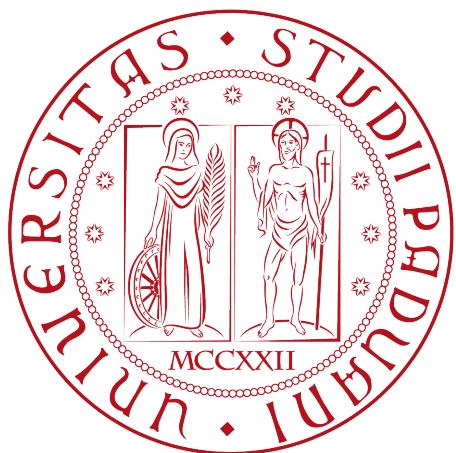


Università degli Studi di Padova
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
CORSO DI LAUREA IN INFORMATICA



**Sviluppo di un'applicazione mobile per la
gestione di eventi sportivi tramite
framework Flutter**

Tesi di laurea triennale

Relatore

Prof.Luigi De Giovanni

Laureando

Matteo Budai

ANNO ACCADEMICO 2020-2021

Matteo Budai: *Sviluppo di un'applicazione mobile per la gestione di eventi sportivi tramite framework Flutter*, Tesi di laurea triennale, © Settembre 2021.

Sommario

Il presente documento descrive lo stage da me svolto nel periodo che va dal 28/06/2021 al 20/08/2021, della durata di trecentoventi ore, presso l'azienda Sync Lab s.r.l. nella sede di Padova.

Lo stage riguarda la realizzazione di varie funzionalità per un'applicazione denominata 'Sportwill' che permette la gestione di eventi sportivi.

Gli obiettivi da raggiungere erano molteplici.

In primo luogo era richiesto il ripasso del linguaggio Java SE e dei concetti Web come Servlet, servizi Rest e Json. In secondo luogo era richiesto lo studio dei principi generali, delle best practice, dei widget e dell'architettura di Flutter e lo studio del linguaggio Dart.

In seguito si è passati allo studio del codice esistente dell'applicazione e allo sviluppo di varie funzionalità che hanno permesso di completarla rendendola utilizzabile.

Il seguente documento è stato diviso in 5 capitoli:

- **Capitolo 1:** Descrizione dell'azienda e delle metodologie utilizzate;
- **Capitolo 2:** Presentazione degli obiettivi, del Piano di Lavoro e delle attività svolte con introduzione al progetto;
- **Capitolo 3:** Descrizione del linguaggio Dart e del framework Flutter e presentazione di alcune piccole applicazioni realizzate per lo studio;
- **Capitolo 4:** Descrizione dettagliata dell'applicazione esistente e delle nuove funzionalità apportate;
- **Capitolo 5:** Resoconto conclusivo con valutazione del percorso svolto.

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	Metodologie utilizzate e principali prodotti	2
2	Descrizione dello stage e obiettivi	5
2.1	Introduzione al progetto e scopo	5
2.2	Obiettivi dello stage	5
2.3	Pianificazione del lavoro svolto	6
2.3.1	Pianificazione iniziale	6
2.3.2	Variazioni	8
2.4	Strumenti di comunicazione per lo svolgimento del lavoro	10
3	Framework Flutter e linguaggio Dart	11
3.1	Sviluppo applicazioni mobile	11
3.1.1	App native	12
3.1.2	Web app	13
3.1.3	App Ibride Web View Wrapper	14
3.1.4	App Ibride Compile to Native	14
3.2	Flutter	15
3.2.1	Dart	16
3.2.2	Componenti	17
3.2.3	Librerie Material e Cupertino	19
3.2.4	Widget	19
3.2.5	Esempi piccole applicazioni realizzate	25
4	Sportwill	29
4.1	Descrizione progetto	29
4.2	Tecnologie e strumenti	30
4.2.1	Flutter e Dart	32
4.2.2	Android Studio	32
4.2.3	GitLab	32
4.2.4	Database	32
4.2.5	Backend	32
4.3	Analisi dei requisiti	33
4.4	Implementazioni	34
4.4.1	Logo	34
4.4.2	Filtro di ricerca testo	35
4.4.3	Campi obbligatori e pagina modifica e crea	36

4.4.4	Colori	37
4.4.5	Eliminazione, Modifica e Aggiunta di un'attività	37
4.4.6	Mappa percorso	38
4.4.7	Aggiornamento automatico mappa	39
4.4.8	Mappa schermo intero	40
4.4.9	Filtro avanzato di ricerca attività	41
4.4.10	Pubblicazione applicazione Play Store	42
5	Conclusioni	43
5.1	Consuntivo finale	43
5.2	Raggiungimento degli obiettivi	43
5.3	Conoscenze acquisite	43
5.4	Valutazione personale	43
	Bibliografia	47

Elenco delle figure

1.1	Logo aziendale Sync Lab	1
1.2	Sedi Sync Lab	2
1.3	SynClinic	3
2.1	Trello	10
2.2	Google sheets	10
3.1	Sviluppo applicazioni mobile	11
3.2	Logo Android	12
3.3	Logo iOS	13
3.4	Logo Windows Phone	13
3.5	Logo Flutter	15
3.6	Statistiche dei programmati che usano Flutter per sviluppare app mobile	15
3.7	Architettura a strati di Flutter	17
3.8	Librerie Material e Cupertino	19
3.9	Stateless widget e Stateful widget	20
3.10	Scaffold widget	21
3.11	Appbar widget	22
3.12	Row e Column widgets	22
3.13	Container widget	23
3.14	Text widget	23
3.15	Icon widget	24
3.16	ElevatedButton widget	24
3.17	Applicazione base	25
3.18	Applicazione 1	26
3.19	Applicazione 2	27
3.20	Applicazione 3	28
4.1	Struttura progetto	30
4.2	Logo Android Studio	31
4.3	Logo GitLab	31
4.4	Logo DBeaver	31
4.5	Logo Spring	32
4.6	Logo Sportwill	34
4.7	Filtro di ricerca testo	35
4.8	Campi obbligatori	36
4.9	Colori	37
4.10	Mappa percorso	38

4.11 Aggiornamento automatico mappa	39
4.12 Mappa schermo intero	40
4.13 Filtro avanzato di ricerca attività	41

Elenco delle tabelle

2.1 Tabella riassuntiva della pianificazione iniziale di stage	8
2.2 Tabella riassuntiva della pianificazione di stage con variazioni	9

Capitolo 1

Introduzione

In questo capitolo viene descritta l'azienda, le metodologie utilizzate e come viene organizzato il lavoro.

1.1 L'azienda

Sync Lab nasce nel 2002 come Software house e si è trasformata rapidamente in System Integrator attraverso uno studiato processo di maturazione delle competenze tecnologiche, metodologiche ed applicative nel dominio del software.



Figura 1.1: Logo aziendale Sync Lab

In seguito all'apertura della sede principale di Napoli, Sync Lab è cresciuta esponenzialmente nel mercato ICT e ha consolidato ottimi rapporti con clienti e partner. Attualmente l'azienda ha più di 150 clienti diretti e finali e vanta un organico di oltre

200 dipendenti, una solida base finanziaria e un'ottima diffusione nel territorio italiano attraverso le sue cinque sedi: Napoli, Roma, Milano, Padova e Verona.

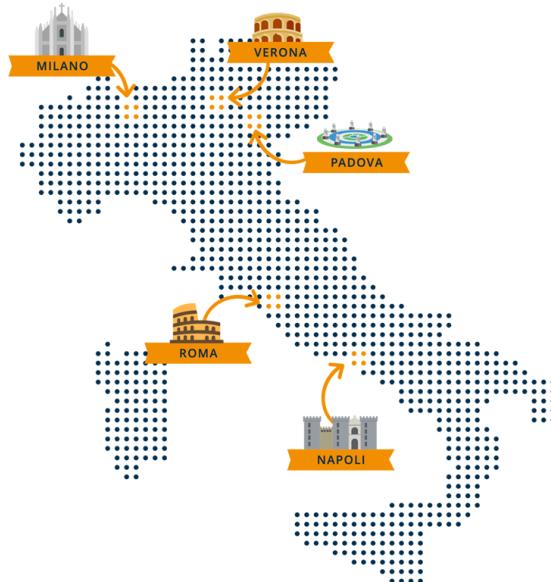


Figura 1.2: Sedi Sync Lab

Sync Lab, propone sul mercato interessanti e innovativi prodotti software, nati nel proprio laboratorio di ricerca e sviluppo. Attraverso questi prodotti, Sync Lab ha gradualmente conquistato significativamente fette di mercato nei seguenti settori: mobile, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali.

1.2 Metodologie utilizzate e principali prodotti

L'azienda adotta un modello di sviluppo agile che pone le proprie basi nel metodo Scrum. Gli stakeholders, infatti, vengono costantemente coinvolti nel processo di sviluppo del prodotto per raccogliere feedback. Gli obiettivi si possono riassumere in tre punti fondamentali:

- Comprendere attentamente il contesto operativo del cliente;
- Fornire al cliente un supporto mirato;
- Accelerare e favorire la formazione di soluzioni.

