Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

REALIZZAZIONE DI UN APPLICATIVO PER LA CATALOGAZIONE COLLABORATIVA DEI BENI CULTURALI

Relatore:
Dott.
LUCA BEDOGNI

Presentata da:
GIACOMO LEIDI

Sessione II
Anno Accademico 2017/2018

Alla mia famiglia, che mi ha supportato in questi anni,
a Giulia, che è riuscita a distendere
e a i miei amici presenza costante e necessaria.

Sommario

OpenArtMap è un applicativo per la catalogazione collaborativa dei beni culturali. È diviso in un applicativo Android per la raccolta dati sul campo e un sistema web per la revisione delle modifiche. Il client Android reperisce i dati da un database del Ministero per i Beni e le Attività Culturali e li mostra su una mappa e permette di contribuire nuovi beni culturali mentre l'interfaccia web permette la modifica, l'inserimento e la rimozione delle contribuzoini. In questo scritto saranno individuate le caratteristiche comuni ai sistemi per la catalogazione collaborativa della conoscenza, per poi applicarle alla progettazione di OpenArtMap. Saranno infine analizzate le principali scelte architetturali e implementative che hanno portato alla realizzazione di questo sistema.

Introduzione

OpenArtMap è un applicativo per la catalogazione collaborativa dei beni culturali. Consiste in un client *Android* per la raccolta dati sul campo e un'applicazione web per la revisione delle contribuzioni. Lo scopo di questo sistema è permettere di ottenere prospettive nuove sui beni culturali incrociando le informazioni presenti nei database ministeriali con quelle create e mantenute dalla comunità. In questo scritto saranno esaminati i sistemi collaborativi odierni più celebri e ne saranno individuate le caratteristiche comuni che saranno poi utilizzate per progettare e realizzare il sistema, sarà analizzata l'architettura dei due applicativi e la natura delle loro interazioni e sarà esposta l'implementazione prodotta.

Nel primo capitolo saranno presi in considerazione *Wikipedia*, *OpenStreetMap* e *MusicBrainz*, tre fra i più celebri progetti di catalogazione collaborativa della conoscenza. Ne saranno elencate le principali caratteristiche e i punti comuni da utilizzare in seguito in fase di progettazione. Nel secondo capitolo saranno analizzate le motivazioni che hanno portato a questo scritto e alla realizzazione di *OpenArtMap*, concentrandosi sui motivi che hanno portato ad una struttura collaborativa. Il terzo capitolo spiega le scelte architetturali relative al client *Android* e all'applicativo web e le motivazioni insite in alcune scelte implementative. Nel quarto capitolo sarà presentata l'implementazione effettiva del client *Android*, dell'applicativo web e delle interazioni tra i due sistemi. Infine nel quinto capitolo saranno analizzati i principali limiti di *OpenArtMap* e saranno elencate alcune possibili modifiche per superarli.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

Un sistema di catalogazione collaborativa non è un'idea del tutto nuova: negli ultimi vent'anni è possibile documentare numerosi progetti di successo che dimostrano come un insieme di persone motivate possa realizzare un catalogo di conoscenza collettiva di qualità. Inoltre la recente standardizzazione di formati liberi per la memorizzazione (RDF [45]) e l'interrogazione (SPARQL [46]) di dataset semantici ha permesso alle istituzioni di pubblicare sul Web enormi quantità di dati, liberamente sfruttabili dagli sviluppatori di applicativi. In questo capitolo saranno elencate alcune delle principali iniziative di catalogazione collaborativa, analizzandone pregi e difetti.

Wikipedia

Con quasi 6 milioni di pagine all'indice nella versione anglofona [51] e quinto sito più visitato al mondo (Alexa [2]), Wikipedia è l'enciclopedia libera che permette a chiunque di aggiungere una nuova voce e contribuire al mantenimento di quelle preesistenti.

Fin dalla sua nascita, il 15 gennaio 2001 ad opera di Jimmy Wales e Larry Sanger, la caratteristica principale di Wikipedia è stata di non essere regolamentata da alcun comitato di redazione centrale: gli utenti si organizzano autonomamente stabilendo da soli le regole interne. L'idea di fondo è che a

lungo termine la collaborazione tra gli utenti possa migliorare i contenutile voci più o meno nello stesso spirito con cui viene sviluppato il software libero.

A causa della natura aperta della piattaforma, vandalismi ed errori o imprecisioni sono fenomeni riscontrabili: il miglioramento dell'enciclopedia è dovuto unicamente alle contribuzioni dell'utenza, sia in termini di creazione di nuove voci che di revisione di quelle esistenti. Per garantire la qualità del contenuto delle voci i *wikipediani* si basano su un insieme di linee guida stabilite dalla comunità, volte all'identificazione delle informazioni adatte a far parte di Wikipedia, in particolare esse si fondano sul diritto di citazione come testimonianza dell'attendibilità dei contenuti presenti. Ad esse si fa riferimento in caso di dispute riguardo aggiunte o modifiche ad una voce.

Nel 2003 Jimmy Wales fonda la *Wikimedia Foundation* con lo scopo di fornire gratuitamente al pubblico l'intero contenuto dei vari progetti wiki che si stavano distaccando da Wikipedia. Ad oggi la Wikimedia Foundation promuove più di dieci diversi progetti per la condivisione di conoscenza collettiva tra cui:

- **Wikidata** [48]: base di conoscenza collaborativa in formato *Linked Open Data*. Funge da fonte comune di dati strutturati per gli altri progetti Wikimedia. È l'unico progetto Wikimedia che **non** offre dati sotto licenza *copyleft* ma usa la licenza per il dominio pubblico *Creative Commons 0* [11].
- **Wikimedia Commons** [49]: è un archivio di immagini, suoni e altri files multimediali liberi. I files presenti su Commons possono essere utilizzati da tutti gli altri progetti Wikimedia.
- **Wikibooks** [47]: progetto multilingue per la raccolta e la creazione collaborativa di e-book con licenza libera, come libri di testo o manuali.

Wikipedia stessa inoltre ospita al suo interno numerosi progetti tematici volti a migliorare la qualità dell'enciclopedia in un certo ambito della conoscenza. Data la natura collaborativa dell'enciclopedia quindi la portata delle modifiche attuabili varia molto a seconda degli interessi dell'utenza.

Tra i più rilevanti ai fini di questo scritto possiamo citare il Progetto *GLAM* [50]. *GLAM* è l'acronimo per **Galleries**, **Libraries**, **Archives** e **Museums** ovvero gallerie, biblioteche, archivi e musei. Il progetto è dedicato alla revisione di pagine di Wikipedia relative al settore culturale, agevolando la collaborazione con gli enti culturali.

Il progetto GLAM promuove diverse iniziative per coinvolgere in prima persona i membri delle istituzioni culturali per facilitarne l'integrazione all'interno della comunità di Wikipedia, al fine di migliorare la qualità complessiva delle voci nell'ambito dei beni culturali.

Per esempio un **wikipediano in residenza** è un wikipediano che dedica parte del suo tempo a lavorare presso un'istituzione culturale. I wikipediani in residenza sono pagati di solito dall'istituzione o da Wikimedia, ma possono essere anche volontari.

Il ruolo non è semplicemente quello dell'editore ma quello di permettere all'organizzazione ed ai suoi membri di continuare ad avere una relazione produttiva con l'enciclopedia e la sua comunità anche dopo che il termine della residenza.

OpenStreetMap

Nel 2004 Steve Coast fonda OpenStreetMap, un progetto collaborativo per la creazione di una mappa del mondo libera ed editabile. A differenza di Wikipedia, che punta alla creazione di un'enciclopedia universale, l'obiettivo di OpenStreetMap non è quello di creare una vera e propria mappa ma di creare una base di dati comune e libera di dati geografici. Ad oggi esistono decine di software capaci di mostrare una mappa editabile a partire dai dati offerti da OpenStreetMap, per tutti i principali sistemi operativi o via web.

La comunità di OpenStreetMap suggerisce due metodologie fondamentali per la raccolta dati [38].

L'**Armchair Mapping** è una tecnica che fa affidamento su dati raccolti da altri utenti, come ad esempio tracciati GPS registrati mediante smart-

phone o dispositivi dedicati, immagini aeree fornite da una compagnia [9] o anche immagini stradali fornite da altri progetti collaborativi come *Mapillary* [30], un servizio per la condivisione di fotografie georeferenziate di tutto il mondo, in maniera molto simile a *Google Street View* [17].

Le tecniche di mappatura raccolte sotto il nome di **Outdoor Mapping** invece implicano la raccolta dati di prima mano, muovendosi per il territorio da mappare. Gli utenti, tramite dispositivi GPS, registrano delle tracce lungo i percorsi che vogliono mappare e tramite l'editor ufficiale della comunità (*iD Editor* [21]), ricreano la forma da mappare.

Infine il sistema fornisce la possibilità di inserire delle note georeferenziate e visibili sulla mappa per segnalare errori e discutere di eventuali modifiche o incongruenze.

Come per Wikipedia anche per OpenStreetMap il progresso e la qualità dei dati sono necessariamente determinati dagli interessi degli utenti. All'interno di OpenStreetMap sono presenti diversi progetti tematici fra cui:

- **Accessibilità**: progetto per la mappatura di informazioni utili per i non vedenti come pavimenti tattili o segnali acustici dei semafori, ma anche per la mappatura di luoghi accessibili alle sedie a rotelle;
- **Natura, Conservazione e Ambiente**: progetto che mappa le informazioni rilevanti la conservazione degli ambienti naturali e degli habitat, così come quelle riguardanti i beni culturali;
- **Sport**: esistono molti progetti che si occupano di mappare informazioni rilevanti per diverse attività sportive incluse piste ciclabili, sentieri da trekking, piste da sci e servizi affini.

La maggior parte di queste informazioni non sono presenti nei sistemi di mappatura commerciali, che tendono ad essere più specializzati in informazioni rilevanti per gli automobilisti.

