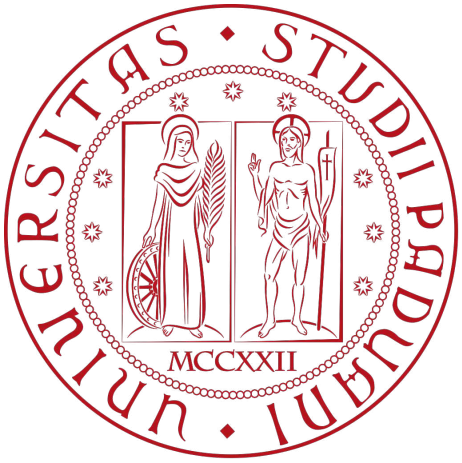
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



RACCOLTA DIGITALE DEL CONSENSO INFORMATO MEDIANTE FIRMA GRAFOMETRICA

Tesi di laurea triennale

Relatore

Prof. Tullio Vardanega

Laureando

Jacopo Angeli

Anno Accademico 2022-23

"Un uomo non muore mai se c'è qualcuno che lo ricorda."
– Ugo Foscolo

Dedicato a Giuseppe che tanto voleva partecipare ma non ha potuto.

Sommario

Durante il percorso di *stage* ho lavorato alla creazione di un'applicazione destinata a cliniche mediche private e focalizzata sulla raccolta del consenso informato. Il suo scopo è permettere al personale di porgere al paziente un *tablet*, con il quale quest'ultimo possa consultare il documento relativo all'operazione che si presta a subire, e firmarlo per acconsentire al trattamento.

L'intero processo prende il nome di acquisizione del consenso informato, che è un documento di valore legale che tutela gli attori coinvolti, pazienti e medici, da errori o incomprensioni che possono risultare in seguito a operazioni medico chirurgiche. Il procedimento rappresenta quindi uno dei passi fondamentali del percorso terapeutico di un paziente.

Esso produce però uno o più documenti cartacei che possono comportare, per cliniche specializzate in interventi chirurgici, un costo non indifferente se considerate le attività di stoccaggio, archiviazione e ricerca. Sempre più cliniche mediche valutano quindi la digitalizzazione dell'intera procedura tentate dalla riduzione dei costi di gestione, fino anche a un decimo.

Essendo il documento di natura legale è sottoposto a una normativa, per essere considerato valido nella sua versione digitale, necessita di una Firma Elettronica Avanzata. Durante lo sviluppo ho utilizzato la tecnologia della Firma Grafometrica, che associa alla rappresentazione grafica del tratto, informazioni biometriche volte a identificare univocamente la firma alla persona.

La tesi è divisa in 4 capitoli nei quali riporto una descrizione del luogo in cui ho svolto il tirocinio, i dettagli del progetto a cui ho lavorato, le modalità con cui ho l'ho portato a termine e in conclusione una valutazione retrospettiva sull'esperienza effettuata. Nel corpo del documento i termini in lingua diversa dall'italiano vengono evidenziati con lo stile corsivo, mentre la didascalia degli elementi grafici utilizzati si trova alla loro base. Nelle ultime pagine sono presenti due sezioni "Acronimi e abbreviazioni" dove sono raccolti tutti gli acronimi utilizzati, e infine "Glossario" dove invece sono presenti approfondimenti su alcuni termini tecnici presenti nel testo.

Ringraziamenti

Desidero ringraziare fortemente i miei genitori per i sacrifici commessi nel corso degli anni senza i quali il mio percorso universitario non avrebbe potuto compiersi, e per essermi stati vicini in ogni momento.

Ringrazio fortemente anche il resto della mia famiglia per avermi sempre sostenuto e per avermi sempre fornito consigli preziosi.

Un grande grazie a tutti i miei amici grazie ai quali ho apprezzato ogni momento durante i miei anni di studi.

Un ultimo sentito ringraziamento al professor Tullio Vardanega per l'attenzione dedicata durante il periodo di stage e per l'aiuto fornito durante la stesura del documento.

Agna, Dicembre 2023 Jacopo Angeli

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Azienda, ambiente, tecnologie e strumenti

1.1 Profilo aziendale

Codice Web Banking Innovation (CWBI), è un'azienda italiana con sede operativa a Padova che opera nel campo dell'*Information Comunication Technology* (ICT), fondata nel 2013. Specializzata nel settore bancario e finanziario le sue attività principali sono lo sviluppo di applicazioni web destinate a banche, assicurazioni e industrie, nonché nella fornitura di servizi di consulenza per la creazione di modelli di business e la realizzazione di software. Lavora inoltre a stretto contatto con SEC Servizi nella quale sono dislocati alcuni dipendenti e alla quale è dedicato un *team* di sviluppo interno.

1.2 Organizzazione del personale

Durante il periodo di *stage* ho avuto modo di studiare la struttura del personale aziendale, nella quale ho riconosciuto una serie di figure aziendali specifiche che ho riportato nelle due tabelle presenti in questa sezione.

All'interno della sede in cui ho lavorato nel corso del tirocinio, ho identificato solo la figura di *Chief Executive Officer* (CEO) appartenente alla parte amministrativa dell'azienda, mentre UI/UX *Designer*, Analista e Sviluppatore, appartenenti invece alla parte operativa.

Ho riportato tutte le figure sopra elencate con una breve descrizione delle loro mansioni all'interno dell'organigramma aziendale nella Tabella 1.1 presente in seguito al paragrafo.

Ruolo	Competenze
CEO	Amministratore delegato dell'azienda definisce le scelte operative dell'azienda ed è responsabile delle relazioni con i clienti nella fase di acquisizione, dell'attuazione di strategie nel processo di sviluppo.
UX/UI *Designer*	Si occupa della realizzazione di interfacce grafiche(UI) con relativa esperienza utente(UX) basandosi sia su requisiti funzionali richiesti dal cliente, sia sulla normativa vigente. Gli sviluppatori utilizzano in seguito i modelli prodotti come riferimento nelle fasi successive dello sviluppo software.
Analista	Si dedicata all'analisi e allo studio del dominio applicativo, questa figura viene ricoperta spesso dagli stessi sviluppatori in collaborazione diretta con il CEO. Occasionalmente, su discrezione di quest'ultimo, viene assunto temporaneamente un libero professionista.
Sviluppatore	Si occupa dello sviluppo dei prodotti *software* utilizzando linguaggi di programmazione e *frameworks* indicati dal cliente o scelti in fase di analisi. All'interno dell'azienda gli sviluppatori sono distinti sulla base della loro esperienza. All'assunzione si ricopre la posizione di *junior* fino a raggiungere la posizione di *senior*.

Tabella 1.1: Elenco dei ruoli aziendali e loro competenze all'interno dell'organigramma aziendale.

1.3 *Design*, sviluppo e manutenzione dei prodotti

1.3.1 Tipologia e processo di acquisizione del cliente

Le tipologie di cliente che solitamente si rivolgono a CWBI sono banche, istituti finanziari o fornitori di servizi bancari. Durante le fasi finali del mio periodo di *stage*, ho infatti avuto l'opportunità di collaborare con altri sviluppatori alla realizzazione di un prodotto destinato a specifici clienti quali CSE, Stantander Consumer Bank e Compass Banca.

Durante il tirocinio non ho avuto contatti con il reparto *marketing* dell'azienda, ma ho assistito indirettamente ad alcune delle fasi del processo di acquisizione. A cadenza circa settimanale avveniva un *meeting* preliminare tra nuovi clienti e il CEO dell'azienda, durante il quale venivano discussi in dettaglio i requisiti funzionali del cliente. In seguito ad un *meeting* del genere il CEO elaborava uno o più preventivi che venivano presentati al cliente.

Il cliente nella maggior parte dei casi si rivolge a CWBI per soluzioni già sviluppate per altri clienti, richiedendo ovviamente adattamenti sullo stile o l'introduzione di funzionalità specifiche. Una volta ricevuti i preventivi, il cliente sceglie o meno l'opzione

più adatta alle sue esigenze, avviando così il processo di sviluppo. In quest'ultima fase, la modalità di comunicazione con il cliente dipende quasi totalmente da esso. Ho osservato, infatti, come alcuni clienti utilizzino *software* dedicati, altri la posta elettronica, altri ancora preferiscano contatti telefonici o *meetings* virtuali.

