di definire funzioni che operano su liste, ed è in grado, con la creazione una nuova lista, di mappare gli elementi già presenti con quella vecchi, permettendo quindi di rilevare gli elementi nuovi e quelli non più presenti. Come mostrato nel listato 4.1 D3 dispone di funzioni chiamate `enter` ed `exit` che rispettivamente definiscono il codice da eseguire per gli elementi nuovi e quelli non più presenti. Per ogni elemento nuovo si visualizzerà un pallino creando un tag `circle`, dettando dimensione, colore e cosa deve accadere nel caso in cui si faccia click su di esso. Le operazioni da applicare agli elementi uscenti dicono al programma di avviare un'animazione che porta la trasparenza a zero, facendoli svanire, per poi eliminare i tag `circle` dal DOM. Questo sistema di *enter* ed `exit` è un'alternativa a scrivere un ciclo e permette di scrivere direttamente cosa deve fare D3 con ogni elemento.

```
1  d3.json('points.json?city=' + settings.city + '&type=' +
       settings.type, function(points){
2      selection.setData(points['data'])
3      showDefaultDetails()
4      // Add an svg:g for each station.
5      var marker = layer.selectAll(".gcircle")
6          .data(dat, function(d){
7              return d._id
8              })
9
10     marker.enter()
11         .append("g")
12         .on("click", selection.selectPoint)
13         .attr('class', 'gcircle')
14         .append("circle")
15             .attr('class','circle')
16             .attr("r", 3);
17
18     marker.exit()
19         .transition()
20         .style('opacity', 0)
21         .duration(1000)
22         .remove()
```

Listing 4.1: `enter()` ed `exit()` di d3.js

Viene poi mossa la mappa in modo che il punto (x, y) sia al centro, ricavato dalla funzione 4.1 che calcola il punto medio tra i valori maggiore e minore di latitudine e longitudine. Infine si regola il focus in modo che tutti i punti siano visibili.

$$(x, y) = \left(\frac{max(lat) + min(lat)}{2}, \frac{max(lon) + min(lon)}{2} \right) \tag{4.1}$$

A questo punto il programma aspetta input da parte dell'utente, e la mappa è interattiva. Grazie all'evento `move` polymaps rivela i drag del mouse sulla mappa, con il risultato di spostare la mappa ed i marker nella direzione dello spostamento del mouse, i tasti `+` e `-` in alto a sinistra permettono di modificare lo zoom, e cliccando su un marker verrà selezionato. Quando un locale rappresentato da un punto viene selezionato, viene richiamata la funzione `selection` e mostrata un'animazione in cui la vecchia selezione torna un normale punto mentre quello nuovo cambia colore e si ingrandisce. D3 invia poi una richiesta al server contenente l'ID del locale e riceve le informazioni riguardanti il *nome*, *tipo*, *periodo storico*, *indirizzo* e *comune*, e li visualizza nel box in alto a destra.

Tramite un url che permette di ottenere un'immagine tratta da Google Street View che mostra una visuale di un certo indirizzo, viene mostrato il luogo che secondo Google Maps corrisponde all'indirizzo memorizzato. Questa feature è stata inserita sperimentalmente, ma non offre una panoramica soddisfacente del posto e non tutti gli indirizzi hanno un'immagine associata.

È possibile selezionare i locali uno per uno, utilizzando i pulsanti di navigazione in alto al centro. La pressione di quei pulsanti non farà altro che aumentare o diminuire l'iterator che punta all'oggetto selezionato nella lista dei risultati e richiamare la funzione `selection` su di esso. Il focus della mappa si sposta in modo da puntare il marker selezionato, e farlo apparire al centro della mappa.

L'ultima funzionalità è quella della ricerca per comune e/o tipo. Selezionando un elemento nel menu `select` dei comuni la lista dei tipi viene modificata mostrando solo i tipi disponibili in quel comune, sempre grazie a d3 ed una funzione scritta nel server. Premendo il tasto *filtra*, verrà richiesto al server l'elenco di locali che soddisfano i criteri selezionati, e si affideranno i risultati a d3, il quale provvederà a visualizzare i nuovi punti ed eliminando quelli non più presenti come descritto in precedenza.

4.2 Lato server

Solitamente quando un server web riceve una richiesta in cui la pagina non è specificata, controlla l'esistenza di un file chiamato *index* e lo seleziona come risposta. Nel caso di Piedmont Heritage non è una pagina, ma una funzione chiamata `index`, riportata nel listato 4.2. Il decoratore specificato

indica il file `index.html` nella cartella `templates`. La natura ad oggetti del framework permette di considerare un oggetto anche una pagina web, in questo caso è stato utilizzato l'oggetto che lo rappresenta.

```
@expose('pheritage.templates.index')
def index(self):
    """Handle the front-page."""
    return dict(page='index')
```

Listing 4.2: root controller, funzione index, richiamata alla richiesta della pagina web da parte del client.

