ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

MOBILE CITY, LA TUA CITTÀ IN TASCA

Relazione finale in
MOBILE WEB DESIGN

Relatore
MIRKO RAVAIOLI

Presentata da
MATTEO BOCCHETTI

I Sessione
Anno Accademico 2012 - 2013

Introduzione

L'avvento di internet ha rivoluzionato, insieme a tanti altri, anche il settore del turismo e sta drasticamente cambiando le relazioni commerciali tra "imprese turistiche" e clienti: i confini geografici, che limitavano l'accesso a beni e servizi stanno a mano a mano scomparendo.
Spesso il turismo elettronico viene erroneamente identificato con la compravendita di viaggi e biglietti su Internet. In realtà esso è molto di più in quanto implica una filosofia completamente nuova: il marketing turistico del nostro secolo infatti vive di reputazione e non più di immagini, si costruisce su idee forti e innovative, deve essere ricco di contenuti di qualità.
Alla base delle nuove strategie di marketing per le "imprese turistiche" c'è la consapevolezza che il nuovo consumatore/turista è autonomo, più critico e competente, è molto più informato sulla composizione di prodotti, servizi e sul rapporto qualità/prezzo; richiede sempre di più qualità e attenzione alle sue esigenze. Ha una particolare attenzione al mondo virtuale ed è sempre alla ricerca di contenuti e dell'opportunità di usufruire di servizi in rete anche grazie a dispositivi come gli smartphone che offrono pratici strumenti per tutti gli utenti che amano viaggiare.
Per affermarsi in questo scenario competitivo, ciascuna azienda turistica deve munirsi di nuove tecnologie: ad esempio molte città, province e regioni italiane stanno sviluppando guide turistiche del nuovo millennio, applicazioni per smartphone di facile utilizzo, pratiche e gratuite in grado di accompagnare i turisti in visita alla città fornendo tutte le informazioni utili, con la speranza di riuscire ad avere un'affluenza sempre maggiore di visitatori. Si tenta di rendere la propria città/provincia/regione "facile" da conoscere, da scoprire, da vivere e più fruibile e appetibile per chi la visita: oltre alla posizione e alle informazioni relative a monumenti, musei e altre mete turistiche vengono anche messe a disposizione tramite l'applicazione tutte le informazioni relative al luogo, alle sue eccellenze, agli eventi che vi si svolgono, ecc.
Il lavoro di questa tesi ha l'obiettivo di completare di una guida turistica per smartphone Android della città di Cesena dal nome "Live Tour" che contiene mappe, luoghi di interesse, ristoranti, pub, bar, hotel e altro ancora, presenti in questa città.

1. Dispositivi mobili

Nell'ultimo decennio, da programmi desktop in esecuzione sui diversi PC si è passati ad applicazioni web accessibili ovunque attraverso dispositivi sempre più piccoli e potenti: i dispositivi mobili.

I dispositivi mobili sono generalmente dotati, in relazione ai dispositivi di elaborazione fissi, di basse capacità elaborative e di ridotta memoria. Essi possono essere classificati in varie categorie.

1.1. PDA (Personal Digital Assistant)

Altrimenti detti Palmari, sono computer a batteria che grazie alle loro dimensioni ridotte possono essere trasportati e utilizzati ovunque. Sono strumenti utili alla pianificazione degli appuntamenti, alla memorizzazione in rubrica di numeri telefonici e indirizzi e altre funzioni, ma non offrono prestazioni comparabili a quelle dei PC. Alcuni modelli più avanzati consentono di telefonare o accedere a Internet. Al posto della tastiera, i palmari hanno uno schermo "touch screen", che si utilizza con le dita o con uno stilo.

1.2. Tablet PC

È simile ad un computer portatile ma di dimensioni minori, paragonabili a quelle di un foglio A5 con spessore di qualche millimetro e sprovvisto di tastiera. Sul lato superiore del dispositivo è presente un display LCD con interfaccia touch che ha consentito di ridurre peso e dimensioni della parte hardware eliminando mouse e tastiera: l'utente infatti può impartire comandi al sistema in modo semplice ed intuitivo, semplicemente toccando con le proprie dita le icone desiderate.

1.3. Smartphone

Telefono cellulare con funzioni e potenzialità di un computer palmare in grado di operare con un sistema operativo completo ed autonomo. Questi dispositivi sono in grado di acquisire e trattare immagini, gestire dati comples-

si, avviare e gestire comunicazioni in rete sfruttando connessioni UMTS, la tecnologia WI-FI, Bluetooth o IrDa.

1.3.1. *Breve storia degli smartphone*

Il 1993 è stato l'anno in cui il mondo ha visto il primo smartphone. E 'stato quando il Simon IBM è stato rilasciato, e aveva App minori. Aveva un touchscreen utilizzabile con una stilo e prestava funzione di fax, PDA, cercapersone, blocco note, client di posta e calendario. L'evoluzione da allora è stata nient'altro che sorprendente. IBM aveva grandi progetti per il Simon, ma non ha mai avuto tanto successo in quanto probabilmente era troppo avanzato per il suo tempo [2].

Figura 1 - Simon IBM

Ericsson è stato il primo marchio a coniare il termine "smartphone", con il rilascio del suo GS88 nel 1997, ma Nokia supera il successo di Ericsson dell'anno precedente con quello che molti considerano essere lo smartphone per eccellenza, il Nokia 9000.

Figura 2 - Nokia 9000

Ericsson R380 è stato il primo telefono ad essere fornito del sistema operativo Symbian.

Figura 3 - Ericsson R380

Nel 2001 Kyocera rilascia il modello 6035 con Palm OS che ebbe molto successo grazie alle caratteristiche di base dello smartphone con la quale venne prodotta il resto della linea di Palm.

Figura 4 - Treo Handspring

Tra il 2002 e oggi i dispositivi sono drasticamente cambiati, ma i sistemi operativi sono rimasti sostanzialmente inalterati, almeno fino al 2007.

Il primo cambiamento reale visto sul mercato è avvenuto quando Apple Inc. ha introdotto l'iPhone. Fino a quel momento lo smartphone era sempre stato considerato come strumento adatto a uomini d'affari: Apple ha fatto un lavoro magnifico rendendolo come qualcosa che un consumatore dovrebbe desiderare.

Inizialmente, tuttavia, l'iPhone aveva il "difetto" di non dare la possibilità agli utenti di installare applicazioni create da terzi.

Figura 5 - iPhone 1

Nel 2008, HTC fu il primo produttore a dotare i suoi telefonini di Android, che diventerà il rivale di Apple. Poco dopo molte altre case produttrici di smartphone (tra cui Samsung, LG, Motorola e molti altri) mostrarono attenzione per il nuovo sistema operativo.

Figura 6 - HTC Dream

Il primo vero grande successo di Android, relativamente al numero di dispositivi venduti, si avrà nel 2009 con Motorola Droid, che supera il milione di dispositivi venduti ad un mese dalla data del lancio.

Seguiranno poi nuovi prodotti sempre più sofisticati anche da parte di Samsung e della stessa HTC che consolideranno la collaborazione con Google.

Nel 2010 anche Windows mette a punto un nuovo sistema operativo, Windows Phone 7, che in Italia si conoscerà per la prima volta grazie a LG, con il suo LG Optimus 7.

Figura 7 - LG Optimus 7

Nel 2012 la corsa al primato nel mercato mobile non si è ancora arrestata e le nuove proposte riguardano prodotti sempre più sottili, veloci, "intelligenti" e ricchi di funzionalità.

A scapito di quasi tutti i produttori, tuttavia, gli smartphone sono anche un terreno fertile per cause legali frivole. Dal 2009 ad oggi, infatti, ci sono stati oltre 100 casi degni di nota riguardanti le funzionalità più svariate: dalle gestures multitouch alle tecnologie 3G. È interessante notare che la ditta che si è evitata la maggioranza delle cause è stata Microsoft con il suo prodotto Windows Phone.

1.3.2. *Store e Market*

Il fattore determinante del successo degli smartphone nel corso degli anni, è stata non solo la loro capacità di funzionare bene, ma anche la possibilità di avere funzionalità estese tramite applicazioni create da terzi tramite dei veri e propri "negozi" in cui è possibile acquistare e scaricare app (come ad esempio Apple App Store e Android Market di Google).

Figura 8 - Store per applicazioni smartphone

2. Sistemi operativi per dispositivi mobili

Quasi tutte le case costruttrici di dispositivi mobili affidano la gestione dei loro dispositivi nelle mani di sistemi operativi non proprietari. Un sistema operativo per dispositivi mobili, in inglese mobile OS, è un insieme di componenti software progettati per il controllo del dispositivo con lo stesso principio con cui Mac OS, Linux o Windows controllano un desktop computer, che si trova però a dover far fronte a problematiche più critiche come ad esempio la limitatezza delle risorse (memoria,CPU), l'assenza di alimentazione esterna, differenti protocolli di trasferimento dati per l'accesso a internet (WiFi, GPRS, HSDPA...), nuovi metodi d'immissione (touchscreen, minitastiere) e le ridotte dimensioni del display, che sono dovute alla natura del dispositivo mobile.

2.1. Android

"[...] *Android è la prima piattaforma veramente aperta e globale per dispositivi mobili. Essa comprende un sistema operativo, un'interfaccia utente e applicazioni: tutto il software per far funzionare un telefono cellulare, ma senza gli ostacoli all'innovazione del mondo mobile dovuti ai sistemi proprietari*". Questa è la definizione data da Andy Rubin uno dei fondatori di Android Inc., azienda acquisita da Google Inc. nel 2005. La prima versione su smartphone è stata rilasciata il 23 settembre 2008 da Google in collaborazione con l'Open Handset Alliance.

Questo sistema operativo ha cambiato in poco tempo il panorama della telefonia mobile trasformando il cellulare in un vero e proprio computer tascabile [7]. Se l'iPhone ha creato il moderno mercato degli smartphone, è senza dubbio Android che l'ha portato al boom.

È un sistema operativo totalmente open source infatti tutti gli aspetti del sistema operativo Android, da quelli più superficiali a quelli più critici possono essere modificati liberamente [4].

È basato sul kernel Linux e consiste in una struttura formata da vari livelli o layer, ognuno di essi fornisce al livello superiore un'astrazione del sistema sottostante.

Tra i suoi punti di forza c'è quello di poter essere installato su una vasta gamma di dispositivi, non solo smartphones ma anche tablet, e-book reader e netbook di diversi produttori. Questo ha favorito in breve tempo una rapida distribuzione e l'ha reso, oggi giorno, il sistema operativo più diffuso.

Le sue versioni sono nominate ufficialmente con un numero di versione ma vengono distinte per il proprio "codename" (ispirato alla pasticceria e rigorosamente in ordine alfabetico): Cupcake (Android 1.5), Donut (1.6), Eclair (2.0/2.1), Froyo (2.2), Gingerbread (2.3/2.4), Honeycomb (3.0), Ice Cream Sandwich...[5]

Android Market (denominato ora Google Play) è lo store di applicazioni online che permette all'utente il download di tantissime applicazioni, anche scritte da sviluppatori di terze parti.

2.2. BlackBerry

BlackBerry OS è un sistema operativo mobile proprietario, sviluppato da Research In Motion per la sua linea di smartphone BlackBerry nel linguaggio di programmazione C ++. È stato inizialmente pensato per le aziende.

Da quando è stato rilasciato, la sua popolarità alle stelle grazie alle seguenti funzioni [3]:

- possibilità di inviare e ricevere posta elettronica Internet con il metodo *push* di consegna;
- supporta la trasmissione fax via Internet e la navigazione Web;
- supporta la visualizzazione di applicazioni di Office;
- capacità di supportare numerosi altri servizi di informazione wireless;
- caratteristiche di multitasking;
- trackball, trackpad e touchscreen.

2.3. iOS

Il 29 giugno del 2007 veniva messo in commercio il primo modello di iPhone, funzionante con quello che veniva definito ancora OS X. L'attenzione principale degli sviluppatori era stata riposta nello sviluppo di un'interfaccia innovativa, semplice e immediata da usare senza l'ausilio di uno stilo.

Da OS X deriva il più noto iOS che, come OS X, è un sistema operativo Unix.

La sua nascita è conseguente allo sviluppo dell'iPhone ed è stato poi utilizzato sui dispositivi portatili dotati di touch screen della Apple. È ottimizzato per i dispositivi su cui viene fatto girare.

Una delle caratteristiche che lo distingue dai sistemi operativi similari, funzionanti su dispositivi di altre case, è la "chiusura" nei confronti delle applicazioni che è possibile installare.

Questo sistema operativo ha il merito di aver dato il via alla rivoluzione del settore mobile: iOS è stato il capostipite assoluto della nuova generazione di sistemi operativi mobile Touch-based e App-based, introducendo tramite l'App Store il concetto di "negozio online" da cui è possibile scaricare ed installare nuove applicazioni, gratuite o a pagamento, per iPhone, iPod touch e iPad [6].

2.4. Symbian

Symbian OS è un sistema operativo per smartphones prodotto da Symbian Foundation ed è il sistema operativo dei cellulari Nokia più vecchi. Fino a pochissimi anni fa era il più diffuso vista l'enorme popolarità dei dispositivi Nokia nel mondo.

Nasce nel 1998 con la creazione della compagnia Symbian Limited.

L'arrivo dell'iPhone di Apple ha rivoluzionato il settore e a subirne maggiormente le conseguenze è stato proprio il sistema Symbian.

Nonostante una serie di finanziamenti europei e l'apertura all'opensource non è riuscito a rimanere al passo con gli altri sistemi operativi e sta, quindi, lasciando spazio a Windows, in virtù di un accordo intercorso tra Nokia e Microsoft.

Gli aggiornamenti più recenti l'hanno portato al passo con gli altri concorrenti, permettendo l'aggiunta di numerose applicazioni e giochi, la visualizzazione e modifica di doucmenti, ecc [7].