In base a questi principi Sync Lab raggiunge i propri obiettivi grazie a:

- Consulenza;
- Fornitura;
- Sviluppo;

- Manutenzione.

Nell'ambito di prodotti e innovazioni, l'azienda ne può vantare un buon numero. Tra questi troviamo:

- **SynClinic** che è un software integrato per la gestione delle strutture sanitarie, che permette di gestire, organizzare e monitorare tutte le fasi del percorso di cura del paziente;



Figura 1.3: SynClinic

- **DPS 4.0** che permette di gestire la General Data Protection Regulation(GDPR) Privacy in pochi semplici passi con una soluzione guidata per aggiornare e modificare i documenti di privacy in modo conforme agli standard di riferimento;
- **StreamLog** che permette di gestire la compliance al provvedimento del Garante per la protezione dei dati personali relativo agli Amministratori di Sistema (AdS). In particolare, permette di soddisfare requisiti fissati dal Garante;
- **StreamCrusher** che è una tecnologia che aiuta ad essere bene informati su quando bisogna prendere decisioni di business, ad identificare velocemente criticità ed a riorganizzare i processi in base a nuove esigenze;
- **Wave** che si propone come integrazione tra i mondi della Videosorveglianza e quello dei Sistemi Informativi Territoriali (GIS) abilitando il controllo totale dell'area da sorvegliare;
- **Seastream** che mette a disposizione un sistema di monitoraggio avanzato delle flotte armatoriali operative in tutto il mondo e una piattaforma integrata di servizi per gli operatori in ambito portuale.

Capitolo 2

Descrizione dello stage e obiettivi

In questo capitolo viene introdotto il progetto e viene descritto come è stato organizzato il lavoro in azienda con gli obiettivi iniziali e le variazioni rispetto a quanto pianificato inizialmente.

2.1 Introduzione al progetto e scopo

Lo scopo del progetto di stage è lo sviluppo di una piattaforma web-mobile per la gestione di eventi sportivi. È stata effettuata una fase iniziale di analisi e progettazione, basata sull'utilizzo del framework Flutter in linguaggio Dart, seguita dalla realizzazione di alcune parti dell'interfaccia mobile. Regolarmente, ci sono stati incontri diretti con il tutor aziendale Fabio Pallaro per verificare lo stato di avanzamento, chiarire eventualmente gli obiettivi, affinare la ricerca e aggiornare il piano di lavoro.

2.2 Obiettivi dello stage

Come obiettivi è stato richiesto di:

1. Realizzare le funzionalità indicate;
2. Produrre un documento Tecnico che descriva le funzionalità realizzate;
3. Rilasciare il codice sul repository che verrà indicato dall'azienda.

Notazione

Per gli obiettivi delle stage si farà riferimento ai requisiti secondo le seguenti notazioni:

- *O* per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- *D* per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;

- *F* per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Obiettivi fissati

Si prevede lo svolgimento dei seguenti obiettivi:

- Obbligatori
 - O01: Acquisizione delle competenze sulle tematiche sopra descritte e sulle attività svolte;
 - O02: Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma;
 - O03: Portare a termine le implementazioni previste con una percentuale di superamento pari all'80%.
- Desiderabili
 - D01: Portare a termine le implementazioni previste con una percentuale di superamento pari al 100%.
- Facoltativi
 - F01: Realizzazione di una nuova funzionalità per l'app che prevede la gestione Signin con il protocollo OAuth2.

2.3 Pianificazione del lavoro svolto

In questa sezione viene mostrato come è stato pianificato il lavoro e le variazioni apportate in seguito agli incontri regolari con il tutor aziendale Fabio Pallaro.

2.3.1 Pianificazione iniziale

All'inizio è stata fatta una pianificazione iniziale basata su 8 settimane con una pianificazione di 40 ore per ciascuna settimana per un totale di 320 ore. La pianificazione iniziale è la seguente:

- **Prima Settimana (40 ore)**
 - Presentazione strumenti di lavoro per la condivisione del materiale di studio e per la gestione dell'avanzamento;
 - Condivisione scaletta di argomenti;
 - Ripasso del linguaggio Java SE;

- Ripasso concetti Web (Servlet, servizi Rest, Json ecc.).

- **Seconda Settimana (40 ore)**

- Studio principi generali di Flutter e linguaggio Dart;
- Studio delle best practice Flutter;
- Studio delle architetture per i services in Flutter.

- **Terza Settimana (40 ore)**

- Studio dei widget in Flutter;
- Studio del prototipo di app SportWill oggi esistente.

- **Quarta Settimana (40 ore)**

- Implementazione del login/signup con salvataggio credenziali su localStorage con aggiunta di foto ed altre info dell’utente corrente;
- Modifica dell’interfaccia grafica nella visualizzazione ‘Elenco Uscite’ con filtri di ricerca.

- **Quinta Settimana (40 ore)**

- Modifica funzionalità ‘Modifica Uscita’;
- Implementazione della funzionalità ‘Inserisci mappa percorso’;
- Implementazione della funzionalità ‘Uscita in esecuzione/archiviata’.

- **Sesta Settimana (40 ore)**

- Implementazione della funzionalità ‘WaitForMe’ per le uscite in esecuzione.

- **Settima Settimana (40 ore)**

- Termine implementazione funzionalità ‘WaitForMe’.

- **Ottava Settimana (40 ore)**

- Termine integrazioni e collaudo finale.

La pianificazione iniziale, in termini di quantità di ore di lavoro, è stata così distribuita:

Durata in ore	Descrizione dell'attività
40	Formazione sulle tecnologie
80	Definizione architettura di riferimento e relativa documentazione
13	<i>Studio principi generali di Flutter e linguaggio Dart</i>
13	<i>Studio delle best practice Flutter</i>
14	<i>Studio delle architetture per i services in Flutter</i>
20	<i>Studio dei widget in Flutter</i>
20	<i>Studio del prototipo di app SportWill oggi esistente</i>
160	Implementazioni
20	<i>Implementazione del login/signup con salvataggio credenziali su localStorage con aggiunta di foto ed altre info dell'utente corrente</i>
20	<i>Modifica dell'interfaccia grafica nella visualizzazione 'Elenco Uscite' con filtri di ricerca</i>
10	<i>Modifica funzionalità 'Modifica Uscita'</i>
15	<i>Implementazione della funzionalità 'Inserisci mappa percorso'</i>
15	<i>Implementazione della funzionalità 'Uscita in esecuzione/archiviata'</i>
40	<i>Implementazione della funzionalità 'WaitForMe' per le uscite in esecuzione.</i>
40	<i>Termine implementazione funzionalità 'WaitForMe'</i>
40	Termine integrazioni e collaudo finale.
Totale ore: 320	

Tabella 2.1: Tabella riassuntiva della pianificazione iniziale di stage

2.3.2 Variazioni

Rispetto all'inizio sono variate solamente le funzionalità da realizzare in quanto quelle già presenti sono state suddivise con un altro stagista. Le variazioni in particolar modo sono dalla quarta alla settima settimana e sono le seguenti:

- **Quarta Settimana (40 ore)**
 - Configurazione iniziale;
 - Modifica logo;
 - Modifica dell'interfaccia grafica nella visualizzazione 'Elenco Uscite' con filtri di ricerca.
- **Quinta Settimana (40 ore)**
 - Modifica funzionalità 'Modifica Uscita' con vari fix e cambio colori;
 - Implementazione della funzionalità 'Visualizza mappa percorso'.

- **Sesta Settimana (40 ore)**

- Implementazione della funzionalità che permette di vedere la mappa a schermo intero;
- Implementazione della funzionalità che permette l'aggiornamento automatico della mappa.

- **Settima Settimana (40 ore)**

- Modifica dell'interfaccia grafica nella visualizzazione 'Elenco Uscite' con filtri di ricerca avanzati.