Si possono ottenere parametri dal client in diversi modi[4], il progetto li inserisce in coda all'url dopo il nome della funzione (vedi paragrafo 2.3.1). Nel listato 4.2 non sono presenti parametri, quindi non ci si aspetta valori passati dalla richiesta HTTP. Nella funzione 4.4 a pagina 30 ci si aspetta un parametro `id`, necessario per individuare il locale nel database e richiederne le informazioni.

Ming come già detto si occupa delle operazioni su database. Esso prevede due modalità, una più a basso livello, l'altra più ad alto livello con maggiore astrazione utilizzata dal progetto, chiamata *ODM*. Esso fornisce alcune funzionalità in più dell'altra versione, tra cui un sistema che ne offre di simili alle transazioni di SQL, e la comodità di operare sugli oggetti veri e propri nel caso di campi che puntano ad altri documenti (chiavi esterne)[5]. Richiede la definizione delle classi che rappresenteranno il model. Esse dovranno estendere la `MappedClass` di Ming, definire una sottoclasse contenente le informazioni necessarie al framework e definire i campi della classe, che saranno effettivamente i campi che verranno memorizzati nel database (vedi esempio nel listato 4.3).

```
class Categoria(MappedClass):
    class __mongometa__:
        session = DBSession
        name = 'categorie'

    _id          = FieldProperty(s.ObjectId)
    nome      = FieldProperty(s.String)
    cont        = FieldProperty(s.Int)
```

Listing 4.3: Classe della collezione di categorie, con i campi da salvare sul database.

La definizione di classi del model permette anche di specificare campi con riferimenti ad altri documenti, anche in altre collezioni, ma in questo progetto non sono stati utilizzati.

[4] In modo simile a quanto descritto in seguito, è possibile ottenere i parametri inviati tramite i metodi *GET* e *POST*.

[5] Maggiori dettagli su http://merciless.sourceforge.net/odm.html

Per il salvataggio e la ricerca su database è necessario introdurre il concetto di *sessione* di Ming: consiste nel collegamento al database, ma fornisce anche i metodi per interagire con esso. La sessione di Ming è l'oggetto responsabile dell'interazione con Mongo, oltre ad occuparsi di generare le query da passare al database, fornisce un ambiente virtuale in cui eseguire le operazioni. Ogni operazione viene memorizzata in una coda di operazioni, ed appare al programma come se fosse effettiva sul database, se per qualche problema la serie di operazioni non avesse successo, è possibile scartarle senza alcuna ripercussione, altrimenti, alla chiamata del metodo `flush()`, tutte le operazioni verranno eseguite sequenzialmente sul database. Mongo non supporta le transazioni come SQL, questa funzionalità cerca di ricalcare alcuni benefici da loro offerti.

Per creare nuovi documenti occorre istanziare la classe prima definita ed effettuare una chiamata al metodo `flush()` della sessione. I dati dell'applicazione inizialmente risiedono in un file json[6] contenente tutti i dati dei Locali, che vengono inseriti nel database durante la procedura di installazione del programma durante il *bootstrap*, una funzione che effettua le operazioni preliminari prima del primo avvio del programma, definita nel file `bootstrap.py`. Essa analizza il file e carica i locali uno per uno.

L'interrogazione del database avviene con il metodo `find()` dell'oggetto sessione: accetta due parametri, con il primo si può specificare tramite un oggetto `dict` di python una query, in modo simile a quelle da effettuare sul client di Mongo in Javascript, permettendo l'equivalente di una *selezione* nel calcolo relazionale. Il secondo permette di specificare quali campi ottenere, se omesso vengono restituiti tutti. L'equivalente in calcolo relazionale di una *proiezione*.

In Piedmont Heritage l'interrogazione del database avviene, prendendo in considerazione il listato 4.4, utilizzando l'oggetto `c_locale`. Esso è l'equivalente dello scrivere `model.mainsession.db.locali`, ed è l'oggetto che fornisce il metodo `find()` per la collezione *locali*

Successivamente verranno descritte le funzioni del root controller ed il loro funzionamento. Il listato completo del root controller, presente nel file textttcontrollers/root.py è a pagina 31.

```
1 @expose('json')
2 def detail(self, id):
3     det = c_locale.find({'_id': ObjectId(id)})
4     return dict(data = list(det))
```

Listing 4.4: Funzione utilizzata per richiedere i dettagli su un locale con ID=`id`

La funzione del Listato 4.4 è utilizzata dalla funzione richiamata dal client quando viene selezionato un marker. Il decoratore specificato (il parametro

[6] Frutto del lavoro di pulizia dei colleghi di progetto, nda.

della direttiva `@expose()`) è JSON, la funzione restituirà quindi il valore di ritorno creando un oggetto in formato JSON equivalente. L'url a cui il client riceve i dati è la seguente:

```
http://piedmontheritage.com/detail/527
    ac93d3573d7707e1a2f27.json
```

`detail` è il nome della funzione, mentre `527ac93d3573d7707e1a2f27` è l'id del locale. La risposta del server, in JSON, è quella del listato 4.5.

```
{"data": [
        {"comune": "ACQUI TERME",
         "indirizzo": "Via G. Garibaldi 73",
         "datazione": "1930",
         "tipo": "Merceria",
         "images": [],
         "lon": 8.4677381,
         "denom": "Merceria Voglino",
         "lat": 44.6762883,
         "denomlocale": "",
         "restauro": "1980",
         "_id": "527ac93d3573d7707e1a2f27",
         "id": "R0335027","
         ": "AL"}
          ]
}
```

Listing 4.5: Dati in formato JSON restituiti dal server.