Dato l'enorme impatto scientifico ed economico che potrebbe avere una base di dati geografici completa e di alta qualità, disponibile pubblicamente sul web, si possono trovare in letteratura numerosi studi sulla qualità delle informazioni presenti.

Haklay [18] mostra come in Inghilterra, a marzo del 2010, la copertura della mappa sia arrivata al 69.8% dal 51.2% dell'anno precedente, con una precisione della sovrapposizione dell'80%. Inoltre, data l'inefficacia dei metodi tradizionali di valutazione della qualità su OpenStreetMap a causa dell'assenza di altre fonti affidabili con cui confrontare i dati di molte delle regioni mappate dal progetto,Mooney, Corcoran e Winstanley [32] si occupano di investigare misure della qualità attuabili in maniera non supervisionata senza una fonte fidata. Infine Neis e Zipf [35] mostrano come, nonostante il numero di utenti registrati totali sia in costante aumento dalla nascita del progetto, solo il 38% degli utenti registrati ha effettuato una modifica alla mappa e solo il 5% degli utenti totali abbia contribuito in maniera più significativa.

Aiuti umanitari

Il 12 gennaio 2010 un catastrofico terremoto di magnitudo 7, con epicentro a circa 25 chilometri dalla capitale, colpisce le coste di Haiti condizionando per sempre le vite di circa 3 milioni di persone. In seguito al terremoto la comunità di OpenStreetMap comincia molto presto [40] a migliorare la qualità delle mappe di Haiti sulla base di immagini aeree pre-terremoto fornite da Yahoo e in seguito, con la disponibilità delle immagini post-terremoto fornite da GeoEye e DigitalGlobe, si adopera per rendere disponibili questi dati in un formato utilizzabile per la produzione di mappe.

I dati geografici risultanti sono forniti in formato shapefile GIS e vengono aggiornate parecchie volte all'ora. ITHACA (Information Technology for Humanitarian Assistance, Cooperation and Action [22]) e il WFP (World Food Program [52]) usano quindi i dati di OpenStreetMap per la prima valutazione del danno a Port-au-Prince.

Soden e Palen [43] investigano come dall'esperienza di Haiti, data la prontezza con cui la comunità di OpenStreetMap è riuscita a gestire l'emergenza, è nato l'Humanitarian OpenStreetMap Team [20], un progetto interno a Open-

StreetMap e associazione noprofit con l'obiettivo di essere punto di contatto tra attori umanitari e comunità di open mapping.

MusicBrainz

MusicBrainz è un progetto che mira a realizzare un database libero e collaborativo di metadati riguardanti la musica. MusicBrainz fu fondato nel 2000 da Robert Kaye in risposta al cambio di licenza imposto al *Compact Disk Database* [10], un database preesistente per l'interrogazione di metadati riguardanti CD musicali. Da allora MusicBrainz ha espanso i suoi obiettivi oltre la catalogazione di CD, fino a diventare un'enciclopedia generale sulla musica.

Nel 2005 nasce la non-profit *MetaBrainz Foundation* [31] con lo scopo di supportare lo sviluppo di *MusicBrainz* e altri progetti simili che si stavano separando da *MusicBrainz* [1] [29]. Il *contratto sociale* di *MetaBrainz* [34] stabilisce chiaramente i pilastri su cui si basano i progetti della fondazione, indicando specificatamente che i progetti rimarranno sempre liberi al 100%, non solo dal punto di vista dello sviluppo del software necessario ma anche dal punto di vista delle licenze sui dati.

MusicBrainz contiene informazioni riguardo gli artisti, i brani da loro registrati, e le relazioni che si creano. A novembre 2018 il database contiene informazioni su circa un milione e mezzo di artisti, 2 milioni di releases, e 19 milioni di brani. Tutte queste voci sono create e mantenute da circa 2 milioni di editori volontari in base a linee guida stabilite dalla comunità. L'utenza può interagire con la piattaforma tramite il portale web che permette di modificare e votare le modifiche o applicazioni dedicate al tagging come l'applicativo desktop ufficiale Picard [33].

Il modello collaborativo di MusicBrainz differisce leggermente da quello di Wikipedia e OpenStreetMap, in quanto affianca all'operazione di editing una votazione da parte della comunità sulla bontà della modifica. Solo gli utenti registrati da più di due settimane e che hanno ricevuto almeno 10 verdetti

favorevoli sulle loro modifiche possono partecipare al processo di votazione. La stragrande maggioranza delle votazioni resta aperta per 7 giorni, durante i quali l'utente che ha proposto la modifica può motivare le proprie scelte stilistiche. Se al termine della settimana i voti contrari sono **almeno** pari a quelli favorevoli la modifica viene rifiutata, se invece i voti favorevoli sono superiori la modifica viene accettata.

Hemerly [19] offre nella sua analisi un efficace spaccato della comunità a marzo del 2011. Osservando i risultati, emerge una notevole disparità tra il numero delle modifiche e il numero di voti sull modifiche, così come esiste una correlazione positiva tra le iscrizioni di un utente (gli utenti hanno la possibilità di indicare sul loro profilo i loro artisti preferiti) e il numero di modifiche effettuate, suggerendo che gli utenti tendano a contribuire ad ondate in base al loro tempo libero e all'acquisizione di nuova musica.

Tra le altre cose, Hemerly esamina le motivazioni che spingono gli utenti alla collaborazione per uno scopo comune, basandosi fondamentalmente sulle idee proposte da Lakhani e G. Wolf [26]. Le interviste e i sondaggi indicano che l'utenza di MusicBrainz è motivata fondamentalmente da un senso di dovere/appartenenza verso la comunità, specialmente per i valori dell'open source e delle Creative Commons. Gli utenti quindi ritengono di sforzarsi per la creazione di una risorsa informativa di valore che intendono utilizzare liberamente.

Capitolo 2

Motivazioni

L'Italia per molteplici ragioni storiche è uno dei paesi con il patrimonio culturale più variegato al mondo. Non è difficile pensare ad esempi celebri nell'ambito di beni archeologici, beni paesaggistici e naturalistici, beni architettonici, storici e artistici che ogni anno attirano milioni di persone da tutto il mondo.

Sebbene soprattutto nelle città d'arte sia da sempre una priorità, tenere traccia, catalogare e promuovere i beni culturali dislocati in tutto il territorio non è mai stato un problema semplice. Esistono diversi enti che, sotto la direzione del *Ministero per i Beni e le Attività Culturali* (**MiBAC**), si adoperano per fare ricerca e sviluppare metodologie in questo ambito, come l'*Istituto Centrale per il Catalogo e la Documentazione* (**ICCD** [24]) che si occupa di svolgere funzioni in materia di ricerca, indirizzo, coordinamento tecnico-scientifico, finalizzate alla catalogazione e documentazione dei beni culturali, o l'*Istituto Centrale per gli Archivi* (**ICAR** [23]), organismo di studio e ricerca applicata che elabora metodologie in materia di inventariazione di archivi storici.

Una parte preponderante delle attività dei vari istituti del *MiBAC* consiste nello sviluppo di software finalizzato alla descrizione archivistica, alla normalizzazione dei dati e all'interoperabilità tra sistemi informativi. Per esempio l'*ICCD* mette a disposizione sul web il **SIGeCweb** (*Sistema Infor-*

mativo Generale del Catalogo [42]) che raccoglie e organizza circa 800.000 schede descrittive di beni mobili, immobili e immateriali. L'*ICAR* invece si occupa, tra le altre mansioni, di coordinare il *Sistema Archivistico Nazionale* (**SAN** [41]) il portale web unificato delle risorse archivistiche nazionali, sviluppate a livello regionale e locale.

Con l'avvento del *Semantic Web* e la standardizzazione di formati e linguaggi per la memorizzazione (*RDF* - Resource Description Framework [45]) e l'interrogazione (*SPARQL* - SPARQL Protocol and Query Language [46]) di dataset semantici, numerosi enti governativi fra cui il *MiBAC* iniziano a prodigarsi per convertire il loro patrimonio informativo in formato *Linked Open Data*.

Nel 2014 nasce così **dati.beniculturali.it** [13], piattaforma web con una forte connotazione sperimentale, finalizzata alla valorizzazione del patrimonio infomativo del *MiBAC* secondo la logica dei *LOD*. La piattaforma è il risultato di un primo step (2014 - 2016 [14]), durante il quale sono stati rilasciati in *LOD* i dati dei **Luoghi della cultura** e le anagrafiche di **Archivi** e **Biblioteche**, e di una seconda fase (da novembre 2017 [15]) che ha visto l'*ICCD* avvalersi dei *Linked Open Data* come veicolo di pubblicazione del *Catalogo generale dei beni culturali*, collegando così i **Luoghi della cultura** con i beni culturali in essi contenuti.

Dalla disponibilità di questa enorme mole di informazioni disponibili liberamente per l'interrogazione tramite endpoint *SPARQL* e collegate ad altri "silos" esterni di dati aperti, è nata l'idea di un sistema per la catalogazione e l'*information retrieval* di dati riguardanti i beni culturali sfruttando i meccanismi virtuosi delle comunità collaborative come Wikipedia, OpenStreetMap e MusicBrainz. L'idea è quella di permettere, attraverso interrogazioni al sistema, di ottenere nuove prospettive sui beni dislocati nel territorio nazionale in grado di offrire maggiori informazioni (o di miglior qualità) grazie alle contribuzioni della comunità, rispetto alla mera interrogazione dei database ministeriali. Come interfaccia principale per l'applicativo è stata scelta una mappa, in quanto permette una visualizzazione orizzontale dei risulta-

ti e permette di vedere naturalmente i principali pattern nelle risposte alle interrogazioni.

Perchè dunque permettere ad ogni utente di avere la possibilità di modificare i dati senza restrizioni? L'approccio comunitario alla catalogazione permette, prendendo spunto dalle tecniche accademiche di *peer review*, di aumentare notevolmente la dimensione e la qualità dei dati catalogabili. Inoltre, data la potenziale capillarità della distribuizione geografica delle comunità online, un sistema collaborativo potrebbe permettere un'analisi dei dati più puntuale rispecchiando, non solo i dati ministeriali, ma anche le informazioni locali raccolte sul campo dalla rete di volontari.