2.5. Windows Phone

Nel 2010 Microsoft ha rinnovato il suo sistema operativo mobile introducendo Windows Phone 7. WP7 ha un'interfaccia utente molto semplice,

composta da tiles, cioè icone quadrate che rimandano alle applicazioni. Le applicazioni principali sono sostanzialmente le versioni mobile dei software Microsoft per computer: si trovano quindi Internet Explorer per navigare sul web, Outlook per la posta, Zune per la musica, Xbox Live per i giochi, Bing per la ricerca e le mappe, Office per i documenti.

Nel 2011 Microsoft e Nokia hanno siglato un accordo per portare il sistema operativo americano sui cellulari finlandesi, puntando così a creare terzo colosso nel settore smartphone, dopo Apple e Google, cercando di scavalcare RIM con i suoi Blackberry.

2.6. Il mercato degli smartphone

Gli ultimi dati della fine del 2012 riportati dalla IDC (una celebre azienda di analisi di mercato) rilevano record di vendite per le due piattaforme leader Android e iOS grazie alle enormi vendite di Samsung e di Apple, mentre segnalano cali di vendite per altri grandi produttori come HTC e RIM.

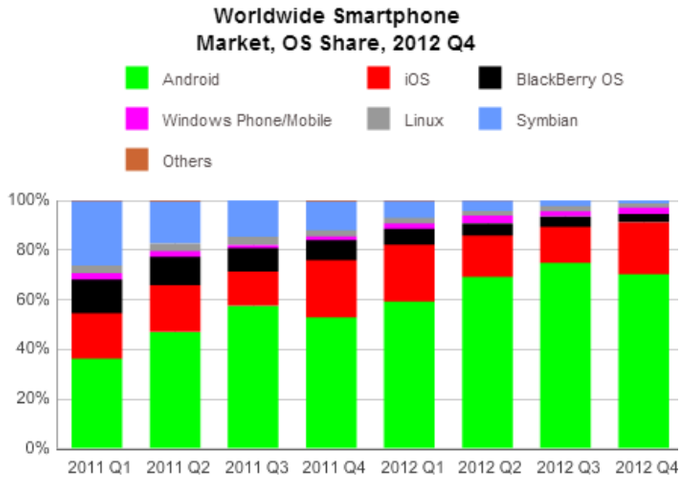

Figura 9 - Mercato degli smartphone 2011 – 2012 [8]

Esiste di fatto quindi un duopolio Android-iOS, che insieme, come si vede dal grafico, nel primo trimestre del 2012 detenevano l'87,6% del mercato smartphone: Android ha avuto una quota di mercato del 68,8%, al secondo posto iOS col 18,8% (Android possiede la maggior parte della quota di mercato mentre iOS registra la maggioranza delle entrate). Subiscono perdite di anno in anno BlackBerry e Symbian che si piazzano rispettivamente al terzo e quarto posto, mentre Windows Phone, che ha quasi raddoppiato le spedizioni, arriva al 2.5%.

I numeri per il quarto trimestre sono ancora più drammatici per i concorrenti poichè Android e iOS salgono al 91,1%. BlackBerry riconferma il terzo posto e Windows Phone sale al quarto posto grazie all'aumento delle vendite pari al 150%.

Le previsioni per il prossimo futuro riportate da ABI Research [9] sono all'incirca come quelle registrate fin'ora: gli **smartphone Android** presenti sul mercato ed attivi potrebbero essere circa **800 milioni** pari al **57%** del mercato globale (percentuale inferiore a quelle riportate nel 2012). Al secondo posto dovrebbe mantenersi iOS con il 21% (294 milioni di smartphone) mentre a giocarsi il terzo posto saranno Windows Phone e BlackBerry (che dovrebbero avere rispettivamente attivi 45 e 20 milioni di smartphone alla fine del 2013).

3. Studio e analisi dei componenti

Il lavoro svolto per il progetto di questa tesi si è articolato in più fasi:

- Analisi dell'applicazione per Android "Live Tour"; (cfr 3.1 – Live Tour)
- Studio dello standard JSON e analisi dei dati in questo formato da inserire nell'applicazione; (cfr 3.2 – JSON)
- Analisi strutturale del database ottimale per l'applicazione. (cfr 3.3 – Database).

3.1. Live Tour

"Live Tour" è un applicazione classificabile come location-based [10]. In questa categoria ricadono tutte quelle applicazioni che interagiscono con la posizione dell'utente e dei luoghi di suo interesse. Un esempio tra i più lampanti è il navigatore satellitare, il quale usa la posizione dello smartphone per condurlo fino al punto di interesse. Allo stesso modo "Live Tour" fornisce all'utente la possibilità di sapere non solo la sua posizione ma anche la posizione dei POI (Point Of Interest) che si trovano nelle immediate vicinanze.

Questi punti vengono disegnati sulla mappa solo se si trovano nel raggio di visualizzazione di grandezza pari a quella che l'utilizzatore sceglie attraverso uno slider posto in fondo alla schermata. L'utente può decidere di visualizzare la mappa in 4 diverse viste messe a disposizione da Google Maps:

- vista normale: vengono visualizzate le strade con i relativi nomi e i nomi delle città;
- vista da satellite: viene visualizzata la fotografia scattata da satellite, dove è possibile vedere non solo le strade ma anche gli edifici, spazi pubblici, ecc. ecc.;
- vista ibrida: è una visualizzazione che vede unite le caratteristiche della vista da satellite e della vista normale;
- vista terrestre: viene visualizzata la conformazione morfologica del territorio.

Oltre alla scelta del tipo di visualizzazione l'utente, agendo sui filtri dei POI, può rendere visibile una o più categorie.

Ogni POI è segnalato sulla mappa da un segnaposto, identico per ogni POI appartenente alla stessa categoria. Cliccando su un punto di interesse vengono fornite delle informazioni sommarie come ad esempio una piccola foto, il nome e la distanza rispetto alla posizione dell'utente. Cliccando ulteriormente sullo stesso POI si ha accesso alla scheda dettagliata in cui vengono visualizzate le seguenti informazioni: descrizione con possibilità di lettura automatica, telefono, indirizzo, mail, sito-web e una galleria costituita da immagini e video.

In questa schermata l'utente può anche condividere il proprio commento sul luogo attraverso i social network più famosi: Facebook e Twitter.

I punti di interesse possono anche essere visualizzati sotto forma di lista, dove a differenza di prima, è possibile selezionarne più di uno per volta, per creare un tour guidato.

Questa opzione fornisce all'utente la mappa con un percorso itinerante tra i POI selezionati.

Un'ultima modalità di visualizzazione dei POI è tramite la realtà aumentata.

3.2. JSON

JSON (acronimo di JavaScript Object Notation) è uno standard aperto basato sul testo e adatto per lo scambio di dati in applicazioni client-server. La sua nascita è dovuta al grande sviluppo che in questi anni ha investito il web 2.0, termine con cui si indica l'insieme di tutti quegli applicativi, usufruibili attraverso il web, che permettono un elevata interazione tra il sito web e l'utente.

Nell'ambito del web 2.0 uno dei protagonisti assoluti è il linguaggio di programmazione JavaScript, molto presente e diffuso in numerose applicazioni web grazie alle sue chiamate asincrone, volte allo scambio di dati con il server. Lo scambio dei dati avviene appunto, nella maggior parte dei casi, secondo il formato standard JSON.

3.2.1. *Struttura del file JSON*

Questo formato ha trovato fin da subito larga applicazione per la sua semplice sintassi, di facile lettura e comprensione, che si rifà alla notazione di JavaScript per rappresentare dati strutturati.

Il formato JSON si basa su due strutture comuni a tutti i linguaggi di programmazione ovvero una collezione di coppie nome e valore e una lista ordinata di valori. La prima è presente nei vari linguaggi di programmazione sotto forma di oggetto, array, hash table e così via; la seconda invece come array, vettore, sequenza e così via. Questo è il maggior punto di forza di questo formato in quanto risulta fruibile dalla totalità dei linguaggi.

Citando testualmente lo standard JSON [11], queste strutture rispettano la seguente sintassi:

"*Un oggetto è una serie non ordinata di nomi/valori. Un oggetto inizia con { (parentesi graffa sinistra) e finisce con } (parentesi graffa destra). Ogni nome è seguito da: (due punti) e la coppia di nome/valore sono separata da , (virgola)*".

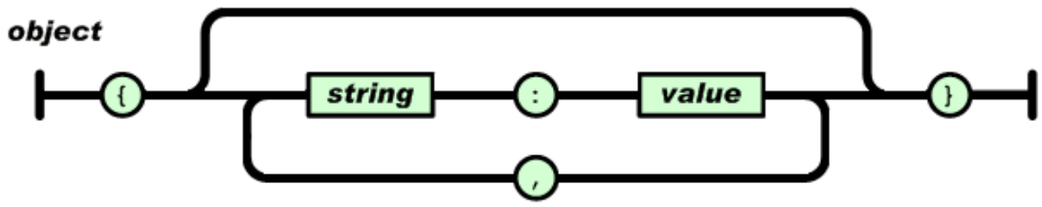

Figure 10 - JSON object

"Un array è una raccolta ordinata di valori. Un array comincia con [(parentesi quadra sinistra) e finisce con] (parentesi quadra destra). I valori sono separati da , (virgola)."

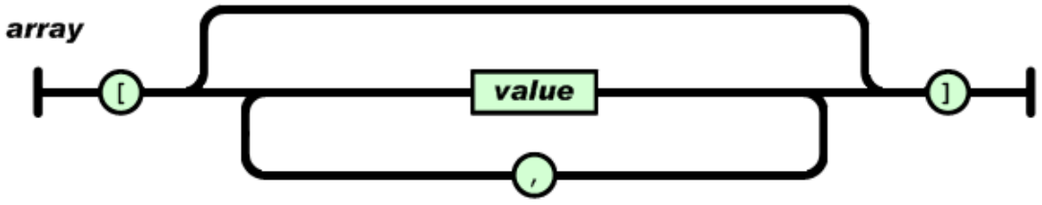

Figure 11 - JSON array

"Un valore può essere una stringa tra virgolette, o un numero, o vero o falso o nullo, o un oggetto o un array. Queste strutture possono essere annidate".

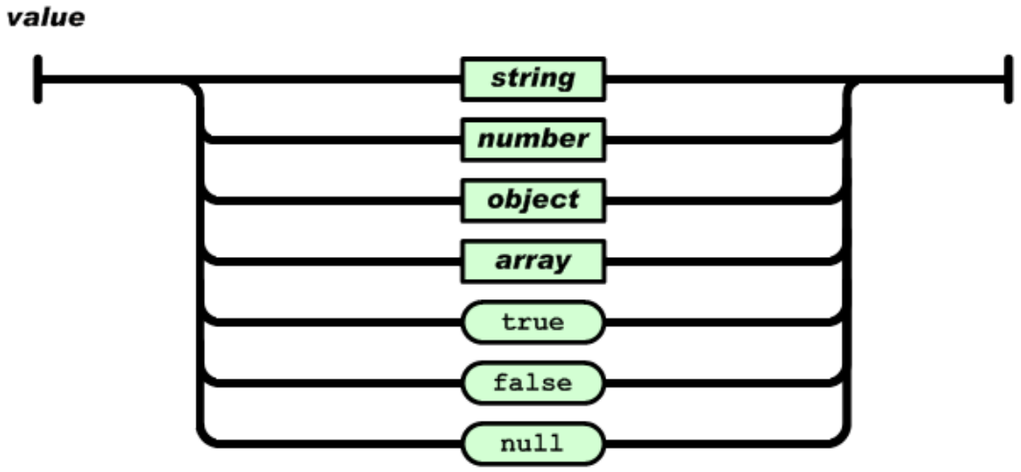

Figure 12 - JSON value

"Una stringa è una raccolta di zero o più caratteri Unicode, tra virgolette; per le sequenze di escape utilizza la barra rovesciata. Un singolo carattere è rappresentato come una stringa di caratteri di lunghezza uno. Una stringa è molto simile ad una stringa C o Java".

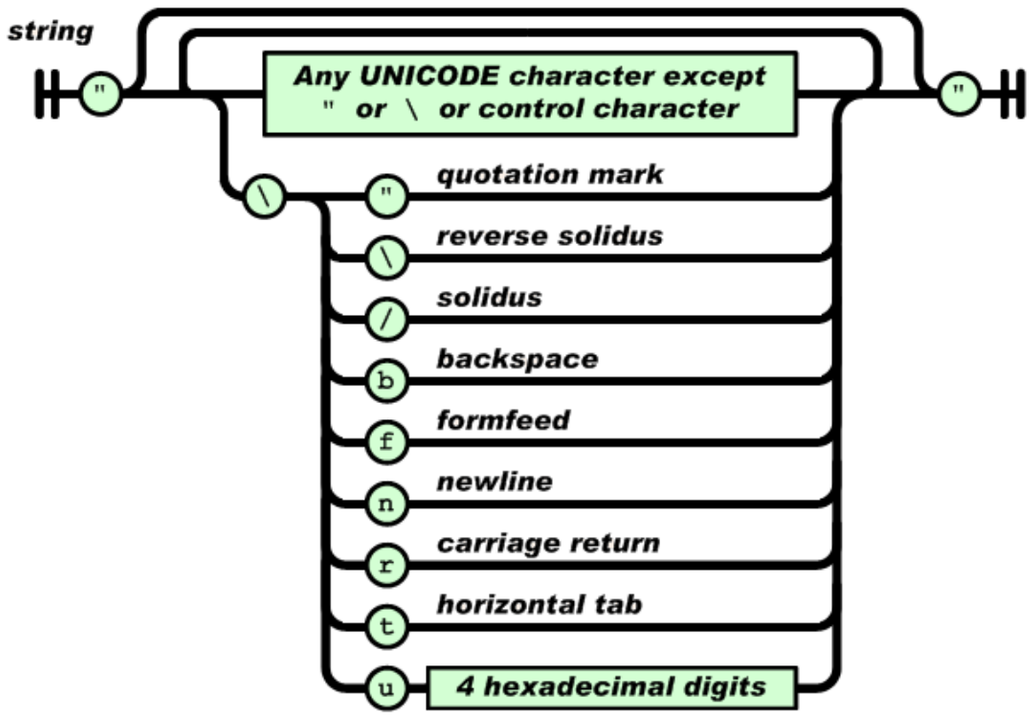

Figure 13 - JSON string

"Un numero è molto simile ad un numero C o Java, a parte il fatto che i formati ottali e esadecimali non sono utilizzati".