Le altre funzioni del controller funzionano in modo analogo:

- La funzione `categories()` si occupa di estrarre dal database l'elenco di tutte delle categorie. Questa funzione è usata quando il client richiede tutte le categorie senza filtrarle per comune.
- La funzione `cities()` effettua una ricerca ricavando l'elenco dei comuni, eliminando le voci duplicate ed ordinadole in redine alfabetico.
- La funzione `types(comune)` si occupa di verificare se il valore di `comune` è *tutti i comuni*, in tal caso estrae l'elenco di tutti i tipi, mentre se è selezionato un comune estrae solo quelli presenti in quel comune.
- La funzione `points(comune, tipo)` interroga il database ottenendo l'elenco di locali che soddisfano i termini di ricerca definiti da `comune` e `tipo`.

```
# -*- coding: utf-8 -*-
"""Main Controller"""

from tg import expose, flash, require, url, lurl, request
    , redirect, tmpl_context
```

```
5  from tg.i18n import ugettext as _, lazy_ugettext as l_
6
7  from pheritage.lib.base import BaseController
8  from pheritage.controllers.error import ErrorController
9  from pheritage.model.guarini import Locale, Categoria
10 from pheritage import model
11 from bson.objectid import ObjectId
12
13
14 c_locale = model.mainsession.db.locali
15 c_categoria = model.mainsession.db.categorie
16
17
18 __all__ = ['RootController']
19
20
21 class RootController(BaseController):
22     """
23     The root controller for the pheritage application.
24
25     All the other controllers and WSGI applications
       should be mounted on this
26     controller. For example::
27
28         panel = ControlPanelController()
29         another_app = AnotherWSGIApplication()
30
31     Keep in mind that WSGI applications shouldn't be
       mounted directly: They
32     must be wrapped around with :class:`tg.controllers.
       WSGIAppController`.
33
34     """
35
36     error = ErrorController()
37
38     def _before(self, *args, **kw):
39         tmpl_context.project_name = "pheritage"
40
41     @expose('pheritage.templates.index')
42     def index(self):
43         """Handle the front-page."""
44         print 'ricevuto'
45         return dict(page='index')
46
47     #@expose('pheritage.templates.data')
48     @expose('json')
49     def data(self, **kw):
```

```
50          """This method showcases how you can use the same
    controller for a data page and a display page"""
51          return dict(page='data', params=kw)
52
53      @expose('json')
54      def categories(self):
55          cat = c_categoria.find({}, fields={'_id': False})
56          return dict(data=list(cat))
57
58      @expose('json')
59      def cities(self):
60          cities = list(c_locale.find().distinct('comune'))
61          cities.sort()
62          return dict(data=cities)
63
64      @expose('json')
65      def types(self, city='TORINO'):
66          print(city)
67          t = {}
68          if city != 'TUTTI I COMUNI':
69                  t['comune'] = city
70          types = c_locale.find(t, fields={'_id': False, '
    tipo': True}).distinct('tipo')
71          try:
72              types.remove('')
73          except ValueError:
74              pass
75          types.sort()
76          return dict(data=types)
77
78      @expose('json')
79      def points(self, **kw):
80          t = {}
81          try:
82              if kw['city'] != 'TUTTI I COMUNI':
83                  t['comune']= kw['city']
84          except KeyError:
85              pass
86          try:
87              if kw['type'] != 'Tutti i tipi':
88                  t['tipo'] = kw['type']
89          except KeyError:
90              pass
91          print(t)
92          points = c_locale.find(t, fields=['lat', 'lon', '
    id']).sort('comune')
93          return dict(data=list(points))
94          #return list(points) //Open to csrf attacks
95
```

```
96      @expose('json')
97      def detail(self, id):
98          det = c_locale.find({'_id': ObjectId(id)})
99          return dict(data = list(det))
```

Listing 4.6: Codice completo del file `controllers/root.py`.

Capitolo 5

Piedmont Heritage in azione

Il progetto Piedmont Heritage ed i suoi obbiettivi è descritto nel capitolo 3, mentre il funzionamento nel capitolo 4. In questo capitolo si esaminerà un'ipotetica sessione utente, corredata di screenshot della videata del browser e con la descrizione di quali operazioni avvengono per ottenere il risultato richiesto.

5.1 Installazione

Per installare il progetto é necessario che nel sistema siano installati *python, pip* e *git*. Pip è uno strumento comodo per installare pacchetti per python, mentre Git è un software di controllo versioni e di gestione del codice. Tramite Git è possibile creare dei repository che permettono di effettuare degli *snapshot*, ossia delle *fotografie* degli stati del progetto ed annullare modifiche se necessario, aggiungere nuove features ed inserirle nel progetto solo quando complete e funzionanti. Inoltre è comodo in caso di lavoro con altre persone, perché fornisce strumenti per unire il proprio lavoro con quello degli altri.

