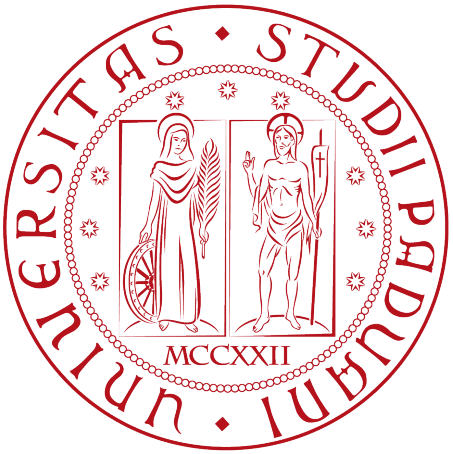
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Refactoring da Qt a Flutter: modernizzazione di un'applicazione ad uso medico

Tesi di laurea triennale

Relatore

Prof. Giovanni Da San Martino

Laureando

Luca Carturan

Anno Accademico 2021-2022

Dedicato a chi mi ha sostenuto in questi anni.

Abstract

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, presso l'azienda Adaptica S.r.l. di Padova.
In esso verranno presentate le conoscenze acquisite durante la formazione nello sviluppo di applicazioni mobile per mezzo del framework Flutter.
Verrà descritto un confronto tra la vecchia e la nuova applicazione e le funzionalità del dispositivo al quale si interfaccia.

"The only real mistake is the one from which we learn nothing."

— Henry Ford

Ringraziamenti

Vorrei esprimere la mia gratitudine al Prof. Giovanni Da San Martino, relatore della mia tesi, per l'affiancamento e l'aiuto durante la stesura di questo lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, la fiducia e per essermi stati vicini in ogni momento durante gli anni di studi.

Infine, un sentito ringraziamento va a all'azienda Adaptica S.r.l. per il grande coinvolgimento e l'accoglienza nello svolgimento del tirocinio, dandomi la possibilità di lavorare in un ambiente lavorativo confortevole.

Padova, Dicembre 2022 Luca Carturan

Indice

Elenco delle figure

1 Introduzione

Lo scopo del progetto di stage consiste nella progettazione e implementazione parziale di un frontend *sviluppato in Flutter/Dart.*
Il progetto comprende una fase iniziale finalizzata allo studio del linguaggio di programmazione, seguita da una fase di ricerca mirata alla comprensione e allo studio dei punti di forza e debolezza del precedente sistema, puntando a migliorarlo in fase di progettazione, utilizzando una serie di best practice *come ad esempio l'applicazione di* design pattern[†] *e principi* SOLID[†] *dove possibile.*

1.1 Organizzazione dell'elaborato

Il primo capitolo espone le norme usate, il contesto aziendale nel quale si è svolta l'attività di stage e un sommario del lavoro.

Il secondo capitolo descrive gli obiettivi e la pianificazione del lavoro.

Il terzo capitolo tratta le tecnologie, gli strumenti, i linguaggi utilizzati nel corso dell'attività di stage.

Il quarto capitolo contiene lo *UML*[†] e use-case relativi al precedente sistema.

Il quinto capitolo contiene la parte di progettazione, *UML* e *use-case* relativa al nuovo sistema.

Il sesto capitolo tratta lo sviluppo del nuovo sistema, diviso in modello e interfaccia utente.

Il settimo capitolo riassume le considerazioni finali relativamente al prodotto implementato e all'attività di stage nel suo complesso.

1.2 Convenzioni adottate

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[†];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

1.3 Contesto aziendale

Figura 1.1: logo di Adaptica S.r.l.

1.3.1 Adaptica

Adaptica, parte del gruppo He Vision, è un'azienda italiana con sede a Padova.
Fondata nel 2009 come spin-off dell'Università di Padova è specializzata in ottica adattiva e optoelettronica applicata all'industria e alla ricerca astronomica. Adaptica ha sfruttato la sua conoscenza tecnologica in astronomia per passare dall'universo all'esplorazione della visione umana.

L'azienda si occupa dello sviluppo di attrezzature elettromedicali nel campo dell'oftalmologia, destinate allo screening di massa e mirate ad accelerare il processo di diagnosi dei difetti oftalmologici.
I prodotti vengono ideati, progettati, sviluppati e prodotti nella sede di Padova per poi entrare in una rete di distribuzione internazionale.

Questo progetto è legato ai prodotti, distribuiti da Adaptica, per l'analisi soggettiva dei pazienti, in particolare ai sistemi VisionFit SC.

1.3.2 VisionFit SC

Figura 1.2: VisionFit SC

VisionFit SC è un innovativo sistema elettronico di lenti adattive, mobile e indossabile. È un correttore refrattivo dell'occhio umano: viene usato per effettuare test soggettivi, in modo da prescrivere occhiali utilizzando due moduli principali:

- Il primo modulo è in grado di correggere la miopia o ipermetropia modificando il potere ottico di defocus di una lente in grado di focalizzare nella gamma di + 20, -20 D manualmente o elettronicamente;

- Il secondo modulo è in grado di effettuare una variabile correzione dell'astigmatismo dell'occhio umano nella gamma di +10 e -10 D.

Il dispositivo medico può essere usato al posto del classico forottero medico in una procedura di facile utilizzo, con una usabilità almeno equivalente, o persino migliore. Non richiede il controllo della posizione del paziente, rendendolo ideale per i pazienti non cooperativi.

Gli occhiali di VisionFit SC sono un sistema di lenti sintonizzabili che fornisce la correzione richiesta dal professionista della cura degli occhi.
Le ottiche sferiche e cilindriche sono controllate in remoto senza alcuna interazione diretta dell'operatore con gli occhiali. Il sistema ottico è dotato di un supporto aggiuntivo per ogni occhio, inteso per inserire qualsiasi lente/filtro ausiliario.

1.3.3 Aquid

Figura 1.3: Aquid

Aquid è un forottero da tavolo meno voluminoso rispetto a quelli disponibili sul mercato. È stato progettato per consentire la rifrazione in qualsiasi ambiente e, grazie al suo braccio trasportabile, può essere fissato a qualsiasi superficie piana accessibile, ovunque ci si trovi.

Utilizza lo stesso sistema di lenti controllate elettronicamente del VisionFit SC. Presenta inoltre funzionalità aggiuntive, come la possibilità di regolare la distanza pupillare sempre da remoto.

1.3.4 2WIN

Figura 1.4: 2WIN

2WIN è un video rifrattometro binoculare mobile e analizzatore di visione che misura entrambi gli occhi contemporaneamente, in condizioni di visione reali.

Il dispositivo incorpora tecnologie per rilevare completamente errori di rifrazione, anomalie oculari e problemi di vista; è in grado di rilevare miopia, ipermetropia, astigmatismo e altri fattori ambliogenici e, inoltre, fornisce prove di anomalie visive che possono essere correlate ad anisometropia, anisocoria, strabismo, forie.

2WIN non è stato trattato durante il periodo di stage per mancanza di tempo ma viene utilizzato nell'applicativo precedente, tramite connessione WiFi o Bluetooth RFCOMM, per ottenere misurazioni oggettive.

1.3.5 Aquid Chart

Figura 1.5: Aquid Chart

Aquid Chart è una tavola optometrica avanzata progettata per funzionare con Aquid. È disponibile in due configurazioni: Aquid Chart 24" e Aquid Chart Mobile - il primo è un monitor LED professionale FULL HD dedicato a tutti gli esperti di oculistica, mentre il secondo è una tabella oculistica portatile per i professionisti della visione che vogliono andare ovunque si trovino i loro pazienti. Inoltre, sono entrambi compatibili con VisionFit SC.

Aquid Chart non è stato trattato durante il periodo di stage per mancanza di tempo, ma viene controllato mediante l'applicativo precedente, tramite connessione Bluetooth RFCOMM.

2 | Piano di progetto

2.1 Analisi dei requisiti

2.1.1 Obiettivi

Questo documento farà riferimento agli obiettivi secondo le seguenti notazioni:

- **«ob»** per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- **«de»** per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- **«op»** per i requisiti opzionali, rappresentanti valore aggiunto non strettamente competitivo.

2.1.2 Elenco degli obiettivi

- **Obbligatori**
  - **ob01:** Apprendimento del funzionamento dispositivi e prodotti aziendali;
  - **ob02:** Apprendimento del funzionamento dei servizi AWS coinvolti nel progetto;
  - **ob03:** Stesura UML precedente modello;
  - **ob04:** Stesura UML e Use Case nuovo modello;
  - **ob05:** Identificazione Design Pattern coinvolti nel nuovo modello;
  - **ob06:** Implementazione Modello e Grafica di Sphinx;
  - **ob07:** Esecuzione di Test TDD per il modello;
  - **ob08:** Implementazione autenticazione mediante AWS;
  - **ob09:** Analisi punti di estendibilità del modello;
  - **ob10:** Documentazione dettagliata relativa a Sphinx.

- **Desiderabili**
  - **de01:** Predisposizione di comandi legati ad AWS Iot Core;
  - **de02:** Predisposizione template per futuri comandi legati ai servizi AWS.

- **Opzionali**
  - **op01:** Configurazione DevOps per CI/CD su Gitlab CE;
  - **op02:** Registrazione dei risultati su sistema qualità aziendale.

2.1.3 Pianificazione del lavoro

Le scadenze settimanali sono state pianificate nel seguente modo:

- lunedì, briefing di programmazione per la settimana;
- al più, ogni due giorni un breve colloquio per vedere i risultati della giornata, e/o discutere di eventuali questioni progettuali, implementative, ecc.;
- venerdì, briefing consuntivo e verifica dell'attività settimanale con stesura di un riassunto del lavoro svolto.

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

Durata in ore	Descrizione dell'attività
60	**Studio del funzionamento strumenti e analisi Use Case**
16	Training strumenti e sistemi aziendali
8	Studio architetture multipiattaforma
12	Studio Test Driven Development
8	Studio predente sistema
8	Stesura UML precedente sistema
8	Stesura Use Case precedente sistema
52	**Progettazione Sphinx**
16	Progettazione UML Sphinx
12	Scrittura Use Case Sphinx
16	Individuazione design pattern
8	Stesura documentazione contenente scelte progettuali
96	**Sviluppo modello Sphinx**
8	Identificazione test modello
24	Implementazione test
36	Implementazione modello
20	Verifica test di integrazione
8	Stesura documentazione e test report
100	**Sviluppo view Sphinx**
12	Test indipendenza modello
16	Identificazione test controller e view
16	Implementazione test
16	Implementazione controller
16	Implementazione view
16	Verifica test di sistema
8	Stesura documentazione finale e test report

2.1.4 Milestone di progetto

Id Milestone	Settimana	Obiettivi	Nome Milestone
M01	Settimana 2	ob01, ob02, ob03	Studio iniziale del progetto
M02	Settimana 3	ob03	Studio del precedente modello
M03	Settimana 5	ob04, 0b05	Stesura Use Case e UML del nuovo modello
M04	Settimana 6	ob06, ob07	Implementazione Sphinx
M05	Settimana 7	ob08, ob09	Test estendibilità
M06	Settimana 8	ob10	Finalizzazione della documentazione

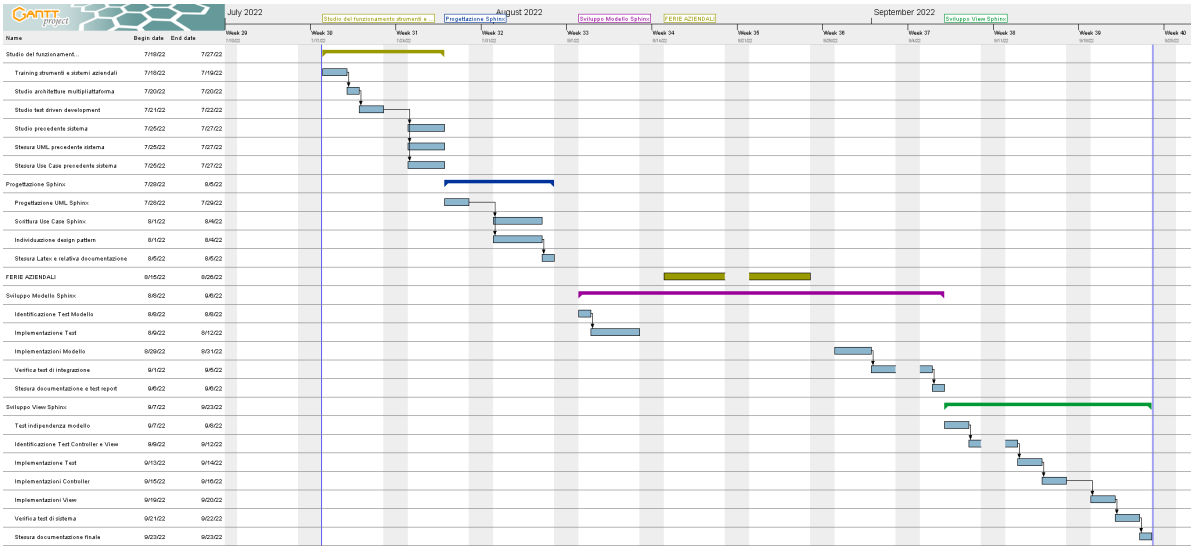


Figura 2.1: Gantt

3 | Tecnologie, strumenti e linguaggi

3.1 Tecnologie

3.1.1 Amazon Web Services

Amazon Web Services (AWS) è una collezione di servizi offerti da Amazon che fornisce *API*[†] e piattaforme di *cloud computing on-demand* a privati, aziende e governi, con pagamento in base al consumo.

I servizi offerti da AWS sono stati visti solo a livello teorico per possibili future implementazioni.

3.1.1.1 AWS Amplify

Figura 3.1: AWS Amplify

AWS Amplify è una soluzione completa che consente agli sviluppatori *front-end* di applicazioni Web e per dispositivi mobili di costruire, distribuire e ospitare facilmente applicazioni *full-stack* in AWS, con la flessibilità data dal poter usufruire dei vari servizi AWS, man mano che i casi d'uso si evolvono. Non sono necessarie competenze cloud.

3.1.1.2 AWS Cognito

Figura 3.2: AWS Cognito

Amazon Cognito permette di aggiungere strumenti di registrazione degli utenti, accesso e controllo degli accessi alle app Web e per dispositivi mobili, in modo rapido e semplice. Rende possibile il ridimensionamento delle risorse per milioni di utenti e supporta l'accesso con provider di identità social quali Apple, Facebook, Google e Amazon e provider di identità aziendali tramite SAML 2.0 e OpenID Connect.

3.1.1.3 AWS API Gateway

Figura 3.3: AWS API Gateway

Amazon API Gateway è un servizio completamente gestito che semplifica per gli sviluppatori la creazione, la pubblicazione, la manutenzione, il monitoraggio e la protezione delle *API* su qualsiasi scala. Le *API* fungono da "porta di entrata" per consentire l'accesso delle applicazioni ai dati, alla logica aziendale o alle funzionalità dai servizi *back-end*. API Gateway consente di creare *API RESTful* e *WebSocket* che rendono possibili applicazioni di comunicazione bidirezionale in tempo reale.