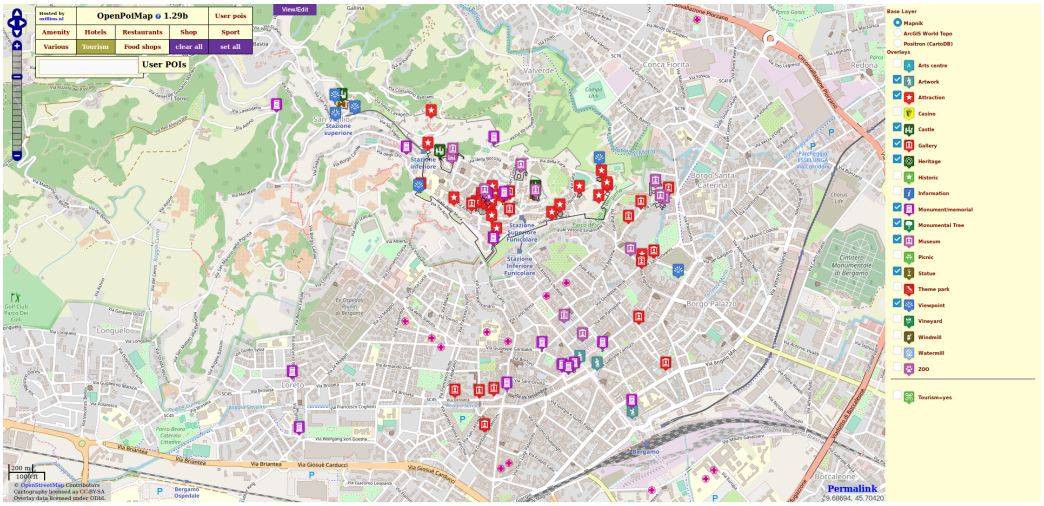

Figura 2.1: Bergamo su *OpenPoiMap*

Per esempio interrogando *OpenPoiMap* [37] (servizio web che permette di visualizzare su una mappa i punti di interesse presenti in *OpenStreetMap*), è possibile osservare in Piazza San Domenico a Bologna quattro punti di interesse rappresentanti le due tombe dei Glossatori e le due colonne presenti nella piazza. All'interno del database di *dati.beniculturali.it* invece nessuno dei quattro beni è presente.

Oppure prendendo in considerazione la città di Bergamo è possibile notare come, per quanto eventualmente incomplete, siano molto maggiori le

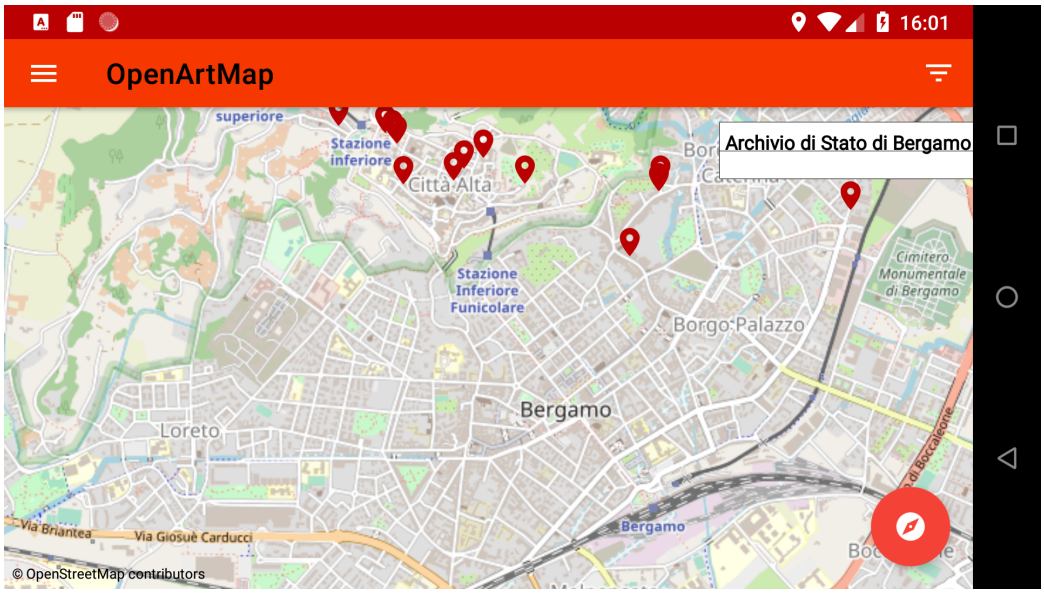

Figura 2.2: Bergamo su *OpenArtMap*

informazioni presenti in *OpenStreetMap* proprio grazie alla capillarità della diffusione della comunità.

In questo scritto sarà presentata la realizzazione di *OpenArtMap* un applicativo per *Android* per la visualizzazione dei dati presenti su *dati.beniculturali.it* e la contribuzione di nuovi beni culturali, e di un applicativo web per la revisione delle contribuzioni.

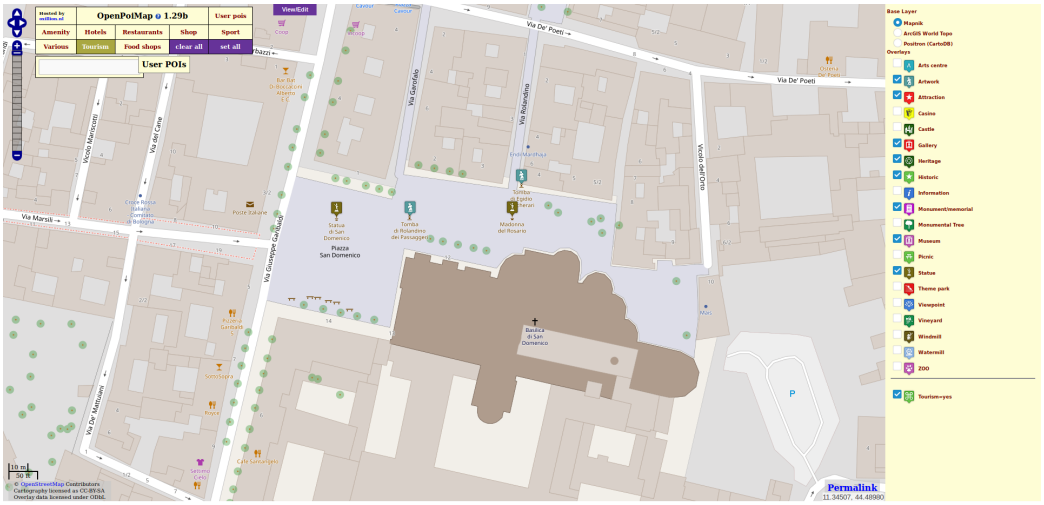

Figura 2.3: Piazza San Domenico su *OpenPoiMap*

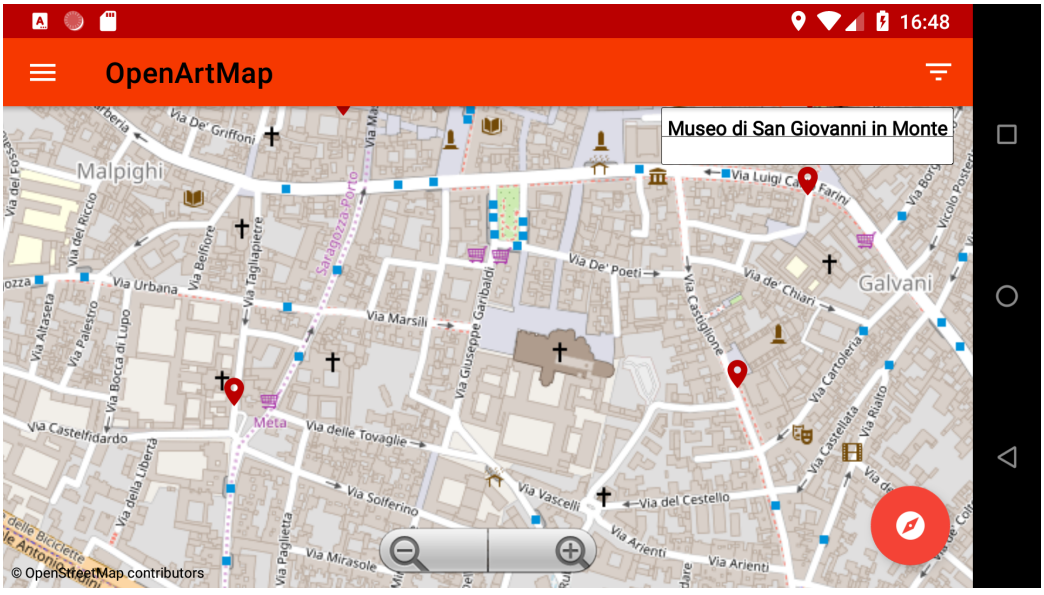

Figura 2.4: Piazza San Domenico su *OpenArtMap*

Capitolo 3

Architettura

OpenArtMap è un sistema per la catalogazione collaborativa dei beni culturali. È diviso in un applicativo Android per la visualizzazione e la raccolta dati sul territorio e un applicativo web per la revisione delle contribuzioni. Dovendo progettare un applicativo per la raccolta dati sul campo, si è deciso di scegliere un'architettura che permettesse la creazione di contributi anche in assenza di una connessione ad Internet. In questo capitolo saranno analizzati i principali casi d'uso presi in considerazione e le architetture del client Android e dell'applicativo web di revisione.

Casi d'uso

Ai fini di questo scritto sono stati presi in considerazione tre casi d'uso principali: la consultazione del catalogo, l'inserimento di una nuova voce riguardo ad un bene culturale e la modifica di una contribuzione. In seguito saranno approfonditi uno per uno al fine di illustrarne le implicazioni.