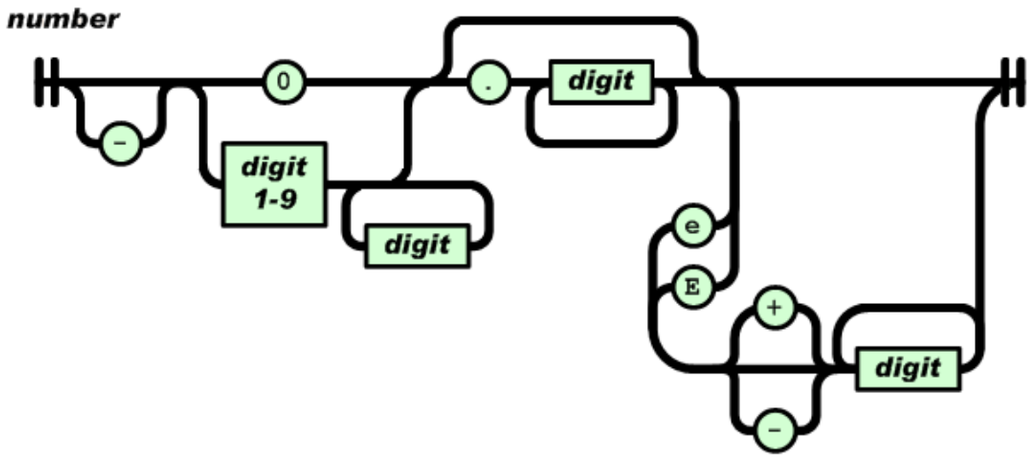

Figure 14 - JSON number

3.2.2. *Vantaggi del JSON*

Il formato JSON presenta numerosi vantaggi: di seguito sono illustrate le caratteristiche che lo hanno portato ad essere uno standard così affermato, scalzando anche formati ben più noti.

Un grosso vantaggio che esso presenta rispetto ai formati binari è la leggibilità, caratteristica che rende più facile il lavoro in fase di sviluppo e di debug.

Numerosi sono anche i vantaggi rispetto ai formati testuali, ovvero stessa famiglia del formato JSON. Confrontandolo con lo standard più noto tra quelli testuali, XML, si notano diversi vantaggi.

XLM (sigla di eXtensible Markup Language) è un standard molto affermato che fa parte dei formati testuali. Come si può dedurre dal nome, si tratta di un linguaggio marcatore ("M" di "Markup") estensibile ("X" di "eXtensible"), in quanto la sua sintassi a markup è volta all'inserimento di tag personalizzati e strutture nidificate, pensate e progettate secondo le esigenze di ogni singolo caso; proprio per questo è versatile e facile da manipolare [12] .

Questi vantaggi vengono tuttavia meno con l'aumentare dei tag. Quando questi ultimi diventano troppo numerosi, il documento tende ad appesantirsi a discapito di vantaggi sopra elencati quali comprensione e la facilità di manipolazione.

JSON, come mostrato nel paragrafo precedente, presenta una struttura molto semplice e risulta perciò molto leggibile e di immediata comprensione, vantaggio non trascurabile, soprattutto quando le dimensioni del file che si sta gestendo iniziano a diventare considerevoli.

L'assenza di tag, oltre a migliorarne la leggibilità, riduce considerevolmente la dimensione del file JSON rispetto alla controparte xml.

Ma il vero punto di forza di JSON consiste, paradossalmente, proprio nella ridotta flessibilità della struttura rispetto alle possibilità offerte da XML. Ad esempio entrambi forniscono la rappresentazione per gli array, tuttavia con JSON è possibile usare indici per muoversi all'interno di un array ed accedere direttamente un preciso record in un determinata posizione, cosa non possibile in XML [13].

3.2.3. *Studio dei dati JSON lato server*

Si vuole creare una funzione che interfacci l'applicativo con il server, nel quale risiedono le informazioni che vengono fornite da una pagina PHP, secondo lo standard JSON, dai seguenti URL:

1. http://www.citylivetour.com/mobile/cesena/check.php
2. http://www.citylivetour.com/mobile/cesena/entita/update.php

Le informazioni sono state divise in due link per evitare che lo smartphone scarichi dati inutili e sovraccarichi le risorse. Nel primo URL sono contenute le informazioni relative alla versione del database, nel secondo il contenuto dell'intero database. Effettuando la richiesta a "check.php" si può controllare la versione attuale del database e, solo nel caso in cui la versione presente sullo smartphone non sia la più recente, allora viene inviata la richiesta a "update.php" per il recupero dei dati.

Sono ora analizzati nel dettaglio i dati visualizzati digitando i due URL. Di seguito il testo restituito da "check.php".

```
http://www.citylivetour.com/mobile/cesena/check.php

{
	"data":{
		"module":"cesena",
		"version":1368808222
	},
	"message":"",
	"success":true
}
```

Si può notare che il primo carattere è { (parentesi graffa sinistra) ad indicare che inizia un oggetto all'interno del quale sono dichiarate le seguenti coppie nome/valore:

- "data": è a sua volta un oggetto costituito da due coppie nome/valore:
  - "module": utilizzato per il nome del database;
  - "version": utilizzato per indicare la versione del database che risiede sul server.
- "message": in questo caso specifico non è stato assegnato nessun valore a questo campo, in generale utilizzato per mandare un messaggio
- "success": indica se la procedura è andata a buon fine o meno.

Di seguito invece un estratto di ciò che viene visualizzato digitando il secondo URL: "update.php"

```
http://www.citylivetour.com/mobile/cesena/update.php
{
	"data":{	...

		"categoria-poi":[
			{	"id":"1",
				"id_utente":"1",
				"data_modifica":1366108379,
				"codice":"-",
				"ordine":0,
				"abilitato":true,
				"icona":"",
				"nome":"Luoghi di culto"		},
			...
		]

		...

		"poi":[
			{	"id":"2",
				"id_utente":"1",
				"data_modifica":1367311055,
				"codice":"-",
				"ordine":"5",
				"abilitato":true,
				"icona":{
					"id":"238",
					"file":"238.jpg",
					"alt":"Abbazia Santa Maria del Monte",
					"name":"C:\\fakepath\\MONTE 01.jpg",
					"ext":"jpg"		},

				"categoria":"43",
				"gallery":[
					{	"image":{
							"id":"87",
							"file":"87.jpg",
							"alt":"",
							"name":"MONTE 02.jpg",
							"ext":"jpg"},
							...
					}
				],
				"latitudine":"44.13184819691395",
				"longitudine":"12.254883589935275",
				"telefono1":"0547.302061",
				"telefono2":"",
				"email":"",
				"indirizzo":"Via Abbazia del Monte....",
				"sito_web":"http:\/\/www.abbaziadelmonte.it\/",
				"orari_apertura":"Basilica...",
				"video":[],
				"link":[
					{	"titolo":"Comune di Cesena",
						"url":"http:\/\/www.comune.cesena...",
					}
				],
				"nome":"Abbazia Santa Maria del Monte",
				"descrizione_breve":"Millenaria abbazia ...",
				"descrizione":"..."
			},
			...
		]
	},
	"message":"",
	"success":true
}
```

Anche in questa pagina, come nella precedente, viene subito dichiarato un oggetto contenente tre coppie nome/valore viste in precedenza:

- "data": è a sua volta un oggetto costituito da coppie nome/valore, al suo interno troviamo tutto il contenuto del database che risiede sul server:
  - "categoria-dovebere": è un array dato che il campo valore inizia con il carattere [(parentesi quadra sinistra). Al suo interno troviamo degli oggetti utilizzati per specificare le singole voci relative alle categorie dei luoghi dove poter bere. Questi oggetti sono composti dalle seguenti coppie nome/valore:
    - "id": identificatore univoco associato al record della tabella categoria dove bere;
    - "id_utente": identificatore univoco relativo all'utente che ha creato il record della tabella categoria dove bere;
    - "data_modifica": data dell'ultima modifica effettuata sul record;
    - "codice";
    - "ordine";
    - "abilitato";
    - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore:
      - "id": è un identificatore univoco associato al record della tabella icona;
      - "file": stringa di testo utilizzata per indicare il nome del file comprensivo di estensione;
      - "alt": stringa di testo relativa all'immagine;
      - "name": stringa di testo utilizzata per salvare il pathname del file immagine, ovvero il percorso in forma esplicita che contiene le informazioni sulla posizione del file all'interno del sistema;
      - "ext": stringa di testo utilizzata per indicare l'estensione del file immagine;
    - "nome": stringa di testo utilizzata per indicare il nome della tabella categoria dove bere
  - "categoria-dovedormire": è un array composto da oggetti utilizzati

per specificare i singoli record della tabella categorie dove dormire. Questi oggetti sono composti dalle coppie nome/valore descritte per "categoria-dovebere": "id, "id_utente", "data_modifica", "codice", "ordine", "abilitato", "icona" e "nome".

- "categoria-dovemangiare": è un array composto da oggetti utilizzati per specificare le singole voci della tabella categorie dove mangiare. Questi oggetti sono composti dalle coppie nome/valore descritte per "categoria-dovebere": "id, "id_utente", "data_modifica", "codice", "ordine", "abilitato", "icona" e "nome".
- "categoria-poi": è un array composto da oggetti ognuno dei quali descrive un record della tabella categorie dei POI. Questi oggetti sono composti dalle coppie nome/valore descritte per "categoria-dovebere": "id, "id_utente", "data_modifica", "codice", "ordine", "abilitato", "icona" e "nome".
- "dettipopolari": è un array all'interno del quale sono descritti oggetti, uno per ogni record presente nella tabella del database detti popolari. Le coppie nome/valore che compongono gli oggetti sono:
  - "id": identificatore univoco associato al record della tabella detti popolari;
  - Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";
  - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
  - "testo": stringa di testo nella quale viene salvato il detto popolare in lingua dialettale;
  - "significato": stringa di testo nella quale viene salvata la traduzione del detto popolare dialettale.
- "dovebere": è un array costituito da oggetti ognuno dei quali associato ad un record della tabella dove bere. Gli oggetti sono descritti dalle seguenti coppie nome/valore:
  - "id": identificatore univoco associato al record della tabella dove bere;
  - Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";

  - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
  - "categoria": campo numerico nel quale viene salvato il campo "id" della tabella categoria dove bere;
  - "latitudine": campo numerico utilizzato per la coordinata della latitudine;
  - "longitudine": campo numerico utilizzato per la coordinata della longitudine;
  - "telefono": stringa di testo utilizzata per salvare il numero primario dell'attività inserita nella tabella dove bere;
  - "telefono2": stringa di testo utilizzata per memorizzare il secondo recapito telefonico;
  - "email": stringa di testo utilizzata per la mail del locale;
  - "indirizzo": stringa di testo nella quale è salvato l'indirizzo del locale;
  - "sito_web": stringa di testo utilizzata per il sito web dell'esercizio commerciale;
  - "titolo": stringa di testo con il nome dell'attività commerciale;
  - "descrizione": stringa di testo nella quale è inserita una descrizione del locale.

- "dovedormire": è un array composto da oggetti, ognuno dei quali ha le informazioni relative ad un record della tabella dove dormire. Gli oggetti sono costituiti dalle stesse coppie nome/valore degli oggetti dell'array "dovebere" con l'aggiunta della coppia "stelle", valore numerico riservato per indicare quante stelle possiede l'esercizio commerciale descritto. I campi identici all'array "dovebere" sono: "id", "id_utente", "data_modifica", "codice", "ordine", "abilitato", "icona", "categoria", "latitudine", "longitudine", "telefono", "telefono2", "email", "indirizzo", "sito_web", "titolo" e "descrizione".
- "dovemagiare": è un array dove sono dichiarati degli oggetti, ognuno dei quali è associato ad un record della tabella dove mangiare. Gli oggetti sono descritti dalle stesse coppie nome/valore degli oggetti dell'array "dovebere": "id", "id_utente", "data_modifica", "codice",

"ordine", "abilitato", "icona ", categoria", "latitudine", "longitudine", "telefono", "telefono2", "email", "indirizzo", "sito_web", "titolo" e "descrizione".