3.1.1.4 AWS S3

Figura 3.4: AWS S3

AWS Simple Storage Service (AWS S3) è un servizio di archiviazione di oggetti che offre scalabilità, disponibilità dei dati, sicurezza e prestazioni all'avanguardia nel settore.

3.1.1.5 AWS Lambda

Figura 3.5: AWS Lambda

AWS Lambda è un servizio di elaborazione *serverless* che permette di eseguire il codice senza effettuare il provisioning o gestire i server, creare una logica di dimensionamento dei cluster in funzione dei carichi di lavoro, mantenere integrazioni degli eventi o gestire i runtime. Con Lambda, si può eseguire codice per qualsiasi tipo di applicazione o servizio di *back-end*, senza alcuna amministrazione.

3.2 Strumenti

3.2.1 Visual Studio Code

Figura 3.6: Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Include il supporto per debugging, un controllo per *Git*† integrato, *syntax highlighting*, *IntelliSense*, *snippet* e *refactoring*† del codice.

3.2.2 Android Studio

Figura 3.7: Android Studio

Android Studio è l'ambiente di sviluppo integrato ufficiale per il sistema operativo Android di Google.

3.2.3 Figma

Figura 3.8: Figma

Figma è uno strumento per la progettazione di interfacce, che si rivolge principalmente ai web designer che hanno bisogno di un software studiato appositamente per realizzare

il design di siti web e applicazioni.

Può essere utilizzato per fare *UI*[†], *UX*[†], prototipazione, graphic design ma anche illustrazione vettoriale.

3.2.4 PlantUML

Figura 3.9: PlantUML

PlantUML è uno strumento *open-source* che consente agli utenti di creare diagrammi da un linguaggio di testo semplice.
Oltre a vari diagrammi *UML*, PlantUML supporta molti altri formati relativi allo sviluppo del software.

3.2.5 Redmine

Figura 3.10: Redmine

Redmine è un software gratuito e *open-source*, per la pianificazione di progetti e per il tracciamento delle segnalazioni di bug tramite interfaccia web.
Consente agli utilizzatori di gestire molteplici progetti e sotto-progetti tramite una singola installazione.
Le funzionalità a corredo per ogni progetto includono: wiki, forum, time tracking e un flessibile controllo degli accessi basato su ruoli.
Si integra con i più comuni sistemi di controllo versione distribuito.

3.2.6 GitLab

Figura 3.11: GitLab

GitLab è una piattaforma web *open-source* che permette la gestione di *repository Git* e di funzioni *trouble ticket*.
Come tutti i software di controllo versione, GitLab permette la creazione di *repository*, pubblici o privati, in cui gli sviluppatori possono caricare il proprio codice e gestire le modifiche alle varie versioni in contemporanea al lavoro di più persone. In GitLab è possibile lavorare parallelamente ad altre persone sullo stesso progetto senza generare conflitti, caricare il proprio lavoro nel *repository* remoto (operazione di *push*) e poter unire alla fine le modifiche di tutti in un unico progetto (operazione di *merge*).

3.3 Linguaggi

3.3.1 Dart

Figura 3.12: Dart

Dart è un linguaggio di programmazione open source sviluppato da Google.

È un linguaggio orientato agli oggetti flessibile usato per creare applicazioni web e mobile con prestazioni elevate. Dart è stato progettato per mantenere una sintassi familiare e facile da imparare (sullo stile della sintassi del linguaggio C), che ne agevolasse l'adozione all'interno di progetti di ogni dimensione, dal singolo programmatore one-man-band ai grandi team strutturati che lavorano su progetti complessi.

La sintassi Dart può compilarsi in codice nativo o direttamente in JavaScript. Secondo le intenzioni di sviluppo, Dart avrebbe dovuto sostituire JavaScript come linguaggio di programmazione principale per le applicazioni web.

3.3.2 Flutter

Figura 3.13: Flutter

Flutter è un*framework*[†] *open-source* ideato direttamente da Google per rispondere alla continua evoluzione dei dispositivi mobile. L'obiettivo è quello di utilizzare una singola codebase per la programmazione informatica di app *cross platform* che girano in maniera nativa su dispositivi Android e sui dispositivi iOS e, inoltre, con le ultime versioni anche Web e Desktop.

Le prestazioni native di Flutter derivano da uno substrato scritto in C/C++, o meglio chiamato *Flutter Engine*, e uno strato scritto in Dart.

4 | Analisi del sistema precedente

La precedente applicazione, a causa di un time to market *estremamente ridotto, era stata rilasciata in fase prototipale. Tale carenza di progettazione ne ha impedito un ciclo di vita e uno sviluppo conformi agli standard di* quality engineering *imposti dalla ISO-13485. Era stata sviluppata in C++ con l'utilizzo del* framework *Qt*[†].

4.1 Diagramma delle classi

Dopo una discussione con l'azienda, data la confusione presente nel codice dell'applicativo precedente, si è deciso di mappare le dipendenze tra le varie classi invece che realizzare un vero diagramma *UML*[†].

Il codice del precedente applicativo non presentava uso di *design pattern* significativi.

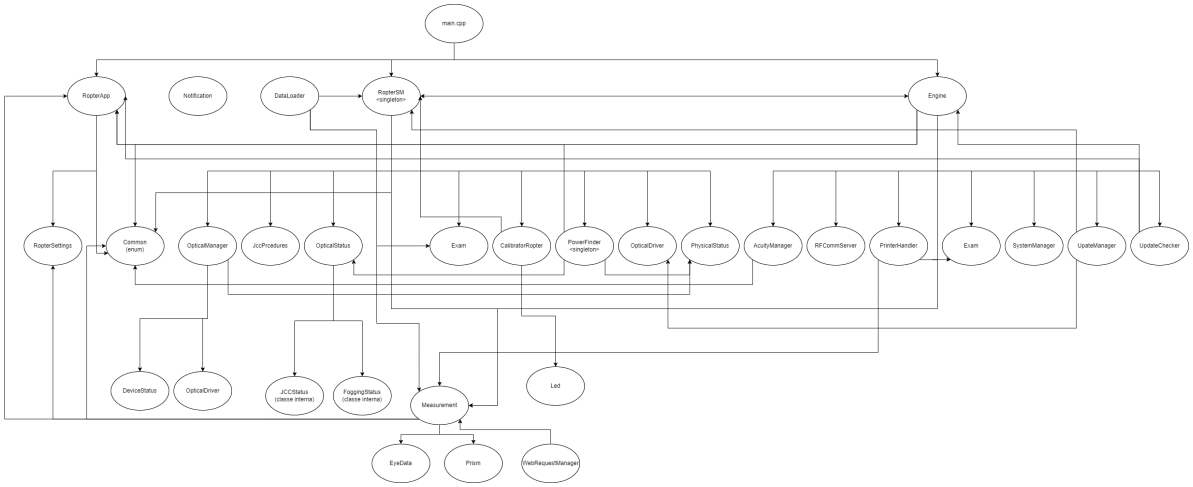


Figura 4.1: Dipendenze tra classi del precedente sistema

4.2 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi dei casi d'uso (in inglese *Use Case Diagrams*), che sono diagrammi di tipo *UML* dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

UC0: main screen

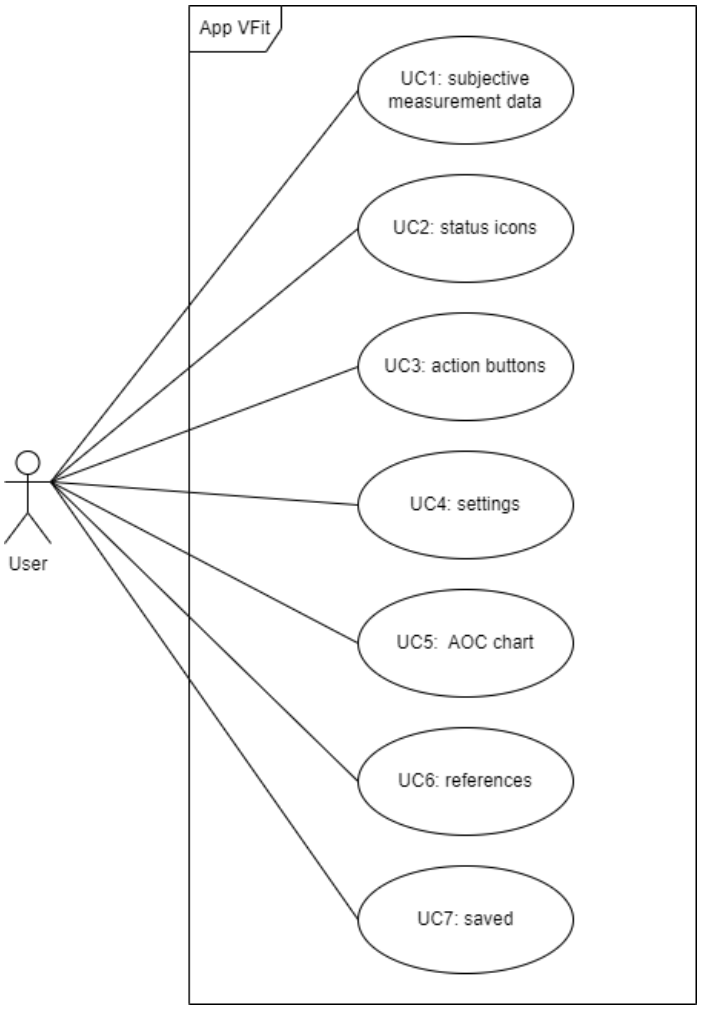


Figura 4.2: UC0: main screen - old

Attori Principali: Utente.

Precondizioni: L'utente ha avviato l'applicazione.

Descrizione: La schermata principale dell'applicazione permette di:

- vedere i dati relativi alla misurazione attuale (UC1);

- vedere delle icone di stato (UC2);
- interagire con dei bottoni (UC3);
- accedere alle impostazioni (UC4);
- accedere alla gestione di Aquid Chart (UC5);
- accedere alle misurazioni oggettive salvate (UC6);
- accedere alle misurazioni soggettive salvate (UC7).

Postcondizioni: Il sistema è pronto per permettere una nuova interazione.

UC1: subjective measurement data

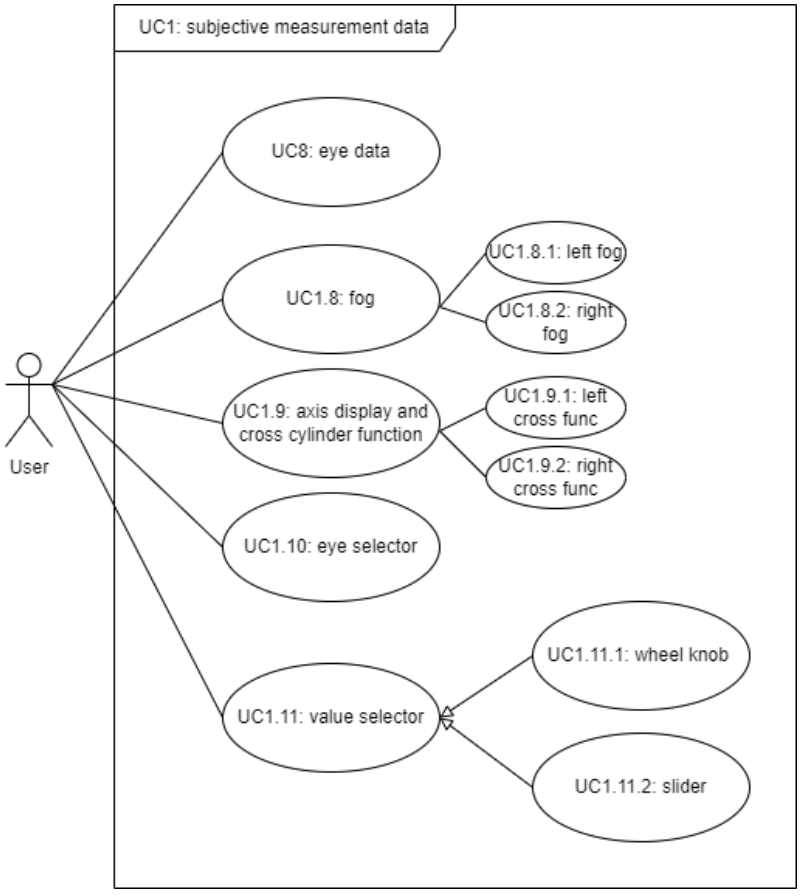


Figura 4.3: UC1: subjective measurement data - old

Attori Principali: Utente.
Precondizioni: L'utente si trova nella schermata principale.

Descrizione: L'utente visualizza i valori delle lenti (UC8), può selezionare l'occhio e quale valore modificare.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC2: status icons

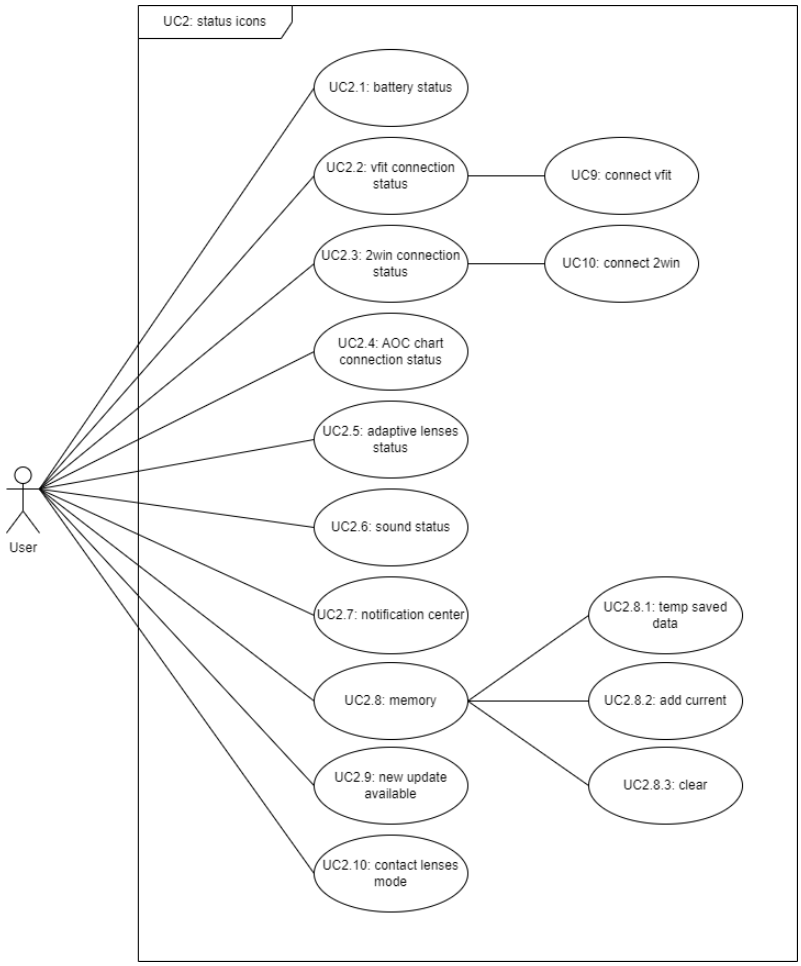


Figura 4.4: UC2: status icons - old

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale.

