Elaborato dell'esame di Stato
su
'Tourm - Gestione audioguide'

Riccardo Calligaro, classe 5IA

ITIS C. Zuccante A. S. 2020/2021

Abstract

The current work aims at analysing and designing the prototype of a mobile application, available both on iOS and Android, intended for use by an eighteenth-century villa that wants to reorganise its audioguide mechanism so that visitors can use their personal devices. The general requirements for such a service will be studied, with particular attention to the basic functionalities that the application must offer; the characteristics of the network infrastructure and the backend will also be evaluated. Furthermore, the administration panel used by the staff to access the actual visitor count and to close or open access to the rooms will be analysed. The tools made available by the various programming languages, frameworks and libraries that allowed the development of some of the integrated the development of some of the functionalities integrated in the prototype under examination. An important part will also be devoted to security, a fundamental aspect of any application. Finally, the results achieved and possible improvements to be made in the future will be analysed in order to obtain a quality product.

Indice

1 Introduzione

Tourm si propone di raggiungere i seguenti obiettivi:

- sviluppare una **applicazione per smartphone**, disponibile sia su ambiente iOS che Android, destinata all'utente finale per consentirgli di:
  - scaricare l'applicazione attraverso un codice QR presente all'interno della villa
  - accedere alla parte espositiva dell'applicazione con il codice del biglietto acquistato online o allo sportello di vendita.
  - interagire con le esposizioni automatizzando la riproduzione di spiegazioni audio pre-registrate
  - interagire con le esposizioni scannerizzando un codice QR che permette di leggere un articolo e ascoltare l'audioguida associata
  - ricevere notifiche automatiche sulla disponibilità di visite guidate o eventi
- sviluppare una **portale web amministrativo** destinato agli organizzatori per consentirgli di:
  - accedere al conteggio effettivo dei visitatori
  - visualizzare statistiche riguardo le esposizioni e le interazioni fatte dagli utenti
  - chiudere o aprire l'accesso alle stanze
- sviluppare un'**infrastruttura di rete** sicura, veloce e affidabile che permette ai visitatori di:
  - collegarsi a una rete wireless gratuita senza accesso ad internet
  - utilizzare l'applicazione senza nessun costo di dati mobili
  - utilizzare la rete in modo sicuro senza doversi preoccupare di eventuali minacce

  e ai gestori di:
  - utilizzare dei dispositivi specifici per accedere al pannello di amministrazione

1.1 Diagramma dei casi d'uso

Gli **Use Case Diagram** (diagrammi dei casi d'uso) sono dei diagrammi dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso, aiutano a definire e organizzare i requisiti.