Con il comando seguente da eseguire sul terminale si effettua una copia nel proprio computer del repository contenente il progetto:

```
$ git clone https://github.com/andxet/piedmont-heritage.
    git
```

In questo modo il progetto sarà scaricato sul computer. Mongo deve essere avviato, basterà eseguire il seguente comando nella cartella dove scaricato e decompresso, che avvierà il demone che resterà in ascolto di connessioni:

```
$ mongo/bin/mongod
```

Ora si può procedere all'installazione vera e propria.

```
$ cd piedmont-heritage
$ python setup.py develop
$ gearbox setup-app
```

Questa sequenza di comandi si occupa di installare le dipendenze del progetto, tra cui Turbogears, inizializzare il progetto eseguendo la funzione di bootstrap, la quale carica sul database tutti i locali.

Infine si deve avviare il server http:

```
$ gearbox serve
```

Questo comando crea un server attraverso gli strumenti di turbogears. Non è consigliato utilizzare questa configurazione per la distribuzione del progetto perché è tarata per un utilizzo in caso di debug, perché non molto sicuro e per la presenza di messaggi di log.

Aprendo il browser all'indirizzo

```
localhost:8080
```

sarà possibile visualizzare il sito. Sarà necessaria una connessione ad internet anche se il sito è in locale, perché le mappe non sono comprese nel progetto e vengono scaricate dal sito opportuno al momento della visualizzazione.

5.2 Esempio di sessione

La pagina è divisa in sezioni: prima la mappa dei locali, poi un grafico che mostra le categorie, uno che mostra il resoconto dell'età dei locali, ed un resoconto degli oggetti storici. Il nostro interesse si focalizzerà sulla mappa, che apparirà inizialmente come nella figura 5.1. La richiesta che riceve il server non contiene parametri, siccome l'url è quello mostrato precedentemente, quindi viene inviata la pagina html. Inizialmente la pagina non contiene la mappa ma un tag `div` che verrà riempito con l'oggetto mappa. Questa operazione è eseguita dalla funzione `drawMap()` richiamata a fondo pagina: gli elementi come il canvas contenente la mappa, i pulsanti di zoom e la mappa stessa vengono quindi aggiunti tramite codice, ed i dati da visualizzare con richieste asincrone dalla pagina al server. Il processo di creazione della mappa prevede anche il caricamento di default di tutti i comuni, i tipi ed i marker dei locali, che verranno ottenuti ognuno con richiesta diversa (Fig. 5.2 e 5.3).

I comuni ed i tipi sono aggiunti all'interno del relativo `select` tramite codice, creando il relativo tag `option`.

Se un comune viene selezionato, si visualizzeranno nell'altro menu a tendina solo i tipi presenti in quel comune. Al click il client invia una richiesta al server con il comune come parametro, e riceve l'elenco di tipi, il menu a tendina viene prima svuotato e poi popolato dei risultati ricevuti.

Premendo filtra, il client effettua una richiesta al server inviando il comune ed il tipo come parametri. Le voci "Tutti i comuni" e "Tutti i tipi" valgono come valori nulli, indicano che non sono state espresse preferenze e non è necessario effettuare selezioni in base a quel campo. Il server restituisce

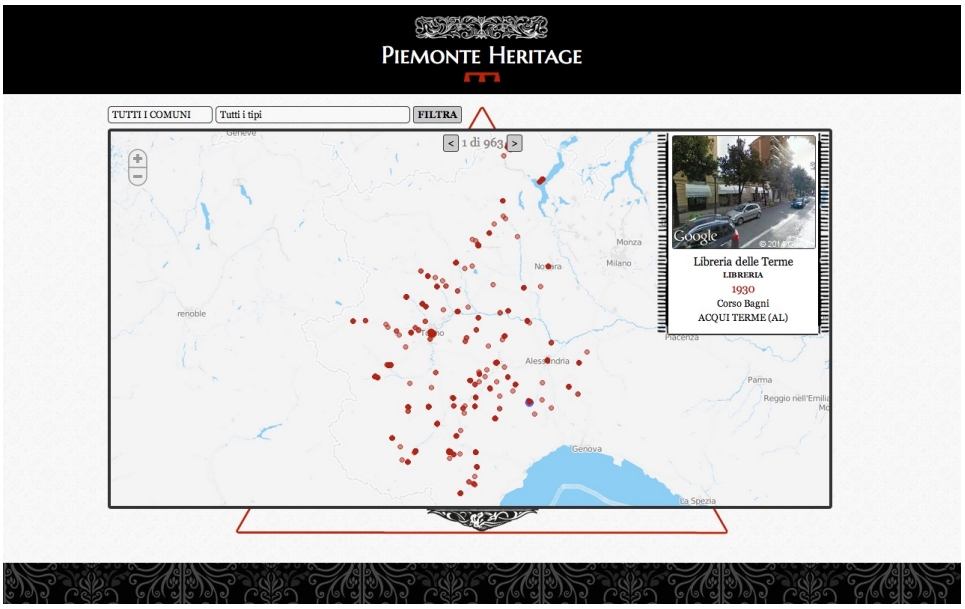


Figura 5.1: Aspetto della mappa all'apertura della pagina.