1.3.2 Ciclo di vita del *software*

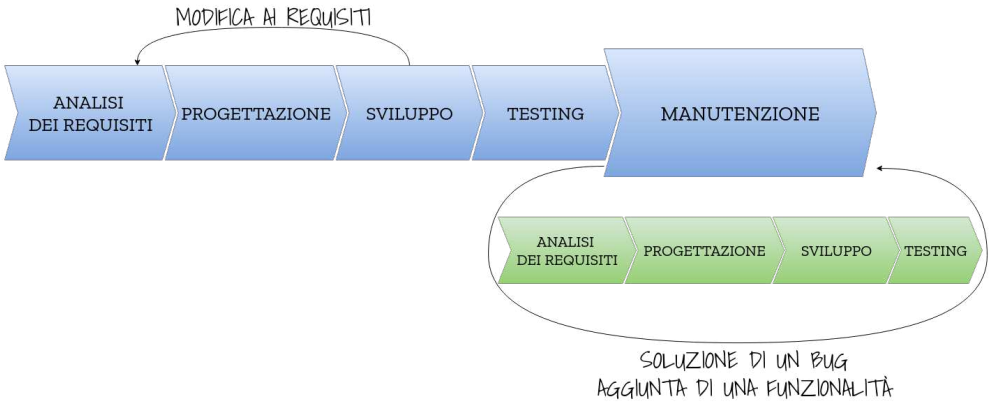


Figura 1.1: Flusso di lavoro applicato dall'azienda allo sviluppo di un *software*.

Dopo la fase di acquisizione del cliente, le sue richieste vengono tradotte in una lista di requisiti e a partire da quest'ultima viene progettato il prodotto. Come anticipato nella sezione precedente tipicamente il cliente richiede prodotti già esistenti, quindi la durata della progettazione varia sulla base della quantità di modifiche richieste.

Il processo di sviluppo utilizza un modello a cascata con possibilità di ritorno nel caso in cui i requisiti del progetto subiscano variazioni sul contenuto o sul numero, in alcuni casi dettate dal cliente stesso, in altri casi invece dovuti a imprevisti derivati dalla progettazione.

Le attività che compongono il processo di sviluppo sono svolte da uno o più *developer*, nel secondo caso solitamente i componenti del gruppo dedicato al progetto si avvicinano in postazioni adiacenti, così da rendere la comunicazione più efficiente e efficace, in entrambi i casi il CEO effettua degli incontri di allineamento su base giornaliera. Ovviamente sviluppo e *testing* vengono effettuati in ambienti diversi con dati diversificati.

Se per quanto riguarda lo sviluppo avviene interamente all'interno di CWBI, non è sempre il caso del *testing*, che in alcuni casi, compreso nel progetto dove ho lavorato nella fase finale dello *stage*, viene effettuato in larga parte dal cliente utilizzando un set di dati più ampio e più completo. I *tester* notificano il *team* di sviluppo su eventuali problemi che possono riscontrare e su eventuali modifiche che possono richiedere. In quest'ultimo caso viene tipicamente richiesta una stima composta da *effort*, ovvero ore di lavoro richieste per effettuare la modifica e *elapsed* ovvero giorni lavorativi che richiede la modifica, il cui calcolo viene effettuato dagli sviluppatori stessi dopo un'analisi dei requisiti e delle possibili misure da prendere.

1.3.3 Scheletro dei prodotti CWBI

CWBI produce tutto il suo software utilizzando come *template* un'applicazione Java che viene chiamata *BaseApp*. Il periodo di formazione di ogni tirocinante comincia infatti con il suo studio e la sua esplorazione. È un'applicazione scritta in JavaEE nella quale vengono utilizzati *Spring MVC* e *Bootstrap* per il *front-end*, mentre *Spring MVC Rest* e *Hybernate* per la componente di *back-end*.

Un'applicazione prodotta da CWBI ha un'architettura strati, così come rappresentato nella Figura 1.2 seguente.

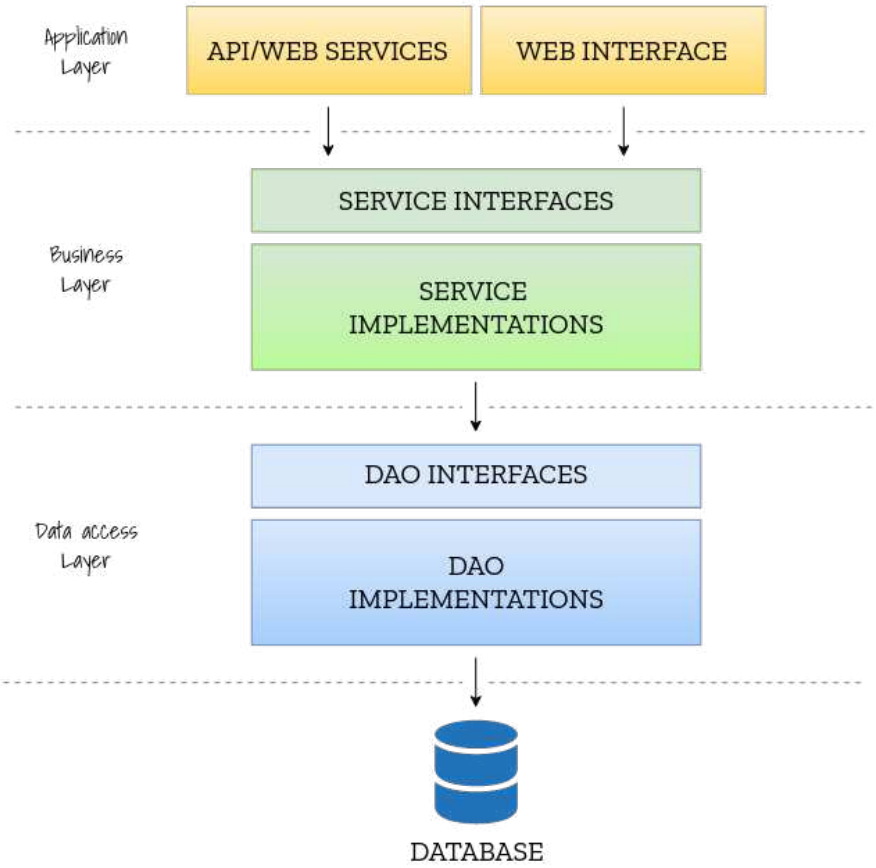


Figura 1.2: Architettura a strati dalla quale vengono costruiti i prodotti sviluppati da CWBI.

Ogni strato ha funzione e contenuto specifici che sono descritti nel dettaglio nella tabella seguente (Tabella 1.2) dove la prima colonna riporta il nome dello strato e la seconda colonna contiene la sua descrizione.

Nome	Descrizione
Application Layer	Nello strato di applicazione sono definiti e implementati i controllori che nel caso di applicazioni *web*, gestiscono interazioni dell'utente e l'interfaccia grafica, nel caso di *web services* o *API* definiscono punti di accesso e gestione delle richieste da servizi esterni.
Business Layer	Contiene tutta la logica di *business* dell'applicazione, al suo interno sono definiti e implementati oggetti dedicati alla risoluzione di compiti specifici che vengono utilizzati dai controllori definiti nello strato superiore.
Data access Layer	Al suo interno ho definito degli oggetti DAO (*Data access object*), che ho richiamato all'interno della definizione dei servizi per effettuare letture e scritture da e nel *database*. Ho creato un oggetto di questo tipo per ogni modello di dato utilizzato dall'applicazione.

Tabella 1.2: Descrizione dettagliata degli strati che compongono l'architettura di partenza dalla quale CWBI sviluppa i prodotti *software*.

1.3.4 Strumenti a supporto dei processi

1.3.4.1 VMware Workstation

Ogni macchina fisica nell'ufficio di CWBI contiene una distribuzione del sistema operativo Linux che i dipendenti utilizzano per eseguire videochiamate con clienti e fornitori, nonché per utilizzare VMware Workstation. Questo programma consente di implementare su una piattaforma con sistema operativo Linux e relativo *hardware* altre macchine virtuali con sistema operativo diverso.

