Progettazione e produzione multimediale

Gabriele Bruni

Dicembre 2019

1 Introduzione

Di seguito si illustra il lavoro svolto per lo sviluppo di un applicazione per dispositivi mobili dedita alla gestione degli interventi che si effettuano sulle filari di un azienda vinicola. In particolare, l'azienda necessita di un applicativo che permetta di annotare le operazioni svolte su vari campi di loro proprietà. La soluzione adottata permette di creare nuovi campi e gestire le operazioni svolte su tali appezzamenti, aggiungendone di nuove o selezionandole da un elenco di preferite. Fornisce infine un resoconto complessivo delle operazioni svolte.

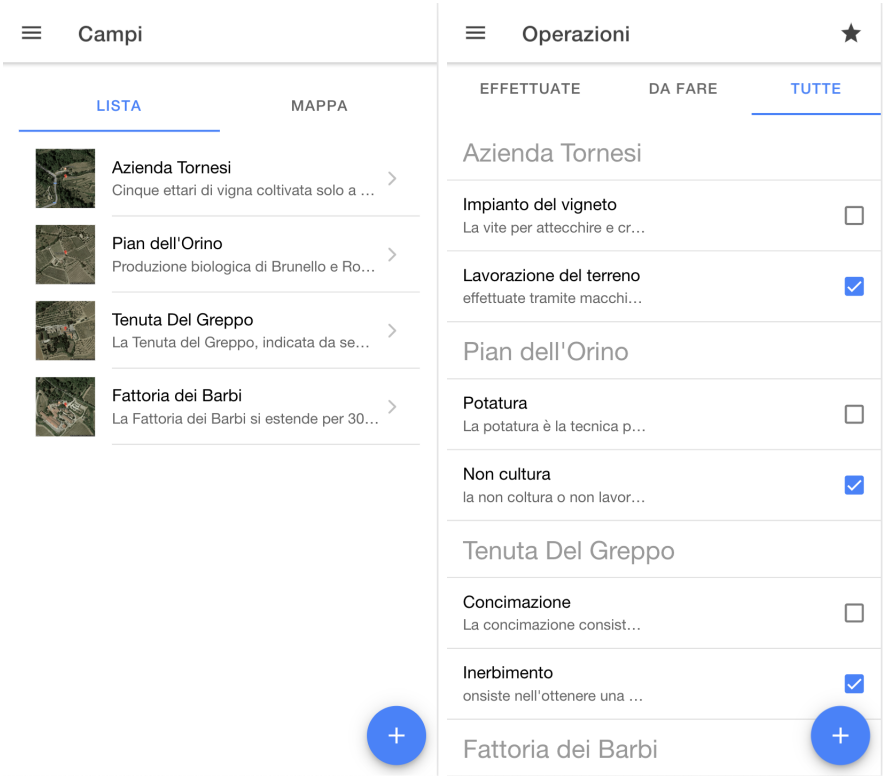

Figure 1: Le due sezioni principali dell'applicazione.

2 Progettazione

Lo sviluppo dell'applicativo è passato attraverso varie fasi, di seguito elencate.

2.1 Obiettivo e requisiti

Nella prima fase si è stabilito il tipo di applicazione da sviluppare e alcuni vincoli riguardanti la piattaforma di sviluppo e le tecnologie da utilizzare. Si è identificato il target di utenza e gli scenari di utilizzo del software, per

definire i casi d'uso e progettare un'esperienza utente adeguata. In particolare, si sono definito le seguenti voci:

1. **Obiettivo:** realizzazione di un applicazione mobile per la gestione degli interventi effettuati sulle vigne di uno o più terreni di un azienda vinicola.
2. **Requisiti:**
   - *Piattaforma di utilizzo*: Android[2]
   - *Tecnologie*: Ionic Framework.[14]
   - *Utente* : operatore di campo di un azienda vinicola.
   - *Caso d'uso*: operatore al lavoro in un campo coltivato o comunque in un contesto agricolo. Essendo il contesto di utilizzo prettamente agricolo, e quindi anche probabilmente scomodo, l'esperienza utente deve risultare immediata: tutte le principali funzioni devono essere raggiungibili in pochi semplici passi, e si dovrà provvedere la possibilità di inserire le operazioni nei vari campi selezionandole da un elenco di precompilate, per evitare il tedioso, oltre che scomodo quando si sta in piedi e si utillizza il dispositivo con una sola mano, lavoro di scrittura all'operatore.

In questa fase si è anche analizzato cosa offrisse la "concorrenza": si è ricercato applicazioni simili e se ne è studiato le funzionalità oltre che che le scelte effettuate per quanto riguarda l'interfaccia utente. Interessanti sono alcune soluzioni adottate da Agricolus [1] e Farmdok [10]: entrambe sono applicazioni troppo complesse e con troppe funzionalità per il nostro caso d'uso, ma si è preso spunto da queste per alcune scelte riguardanti l'interfaccia utente, come la visualizzazione a mappa.