1. **Consultazione del catalogo**: All'apertura dell'applicativo viene mostrata la mappa all'utente. Attraverso la mappe l'utente deve avere la possibilità di consultare sia il catalogo dei beni disponibili su *dati.beniculturali.it* , che quello del server delle contribuzioni. La mappa può essere trascinata, ruotata e ingrandita, e nell'interfaccia deve essere

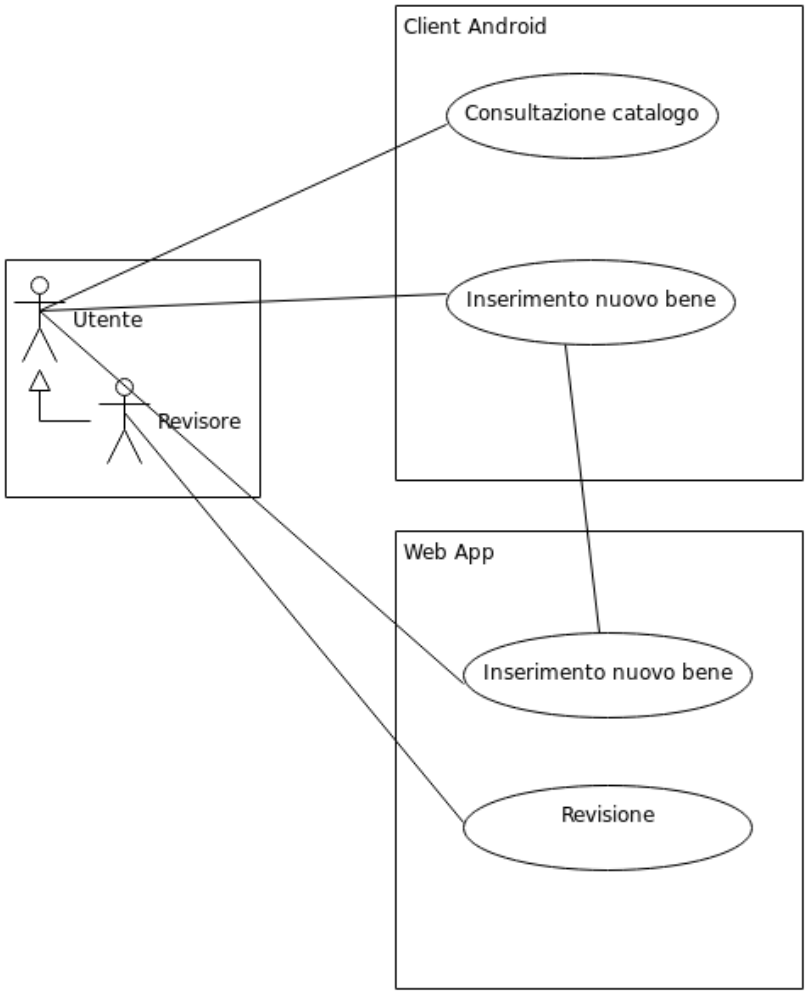

Figura 3.1: Casi d'uso OpenArtMap

presente un bottone per scaricare i beni culturali all'interno della vista sulla mappa. I beni culturali provenienti da sorgenti diverse appaiono sulla mappa con icone di colore diverso. Selezionando un marker sulla mappa viene caricata l'interfaccia di dettaglio di un bene culturale.

2. **Inserimento di un nuovo bene culturale**: L'utente deve avere la possibilità di inserire un nuovo bene culturale all'interno del sistema. L'utente seleziona sulla mappa una posizione e gli viene chiesto di inserire un insieme di informazioni di base atte all'identificazione del bene come nome, descrizione, indirizzo etc. Una volta confermato il client salva su disco le contribuzioni e successivamente il client le sincronizza con il server di revisione alla prima occasione utile.

3. **Revisione di una voce**: L'utente (o un utente con il ruolo di revisore) può accedere ad un interfaccia web sul database dei beni culturali contribuiti. Nella schermata sono presentate tutte le contribuzioni sincronizzate che l'utente può selezionare, inserire e cancellare. Selezionando una voce viene mostrata una vista di dettaglio su tale contribuzione. Dalla vista di dettaglio l'utente può modificare singolarmente i campi della contribuzione in questione e salvare o annullare le modifiche.

Applicativo Android

La capillarità con cui si sono diffusi gli smartphone ha portato, quasi naturalmente, alla scelta di Android come piattaforma per il client per la raccolta dati sul campo.

Per l'architettura dell'applicativo Android si è deciso, in fase di progettazione, di aderire il più strettamente possibile alle guidelines dette *Android Jetpack* [4] fornite da *Google* insieme ad *AndroidX* [8], la nuova versione della *Android Support Library* [7].

L'architettura proposta da Google è un'architettura *reactive*, fondamentalmente basata sul pattern **Model-View-ViewModel**. Colonna portante

dell'architettura è la classe `LiveData` (inclusa in *AndroidX*) che effettivamente permette di reagire a dei cambiamenti di uno stato e permette la costruzione di pipelines asincrone per l'elaborazione dei dati.

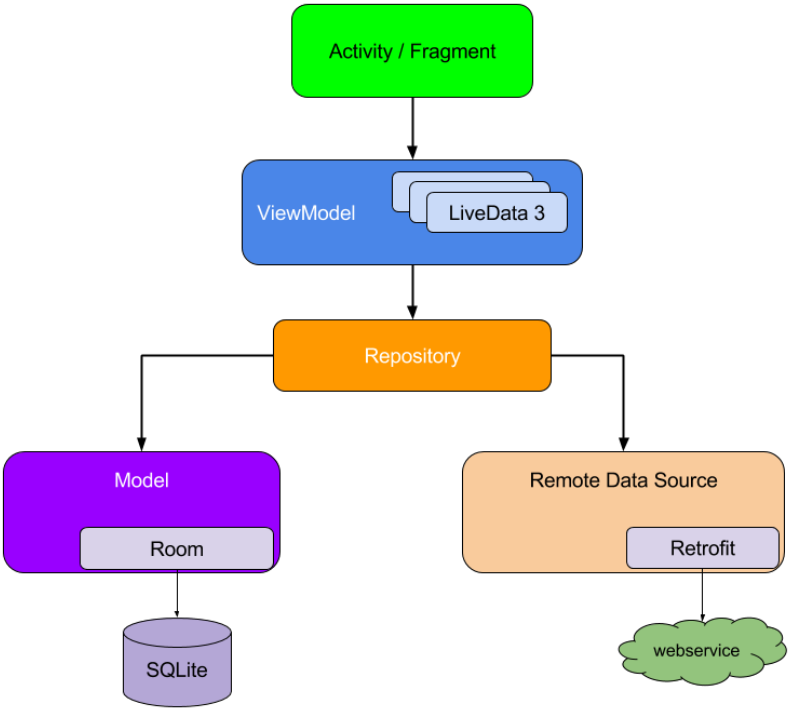

Figura 3.2: Architettura OpenArtMap. Fonte: [5]

L'idea fondamentale è che - dato che `LiveData` è una classe osservabile che comincia a trasmettere aggiornamenti solo se il suo osservatore si trova in uno stato attivo - le *View* osservando un `LiveData` esposto da un *ViewModel* scatenino una richiesta di rete a livello del *Repository* che, essendo avvolta da un oggetto `LiveData`, verrà processata dal *ViewModel* in maniera asincrona; una volta processata verrà automaticamente notificato il cambio di stato alla *View* che provvederà ad utilizzarlo.

Tutte le interazioni con il database di *dati.beniculturali.it* avverranno tramite l'endpoint *SPARQL* offerto dal portale, mentre il dialogo con il server di *OpenArtMap* avverrà attraverso l'API *RESTful* fornita. Data la sempli-

cità del protocollo *SPARQL* (le richieste *SPARQL* possono essere sia `GET` che `POST` e devono contenere un parametro `query` che contiene la query da eseguire) è stata scelta la libreria *Retrofit* che permette un accesso *type-safe* ad un'API web remota.

MVVM

Il pattern **MVVM** è un pattern architetturale creato dagli architetti di *Microsoft* Ken Cooper e Ted Peters per semplificare la programmazione ad eventi delle interfacce utente.

MVVM facilita la separazione dello sviluppo di un'interfaccia utente grafica (**View**) - via linguaggio di markup o via codice - dallo sviluppo della business logic (*data* ***Model***). Il **ViewModel** invece è un convertitore di valori, ovvero si occupa di esporre i dati dal *Model* in maniera tale da poterli maneggiare e presentare facilmente.

La principale differenza tra il *ViewModel* e il *Presenter* del pattern *Model-View-Presenter* è che il *Presenter* possiede un riferimento alla *View*, al contrario del *ViewModel*. Nel *MVVM* infatti la *View* viene legata direttamente ai metodi del *ViewModel* per inviare e ricevere aggiornamenti. Per funzionare in maniera efficiente quindi questo approccio necessita di una tecnologia di *data binding*.

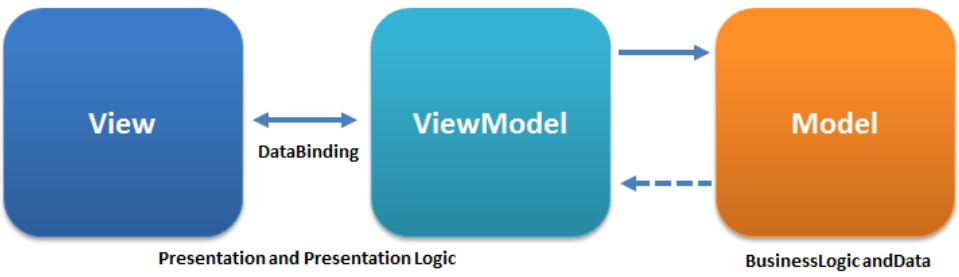

Figura 3.3: Pattern Model-View-ViewModel. Fonte: [44]

Anche in questo frangente sfruttiamo una libreria di *AndroidX* chiamata *DataBinding* [3], che permette di legare dichiarativamente gli elementi del-

l'interfaccia utente all'interno dei file *XML* di layout ai dati. Un ulteriore vantaggio della libreria di *DataBinding* è il supporto nativo per il binding con oggetti di tipo `LiveData`. Il framework si occupa infatti di generare delle classi che osservano l'oggetto `LiveData` e aggiornano il contenuto della *View* di conseguenza.