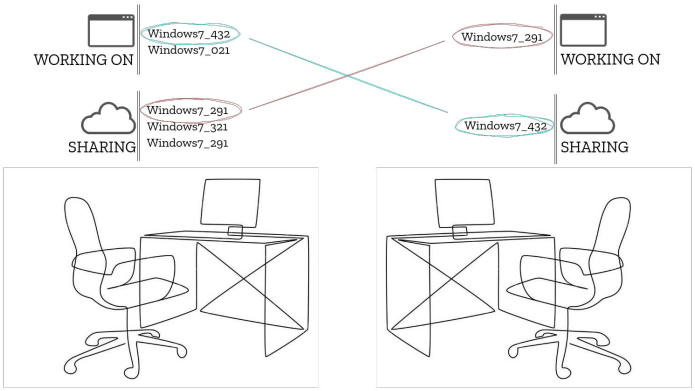


Figura 1.3: Modalità di utilizzo di VMware Workstation tra postazioni dell'ufficio

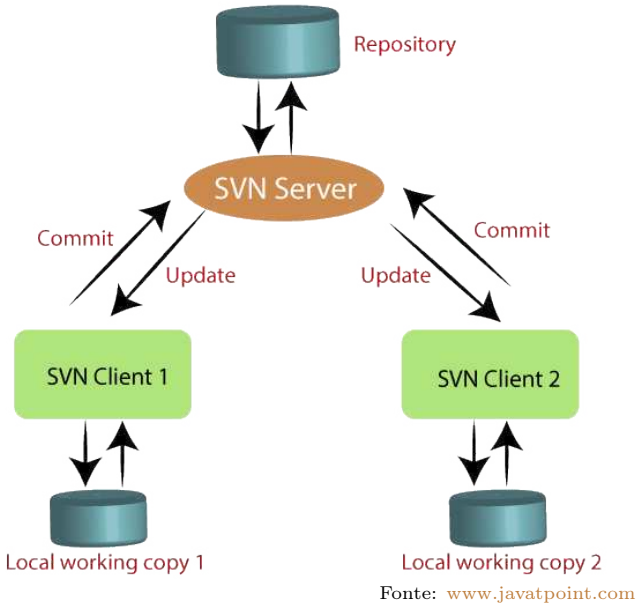
Tipicamente una macchina fisica condivide una serie di macchine virtuali e l'operatore lavora su una di queste condivise da altre macchine fisiche, collegandosi ad esse utilizzando l'indirizzo indirizzo IP della macchina fisica che le condivide, esempio nella Figura 1.3. Questo sistema garantisce un'altissima efficienza nella condivisione di informazioni tra dipendenti, e rappresenta un vantaggio di alto valore soprattutto per i membri dello stesso gruppo di lavoro.

La possibilità di risparmiare ore di lavoro e di creare macchine con configurazioni mirate al progetto, eliminando di fatto il tempo impiegato da un operatore nell'inserirsi in un gruppo di lavoro, ha spinto CWBI ad adottare questo *software* nella quasi totalità dei processi.

1.3.4.2 SVN

Per mantenere tutto il *software* prodotto, CWBI utilizza il controllo di versione, e tra i vari strumenti disponibili a tale scopo, utilizza *Subversion* (SVN), un sistema di controllo di versione centralizzato *open-source* fornito da Apache. Come schematizzato in Figura 1.4, in un ambiente SVN, c'è un *repository* centrale che contiene tutte le versioni dei file e tiene traccia delle modifiche apportate dagli utenti. Gli sviluppatori possono controllare i file dal *repository*, modificarli in locale e poi fare il *commit* delle modifiche al *repository* centrale.

A differenza dei sistemi di controllo versione distribuiti come Git, SVN richiede una connessione continua al *repository* centrale per registrare le modifiche, rendendolo più vulnerabile a interruzioni di rete e meno adatto a scenari di sviluppo decentralizzato.



Fonte: www.javatpoint.com

Figura 1.4: Schema di funzionamento di Subversion (SVN).

L'azienda ha scelto di adottare SVN rispetto a un sistema decentralizzato o ad altri simili, non per motivazioni tecnologiche particolari. Secondo il *tutor* aziendale, non si sono mai verificate situazioni in cui le funzionalità fornite da questo strumento siano

state limitanti, e inoltre, la migrazione a uno strumento diverso comporterebbe una spesa in termini di ore di lavoro.

1.3.4.3 JIRA

JIRA è una piattaforma di gestione dei progetti e tracciamento delle attività sviluppata da Atlassian. La sua caratteristiche principale, largamente utilizzata anche in CWBI, è la possibilità di creare, assegnare e monitorare dettagliatamente attività e compiti.

Fonte: www.geekwire.com

Figura 1.5: Flusso di lavoro con software JIRA.

L'azienda utilizza la piattaforma solo su accordi diretti con il cliente, l'ho utilizzata in alcuni tra i progetti a cui ho partecipato, mentre per altri progetti utilizzavano *software* diversi.

1.4 Ambiente di lavoro, tecnologie e strumenti in uso

1.4.1 Eclipse IDE

Eclipse IDE è un ambiente di sviluppo integrato molto utilizzato nel mondo IT ed è noto per la sua versatilità di utilizzo con una vasta gamma di linguaggi di programmazione. Essendo *open source* ha una grande comunità attiva di sviluppatori pronti a dare supporto e a contribuire al miglioramento del prodotto.

La scelta di CWBI è dovuta soprattutto alla presenza di un ecosistema di *plugin*, che permette un alta integrazione con le tecnologie maggiormente utilizzate nell'ambiente di sviluppo. Nonostante l'apprendimento del *software* risulta complesso, all'aumentare delle *skills* garantisce una forte riduzione delle ore di lavoro dedicate all'avvio e allo sviluppo di ogni progetto.

1.4.2 Java Enterprise Edition

Tutti i prodotti etichettati CWBI sono interamente o parzialmente realizzati mediante JavaEE. È una tipologia del linguaggio di programmazione Java che fornisce una serie di servizi aggiuntivi, tra cui la gestione delle transazioni, la sicurezza, la persistenza dei dati e l'accesso ai servizi web, per supportare le esigenze delle applicazioni aziendali. A differenza di Java SE (*Standard Edition*), che è principalmente utilizzato per lo sviluppo di applicazioni desktop e applicazioni Java *standalone*.

1.4.3 Spring MVC e Bootstrap

Per la creazione di applicazioni web CWBI combina le tecnologie di Java e Bootstrap.

Spring MVC è un *framework* di sviluppo web basato su Java che offre un'architettura *Model-View-Controller* (MVC) per la creazione di applicazioni web scalabili e robuste. Semplifica la gestione delle richieste e delle risposte HTTP, consentendo agli sviluppatori di concentrarsi sulla logica applicativa e sul design dell'interfaccia utente. In oltre anch'esso è supportato da una grandissima comunità di sviluppatori che lo rende altamente affidabile e facilmente mantenibile.

Bootstrap, d'altra parte, è un *framework front-end open source* che offre una raccolta di componenti HTML, CSS e JavaScript predefiniti per la creazione rapida e la personalizzazione di interfacce utente. È ampiamente utilizzato all'interno di CWBI dove viene sfruttato principalmente per l'estrema velocità con il quale si possono creare interfacce utente *responsive* e interattive di *default*. In'oltre l'utilizzo di un *framework* rende tutte le interfacce prodotte dall'azienda facilmente riconoscibili dando un'identità alle aplicazioni stesse.

1.4.4 Spring MVC Rest

Per quanto riguarda invece la componente di *back-end* CWBI utilizza il *framework open source* Spring MVC Rest. Un applicazione creata utilizzando quest'ultimo espone risorse e dati tramite *endpoint* HTTP, ovvero dei *link* ai quali effettuare una richiesta HTTP a seconda dell'operazione di tipo CRUD che l'applicazione riesce a gestire. Facilita inoltre la gestione dei due formati di dati utilizzati principalmente dalle applicazioni di questo tipo, XML se l'applicazione fornisce *web services*, JSON se invece l'applicazione espone un'API di tipo REST.

1.4.5 Flutter

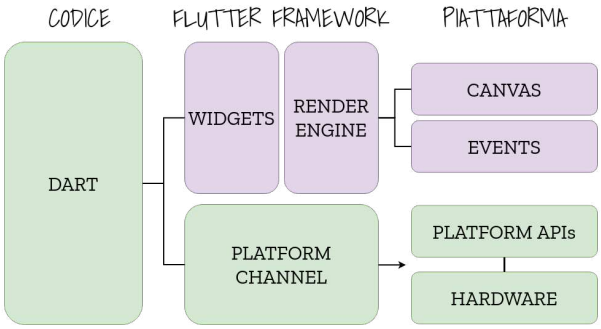


Figura 1.6: Schema di funzionamento del *framework* Flutter nel contesto di sviluppo di applicazioni multipiattaforma.

Tra i vari servizi presenti tra le proposte dell'azienda è presente anche la creazione di applicazioni *mobile*. A questo scopo viene utilizzato all'interno di CWBI il *framework* Flutter, creato da Google nel 2014 e distribuito con licenza *open source*.

Il vantaggio che ha orientato l'adozione del *framework* è la sua capacità di sviluppare applicazioni utilizzabili su più piattaforme utilizzando lo stesso codice sorgente. Ho rappresentato graficamente la modalità di sviluppo che adotta Flutter nella Figura 1.6, dove si può notare che la base di partenza, ovvero il codice, è condiviso, e in fase di compilazione il *framework* utilizza metodi e oggetti specifici delle piattaforme di destinazione per creare gli eseguibili che poi rappresentano l'applicazione finale.