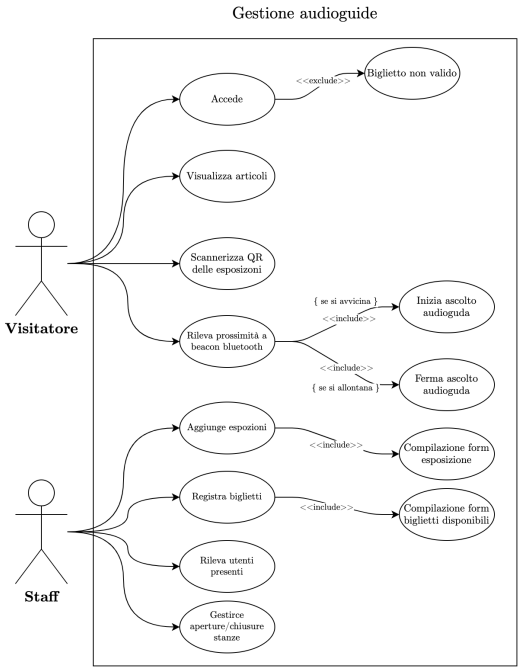

Figura 1: Diagramma dei casi d'uso

1.2 WBS

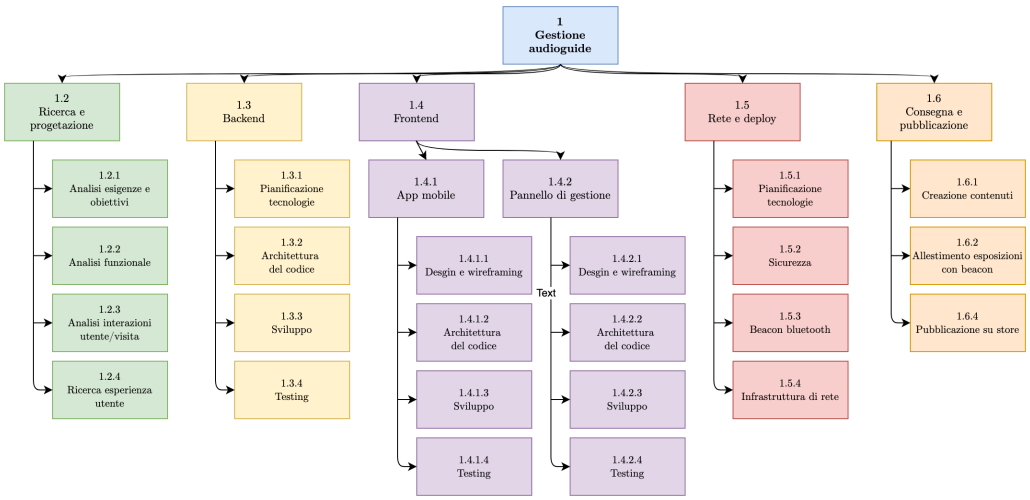

Figura 2: Work Breakdown Structure

2 Infrastruttura di rete

Avere un'infrastruttura di rete progettata in maniera efficace è di **fondamentale importanza**, essa è la spina dorsale di tutte le operazioni quotidiane. Quando si progetta un'infrastruttura non bisogna soltanto considerare la topologia di rete; è necessario infatti tenere conto anche di:

- **dispositivi hardware**: server, datacenter, personal computer, router, switch. È importante fare delle scelte adatte e mirate al caso d'uso.
- **applicazioni e software**: applicazioni utilizzate dall'azienda, come web server, sistemi di gestione del contenuto e sistema operativo, ad esempio Linux.

Quali sono invece alcuni aspetti da considerare?

- **Efficienza**: creare un'infrastruttura di rete efficiente riduce al minimo i tempi di down e assicura la stabilità della rete.
- **Scalabilità**: una solida infrastruttura di rete supporta la crescita senza doverla riprogettare.
- **Sicurezza**: deve fornire sicurezza e protezione da spam, malware e virus. Deve anche mantenere i dati al sicuro.
- **Portata**: deve permettere agli utenti di essere connessi alla rete, indipendentemente dalla loro posizione.

2.1 I beacon Bluetooth

Per gestire la riproduzione automatizzata delle audioguide è stato deciso di usare dei beacon Bluetooth, dei piccoli dispositivi disponibili a un prezzo relativamente basso.

2.1.1 Cosa sono

I **beacon Bluetooth** sono dei trasmettitori hardware - una classe di dispositivi *Bluetooth low energy* che trasmettono il loro identificatore ai dispositivi elettronici portatili vicini. Questa tecnologia permette a smartphone, tablet e altri dispositivi di eseguire azioni quando sono in prossimità di essi. Il beacon, tuttavia, non è in grado di rilevare alcun segnale. Non richiede una connessione Internet e agisce come un emittente entro un raggio breve.[1]

2.1.2 Perché i beacon Bluetooth?

Attraverso la precisa localizzazione (la distanza di trasmissione si aggira intorno ai 10-30 metri), i beacon creano un **engagement mirato**: nel contesto opportuno, al momento opportuno, al pubblico interessato. Per questo e per i **costi accessibili** (un dispositivo costa in media 10 euro) sono la scelta più adatta al caso d'uso desiderato.