- "news": è un array composto da oggetti, ognuno dei quali è un record della tabella news presente sul database. Questi oggetti sono composti dalle seguenti coppie nome/valore:
  - "id": identificatore univoco associato al record della tabella news;
  - Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";
  - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
  - "titolo": stringa di testo nella quale viene salvato il nome della news;
  - "descrizione": stringa di testo per la descrizione della news.
- "newseventi": è un array composto da oggetti, ognuno dei quali è un record della tabella news presente sul database. Questi oggetti sono composti dalle seguenti coppie nome/valore:
  - "id": identificatore univoco associato al record della tabella newseventi;
  - Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";
  - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
  - "titolo": stringa di testo nella quale viene salvato il nome dell'evento;
  - "periodo": stringa di testo utilizzata per indicare la durata dell'evento;
  - "descrizione": stringa di testo per la descrizione dell'evento.
- "percorsiguidati": è un array formato da oggetti i quali hanno le informazioni relative ad un record della tabella percorsi guidati. Gli oggetti sono costituiti dalle seguenti coppie nome/valore:

  - "id": identificatore univoco associato al record della tabella percorsi guidati;
  - Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";
  - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
  - "elencopoi": è un array costituito da oggetti formati da una unica coppia nome/valore. La stringa associata al nome è "poi" e il valore è numerico nel quale è salvato l'identificativo del record della tabella poi.
  - "descrizione_breve": stringa di testo utilizzata per una breve descrizione del percorso guidato;
  - "descrizione": stringa di testo per la descrizione del percorso guidato.
- "personaggiillustri": è un array composto da oggetti, ognuno dei quali è un record della tabella personaggi illustri presente sul database. Questi oggetti sono composti dalle seguenti coppie nome/valore:
  - "id": identificatore univoco associato al record della tabella personaggi illustri;
  - Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";
  - "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
  - "dataNascitaMorte": stringa di testo riservata per salvare la data di nascita e di morte del personaggio illustre;
  - "nome": stringa di testo per il nome del personaggio;
  - "descrizione": stringa di testo utilizzata per una descrizione della vita del personaggio.
- "poi": è un array costituito da oggetti, ognuno dei quali ha le informazioni relative ad un record della tabella poi. Gli oggetti sono costituiti dalle stesse coppie nome/valore degli oggetti dell'array "dovebere": "id", "id_utente", "data_modifica", "codice", "ordine", "abilitato",

"icona", "categoria", "latitudine", "longitudine", "telefono", "telefono2", "email", "indirizzo", "sito_web", "titolo" e "descrizione". In aggiunta qui troviamo anche:

- "gallery": è un array costituito da oggetti che prendono il nome di "image", questi oggetti sono costituiti dagli stessi campi dell'oggetto "icona": "id", "file", "alt", "name" e "ext";
- "orari_apertura": stringa di testo utilizzata per memorizzare in quali orari è possibile accedere al POI, nel caso in cui si trattasse di un museo o strutture simili;
- "video": stringa di testo utilizzata per il salvataggio dell'URL del video relativo al POI in questione;
- "link": è un oggetto costituito da due coppie nome/valore:
  - "titolo": stringa di testo utilizzata per il titolo del link;
  - "url": stringa di testo per l'url del link;
- "descrizione_breve": stringa di testo utilizzata per una breve descrizione del POI;

• "storia": è un array composto da oggetti, ognuno dei quali è un record della tabella storia presente sul database. Questi oggetti sono composti dalle seguenti coppie nome/valore:

- "id": identificatore univoco associato al record della tabella storia;
- Informazioni relative al record già descritte in precedenza: "id_utente", "data_modifica", "codice", "ordine" e "abilitato";
- "icona": è a sua volta un oggetto composto dalle seguenti coppie nome/valore: "id", "file", "alt", "name" e "ext" già descritte in precedenza;
- "testo": stringa di testo utilizzata per il test di una storia riguardante la città.

o "message": in questo caso specifico non è stato assegnato nessun valore a questo campo, in generale utilizzato per mandare un messaggio

o "success": indica se la procedura è andata a buon fine o meno

3.3. Database

Un database, come dice la parola stessa è una base (archivio) di dati all'interno del quale risiedono informazioni organizzate e strutturate secondo un modello logico volto a ottimizzare, in termini di efficienza e di prestazioni, la gestione e l'organizzazione dei dati stessi [14].

Dopo un analisi approfondita dei dati presenti nel database on line e seguendo come filo conduttore la definizione appena riportata, è stato effettuato lo studio e la progettazione della struttura del database, pensato e ottimizzato per l'applicativo dello smartphone.

Di seguito sono illustrate sia la rappresentazione del database attraverso il modello E/R che tramite il modello relazionale.

3.3.1. *Modello E/R*

Il modello E/R (Entity–relationship model) anche detto modello entità relazione è stato ideato dal prof. Peter Chen per la rappresentazione concettuale dei dati ad un alto livello di astrazione.

Esistono diversi "dialetti" del modello E/R, che non interferiscono con la struttura del modello ma si differenziano per le notazioni grafiche. Per questa tesi è stata adottata la grafica fornita dall'applicativo DB-Main (applicativo free pensato per sostenere sviluppatori e analisti nella maggior parte delle fasi di progettazione) [15]. Di seguito sono spiegati i principali costrutti del modello E/R e la relativa notazione adottata da DB-Main.

- Entità: è un insieme di oggetti con caratteristiche comuni che appartengono alla realtà che vogliamo descrivere. L'istanza di un entità è uno specifico oggetto appartenente a quella entità.

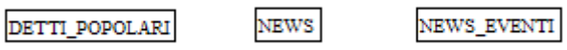

Figure 15 – Struttura grafica dell'entità

- Associazione: rappresenta un legame logico tra entità, rilevante nella realtà che si sta considerando.

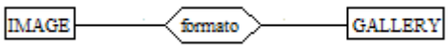

Figure 16 – Struttura grafica dell'associazione

o Attributo: Un attributo è una proprietà elementare di un'entità o di un'associazione.

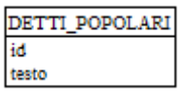

Figure 17 – Struttura grafica dell'associazione

o Identificatori: hanno lo scopo di permettere l'individuazione univoca delle istanze di un'entità; sono rappresentato graficamente da una riga sottostante l'attributo che svolge la funzione di identificatore.

o Gerarchie: Un'entità "CATEGORIA" è una generalizzazione di un gruppo di entità "CATEGORIA-DOVEBERE", ... se ogni istanza di "CATEGORIA-DOVEBERE", ... è anche un'istanza di "CATEGORIA". Le entità "CATEGORIA-DOVEBERE", ... sono dette specializzazioni di "CATEGORIA".

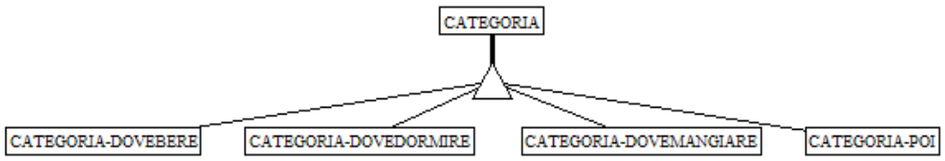

Figure 18 – Struttura grafica della gerarchia

o Vincoli di cardinalità: Sono coppie di valori (min-card,max-card) associati a ogni entità che partecipa a un'associazione; specificano il numero minimo e massimo di istanze dell'associazione a cui un'istanza dell'entità può partecipare.

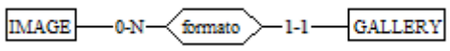

Figure 19 – Struttura grafica dei vincoli di cardinalità

Viene ora illustrato il modello E/R studiato appositamente per la struttura del database. Al fine di una migliore analisi viene descritto suddividendolo in due parti: dettaglio Luoghi e dettaglio Curiosità.

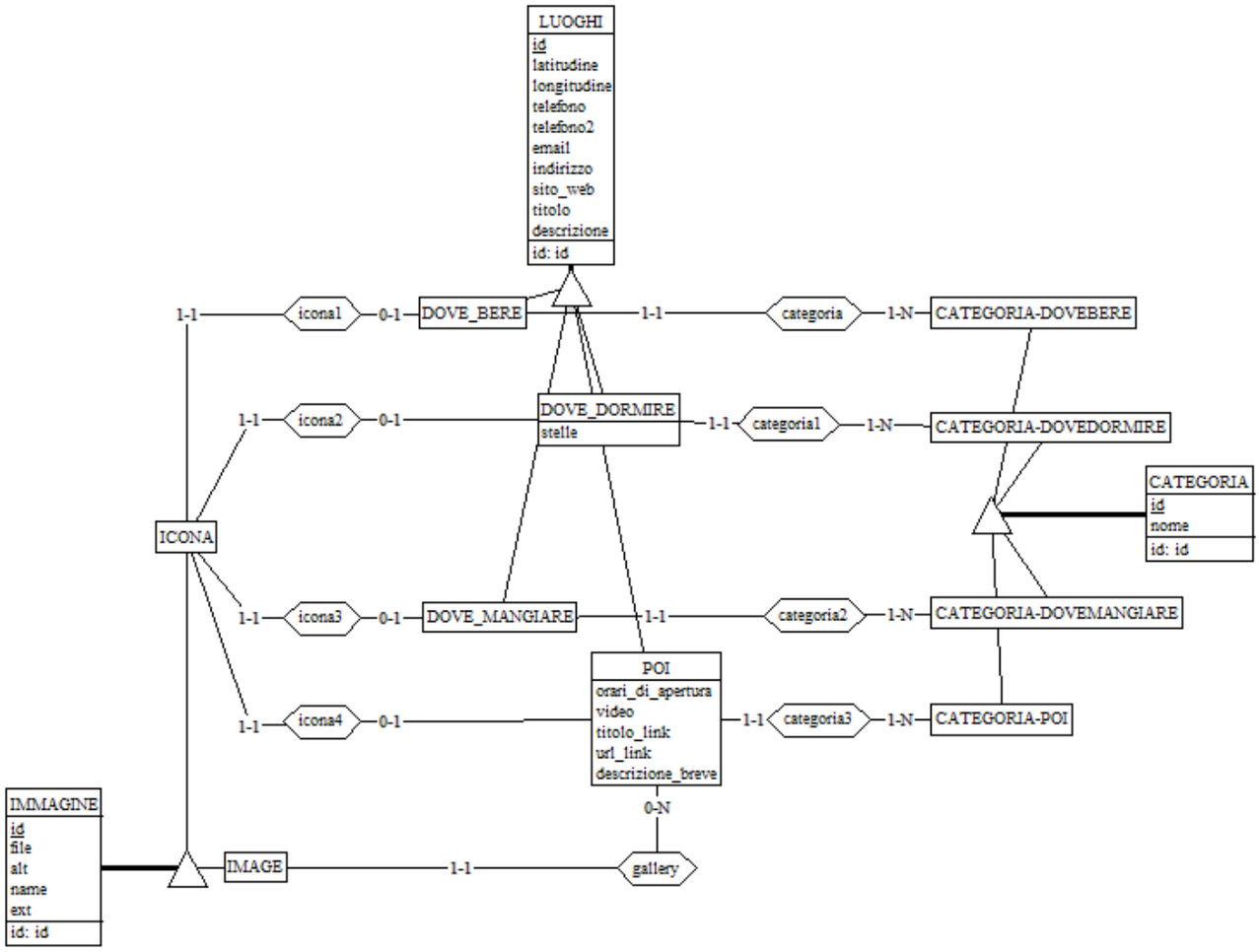

Figure 20 – Modello E/R dettaglio luoghi

Partendo dalla destra dell'immagine, la gerarchia si sviluppa partendo dall'entità "CATEGORIA", che è una generalizzazione delle specializzazioni "CATEGORIA-DOVEBERE", "CATEGORIA-DOVEDORMIRE", "CATEGORIA-DOVEMANGIARE" e "CATEGORIA-POI". Queste non hanno attributi aggiuntivi rispetto alla generalizzazione "CATEGORIA", che invece è composta da:

- "id": attributo identificatore e come tale permette di identificare ogni singola istanza dell'entità in maniera univoca;
- "nome": attributo di testo nel quale viene salvato il nome della categoria.

In questo primo dettaglio del modello E/R è presente un'altra gerarchia, visibile sulla parte sinistra dell'immagine, in cui l'entità "IMMAGINE" è la generalizzazione le entità "ICONA" e "IMAGE" sono specializzazioni. Anche in questo caso le specializzazioni non presentano attributi aggiuntivi, pertanto è sufficiente elencare quelli dell'entità "IMMAGINE":

- “id”: attributo identificatore dell’entità
- “file”: attributo di tipo testo utilizzato per indicare il nome del file comprensivo di estensione;
- “alt”: stringa di testo relativa all’immagine;
- “name”: attributo testuale utilizzata per salvare il pathname del file immagine, ovvero il percorso in forma esplicita che contiene le informazioni sulla posizione del file all’interno del sistema;
- “ext”: attributo testuale utilizzato per indicare l’estensione del file immagine.

La terza ed ultima gerarchia presente in questo dettaglio è composta dalla generalizzazione “LUOGHI” e dalle specializzazioni “DOVEBERE”, “DOVEMANGIARE”, “DOVEDORMIRE” e “POI”.

Le entità “DOVEBERE” e “DOVEMANGIARE” non aggiungono altri attributi oltre alla generalizzazione “LUOGHI” e sono composte dai seguenti attributi:

- “id”: identificatore univoco;
- “latitudine”: attributo di tipo numerico utilizzato per la coordinata della latitudine;
- “longitudine”: attributo di tipo numerico utilizzato per la coordinata della longitudine;
- “telefono”: attributo di tipo testuale utilizzato per salvare il numero primario del luogo descritto;
- “telefono2”: attributo di tipo testuale utilizzato per memorizzare il secondo recapito telefonico;
- “email”: attributo di tipo testuale utilizzato per la mail;
- “indirizzo”: attributo di tipo testuale nella quale è stato salvato l’indirizzo del luogo;
- “sito_web”: attributo di tipo testuale utilizzata per il sito web del luogo;
- “titolo”: attributo di tipo testuale riservato al nome del luogo;
- “descrizione”: attributo di tipo testuale nella quale è inserita una descrizione del luogo.

L’entità “DOVEDORMIRE” invece presenta un attributo aggiuntivo rispetto alle due specializzazioni appena descritte:

- “stelle”: attributo numerico utilizzato per salvare il numero di stelle associate al luogo in cui poter albergare.

Anche l’ultima specializzazione “POI” ha, in aggiunta alla generalizzazione “LUOGHI”, altri cinque attributi:

- “orari_apertura”: attributo di tipo testuale utilizzato per indicare gli orari di apertura al pubblico;
- “video”: attributo testuale utilizzato per il salvataggio dell’URL del video relativo al POI in questione;
- “titolo_link”: attributo di tipo testuale utilizzato per specificare il titolo del link associato al POI.
- “url_link”: attributo di tipo testuale utilizzato per specificare l’url del link con informazioni correlate al POI;
- “descrizione_breve”: attributo di tipo testuale utilizzato per salvarci un estratto della descrizione completa relativa al POI.

Di seguito sono descritte le associazioni e i vincoli di cardinalità che intercorrono tra le entità appena illustrate.

Come è possibile notare dalla Figura 20, esistono quattro associazioni che mettono in relazione l’entità “ICONA” con le specializzazioni della generalizzazione “LUOGHI”: “icona1”, “icona2”, “icona3” e “icona4”. In tutte e quattro la coppia di valori (card. min, card. max) dell’entità “ICONA” è 1-1: questo indica che ogni istanza di “ICONA” è associata ad una e sola una istanza delle specializzazioni che fanno parte dell’associazione. Dall’altro lato le quattro specializzazioni di “LUOGHI” presentano il valore 0-1, che indica che le istanze di queste entità possono essere associate con una o nessuna istanza di “ICONA”.