1.4.6 AngularJS

In alcuni progetti di applicazioni *web*, l'adozione dello standard aziendale *Spring MVC* e di conseguenza Java non è sempre di comune accordo con il cliente. Durante il tirocinio ho lavorato per brevi periodi di tempo in prodotti il cui *front end* era scritto in JavaScript con *framework* AngularJS.

AngularJS è un *framework* JavaScript *open source* sviluppato da Google, progettato per semplificare lo sviluppo di applicazioni web dinamiche e *single-page* (SPA). Uno dei suoi punti forti è il *binding* dei dati bidirezionale e in tempo reale, ovvero permette all'interfaccia grafica l'aggiornamento dei dati in tempo reale sulla base del loro valore contenuto all'interno dell'applicazione.

È importante notare, e questo caso ne è l'esempio, che per alcuni clienti la scelta delle tecnologie è un *deal breaker* in fase di stipula di un contratto, AngularJS è una tecnologia "obsoleta", in quanto meno performante e meno utilizzata della sua versione più recente Angular. Le motivazioni per cui CWBI abbia adottato l'utilizzo di questo *framework* sono prettamente economiche.

1.5 Approccio all'innovazione

Spinto dalla curiosità ne ho parlato anche con il *tutor* aziendale, che ricopre il ruolo di CEO, e il risultato della conversazione è stato che all'interno dell'azienda non c'è una ricerca attiva nel campo tecnologico.

Alcuni dipendenti condividono all'interno di una *chat* condivisa articoli e proposte su nuove tecnologie delle quali vengono a conoscenza durante il loro tempo libero, e delle quali sarebbero interessati parlarne più approfonditamente a tutto il personale. In periodi dove il carico di lavoro lo permette, avvengono delle riunioni nelle quali i dipendenti che lo desiderano possono presentare una proposta sull'adozione di una tecnologia nuova, ma la scelta finale è comunque presa dal CEO che prende in considerazione i pareri del personale e si informa più approfonditamente sui vantaggi che possono portare.

Nei due mesi durante i quali l'azienda mi ha accolto come tirocinante, non ho assistito a riunioni di questo tipo anche se alcune proposte che ho espresso, spesso anche in presenza del *tutor* aziendale, sono sempre state ascoltate e discusse con interesse.

Capitolo 2

Visione d'insieme del periodo di *stage*

2.1 Posizione del tirocinante nell'organigramma

CWBI vede nei tirocinanti una grandissima opportunità ed è altamente coinvolta nella loro ricerca e nella loro formazione. Durante il tirocinio ho osservato infatti che il personale dell'ufficio era composto dal 50% di tirocinanti, alcuni dei quali provenienti da un percorso universitario, la restante parte divisa tra studenti di una scuola superiore di secondo grado convenzionata con l'azienda, e neodiplomati con contratto di apprendistato.

Lo scopo dell'azienda è la formazione mirata dello *stagista* sulle tecnologie utilizzate nella produzione dei loro prodotti, con il fine ultimo dell'assunzione. Per questo motivo il *tutor* assegna spesso al candidato compiti finalizzati alla pura formazione senza aspettative sul prodotto finale, viene incoraggiato ad essere autonomo ma anche alla comunicazione con il personale nel caso non riuscisse a portare a termine quanto assegnato.

2.2 Progetto di stage

2.2.1 Proposta aziendale e obiettivi da raggiungere

Lo scopo del tirocinio è stata la progettazione e la realizzazione di una applicazione per dispositivi *mobile*. Lo scopo del prodotto era la raccolta del documento "consenso informato" firmato dal paziente e l'applicativo era destinato a cliniche mediche private specializzate in interventi di chirurgia estetica.

L'atto di acquisire il "consenso informato" rappresenta uno dei passi essenziali nel percorso clinico di un paziente, in quanto questo documento possiede un valore legale significativo. Quest'ultimo è necessario a tutelare medici e pazienti coinvolti in un intervento medico di natura invasiva, al fine di prevenire errori o malintesi che potrebbero derivare dall'operazione stessa. Il documento contiene una descrizione dettagliata di tutti i potenziali esiti che potrebbero derivare dall'intervento a cui il paziente acconsente a sottoporsi. Affinché il paziente possa effettivamente dare il suo consenso per l'operazione, è necessario che firmi il documento, che a sua volta deve

essere firmato anche dal medico.

Il prodotto del processo sopra descritto consiste in documento cartaceo composto a seconda dell'intervento da una o più pagine, e considerando costi di stampa, ricerca e stoccaggio, la gestione dell'insieme della documentazione comporta un costo sempre più rilevante per le aziende. L'entità della spesa spinge sempre più imprese a digitalizzare l'intero processo affidandosi a *software house* come CWBI.

2.2.2 Aspetti normativi

Il documento di "consenso informato" ha, come anticipato in precedenza, alto valore legale. Nella sua versione digitale infatti, necessita di una firma effettuata con tecnologia di firma elettronica, della quale esistono tre versioni.

Fonte: www.aruba.it

Figura 2.1: Tipologie di firma elettronica definite dal regolamento eIDAS (*electronic IDentification Authentication and Signature*) .

Vista la natura dell'applicazione e le esigenze del cliente, la tecnologia integrata all'interno dell'applicazione è stata utilizzata la firma grafometrica.

La firma grafometrica è un particolare tipo di FEA che è in grado utilizzando dispositivi compatibili di associare ad un tratto di firma, una serie di dati biometrici del firmatario. Per essere in grado di raccogliere una firma di questo tipo un dispositivo *mobile* deve essere equipaggiato con un digitalizzatore (componente *hardware* in grado di rilavare il tocco di penna o dito e di trasformarlo in un segnale interpretabile dal dispositivo) e una penna capacitiva attiva .

I parametri biometrici che compongono una firma grafometrica sono cinque:

- il ritmo con il quale le sezioni del tratto sono state disegnate,
- la velocità di azione della firma,
- la pressione con la quale è stata applicata la firma,
- l'accelerazione con la quale le sezioni del tratto sono state disegnate,

- il movimento aereo associato ad ogni parte del tratto.

Una penna di tipo passivo, sostituisce semplicemente la funzione del dito, del quale un *display touch* anche se equipaggiato con digitalizzatore, non è in grado di registrare alcun parametro se non la posizione. Solo una penna di tipo attivo di qualità medio/alta è in grado di registrare i parametri necessari per definire una firma elettronica come grafometrica.

2.2.3 *White Label Software*

L'azienda ha colto l'occasione di produrre una soluzione personalizzabile nello stile dal cliente che la utilizza. Questa caratteristica prende il nome di *white label* o, in italiano, senza etichetta.

Oltre a fornire al cliente la funzione aggiuntiva di personalizzazione, lo sviluppo di un'applicazione di questa tipologia consente all'azienda anche di rivendere lo stesso prodotto a clienti diversi mantenendo un unico ramo per la sua manutenzione.

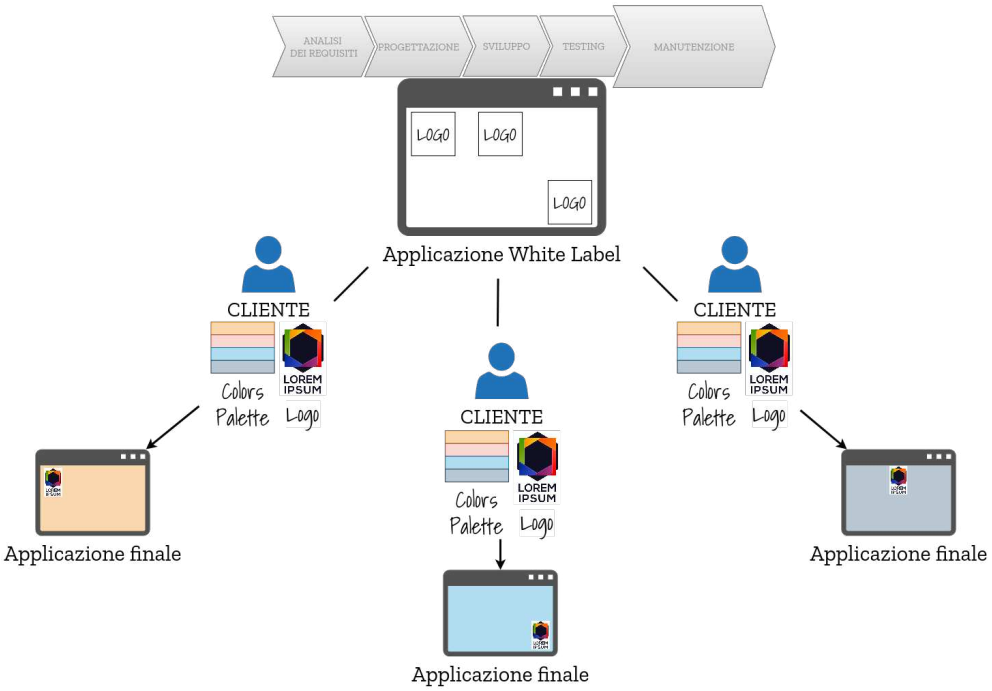


Figura 2.2: Esemplificazione del modello di *business* per un'applicazione con caratteristica *white label*.