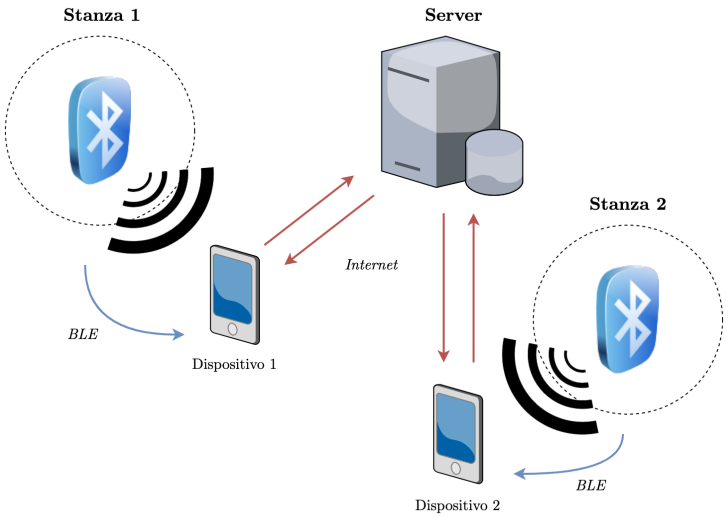

Figura 3: Comunicazione con Beacon

2.1.3 In sintesi

Riassumendo, la comunicazione sarà composta da due elementi:

- **Dal presentatore**: il beacon bluetooth, questo spedisce soltanto informazioni. L'informazione standard dei beacon consiste in un *UUID*, e solo un valore major o minor. Per esempio: UUID: 9e747915-5996-4b5b-8d3f-335ba7d0745e Major ID: 1 Minor ID: 2. Il trasmettitore non fa niente altro che inviare questa informazione ogni frazione di secondo. Ogni *UUID* sarà associato ad ogni stanza così il client sarà in grado di rilevare in che posizione l'utente si trova.
- **Dall'osservatore**: l'applicazione mobile, sarà responsabile di ricevere i segnali e di far partire o fermare l'audio in base alla posizione.

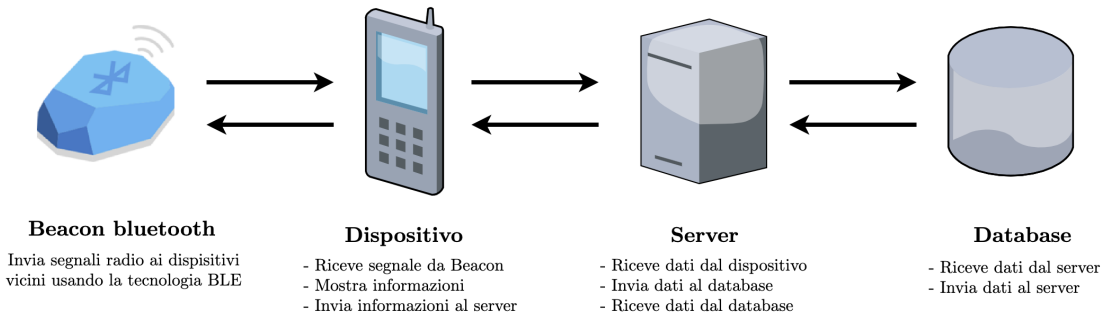

Figura 4: Architettura della comunicazione

2.1.4 Utilizzare un beacon bluetooth

Per realizzare un beacon Bluetooth vi sono farie opzioni:

1. Acquistare un dispositivo già programmato con alimentazione propria ad un costo di circa 5/10 euro.
2. Utilizzare un modulo Bluetooth BLE (ad esempio l'HM-10) e alimentarlo via USB
3. Traformare il proprio telefono cellulare in un trasmettitore BLE con una semplice applicazione

A seguito della natura dimostrativa di questo progetto è stata scelta la terza opzione.

2.2 Internet of Things

Tutta questa interconnettività forma quello che viene chiamato **Internet of Things**, un termine che descrive l'estensione della connessione Internet alle più svariate tipologie di oggetti, in questo caso i beacon bluetooth. I dati rilevati grazie ad appositi sensori possono essere scambiati e comunicati tramite Internet e gli oggetti possono essere monitorati e gestiti da remoto.

2.3 Rete

Il migliore modo per rappresentare la rete nel suo insieme è attraverso una **topologia**. Essa è una **mappa** che descrive i dispositivi e come sono collocati. Esistono due tipi di topologie:

- **Topologia fisica**: indica la disposizione fisica dei dispositivi e dei mezzi trasmissivi
- **Topologia logica**: evidenzia il modo in cui gli host accedono al mezzo trasmissivo per inviare dati

Essendo il luogo non definito, è impossibile realizzare la topologica fisica. Riguardo la topologia logica la maggior parte delle reti wireless utilizza una **topologia a stella**. Essa consiste in un nodo gateway (in questo caso un **Access Point**, AP) a cui tutti gli altri nodi si collegano. I vantaggi di questa scelta sono sicuramente:

- prestazione della rete veloce e affidabile.
- i nodi/dispositivi difettosi possono essere identificati e isolati rapidamente.