Altre associazioni sono “categoria1”, “categoria2”, “categoria3” e “categoria4” che legano le specializzazioni dell’entità “LUOGHI” con le specializzazioni dell’entità “CATEGORIA”. Qui la cardinalità delle specializzazioni di “LUOGHI” presenta il valore 1-1 che indica che ogni istanza dell’entità ha associata un’istanza dell’entità coinvolta nella relazione. La cardinalità delle specializzazioni “CATEGORIA” partecipa all’associazione con il valore 1-N per a quale ogni istanza delle specializzazioni “CATEGORIA” è legato ad una o più istanze dell’entità delle specializzazioni “LUOGHI”.

“IMAGE” e “POI” partecipano all’associazione “gallery” rispettivamente con i valori 1-1 e 0-N : ogni istanza di “IMAGE” è in associazione con una

sola istanza di “GALLERY”, viceversa, ogni istanza di “POI” è associata con più di una o nessuna istanza di “IMAGE”.

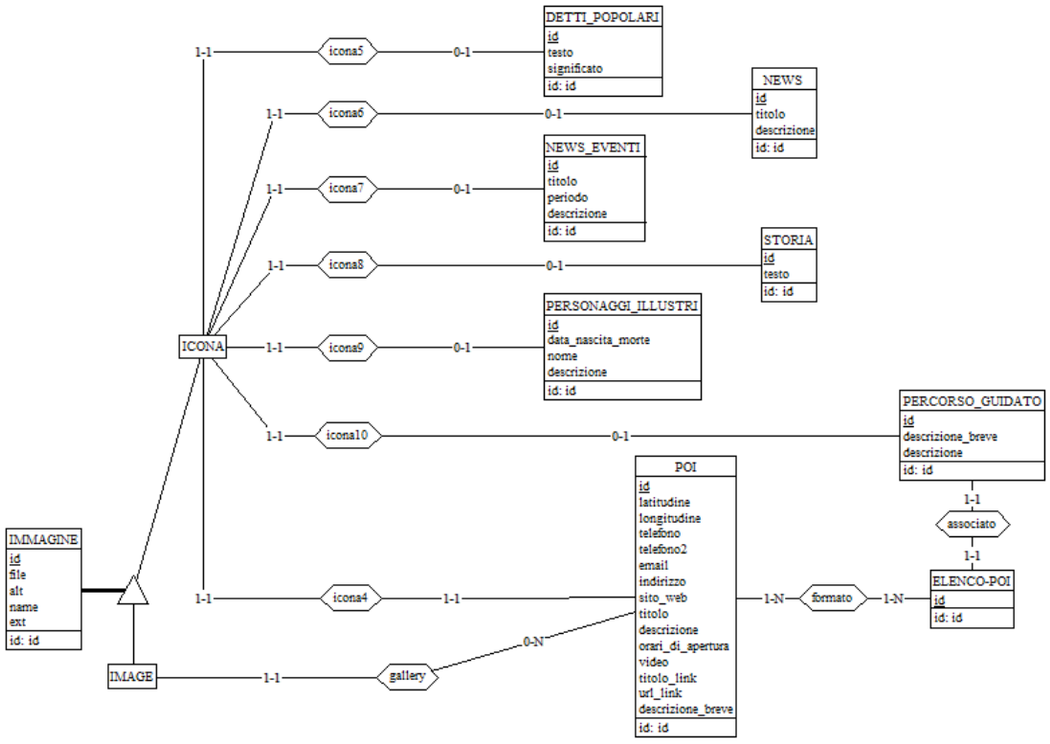

Figure 21 – Modello E/R dettaglio “curiosità”

La seconda parte da analizzare è quella del dettaglio “curiosità”.

La tabella “DETTIPOPOLARI” è utilizzata per raccogliere i proverbi tipici del luogo nel dialetto locale con la relativa traduzione in italiano. Questa tabella è costituita dai seguenti attributi:

- “id”: attributo numerico utilizzato come identificatore univoco;
- “testo”: attributo testuale utilizzato per il testo del detto popolare;
- “significato”: attributo testuale utilizzato per la traduzione e il significato del detto popolare.

Nella tabella “NEWS” sono raccolte le notizie giornalistiche relative alla città e al suo interno sono contenuti i campi:

- “id”: attributo numerico utilizzato come identificatore univoco;
- “titolo”: attributo testuale in cui viene riportato il nome della news;
- “descrizione”: attributo testuale contenente la descrizione della news.

La tabella “NEWSEVENTI”, molto simile alla precedente, presenta i suoi stessi campi con l’aggiunta dell’attributo di testo “periodo”, utilizzato per

indicare la durata dell'evento.

I racconti e gli avvenimenti storici riguardanti la città sono inseriti nella tabella "STORIA", i cui attributi sono:

- "id": identificatore univoco della tabella;
- "testo": campo testuale nel quale troviamo descritto l'intero aneddoto.

"PERSONAGGIILLUSTRI" è una tabella in cui vengono collocate le informazioni relative a persone famose, sia decedute che in vita, che sono nate o vissute nella città. In questa tabella troviamo:

- "id": identificatore univoco;
- "dataNascitaMorte": campo testuale nel quale sono inserite le date di nascita e di morte del personaggio;
- "nome": campo testuale utilizzato per il nome;
- "descrizione": campo testuale utilizzato per raccontare il vissuto del personaggio.

Gli itinerari percorribili in città sono memorizzati in "PERCORSOGUIDATO" che possiede i seguenti campi:

- "id": identificatore univoco;
- "descrizione_breve": campo testuale utilizzare per riportare un estratto dell'intera descrizione;
- "descrizione": campo testuale riservato per la descrione del percorso guidato.

L'ultima tabella è "ELENCOPOI" costituita dall'unico campo "id", identificatore univoco.

Ad eccezione di "ELENCOPOI", tutte le tabelle presenti in questo dettaglio sono associate con "ICONA" con cardinalità 0-1; viceversa la tabella "ICONA" partecipa all'associazione con cardinalità 1-1.

Le entità "PERCORSOGUIDATO" e "ELENCOPOI" partecipano alla loro associazione rispettivamente con valori 1-1 e 1-1; infatti ogni entità di "PERCORSOGUIDATO" può avere un'unica entità di "ELENCOPOI" associata, lo stesso vale per il viceversa.

L'ultima associazione presente è quella che lega l'entità "ELENCOPOI" con "POI", entrambe con valore 1-N: ogni istanza di "ELENCOPOI" è associata ad uno o più istanze di "POI", lo stesso si verifica per i valori che interessano "POI".

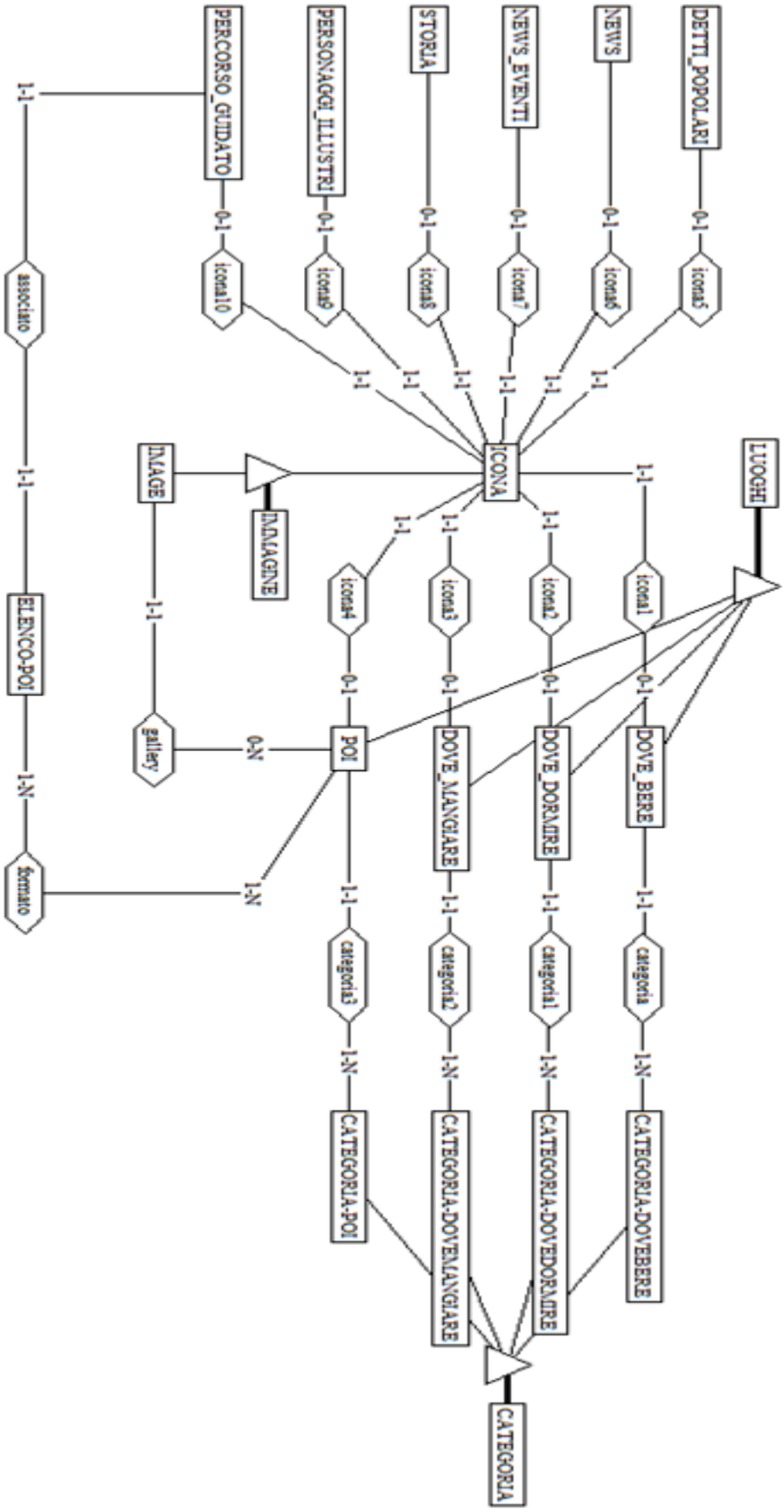

Figure 21 – Modello E/R

3.3.2. *Modello logico*

La necessità di descrivere il modello logico relativo al modello E/R sopra illustrato è dovuta al fatto che la sua espressività non è normalmente sufficiente in fase di progettazione ma necessita di documentazione di supporto.

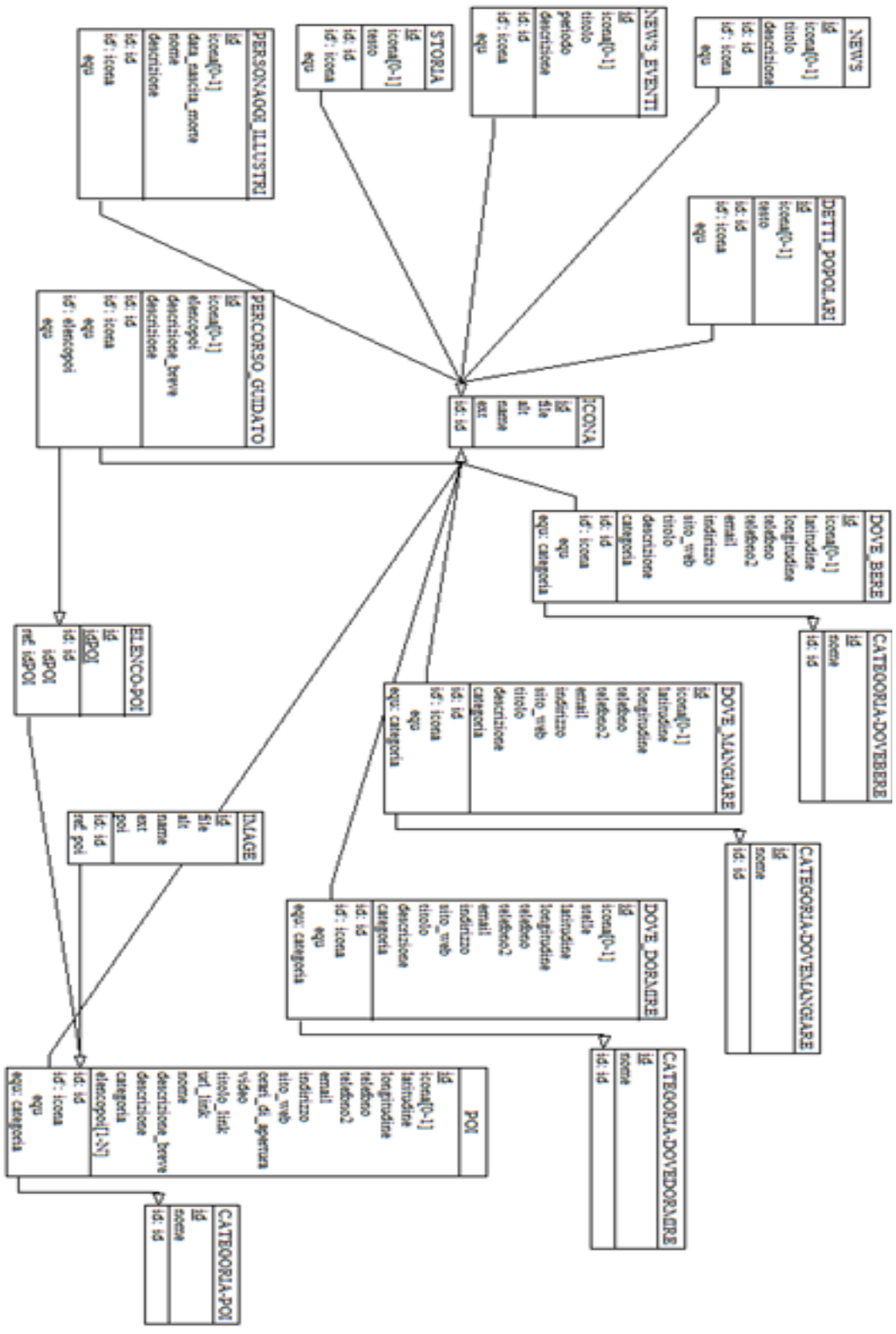

Figure 22 – Modello logico

La fase di progettazione logica ha lo scopo di derivare uno schema logico che rispetti quanto più possibile i concetti espressi nello schema E/R di partenza e che sia al tempo stesso "efficiente".