2.2.4 Obiettivi aziendali

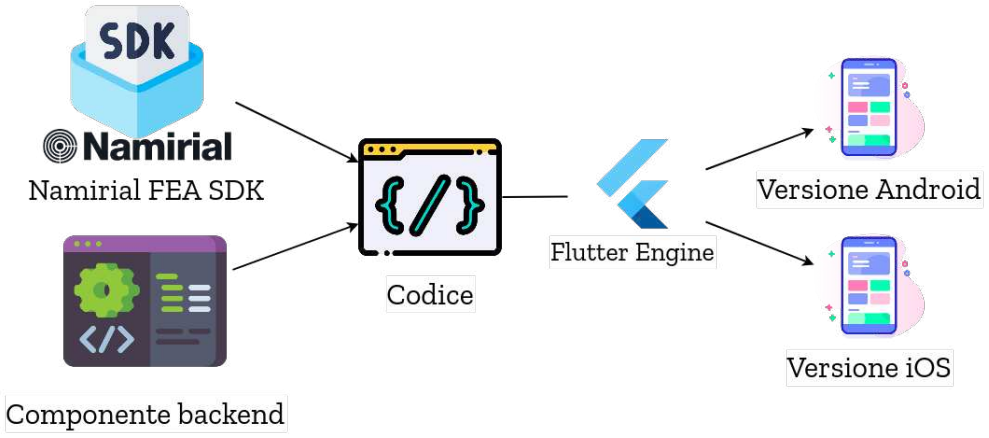
Ho raccolto gli obiettivi aziendali per la realizzazione del progetto di *stage* nella Tabella 2.1, posizionata in seguito al paragrafo, dove li ho riportati in coppia con la loro rilevanza, infatti alcuni obiettivi erano primari altri invece secondari.

Obiettivo	**Rilevanza**
Implementazione della caratteristica *white label* per la personalizzazione dell'interfaccia grafica	Primario
Implementazione della funzione di *login* di primo livello (richiesti solo *email* e *password*).	Primario
Implementazione del flusso di *download* e *upload* dei documenti da e nel *server*.	Primario
Implementazione della visualizzazione dei documenti scaricati e modificati in formato lista.	Primario
Implementazione della funzione di firma grafometrica parziale di un documento in formato PDF.	Primario
Implementazione della funzione di personalizzazione dell'ordine della lista di documenti.	Secondario
Implementazione della funzione di *auto-login* all'avvio dell'applicazione.	Secondario

Tabella 2.1: Obiettivi aziendali e loro rilevanza, posti sulla realizzazione dell'applicazione

2.3 Vincoli sulla realizzazione

2.3.1 Tecnologie e integrazione con prodotti esterni ed interni



Fonte: www.flaticon.com

Figura 2.3: Struttura dell'applicazione e rappresentazione grafica dei vincoli tecnologici rispettati durante il suo sviluppo.

Flutter

Prima dell'inizio del tirocinio, ho effettuato due incontri con il *tutor* aziendale nei quali abbiamo discusso i dettagli dell'esperienza che avrei effettuato nei mesi seguenti. Duranti gli incontri ho parlato della mia esperienza in ambito programmazione e nel momento in cui ho nominato Flutter, il tutor ha proposto il progetto sopra descritto.

Vista la sua versatilità nella piattaforma di destinazione, Flutter era già stato utilizzato per altri progetti all'interno dell'azienda, e per passione personale la mia scelta è stata immediata. Ovviamente, una volta confermato l'obiettivo dello *stage*, il *framework* è diventato un vincolo tecnologico al quale mi sono attenuto per tutta la durata del tirocinio.

Namirial FEA SDK

La raccolta digitale di una firma grafometrica richiede un grosso sforzo di progettazione e codifica. L'azienda aveva quindi già avuto contatti diretti con la società Namirial, che è l'unica azienda italiana che offre un pacchetto di sviluppo totalmente indipendente dedito all'acquisizione della tipologia di firma in questione.

Quando si parla di un pacchetto di sviluppo in ambito informatico si usa nella maggior parte dei casi la traduzione in inglese *Software Development Kit* (SDK), e si intende un insieme di strumenti che permette a un team di sviluppatori di utilizzare una funzionalità specifica in ambito *mobile*.

In particolare Namirial fornisce un SDK in grado di aprire un interfaccia grafica che permette di applicare una o più firme su un *file* pdf. Quest'ultimo deve essere adeguatamente decorato con delle etichette così da permettere all'applicazione di applicare la firma nella corretta posizione.

***Backend* già consolidato**

La componente di *backend* dalla quale l'applicazione deve recuperare i documenti pdf da firmare, e alla quale deve poi inviarli con le firme richieste era già presente e completamente funzionante al momento in cui ho iniziato il tirocinio.

Di conseguenza l'utilizzo di quest'ultima rappresentava un vincolo tecnologico al mi sono conformato, e per il quale ho dovuto documentarmi e formarmi sul *framework* Spring MVC Rest.

2.3.2 Tempistiche

Nonostante il vincolo di 300 ore posto dal piano di studi, durante gli incontri preliminari il *tutor* ha voluto assicurarmi che da parte dell'azienda non avrei avuto alcuna limitazione temporale. Ovviamente terminare la progettazione e la codifica del prodotto era un risultato auspicabile, quindi ho rispettato il vincolo imposto dall'università consegnando la prima versione del prodotto prima del termine del tirocinio.

2.4 Flusso di lavoro

2.4.1 Formazione

Le modalità di utilizzo degli strumenti di lavoro principali utilizzati all'interno dell'azienda le ho apprese dal *tutor* aziendale, che nei primi due giorni mi ha fornito una panoramica sul loro funzionamento. Per tutta la durata del tirocinio mi sono appoggiato ad alcuni colleghi per domande o approfondimenti, soprattutto per l'utilizzo di Eclipse IDE.

Per quanto riguarda il progetto invece, mi sono formato in autonomia e ho utilizzato qualsiasi fonte disponibile. Le attività di ricerca hanno coinvolto sia orario di lavoro, sia, per mia scelta, tempo libero che ho utilizzato per approfondire le mie conoscenze.

Per alcuni requisiti ho collaborato con un secondo tirocinante, con il quale ho cooperato anche alla produzione di alcuni moduli non rientranti nel progetto che mi è stato assegnato.

2.4.2 Interazione con il personale e con il *tutor* aziendale

Su base quotidiana effettuavo brevi incontri di allineamento con il *tutor* per aggiornarlo sullo stato del progetto, e per chiarimenti su dubbi incontrati durante lo sviluppo. Nonostante questo si è sempre dimostrato disponibile per chiarimenti che non potevano aspettare il termine della giornata o l'inizio di quella seguente.

Il resto del personale è stato altamente amichevole e accogliente nei miei confronti, con alcuni di essi ho anche coltivato delle amicizie che sono proseguite anche dopo il termine del tirocinio. Mi sono spesso affidato al loro aiuto sia per questioni tecniche che per aspetti generali sulle regole dell'ufficio delle quali non ero a conoscenza.

2.5 Ritorno sull'investimento dell'azienda nello *stage*

L'azienda spende molte risorse sulla ricerca e sull'acquisizione di tirocinanti, il CEO stesso ripone molta fiducia sulle nuove generazioni di programmatori e punta all'assun-

zione di personale altamente qualificato. Nonostante questo le competenze proprie dei candidati non sono considerate nel processo di avvio dello *stage* e l'obiettivo finale di ogni periodo di tirocinio è quello di individuare se il candidato sia veloce nell'apprendimento e abbia un'elevata capacità di *problem solving*.

Per quanto riguarda invece il *software* che viene prodotto, il suo valore di *business* non viene mai dato per scontato, il prodotto del lavoro dei tirocinanti viene sempre valutato da personale *senior* prima di essere inserito in produzione. Nel mio caso l'applicazione era completamente funzionante e ben strutturata, ma la versione consegnata è stata messa in pausa per progetti di maggiore priorità.

Il ritorno economico dell'azienda non è quindi mai garantito, un tirocinio può portare all'acquisizione di un candidato già avanzato nel processo di apprendimento di tecnologie e metodologie aziendali, e nel migliore dei casi se il tirocinante produce *software* con valore di *business*, l'azienda può avere anche un ritorno economico monetizzando quest'ultimo.