Descrizione: L'utente visualizza le icone di stato e può interagire con alcune di esse:

- icona di connessione del VisionFit/Aquid per connettere un dispositivo (UC9);
- icona di connessione 2WIN per connettere un dispositivo (UC10);
- icona memory per visualizzare e/o salvare misurazioni temporanee.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC3: action buttons

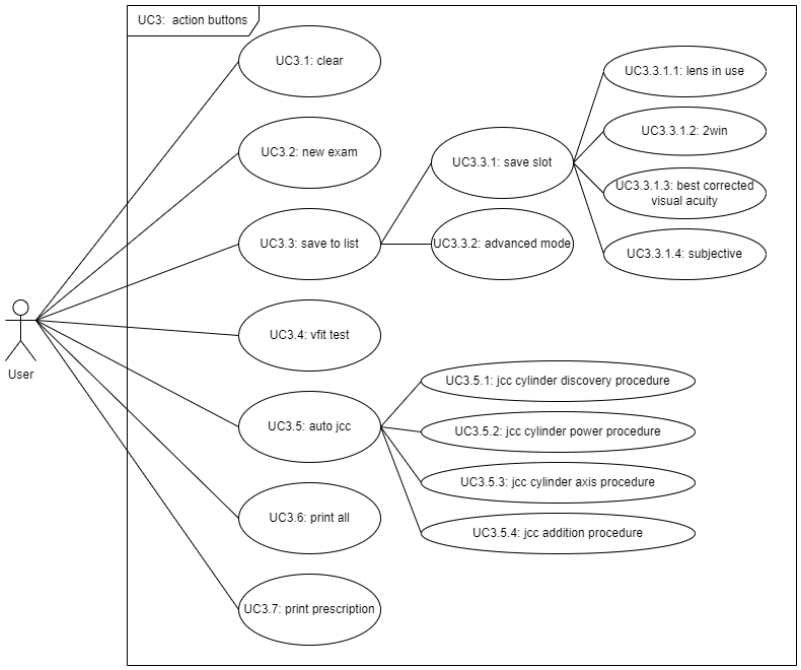


Figura 4.5: UC3: action buttons - old

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale.

Descrizione: L'utente può interagire con i bottoni:

- clear: ripristina i valori attuali ai valori di default;
- new exam: crea una nuova misurazione;
- save to list: salva la misurazione attuale come misurazione oggettiva o soggettiva;
- vfit test: funzione per confrontare le misurazioni salvate;
- auto jcc: funzione semi-automatizzata per il jcc;
- print all: stampa in formato pdf tutte le misurazioni salvate;
- print prescription: stampa in formato pdf la prescrizione oculistica.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC4: settings

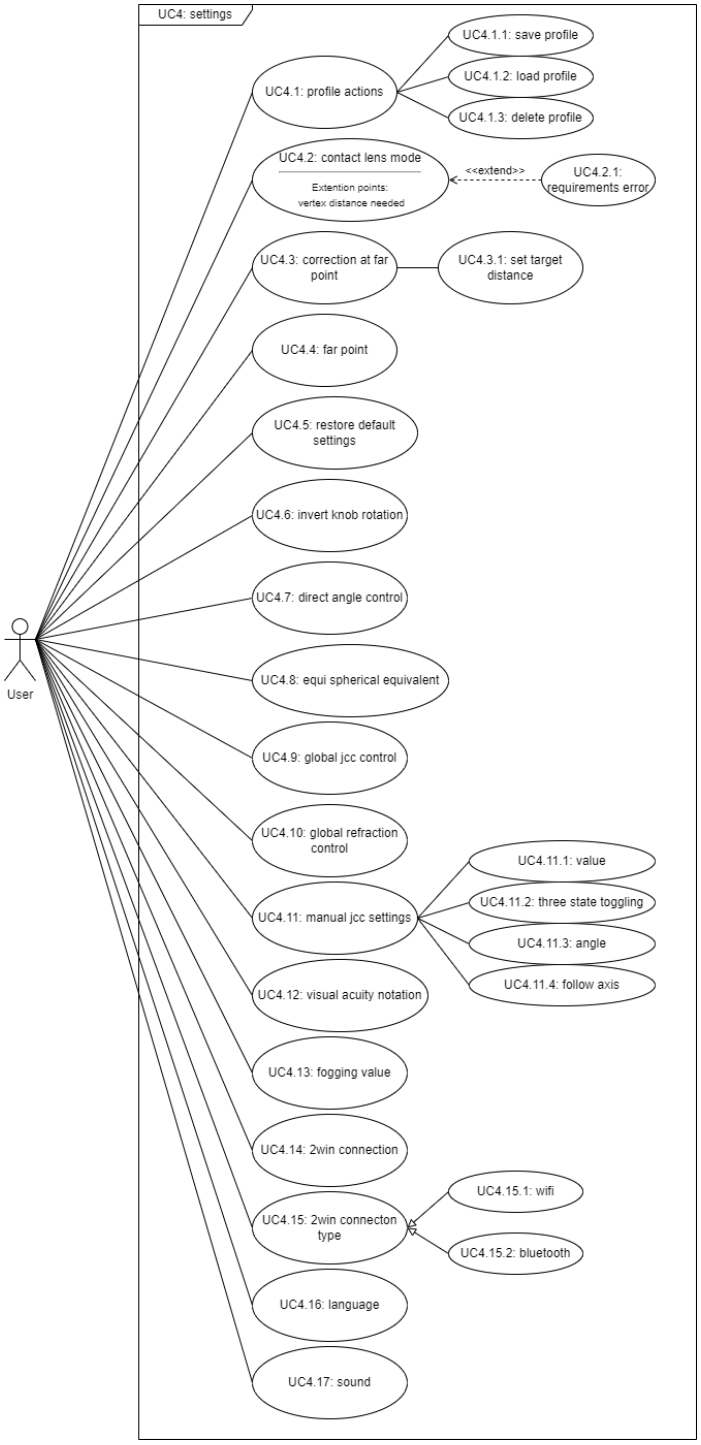


Figura 4.6: UC4: settings - old

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale e ha aperto la scheda delle impostazioni.

Descrizione: L'utente visualizza e può modificare le varie impostazioni, può anche creare e salvare dei profili di impostazioni.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC6: references

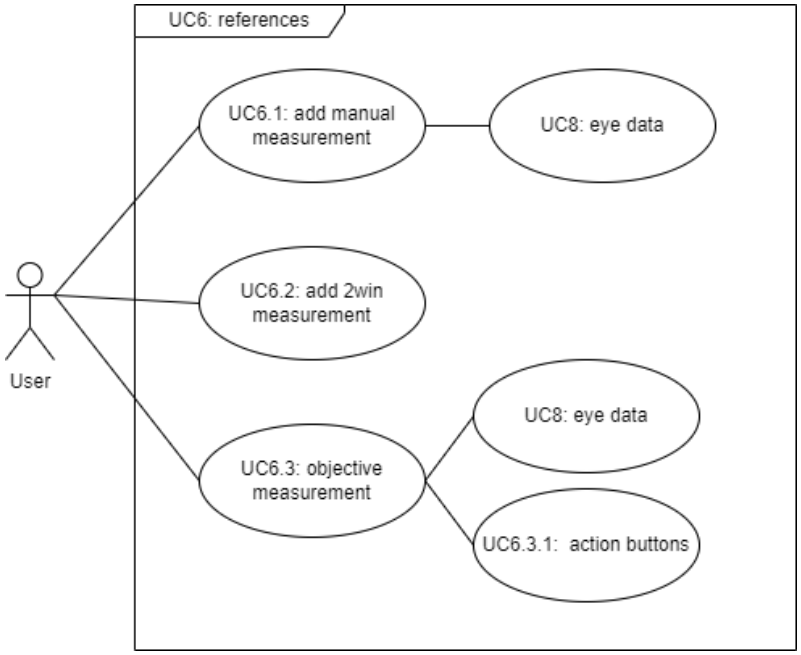


Figura 4.7: UC6: references - old

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale e ha aperto la scheda delle misurazioni oggettive.

Descrizione: L'utente visualizza, se presenti, le misurazioni oggettive salvate. Può inoltre aggiungerne manualmente e richiedere ad un dispositivo 2WIN, se connesso, la sua ultima misurazione. Le misurazioni possono essere eliminate o aggiunte/rimosse dal Vfit Test.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC7: saved

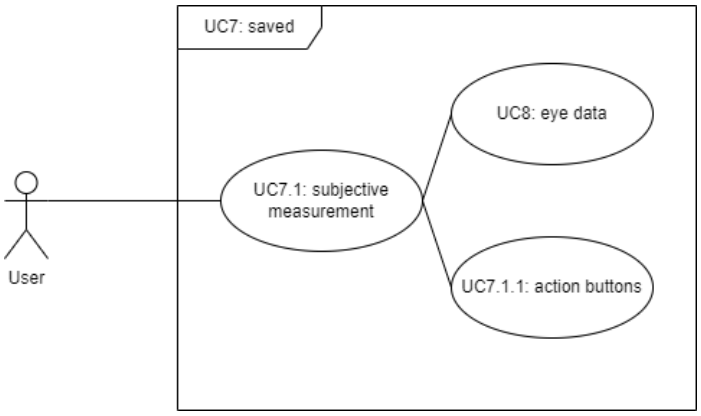


Figura 4.8: UC7: saved - old

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale e ha aperto la scheda delle misurazioni soggettive.

Descrizione: L'utente visualizza, se presenti, le misurazioni soggettive salvate. Le misurazioni possono essere eliminate o aggiunte/rimosse dal Vfit Test.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC8: eye data

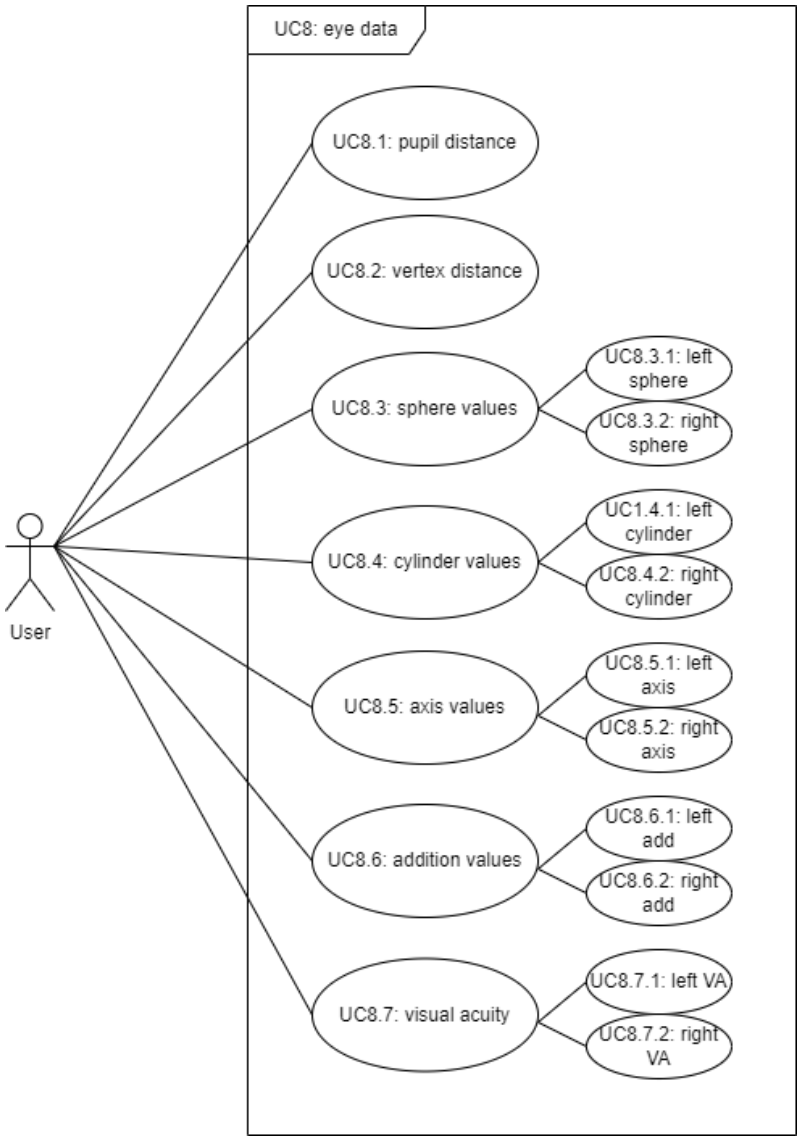


Figura 4.9: UC8: eye data - old

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale.

Descrizione: L'utente visualizza i dati della misurazione attiva divisi tra destra e sinistra.

Postcondizioni: Il sistema attende una nuova interazione.

5 | Architettura del software

5.1 Design pattern e scelte progettuali

Il precedente applicativo non presentava l'utilizzo di alcun *design pattern* significativo e non presentava una chiara distinzione tra modello e viste a livello architetturale.

Per la realizzazione del nuovo applicativo è stato deciso, ed è stato approvato dall'azienda, l'utilizzo di *MVC*† come *design pattern* architetturale e di utilizzare il pacchetto GetX per la gestione degli stati, dei percorsi e delle dipendenze.
Non sono stati inseriti ulteriori *design pattern* poiché ritenuti superflui dato l'utilizzo di GetX.

5.1.1 GetX

GetX è una soluzione estremamente leggera e potente per Flutter. Combina la gestione degli stati, delle dipendenze e dei percorsi in modo rapido e pratico.
Si basa su tre principi fondamentali: **produttività**, **prestazioni** e **organizzazione**. Ciò significa che questi sono la priorità per tutte le risorse della libreria.

- PRESTAZIONI: si concentra sulle prestazioni e sul consumo minimo di risorse;
- PRODUTTIVITÀ: utilizza una sintassi facile, permettendo di risparmiare tempo nello sviluppo e fornendo ottime prestazioni all'applicazione;
- ORGANIZZAZIONE: consente il disaccoppiamento totale della *view*, *presentation logic*†, *business logic*†, *Dependency Injection*† e della navigazione.

5.2 Diagramma delle classi

Le classi del precedente sistema sono state riorganizzate per maggior chiarezza e per garantire il rispetto del *Single Responsibility Principle*†.
I diagrammi *UML*, realizzati mediante PlantUML, sono stati semplificati rimuovendo parte di attributi e metodi per motivi di spazio all'interno del documento.