Mentre gli svantaggi sono:

- la portata è limitata al raggio di trasmissione di un singolo dispositivo.
- se il nodo gateway fallisce, l'intera rete smette di funzionare

Ai vari dispositivi dei vistatori è necessario aggiungere:

- il **server**
- il **router**
- uno o più **switch**
- i vari **access point** a cui si collegano i dispositivi dei vistatori

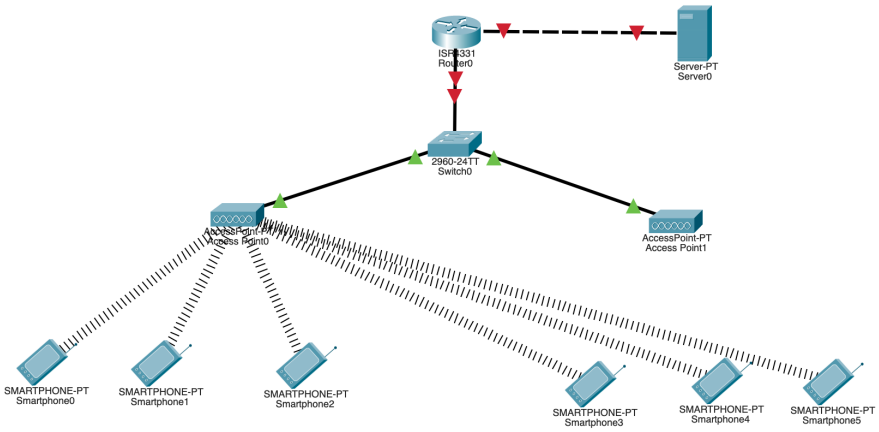

Figura 5: Topologia logica

2.4 Server

Come server è stata utilizzata una **virutal machine** con **Ubuntu Server 21.04**, una distribuzione ufficiale di Ubuntu dedicata all'ambito server. La principale differenza con la distribuzione standard è l'ambiente desktop: mentre Ubuntu Desktop include un'interfaccia grafica utente, Ubuntu Server no. Inoltre la versione server include anche pacchetti standard che si concentrano principalmente sulla connettività e sicurezza.[5].

Per fornire le API è stato usato **nginx**, un web server/reverse proxy leggero ad alte prestazioni. Questa virtual machine è contenuta in un container **Docker**, un sistema per l'automazione del deployment che considera i container come macchine virtuali modulari estremamente leggere, offrendo la flessibilità di creare, distribuire, copiare e spostare i container da un ambiente all'altro in modo sicuro.[2]

2.5 Uso delle API

L'applicazione non accede direttamente al database, usa delle API. Creare un livello extra di astrazione non è solo utile per la sicurezza, ma anche per la scalabilità.

2.5.1 Cosa sono le API

Un'API, interfaccia di programmazione di un'applicazione, è un insieme di comandi formalizzati che consentono alle applicazioni software di comunicare tra loro in modo uniforme e di sfruttare i servizi di base per creare servizi incentrati sul cliente. Quindi in questo caso il server esegue una applicazione che espone delle API che fanno comunicare il client Flutter con il database MySQL.

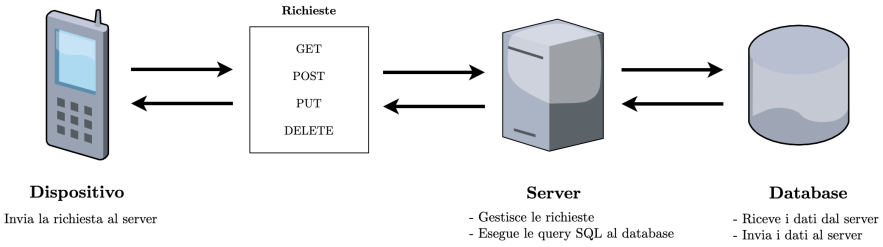

Figura 6: L'interazione tra dispositivo, server e database

3 Il backend

Il back-end, cioè la parte di ogni sito web o applicazione che gli utenti non vedono è composto da:

- **server**: è il luogo in cui l'applicazione viene contenuta ed eseguita
- **applicazione**: l'applicazione che viene eseguita nel server, in questo caso si occupa di esporre delle API di tipo REST e di gestire il pannello di amministrazione
- **database**: la base dati, contiene tutti i dati necessari per l'utilizzo dell'applicazione

3.1 Il database

Il database svolge una parte fondamentale di questa applicazione, per descriverlo ci sono vari metodi, io tratterò lo schema entità relazioni e quello logico. Come DBMS è stato usato MySQL.

3.1.1 Schema E/R

Lo schema E/R traduce le problematiche reali in uno schema concettuale facilmente capibile, senza occuparsi di come sarà costruito il database.

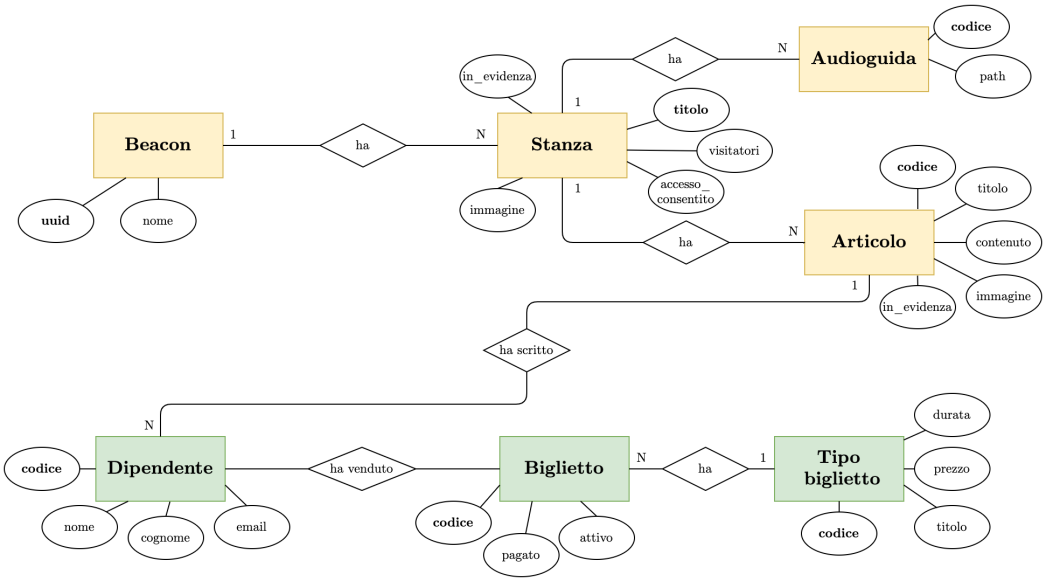

Figura 7: Schema E/R

3.1.2 Schema logico

L'obiettivo di questa fase è pervenire, a partire dallo schema concettuale, a uno schema logico che lo rappresenti in modo fedele, "efficiente" e indipendente dal particolare DBMS adottato. Questo modello logico viene sviluppato anche per arricchire il modello concettuale visto precedentemente definendo esplicitamente le colonne di ogni entità e introducendo entità operative e transazionali.