In base all'architettura di *Android Jetpack*, partendo dall'assunto di permettere le contribuzioni e la consultazione del catalogo sincronizzato anche in assenza di un collegamento ad Internet, si è scelto di utilizzare il pattern **Repository** per fornire un'interfaccia unica sopra le varie sorgenti di **Model**.

Il pattern **Repository** infatti astrae le sorgenti di dati occupandosi, in questo caso, di implementare la logica che decide se recuperare dei dati dal disco o scaricarne una versione aggiornata dalla rete. È stato prodotto un *Repository* per ogni entità nel database che dovesse essere manipolata direttamente da un *ViewModel*.

In definitiva è necessario costruire un sistema che colleghi le due sorgenti di dati, disco e rete, in un unico oggetto `LiveData` in modo da poter fornire l'ultimo risultato scaricato nel caso di assenza di connettività.

Infine l'oggetto `LiveData` può essere ritornato al *ViewModel* che si occupa di preprocessarlo (per esempio nel caso di una richiesta di rete può creare altri `LiveData` che notificano alla *View* lo stato di completamento) e infine consegnarlo alla *View*. Tutti i successivi cambiamenti su quell'oggetto verranno trasmessi automaticamente alla *View* senza necessità di interazione con il *ViewModel* e senza bloccare il thread principale.

Applicativo Web

Data la semplicità dei requisiti si è deciso di strutturare l'applicativo web secondo il pattern architetturale **Model-View-Controller** per implementare un'API *RESTful* per l'inserimento, la modifica e la cancellazione di contribuzioni ad *OpenArtMap*. Come si vedrà nel capitolo 4 sarà necessario

implementare un endpoint per l'interfaccia *HTML* sul database, un endpoint per l'inserimento di nuove contribuzioni e un endpoint per il recupero dei cambiamenti in un formato utile al client *Android*.

Per quanto riguarda la persistenza su disco delle contribuzioni sarà implementato un database *SQLite* con un'unica tabella contenente le contribuzioni.

La tabella, nell'architettura di *OpenArtMap Server*, rappresenta il **Model**. Gli endpoint, leggendo la tabella, agiscono da **Controller** e ritornano al client una **View** sui dati nel formato richiesto.

Capitolo 4

Realizzazione

Android

Per l'implementazione del client *Android* è stato scelto il linguaggio *Kotlin*, principalmente per la sintassi più leggera e l'approccio conservatore all'utilizzo del `null`, nonostante la completa interoperabilità, rispetto a *Java*. Inoltre la tendenza a forzare uno stile di programmazione più funzionale (per esempio tramite l'utilizzo delle funzioni standard `let`, `run`, `with`, `also` e `apply` [25]) ha portato in diversi casi ad una migliore leggibilità del codice.

Nell'esaminare l'implementazione dell'applicativo, per semplificarne l'esposizione, sarà adottato un approccio **top-down**, partendo dall'interfaccia utente fino a scendere al livello dei dati.

Livello UI

OpenArtMap è stata struttutata come una *Single Activity Application*, ovvero un'applicazione con una sola `Activity` che si occupa di gestire la navigazione tra i `Fragment` e ripristinarne lo stato dopo i cambiamenti di configurazione.

Per implementare la `MapView` si è scelto di utilizzare la libreria *OSMdroid* [39] che, per mezzo delle tiles di *OpenStreetMap*, fornisce un'implementazione libera delle API(*v1*) fornite dalla `MapView` di *Google*.

Al primo avvio l'`Activity` richiede i permessi di accesso in **scrittura al disco** (`WRITE_EXTERNAL_STORAGE`) e di accesso alla **localizzazione** (`ACCESS_FINE_LOCATION`) e una volta ottenuti, carica il `PlacesMapFragment`.

OpenArtMap contiene sei `Fragment` principali: `PlacesMapFragment`, `EventMapFragment`, `NewPlaceFragment`, `FilterFragment`, `DetailsFragment` e `EditPlaceFragment`. Data la necessità di mostrare una mappa con un *overlay* di icone nei primi tre `Fragment`, per ridurre al minimo la duplicazione del codice, è stata implementata una classe astratta `BaseMapFragment` che si occupa di inizializzare la mappa e configurarne la cache su disco, inizializzare un overlay con la posizione dell'utente e centrare la visuale sull'utente.

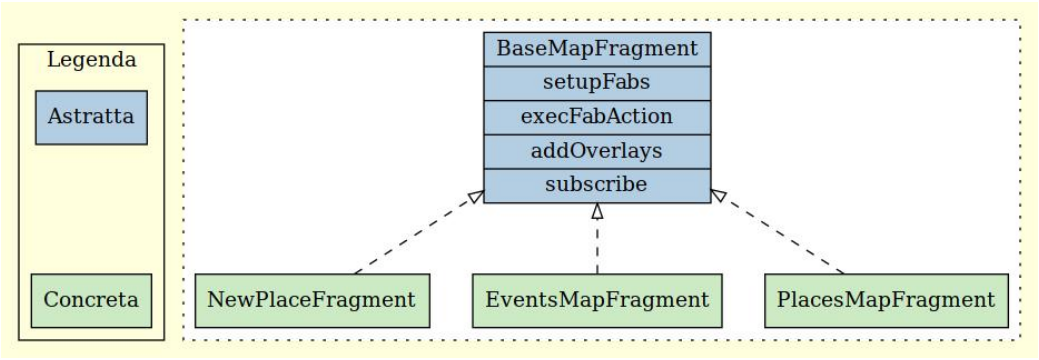

Figura 4.1: `BaseMapFragment`

Il `BaseMapFragment` possiede come unici membri privati un riferimento alla `MapView` e un riferimento all'overlay per la posizione dell'utente, e fornisce metodi implementabili dalle sottoclassi che vengono chiamati durante i rispettivi passaggi di stato nel *lifecycle* del `Fragment`:

- `setupFabs`: inserire uno o più `FloatingActionButton` o *FAB*,
- `execFabAction`: l'azione da assegnare al *FAB* principale,
- `addOverlays`: inserire uno o più overlay di icone sulla mappa,
- `subscribe`: dove le sottoclassi ottengono un riferimento a un `ViewModel` e inizializzano le *View* osservando i `LiveData` esposti dal `ViewModel`.

Il `PlacesMapFragment` mostra, oltre alla posizione dell'utente, un overlay di beni culturali provenienti da *dati.beniculturali.it* (icone rosse) e un overlay con i beni provenienti dal server di *OpenArtMap* (icone gialle). L'`EventsMapFragment` mostra invece un overlay di eventi culturali futuri provenienti da *dati.beniculturali.it* (icone verdi).

Tenendo premuto sull'icona di un bene culturale appare il `DetailsFragment`, una schermata di dettaglio che contiene numerose informazioni riguardo al bene tra cui un'immagine, una descrizione, l'indirizzo, i contatti sia del bene che dei servizi correlati (come la biglietteria) e il prezzo dell'eventuale biglietto.

Sia il `PlacesMapFragment` che l'`EventsFragment` mostrano nell'angolo in basso a destra un *FAB* che permette di scaricare i beni culturali all'interno della visuale corrente. Inoltre il `PlacesMapFragment` mostra nella `ToolBar` un bottone che richiama il `FilterFragment`.

Il `FilterFragment` è stato implementato per mezzo di un `DialogFragment` che, data una lista di tipi di bene culturale, mostra una lista di *radio buttons*. Quando l'utente seleziona un *radio button* il *ViewModel* viene notificato tramite un `LiveData` che contiene il tipo di bene culturale da visualizzare e se l'utente conferma la scelta viene aggiornata la mappa. Se l'utente annulla l'operazione lo stato del filtro viene azzerato e viene mostrata la mappa.

Quando l'utente seleziona dal `NavigationDrawer` la voce "Contribuisci" viene lanciato il `NewPlaceFragment`. Il `NewPlaceFragment` mostra un'icona gialla al centro della visuale della mappa e un *FAB* per l'inserimento temporaneo dell'icona sulla mappa. Specificamente quando viene premuto il *FAB* viene nascosta l'`ImageView` che era stata sovrapposta alla mappa e viene inserita un'icona uguale nell'overlay presente nella `MapView` permettendo all'utente di accertarsi dell'esattezza della posizione e infine confermare o annullare l'inserimento.

Nel caso l'utente confermi l'inserimento gli viene presentato l'`EditPlaceFragment`. L'`EditPlaceFragment` contiene una serie di `CardView` che l'utente può compilare per aggiungere informazioni riguardo al bene di cui gli unici campi

obbligatori sono il nome e il tipo di bene. Quando l'utente conferma i dati premendo il bottone di salvataggio il `ViewModel` viene notificato tramite il metodo `saveChangeset` e salva la contribuzione su disco. Infine viene lanciato un `JobService` che attende la presenza di una rete *WiFi* e carica le contribuzioni locali sul server di *OpenArtMap*, tramite una richiesta `POST` all'endpoint `/changesets`.

Livello ViewModel

La funzione dei `ViewModel` in *OpenArtMap*, e più in generale nel *MVVM*, è quella di esporre dati da legare alle *View*. Una particolarità della classe `ViewModel` di *AndroidX* è quella di avere un *lifecycle* completamente scollegato dalla *View* a cui è legato tramite `LiveData` e legato solo al ciclo di vita dell'applicazione. Quindi può sopravvivere immune ai cambi di configurazione che solitamente distruggono e ricreano la maggior parte delle componenti di un'applicazione.