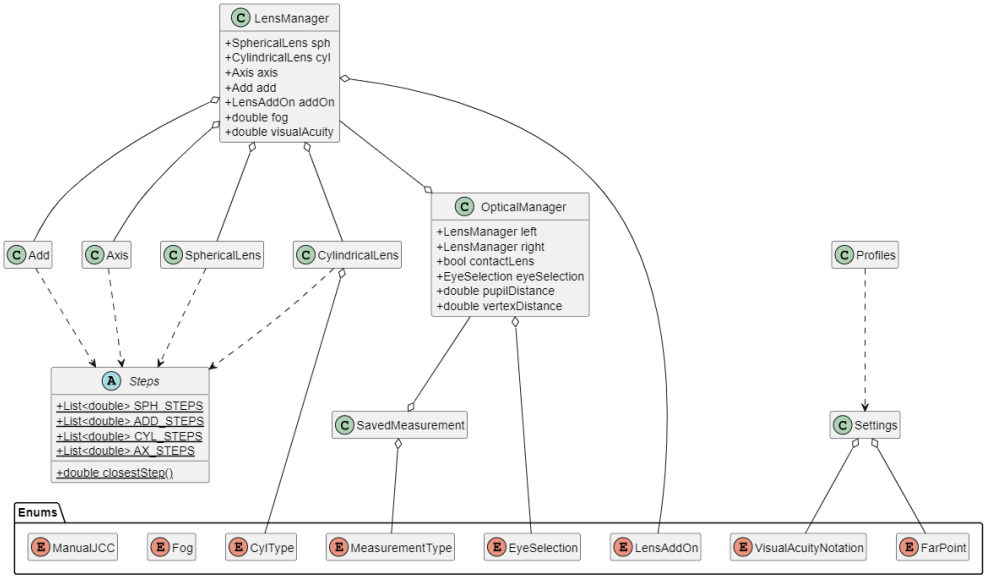


Figura 5.1: UML: model - pt. 1

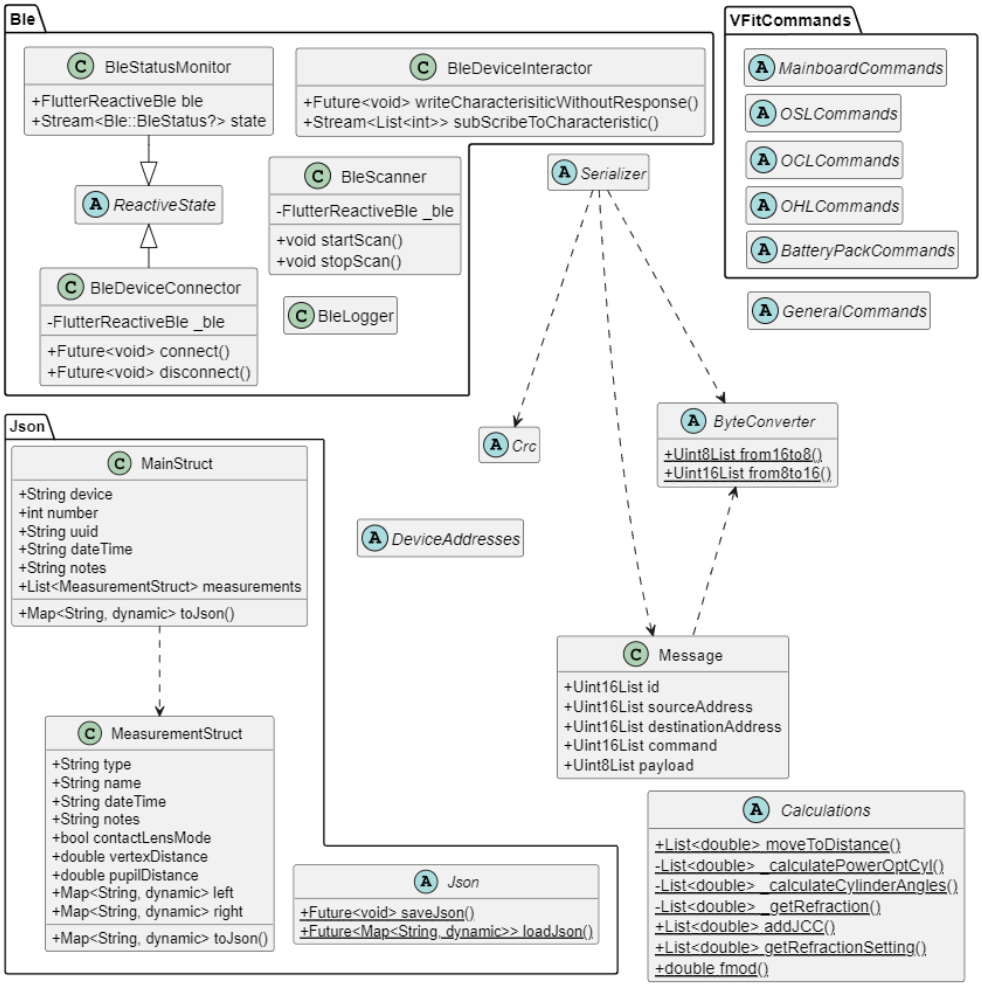


Figura 5.2: UML: model - pt. 2

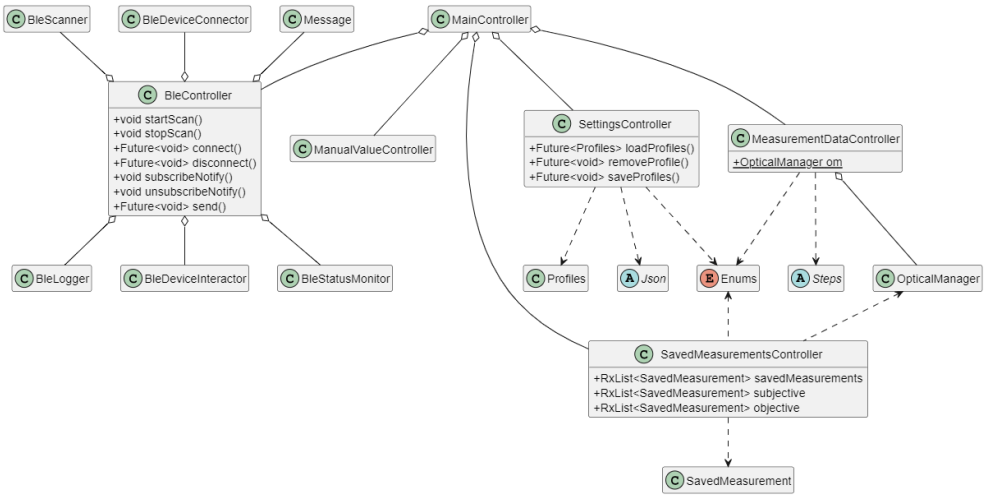


Figura 5.3: UML: controller

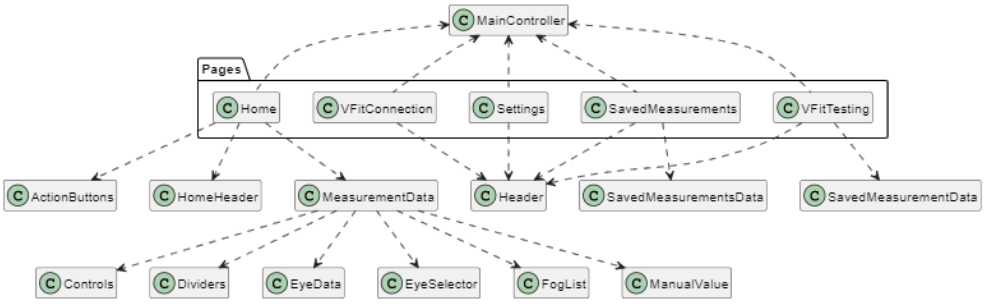


Figura 5.4: UML: view

5.3 Casi d'uso

UC0: main screen

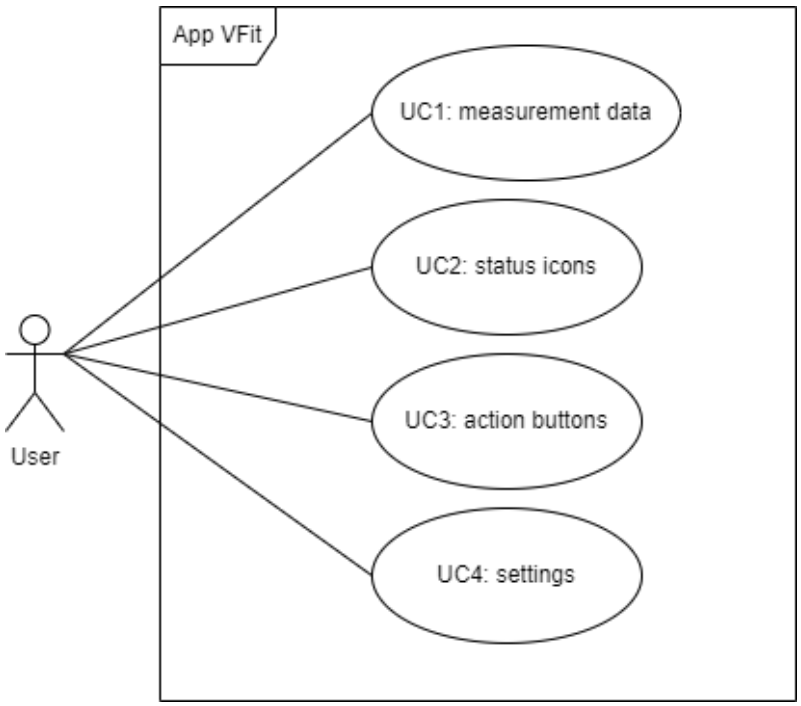


Figura 5.5: UC0: main screen

Attori Principali: Utente.
Precondizioni: L'utente ha avviato l'applicazione.
Descrizione: La schermata principale dell'applicazione permette di:

- vedere i dati relativi alla misurazione attuale (UC1);
- vedere delle icone di stato (UC2);
- interagire con dei bottoni (UC3);
- accedere alle impostazioni (UC4).

Postcondizioni: Il sistema è pronto per permettere una nuova interazione.

UC1: measurement data

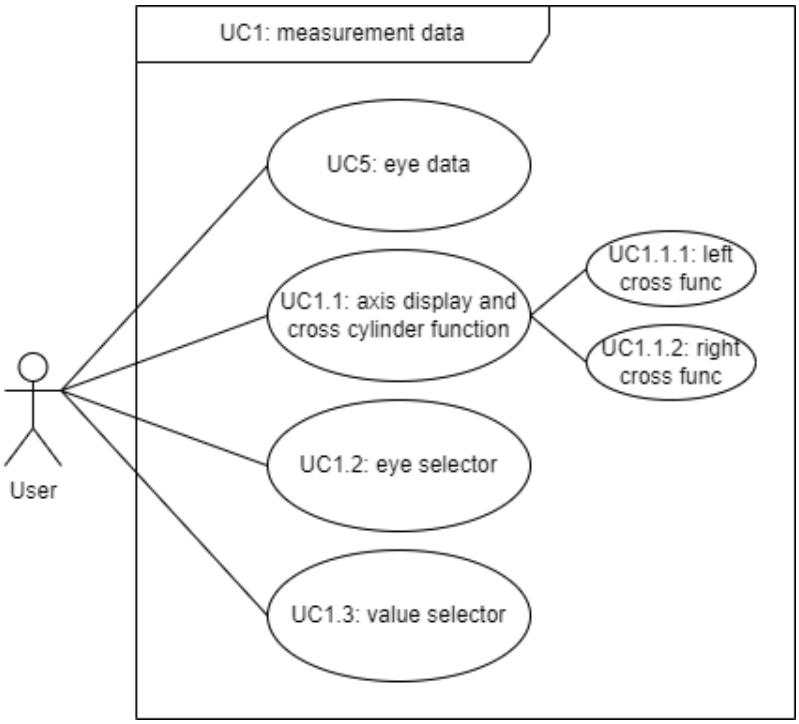


Figura 5.6: UC1: measurement data

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale.

Descrizione: L'utente visualizza i valori delle lenti (UC5), può selezionare l'occhio e quale valore modificare.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC2: status icons

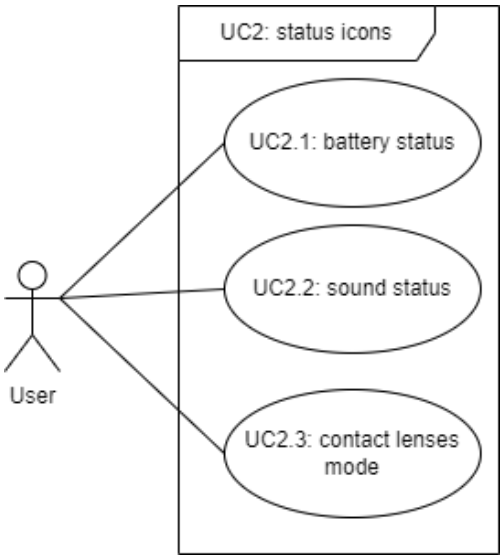


Figura 5.7: UC2: status icons

Attori Principali: Utente.
Precondizioni: L'utente si trova nella schermata principale.
Descrizione: L'utente visualizza le icone di stato.
Postcondizioni: Il attende una nuova interazione.

UC3: action buttons

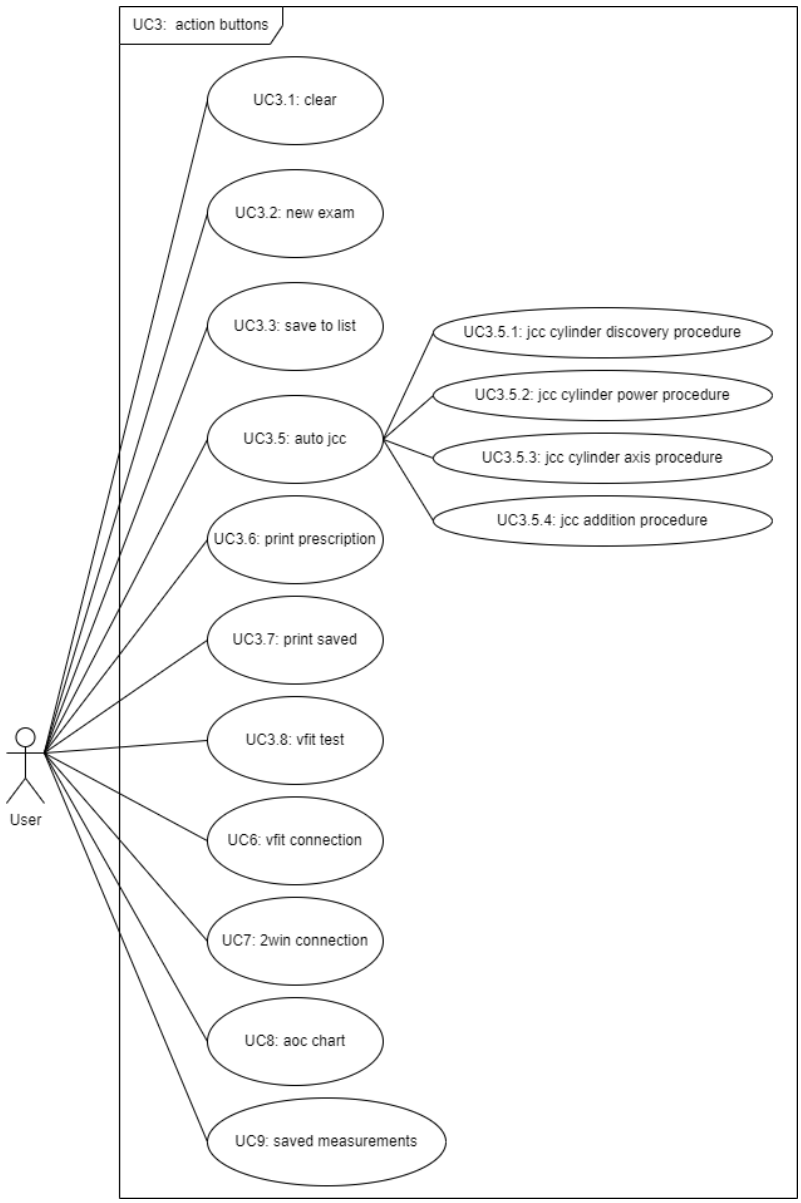


Figura 5.8: UC3: action buttons

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale.