2.2 Funzionalità

Nella seconda fase si è passato è definire le funzionalità del software. Nel dettaglio, si sono definite le seguenti funzioni per la versione iniziale del software:

- *Creazione dei terreni agricoli*: possibilità di creare nuovi campi sui quali sarà possibile successivamente aggiungere nuove operazioni. In fase di creazione sarà possibile inserire informazioni quali: il nome, una breve descrizione, il proprietario, la superficie occupata, l'altitudine e la posizione geografica.
- *Modifica dei terreni agricoli esistenti*: possibilità di modificare successivamente le informazioni definite al momento della creazione.
- *Eliminazione dei terreni agricoli esistenti.*
- *Aggiunta di nuove operazioni*: possibilità di aggiungere nuove operazioni effettuate su un terreno. Per ogni operazione sarà necessario specificare: il nome, una breve descrizione, l'operatore che ha effettuato l'operazione e la data dell'operazione. Sarà possibile inoltre selezionare le operazione da aggiungere da un elenco di operazioni preferite, per velocizzare il processo di aggiunta.
- *Modifica delle operazioni esistenti*: possibilità di modificare le informazioni inserite in fase di aggiunta delle operazioni.
- *Eliminazioni delle operazioni esistenti.*
- *Aggiunta di operazioni preferite*: mantenimento di un elenco di operazioni che si presuppone l'operatore effettuerà maggiormente. Tali operazioni saranno poi selezionabili al momento di aggiunta di una nuova operazione sul campo. Il software compilerà automaticamente le voci della nuova operazione con quelle dell'operazione preferita.
- *Eliminazione delle operazioni preferite.*
- *Resoconto complessivo*, divise per terreno agricolo, *di tutte le operazioni effettuate.*
- *Aggiunta sul calendario* di un server remoto [9] di un *evento riguardante l'operazione effettuata.*

2.3 Wireframing

Si è poi passato alla progettazione del layout e della struttura dell'interfaccia utente dell'applicazione.In questa fase si è utilizzato un software di prototyping quale Mockflow [16].

Fulcro dell'applicazione è la side-bar, attraverso la quale si raggiunge tutte le funzionalità principali, vedi figura 2.

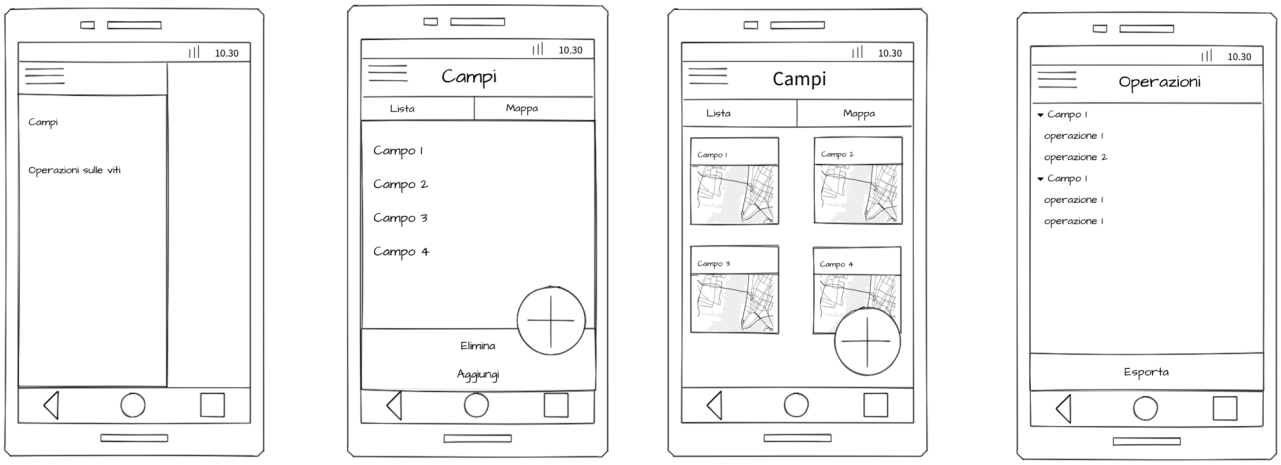

Figure 2: La side-bar.

Dalla voce "Campi" si accede ai campi di proprietà dell'azienda vinicola. Sono possibili due tipi di visualizzazione: lista e mappa. Con quest'ultima modalità è possibile accedere ad ogni campo attraverso un'interfaccia a schede che mostra l'immagine satellitare di ogni campo, vedi figura 2. In fase di aggiunta è possibile, oltre che definire tutte le informazioni specificate nella sezione precedente, localizzare il campo su una mappa e ritagliare una porzione di mappa per impostarla come immagine nella vista "mappa", vedi figura 3. Dalla voce "Operazioni sulle viti" si accede al resoconto delle operazioni, divise per campo, vedi figura 2.

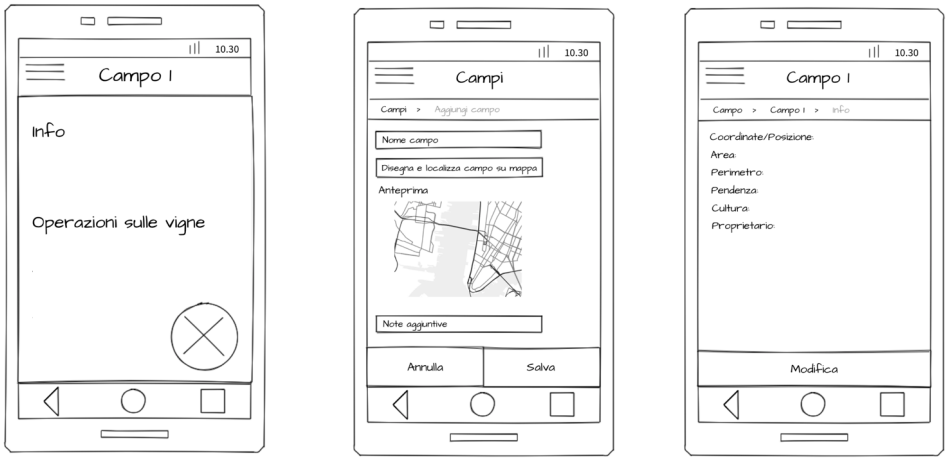

Figure 3: La sezione Campi.