La pianificazione, in termini di quantità di ore di lavoro dopo la variazione è stata così distribuita:

Durata in ore	Descrizione dell'attività
40	Formazione sulle tecnologie
80	Definizione architettura di riferimento e relativa documentazione
13	<i>Studio principi generali di Flutter e linguaggio Dart</i>
13	<i>Studio delle best practice Flutter</i>
14	<i>Studio delle architetture per i services in Flutter</i>
20	<i>Studio dei widget in Flutter</i>
20	<i>Studio del prototipo di app SportWill oggi esistente</i>
160	Implementazioni
15	<i>Configurazione iniziale</i>
5	<i>Modifica logo</i>
20	<i>Modifica dell'interfaccia grafica nella visualizzazione 'Elenco Uscite' con filtri di ricerca</i>
10	<i>Modifica funzionalità 'Modifica Uscita' con vari fix e cambio colori</i>
30	<i>Implementazione della funzionalità 'Visualizza mappa percorso'</i>
10	<i>Implementazione della funzionalità che permette di vedere la mappa a schermo intero</i>
30	<i>Implementazione della funzionalità che permette l'aggiornamento automatico della mappa.</i>
40	<i>Modifica dell'interfaccia grafica nella visualizzazione 'Elenco Uscite' con filtri di ricerca avanzati.</i>
40	Termine integrazioni e collaudo finale.
Totale ore: 320	

Tabella 2.2: Tabella riassuntiva della pianificazione di stage con variazioni

2.4 Strumenti di comunicazione per lo svolgimento del lavoro

Per ottimizzare lo svolgimento del lavoro sono stati usati i seguenti strumenti di comunicazione:

- **Trello** che mi ha permesso di organizzare e di gestire il progetto tramite la creazione di una bacheca condivisibile;

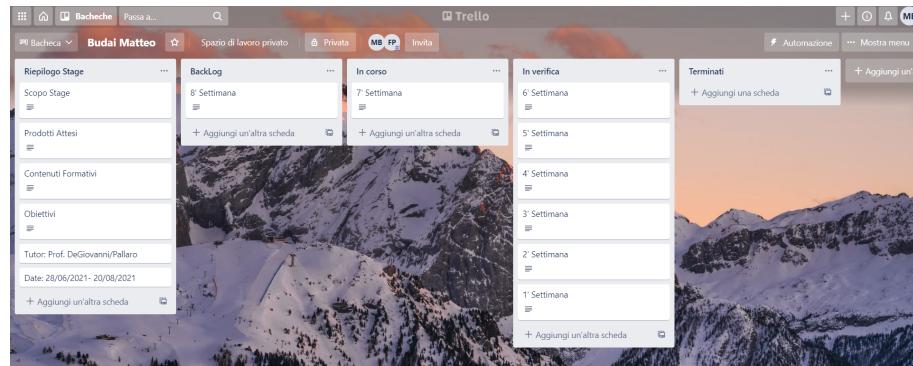


Figura 2.1: Trello

- **Google sheets** in cui venivano riportate ogni giorno le attività che si andavano a svolgere. Veniva segnalato con una spunta se l'attività era corretta oppure nella colonna aggiustamenti veniva spiegato in che modo o cosa svolgere;

A	B	C	D	E	F
1	DATA				
2	28/06/2021 Ripasso Java del corso "altri paradigmi di programmazione"	✓			
3	29/06/2021 Fine ripasso Java	✓			
4	30/06/2021 Ripasso Concetti Web(JSON, servlet e REST)	✓			
5	01/07/2021 Corso flutter Udemy (basati Dart)	✓			
6	02/07/2021 Corso flutter Udemy (basati Dart)	✓			
7	05/07/2021 Corso flutter Udemy(best Practice e architettura)	✓			
8	06/07/2021 Corso flutter Udemy(best Practice e architettura)	✓			
9	07/07/2021 Corso flutter Udemy	✓			
10	08/07/2021 Corso flutter Udemy	✓			
11	09/07/2021 Corso flutter Udemy	✓			
12	12/07/2021 Corso flutter Udemy	✓			
13	13/07/2021 Corso flutter Udemy	✓			
14	14/07/2021 Corso flutter Udemy e studio codice esistente	✓	✓		
15	15/07/2021 Studio codice esistente	✓			
16	16/07/2021 Studio codice esistente	✓			
17	17/07/2021 Studio implementazione per filter	✓			
18	20/07/2021 Ricontrollo filter, cambiato logo app, modifica interfaccia modifica con cambio pulsante e sistemazione problema minuti	✓	✓		
19	21/07/2021 Inizio studio per inserimento mappa	✓			
20	22/07/2021 Funzionalità mappa	✓			
21	23/07/2021 Funzionalità mappa	✓			
22	26/07/2021 Funzionalità mappa con leaflet	✓			
23	27/07/2021 Funzionalità mappa	✓			
24	28/07/2021 Funzionalità mappa	✓			
25	29/07/2021 Funzionalità mappa	✓			

Figura 2.2: Google sheets

- **Discord** che mi ha permesso di interracciarmi direttamente con il tutor aziendale Fabio Pallaro sfruttando più canali di comunicazione vocali e testuali, divisi per argomenti.

Capitolo 3

Framework Flutter e linguaggio Dart

In questo capitolo viene inizialmente introdotto come un'applicazione mobile può essere sviluppata per poi concentrarsi su Flutter e tutte le sue componenti. Infine saranno mostrate come esempio alcune piccole applicazioni realizzate per imparare a utilizzare il framework Flutter.

3.1 Sviluppo applicazioni mobile

Nel quotidiano, non solo in Italia ma in tutto il mondo, l'uso dello smartphone è in costante aumento. Mentre fino a qualche anno fa il cellulare veniva usato solo per telefonare o mandare qualche messaggio, oggi lo smartphone viene utilizzato in qualsiasi ambito: lavoro, comunicare, divertirsi, video, musica o svago. Ormai nei cellulari sono presenti applicazioni per qualsiasi esigenza ed è proprio per questo che chi sviluppa applicazioni ha dovuto considerare lo sviluppo per mobile come fattore primario.



Figura 3.1: Sviluppo applicazioni mobile

Esistono quattro diversi approcci di implementazione:

- App native;
- Web app;
- App Ibride Web View Wrapper;
- App Ibride Compile to Native.

3.1.1 App native

Il metodo nativo dà la possibilità all'applicazione di integrarsi con la parte hardware del dispositivo, sfruttando così tutte le funzionalità del sistema operativo. Le app native vengono realizzate utilizzando gli strumenti di sviluppo software e la documentazione fornita dai produttori del sistema operativo per il quale si ha l'intenzione di sviluppare. Questo metodo è scelto soprattutto degli sviluppatori attenti alle prestazioni e alle performance dell'applicazione. I vantaggi principali di sviluppare App native sono:

- Maggiore velocità, affidabilità e reattività;
- Accesso diretto alla parte hardware e al software installato nel device;
- Notifiche dirette;
- Funzionamento offline.

Attualmente i sistemi operativi più utilizzati sono:

- Android;
- iOS;
- Windows Phone.

Android



Figura 3.2: Logo Android

Android è il sistema operativo più utilizzato e diffuso. È stato sviluppato da Google ed è stato scelto da multinazionali importanti come Samsung, Huawei e Amazon per il funzionamento dei loro dispositivi. Il linguaggio per sviluppare un'applicazione Android è Java. Negli ultimi anni è nato anche Kotlin che è un altro linguaggio ufficiale per la progettazione di applicazioni Android che è più moderno, meno complesso ma performante e compatibile con l'ambiente Android quanto Java.

iOS**Figura 3.3:** Logo iOS

iOS è il sistema operativo sviluppato da Apple per dispositivi iPhone, iPod touch e iPad. Per un lungo periodo il linguaggio per sviluppare un'applicazione iOS è stato Objective-C che deriva da C e C++. Per aumentare la produttività Apple ha lanciato un linguaggio di più alto livello ovvero Swift. Swift è veloce, più leggibile e meno prolioso. Nonostante ciò, Objective-C viene ancora preferito quando si sta lavorando più a basso livello.

Windows Phone**Figura 3.4:** Logo Windows Phone

Windows Phone è il sistema operativo sviluppato da Microsoft. Il linguaggio utilizzato per sviluppare un'applicazione Windows Phone è C# che è un linguaggio semi-compilato orientato agli oggetti. Con Android e iOS in ambito mobile non c'è paragone. Invece per quanto riguarda sistemi desktop Windows risulta una delle migliori.

3.1.2 Web app

Una web app è un'applicazione che funziona come un sito web adattandosi al dispositivo utilizzato. Queste applicazioni non necessitano di essere installate sugli smartphone e quindi non andranno ad aumentare la memoria utilizzata nel dispositivo. Inoltre non possono essere nemmeno pubblicate sugli Store e quindi non godono di questa enorme visibilità. I principali framework e librerie per creare una web app sono:

- Angular;
- PolymerJS;
- React.

I vantaggi di sviluppare una web app sono:

- Scritte con Markup HTML;
- non essendo pubblicate sul Market non devono essere sottoposte al processo di approvazione;
- Minor tempo di sviluppo.

3.1.3 App Ibride Web View Wrapper

Questo tipo di metodo permette di creare applicazioni senza alcuna conversione del codice in base al sistema operativo. In pratica l'applicazione rileva inizialmente il sistema operativo utilizzato e successivamente imita l'aspetto dell'interfaccia utente utilizzando CSS, Sass...

Le piattaforme più usate sono:

- Ionic;
- Apache Cordova;
- PhoneGap.

I vantaggi principali di Ionic e delle App Ibride Web View Wrapper sono:

- Riutilizzo facile del codice;
- Ionic utilizza JavaScript e fornisce un supporto per Angular;
- Addatamento automatico in base alla piattaforma;

Lo svantaggio principale di questo tipo di applicazioni sta in termini di velocità di calcolo e quindi avranno prestazioni inferiori a quelle compilate in nativo.

3.1.4 App Ibride Compile to Native

Le applicazioni ibride che compilano in nativo utilizzano un unico linguaggio di programmazione per la scrittura del codice. Una volta compilato i componenti dell'interfaccia utente del codice vengono convertiti nei componenti dell'interfaccia utente nativi senza bisogno di configurazioni particolari.

Le principali piattaforme utilizzate per compilare in nativo sono:

- React Native;
- NativeScript;
- Xamarin;
- Flutter.