In questa fase vengono eliminati tutti i costrutti che non possono essere direttamente rappresentati nel modello logico, apportando modifiche strutturali sulla base di considerazioni di efficienza rispetto ai volumi di dati in gioco.

Leggendo lo schema logico si può notare che le tre gerarchie presenti nello schema E/R sono state collassate verso il basso, generando quindi una tabella per ogni specializzazione della gerarchia.

La generalizzazione "LUOGHI" è stata sostituita dalle quattro tabelle: "DOVEBERE", "DOVEMANGIARE", "DOVEDORMIRE" e "POI".

Questa scelta è stata fatta per ottenere dei miglioramenti a livello di prestazioni: l'applicativo Live Tour, infatti, utilizza i dati di ogni singola tabella delle specializzazioni in activity separate che non utilizzano i dati relativi ad un'altra tabella della gerarchia. Senza il collasso verso il basso si sarebbe incorsi in un carico di lavoro maggiore e inutile poichè ogni activity avrebbe dovuto lavorare con tutti i luoghi presenti nel database e quindi con una mole di dati maggiore, ma soprattutto con dati non pertinenti.

Un'altra motivazione che ha inciso nella scelta è il risparmio di memoria. Collassando verso l'alto la tabella "LUOGHI avrebbe presentato campi con valore nullo per istanze non attinenti.

Lo stesso ragionamento è stato fatto per il collasso verso il basso delle generalizzazioni "CATEGORIA" e "IMMAGINE".

Osservando lo schema si riesce ad apprezzare la trasformazione delle associazioni schematizzate attraverso il modello E/R.

Le associazioni che interessano la tabella "ICONA" ("icona1", "icona2", "icona3", … , "icona10") sono state tutte sostituite da un campo numerico "icona" che riporta l'identificatore univoco della tabella "ICONA" ed è posizionato all'interno delle tabelle in relazione con essa. Un'altra opzione sarebbe stata quella di inserire i riferimenti univoci delle tabelle all'interno di "ICONA" questa modalità avrebbe causato spreco di memoria e minor leggibilità dei dati in quanto sarebbero stati riportati per ogni istanza di "ICONA" tutti i riferimenti agli identificatori delle tabelle associate.

Anche le associazioni “categoria”, “categoria1”, “categoria2” e “categoria3” sono state trasformate posizionando nelle tabelle “DOVEBERE”, ”DOVEMANGIARE”, “DOVEDORMIRE” e “POI” il riferimento univoco dell’entità “CATEGORIA” associata. Questo perchè l’utente durante l’utilizzo dell’applicazione può filtrare i “LUOGHI” in base alla categoria, informazione già salvata in ogni istanza e fruibile fin da subito.

“gallery” è stata sostituita dal riferimento univoco della tabella “POI” posto all’interno della tabella “IMAGE”. Questa è risultata la scelta migliore in quanto non necessita la costruzione di tabelle aggiuntive per mantenere la fedeltà dei due schemi.

La modifica fatta ad “associato” è stata quella di aggiungere alla tabella “PERCORSOGUIDATO” l’identificatore univoco della tabella “ELENCOPOI”.

Per “formato” invece si è scelto di posizionare il riferimento univoco di “POI” all’interno di “ELENCOPOI” con il nome di “poi”. Il nuovo campo “poi” è chiave assieme al campo “id”, in quanto un POI può essere presente al massimo una volta in un percorso guidato.

In questa fase sono state rispettate le forme di normalizzazione per far si che il database funzioni in maniera ottimale, senza incorrere nel rischio di avere dati ridondanti o inconsistenti. Le forme di normalizzazioni adottate sono le seguenti:

- Si dice che una relazione è in 1FN (prima forma normale) se e solo se il dominio di ciascun attributo comprende solo valori atomici.
- Uno schema è in 2FN se non c’è dipendenza parziale di un attributo non-primo da una chiave.
- Una relazione si dice in terza forma normale (3FN) quando è innanzitutto in seconda forma normale e tutti gli attributi non-chiave dipendono dalla chiave soltanto, ossia non esistono attributi che dipendono da altri attributi non-chiave [14].

4. Sviluppo

Viene ora presentata la parte dello sviluppo per il completamento dell'applicativo e il codice implementato. Il lavoro può essere illustrato dividendolo in tre parti.

4.1. Splash Activity

La prima operazione necessaria al completamento dell'applicativo è quella della creazione e il riempimento del database sul dispositivo mobile. Queste operazioni onerose in termini di risorse, vengono eseguite durante la Splash Activity.

Una Splash Activity è un immagine o un animazione che viene mostrata all'utente in attesa che l'applicativo esegua operazioni di inizializzazione o di caricamento. Se viene scelta un immagine piuttosto che un animazione è buona norma sovrapporre una scritta o una barra di progresso le quali mostrano all'utente che sono in atto operazioni indipendentemente dalla schermata fissa, al fine di evitare che venga arrestata l'applicazione pensando che si sia bloccata. In altre parole, essi forniscono all'utente un feedback che un lungo processo è in corso. La splash Activity viene fatta scomparire al termine delle operazioni di inizializzazione e di caricamento per lasciare posto alla schermata del programma.

Una Splash Activity è inoltre un buon modo per dare un ottima impressione grafica, per tanto va creata rispettando lo stile e i colori dell'applicazione e dell'icona di lancio.

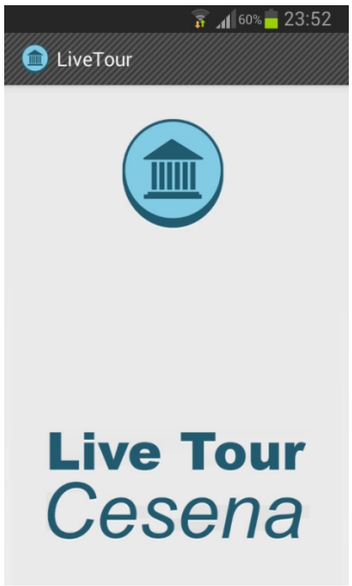

Figura 23 – Splash screen dell'applicazione Live Tour

Lo splash screen creato richiama il colore azzurro, coloro dominante dello stile dell'applicazione e dell'icona, inoltre riporta l'icona stessa e il nome dell'applicazione.

Figura 24 – Icona dell'applicazione Live Tour

Come già detto le funzioni svolte dalla Spalsh Activity sono onerose in termini di risorse e se svolte all'interno del thread principale dell'applicazione, rischiano di causare l'arresto dell'applicazione stessa.

Nel thread principale infatti vanno solo svolte le funzioni che si occupano della gestione degli eventi che arrivano dall'utente (ad esempio il click di un tasto o il tocco dello schermo), nel caso in cui l'applicazione non risponda entro 5 secondi viene mostrato il dialog "*Application Not Responding*" e l'utente può decidere di chiuderla.

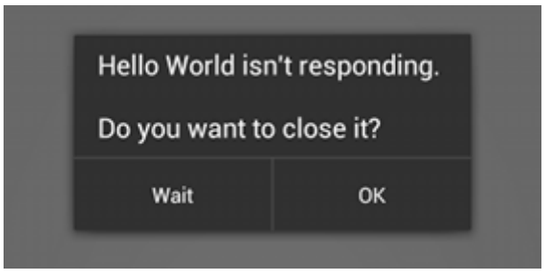

Figure 25 – Allert dialog "Application Not Responding"[21]

Per riuscire a smaltire il carico oneroso di lavoro e non bloccare l'applicativo viene utilizzata *AsyncTask*. [22]

La classe *AsyncTask* definisce tre generics: *AsyncTask<Params, Progress, Result>*. I generics servono ad aumentare la "sicurezza in compilazione" ed a evitare *ClassCastException* runtime.

Questi parametri vengono utilizzati per i metodi principali di *AsyncTask*:

- *onPreExecute*: eseguito sul thread principale, contiene il codice di inizializzazione dell'interfaccia grafica (per esempio l'inabilitazione di un button);
- *doInBackground*: eseguito in un thread in background si occupa di eseguire il task vero e proprio. Accetta un parametro di tipo *Params* (il primo generic definito) e ritorna un oggetto di tipo *Result;*
- *onPostExecute*: eseguito nel thread principale e si occupa di aggiornare l'interfaccia dopo l'esecuzione per mostrare i dati scaricati o calcolati del task che vengono passati come parametro.

Per il progetto in questione è stata creata una classe *BackgroundAsyncTask* che estende la classe *AsyncTask* e che accetta come generics in ingresso tre Void *che sono* classi "segnaposto" usate nei casi in cui non serve una classe vera e propria (da non confondere con la keyword java *void* con la iniziale minuscola).

```
public class BackgroundAsyncTask extends AsyncTask <Void, Void, Void> {

        public JSONParser json;
        public FeedReaderDbHelper DbHelper;
        public Cursor c;


        @Override
        protected void onPreExecute() {
              proDialog = ProgressDialog.show(context, getString
                    (R.string.progressDialog_title_wait), getString
                    (R.string.progressDialog_downloadDB));
              proDialog.setIcon(R.drawable.ic_launcher_live_tour);
        }

        @Override
        protected Void doInBackground(Void... params) {

              SharedPreferences      sharedPref      =      SplashActivity.this.
              getPreferences(Context.MODE_PRIVATE);
             int oldVersion = sharedPref.getInt("dataBaseVersion", 1);

             json = new JSONParser(context);

             int newVersion = json.dbVersionCheck(context, oldVersion);

              SharedPreferences.Editor editor = sharedPref.edit();
              editor.putInt("dataBaseVersion", newVersion);
              editor.commit();

              return null;
        }

        @Override
        protected void onProgressUpdate(Void... values) {

            }

        @Override
        protected void onPostExecute(Void result) {

              proDialog.dismiss();
              Intent intent = new Intent(context, Map_Activity.class);
              startActivity(intent);

        }
}
```

Sono inoltre riscritti i metodi:

- *onPreExecute*: crea un oggetto *ProgressDialog* che viene mostrato durante lo splashscreen per far sapere all'utente quello che sta elaborando l'*AsyncTask*

- *doInBackground*: crea un oggetto di tipo *SharedPreferences* (framework messo a disposizione da Android per la gestione della persistenza delle impostazioni [23]) nella quale è contenuto il valore della versione del database presente sul dispositivo. Questa informazione viene recupe-

rata e passata alla funzione *dbVersionCheck*, appartanente alla classe *JSONParser*, che, dopo aver stabilito se c'è la necessità o meno di fare l'aggiormanento, restituisce il valore della versione del nuovo database da sovrascrivere sulla *SharedPreferences*.

- *onPostExecute*: cancella l'oggetto *ProgressDialog* e avvia la schermata principale dell'applicazione.

Durante l'*AsyncTask* e precisamente all'interno del metodo *doInBackground* viene istanziato un oggetto di tipo *JSONParser* che contiene la classe per la creazione e la cancellazione del database (*dbHelper*) e i metodi per la gestione dei dati (*dbVersionCheck e dbUpdate*).

4.1.1. *Creazione del database: definizione dello schema*

Lo schema di un database rispecchia la sua organizzazione ed è strutturato tramite istruzioni SQL.

Per la creazione e la gestione sono state utilizzate le funzioni messe a disposizione dalla classe *SQLiteDatabase* disponibili nel package *android.database.sqlite*.

È buona norma creare delle classi guida (conosciute come classi *Helper* e *Contract*): esse sono contenitori per le costanti che definiscono i nomi per URL, tabelle e colonne.

La classe *Contract* implementata per questa applicazione è stata denominata *FeedReaderContract* ed è composta soltanto da stringhe definite *public*, *final* (non modificabile) e *static* (rende l'attributo della classe comune a tutte le sue istanze). Questa scelta consente di utilizzare le stesse costanti in tutte le altre classi dello stesso pacchetto. In questo modo si può cambiare un nome di colonna in un punto per far si che si modifichi in tutto il codice [16].

```
public class FeedReaderContract {

	public static final String URL_CHECK = "http://www.citylivetour.com/mo-
	bile/cesena/check.php";
	public static final String URL_UPDATE = "http://www.citylivetour.com/mo-
	bile/cesena/entita/update.php";

	public static final String URL_DOWNLOAD_ICON = "http://www.citylivetour.
	com/images/80x60/";
	public static final String URL_DOWNALOD_IMAGE = "http://www.citylivetour.
	com/images/640x480/";
	public static final String PATH_STORE = "/data/data/it-mwd.placeFinder/
	files/";

	private FeedReaderContract(){
	}

	public static abstract class FeedEntry implements BaseColumns{


		public static final String TAG_POI = "poi";

		...
		...

		public static final String TABLE_POI = "poi";
		public static final String COLUMN_POI_ID = "id";

		...
		...
	}
}
```

Un buon modo per organizzare questa classe è quello di mettere le definizioni globali dell'intero database nel livello principale della classe e, successivamente, creare una classe interna per ogni tabella che enumera le sue colonne. Nel caso in questione non è stato scelto questo metodo: nelle definizioni globali sono state inserite le informazioni relative agli indirizzi web, inoltre non sono state create più classi interne ma una unica in cui sono riposte tutte le informazioni relative alle tabelle.

Nell'unica classe interna creata i nomi delle variabili sono stati suddivisi in:

- TAG_[nome attributo]: nome della coppia nome/valore dei file JSON;
- TABLE_[nome tabella]: nome della tabella (tra quelle presenti sul database);
- COLUMN_[nome tabella]_[nome colonna]: nome della colonna relativa ad una tabella.