Descrizione: L'utente può interagire con i bottoni:

- clear: ripristina i valori attuali ai valori di default;
- new exam: crea una nuova misurazione;

- save to list: salva la misurazione attuale come misurazione soggettiva;
- vfit test: funzione per confrontare le misurazioni salvate;
- auto jcc: funzione semi-automatizzata per il jcc;
- print all: stampa in formato pdf tutte le misurazioni salvate;
- print prescription: stampa in formato pdf la prescrizione oculistica;
- vfit connection: reindirizza alla pagina per la connessione del dispositivo Vision-Fit/Aquid;
- 2win connection: reindirizza alla pagina per la connessione del dispositivo 2WIN;
- aoc chart: reindirizza alla pagina per la connessione del dispositivo Aquid Chart;
- saved measurements: reindirizza alla pagina per la gestione delle misurazioni salvate.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate, o reindirizza alla pagina desiderata, e attende una nuova interazione.

UC4: settings

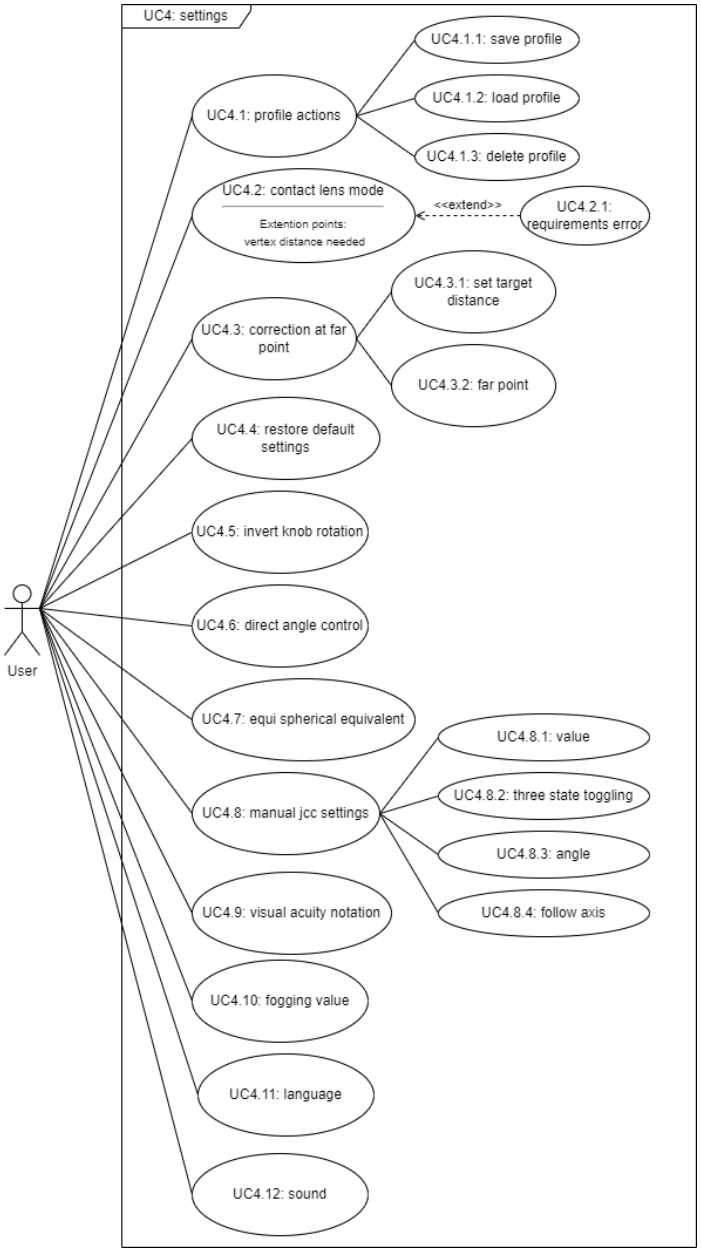


Figura 5.9: UC4: settings

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale e ha aperto la scheda delle impostazioni.

Descrizione: L'utente visualizza e può modificare le varie impostazioni, può anche

creare e salvare dei profili di impostazioni.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC5: eye data

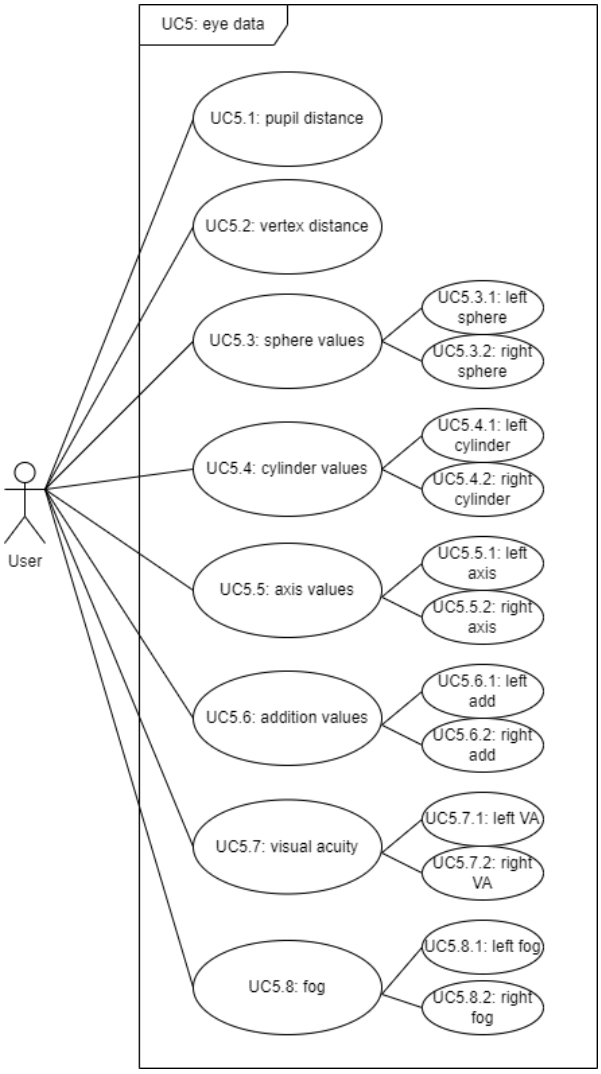


Figura 5.10: UC5: eye data

Attori Principali: Utente.

Precondizioni: L'utente si trova nella schermata principale o nella pagina delle misurazioni salvate.

Descrizione: L'utente visualizza i dati della misurazione attiva divisi tra destra e sinistra.

Postcondizioni: Il sistema attende una nuova interazione.

UC6: vfit connection

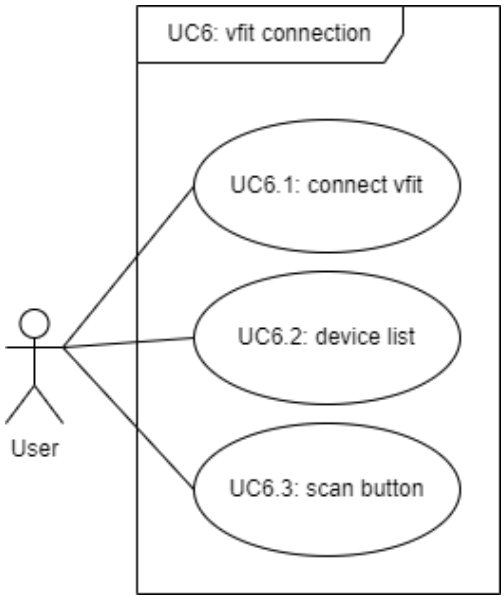


Figura 5.11: UC6: vfit connection

Attori Principali: Utente.

Precondizioni: L'utente ha premuto il bottone per la connessione del dispositivo VisionFit/Aquid e si trova nella corrispondente pagina.

Descrizione: L'utente visualizza un bottone per effettuare una scansione Bluetooth, la lista dei dispositivi scansionati e dopo la connessione il dispositivo connesso, avendo anche la possibilità di disconnetterlo.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC7: 2win connection

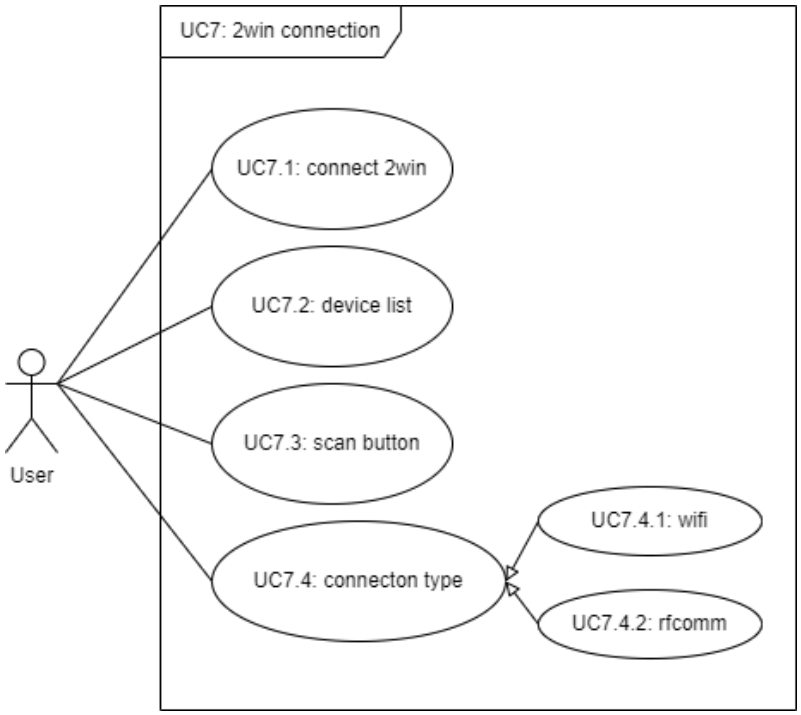


Figura 5.12: UC7: 2win connection

Attori Principali: Utente.

Precondizioni: L'utente ha premuto il bottone per la connessione del dispositivo 2WIN e si trova nella corrispondente pagina.

Descrizione: L'utente visualizza un bottone per effettuare una scansione dei dispositivi, la lista dei dispositivi scansionati e dopo la connessione il dispositivo connesso, avendo anche la possibilità di disconnetterlo. È possibile effettuare la scelta del tipo di connessione, se WiFi o Bluetooth RFCOMM.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC8: aoc chart

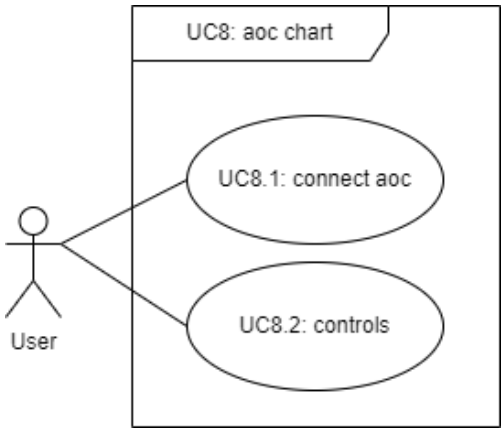


Figura 5.13: UC8: aoc chart

Attori Principali: Utente.

Precondizioni: L'utente ha premuto il bottone per la connessione del dispositivo Aquid Chart e si trova nella corrispondente pagina.

Descrizione: L'utente visualizza un tasto per la connessione e i controlli per gestire il dispositivo.

Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

UC9: saved measurements

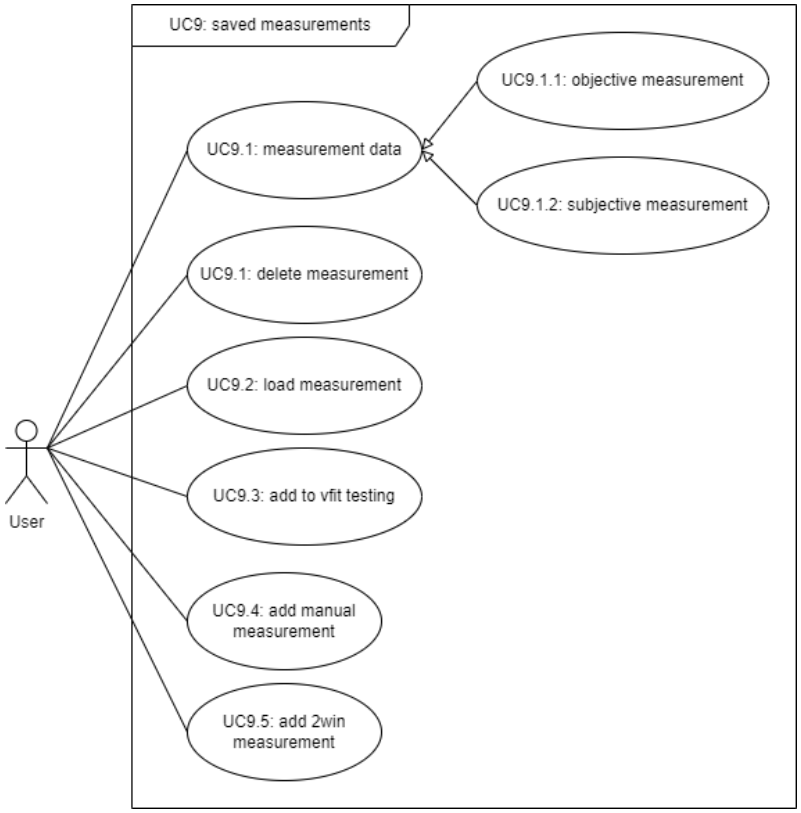


Figura 5.14: UC9: saved measurements

Attori Principali: Utente.

Precondizioni: L'utente ha premuto il bottone per visualizzare le misurazioni salvate e si trova nella corrispondente pagina.

Descrizione: L'utente visualizza le misurazioni salvate, divise tra oggettive e soggettive, ed una serie di pulsanti:

- delete: elimina la misurazione selezionata;
- load: carica la misurazione selezionata come misurazione attiva nella schermata principale;
- add to vfit testing: aggiunge la misurazione selezionata alla lista di misurazioni da testare con il VFit Testing;
- manual measurement: permette di aggiungere una misurazione manualmente;
- 2WIN measurement: permette di aggiungere alle misurazioni oggettive l'ultima misurazione effettuate con il dispositivo 2WIN connesso, se connesso.