I vantaggi principali di utilizzare piattaforme che compilano in nativo sono:

- Anche se minore delle App Ibride Web View Wrapper hanno un elevata riutilizzabilità del codice;
- Elevato numero di librerie utilizzabili;
- Compilando in nativo offrono prestazioni elevate.

3.2 Flutter



Figura 3.5: Logo Flutter

Flutter è un framework nato abbastanza di recente per lo sviluppo di applicazioni per diverse piattaforme.

È stato ideato da Google come progetto open source e pubblicato ufficialmente per la prima volta a dicembre del 2018 nella versione 1.0 all'evento Flutter Live.

Il 3 marzo 2021 è stata rilasciata la versione 2.0 che consente agli sviluppatori di generare in maniera stabile applicazioni multipiattaforma.

Flutter offre una vasta serie di librerie di elementi di interfaccia utente e combina la facilità di sviluppo con prestazioni simili alle prestazioni native, mantenendo una corretta distinzione visiva tra le diverse piattaforme senza che il programmatore debba prestare particolari attenzioni. Viene utilizzato soprattutto per applicazione Android e iOS e funziona come una vera applicazione nativa.

Utilizzando lo stesso codebase è possibile creare l'applicazione per diverse piattaforme. Il linguaggio di programmazione di Flutter è Dart, sviluppato anche esso da Google, ed è stato pensato per sostituire JavaScript.

Flutter è completamente gratuito e come possiamo vedere dalla figura sottostante, già nell'ultimo anno la sua popolarità e utilizzo è cresciuta notevolmente.

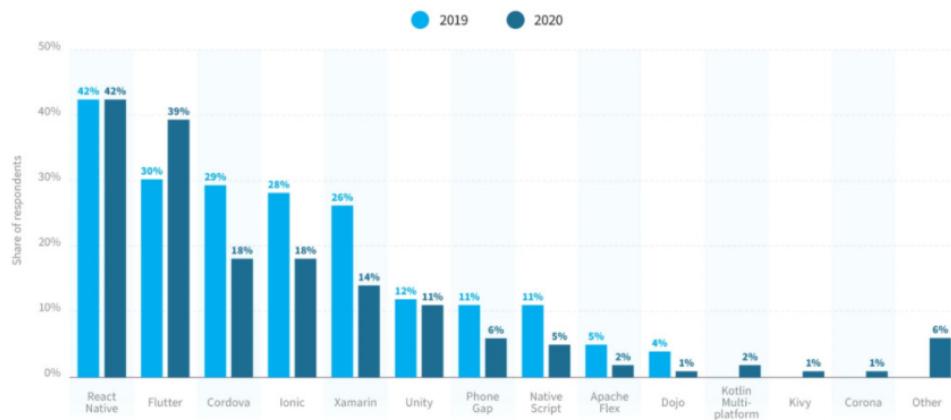


Figura 3.6: Statistiche dei programmatori che usano Flutter per sviluppare app mobile

Anche se molto recente, nel Google Play Store possiamo contare oltre 50000 applicazioni Flutter.

I vantaggi di utilizzare Flutter sono:

- Un codebase per tutte le piattaforme;
- Utilizzo di [Dart](#) che è un linguaggio facile da apprendere;
- Più facile da sviluppare applicazioni con Flutter che quindi entrano sul mercato più in fretta;
- Si basa sul principio 'Tutto è un Widget' che sarà spiegato nella [sezione Widget](#);
- Basso consumo di risorse;
- Esecuzione performante delle app native su smartphone;
- Ottima interfaccia utente che può essere anche personalizzata;
- L'hot reload permette di vedere le modifiche in tempo reale accelerando lo sviluppo.

Invece tra gli svantaggi possiamo citare che:

- Dart è un linguaggio nuovo e quindi non molto diffuso;
- Essendo nuovo mancano librerie di terze parti che facilitano lo sviluppo;
- Vengono create app di grandi dimensioni rispetto agli altri framework o anche a Java stesso.

3.2.1 Dart

Dart è un linguaggio di programmazione sviluppato da Google e presentato per la prima volta il 10 ottobre del 2011 alla conferenza 'GOTO Aarhus 2011'.

Lo scopo principale è quello di sostituire JavaScript per lo sviluppo delle applicazioni. Dart costituisce la base di Flutter e inoltre supporta molte attività di sviluppo di base come la formattazione, l'analisi, il test del codice...

Dart è type-safe. Inoltre i valori in Dart non possono essere null tranne nei casi in cui viene indicato che questi valori possono esserlo, così da evitare possibili errori nel codice.

Dart ha un vasto numero di librerie di base e di pacchetti per le API aggiuntive.

Dart permette di scrivere programmi attraverso due distinte piattaforme:

- Dart Native: per le applicazioni sviluppate per dispositivi mobili e desktop. Include sia una macchina virtuale Dart con compilazione JIT (just-in-time), ovvero viene compilato durante l'esecuzione del programma, sia un compilatore AOT (Ahead-of-Time) per la produzione di codice macchina, ovvero viene compilato prima dell'esecuzione, tipicamente durante l'installazione del programma così migliorando le prestazioni evitando la fase di compilazione durante l'esecuzione del programma;

- Dart Web: per le applicazioni destinate al Web. Include sia un compilatore del tempo di sviluppo (dartdevc) che consente di eseguire il debug dell'applicazione nel browser Chrome e vedere le modifiche quasi immediatamente, che un compilatore del tempo di produzione (dart2js) che fornisce suggerimenti per migliorare il codice Dart e rimuovere il codice inutilizzato. Entrambi i compilatori traducono Dart in JavaScript.

3.2.2 Componenti

Flutter è formato da un'architettura a strati e i suoi tre strati fondamentali sono:

- Framework;
- Engine;
- Embedder.

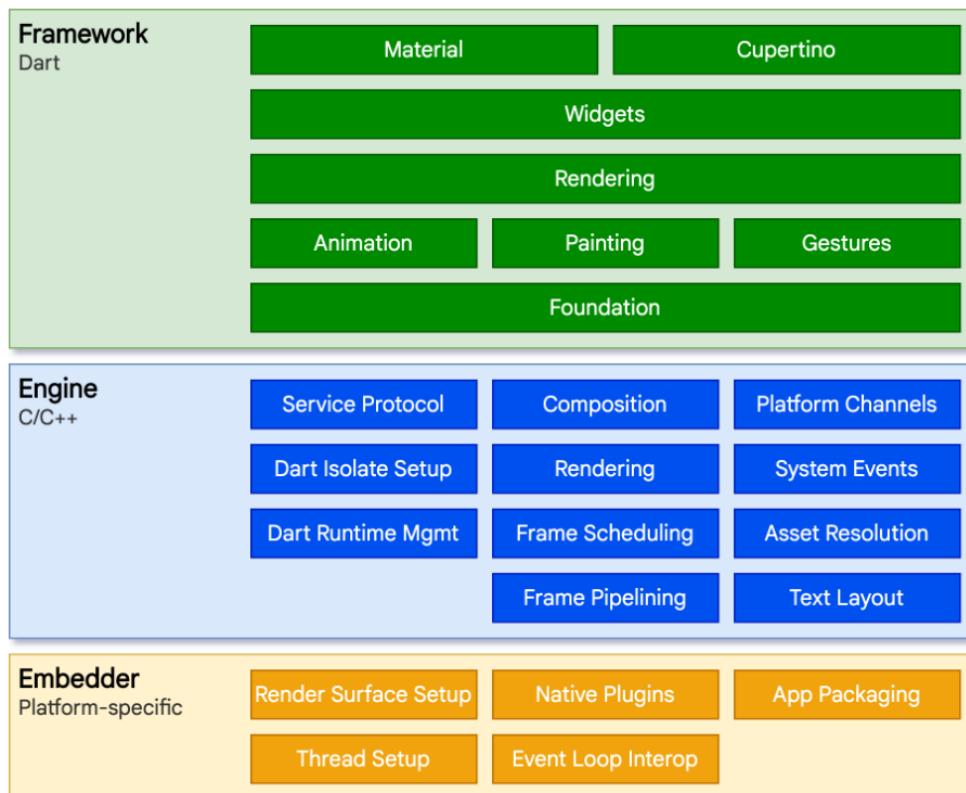


Figura 3.7: Architettura a strati di Flutter

Embedder

Embedder è il livello più basso dell'architettura.

Il linguaggio utilizzato è definito in base alla piattaforma e attualmente può essere: Java e C++ per Android, Objective-C/Objective-C++ per iOS e macOS e C++ per Windows e Linux. Ha lo scopo di legare il rendering della schermata nativa, la gestione degli eventi,...

Per fare ciò lo strato Embedder interagisce con lo strato Engine tramite delle API C/C++. Inoltre lo strato Embedder è composta da una Shell che ospita anche la Dart VM.

Ogni Shell è specifica per ogni piattaforma e offre un accesso alle API native della piattaforma in questione.

Engine

Engine è lo strato intermedio dell'architettura.

Il linguaggio utilizzato è principalmente il C++ e il C per rendere più veloci ed efficienti le applicazioni realizzate in Flutter.