`LiveData` inoltre è una classe osservabile e *lifecycle aware*, ovvero rispetta i *lifecycles* delle altre componenti dell'applicazione come `Activity` o `Fragment`. Questa coscienza permette ai `LiveData` di aggiornare solo i componenti che si trovano in uno stato **attivo** (`STARTED` o `RESUMED`) del *lifecycle*.

Questo spiega il motivo per cui i `ViewModel` sono i perfetti convertitori di dati tra i *Models* e le *View*: un `ViewModel` viene inizializzato in maniera *lazy*, solitamente nel metodo `onCreate` o `onCreateView`, dal primo componente che lo necessita e **ogni successiva** istanziazione di quella classe `ViewModel` all'interno di quel dato *lifecycle* riceve la stessa copia dell'oggetto `ViewModel`.

Prendiamo in considerazione per esempio il `PlacesViewModel`. Il `PlacesViewModel` è il `ViewModel` responsabile di reperire i dati sui luoghi da mostrare nel `PlacesMapFragment` e la lista dei tipi di bene visualizzabile nel `FilterFragment`. Il codice per l'istanziazione del `ViewModel` è molto simile a questo in entrambi i `Fragment`:

```
activity?.let {
    val mapFactory = providePlacesViewModelFactory(it)
    placesViewModel = ViewModelProviders.of(this, mapFactory)
                                        .get(PlacesViewModel::class.java)
} ?: throw IllegalStateException("Activity cannot be null")
```

Questo è uno dei casi in cui brilla la capacità di *Kotlin* di incapsulare la *"nullabilità"* tramite l'utilizzo del cosiddetto *Elvis operator* `?:` il quale, dopo aver valutato l'espressione alla sua sinistra, se il valore è **diverso** da `null` lo ritorna, altrimenti valuta l'espressione a destra e ne ritorna il valore.

Il metodo `providePlacesViewModelFactory` è un metodo dell'oggetto statico `ServiceLocator` che implementa l'omonimo pattern architetturale per la costruzione delle dipendenze. Basti dire che all'interno del metodo `let` di un oggetto della proprietà `activity` (derivata da *Kotlin* dal metodo `Fragment.getActivity()`), la keyword `it` corrisponde all'oggetto `Activity` e il suo *context* viene utilizzato dalla classe `ViewModelProviders` per recuperare il `ViewModel` del tipo passato al metodo `get`.

Una volta istanziato il `ViewModel` inizializza i `LiveData` che deve esporre, chiamando i rispettivi metodi del *Repository*, e si occupa di applicare le trasformazioni necessarie. Per esempio la maggior parte dei `ViewModel` deve gestire un qualche tipo di richiesta di rete che può essere in stato di *caricamento*, *successo* o *fallimento*. Dato che la richiesta di rete è rappresentata a livello del *Repository* da un `LiveData` quasi tutti i `ViewModel` hanno dovuto fornire un metodo per la gestione delle richieste di rete simile a questo. In questo estratto è possibile notare le potenzialità della classe `LiveData` per costruire vere e proprie pipelines componibili per l'elaborazione del dato.

```
init {
    val liveData = repository.getLiveData()
    exposedLiveData.addSource(liveData) {
        handleResponse(it, loadingLD, exposedLiveData)
    }
```

```
}

private fun <T> handleResponse(
            response: ResourceState<T>,
            loadingLiveData: MutableLiveData<Boolean>?,
            destination: MediatorLiveData<T>
    ) {
        when (response.status) {
            Status.SUCCESS -> {
                loadingLiveData?.value = false
                destination.value = response.data
            }
            Status.LOADING -> {
                loadingLiveData?.value = true
            }
            Status.ERROR -> {
                loadingLiveData?.value = false
                networkErrors.value = response.message
            }
        }
    }
```

Livello Dati

Per modellare i dati si è scelto di ricalcare il più possibile l'ontologia **Cultural-ON** [12], prodotta nell'ambito del progetto *dati.beniculturali.it*, al fine di minimizzare la necessità di convertire i dati. Dato il supporto nativo per il tipo `LiveData` per l'implementazione del database è stata scelta la *Room Persistence Library* [6]. È stata creata un'entità (tabella) per quasi ogni classe definita nell'ontologia *Cultural-ON* inclusi `CulturalInstituteOrSite`, `CulturalEvent`, `Site`, `Address`, `Contact Point`, `BookingType`, `Ticket` e altri. Per ogni tabella o entità nel database,

Room necessita di un **Data Access Object** o *DAO* un'interfaccia o una classe astratta dove sono definite le interazioni con le tabelle sotto forma di metodi.

Per decidere se recuperare i dati dal disco o dalla rete è stata implementata (seguendo l'albero decisionale proposto da *Google* [36]) una classe astratta `NetworkDataSource` che fornisce un `MediatorLiveData` che unisce la richiesta di dati dal disco e dalla rete in un solo `LiveData`, fornendo in un primo momento i risultati nella *cache* offline e in seguito i risultati aggiornati della rete.

A questo punto manca solamente un collegamento con il database di *dati.beniculturali.it* e uno con il server di *OpenArtMap*. Data la semplicità delle interazioni con l'*API REST* del server di *OpenArtMap* le chiamate sono state implementate direttamente come metodi dell'interfaccia richiesta da *Retrofit*. Per quanto riguarda l'interazione con l'endpoint *SPARQL*, data la *type-safeness* di *Retrofit*, il discorso è leggermente più complesso. Per minimizzare la duplicazione del codice si è creata una classe astratta `SparqlService` con un metodo per ogni tipo di dato di risposta. Ogni query *SPARQL* necessita di una classe rappresentante la query e una classe rappresentante la risposta. Infine è bastato chiamare lo `SparqlService` con la query adatta all'interno dei callback forniti da `NetworkDataSource`.

Flask

Per l'implementazione dell'applicativo web per la revisione delle contribuzioni, si è scelto **Flask**, un *microframework* web per *Python* basato sulla libreria di templating *Jinja2* e sulla libreria di rete *Werkzeug*. La parola *"micro"* [16] indica che *Flask* mira a mantenere il proprio nucleo semplice ma estendibile: infatti non si occupa di prendere decisioni supponenti al posto del progettista di sistemi - come spesso accade in framework più massicci, i quali forniscono numerose funzionalità di base ma poco estendibili - bensì si occupa di creare un minimo strato di "collante" attorno a *Jinja2* e *Werkzeug*