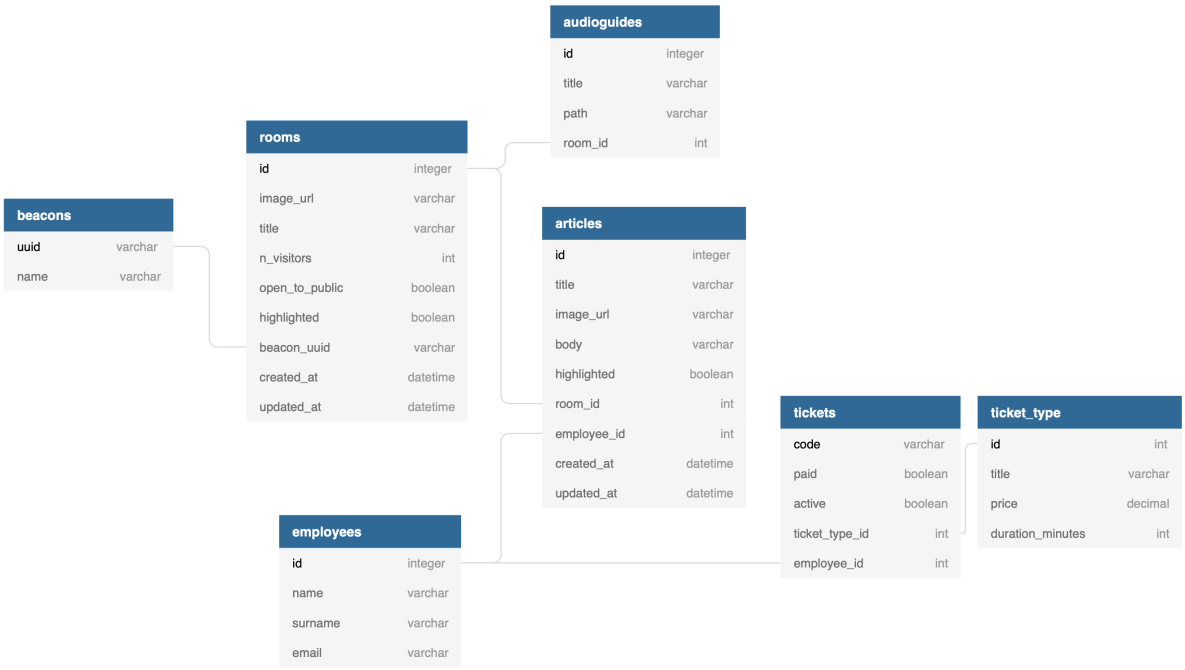

Figura 8: Schema logico

3.1.3 API in PHP

Le API realizzate con il framework **Lumen** offrono vari endpoint che conformano allo stile **REST** (Representational state transfer). Alcune di queste sono:

- `/api/v1/rooms`: lista delle stanze
- `/api/v1/audioguides`: lista delle audioguide
- `/api/v1/articles`: lista degli articoli
- `/api/v1/audioguides/{beacon_id}`: audioguide per il beacon indicato

Questi restituiscono i dati ottenuti dal database con le query in formato **JSON**, un formato adatto all'interscambio di dati fra applicazioni client/server.

3.1.4 Query SQL

Per ottenere i dati il server esegue delle interrogazioni al database, di cui prima è stata descritta la struttura. Di seguito vengono riportate alcune query significative; i parametri vengono indicati con le parentesi graffe, nel codice PHP sono stati ineriti tramite PDO per evitare le SQL injection.

Audioguide per il beacon indicato

```
01 | SELECT * FROM audioguides INNER JOIN rooms ON audioguides.room_id =
       rooms.id WHERE beacon_id={beacon_code}
```

Articoli di una stanza

```
01 | SELECT employees.name, employees.surname, articles.id, articles.title,
       articles.body
02 | FROM articles
03 | INNER JOIN room_articles ON room_articles.room_id = {room_id}
04 | INNER JOIN employees ON articles.employee_id = employees.id
```

Aritcoli scritti da un autore (ricerca per nome)

```
01 | SELECT employees.name, employees.surname, articles.id, articles.title,
       articles.body
02 | FROM articles
03 | INNER JOIN employees ON articles.employee_id = employees.id
04 | WHERE employees.name LIKE {name}
```

4 Il client

4.1 App mobile

L'applicazione viene scaricata dai visitatori attraverso un QRCode. Avviata e inserito il codice presente nel biglietto, l'app viene impostata per la visita alla villa. Avviene poi l'interazione con i beacon bluetooth, quando l'app rileva che l'utente sta entrando nel range del trasmettitore avvisa il sistema che il visitatore è nella stanza e manda in esecuzione il commento audio. Quando l'app non percepisce più il beacon avvisa l'app che l'utente è uscito e termina la presentazione.