Contiene componenti di basso livello essenziali per il funzionamento del framework. All'interno troviamo il motore grafico *Skia*, una libreria grafica 2D open source scritta in C++ creata da Google, e le shell a cui è possibile accedervi tramite le API esposte dalla libreria *dart:ui*.

Framework

È lo strato principale dell'architettura.

Il linguaggio utilizzato è [Dart](#).

All'interno sono presenti classi fondamentali di base e servizi di base come l'animazione.

Inoltre è presente il Rendering che permette di gestire il layout attraverso un albero di oggetti che viene visualizzato a schermo e che si aggiorna automaticamente.

Sempre all'interno di questo strato sono presenti i Widgets e le due librerie: Material e Cupertino.

3.2.3 Librerie Material e Cupertino

Le due principali librerie di Flutter sono Material e Cupertino.

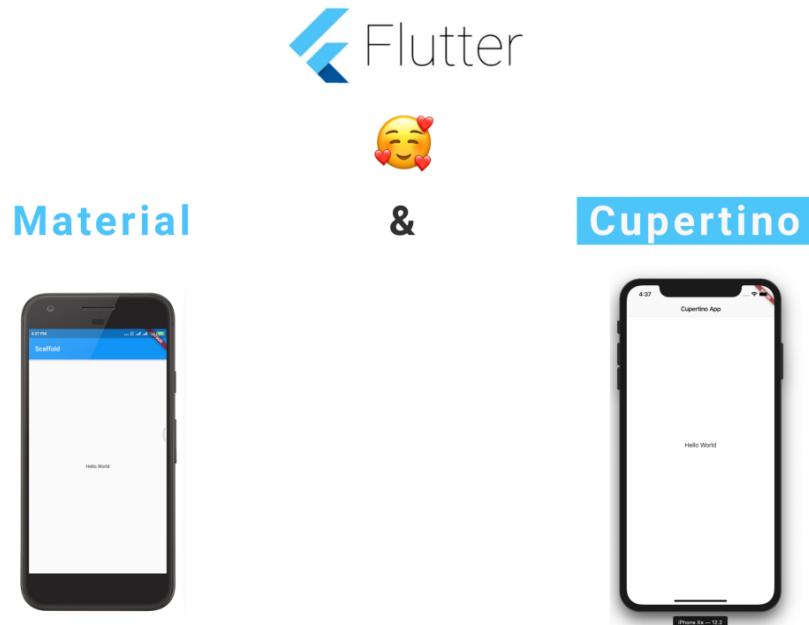


Figura 3.8: Librerie Material e Cupertino

All'interno di Flutter è possibile capire su che piattaforma si sta eseguendo l'applicazione attraverso '*Platform.isIOS*'. Se ritornerà *true* significa che saremo su iOS altrimenti saremo su Android.

La libreria Cupertino permette di implementare un design specifico per le applicazioni iOS.

A differenza della Cupertino la libreria Material permette di implementare un design specifico per le applicazioni Android.

3.2.4 Widget

Flutter è basato sul principio '*Tutto è un widget*' in quanto l'interfaccia di un programma è composta da diversi widget nidificati.

Ogni widget può essere testo o un pulsante o qualsiasi altro elemento grafico che contiene varie caratteristiche. Tutti questi widget possono influenzare altri widget nella costruzione dell'applicazione.

Il vantaggio principale di questa struttura a widget consiste nella flessibilità, invece lo svantaggio di adottare questa strategia sta nel fatto che tutti i widget sono situati nel codice sorgente del programma e pertanto risulteranno fortemente nidificati.

Il framework contiene due classi principali di widget:

- Stateless widget;
- Stateful widget.

Everything is a Widget

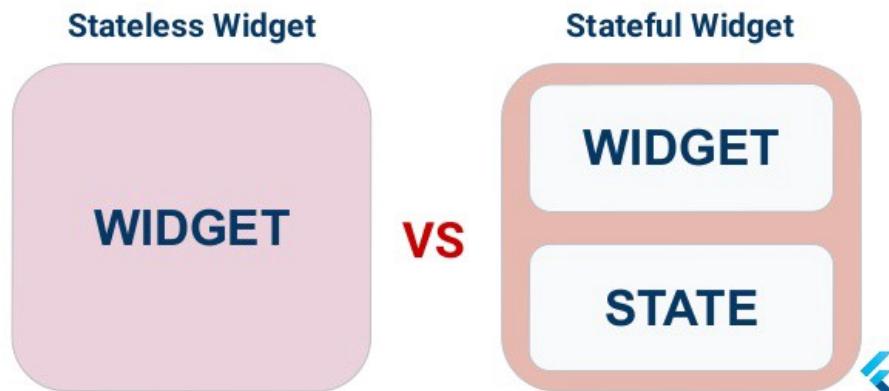


Figura 3.9: Stateless widget e Stateful widget

Stateless widget

Gli stateless widget non hanno uno stato mutabile e quindi non cambieranno nel tempo neanche in seguito a comportamenti effettuati dall'utente.

Alcuni esempi di questi widget sono: Text, Row, Column e Container.

Per creare uno stateless widger bisogna estendere la classe *StatelessWidget* che richiede l'override del metodo *build()*.

Questo metodo viene invocato la prima volta per costruire l'albero dei widget e quando le loro dipendenze cambiano.

Si puo usare uno stateless widget solamente quando i campi dei widget non cambiano nel tempo neanche dopo azioni dell'utente. Negli altri casi sarà da utilizzare uno stateful widget.

Stateful widget

Gli stateful widget sono dinamici e mutano nel tempio in base all'interazione dell'utente o in base ad altri fattori.

Alcuni esempi di questi widget sono: Iamge, Form e Checkbox.

Per creare uno stateful widger bisogna estendere la classe *StatefulWidget*.

Dipende dallo stato dell'oggetto. Infatti ogni volta che si vuole modificare lo stato di un qualsiasi oggetto bisogna chiamare il metodo *setState()* segnalando così al framework di aggiornare l'interfaccia utente chiamando il metodo build e tenendo in considerazione i nuovi stati degli oggetti segnalati nel metodo appena descritto.

Possiamo quindi creare uno stateful widget quando il widget potrebbe cambiare durante il suo ciclo di vita.

Principali Widget

Di seguito saranno spiegati brevemente i widget più comuni e utilizzati in Flutter.

Scaffold Scaffold implementa la struttura del layout visivo.

Contiene principalmente 5 elementi:

- **appBar** che viene descritta di seguito;
- **body** che rappresenta il corpo situato sotto l'Appbar;
- **floatingActionButton** che è un bottone che è situato di default in basso a destra;
- **drawer** che è un menù laterale visibile dall'utente scorrendo da sinistra a destra o viceversa. All'interno di questo menù sono presenti altri widget che permettono di effettuare diverse azioni;
- **bottomNavigationBar** che permette di gestire un menù nella parte inferiore dell'applicazione.



Figura 3.10: Scaffold widget

AppBar L'AppBar viene gestita da Scaffold ed è solitamente situata nella parte superiore dell'applicazione e espone solitamente anche altri widget che permettono di eseguire una o più azioni.

AppBar ha diverse elementi grafici come elevazione e titolo.

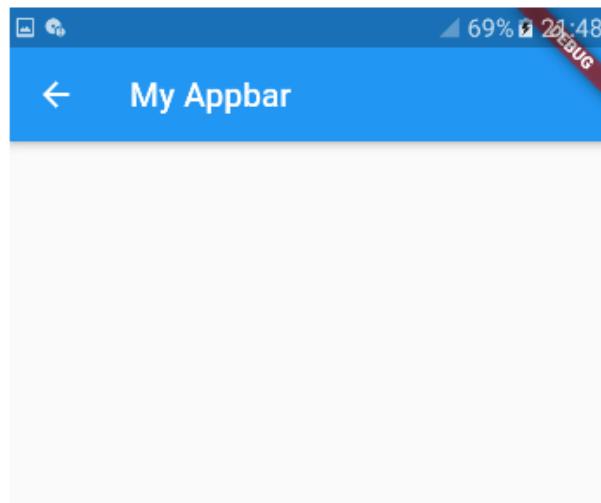


Figura 3.11: Appbar widget

Row, Column Il widget Row permette di creare una famiglia di widget in una riga orizzontale invece il widget Column permette di creare sempre una famiglia di widget ma in verticale.

Entrambi non sono scorrevoli quindi nel caso si abbia bisogno di una famiglia di widget scorrevoli bisogna utilizzare il widget *ListView*.