2.6 Selezione dell'offerta di *stage*

Sono venuto a conoscenza dell'azienda iscrivendomi e partecipando a Stage-IT, promosso da Confindustria Veneto Est in collaborazione con il Dipartimento di Matematica e Scienze Statistiche. Alcune aziende che lavorano in campo ICT partecipano all'evento dove propongono agli studenti partecipanti un progetto sul quale poi propongono uno *stage*.

Prima di partecipare all'evento ho analizzato tutte le proposte e ho identificato alcune aziende con le quali avevo l'interesse di effettuare un colloquio. Nonostante CWBI non fosse presente tra queste ho comunque scelto di presentarmi e ascoltare la loro proposta, attirato da uno dei rappresentanti aziendali presenti all'evento.

I punti essenziali che hanno inciso nella mia scelta definitiva sono i seguenti

Impatto iniziale

Durante l'evento citato ho discusso con il capo dell'azienda, che è riuscito a trasmettermi la sua grande passione per l'informatica. L'impatto iniziale è stato molto positivo, ho avuto l'impressione di parlare con una persona con grande esperienza che allo stesso tempo riusciva ad ascoltare e a comunicare in modo amichevole.

Distanza

La rilevante distanza da casa ha influito in buona parte sulla scelta, avevo bisogno di uscire dalla mia zona di *comfort* mettendomi alla prova.

Tecnologie utilizzate

L'utilizzo del *framework* Flutter è stata una decisione presa solo in seguito, quello che mi interessava maggiormente era lavorare e conoscere il linguaggio Java. È un linguaggio che non ho mai avuto molte opportunità di utilizzare ma al quale riconosco una notevole rilevanza, soprattutto nel campo *enterprise*.

Clientela

Lavorare con istituti bancari e finanziari da l'idea di lavorare un ambiente di prestigio, dove i processi sono progettati in modo pulito ed eseguiti con precisione.

Concorrenza

La progettazione di *software* che utilizzano risorse in modo concorrente mi ha sempre affascinato, il contesto bancario delle applicazione descritte nel colloquio sono maggiormente affette da questo problema dove la gestione coordinata dei dati è di alta importanza.

2.7 Obiettivi personali alla partenza

All'avvio dello *stage* mi sono posto una serie di obiettivi personali da raggiungere entro il termine dell'esperienza, e li ho raccolti nella Tabella 2.2 presente in seguito al paragrafo.

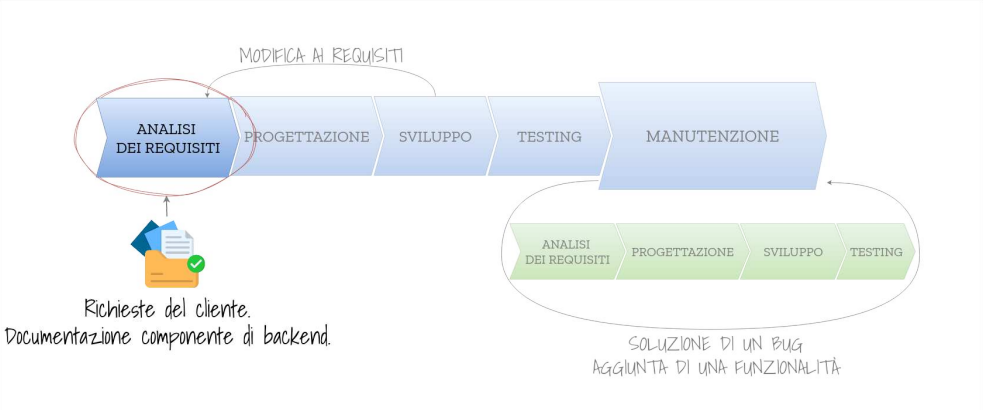
Competenze tecniche	Al termine dell'esperienza lavorativa volevo una solida base sul linguaggio Java, e più nello specifico della struttura di un'applicazione professionale. In oltre per ambizione personale volevo raggiungere un buon livello di conoscenza del *framework* Flutter.
Competenze trasversali	Riuscire ad acquisire la capacità di lavorare come parte di un *team* e raggiungere un buon livello di autogestione erano due obiettivi che speravo di conquistare al termine del tirocinio.
Ambiente aziendale	Non avendo mai lavorato in aziende ICT, uno degli obbiettivi che mi sono posto era quello di acquisire più informazioni possibili sulla modalità di esecuzione e sulla coordinazione delle attività che compongono il ciclo di vita del *software* in un'azienda consolidata nel campo in cui opera.

Tabella 2.2: Obiettivi personali posti all'avvio dell'esperienza di *stage*.

Capitolo 3

Svolgimento dello *stage*

3.1 Stato iniziale del prodotto



Fonte delle immagini utilizzate: www.flaticon.com

Figura 3.1: Stato del progetto alla partenza dell'esperienza di *stage*.

L'applicazione *mobile* alla quale ho lavorato non era ancora stata progettata, sono subentrato durante la fase di analisi dei requisiti, dove ho utilizzato il prodotto del processo di raccolta dei requisiti. Ovviamente data la presenza della componente di *backend* già sviluppata, l'analisi e la progettazione del prodotto le ho svolte utilizzando la documentazione preesistente.

3.2 Analisi del problema e identificazione dei requisiti

3.2.1 Richiesta del cliente

La richiesta del cliente era di avere un applicativo per dispositivi *mobile* sia Android che iOS, in grado di scaricare una serie di documenti senza firma dalla componente di *backoffice*, di applicare una serie di firme e in seguito di caricare la versione firmata del documento.

Il cliente stesso caricava i documenti bianchi nel *database* corredati da alcuni meta

dati, ovvero informazioni descrittive o informative associate a un determinato oggetto, che indicavano quantità, destinazione e posizione delle firme necessarie.

È stata inoltre richiesta una funzione di gestione di sessione, tutte le funzionalità dovevano essere accessibili solo mediante autenticazione, nonostante questo non è stata richiesta la funzionalità di registrazione.

3.2.2 Attori del sistema

Analizzando le richieste del cliente ho individuato tre attori principali descritti dalla tabella seguente, che include anche la tipologia degli attori. Un attore è detto interno quando è parte integrante del sistema che si sta modellando, e possono essere di diverso tipo come moduli, sottosistemi o convenzioni interne. D'altro canto un attore esterno sono entità sulle quali l'applicazione non ha controllo diretto, ma che utilizza per portare a termine alcune funzionalità che fornisce.

Attore	**Tipo**	**Descrizione**
Utente non autenticato	Interno	L'unica distinzione che l'applicazione deve gestire è tra utente con sessione attiva e utente non in sessione, al fine di prevenire l'accesso ad alcune funzionalità specifiche.
Utente autenticato	Interno	Per gli stessi motivi di cui sopra, l'applicazione è in grado di riconoscere un utente autenticato solo se ha eseguito con successo l'accesso.
Namirial FEA SDK	Esterno	Il pacchetto fornito dall'azienda Namrial fa largo utilizzo della tecnica dell'*information hiding*, ovvero nasconde l'implementazione delle funzionalità che offre, fornendo al *developer* solo un'interfaccia da poter utilizzare, viene quindi considerato come attore esterno.

Tabella 3.1: Attori individuati nella fase di analisi dei requisiti.

3.2.3 Casi d'uso individuati

Definiti gli attori del sistema ho definito una serie di casi d'uso che l'applicazione doveva coprire e per rappresentarli graficamente ho utilizzato UML, un linguaggio di modellazione grafica utilizzato nell'ambito dell'ingegneria del software.

Il seguente modello UML d'esempio descrive una parte delle funzioni dell'applicazione e al suo interno compaiono delle ellissi contenenti del testo a rappresentare i casi d'uso, e delle persone stilizzate a rappresentare gli attori che possono interagire con l'applicazione. Gli attori sono collegati ai casi d'uso mediante una linea nera solida ad indicare che possono accedere alla funzione.