La voce "Operazioni sulle vigne" consente di visualizzare ed aggiungere in ogni singolo campo gli interventi effettuati, selezionandoli da una lista di interventi prestabiliti ed offrendo la possibilità di aggiungerne di nuovi. In fase di aggiunta è possibile definire tutte le voci definite nel capitolo precedente, vedi figura 4.

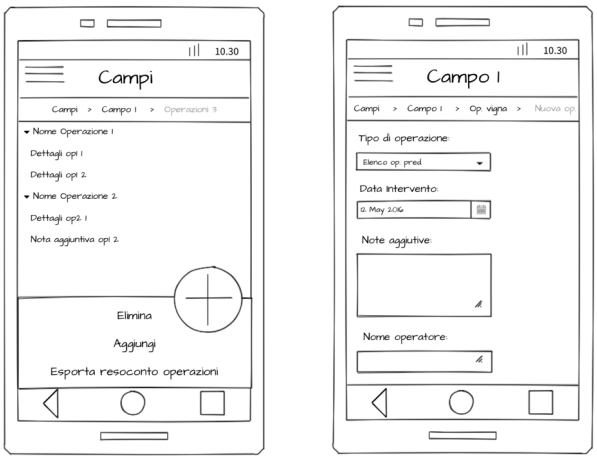

Figure 4: La vista Operazioni di un campo.

2.4 Architettura software

L'ultima fase della progettazione, che precede l'implementazione, è quella che ha visto la definizione dell'architettura software.

2.4.1 Motivazioni

La nostra applicazione utilizza il back-end principalmente come livello di persistenza dei dati. Questo significa che molte delle responsabilità sono spostate nel front-end dell'applicazione. Da qui la necessità di ideare un architettura software che permetta di definire un ordine nelle numerose responsabilità di cui il front-end si fa carico. L'obiettivo della nostra architettura è quello di rendere il software mantenibile nel tempo, permettere l'aggiunta di nuove funzionalità facilmente, ed accelerare, oltre che rendere più logico e "ragionato", il processo di sviluppo.

2.4.2 Le soluzioni già adottate

Nella fase iniziale dello sviluppo dell'architettura ci si è documentati su quali fossero le soluzioni maggiormente adottate da persone con più esperienza nel campo, e come queste soluzioni si potessero adottare al nostro contesto. Interessanti sono i lavori di Kristian Poslek [18], Balram Chavan [5] e Bartosz Pietrucha [17], sui quali la nostra architettura si basa, oltre che di Martin Fowler [11], sui quale i precedenti lavori poggiano.

2.4.3 Un cenno alle tecnologie utilizzate

Basandosi il nostro front-end su Angular [3] (vedi capitolo 3), l'architettura risentirà inevitabilmente della sua presenza. Senza scendere nel dettaglio, per ora basterà sapere che due dei concetti principali attorno al quale si sviluppa un applicazione Angular sono quello di *Component* e *Service*: ogni componente definisce una classe che contiene dati e logica dell'applicazione ed è associato a un modello HTML che definisce una vista da visualizzare in un ambiente di destinazione. Per i dati o una logica non associati a una vista specifica e che si desidera condividere tra i componenti, si crea una classe di servizio. Per maggiori informazioni si rimanda al capitolo 3.

2.4.4 L'architettura

L'idea alla base della nostra architettura è quella di dividere il sistema in layer separati, ognuno con una propria responsabilità, e di inserire ogni modulo nel corretto layer, vedi figura 7. Ne deriva una suddivisione del sistema sul piano orizzontale data dai seguenti layer: **core layer**, **abstraction layer**, **presentation layer**. Questa suddivisione del sistema influenza anche il modo di comunicare dei vari componenti: il presentation layer può parlare al core layer solo attraverso l'abstraction layer. Di seguito illustriamo le responsabilità principali dei vari strati:

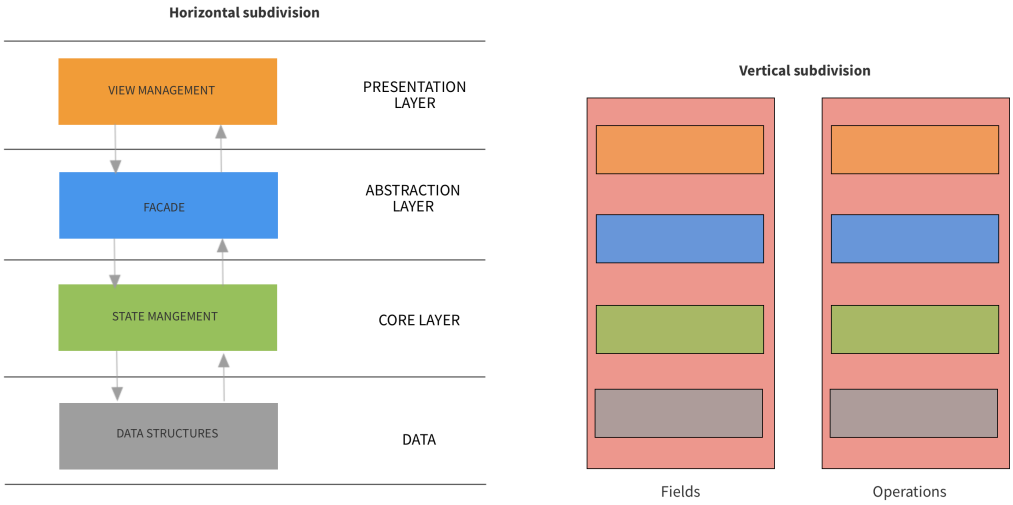

Figure 5: Suddivisione orizzontale e verticale del sistema.