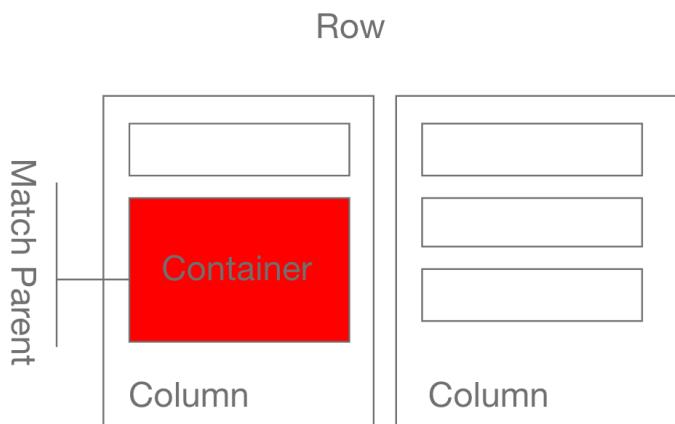


Figura 3.12: Row e Column widgets

Container Il widget Container funge da contenitore all'interno dell'applicazione e può racchiudere altri widget.

Ha un margine, un bordo e un padding di default che possono essere modificati.

Oltre a questi elementi questo widget contiene altri elementi che permettono di decorare la sua area di competenza.

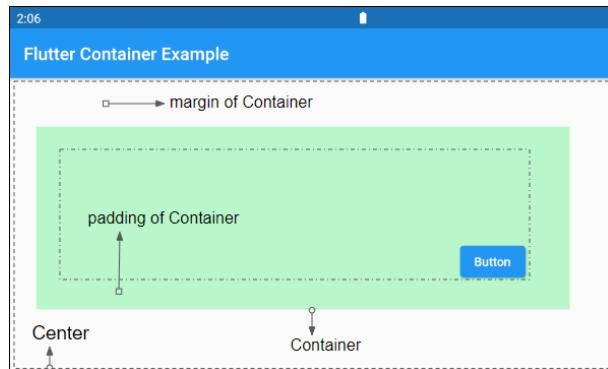


Figura 3.13: Container widget

Text Il widget Text consente di creare una porzione di testo all'interno dell'applicazione che può essere decorata in base alle diverse esigenze.

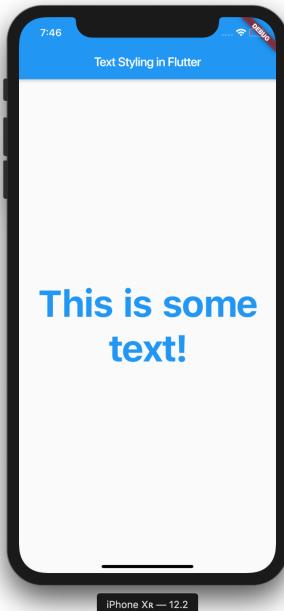


Figura 3.14: Text widget

Image Consente di apportare all'interno dell'applicazione un'immagine. Ci sono vari formati disponibili ovvero JPEG, PNG, GIF.

- **new Image** che permette di ottenere un’immagine da un ImageProvider(utilizza imageCache globale per memorizzare nella cache le immagini);
 - **new Image.asset** che permette di ottenere un’immagine da un AssetBundle(una raccolta di risorse utilizzate dall’applicazione) utilizzando una chiave;
 - **new Image.network** che permette di ottenere un’immagine da un URL;
 - **new Image.file** che permette di ottenere un’immagine da un File;
 - **new Image.memory** che permette di ottenere un’immagine da un Uint8List.

Icon Il widget Icon permette di inserire icone all'interno dell'applicazione. Le icone sono quadrate e non sono interattive.

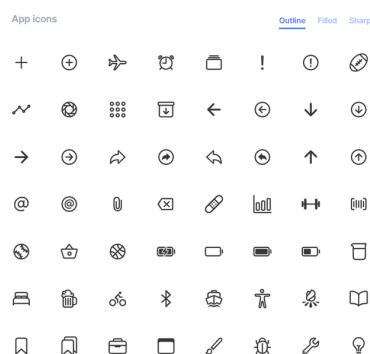


Figura 3.15: Icon widget

ElevatedButton Il widget ElevatedButton permette di inserire un bottone con uno stile di default all'interno dell'applicazione.

Contiene l'elemento `onPressed` che permette di eseguire operazioni quando il bottone sarà premuto. Nel caso `onPressed` sia uguale a `null` il bottone sarà disabilitato.

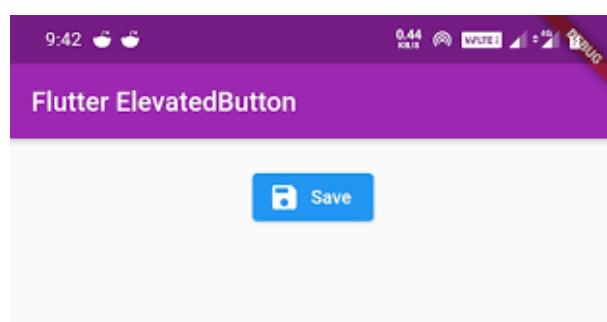


Figura 3.16: ElevatedButton widget

3.2.5 Esempi piccole applicazioni realizzate

Di seguito verranno mostrate alcune applicazioni realizzate seguendo un corso su Udemy per imparare a usare il framework Flutter.

Applicazione base

Dopo aver effettuato la giusta configurazione e scaricato tutto il necessario, per creare la prima applicazione di base messa a disposizione da Flutter basterà digitare sul terminale il comando: `flutter create "nome_applicazione"`.

Questo ci permette di creare la nostra prima applicazione base come esposto nella figura sottostante.

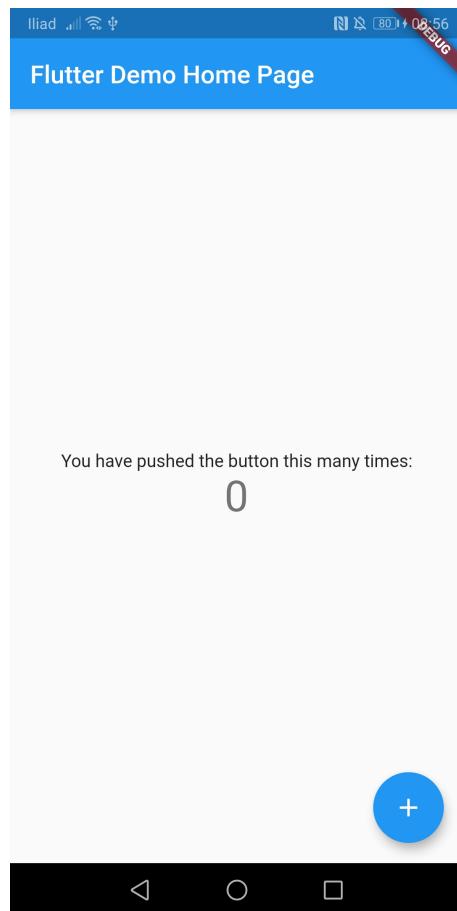


Figura 3.17: Applicazione base

Applicazione 1

Questa applicazione permette di rispondere a una serie di domande e successivamente in base a un punteggio assegnato alla risposta selezionata da un giudizio finale.

Una volta terminato, il quiz può essere ritentato.

In particolare la creazione di questa applicazione mi ha permesso di apprendere meglio il funzionamento di alcuni widget fondamentali.

Tra questi sono stati visti principalmente i seguenti: AppBar, Text, Container, ElevatedButton.

Inoltre mi ha permesso anche di capire meglio gli attributi dei widget precedentemente elencati e mi ha fatto comprendere il funzionamento dello `setState()`.

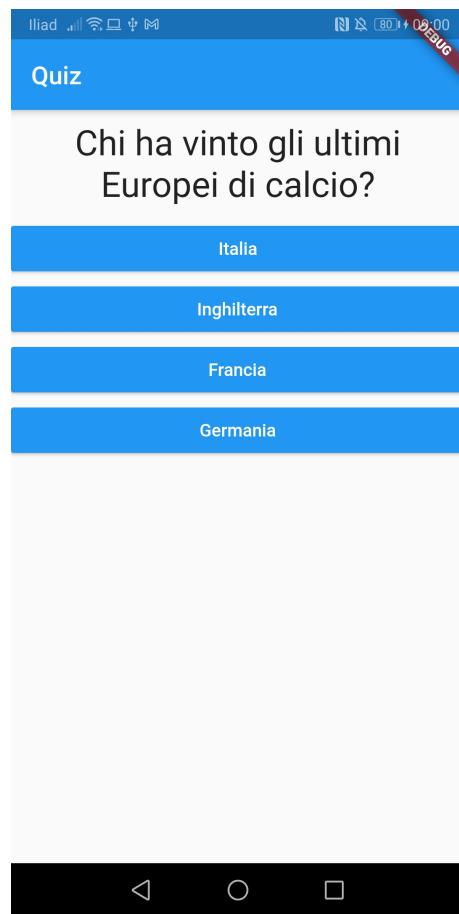


Figura 3.18: Applicazione 1

Applicazione 2

Questa applicazione permette di inserire delle transizioni che un utente ha effettuato così da monitorare le spese sostenute.

In particolare mi ha permesso di comprendere alcuni attributi e widget più complessi rispetto a quelli base, studiare per bene il layout in modo che si adattasse alle dimensioni del cellulare, configurare una pagina in modo scrollabile e gestire widget a comparsa e scomparsa.