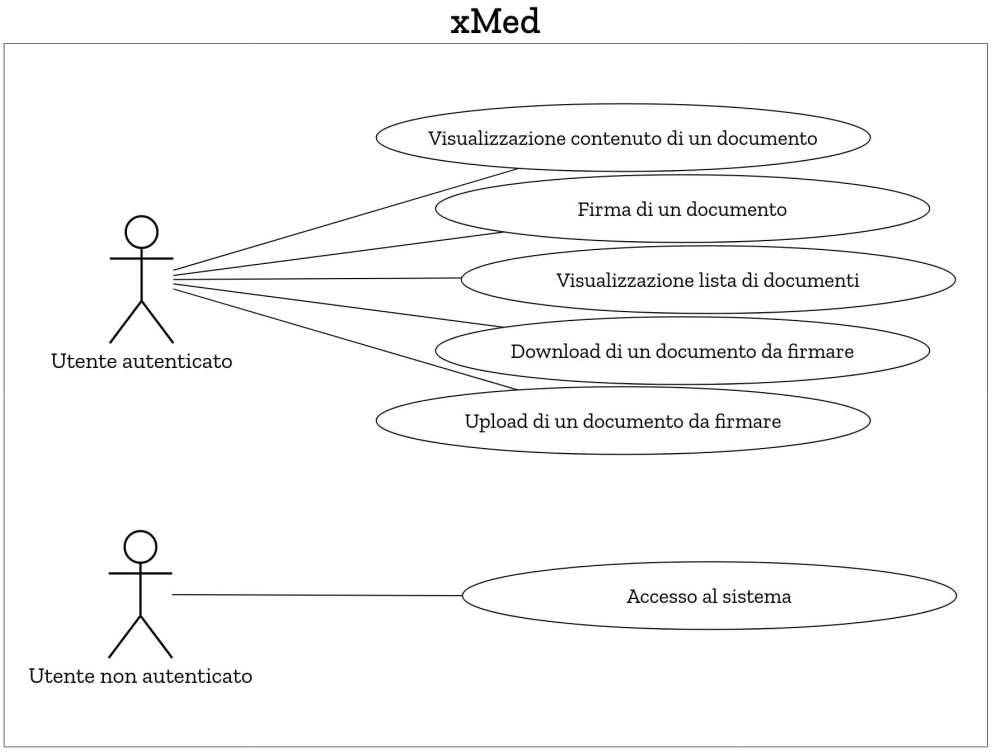


Figura 3.2: Esempio di diagramma UML utilizzato per la rappresentazione grafica dei casi d'uso individuati in fase di analisi dei requisiti.

Nella tabella 3.2, che si trova nella pagina seguente, ho elencato alcuni dei casi d'uso identificati durante l'analisi dei requisiti, selezionandoli in base alla loro importanza e al loro impatto durante lo sviluppo dell'applicazione.

Denominazione	Attori	Descrizione
Visualizzazione contenuto di un documento	Utente autenticato, Namrial FEA SDK	L'utente consulta il contenuto del documento.
Firma di un documento	Utente autenticato, Namrial FEA SDK	L'utente applica una firma al documento.
Visualizzazione lista di documenti	Utente autenticato	L'utente visualizza la lista di tutti i documenti.
Download di un documento da firmare	Utente autenticato	L'utente effettua il *download* del documento dal *server* alla memoria locale.
Upload di un documento firmato	Utente autenticato	L'utente effettua l'*upload* di un documento firmato nel *server*.
Accesso al sistema	Utente non autenticato	L'utente effettua l'accesso al sistema.

Tabella 3.2: Descrizione e attori dei casi d'uso principali individuati durante la fase di analisi dei requisiti.

3.2.4 Requisiti individuati

A partire dai casi d'uso individuati ho prodotto una lista di quattordici requisiti dell'applicazione, alcuni dei quali sono presenti nella tabella successiva al paragrafo. All'interno della quale compaiono le seguenti due colonne:

- **Rilevanza:** Classifica il requisito in base al suo valore di *business*. Per facilitare il termine del progetto, i requisiti sono stati suddivisi in Obbligatori e Desiderabili, consentendo così di definire quando una versione dell'applicazione poteva essere presentata al cliente.
- **Fonte:** Indica la provenienza del requisito.

All'interno della tabella, le voci sono separate da due righe che indicano la tipologia dei requisiti che le seguono. Se il requisito rappresenta una funzionalità che l'applicazione deve includere, viene classificato come funzionale, se altrimenti rappresenta una caratteristica o una proprietà del prodotto finale, viene classificato come non funzionale.

Descrizione	Rilevanza	Fonte
Requisiti funzionali		
L'utente deve poter accedere al sistema.	Obbligatorio	Casi d'uso
L'utente deve poter visualizzare la lista dei documenti in lavorazione.	Obbligatorio	Casi d'uso
L'utente deve essere in grado di visualizzare il contenuto di un documento.	Obbligatorio	Casi d'uso
L'utente deve poter ordinare gli elementi della lista alfabeticamente rispetto al loro nome.	Desiderabile	Casi d'uso
L'utente deve poter ordinare alfabeticamente gli elementi della lista rispetto alla loro tipologia.	Desiderabile	Casi d'uso
L'utente deve poter ordinare alfabeticamente gli elementi della lista rispetto al loro nome.	Desiderabile	Casi d'uso
Requisiti non funzionali		
L'applicazione deve poter essere utilizzata in sistema operativo Android.	Obbligatorio	Richiesta del cliente
L'applicazione deve poter essere utilizzata in sistema operativo iOS.	Obbligatorio	Richiesta del cliente
L'applicazione deve poter raccogliere una firma di tipo Firma Grafometrica	Obbligatorio	Richiesta del cliente.

Tabella 3.3: Descrizione, rilevanza e fonte di alcuni dei requisiti funzionali e non funzionali identificati durante l'analisi dei requisiti.

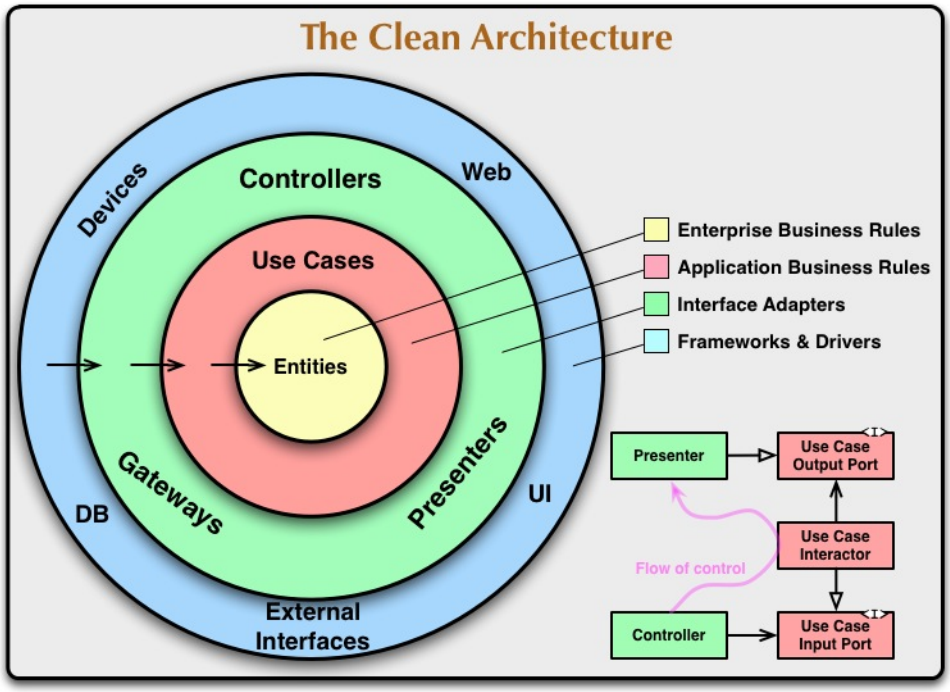
3.3 Progettazione del prodotto

3.3.1 *Clean architecture*

In seguito alla fase di analisi dei requisiti, mi sono concentrato sullo studio delle principali architetture utilizzate in ambito professionale per applicazioni scritte con *framework* Flutter. Durante la ricerca, ho considerato anche l'architettura delle applica-

zioni sviluppate in precedenza dall'azienda e ho optato infine per la *clean architecture*.

È un approccio all'organizzazione del codice e all'architettura del *software* finalizzato a semplificare il processo di manutenzione, rendendo un prodotto scalabile e estremamente testabile. Un applicazione scritta seguendo i principi definiti dalla *clean architecture* è suddivisa in strati concentrici, con responsabilità specifiche e mai sovrapposte.



Fonte: blog.cleancoder.com

Figura 3.3: Rappresentazione grafiche della struttura della *Clean Architecture* proposta da Robert C. Martin.

- ***Enterprise Business Rules***: Questo strato centrale contiene le regole di business fondamentali specifiche dell'applicazione, ad esempio viene regolato l'accesso all'applicazione mediante modello basato sulle licenze,
- ***Application Business Rules***: Traduce le regole aziendali in logiche applicative più specifiche, come le regole sul *donwload* dei documenti da firmare,
- ***Interface Adapters***: Questo strato funge da ponte tra gli strati interni ed esterni, adattando i dati e le interfacce dell'applicazione per soddisfare le esigenze degli strati superiori e inferiori, gestendo la comunicazione con interfacce utente, servizi web e database,
- ***Frameworks and Drivers***: Il più esterno dei quattro strati contiene i dettagli tecnici e le implementazioni specifiche della piattaforma, inclusi *framework*, librerie di terze parti e componenti per l'interfaccia utente, che consentono all'applicazione di interagire con il sistema operativo o l'infrastruttura sottostante.