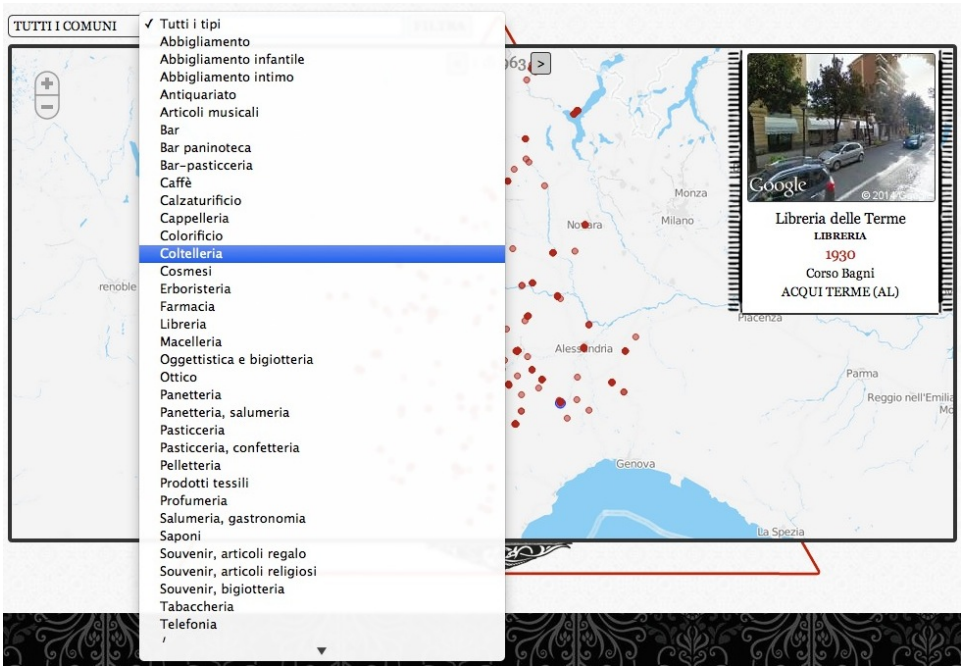


Figura 5.2: Elenco di tutte le tipologie disponibili.

una lista contenente le coordinate di tutti i punti e l'id del locale che rappresentano. Vengono quindi mostrati i marker sulla mappa e viene selezionato il primo, ossia viene effettuata una chiamata a `select()`, che diverrà blu e

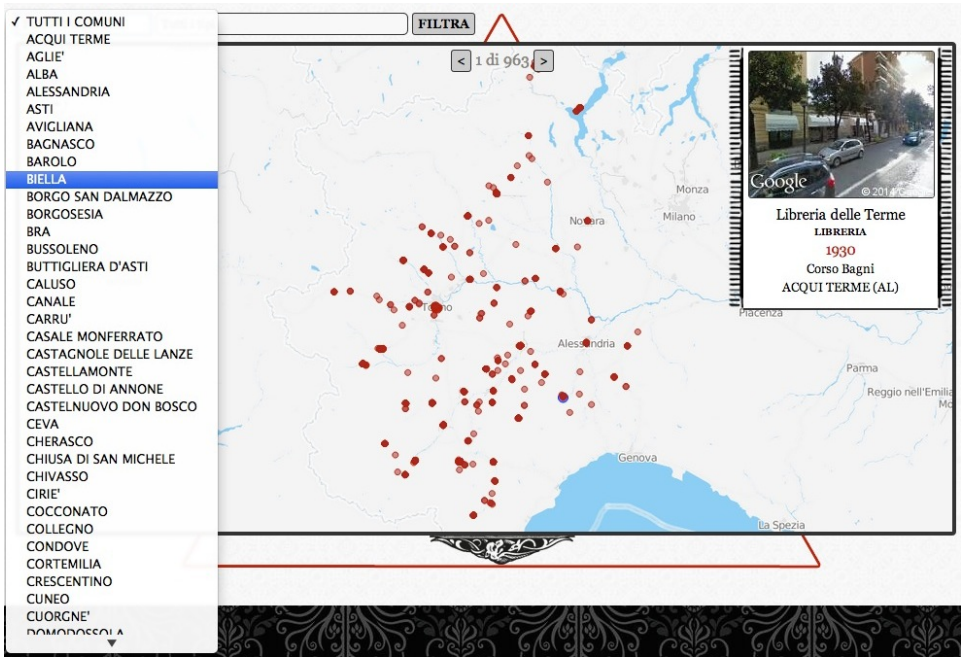


Figura 5.3: Selezione del comune.