- **Component/Presentation layer:** iniziamo a descrivere il layer di presentazione. Questo è il luogo dove si trovano tutte le classi Component di Angular. La sue uniche responsabilità sono quelle di presentare e delegare. In altre parole, presenta l'interfaccia utente e delega le azioni dell'utente al core layer, attraverso il livello di astrazione. Sa cosa visualizzare e cosa fare, ma non sa come gestire le interazioni dell'utente. Non si occupa della logica di business.
- **Facade/Abstraction layer:** nella nostra architettura, il suo unico compito è quello di astrarre il livello dei servizi dal livello dei componenti. Svolge quindi il ruolo di Facade. I blocchi in questo livello sono servizi i cui metodi vengono richiamati dal livello presentazione e vengono quindi reindirizzati ai servizi corrispondenti nel core layer. In questo modo, le modifiche apportate al core layer non influiscono mai sul livello dei componenti. Osservare i metodi pubblici del nostro Facade ci da una visione d'insieme dei casi d'uso di alto livello che la nostra applicazione deve gestire. Sottolineiamo tuttavia che neanche il livello di astrazione è un luogo per implementare la logica di business. Qui vogliamo solo connettere il livello di presentazione alla nostra logica, astraendo il modo in cui è collegato. Fra i vantaggi di avere questa astrazione annoveriamo il fatto di potere cambiare il modo in cui gestiamo lo stato senza nemmeno toccare il livello di presentazione. A titolo di esempio, durante la fase sviluppo, lo stato è stato prima gestito a livello locale e non permanente usando semplici vettori, e, successivamente si è migrato il tutto su una soluzione che consentisse la gestione dello stato dell'applicazione in modo permanente, un database su un server Django [12]. Un altro dei vantaggi che abbiamo adottando questa soluzione riguarda la gestione centralizzata dei servizi. Nell'estendere le funzionalità della nostra applicazione è probabile che definiremo un numero sempre maggiore di servizi che inietteremo successivamente nei nostri componenti. Indipendentemente dalla buona convenzione di denominazione adottata, sarà necessario sempre più tempo per trovare il nome del servizio giusto, e le difficoltà saranno ancora maggiori nel caso in cui un esterno decida di mettere mano al codice. Utilizzando quindi il Facade si risolve anche questo tipo di problema: tutti i servizi necessari sono presenti e richiamabili da un unico luogo.
- **Service/Core layer:** l'ultimo livello è il core layer. Qui è dove viene implementata tutta la logica dell'applicazione principale e la gestione dello stato. Più in generale, questo livello si occupa dell'intera manipolazione dei dati, dal loro recupero alla loro creazione, e del loro successivo salvataggio in modo permanente o temporaneo, oltre che a gestire qualsiasi comunicazione con servizi esterni all'applicazioni.

Sul piano verticale è inoltre possibile individuare un altra suddivisione. Qui l'idea è quella di separare l'applicazione in moduli rappresentanti diverse funzionalità, vedi figura 7. In particolare, ciascuno layer verticale si occupa di diverse logiche di business. A titolo di esempio, nella nostra applicazione, troviamo due layer verticali separati: uno si occupa della logica e della presentazione riguardanti i terreni agricoli, l'altro della logica e della presentazione riguardanti la gestione delle operazioni. I due layer non condividono funzionalità in comune e delagano all'altro funzioni che non gli appartengono: lo strato che si occupa di terreni agricoli reindirizzerà ad una vista e ad una relativa logica presente nello strato delle operazioni quando si tratterà di dovere aggiungere un operazione a tale

campo. La comunicazione fra i due strati avviene usando tecniche standard come servizi per la condivisione dei dati o parametri passati nell'URL. Nella figura 11 è presente il diagramma UML delle classi della nostro sistema. Questo mostra come l'architettura appena descritta è stata adattata alla nostra applicazione. Nel diagramma sono state inserite solo le relazioni fra le classi necessarie a comprendere come il sistema funziona. Altre relazioni secondarie al fine della comprensione del suddetto funzionamento sono state omesse in favore della leggibilità.

2.5 Interfaccia utente

Essendo la piattaforma di riferimento Android[2], le linee guida da seguire per quanto riguarda la progettazione dell'interfaccia utente fanno riferimento al Material Design [15]. Il vantaggio di usare un framework come Ionic[14] per la creazione della UI, vedi capitolo 3, risiede in una delle sue feature chiamata *Adaptive Styling*. Adaptive Styling è una funzionalità integrata di Ionic Framework che consente agli sviluppatori di utilizzare la stessa base di codice per più piattaforme. Ogni componente Ionic adatta il suo aspetto alla piattaforma su cui è in esecuzione l'app. Ad esempio, i dispositivi Apple, come iPhone e iPad, utilizzano le linea guida iOS di Apple. Allo stesso modo, i dispositivi Android utilizzano il linguaggio visuale di Google, il Material Design.

2.5.1 Colori

Ionic ha nove colori predefiniti che possono essere usati per cambiare il colore di molti componenti. Questi colori si adattano automatiacamente agli standard della piattaforma di destinazione. Per quanto riguatda Android, i colori sono quelli mostrati in figura 6.