La classe *Helper* è anch'essa costituita da stringhe definite *private*, *final* e *static* che descrivono la struttura della tabella secondo la sintassi del linguaggio SQL. Contiene inoltre i metodi che creano e gestiscono il database. Essa estende la classe *SQLiteOpenHelper* [17] e le impone l'utilizzo e la riscrittura dei seguenti metodi:

- *FeedReaderHelper* crea un oggetto *FeedReaderHelper* con il riferimento al database;
- *onCreate* richiama il metodo *execSQL* per creare ogni tabella del database;
- *onUpgrade* richiama il metodo *execSQL* per cancellare le tabelle presenti e ricrearne altre nuove tramite il metodo *onCreate*.

Nel codice riportato si possono osservare le stringhe con le istruzioni SQL per la creazione e la cancellazione delle tabelle utilizzate poi come parametro per le funzioni *execSQL*.

```
public class FeedReaderDbHelper extends SQLiteOpenHelper{

	public static final int DATABASE_VERSION = 1;
	public static final String DATABASE_NAME = "LiveTour.db";
	private static final String TEXT_TYPE = " TEXT";
	private static final String INT_TYPE = " INTEGER";
	private static final String FLOAT_TYPE = " FLOAT";
	private static final String COMMA_SEP = ", ";

	...


	private static final String SQL_CREATE_CAT_POI =
			"CREATE TABLE IF NOT EXISTS " + FeedReaderContract.FeedEn-
	try.TABLE_CAT_POI + " (" + FeedReaderContract.FeedEntry.
	COLUMN_CAT_POI_ID + INT_TYPE + " PRIMARY KEY" + COMMA_SEP +
	FeedReaderContract.FeedEntry.COLUMN_CAT_POI_ICONA + INT_TYPE +
	COMMA_SEP + FeedReaderContract.FeedEntry.COLUMN_CAT_POI_NOME + TEXT_
	TYPE + COMMA_SEP + " FOREIGN KEY( " + FeedReaderContract.FeedEntry.
	COLUMN_CAT_POI_ICONA + " ) REFERENCES " + FeedReaderContract.FeedEntry.
	TABLE_ICONA +"(" + FeedReaderContract.FeedEntry.COLUMN_ICONA_ID + "))";

	...

	private static final String SQL_DELETE_CAT_POI =
		"DROP TABLE IF EXISTS " + FeedReaderContract.FeedEntry.
	TABLE_CAT_POI;
	...

public FeedReaderDbHelper (Context context){
		super(context, DATABASE_NAME, null, DATABASE_VERSION);
	}

	public void onCreate(SQLiteDatabase db){
		db.execSQL(SQL_CREATE_ICONA);
		...
		db.execSQL(SQL_CREATE_STORIA);
	}

	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{

		db.execSQL(SQL_DELETE_ICONA);
		...
		db.execSQL(SQL_DELETE_STORIA);


		onCreate(db);
	}


}
```

4.1.2. *Inserimento e gestione dei dati*

Come descritto in precedenza, i dati da inserire nel database vengono recuperati da una pagina PHP, pertanto sono necessarie due funzioni, una per il recupero e una per il salvataggio, che sono descritte nella classe *JSONParser*.

```
    public JSONObject getJSONFromUrl(String url) {


     try {

         DefaultHttpClient httpClient = new DefaultHttpClient();
         HttpPost httpPost = new HttpPost(url);

         HttpResponse httpResponse = httpClient.execute(httpPost);
         HttpEntity httpEntity = httpResponse.getEntity();
         is = httpEntity.getContent();

     } catch (UnsupportedEncodingException e) {
         e.printStackTrace();
     } catch (ClientProtocolException e) {
         e.printStackTrace();
     } catch (IOException e) {
         e.printStackTrace();
     }

     try {
         BufferedReader reader = new BufferedReader(new InputStreamReader(
                 is, "iso-8859-1"), 8);
         StringBuilder sb = new StringBuilder();
         String line = null;
         while ((line = reader.readLine()) != null) {
             sb.append(line + "\n");
         }
         is.close();
         json = sb.toString();
     } catch (Exception e) {

     }


     try {
         jObj = new JSONObject(json);
     } catch (JSONException e) {

     }

     return jObj;
 }
```

Il download dei dati è gestito dalla funzione *getJSONFromUrl* che prende in ingresso l'URL a cui si trova il file JSON. All'interno del primo *try* presente nel codice viene stabilita la connessione con il server per richiedere la pagina presente all'URL preso in ingresso; la risposta del server viene in seguito salvata all'interno di una variabile InputStream che viene poi codificata nel secondo *try* in una stringa secondo lo standard *ISO 8859* utilizzato per la codifica dei caratteri a 8 bit.

Nel terzo ed ultimo *try* viene creato un oggetto di tipo JSON che riporta la stringa di testo in coppie nome/valore secondo lo schema originario.

Il download viene eseguito per due file diversi con due funzioni differenti:

- *dbVersionCheck* è utilizzata per il download del file check.json che viene sempre scaricato ed è quello in cui è contenuta solamente la versione del database online. Essa viene estrapolata attraverso la funzione *getInt* appartenente alla classe *JSONObject* e confrontata con la vecchia versione: soltanto se risulta che la versione presente online è aggiornata rispetto a quella che si trova già sul dispositivo allora viene chiamata la funzione *dbUpdate* [18].

```
public int dbVersionCheck(Context context, int oldVersion){
        int newVersion = 1;
        try {
                JSONObject JSONCheck;
                JSONCheck = getJSONFromUrl
                        (FeedReaderContract.FeedEntry.URL_CHECK);

                 JSONObject data = JSONCheck.getJSONObject
                        (FeedReaderContract.FeedEntry.TAG_DATA);
                newVersion = data.getInt
                        (FeedReaderContract.FeedEntry.TAG_VERSION);

                if(oldVersion < newVersion){
                        dbUpdate(oldVersion, newVersion);
                }
        }
        catch (JSONException e) {
            e.printStackTrace();
        }
        db.close();
        return newVersion;
}
```

- *dbUpdate* scarica il file update.json per l'immagazzinaggio di tutti i dati presenti sul database online. Di seguito viene illustrato l'estratto di codice relativo all'inserimento dei dati in una sola tabella: l'estrapolazione dei contenuti avviene tramite la funzione *getInt* per i valori di tipo numerico e *getString* per i valori di tipo stringa. Una volta raccolti tutti i dati essi vengono salvati dentro un oggetto di tipo *ContentValue* [19] che viene utilizzato come parametro di ingresso per il metodo *insert* [16] che ha il compito di inserire i valori all'interno del database.

  All'interno di *dbUpdate* viene richiamata anche la funzione storeImage che verrà illustrata nel paragrafo 4.3.

```
public boolean dbUpdate(int oldVersion, int newVersion){
try {

	JSONObject JSONUpdate;
	JSONUpdate = getJSONFromUrl
		(FeedReaderContract.FeedEntry.URL_UPDATE);
	...
	...

	JSONObject data = JSONUpdate.getJSONObject
			(FeedReaderContract.FeedEntry.TAG_DATA);
	JSONArray poi = data.getJSONArray
			(FeedReaderContract.FeedEntry.TAG_POI);
	for(int i = 0; i < poi.length(); i++){
		try{
			element = poi.getJSONObject(i);

			id = element.getInt
				(FeedReaderContract.FeedEntry.TAG_ID);
			nome = element.getString
				(FeedReaderContract.FeedEntry.TAG_NOME)
			try{
				icona = element.getJSONObject
					(FeedReaderContract.FeedEntry.TAG_ICONA);

				idIcona = icona.getInt
					(FeedReaderContract.FeedEntry.TAG_ID);
				file = icona.getInt
					(FeedReaderContract.FeedEntry.TAG_FILE);
				ext = icona.getInt
					(FeedReaderContract.FeedEntry.TAG_EXT);
				...

				esito = storeImage(file, ext);
				if(esito){
					values.clear();
					values.put(FeedReaderContract.
						FeedEntry.COLUMN_ICONA_ID,idIcona);
					if(ext.equals("gif")){
						file = idIcona + "." + ext;
						values.put(FeedReaderContract.
						FeedEntry.COLUMN_ICONA_FILE, file);
					}else{
						values.put(FeedReaderContract.
						FeedEntry.COLUMN_ICONA_FILE, file);
					}
					if(ext.equals("gif")){
						values.put(FeedReaderContract.
						FeedEntry.COLUMN_ICONA_EXT, "png");
					}else{
						values.put(FeedReaderContract.
						FeedEntry.COLUMN_ICONA_EXT, ext);
					}
					...
					db.insert(FeedReaderContract.FeedEntry.
					TABLE_ICONA, "null", values);
				}
			}catch(JSONException e){
				icona = null;
				esito = false;
			}

			...

			values.clear();
			db.insert(FeedReaderContract.FeedEntry.
					TABLE_POI, "null", values);
		}catch(JSONException e){
			e.printStackTrace();
		}

		...
}
```

4.2. Visualizzazione dei POI sulla mappa

Una volta completato il download del database, è stato possibile ottenere una corretta distribuzione dei POI sulle mappe di Google inserendo negli opportuni *ArrayList* le informazioni relative ai POI.

```
private ArrayList<LatLng> locations;

private ArrayList<Marker> myMarker;

private ArrayList<Drawable> imagesMarker;

private ArrayList<Drawable> imagesPopup;
```

- *ArrayList<LatLng> locations*: al suo interno sono contenuti oggetti di tipo *LatLng* che consistono in coppie di coordinate in gradi riferite alla latitudine e la longitudine del POI [28];
- *ArrayList<Marker> myMarker*: è il vettore nel quale vengono inseriti tutti i punti di interesse sottoforma di oggetti marker. Questo tipo di oggetto contiene gli attributi:
  - *Position*, posizione del POI sulla mappa sottoforma di oggetto *LatLong*,
  - *Title*, stringa di testo visualizzata nel popup al tocco dell'utente sul marker,
  - *Snippet*, testo aggiuntivo o sottotitolo del popup,
  - *Icon*, immagine visualizzata sulla mappa in corrispodenza delle coordinate di Position. [29]
- *ArrayList<Drawable> imagesMarker*: in questo vettore viene salvata l'icona (per questa applicazione è una puntina da disegno) che viene visualizzata sulla mappa per ogni punto di interesse;[30]
- *ArrayList<Drawable> imagesPopup*: immagazzina tutte le immagini che sono contenute nel popup che viene visualizzato cliccando sui POI indicati sulla mappa.

La funzione che si occupa dell'inserimento dei valori in questi vettori si chiama *setMyArrayList*.

```
public void setMyArrayList() {

      ...

      String selection = FeedReaderContract.FeedEntry.COLUMN_ICONA_ID + " = ?";
      String snippetText = getString(R.string.distance) + ": ";
      float lat, lon;


       String[] projection = {
             FeedReaderContract.FeedEntry.COLUMN_POI_ID,
             FeedReaderContract.FeedEntry.COLUMN_POI_LATITUDINE,
             FeedReaderContract.FeedEntry.COLUMN_POI_LONGITUDINE,
             FeedReaderContract.FeedEntry.COLUMN_POI_NOME,
             FeedReaderContract.FeedEntry.COLUMN_POI_ICONA,
             FeedReaderContract.FeedEntry.COLUMN_POI_DESCRIZIONE_BREVE,
        };

        Cursor c = db.query(
             FeedReaderContract.FeedEntry.TABLE_POI,
                  projection,
             null,
             null,
             null,
             null,
             null
         );

        if(c.moveToFirst())
        {
             for (int i = 0; i < c.getCount(); i++) {

                   id = c.getInt(c.getColumnIndexOrThrow(FeedReaderContract.
                         FeedEntry.COLUMN_POI_ID));
                  name = c.getString(c.getColumnIndexOrThrow
                         (FeedReaderContract.FeedEntry.COLUMN_POI_NOME));
                   lat = c.getFloat(c.getColumnIndexOrThrow
                     (FeedReaderContract.FeedEntry.COLUMN_POI_LATITUDINE));
                   lon = c.getFloat(c.getColumnIndexOrThrow
                   (FeedReaderContract.FeedEntry.COLUMN_POI_LONGITUDINE));
                   try{
                         icon = c.getInt(c.getColumnIndexOrThrow
                         (FeedReaderContract.FeedEntry.COLUMN_POI_ICONA));
                   }catch(IllegalArgumentException e)
                   {
                         icon = 0;
                   }
```

```
LatLng position = new LatLng(lat, lon);
                    locations.add(position);

                          imagesMarker.add(getResources().getDrawable
                                (R.drawable.icona_pin));

                    myMarker.add(myMap.addMarker(new MarkerOptions()
                          .position(position)
                          .title(name)
                          .snippet(snippetText + "")
                          .icon(BitmapDescriptorFactory
                          .fromResource(R.drawable.icona_pin))));

                    if(icon == 0){
                          imagesPopup.add(getResources().
                                getDrawable(R.drawable.stemma_cesena));
                    }else{
                          try{

                       String[] selectionArgs = {String.valueOf(icon)};
                             String[] projectionIcon = {
                                FeedReaderContract.FeedEntry.COLUMN_ICONA_
                            FILE,
                                 FeedReaderContract.FeedEntry.COLUMN_ICONA_EXT,
                                };

                                Cursor c = db.query(
                                      FeedReaderContract.FeedEntry.TABLE_
                                      ICONA, ,nocInoitcejorp

                                 selection,
                                 selectionArgs,
                                 null,
                                 null,
                                 null
                                   );

                                 cIcon.moveToFirst();
                              file = cIcon.getString(cIcon.getColumnIndexOrThrow
                            (FeedReaderContract.FeedEntry.COLUMN_ICONA_FILE));
                              ext = cIcon.getString(cIcon.getColumnIndexOrThrow
                            (FeedReaderContract.FeedEntry.COLUMN_ICONA_EXT));
                              filePath = FeedReaderContract.
                                    FeedEntry.PATH_STORE + file;

                               img = Drawable.createFromPath(filePath);
                                    imagesPopup.add(img);
                              }catch(IllegalArgumentException e)
                              {
                                    imagesPopup.add(getResources().
                                    getDrawable(R.drawable.stemma_cesena));
                              }
                    }
                    c.moveToNext();

                    }
        }
}
```

La prima operazione effettuata è l'apertura del database richiamando la funzione *getReadableDatabase* [16] per poi effettuare l'estrapolazione dei dati della tabella POI attraverso la funzione *query* [24].