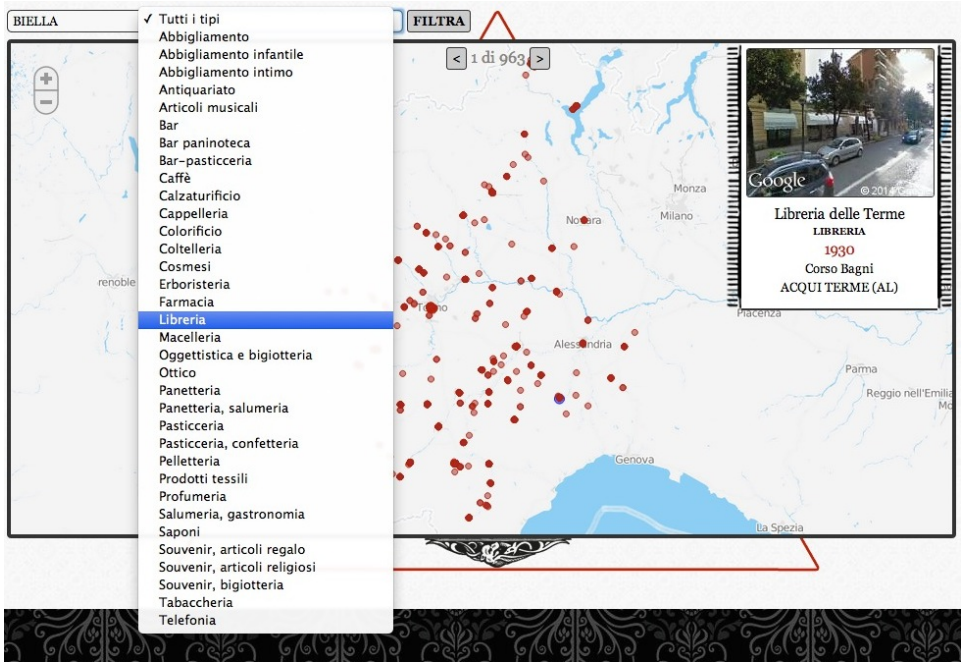


Figura 5.4: Elenco di tipologie disponibili se viene selezionato il comune di Biella.

più grande, ed i dati verranno richiesti al server e mostrati nello specchietto in alto a destra. La mappa mostra solamente i punti filtrati ed adatta lo zoom in modo da mostrarli tutti .

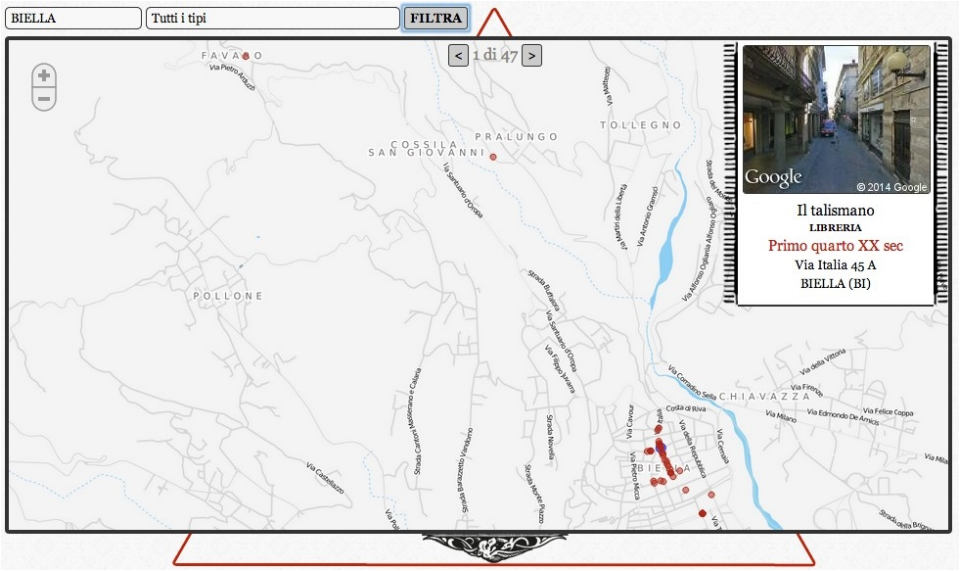


Figura 5.5: Locali storici di Biella, con dettaglio in alto a destra.

I pulsanti di navigazione in alto al centro, richiamano rispettivamente le funzioni `prev()` e `next()`, che fanno retrocedere o proseguire l'iteratore della lista dei locali.