Nel nostro software abbiamo quindi deciso di applicare lo stesso colore alla barra superiore e allo sfondo. Questa è una pratica suggerita dal Material Design, in quanto permette di focalizzare l'attenzione sul contenuto dell'applicazione anzichè sulla sua struttura [6]. Si è scelto il colore "light" predefnito di Ionic, vedi figura 6. Per quanto riguarda testo ed icone, si è continuato ad utilizzare le scelte predefinite di Ionic: testi neri con opacità ridotta a seconda dell'enfasi che si vuole dare al contenuto, in rispetto del WCAG standards [21], il quale richiede un contrasto di colore 4.5: 1 tra testo e sfondo per il testo normale e 3: 1 per testo grande.
Per quanto riguarda il floating action button (FAB), le linee guida suggeriscono che questo debba essere uno degli elementi più ricoscibili della UI, in quanto si presuppone associato ad attività importanti, e, pertanto, sia necessario usare un colore che garantisca il giusto contrasto con quelli utlizzati per lo sfondo e la app-bar [6]. La scelta è ricaduta sul colore "primary" di Ionic. Si è utilizzato lo stesso colore anche per i segment e le checkbox, in quanto comunque rappresentanti funzioni basiliari dell'applicazione che necessitano essere messi in risalto rispetto a sfondo ed app-bar, e la cui presenza non preclude la visibilità del FAB. Per quanto riguarda l'icona per accedere alle operazioni preferite si è scelto di utilizzare la stessa scala di grigi utlizzata per il testo: non è una funzione che si suppone l'utente utilizzi con frequenza e che necessiti di visibilità aggiuntiva rispetto ad altri elementi della UI. Inoltre le linea guida incoraggiano ad usare gli stessi colori per tutti gli elementi della app-bar [6]. Per le icone riguardanti le funzioni di eliminazione, si è scelto il colore "Danger" predefinito di Ionic.

2.5.2 Layout

Sebbene non sia possibile considerare la nostra app propriamente responsive - il target è un applicazione android per smartphone - si è fatto comunque uso in modo consistente di griglie in ogni vista, al fine di assicurarsi se non altro che l'applicazione fosse correttamente visualizzabile su ogni schermo e con ogni orientazione, come stabiliscono le linea guida android per un applicazione di qualità [4]. In futuri aggiornamenti sarà pertanto più facile rendere l'applicazione propriamente responsive ottimizzando il layout per schermi di grosse dimensioni.

2.5.3 Component

Si è scelto di utilizzare un menù laterale per quanto riguarda l'accesso alle due viste principali, "Campi" ed "Operazioni". La scelta ricadeva fra il menù laterale ed una bottom/tab bar. Le linea guida sconsiglivano l'uso della bottom-bar con soli due elementi [7], ed inoltre si è deciso di riservare lo spazio superiore per altri componenti, quali i tabs per la selezione di viste con layout differenti ("Campi") o dati differenti ("Operazioni"). La scelta è quindi ricaduta sul menù laterale. È stato poi utlizzato un floating action button per tutte le operazioni di aggiunta e di modifica (campi ed operazioni) ,ed un sliding button per tutte le operazioni di eliminazione, prestando così particolare attenzione alla consistenza dell'esperienza utente. checkbox sono stati posti immediatamenti adiacenti agli elementi cui si riferivano, sempre in accordo alle linea guida.

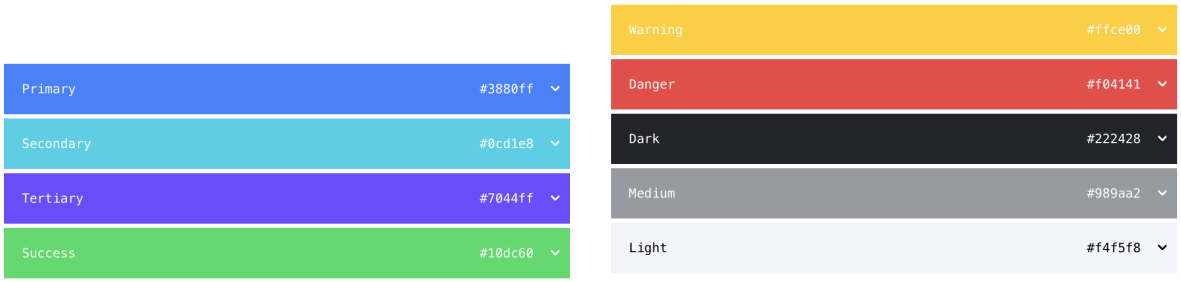

Figure 6: I 9 colori predefiniti di Ionic aderenti alle specifiche del Material Design.

3 Tecnologie utilizzate

. Di seguito forniamo una rapida descrizione dello stack tecnologico utilizzato per realizzare il software.

3.1 Front-end

3.1.1 Ionic Framework

Ionic Framework [14] è un toolkit open-source per la creazione di UI per app mobili e desktop che usa tecnologie web quali HTML, CSS e Javascript. Ionic fornisce quindi componenti per la UI estendendo la normale libreria HTML e metodi per la gestione delle gesture, delle animazioni e delle interazioni con l'interfaccia utente. Si interfaccia poi con framework quali Angular [3] e Apache Cordova [8] per creare applicazioni web, mobili e desktop ibride. Dalla versione 4 è stato anche aggiunto il supporto a React, Vue ed aggiunta la possibilità di usarlo utilizzando semplicemente del vanilla Javascript. È stato inoltre aggiunto il supporto a Ionic Capacitor, la controparte proprietaria di Apache Cordova. Ionic fornisce inoltre una interfaccia a riga di comando CLI che ci aiuta nel processo di creazione e gestione dell'applicazione, automatizzando molto processi e rendendo lo sviluppo dell'applicazione più facile e rapido.

Figure 7: Riassunto delle funzionalità principali di Ionic Framework. Si ringrazia Maximilian Schwarzmüller per la slide.