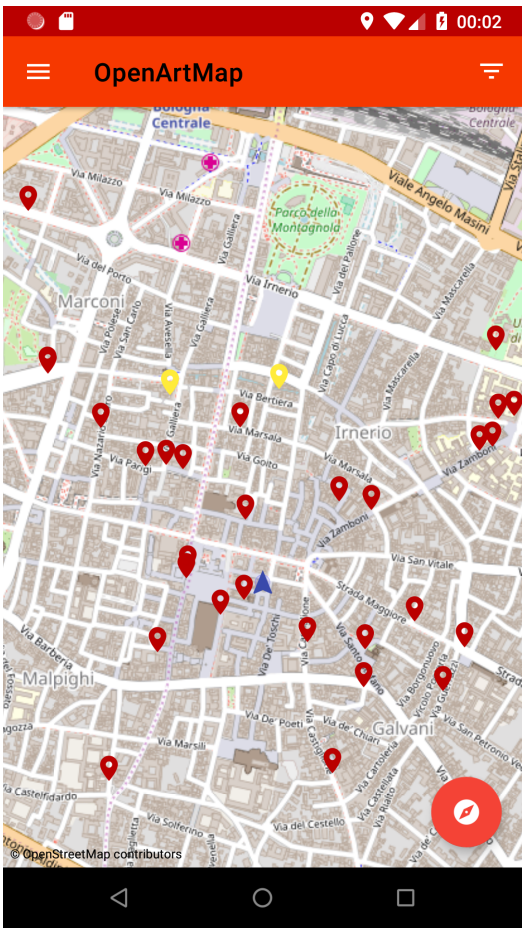

Figura 4.2: `PlacesMapFragment`

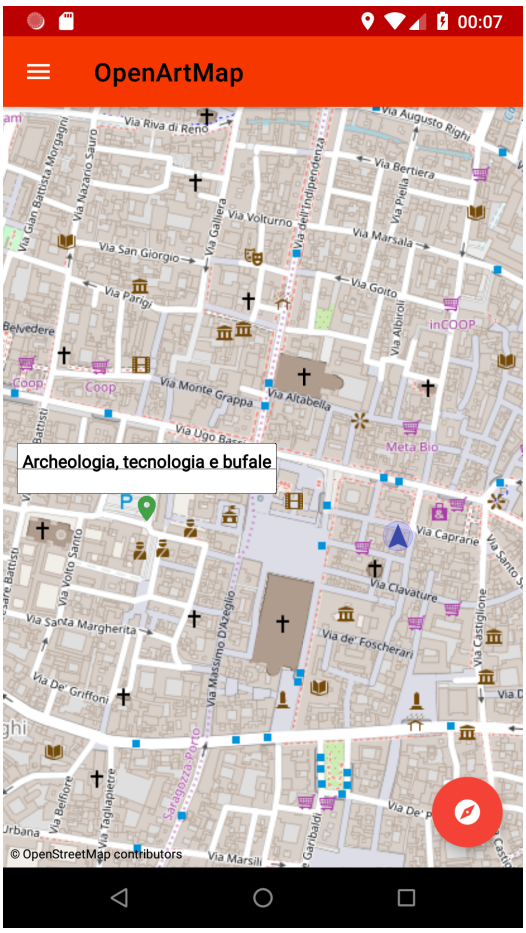

Figura 4.3: `EventsMapFragment`

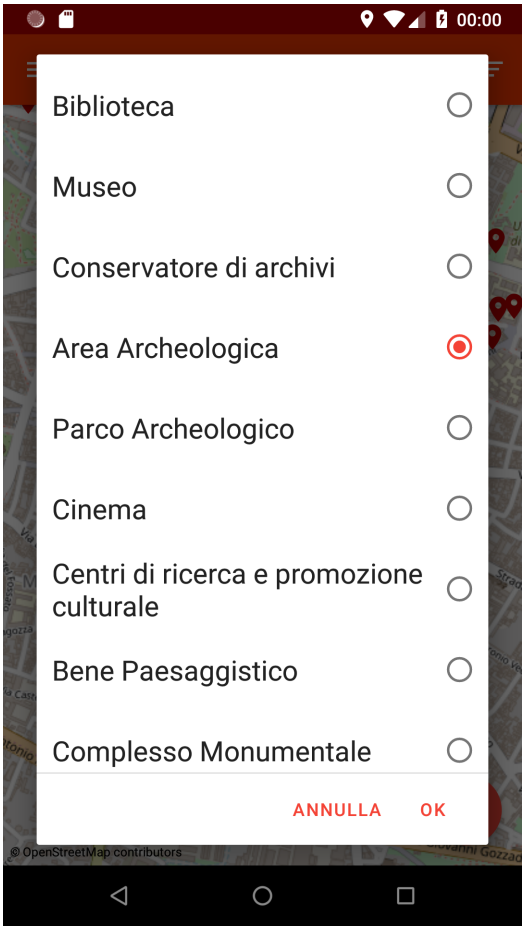

Figura 4.4: `FilterFragment`

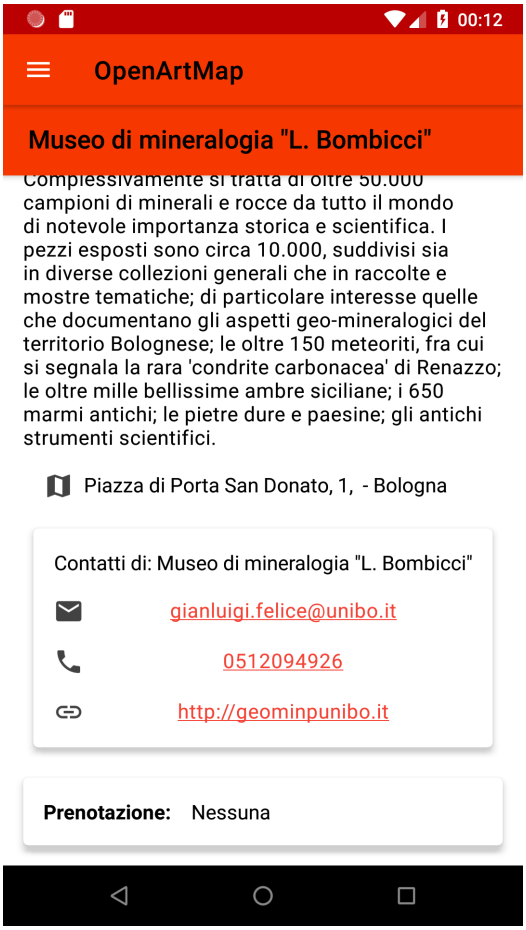

Figura 4.5: `DetailsFragment`

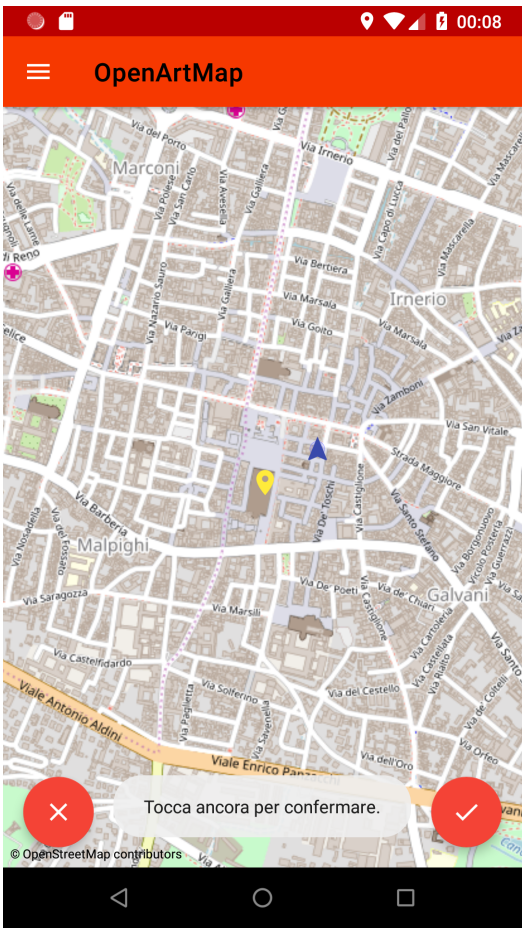

Figura 4.6: `NewPlaceFragment`

Figura 4.7: `EditPlaceFragment`

lasciando al progettista la scelta del *DBMS* o le strategie da adottare nell'ambito della sicurezza o i pattern architetturali più adatti al dominio del problema. Per gli scopi di questo scritto sono state utilizzate le estensioni *Flask-SQLAlchemy* per fornire un'integrazione con l'*ORM* **SQLAlchemy** e *Flask-Admin* che crea un'interfaccia *Admin* sul database, per permettere la modifica delle contribuzioni.

Routes

Data la semplicità dell'applicativo web si è deciso di procedere con uno schema del database a singola tabella con *SQLite*. Ogni endpoint o *route* interroga il database se necessario e ritorna al client la risposta in formato *JSON*. Le *routes* implementate da *OpenArtMap Server* sono:

- `/`: l'index dell'applicativo che redirige alla pagina di *review*. Viene ritornato un file *HTML* che fa da interfaccia di revisione sul database. Supporta solo il metodo `GET`.
- `/changesets`: se riceve una richiesta `POST` valida (ovvero che contiene i dati minimi necessari alla persistenza della contribuzione), inserisce il *payload* della richiesta nel database. Supporta solo il metodo `POST`.
- `/changesets/<int:changeset_id>`: endpoint per il recupero dei dettagli riguardanti una certa contribuzione. `changeset_id` è un intero rappresentante l'*UID* della contribuzione.
  - `/changesets/<int:changeset_id>/address`: sotto-percorso che ritorna in *JSON* i dati riguardanti l'indirizzo di una contribuzione;
  - `/changesets/<int:changeset_id>/contacts`: ritorna i dati riguardo i punti di contatto;
- `/changesets/minimals`: route per il recupero di beni culturali (minimali nel senso che l'API ritorna solo nome, tipo e coordinate) all'interno di una *bounding box*. `north`, `south`, `east`, `west` sono i quattro argomenti obbligatori e indicano le coordinate dei vertici opposti della *bounding box*. Supporta solo il metodo `GET`.

Reviewer UI

L'interfaccia per la modifica e la revisione delle contribuzioni è divisa in due schermate. La schermata principale mostra una tabella con la lista delle contribuzioni presenti nel database. È possibile selezionare una o più contribuzioni per svolgere azioni di gruppo come la rimozione. Cliccando su una contribuzione viene aperta una schemata di dettaglio che mostra la lista dei campi della contribuzione e permette di modificarli singolarmente.