Essa in base ai parametri passati, genera le query necessarie per interrogare il database e recuperare i dati necessari. I parametri che la funzione richiede in ingresso sono sette:

- il nome della tabella in cui deve essere eseguita la query;
- la lista delle colonne da restituire;
- un filtro per stabilire quali righe restituire (corrisponde alla clausola SQL WHERE);
- un array di stringhe per inserire dinamicamente alcuni valori nella SELECT;
- un filtro che corrisponde alla clausola SQL GROUP BY;
- un filtro che corrisponde alla clausola HAVING;
- un filtro che corrisponde alla clausola ORDER BY.

La funzione *query* restituisce un oggetto di tipo *Cursor*: questo oggetto fornisce l'accesso in modalità di lettura al result set restituito dalla query. Esso è ciclato per avere accesso a tutti i dati della tabella POI.

Per ogni singola tupla di questa tabella viene effettuata un'ulteriore interrogazione, sempre attraverso il metodo *query*, per il recupero dei dati dell'icona associata al POI.

Per ciclare l'elemento *Cursor* sono stati utilizzati i metodi *moveToFirst* che sposta il cursore alla prima tupla e *moveToNext* che porta il cursore sul record successivo [26].

Per estrapolare i dati dal cursore invece sono state scelte le funzioni *getInt* per i dati numerici di tipo intero, *getFloat* per i dati numerici di tipo razionale e *getString* per i dati testuali.

Figura 26 – Visualizzazione dei marker sulla mappa

Ciò che viene visualizzato a video è quello che si vede in figura 26. Effettuando un tocco su uno dei marker compare il popup del POI selezionato in cui si leggono nome e distanza dell'utente e viene mostrata l'immagine principale del punto selezionato (Figura 27).

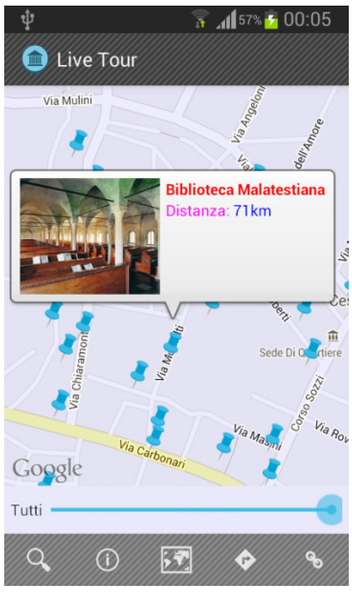

Figura 27 – Popup di un POI

4.3. Dettaglio dei POI

L'ultima parte di lavoro consiste nell'implementazione del codice per la visualizzazione del dettaglio.

Figura 28 – Dettaglio di un POI

Con un ulteriore tocco sul popup appare una schermata in cui sono riportate tutte le informazioni utili sul punto di interesse selezionato: telefono, indirizzo, orari di apertura.

Queste informazioni sono contenute all'interno del database, vengono lette e poi passate all'activity della scheda del dettaglio.

Entrambe le operazioni vengono effettuate all'interno del metodo *OnInfoWindowClick* in questo modo:

o Lettura:

```
String[] projection = {
      FeedReaderContract.FeedEntry.COLUMN_POI_ID,
      FeedReaderContract.FeedEntry.COLUMN_POI_NOME,
      FeedReaderContract.FeedEntry.COLUMN_POI_GALLERY,
      FeedReaderContract.FeedEntry.COLUMN_POI_ORARI,
      FeedReaderContract.FeedEntry.COLUMN_POI_TELEFONO,
      FeedReaderContract.FeedEntry.COLUMN_POI_INDIRIZZO,
      FeedReaderContract.FeedEntry.COLUMN_POI_VIDEO,
      FeedReaderContract.FeedEntry.COLUMN_POI_URL_LINK,
      FeedReaderContract.FeedEntry.COLUMN_POI_DESCRIZIONE,
      FeedReaderContract.FeedEntry.COLUMN_POI_DESCRIZIONE_BREVE,
};
Cursor c = db.query(
      FeedReaderContract.FeedEntry.TABLE_POI,
      projection,
      null,
      null,
      null,
      null,
      null
);

if(c.moveToFirst())
{
      for (int i = 0; i < c.getCount(); i++) {
            nome = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_NOME));
            if(marker.getTitle().equals(nome))
            {
                  id = c.getInt(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_ID));
                  idGallery = c.getInt(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_GALLERY));
                  telefono = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_TELEFONO));
                  orari_apertura = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_ORARI));
                  indirizzo = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_INDIRIZ-
            ZO));
                  path_video = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_VIDEO));
                  url_link = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_URL_LINK));
                  descrizione = c.getString(c.getColumnIndexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_DESCRIZIO-
            NE));
                  descrizione_breve    =    c.getString(c.getColumnIn-
dexOrThrow
                  (FeedReaderContract.FeedEntry.COLUMN_POI_DESCRIZIO-
                  NE_BREVE));

                  break;
            }
            c.moveToNext();

      }
}
```

- Passaggio dei dati all'activity:

```
Intent intent = getIntent();

id = intent.getIntExtra(FeedReaderContract.FeedEntry.TAG_ID, 0);
idGallery = inent.getIntExtra(FeedReaderContract.FeedEntry.
                                                TAG_GALLERY, 0);
nome = intent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_NOME);
indirizzo = intent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_INDIRIZZO);
orari_apertura = inent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_ORARI_DI_APERTURA);
telefono = intent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_TELEFONO1);
path_video = intent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_VIDEO);
url_link = intent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_URL_LINK);
descrizione_breve = intent.getStringExtra(FeedReaderContract.
                                FeedEntry.TAG_DESCRIZIONE_BREVE);
descrizione = intent.getStringExtra(FeedReaderContract.FeedEntry.
                                                TAG_DESCRIZIONE);
```

Viene creato un *intent* che, come suggerisce il nome, è un'"intenzione", una sorta di richiesta da parte di un'applicazione di poter accedere ad una risorsa o componente che sia in grado di esaudire la sua richiesta.

A volte l'*intent* è esplicito e quindi si conosce a priori chi è in grado di risolverlo.

In altri casi invece non è possbile sapere chi risolve l'*intent* che viene quindi definito come implicito.

Gli *intent* espliciti, come nel caso descritto, sono il modo con cui due activity diverse possono scambiare dati tra loro attraverso la funzione *putExtra*. Questa funzione permette di associare un particolare valore ad una chiave ti tipo stringa.

Inviare un *intent* ad una activity in certi casi significa attivarla e chiederle di processare i dati dell'*intent*.

L'activity che riceve l'*intent* può recuperare i dati utilizzando la funzione *getExtra*[27]:

```
Intent intent = new Intent(this, DettaglioActivity.class);

intent.putExtra(FeedReaderContract.FeedEntry.TAG_ID, id);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_GALLERY, idGallery);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_NOME, nome);
intent.putExtra(FeedReaderContract.FeedEntry.
                        TAG_ORARI_DI_APERTURA, orari_apertura);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_INDIRIZZO,   indiriz-
zo);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_TELEFONO1, telefono);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_VIDEO, path_video);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_URL_LINK, url_link);
intent.putExtra(FeedReaderContract.FeedEntry.TAG_DESCRIZIONE,  descri-
zione);
intent.putExtra(FeedReaderContract.FeedEntry.
                        TAG_DESCRIZIONE_BREVE, descrizione_breve);

startActivity(intent);
```

Dalla schermata dettaglio è anche possibile accedere alla galleria, al video e ad un indirizzo web riferito al POI tramite i bottoni presenti nell'action bar in basso. Nel caso in cui questi contenuti non ci siano, al tocco viene visualizzato un *Toast* con il messaggio che indica che il materiale non è disponibile (Figura 29).

Figura 29 – Messaggio Toast

Le immagini della galleria sono scaricate e salvate nel dispositivo attraverso la funzione storeImage, rigorosamente posta all'interno di un *AsynkTask*.

```
public boolean storeImage(String file, String ext)
{
      String url = FeedReaderContract.FeedEntry.
                              URL_DOWNLOAD_ICON + file;
      String filePath = file;
      try{
            Bitmap img = downloadImage(url);
            if(img!=null)
            {
                  fos = mainContext.openFileOutput(filePath,
                                          Context.MODE_PRIVATE);

                  if(ext.equals("jpg")){
                        img.compress(CompressFormat.JPEG, 100, fos);
                  }else if (ext.equals("png")) {
                        img.compress(CompressFormat.PNG, 100, fos);
                  }else if (ext.equals("gif")){
                        img.compress(CompressFormat.PNG,100,fos);
                  }
                  fos.close();
            }
      }catch(IOException e){
            e.printStackTrace();
            return false;
      }
      return true;
}
```

Il download dell'immagine è affidato al metodo *downloadImage* che preso in ingresso l'indirizzo web a cui risiede l'immagine, restituisce un oggetto *Bitmap*. Questo oggetto è poi salvato nel dispositivo con un metodo di compressione scelto in base all'estensione dell'immagine [31].

5. Conclusioni

Live Tour ha raggiunto gli obiettivi prefissati all'inizio dell'implementazione: è una guida completa e interattiva di Cesena, utile per chi desidera visitare la città senza affidarsi ad agenzie turistiche; è inoltre in grado di fornire le stesse funzionalità anche il lingua inglese.

Il prossimo passo potrebbe essere quello di approdare sul mercato delle App di Android, per rendere la guida fruibile a tutti gli utenti che ne avessero bisogno.

Live Tour è una applicazione completa, ma non per questo priva di potenziali migliorie e/o aggiornamenti: si può pensare ad esempio ad una migliore gestione delle immagini, ad aggiunte di funzionalità per adattarsi meglio alle esigenze di chi ne usufruisce. Un esempio potrebbe essere la realizzazione di percorsi guidati enogastronomici pensati con la collaborazione dei locali del luogo che dovrebbero mettersi a disposizione per offrire sconti sulle consumazioni in cambio di pubblicità attraverso l'applicazione.

Un'altra direzione in cui sarebbe utile far crescere l'applicazione è l'aspetto grafico, in modo tale da risultare più "accattivante" per chi deve scegliere se scaricarla sul suo dispositivo.

Bibliografia

[1] Introduzione Tesi “Guida Turistica multilingua mobile” di Rachid Ghmid

[2] The Next Web: “The history of the smartphone” - http://thenextweb.com/mobile/2011/12/06/the-history-of-the-smartphone/

[3] “Black Berry Operating system” di Liam Barrett e Mary O’Riordan - http://garryowen.csisdmz.ul.ie/~cs5212/resources/oth4.pdf

[4] About Android - http://developer.android.com/about/index.html

[5] Android Italy: “Cos’è Android? La storia del sistema operativo mobile di Google” - http://www.androiditaly.com/articoli/speciali/189-cose-android-la-storia-el-sistema-operativo-mobile-di-google.html

[6] In Apple: “Che cosa è IOS? Storia del sistema operativo per iPhone, iPad e iPod touch”- http://www.inapple.it/articoli/34-speciali/211-cose-ios-storia-del-sistema-operativo-per-iphone-ipad-e-ipod-touch.html

[7] Supermedia: “Tecnoguida: Sistemi operativi mobile” - http://www.supermedia.it/tecnoguide/tecnoguida-sistemi-operativi-mobile

[8] Androidiani: “Il mercato smartphone 2012: Android e IOS cosstituiscono l’87,6% secondo IDC” - http://www.androidiani.com/news/mercato-smartphone-2012-android-ed-ios-costituiscono-l87-6-secondo-idc-155152

[9] Androidiani: “Dicembre 2013: 800 milioni di smartphone android vs 300 milioni di iPhone” - http://www.androidiani.com/dispositivi-android/cellulari/dicembre-2013-800-milioni-di-smartphone-android-vs-300-milioni-iphone-154638

[10] Location-Based: Fundamentals and operation, Axel Kupper; John Wiley & Sons, 2005

[11] Introducing JSON - http://www.json.org

[12] Extensible Markup Language (XML) - http://www.w3.org/XML/

[13] Your Inspiration Web: “Vuoi utilizzare JSON ma non sai da dove iniziare?” - http://www.yourinspirationweb.com/2010/03/12/vuoi-utilizzare-json-ma-non-sai-da-dove-iniziare/

[14] Lezioni di basi di dati, Paolo Ciaccia, Dario Maio; Esculapio, 2002

[15] DB-Main: The modeling Framework - http://www.db-main.eu/?q=en

[16] Developer Android: Saving Data in SQL database - http://developer.android.com/training/basics/data-storage/databases.html

[17] Developer Android: SQLiteOpenHelper – http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

[18] Developer Android: package org.json - http://developer.android.com/reference/org/json/package-summary.html

[19] Developer Android: ContentValues - http://developer.android.com/reference/android/content/ContentValues.html

[20] devAPP: Eseguire task in modo asincrono con AsyncTask - http://android.devapp.it/t014-eseguire-task-in-modo-asincrono-con-asynctask

[21] Developer Android: Keeping Your App Responsive - http://developer.android.com/training/articles/perf-anr.html

[22] Developer Android: AsynkTask - http://developer.android.com/reference/android/os/AsyncTask.html

[23] Developer Android: SharedPreferences - http://developer.android.com/reference/android/content/SharedPreferences.html

[24] HTML.it: SQLite è il database naturale per applicazioni Android - http://www.html.it/articoli/la-gestione-dei-database-in-android-1/

[26] Developer Android: Cursor - http://developer.android.com/reference/android/database/Cursor.html

[27] Developer Android: Starting Another Activity - http://developer.android.com/training/basics/firstapp/starting-activity.html

[28] Developer Android: LatLng - http://developer.android.com/reference/com/google/android/gms/maps/model/LatLng.html

[29] Developer Android: Marker - http://developer.android.com/reference/com/google/android/gms/maps/model/Marker.html

[30] Developer Android: Drawable - http://developer.android.com/reference/com/google/android/gms/maps/model/Marker.html

[31] Developer Android: Bitmap - http://developer.android.com/reference/android/graphics/Bitmap.html

Indice