3.1.2 Angular

Angular [3] è un framework Javascript client-side usato per sviluppare applicazioni web. Angular usa Typescript, un super-set di Javascript mantenuto da Microsoft basato sulle specifiche ECMAScript 6 [20]. In altre parole è un framework che facilita la creazione di applicazioni single web page reattive, che necessitano di aggiornare spesso la vista senza dover ricaricare la pagina. È il cuore della nostra applicazione. Un app Ionic è in fin dei conti una semplice applicazione Angular con in aggiunta una semplice libreria di web component. Di seguito forniamo una ponaramica dell'architettura di un applicazione Angular:

- *NgModules:* i mattoni di base di un'applicazione Angular sono gli NgModules, questi forniscono un contesto di compilazione per i componenti e raccolgono il codice in set funzionali. Un'app ha sempre almeno un modulo radice che abilita il bootstrap, e in genere ha molti moduli funzione.

- *Components:* ogni applicazione Angular ha almeno un componente, il root component, che collega una gerarchia di componenti con il modello a oggetti del documento della pagina (DOM). Ogni componente definisce una classe che contiene dati e logica dell'applicazione ed è associato a un modello HTML che definisce una vista da visualizzare in un ambiente di destinazione. In altre parole, i componenti definiscono le view, le viste, che sono insiemi di elementi dello schermo che Angular può scegliere e modificare in base alla logica e ai dati del programma. Inoltre i componenti utilizzano i service, dei servizi che forniscono funzionalità specifiche non direttamente correlate alle viste. I fornitori di servizi possono essere iniettati nei componenti come dipendenze, rendendo il codice modulare, riutilizzabile ed efficiente.

- *Service e Dipendent injection:* per i dati o una logica non associati a una vista specifica e che si desidera condividere tra i componenti, si crea una classe di servizio. Un servizio è in genere una classe con uno scopo ristretto e ben definito. Angular distingue i componenti dai servizi per aumentare la modularità e la riusabilità. Separando la funzionalità relativa alla vista di un componente da altri tipi di elaborazione, è possibile rendere le classi dei componenti snelle ed efficienti. Idealmente, il lavoro di un componente abilita solo l'esperienza utente dell'applicazione. Un componente può delegare determinate attività ai servizi, come il recupero dei dati dal server, la convalida dell'input dell'utente o la registrazione direttamente sulla console. Definendo tali attività di elaborazione in una classe di servizio iniettabile, si rendono tali attività disponibili per qualsiasi componente. Si rende anche l'applicazione più adattabile iniettando diversi provider dello stesso tipo di servizio, a seconda dei casi in circostanze diverse.

- *Template, Directive e Data binding:* un template(modello) combina del codice HTML con del codice markup definito da Angular che permette la modifica degli elementi HTML prima che vengano visualizzati. Le direttive(direttive) chiamate all'interno dei template grazie al markup permettono di collegare i dati dell'applicazione, definiti nei component, ed il DOM. Esistono due tipi di data binding (associazione dei dati):

  1. il binding di eventi consente all'app di rispondere all'input dell'utente nell'ambiente di destinazione aggiornando i dati dell'applicazione nel component.
  2. L'associazione delle proprietà consente di interpolare i valori dai dati dell'applicazione, calcolati e memorizzati nel component, in HTML.

  Prima di visualizzare una view, Angular valuta le direttive e risolve la sintassi di associazione nel template per modificare gli elementi HTML e il DOM, in base ai dati e alla logica del programma. Angular supporta l'associazione dati bidirezionale, il che significa che le modifiche al DOM, come le scelte dell'utente, si riflettono anche nei dati del programma.

- *Routing:* Angular offre un servizio che consente di definire un percorso di navigazione tra i diversi stati dell'applicazione e abilitare una gerarchia fra le viste della propria app. È modellato sulle convenzioni di navigazione dei normali browser, come inserire una URL nella barra degli indirizi, fare click su collegamenti presenti all'interno di una vista, o fare click sul tasto indietro del browser. Anche Ionic dalle ultime versioni fornisce un servizio simile, ma all'interno della nostra applicazione è stato usato maggiormente ciò che angular metteva già a disposizione.

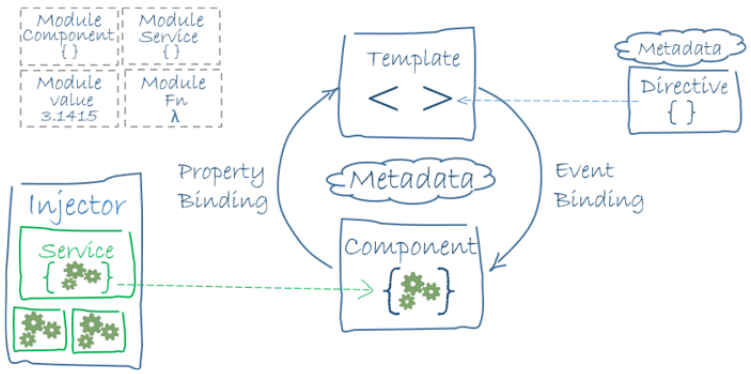

Figure 8: Una panoramica del funzionamento di un'applicazione Angular, reperebile dal sito della documentazione officiale.

3.1.3 Apache Cordova

Apache Cordova [8] è l'anello di giunzione fra applicazioni web e dispositivi mobili e desktop. Il framework permette di incapsulare codice HTML, CSS, Javascript all'interno di un contenitore che offre, grazie a dei plugin forniti dal framework stesso, accesso alle funzionalità native della piattaforma di destinazione (quale android, ios, piattaforme desktop grazie ad Electron). In altre parole consente di interfacciare una tipica applicazione scritta con le comuni tecnologie web come quelle sopra elencate, con le API native della piattaforma di riferimento (android, ios, electron). Esistono diversi componenti in un'applicazione Cordova. Il diagramma in figura 9 mostra una vista di alto livello dell'architettura di un applicazione Cordova.