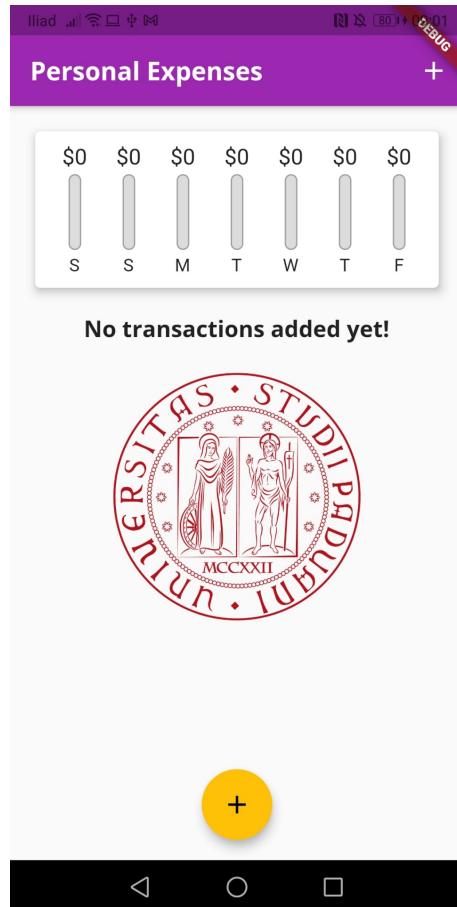


Figura 3.19: Applicazione 2

Applicazione 3

Quest'ultima applicazione permette di gestire diverse categorie di cibi i quali contengono diverse ricette che possono essere aggiunte ai preferiti.

In particolare mi ha permesso di capire il corretto funzionamento delle Card, delle liste e come gestire nel menu laterale(drawer) i preferiti e i filtri.



Figura 3.20: Applicazione 3

Capitolo 4

Sportwill

In questo capitolo viene descritto il progetto, effettuata un'analisi iniziale dei requisiti per poi definire dettagliatamente come le funzionalità sono state implementate e che tecnologie e strumenti sono stati usati.

4.1 Descrizione progetto

Sportwill è un'applicazione che permette di gestire gli eventi sportivi. Dopo essersi autenticati al sistema è possibile visualizzare tutte le attività presenti, passate e future. È possibile cercare queste attività utilizzando i vari filtri presenti. Ogni utente può gestire i propri eventi creando nuove attività e modificarle successivamente. Una volta che un utente vuole iniziare un evento dovrà andare sulle specifiche dell'attività e premere sull'apposito bottone per farla iniziare e successivamente la medesima potrà essere messa anche in pausa. Una volta iniziata l'attività, verranno presi i dati relativi alla posizione così da poter far vedere agli altri utenti in una mappa la posizione in tempo reale. Grazie a questa mappa l'utente premendo sull'apposito bottone segnalerà a chi ha fatto partire l'evento che vuole partecipare anche lui e potrà raggiungerlo.

4.2 Tecnologie e strumenti

All'interno del progetto i file sono suddivisi nel varie sottocartelle in base al loro compito.

Oltre al file main.dart che è il file principale sono presenti altre 4 cartelle che permettono di organizzare meglio il lavoro:

- **models**: che permette di gestire le eccezioni;
- **providers**: che permette di gestire il modello dei dati e fare le varie chiamate al backend;
- **screens**: che crea tutte le pagine dell'applicazione con gli appositi widgets;
- **widgets**: che offrono supporto agli screens tramite la creazione di widget.

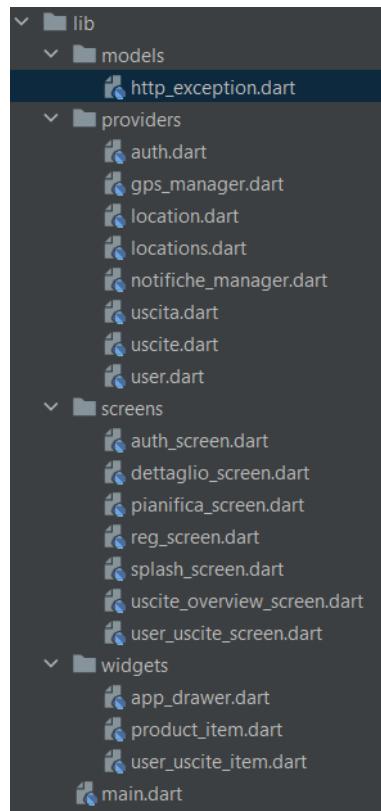


Figura 4.1: Struttura progetto



Figura 4.2: Logo Android Studio



Figura 4.3: Logo GitLab



Figura 4.4: Logo DBeaver

- 4.2.1 Flutter e Dart**
- 4.2.2 Android Studio**
- 4.2.3 GitLab**
- 4.2.4 Database**
- 4.2.5 Backend**

Spring boot e Spring jpa



Figura 4.5: Logo Spring

4.3 Analisi dei requisiti

Tipologie di utenti: utente autenticato(come ospite o normale), utente non autenticato,
Casi d'uso:

UC1 - Visualizzazione mappa

- **Attori Primari:** .
- **Descrizione:** .
- **Scenario principale:** .
- **Precondizione:** .
- **Postcondizione:** .

UC2 - Visualizzazione mappa schermo intero

- **Attori Primari:** .
- **Descrizione:** .
- **Scenario principale:** .
- **Precondizione:** .
- **Postcondizione:** .

UC3 - Filtro base

- **Attori Primari:** .
- **Descrizione:** .
- **Scenario principale:** .
- **Precondizione:** .
- **Postcondizione:** .

UC4 - Filtro avanzato

- **Attori Primari:** .
- **Descrizione:** .
- **Scenario principale:** .
- **Precondizione:** .
- **Postcondizione:** .

4.4 Implementazioni

Le implementazioni realizzate durante lo stage sono state:

- Logo;
- Filtro di ricerca testo;
- Pagina Modifica e campi obbligatori;
- Mappa percorso;
- Aggiornamento automatico mappa;
- Mappa schermo intero;
- Colori;
- Pagina Modifica;
- Eliminazione, Modifica e Aggiunta di un'attività;
- Filtro avanzato di ricerca attività.

Tutti i servizi sono stati rilasciati su un'apposita repository aziendale su *GitLab*.

4.4.1 Logo

È stato modificato il logo di base di Flutter con quello di Sportwill.
Per fare ciò è stato inserita l'immagine del logo all'interno della seguente cartella *app/src/main/res/mipmap/*.



Figura 4.6: Logo Sportwill

4.4.2 Filtro di ricerca testo

All'interno della pagina principale dell'applicazione ovvero la pagina `uscite_overview_screen.dart` è stato aggiunto un widget che permette di ricercare le varie card in base all'evento, al luogo o alla persona.

Digitando nella barra di ricerca quindi si potranno ricercare solo le card a cui si è interessati.

Per fare ciò è stato inserita una funzione chiamata `ricerca()` che ritorna una `TextField` come Widget nel file `uscite_overview_screen.dart`.



Figura 4.7: Filtro di ricerca testo

4.4.3 Campi obbligatori e pagina modifica e crea

All'interno di questa pagina sono stati tolti tutti i campi obbligatori non necessari. Per fare ciò sono stati tolti tutti i validator non necessari nel file *pianifica_screen.dart* e lasciato gli altri validator con le apposite funzioni.

Inoltre all'interno di questa pagina sono stati aggiustati i campi degli orari che nella funzione *selezionaOra()* tornava sempre i minuti attuali.

Per rendere all'utente più comprensibile il salvataggio è stato sostituita l'icona di salvataggio con l'icona *Icon(Icons.save)*.



Figura 4.8: Campi obbligatori

4.4.4 Colori

All'interno dei vari file sono stati cambiati diversi colori in modo che venissero utilizzati principalmente quelli dell'applicazione, ovvero il bianco e l'arancione.

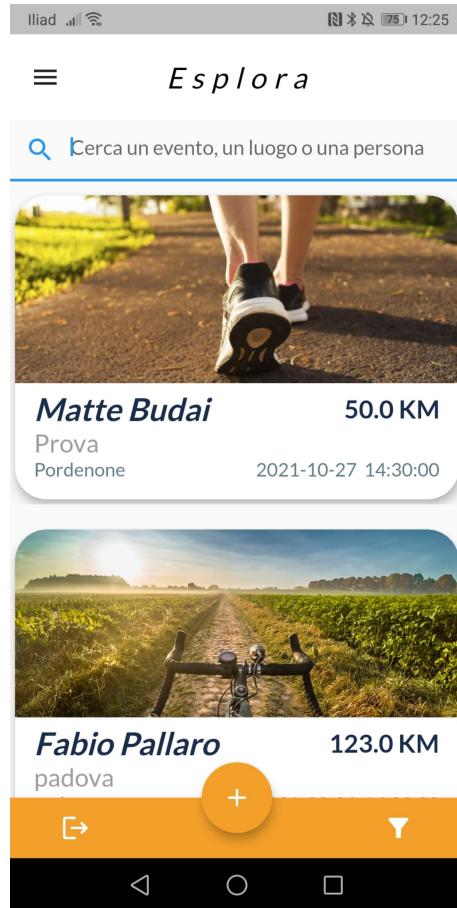


Figura 4.9: Colori

4.4.5 Eliminazione, Modifica e Aggiunta di un'attività

In seguito alle modifiche di vari campi sono state sistemate le pagine di eliminazione, modifica e aggiunta di un'attività con nuovi valori e formati in modo che venissero rispettate le richieste del backend.