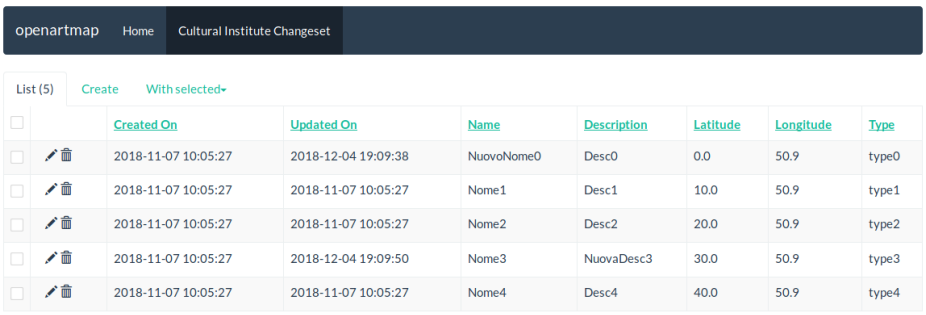

Figura 4.8: *OpenArtMap Server* pagina principale

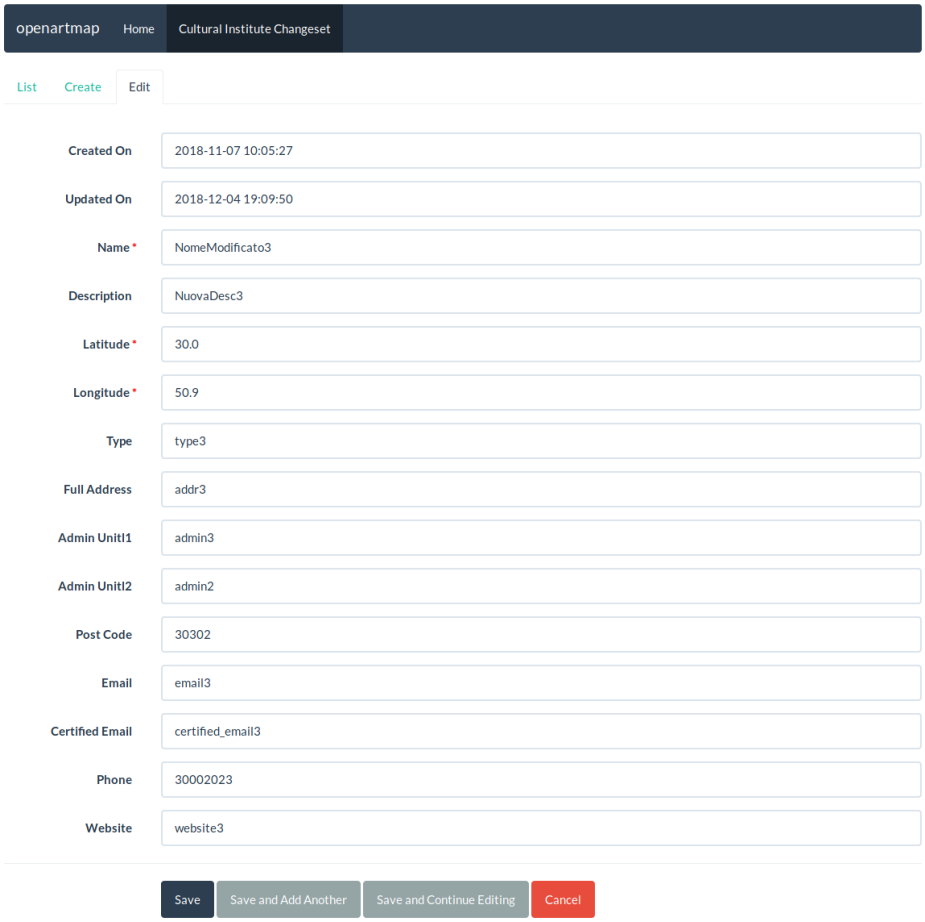

Figura 4.9: *OpenArtMap Server* pagina di revisione

Capitolo 5

Conclusioni

Nei capitoli precedenti è stato presentato *OpenArtMap* un applicativo per la catalogazione collaborativa dei beni culturali. Dopo aver esaminato tre progetti collaborativi di successo ed averne analizzato il *modus operandi* è stato possibile progettare e realizzare un sistema analogo. Ovviamente *OpenArtMap* non è ancora un sistema pronto per essere usato in quanto manca di funzionalità basilari per il sano sviluppo di una comunità collaborativa. In questo capitolo saranno presentati i limiti attuali di *OpenArtMap* e delle possibili modifiche da introdurre per migliorare il sistema.

Limiti

Il principale limite di *OpenArtMap*, è l'assenza di funzionalità di comunicazione interne al sistema. I sistemi di catalogazione collaborativa odierni si fondano su una comunità di volontari che autonomamente organizzano la revisione delle contribuzioni e i criteri per la valutazione delle modifiche. Ognuno di questi compiti necessita di un mezzo di comunicazione neutrale - idealmente integrato nel sistema - che permetta a tutti membri della comunità di contribuire alla discussione.

Un ulteriore limite di *OpenArtMap* è il non sfruttare a pieno le potenzialità offerte dai *Linked Open Data*. La peculiarità dei *LOD* è la possibilità

di modellare relazioni tra risorse, disponibili in due grafi *RDF* "fisicamente" scollegati, ma espresse sotto forma di *URI* risolvibili via *HTTP*. L'endpoint *SPARQL* offerto da *dati.beniculturali.it* supporta le query federate, permettendo di incrociare i dati con altre *knowledge bases* della *Linked Open Data Cloud* [27] come *Wikidata*, *DBpedia* o i dati geografici forniti dall'*Istituto Superiore per la Protezione e la Ricerca Ambientale* [28].

Sviluppi futuri

Al fine di favorire la comunicazione e la discussione delle direttive editoriali, uno dei primi passi potrebbe essere fornire un sistema *wiki-like* che permetta una comunicazione trasparente all'interno della comunità. Nello specifico possono essere creati spazi web integrati con il server delle contribuzioni di *OpenArtMap* in cui gli utenti possano discutere la qualità delle contribuzioni e agire direttamente sul dato; si potrebbe inoltre permettere di inserire e visualizzare commenti alle contribuzioni sulla mappa del client *Android* al fine di migliorare la consapevolezza dei contributori sul campo riguardo ad eventuali contribuzioni critiche o controverse.

Si potrebbe inoltre fornire la possibilità di creare filtri più complessi all'interno del client *Android*, per esempio per mostrare tutti i musei contenenti opere del '700, o tutti i siti archeologici della provincia di Roma.

Infine integrando i dati con altre *knowledge bases* si potrebbe offrire una migliore esperienza fornendo risultati più accurati o completamente impossibili da ottenere con i semplici dati del *MiBAC*: per esempio interrogando l'endpoint *SPARQL* dell'*ISPRA* è possibile ottenere le geometrie delle piante degli edifici presenti nel database, con queste informazioni e quelle ottenibili da altre fonti di *LOD* come *Wikidata* si potrebbe costruire un motore di ricerca *semantico* utilizzabile dalla comunità per completare i dati assenti, parziali o malformati presenti nel sistema.

Bibliografia

[1] *AcousticBrainz*. URL: https://acousticbrainz.org/.

[2] Alexa, cur. *wikipedia.org Traffic Statistics*. Nov. 2018. URL: https://www.alexa.com/siteinfo/wikipedia.org.

[3] *Android Data Binding Library*. URL: https://developer.android.com/topic/libraries/data-binding/.

[4] *Android Jetpack*. URL: https://developer.android.com/jetpack/.

[5] *Android Jetpack Final Architecture*. URL: https://developer.android.com/topic/libraries/architecture/images/final-architecture.png.

[6] *Android Room Persistence Library*. URL: https://developer.android.com/topic/libraries/architecture/room.

[7] *Android Support Library*. URL: https://developer.android.com/topic/libraries/support-library/.

[8] *AndroidX*. URL: https://developer.android.com/jetpack/androidx/.

[9] *Bing Aerial Imagery*. URL: https://wiki.openstreetmap.org/wiki/Bing_Maps.

[10] *CDDB - Compact Disc Database*. URL: https://it.wikipedia.org/wiki/CDDB.

[11] *Creative Commons 0 - Public Domain*. URL: https://creativecommons.org/publicdomain/zero/1.0/.

[12] *Cultural-ON Ontology*. URL: http://dati.beniculturali.it/cis/.

[13] *dati.beniculturali.it*. URL: http://dati.beniculturali.it/.

[14] *dati.beniculturali.it - I Fase Operativa*. URL: http://dati.beniculturali.it/il-progetto/fase-del-progetto-2014-2016/.

[15] *dati.beniculturali.it - II Fase Operativa*. URL: http://dati.beniculturali.it/il-progetto/ii-fase-del-progetto-2017-2018/.

[16] *Flask, a Foreword*. URL: http://flask.pocoo.org/docs/1.0/foreword/.

[17] *Google Street View*. URL: https://www.google.com/streetview/.

[18] Mordechai Haklay. «How good is volunteered geographical information ? A comparative study of OpenStreetMap and Ordnance Survey datasets». In: 2008.

[19] Jess Hemerly. «Making metadata: The case of MusicBrainz». In: (2011).

[20] *Humanitarian OpenStreetMap Team*. URL: https://www.hotosm.org/.

[21] *iD Editor*. URL: http://ideditor.com/.

[22] *Information Technology for Humanitarian Assistance, Cooperation and Action*. URL: http://ithacaweb.org/.

[23] *Istituto Centrale per gli Archivi*. URL: http://www.icar.beniculturali.it/.

[24] *Istituto Centrale per il Catalogo e la Documentazione*. URL: http://iccd.beniculturali.it/.

[25] *Kotlin Standard Functions*. URL: https://github.com/JetBrains/kotlin/blob/master/libraries/stdlib/src/kotlin/util/Standard.kt.

[26] Karim Lakhani e Robert G. Wolf. «Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects». In: *Perspectives on Free and Open Source Software* (set. 2003). DOI: 10.2139/ssrn.443040.

[27] *Linked Open Data Cloud*. URL: https://www.lod-cloud.net/.

[28] *Linked Open Data ISPRA*. URL: http://dati.isprambiente.it/.

[29] *ListenBrainz*. URL: https://listenbrainz.org/.

[30] *Mapillary*. URL: https://www.mapillary.com/.

[31] *MetaBrainz*. URL: https://metabrainz.org/.

[32] Peter Mooney, Padraig Corcoran e Adam C. Winstanley. «Towards quality metrics for OpenStreetMap». In: *GIS*. 2010.

[33] *MusicBrainz Picard*. URL: https://picard.musicbrainz.org/.

[34] *MusicBrainz Social Contract*. URL: https://metabrainz.org/social-contract.

[35] Pascal Neis e Alexander Zipf. «Analyzing the Contributor Activity of a Volunteered Geographic Information Project - The Case of OpenStreetMap». In: *ISPRS Int. J. Geo-Information* 1 (2012), pp. 146–165.

[36] *NetworkBoundResource's Decision Tree*. URL: https://developer.android.com/topic/libraries/architecture/images/network-bound-resource.png.

[37] *OpenPoiMap*. URL: http://openpoimap.org/.

[38] *OpenStreetMap - Mapping Techniques*. URL: https://wiki.openstreetmap.org/wiki/Pick_your_mapping_technique.

[39] *OSMdroid*. URL: http://osmdroid.github.io/osmdroid/.

[40] *Project Haiti*. URL: https://wiki.openstreetmap.org/wiki/WikiProject_Haiti.

[41] *Sistema Archivistico Nazionale*. URL: http://san.beniculturali.it/.

[42] *Sistema Informativo Generale del Catalogo*. URL: http://catalogo.beniculturali.it/.

[43] Robert Soden e Leysia Palen. «From Crowdsourced Mapping to Community Mapping: The Post-earthquake Work of OpenStreetMap Haiti». In: *COOP*. 2014.

[44] Ugaya40. *Pattern Architetturale MVVM*. 2018. URL: https://commons.wikimedia.org/w/index.php?curid=19056842.

[45] W3C. *RDF Standards*. URL: https://www.w3.org/standards/techs/rdf#stds.

[46] W3C. *SPARQL Query Language*. Mar. 2013. URL: https://www.w3.org/TR/sparql11-query/.

[47] *Wikibooks*. URL: https://www.wikibooks.org/.

[48] *Wikidata*. URL: https://www.wikidata.org/wiki/Wikidata:Main_Page.

[49] *Wikimedia Commons*. URL: https://commons.wikimedia.org/wiki/Main_Page.

[50] Wikipedia, cur. *Progetto GLAM*. URL: https://it.wikipedia.org/wiki/Progetto:GLAM.

[51] Wikipedia, cur. *Statistiche*. Nov. 2018. URL: https://wikipedia.org/wiki/Special:Statistics.

[52] *World Food Program*. URL: https://wfp.org.