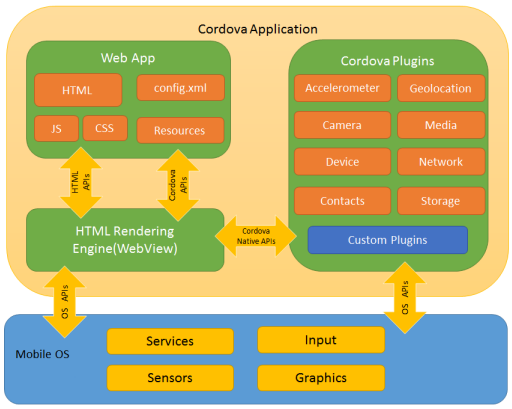

Figure 9: L'architettura di un'applicazione Cordova, reperebile dal sito della documentazione officiale.

- *WebApp:* è la parte in cui risiede il codice dell'applicazione. L'applicazione stessa è implementata come una pagina Web, e, per impostazione predefinita, è necessario un file locale denominato index.html, che fa riferimento a CSS, JavaScript, immagini, file multimediali o altre risorse per l'esecuzione. L'app viene eseguita in una WebView all'interno del wrapper dell'applicazione nativo, che viene distribuito negli app store.

- *Plugins:* i plugin sono parte integrante dell'ecosistema Cordova. Forniscono un'interfaccia fra le tecnologie Web e le API standard dei dispositivi, che consente di richiamare il codice nativo da JavaScript. Il progetto Apache Cordova mantiene una serie di plugin chiamati Core Plugin. Questi plug-in permettono di accedere alle funzionalità di base del dispositivo come batteria, fotocamera, contatti, ecc. Oltre a questi, esistono diversi plug-in di terze parti che forniscono collegamenti aggiuntivi a funzionalità non necessariamente disponibili su tutte le piattaforme.

3.2 Back-end

3.2.1 Django Web Framework

Django [12] è un framework server-side scritto in Python. In analogia alle precedenti presentazioni, forniamo anche qui una breve descrizione della sua architettura, vedi figura 10.
In un sito Web tradizionale, un'applicazione Web attende le richieste HTTP dal browser Web (o un altro client). Quando viene ricevuta una richiesta, l'applicazione risolve ciò che è necessario in base all'URL e, eventualmente, alle informazioni nei dati POST o GET. A seconda di ciò che è richiesto, può quindi leggere o scrivere informazioni da un database o eseguire altre attività per soddisfare la richiesta. L'applicazione restituirà quindi una risposta al client, spesso creando dinamicamente una pagina HTML da visualizzare per il browser inserendo i dati recuperati nei segnaposto in un modello HTML.

Le applicazioni Web di Django in genere raggruppano il codice che gestisce ciascuno di questi passaggi in file separati:

- *URLs:* sebbene sia possibile elaborare richieste da ogni singola URL tramite una singola funzione, è molto più gestibile scrivere una funzione di visualizzazione separata per gestire ciascuna risorsa. Un mappatore URL viene utilizzato per reindirizzare le richieste HTTP alla vista appropriata in base all'URL della richiesta. Tale mappatore può anche abbinare particolari schemi di stringhe o cifre che compaiono in un URL e passarli a una funzione di visualizzazione come dati.

- *View:* una vista è una funzione del gestore di richieste, che riceve richieste HTTP e restituisce risposte HTTP. Le viste accedono ai dati necessari per soddisfare le richieste tramite i modelli e delegano la formattazione della risposta ai templates.
- *Models:* i modelli sono oggetti Python che definiscono la struttura dei dati di un'applicazione e forniscono meccanismi per gestire (aggiungere, modificare, eliminare) e interrogare i record nel database.
- *Templates:* un template è un file di testo che definisce la struttura o il layout di un file (come una pagina HTML). Una vista può creare dinamicamente una pagina HTML utilizzando un template HTML, popolandolo con i dati di un modello. Un template può essere utilizzato per definire la struttura di qualsiasi tipo di file.

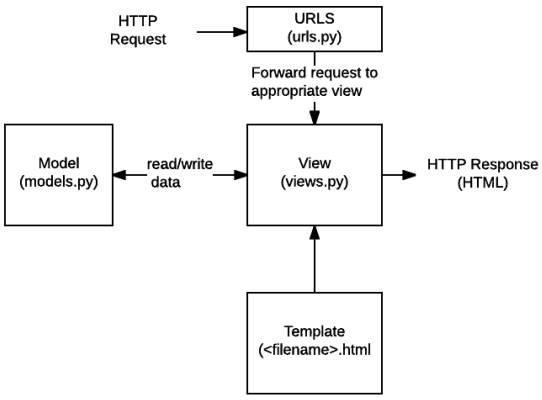

Figure 10: L'architettura di un'applicazione Django, reperebile dal sito della documentazione Mozilla.

3.2.2 Django Rest Framework

Django Rest Framework[13], è un toolkit per la creazione di Web Api basate su architettura REST. REST definisce un insieme di principi architetturali per la progettazione di un sistema. In altre parole Django Rest Framework fornisce gli strumenti per implementare l'architettura Rest all'interno di una web application, ovvero un architettura server-side capace di resituire ad un client dei dati nel formato a lui più idoneo, quale, nel nostro caso, JSON.

3.3 Un cenno ad Rxjs ed al paradigma Reactive Programming

La gestione dello stato nella nostra applicazione è stata realizzata usando la libreria Rxjs [19]. Rxjs abilita linguaggi come Javascript all'uso del paradigma di programmazione Reactive. In altre parole è una libreria per la composizione di programmi asincroni e basati su sequenze osservabili di eventi. Fornisce classi come Observable, Observer, Scheduler, Subject ed operatori come map, filter, reduce, tap, ecc. che consentono la creazione e la successiva gestione di tali sequenze o flussi di eventi.