In particolar modo sono stati sistemati i valori che riguardavano la data e l'orario con i nuovi formati richiesti e la lunghezza e il numero di partecipanti che richiedevano un double e un intero.

4.4.6 Mappa percorso

Per gestire la mappa sono stati creati due providers:

- location.dart: che è il modello e contiene tutti i dati che riguardano una posizione;
- locations.dart: che permette di chiamare il backend per ottenere i vari valori da visualizzare nella mappa.

La mappa viene creata nel file *dettaglio_screen.dart* e viene mostrata solo se c'è almeno una posizione per l'attività selezionata.

Per visualizzare la mappa viene usato Leaflet e viene creato un array con tutte le posizioni che vengono poi disegnate sulla mappa mettendo i marker alla prima posizione e all'ultima in ordine temporale.

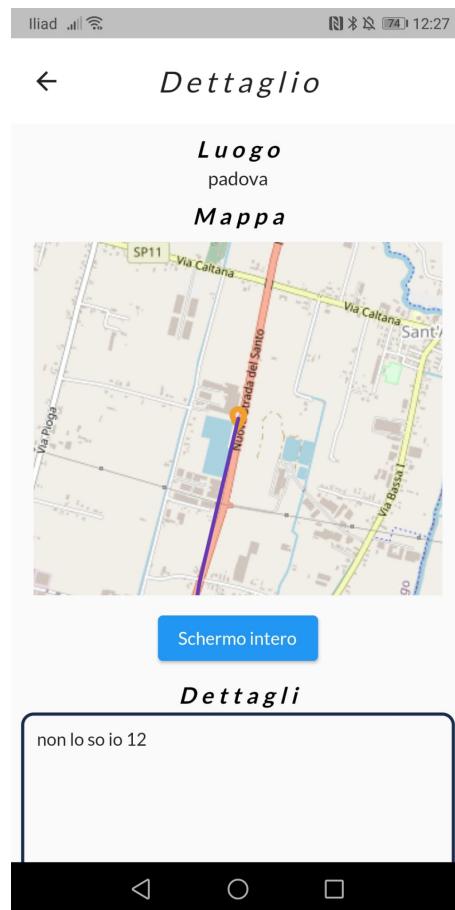


Figura 4.10: Mappa percorso

4.4.7 Aggiornamento automatico mappa

Per fare in modo che la mappa si aggiornasse in modo automatico è stato inserito un Timer che ogni 30 secondi va a chiamare la funzione che ritorna tutte le posizioni di quell'attività.

Per evitare che il Timer venisse ricostruito ogni volta è stato usato un valore booleano che permette la costruzione del Timer solo la prima volta.

```
if (inizio) {
    inizio = false;
    _timer = Timer.periodic(
        Duration(seconds: 30),
        (Timer t) => aggiornaMappa(
            context, _locationProduct, points, _sizeLocation, _productId)); // Timer.periodic
}
```

Figura 4.11: Aggiornamento automatico mappa

4.4.8 Mappa schermo intero

Per fare in modo che la mappa fosse più utilizzabile dall'utente è stata creata la possibilità di vederla a schermo intero. Per fare ciò è stato usato un valore booleano, ovvero *ingradisci* inizialmente uguale a false, che se settato a true permetteva di vedere nello schermo solo la mappa.

Cliccando sul bottone *schermo intero* veniva settato questo valore a true e veniva ricostruita la pagina invece premendo sull'icona con la croce veniva settato il valore a false ricostruendo sempre la pagina.

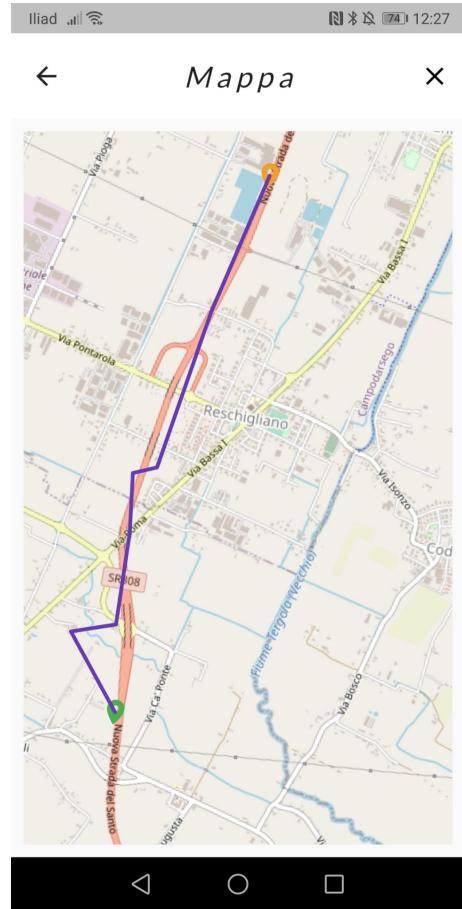


Figura 4.12: Mappa schermo intero

4.4.9 Filtro avanzato di ricerca attività

Per creare il filtro avanzato è stata creata una funzione `showFilterMenu()` nel file `uscite_overview_screen.dart` che contiene una `showDialog`.

Per aprire il menù dei filtri bisognerà premere sull'icona dei filtri situata in basso a destra.

All'interno di questo menu è possibile filtrare le attività in base allo sport, alla data e se si tratta di tue attività.

Per applicare i vari filtri, ovvero quando si preme sull'icona check, è stata utilizzato un booleano che ricostruisce la pagina con filtro uguale a true e va a chiamare nel provider `uscite.dart` la funzione `findBySportData` in base ai valori messi nel filtro.

L'icona con la croce, una volta premuta, setta il valore del filtro a false e ricostruisce la pagina chiudendo la showdialog dei filtri e cancellando i filtri precedentemente applicati. Per poter capire se abbiamo applicato i filtri basterà guardare se l'icona dei filtri ha il colore verde.

Invece se non avremo filtri applicati l'icona avrà il solito colore bianco.

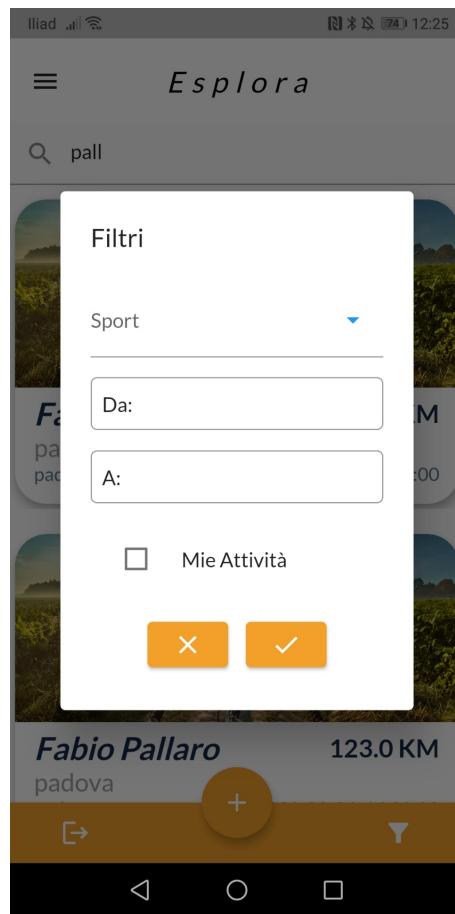


Figura 4.13: Filtro avanzato di ricerca attività

4.4.10 Pubblicazione applicazione Play Store

Capitolo 5

Conclusioni

- 5.1 Consuntivo finale**
- 5.2 Raggiungimento degli obiettivi**
- 5.3 Conoscenze acquisite**
- 5.4 Valutazione personale**

Bibliografia

- <https://www.synclab.it/>
- <https://www.bintmusic.it/uso-smartphone-statistiche-controllo/>
- <https://pan-webdesign.it/pages/sviluppo-applicazioni-mobile-possibili-scenari-metodi-applicativi.html>
- <http://www.mr-apps.com/it/blog/la-differenza-tra-app-native-app-ibride-e-web-app>
- <https://www.icicletech.com/blog/react-native-flutter-ionic-xamarin-nativescript>
- <https://dart.dev/>
- <https://flutter.dev/>
- <https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/cose-flutter/>
- <https://nix-united.com/blog/the-pros-and-cons-of-flutter-in-mobile-application-development/#benefits>
- [https://it.wikipedia.org/wiki/Dart_\(linguaggio\)](https://it.wikipedia.org/wiki/Dart_(linguaggio))
- <https://www.html.it/pag/377313/il-framework-dettagli-tecnici/>
- <https://www.html.it/pag/378347/stateful-e-stateless-widget-le-fondamenta-di-unapp-flutter/>
- <https://www.geeksforgeeks.org/scaffold-class-in-flutter-with-examples/>
- <https://www.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps/>