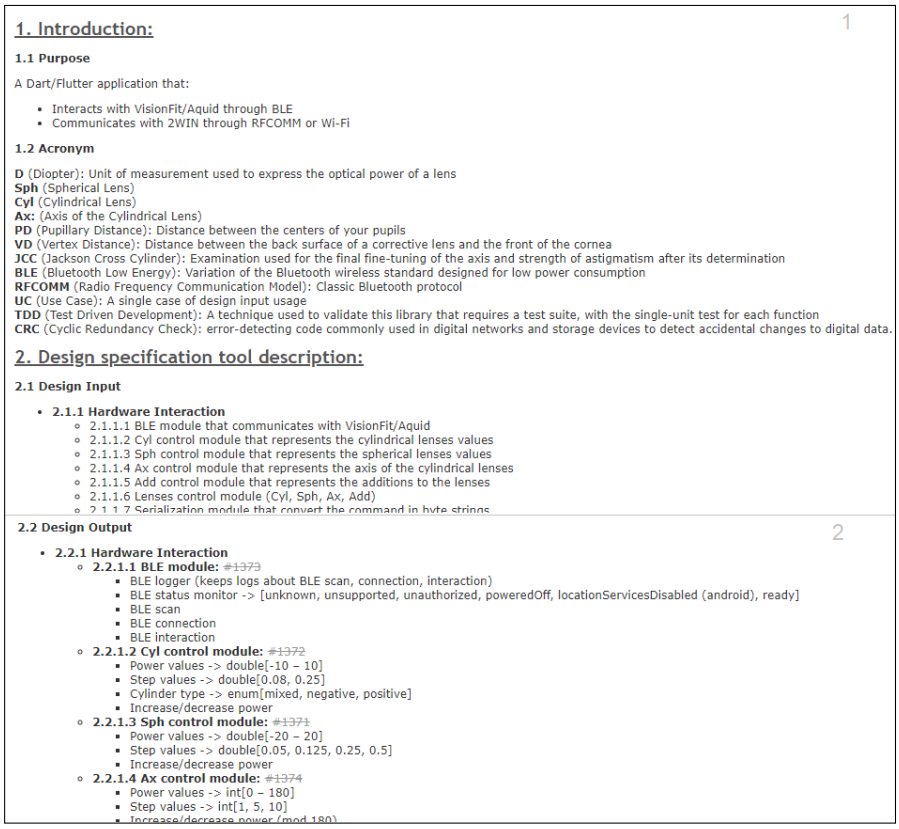
Postcondizioni: Il sistema si aggiorna secondo le modifiche effettuate e attende una nuova interazione.

6 | Sviluppo del software

6.1 Qualità

Prima d'iniziare la stesura del codice, il reparto qualità dell'azienda ha dato una breve spiegazione per quanto riguardasse gli standard ISO utilizzati e l'utilizzo di Redmine e GitLab.

Come prima cosa è stato creato un wiki su Redmine relativo al progetto, definendo un'introduzione, un glossario degli acronimi, la parte di *design input*† richiesta e la parte di *design output*†.



1

1. Introduction:

1.1 Purpose

A Dart/Flutter application that:

- Interacts with VisionFit/Aquid through BLE
- Communicates with 2WIN through RFCOMM or Wi-Fi

1.2 Acronym

D (Diopter): Unit of measurement used to express the optical power of a lens
Sph (Spherical Lens)
Cyl (Cylindrical Lens)
Ax: (Axis of the Cylindrical Lens)
PD (Pupillary Distance): Distance between the centers of your pupils
VD (Vertex Distance): Distance between the back surface of a corrective lens and the front of the cornea
JCC (Jackson Cross Cylinder): Examination used for the final fine-tuning of the axis and strength of astigmatism after its determination
BLE (Bluetooth Low Energy): Variation of the Bluetooth wireless standard designed for low power consumption
RFCOMM (Radio Frequency Communication Model): Classic Bluetooth protocol
UC (Use Case): A single case of design input usage
TDD (Test Driven Development): A technique used to validate this library that requires a test suite, with the single-unit test for each function
CRC (Cyclic Redundancy Check): error-detecting code commonly used in digital networks and storage devices to detect accidental changes to digital data.

2. Design specification tool description:

2.1 Design Input

- **2.1.1 Hardware Interaction**
  - 2.1.1.1 BLE module that communicates with VisionFit/Aquid
  - 2.1.1.2 Cyl control module that represents the cylindrical lenses values
  - 2.1.1.3 Sph control module that represents the spherical lenses values
  - 2.1.1.4 Ax control module that represents the axis of the cylindrical lenses
  - 2.1.1.5 Add control module that represents the additions to the lenses
  - 2.1.1.6 Lenses control module (Cyl, Sph, Ax, Add)
  - 2.1.1.7 Serialization module that convert the command in byte strings

2

2.2 Design Output

- **2.2.1 Hardware Interaction**
  - **2.2.1.1 BLE module:** ~~#1373~~
    - BLE logger (keeps logs about BLE scan, connection, interaction)
    - BLE status monitor -> [unknown, unsupported, unauthorized, poweredOff, locationServicesDisabled (android), ready]
    - BLE scan
    - BLE connection
    - BLE interaction
  - **2.2.1.2 Cyl control module:** ~~#1372~~
    - Power values -> double[-10 – 10]
    - Step values -> double[0.08, 0.25]
    - Cylinder type -> enum[mixed, negative, positive]
    - Increase/decrease power
  - **2.2.1.3 Sph control module:** ~~#1371~~
    - Power values -> double[-20 – 20]
    - Step values -> double[0.05, 0.125, 0.25, 0.5]
    - Increase/decrease power
  - **2.2.1.4 Ax control module:** ~~#1374~~
    - Power values -> int[0 – 180]
    - Step values -> int[1, 5, 10]
    - Increase/decrease power (mod 180)

Figura 6.1: Parte del wiki su Redmine

Si è passati poi alla creazione delle *issue*, sempre su Redmine, che sono state collegate tramite riferimento ai *commit* di *Git*.
Nelle *issue* è stato possibile tenere traccia del progresso e delle tempistiche, che sono state automaticamente aggiunte ad un diagramma di Gantt interno a Redmine. Alla chiusura delle *issue* è sempre stato associato un report del superamento dei test.

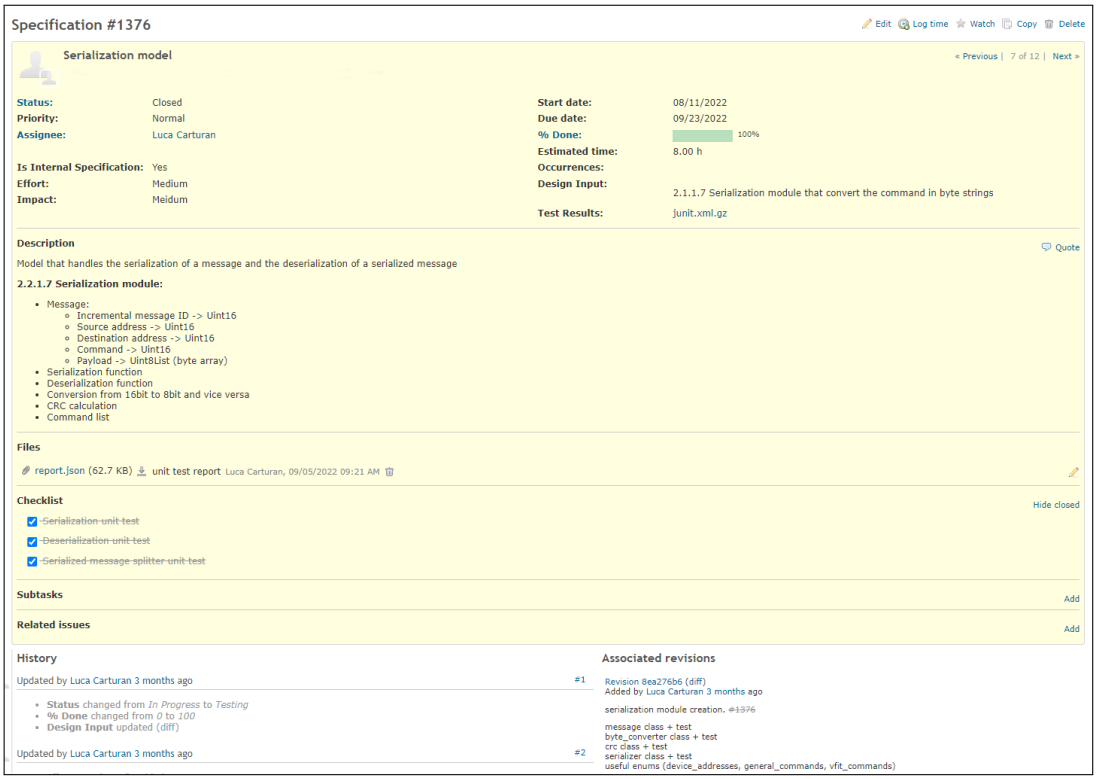


Figura 6.2: Esempio di issue su Redmine

Per la gestione dei *branch* su GitLab ci si è basati su un modello preso da "A successful Git branching model", un famoso post scritto da Vincent Driessen.
L'azienda ha tenuto a ricordare che Git è uno strumento di versionamento e, in quanto tale, non dovrebbe essere utilizzato come strumento di backup o sincronizzazione tra differenti PC o posti di lavoro. Questo perché la versione precedente del prodotto presentava *commit* per effettuare backup o per modifiche insignificanti.

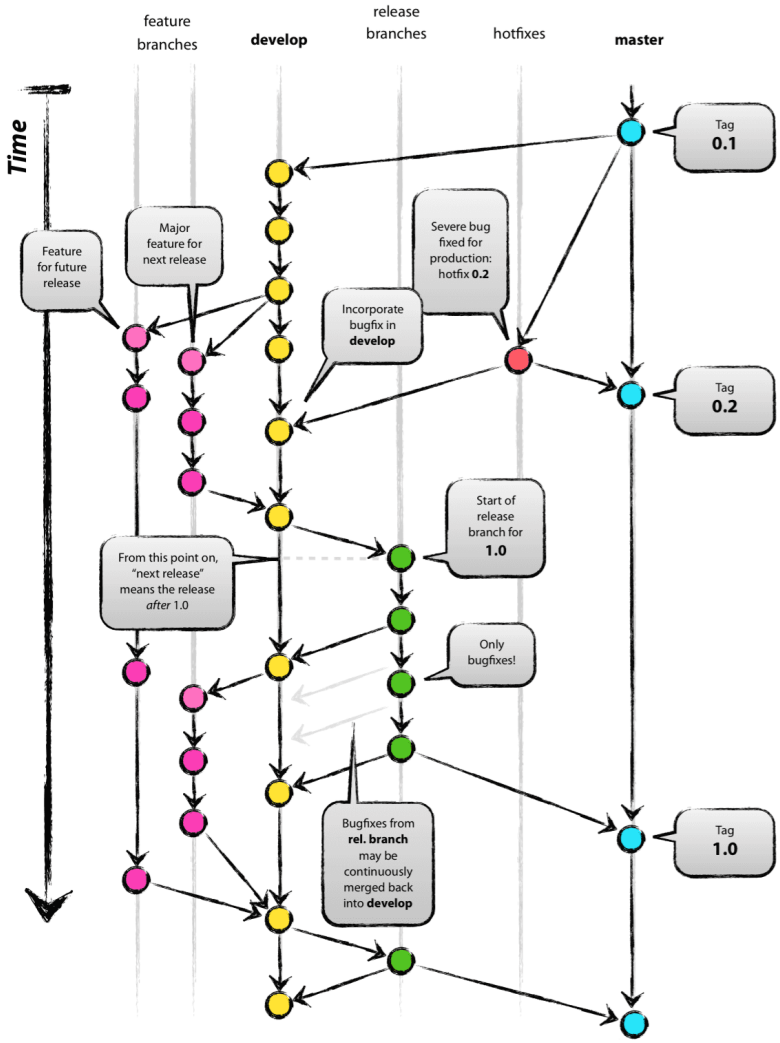


Figura 6.3: Git branching model

È stata inoltre realizzata una *pipeline*† su GitLab per avere *continuous integration*† e *continuous delivery*†, la quale si compone di quattro passi che, come per definizione di *pipeline*, devono essere tutti superati in sequenza:

- analisi statica del codice;
- controllo dei test con generazione del report;
- build dell'applicazione in formato apk;
- generazione della documentazione.

```yaml
---
image:
  name: adaptica/build-android-flutter
stages:
- ".pre"
- analyze
- test
- build
- deploy
- ".post"
before_script:
- export PATH="$PATH":"/home/flutter/.pub-cache/bin"
- export PATH="$PATH":"$HOME/.pub-cache/bin"
- flutter pub get
- flutter clean
- flutter pub global activate junitreport
analyze:
  stage: analyze
  script:
  - flutter analyze
test:
  stage: test
  script:
  - ls /home
  - flutter pub global activate junitreport
  - flutter test --machine | tojunit -o report.xml
  artifacts:
    when: always
    reports:
      junit:
      - report.xml
build:
  stage: build
  script:
  - flutter build apk
  artifacts:
    paths:
    - build/app/outputs/flutter-apk/app-release.apk
documentation:
  stage: deploy
  script:
  - flutter pub global activate dartdoc
  - dartdoc
  artifacts:
    paths:
    - doc/api
```

Figura 6.4: Pipeline CI/CD

6.2 Model

Il modello è stato scritto seguendo le direttive del *Test Driven Development (TDD)*†, risultando così interamente validato al suo completamento.
Si è cercato di mantenere il *Single Responsibility Principle* riadattando e dividendo in più parti le classi del precedente sistema.
Il modello è stato suddiviso in tre gruppi principali:

- gestione delle lenti;
- gestione della connessione Bluetooth;
- gestione dei messaggi.

6.2.1 Gestione delle lenti

La gestione delle lenti nel sistema precedente era gestita interamente da una classe contenente gli attributi per ogni valore di ciascuna lente, nel nuovo sistema è gestita da singole classi aggregate in una classe padre che le gestisce per mezzo di funzioni, senza preoccuparsi dell'effettivo funzionamento delle singole.

6.2.2 Gestione della connessione Bluetooth

La gestione della componente Bluetooth è risultata differente tra Qt e Flutter ed è stata realizzata col supporto del pacchetto *flutter_ reactive_ ble* che aiuta nella gestione della connessione e interazione tramite *Bluetooth Low Energy*†.
La componente Bluetooth è stata gestita mediante cinque classi:

- **BleScanner**: gestisce la parte di scansione dei dispositivi;
- **BleDeviceConnector**: gestisce la parte di connessione ai dispositivi;
- **BleDeviceInteractor**: gestisce la scansione e l'interazione coi servizi disponibili;
- **BleLogger**: salva in un *log* le azioni effettuate;
- **BleStatusMonitor**: controlla lo stato del Bluetooth.

6.2.3 Gestione dei messaggi

I messaggi sono gestiti secondo un protocollo Bluetooth realizzato dall'azienda, per gestirlo tramite Dart sono stati usati i tipi di dato Uint8List e Uint16List, che corrispondono rispettivamente a liste di elementi formati da 8 e 16 bit, e per la conversione tra i due tipi di liste sono state create delle apposite funzioni, poiché di base non presenti.
Sono state inoltre create una classe per il calcolo del *CRC*† e la classe **Serializer**, che è gestita come una macchina a stati per controllare la validità del messaggio in ingresso o in uscita e convertirlo nella forma adatta.