References

[1] Agricolus. https://www.agricolus.com.

[2] Android. https://www.android.com/intl/it$_i t/gms$/.

[3] Angular. https://angular.io.

[4] Android app quality.

[5] Balram Chavan. Best practices: Building Angular Services using Facade design pattern for complex systems.

[6] Applying color to UI-Material Design.

[7] Material component guidelines. https://material.io/components/app-bars-bottom/anatomy.

[8] Apache Cordova. https://cordova.apache.org.

[9] Vineayard data analysis. Un progetto precedente effettuato pressso il mic.

[10] Farmdok. https://www.farmdok.com/en/.

[11] Martin Fowler. PresentationDomainDataLayering.

[12] Django Framework. https://www.djangoproject.com.

[13] Django Rest Framework. https://www.django-rest-framework.org.

[14] Ionic Framework. https://ionicframework.com.

[15] Material Design guidelines. https://material.io/design/foundation-overview/addition.

[16] Mockflow. https://mockflow.com/.

[17] Bartosz Pietrucha. Angular Architecture Patterns and Best Practices (that help to scale).

[18] Kristian Poslek. Anatomy of a large Angular application.

[19] Rxjs. https://rxjs-dev.firebaseapp.com.

[20] ECMAScript® 2015 Language Specification. http://www.ecma-international.org/ecma-262/6.0/.

[21] WCAG standards. https://www.w3.org/tr/understanding-wcag20/visual-audio-contrast-contrast.html.

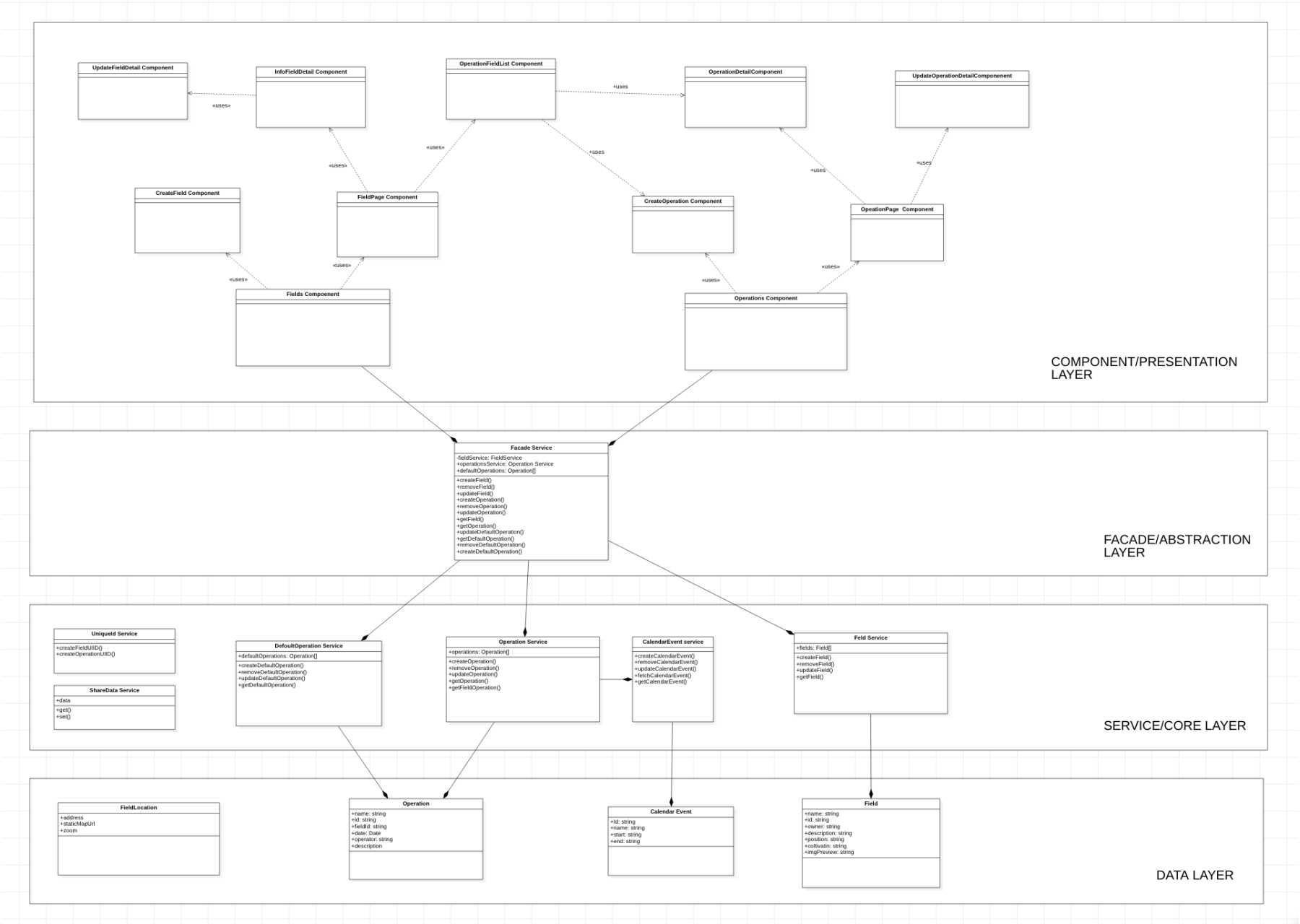

Figure 11: Il diagramma UML dell'architettura software.