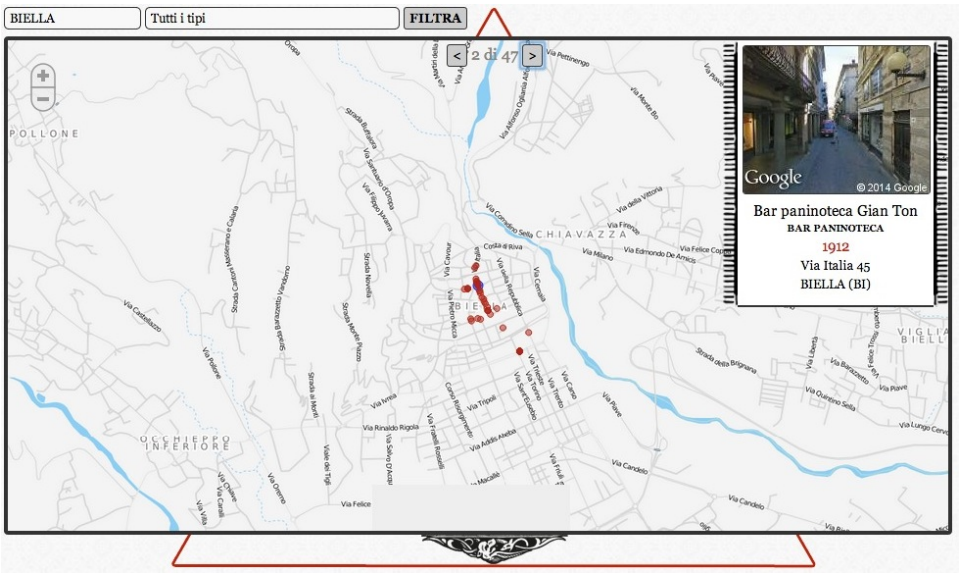


Figura 5.6: Il locale successivo, il numero 2 su 47, è stato selezionato.

Nel caso della figura 5.7, si sceglie di visualizzare i due bar di Biella presenti nel dataset. Il procedimento è uguale a quello descritto precedentemente, in questo caso si effettua la ricerca dei locali con comune="Biella" e tipo="Bar".

Figura 5.7: Selezionando una categoria e poi filtra, vengono mostrati solo i relativi punti.

Nel caso della fig. 5.8 i criteri di ricerca sono solamente quelli per il tipo: tipo="Drogheria, salumeria, pizzicherie e simili".

Come descritto nei precedenti capitoli, la natura del dataset comporta la presenza di dati errati o non coerenti, come quelli mostrati nella figura 5.9.

La mappa è completamente interattiva: oltre alla possibilità di filtrare i risultati, è possibile scorrere la mappa cliccando su di essa e facendo scorrere il mouse, aumentare o diminuire lo zoom con la rotellina del mouse o con i pulsanti in alto a sinistra. Cliccando sui punti essi verranno selezionati, ed i dettagli del locale che rappresentano mostrati nello specchietto in alto a destra.

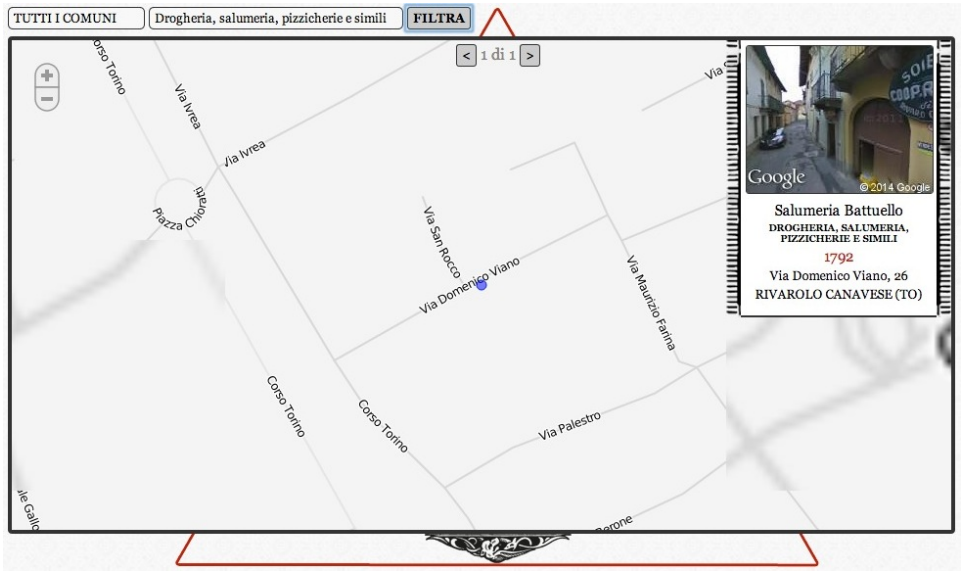


Figura 5.8: Se lo si desidera, si può visualizzare tutti i locali dello stesso tipo in tutti i comuni.

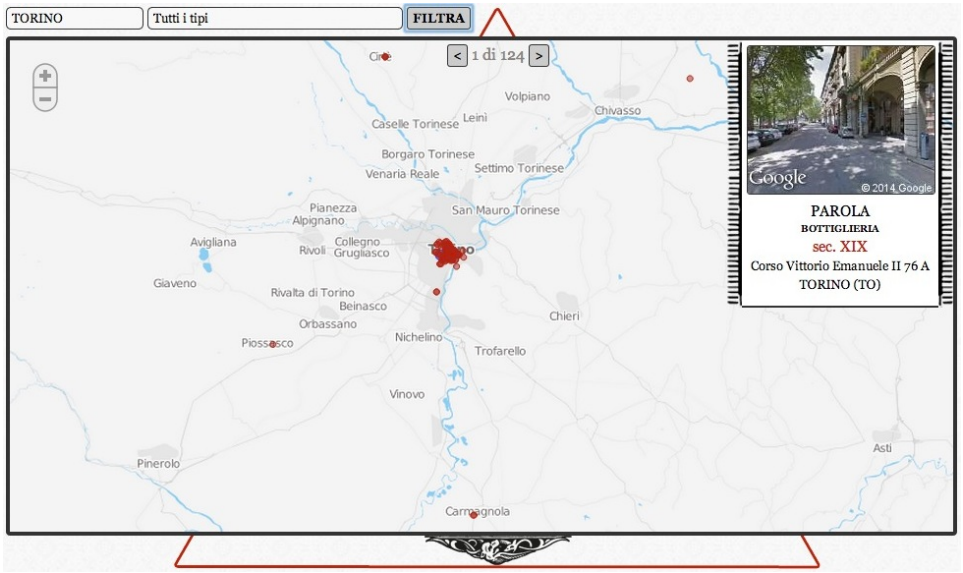


Figura 5.9: Esempio di dati inesatti del dataset.