6.3 View

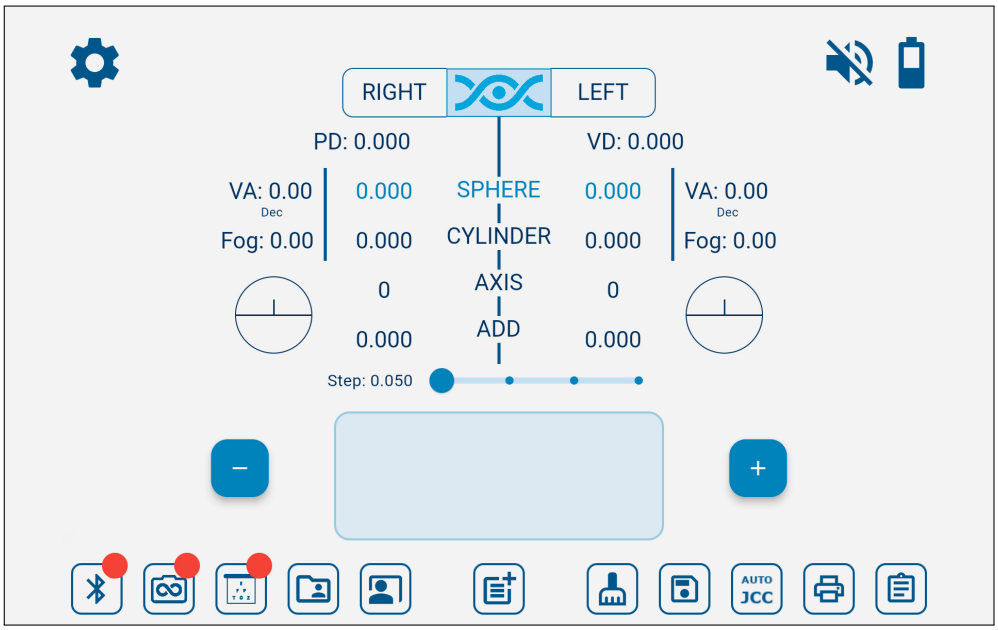
Lo sviluppo dell'interfaccia è cominciato con la progettazione tramite Figma, la quale ha permesso di realizzare, e prototipizzare, varie possibili versioni per le diverse pagine da creare.

Figura 6.5: Progettazione tramite Figma

Dopo aver sottoposto le idee all'azienda, sono state scelte le versioni ritenute migliori e si è passati all'implementazione.
È stata realizzata una serie di *widget*[†], di cui alcuni parametrizzati così da permettere un maggior riutilizzo del codice, per poi essere assemblati nelle classi relative alle pagine complete.
Le pagine realizzate sono le seguenti:

- **Home**: è la schermata principale che contiene i dati della misurazione e una serie di bottoni con cui interagire per cambiare pagina o effettuare determinate azioni;

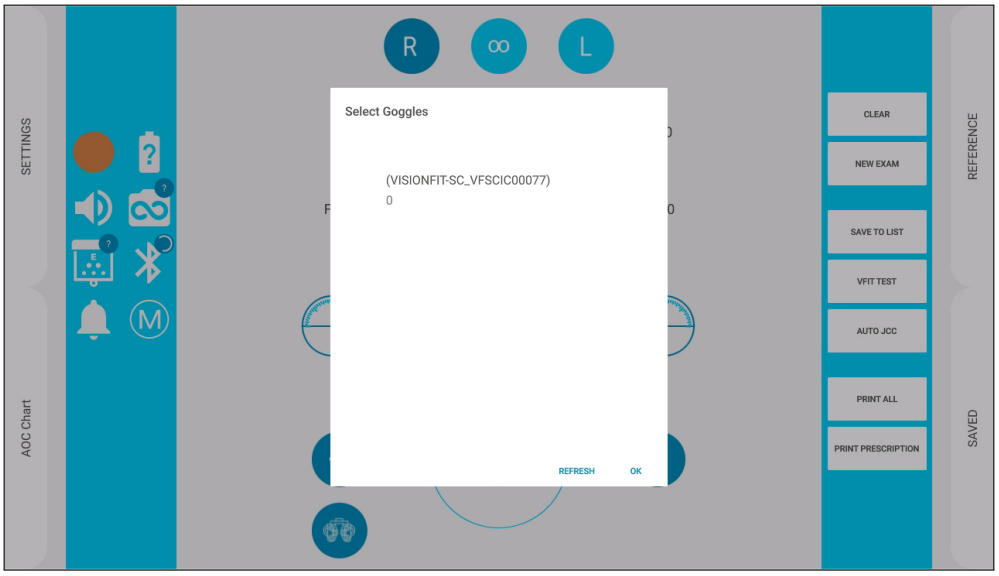
(a) vecchio



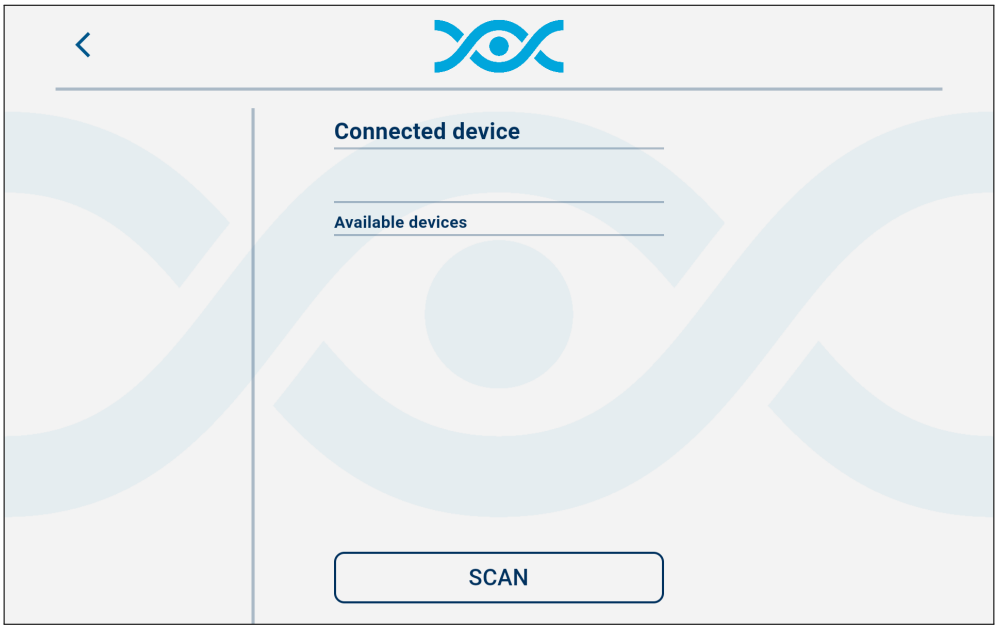
(b) nuovo

Figura 6.6: Confronto tra il vecchio e il nuovo pannello principale

- **VFitConnection**: è la pagina in cui cercare un dispositivo VisionFit/Aquid da connettere;



(a) vecchio

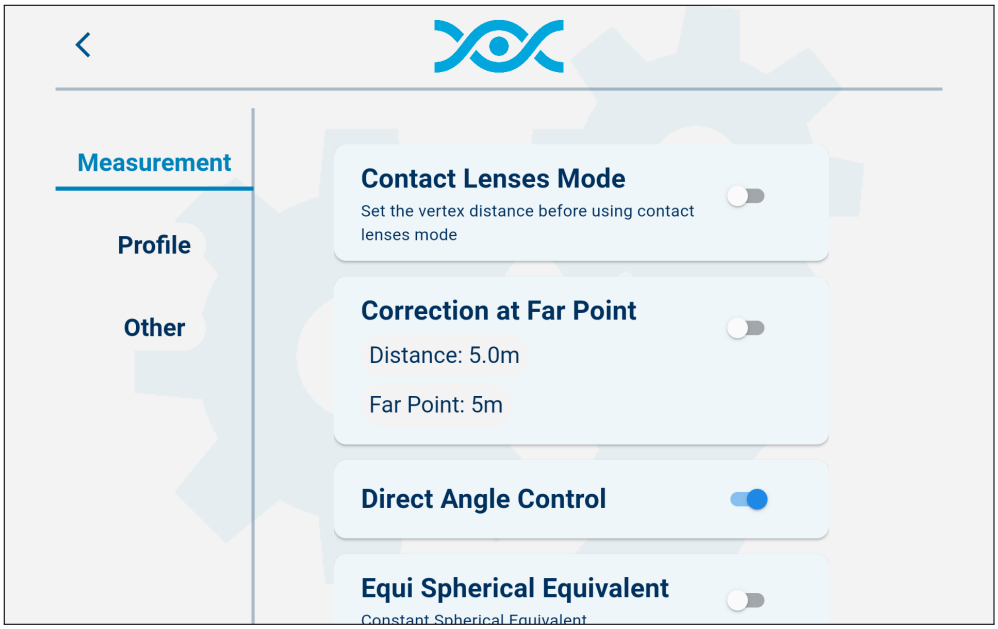


(b) nuovo

Figura 6.7: Confronto tra il vecchio e il nuovo pannello della connessione al VisionFit/Aquid

- **Settings**: è la pagina contenente le impostazioni suddivise per categorie;

(a) vecchio

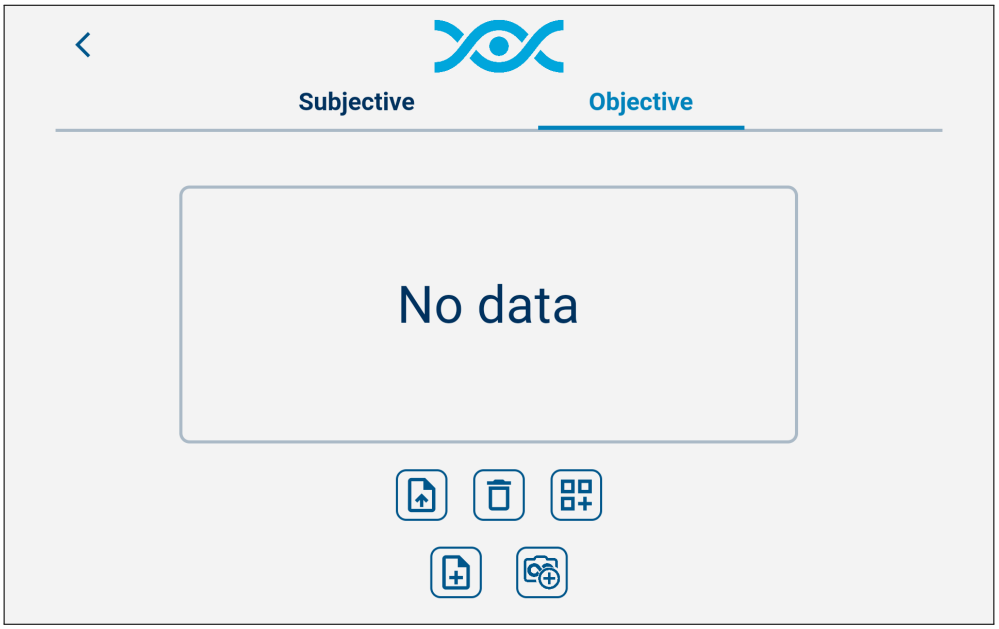


(b) nuovo

Figura 6.8: Confronto tra il vecchio e il nuovo pannello delle impostazioni

- **SavedMeasurements**: è la pagina in cui trovare le misurazioni salvate, divise tra soggettive e oggettive;

(a) vecchio

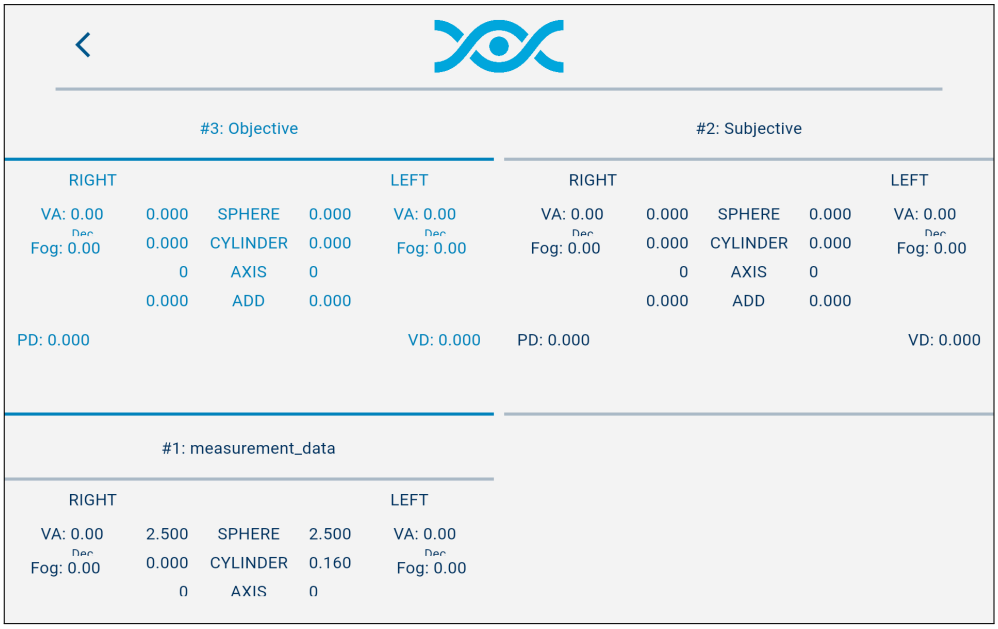


(b) nuovo

Figura 6.9: Confronto tra il vecchio e il nuovo pannello delle misurazioni oggettive salvate

- **VFitTesting**: è la pagina dove effettuare la funzione di VFit Testing per confrontare le misure salvate selezionate.

(a) vecchio



(b) nuovo

Figura 6.10: Confronto tra il vecchio e il nuovo pannello del VFit Testing

È stato creato un controller per ogni funzionalità principale e un "MainController" contenente le istanze dei precedenti, così da permettere un dialogo tra le varie componenti.
Ogni controller eredita dalla classe GetxController, così da permetterne la gestione mediante il pacchetto GetX, e tutte le variabili da rendere osservabili utilizzano Rx<T> come tipologia di dato (T è un generico tipo di dato).

Dove si presentava la necessità di aggiornare il *widget* in base alla variazione di un dato, quel *widget* è stato avvolto in un *widget* Obx, appartenente al pacchetto GetX, il quale permette di ascoltare le modifiche effettuate agli oggetti resi osservabili aggiornando in maniera reattiva solo i valori necessari.

7 | Conclusioni

Lo stage è stato svolto a partire dal 18/07/2022 e doveva terminare il 23/09/2022 ma, a causa di un mio infortunio che ha richiesto circa un mese di pausa, si è concluso il 31/10/2022 per un totale di 308 ore rendicontate ed una valenza di 11 crediti formativi.