4.1.1 L'architettura del codice

L'architettura di un sistema software è la "forma" data a quel sistema da coloro che la costruiscono. Per "forma" si intende la divisione di tale sistema in componenti, nella disposizione di essi e nei modi in cui tali componenti comunicano tra loro. Lo scopo è quello di facilitare lo sviluppo, la distribuzione, il funzionamento e la manutenzione del sistema software in esso contenuto.[3] Come architettura è stato deciso di usare la **clean architecture**, il cui scopo principale è quello di permettere al business di adattarsi ai cambiamenti della tecnologia e delle interfacce cambiando il meno codice possibile. Aiuta a rimuovere quello stretto accoppiamento tra la logica di business e il livello di presentazione.[4] Vi sono tre livelli principali:

- **Presentazione:** è il livello responsabile della UI, dove è presente il codice Flutter.

- **Dominio:** è dove si definiscono le entità e i casi d'uso. Non vi sono classi concrete in quel livello, solo interfacce (classi astratte). Tutte le implementazioni nel livello più sotto, il *data*.
- **Data**: questo livello è responsabile di tutte le implementazioni dei casi d'uso e delle interfacce. Solitamente contiene classi per l'accesso ai dati del database o di una sorgente remota.

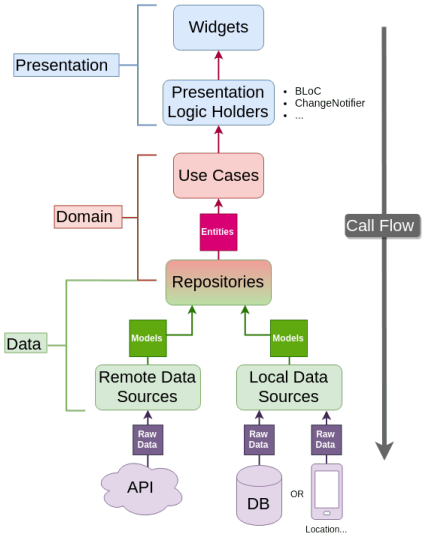

Figura 9: Lo schema della clean architecture

4.1.2 La gestione dello stato

Un'applicazione non è mai del tutto statica e richiede in quasi tutti i casi di:

- tracciare i cambiamenti
- dare l'opportunità all'utente di poter interagire con la UI
- aggiornare la UI dinamicamente in caso di cambiamenti nelle informazioni mostrate.

Per gestire tutti questi aspetti in Flutter è possibile utilizzare una soluzione più elegante rispetto ai widget *stateful*, il **BloC Pattern**. Questo pattern, che è implementato su Flutter con la libreria *flutter_bloc* permette di gestire il flow dell'applicazione tramite una stream di eventi.

4.2 Pannello di amministrazione web

Il pannello di amministrazione permette di:

- gestire la vendita dei biglietti
- osservare il conteggio effettivo dei visitatori
- chiudere o aprire l'accesso alle stanze

L'accesso è limitato, può essere visualizzato soltanto dallo staff. Per realizzarlo è stato usato il framework PHP **Laravel**. L'applicativo accede allo stesso database delle API Rest ma è totalmente separato in termini di codice.

5 Sicurezza

L'infrastruttura di rete, i servizi e i dati contenuti nei dispositivi collegati in rete sono risorse personali e aziendali fondamentali. Nessuna soluzione singola può proteggere la rete dalla varietà di minacce esistenti. Per questo motivo, la sicurezza deve essere implementata su **più livelli**, utilizzando più di una soluzione. Se un componente di sicurezza non riesce a identificare e proteggere la rete, possono farlo altri.

Le misure di sicurezza implementate sono state:

- Sql Injection

5.1 Docker

6 Conclusioni

Riferimenti bibliografici

[1] Dave Addey. *iBeacons*. URL: https://web.archive.org/web/20131203014352/http://daveaddey.com/?p=1252.

[2] *Docker: get started*. URL: https://www.docker.com/get-started.

[3] Dario Frongillo. *Architettura software: introduzione, importanza e compiti di un sw architect*. URL: https://italiancoders.it/architettura-software-introduzione-importanza-e-compiti-di-un-sw-architect/.

[4] joaosczip. *Clean Architecture — A little introduction*. URL: https://dev.to/joaosczip/clean-architecture-a-little-introduction-4ag6.

[5] *Ubuntu Server Guide*. URL: https://ubuntu.com/server/docs.