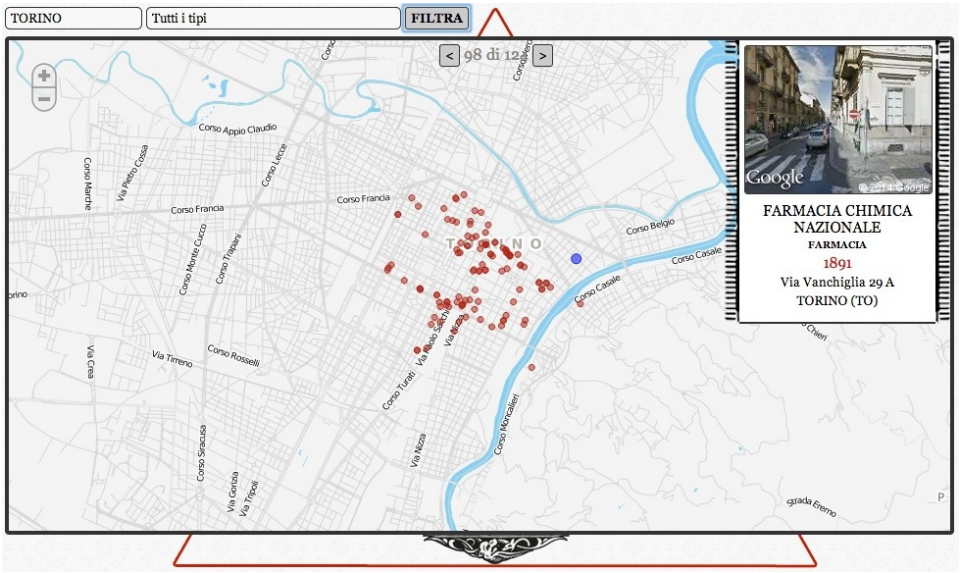


Figura 5.10: Un locale è stato selezionato cliccando sul suo marker.

Capitolo 6

Conclusioni

Le soluzioni presentate sono ovviamente solo alcune delle soluzioni possibili, e sono state scelte in seguito al contesto dello sviluppo del progetto e per sperimentare nuove tecnologie. In questo testo abbiamo visto la soluzione adottata per la visualizzazione di dati georeferenziati con attenzione alla possibilità di crescita della quantità dei dati. Dopo un'introduzione all'argomento nel cap. 1, nel cap. 2 si è vista una definizione di BigData ed una descrizione delle tre fasi che caratterizzano l'elaborazione di grandi moli di dati, una descrizione del pattern MVC e gli strumenti utilizzati. Nel cap. 3 è stato introdotto il progetto Piedmont Heritage e descritto insieme al suo contesto, Mentre nel cap. 4 è stato descritto nei minimi particolari il funzionamento "sotto il cofano" del progetto, mostrando le soluzioni adottate per la risoluzione di numerosi problemi utilizzando gli strumenti presentati nei cap. 2.3 e 2.4. Infine, nel cap. 5 è stata presentata una sessione di esempio corredata di screenshot e descrizione di ciò che succedeva sullo schermo ed del codice che veniva eseguito.

Bibliografia

[Gam02] E. Gamma. *Design Patterns: Elementi Per Il Riuso Di Software a Oggetti*. Professionale / [Pearson education Italia]. Pearson Education Italia, 2002. ISBN: 9788871921501. URL: http://books.google.it/books?id=IkzmAAAACAAJ.

[Jam12] Josh James. *How Much Data is Created Every Minute?* Giu. 2012. URL: http://www.domo.com/blog/2012/06/how-much-data-is-created-every-minute/?dkw=socf3.

[Rez13] A. Rezzani. *Big Data: Architettura, tecnologie e metodi per l'utilizzo di grandi basi di dati*. PerCorsi di studio. Apogeo Education, 2013. ISBN: 9788838789892. URL: http://books.google.it/books?id=PYoFAgAAQBAJ.

[Rus12] Stefano Alberto Russo. *Esperienze di elaborazione dati con Hadoop al CERN e ad ATLAS*. Ott. 2012. URL: http://trieste.linux.it/wp-content/uploads/2012/10/Haddop-LD2012.pdf.

[Vai13] G. Vaish. *Getting Started with Nosql*. Packt Publishing, 2013. ISBN: 9781849694995. URL: http://books.google.it/books?id=oPiT-V2eYTsC.