7.1 Raggiungimento degli obiettivi

7.1.1 Obiettivi obbligatori

Id	Risultato
ob01	Concluso
ob02	Concluso
ob03	Concluso
ob04	Concluso
ob05	Concluso
ob06	Concluso
ob07	Concluso
ob08	Non concluso
ob09	Concluso
ob10	Concluso

7.1.2 Obiettivi desiderabili

Id	Risultato
de01	Concluso
de02	Concluso

7.1.3 Obiettivi obbligatori

Id	Risultato
op01	Concluso
op02	Concluso

Il requisito ob08 non è stato portato a termine per un problema di tempistiche da parte dell'azienda.

7.2 Conoscenze acquisite

Questo stage formativo mi ha portato ad avere una buona conoscenza del linguaggio Flutter/Dart e una buona capacità di gestione di progetto tramite *Git*, questo molto utile nella gestione di un qualsiasi progetto, sia esso grande o piccolo.

7.3 Valutazione personale

Sono soddisfatto di quello che ho imparato durante il mio tirocinio ed è stata una grande soddisfazione vedere un prodotto prendere forma passo passo, fino ad arrivare ad un prodotto funzionante.
Per quanto riguarda l'ambiente lavorativo ho trovato professionalità in un ambiente giovanile e allegro dove è stato facile trovare sostegno ed aiuto quando necessario.

Acronimi e abbreviazioni

API Application Program Interface. 11, 12, 59

CRC Cyclic Redundancy Check. 47, 60

MVC Model-View-Controller. 27, 61

TDD Test-Driven Development. 47, 63

UI User Interface. 14, 63

UML Unified Modeling Language. 1, 14, 17, 18, 27, 63

UX User Experience. 14, 63

Glossario

API In informatica con il termine *Application Programming Interface, API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 57

Bluetooth Low Energy Colloquialmente BLE, è una tecnologia wireless personal area network progettata e commercializzata dal Bluetooth Special Interest Group (Bluetooth SIG) per nuove applicazioni nel settore dell'assistenza sanitaria, fitness, per i beacon, per la sicurezza, per l'industria dell'intrattenimento domestico e per le industrie automobilistiche e dell'automazione. Rispetto al Bluetooth "classico", il Bluetooth Low Energy ha lo scopo di fornire un consumo energetico e un costo notevolmente ridotto, mantenendo un intervallo di comunicazione simile.
Sistemi operativi mobili inclusi iOS, Android, Windows Phone e BlackBerry, nonché macOS, Linux, Windows 8 e Windows 10, supportano nativamente Bluetooth Low Energy. 47

Business Logic In informatica, nell'ambito dello sviluppo software, l'espressione business logic (ing. logica di business) si riferisce a tutta quella logica o nucleo (core) di elaborazione (sotto forma di codice sorgente) che rende operativa un'applicazione. Con tale nome ci si riferisce quindi all'algoritmica che gestisce lo scambio di informazioni tra l'interfaccia utente attraverso la logica di presentazione con le elaborazioni intermedie sui dati estratti ed eventualmente una sorgente dati (generalmente una base dati) deputata alla gestione della persistenza dei dati stessi (nel caso di applicazioni web). 27

Continuous Delivery Spesso abbreviata in CD, è un approccio di ingegneria del software in cui i team producono software in cicli brevi, garantendo che il software possa essere rilasciato in modo affidabile in qualsiasi momento e, quando si rilascia, senza farlo manualmente. L'obiettivo è costruire, testare e rilasciare software con maggiore velocità e frequenza. L'approccio contribuisce a ridurre i costi, i tempi e i rischi legati alla fornitura di modifiche, consentendo un maggior numero di aggiornamenti incrementali alle applicazioni in produzione. Un processo di distribuzione semplice e ripetibile è importante per la continuous delivery.
La continuous delivery si contrappone alla continuous deployment (abbreviata anch'essa in CD), un approccio simile in cui il software viene prodotto in cicli brevi, ma attraverso distribuzioni automatizzate anziché manuali. Per questo

motivo, la continuous deployment può essere vista come una forma di automazione più completa rispetto alla continuous delivery.. 45

Continuous Integration Nell'ingegneria del software, l'integrazione continua (continuous integration in inglese, spesso abbreviata in CI) è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di controllo versione. Consiste nell'allineamento frequente (ovvero "molte volte al giorno") dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso (mainline). Il concetto è stato originariamente proposto nel contesto dell'extreme programming (XP), come contromisura preventiva per il problema del "integration hell" (le difficoltà dell'integrazione di porzioni di software sviluppati in modo indipendente su lunghi periodi di tempo e che di conseguenza potrebbero essere significativamente divergenti).
La CI è stata originariamente concepita per essere complementare rispetto ad altre pratiche, in particolare legate al Test Driven Development (sviluppo guidato dai test, TDD). In particolare, si suppone generalmente che siano stati predisposti test automatici che gli sviluppatori possono eseguire immediatamente prima di rilasciare i loro contributi verso l'ambiente condiviso, in modo da garantire che le modifiche non introducano errori nel software esistente. Per questo motivo, la CI viene spesso applicata in ambienti in cui siano presenti sistemi di build automatico e/o esecuzione automatica di test, come Jenkins. Tuttavia, l'identificazione della CI con l'uso di strumenti di questo tipo, frequente in letteratura, è scorretta: di per sé, la pratica della CI può essere applicata anche a prescindere da sistemi di testing automatico. 45

CRC Il controllo di ridondanza ciclico o cyclic redundancy check (CRC) è un metodo per il calcolo di somme di controllo (checksum). Il nome deriva dal fatto che i dati d'uscita sono ottenuti elaborando i dati di ingresso i quali vengono fatti scorrere ciclicamente in una rete logica. Il controllo CRC è molto diffuso perché la sua implementazione binaria è semplice da realizzare, richiede conoscenze matematiche modeste per la stima degli errori e si presta bene a rilevare errori di trasmissione su linee affette da elevato rumore di fondo. Le tecniche CRC furono inventate da Wesley Peterson che pubblicò il suo primo lavoro sull'argomento nel 1961.
Utile per l'individuazione di errori casuali nella trasmissione dati (a causa di interferenze, rumore di linea, distorsione), il CRC non è invece affidabile per verificare la completa correttezza dei dati contro tentativi intenzionali di manomissione. A tal fine sono utilizzati algoritmi di hash quali MD5 e SHA1, più robusti seppur computazionalmente meno efficienti. 57

Dependency Injection Nell'ingegneria del software, la dependency injection (ing. iniezione di dipendenza) è un modello di progettazione in cui un oggetto o una funzione riceve altri oggetti o funzioni da cui dipende. Una forma di inversione del controllo, l'iniezione di dipendenze mira a separare le preoccupazioni relative alla costruzione degli oggetti e al loro utilizzo, portando a programmi liberamente accoppiati. Il pattern garantisce che un oggetto o una funzione che voglia utilizzare un determinato servizio non debba sapere come costruire tali servizi. Al contrario, il "client" ricevente (oggetto o funzione) riceve le sue dipendenze da codice esterno (un "iniettore"), di cui non è a conoscenza. 27

Design Input Per design input si intendono i requisiti fisici e prestazionali di un dispositivo utilizzati come base per la progettazione dello stesso. 43

Design Output Per design output si intendono i risultati di uno sforzo di progettazione in ciascuna fase di progettazione e al termine dello sforzo di progettazione totale. L'output di progettazione finito è la base per il master record del dispositivo. Il risultato totale della progettazione è costituito dal dispositivo, dall'imballaggio e dall'etichettatura e dal master record del dispositivo. 43

Design Pattern In informatica e specialmente nell'ambito dell'ingegneria del software, è un concetto che può essere definito "una soluzione progettuale generale ad un problema ricorrente". Si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale. È un approccio spesso efficace nel contenere o ridurre il debito tecnico.
I design pattern orientati agli oggetti tipicamente mostrano relazioni ed interazioni tra classi o oggetti, senza specificare le classi applicative finali coinvolte, risiedendo quindi nel dominio dei moduli e delle interconnessioni. Ad un livello più alto sono invece i pattern architetturali che hanno un ambito ben più ampio, descrivendo un pattern complessivo adottato dall'intero sistema, la cui implementazione logica dà vita a un framework. 1, 17, 27

Framework In informatica e specificamente nello sviluppo software, un framework (anglicismo che può essere tradotto come struttura o quadro strutturale) è un'architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) sulla quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.
Un framework è definito da un insieme di classi astratte e dalle relazioni tra esse. Istanziare un framework significa fornire un'implementazione delle classi astratte. L'insieme delle classi concrete, definite ereditando il framework, eredita le relazioni tra le classi; si ottiene in questo modo un insieme di classi concrete con un insieme di relazioni tra classi.
Alla base di un framework c'è sempre una serie di librerie di codice utilizzabili in fase di linking con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un IDE, un debugger o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito. L'utilizzo di un framework impone dunque al programmatore una precisa metodologia di sviluppo del software. 16, 17

Git Git è un software di controllo di versione (VCS, Version Control System). Sviluppato da Linus Torvalds, il creatore di Linux, Git è un software gratuito e open source, e le sue eccellenti prestazioni lo rendono il VCS più diffuso e utilizzato.
Git è un sistema distribuito utilizzabile da interfaccia a riga di comando, con funzionalità come branching e merging, aree di staging e flussi di lavoro multipli. 13, 15, 44, 56

MVC *Model-View-Controller* è un modello di architettura del software.
Esso prevede un'architettura composta da tre parti diverse: i dati (Model), la visualizzazione dei dati (View) e la gestione degli input (Controller). Questi

tre componenti sono interconnessi: il Model viene mostrato tramite la View all'utente, il quale produce gli input con cui il Controller aggiorna il Model.
Mantenerli logicamente separati però ha grandi vantaggi nella gestione del codice, infatti questo pattern favorisce lo sviluppo, il test e la manutenzione di ciascuna parte indipendentemente dall'altra. 57

Pipeline In informatica, il concetto di pipeline (ing. tubatura — composta da più elementi collegati — o condotto) viene utilizzato per indicare un insieme di componenti software collegati tra loro in cascata, in modo che il risultato prodotto da uno degli elementi (output) sia l'ingresso di quello immediatamente successivo (input). 45

Presentation Logic In italiano logica di presentazione è il luogo in un'applicazione in cui avviene quasi tutta l'interazione con l'utente finale. È la parte dell'applicazione che si occupa di ricevere gli input dall'utente finale e di presentare i risultati dell'applicazione all'utente finale. 27

Qt In informatica ed in particolare nel campo della programmazione, è una libreria multipiattaforma per lo sviluppo di programmi con interfaccia grafica (a parte per la divisione menzionata sotto 'console') tramite l'uso di widget (congegni o elementi grafici). Qt, ampiamente utilizzato nell'ambiente desktop KDE, viene sviluppato dall'azienda Qt Software (meglio conosciuta come Trolltech o Quasar Technologies) di proprietà di Digia. 17, 47

Refactoring Nell'ingegneria del software, indica una "tecnica strutturata per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno", applicata per migliorare alcune caratteristiche non funzionali del software quali la leggibilità, la manutenibilità, la riusabilità, l'estensibilità del codice nonché la riduzione della sua complessità, eventualmente attraverso l'introduzione a posteriori di design pattern. Si tratta di un elemento importante delle principali metodologie emergenti di sviluppo del software (soprattutto object-oriented), per esempio delle metodologie agili, dell'extreme programming, e del test driven development. 13

Single Responsibility Principle Nella programmazione orientata agli oggetti, il single responsibility principle (ing. principio di singola responsabilità) afferma che ogni elemento di un programma (classe, metodo, variabile) deve avere una sola responsabilità, e che tale responsabilità debba essere interamente incapsulata dall'elemento stesso. Tutti i servizi offerti dall'elemento dovrebbero essere strettamente allineati a tale responsabilità. 27, 47

SOLID In informatica, e in particolare in programmazione, l'acronimo SOLID si riferisce ai "primi cinque principi" dello sviluppo del software orientato agli oggetti descritti da Robert C. Martin in diverse pubblicazioni dei primi anni 2000. Tali principi vengono detti SOLID principles (letteralmente "principi solidi"). La parola è un acronimo che serve a ricordare tali principi (Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion), e fu coniata da Michael Feathers.
I principi SOLID sono intesi come linee guida per lo sviluppo di software leggibile, estendibile e manutenibile, in particolare nel contesto di pratiche di sviluppo agili e fondate sull'identificazione di code smell e sul refactoring.. 1

TDD In informatica, il *TDD, Test-Driven Development* (ing. sviluppo guidato dai test) è un modello di sviluppo del software che prevede che la stesura dei test automatici avvenga prima di quella del software che deve essere sottoposto a test, e che lo sviluppo del software applicativo sia orientato esclusivamente all'obiettivo di passare i test automatici precedentemente predisposti.
Più in dettaglio, il TDD prevede la ripetizione di un breve ciclo di sviluppo in tre fasi, detto "ciclo TDD". Nella prima fase (detta "fase rossa"), il programmatore scrive un test automatico per la nuova funzione da sviluppare, che deve fallire in quanto la funzione non è stata ancora realizzata. Nella seconda fase (detta "fase verde"), il programmatore sviluppa la quantità minima di codice necessaria per passare il test. Nella terza fase (detta "fase grigia" o di refactoring), il programmatore esegue il refactoring del codice per adeguarlo a determinati standard di qualità. 57

UI *User Interface* (ing. interfaccia utente), è un'interfaccia uomo-macchina, ovvero ciò che si frappone tra una macchina e un utente, consentendone l'interazione reciproca: in generale può riferirsi ad una macchina di qualsiasi natura, tuttavia l'accezione più nota è in ambito informatico con l'interazione utente-computer. 57

UML In ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato), è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 57

UX *User Experience* (ing. esperienza utente), è un termine utilizzato per definire la relazione tra una persona e un prodotto, un servizio, un sistema. Il termine presuppone un approccio olistico: cerca di comprendere tutto ciò che ruota attorno all'interazione di un utente con un'azienda, un marchio o un'istituzione. L'esperienza utente coinvolge tutti gli aspetti esperienziali, affettivi, l'attribuzione di senso e di valore collegati ad un prodotto o servizio, all'interazione con esso e quanto ad esso correlato, ma include anche le percezioni personali su aspetti quali l'utilità, la semplicità d'utilizzo e l'efficienza del sistema. 57

Widget In informatica, nell'ambito della programmazione, è un componente grafico di una interfaccia utente di un programma, che ha lo scopo di facilitare all'utente l'interazione con il programma stesso. Il termine deriva dalla contrazione dei termini "window" e "gadget". 48, 54

Bibliografia

Siti web consultati

A successful Git branching model. URL: https://nvie.com/posts/a-successful-git-branching-model/.

Flutter Complete Reference. URL: https://fluttercompletereference.com/.

Flutter Design Patterns. URL: https://flutterdesignpatterns.com/.

Flutter documentation. URL: https://docs.flutter.dev/.

Flutter Youtube Channel. URL: https://www.youtube.com/@flutterdev.

Material Design. URL: https://m3.material.io/.

pub.dev. URL: https://pub.dev/.

The ultimate guide to GetX state management in Flutter. URL: https://blog.logrocket.com/ultimate-guide-getx-state-management-flutter/.

Working with bytes in Dart. URL: https://suragch.medium.com/working-with-bytes-in-dart-6ece83455721.