3.3.2 Archiettura dell'applicazione

Ho applicato la *Clean Architecture* in Flutter suddividendo il progetto in quattro moduli o *features* con funzionalità specifiche:

- ***connection***: modulo dedicato alla gestione dello stato della connessione del dispositivo,
- ***documents_ managment***: modulo dedicato alla flusso di gestione dei documenti, dal *download*, all'*upload*, passando per gli *step* di firma,
- ***login***: dedicato alla gestione della sessione degli utenti,
- ***whitelabeling***: modulo per la gestione del tema dell'applicazione.

Ho suddiviso infine ogni modulo in tre strati:

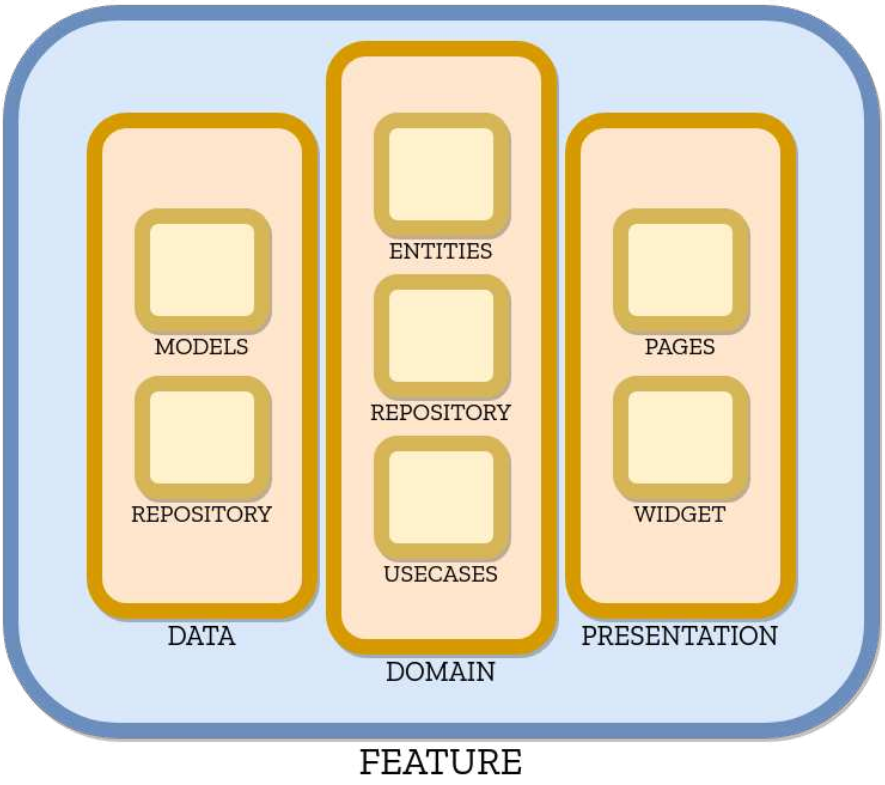


Figura 3.4: Struttura esemplificativa della divisione di file e tipologia di oggetti applicata ad ogni modulo che compone l'applicazione consegnata.

Nel *Domain layer* ho inserito le interfacce degli oggetti *repository* utilizzati per recuperare e inviare dati dal *backend*, le classi che modellano i dati utilizzati dal modulo e degli oggetti che rappresentano i casi d'uso. Nel *Data Layer* ho inserito l'implementazione delle *repositories* definite nello strato *domain* e gli oggetti che esse utilizzano per l'invio e il recupero dei dati. Infine nello strato *Presentation* ho raccolto l'implementazione dell'interfaccia grafica utilizzata dal modulo.

3.3.3 *Design patterns*

Durante lo sviluppo dell'applicazione ho applicato una serie di *design pattern*, ovvero modelli e schemi riusabili che forniscono una soluzione generale a problemi comuni nello sviluppo del software. Alcuni di essi sono indipendenti dalla tecnologia utilizzata, altri invece più specifici, per questo ho iniziato la progettazione del prodotto solo dopo aver effettuato una ricerca esaustiva sull'argomento, che ha portato all'implementazione dei *design pattern* elencati di seguito.

BLoC

Il BLoC (*Business Logic Component*) è un utilizzato comunemente nello sviluppo di applicazioni *mobile* e *web* per gestire la logica di *business* e lo stato dell'applicazione in modo separato dall'interfaccia utente.

Nel contesto di Flutter, o utilizzato il pacchetto "BLoC" fornito da Google. Ogni modulo era composto da un BLoC specifico.

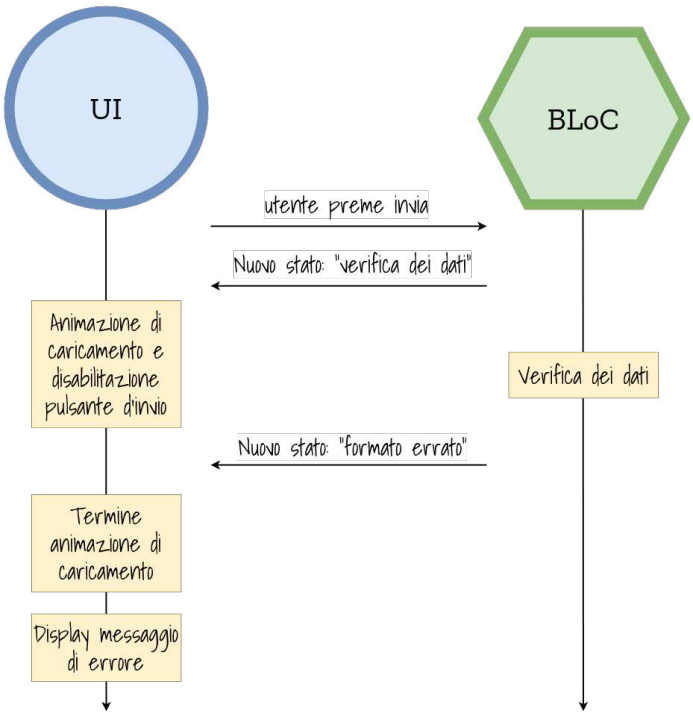


Figura 3.5: Esempio di funzionamento del BLoC *pattern* basato sull'inserimento malformato di dati generici da parte di un utente.

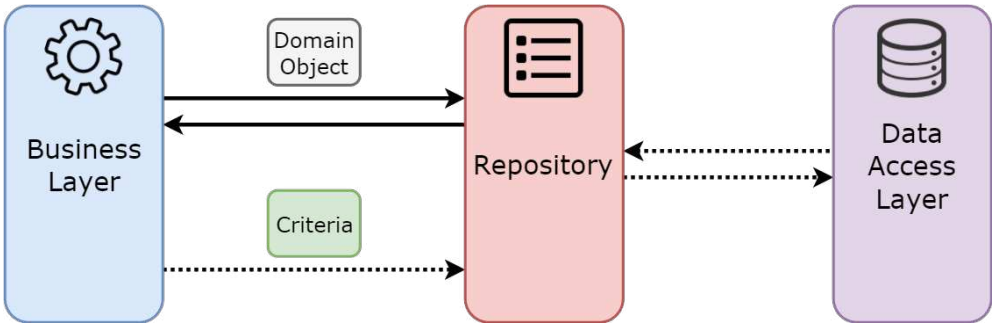
Si può considerare un BLoC come un controllore che gestisce i cambi di stato del programma, ad esempio la pressione di un pulsante da parte dell'utente corrisponde ad un evento specifico. In Figura 3.5 ho rappresentato un esempio di invio di una *form* dove uno sviluppatore potrebbe prevedere un controllo sui dati inseriti prima di effettuare l'effettivo invio dei dati. Il flusso rappresentato evita l'errore comune di non comunicare all'utente dell'esistenza di un processo in corso, portando ad una serie di complicazioni facilmente evitabili.

DTO

Il *Data Transfer Object* è utilizzato nello sviluppo software per semplificare il trasferimento di dati tra diverse parti dell'applicazione o tra sistemi distribuiti. Consiste in una classe contenente campi dati e metodi di accesso che rappresentano i dati da trasferire, e all'interno dell'applicazione è stato utilizzato per rappresentare il corpo delle chiamate e delle risposte al e dal *backend*.

Un esempio preso dall'applicazione prodotta è la richiesta di *upload* di un documento. Quest'ultima richiede una serie di dati provenienti da modelli diversi il che rende l'operazione fortemente dipendente da essi. Utilizzando un oggetto DTO apposito invece, la richiesta diventa indipendente dagli oggetti coinvolti nella sua costruzione e una possibile modifica del corpo della chiamata comporta una modifica solo all'oggetto DTO.

Respository

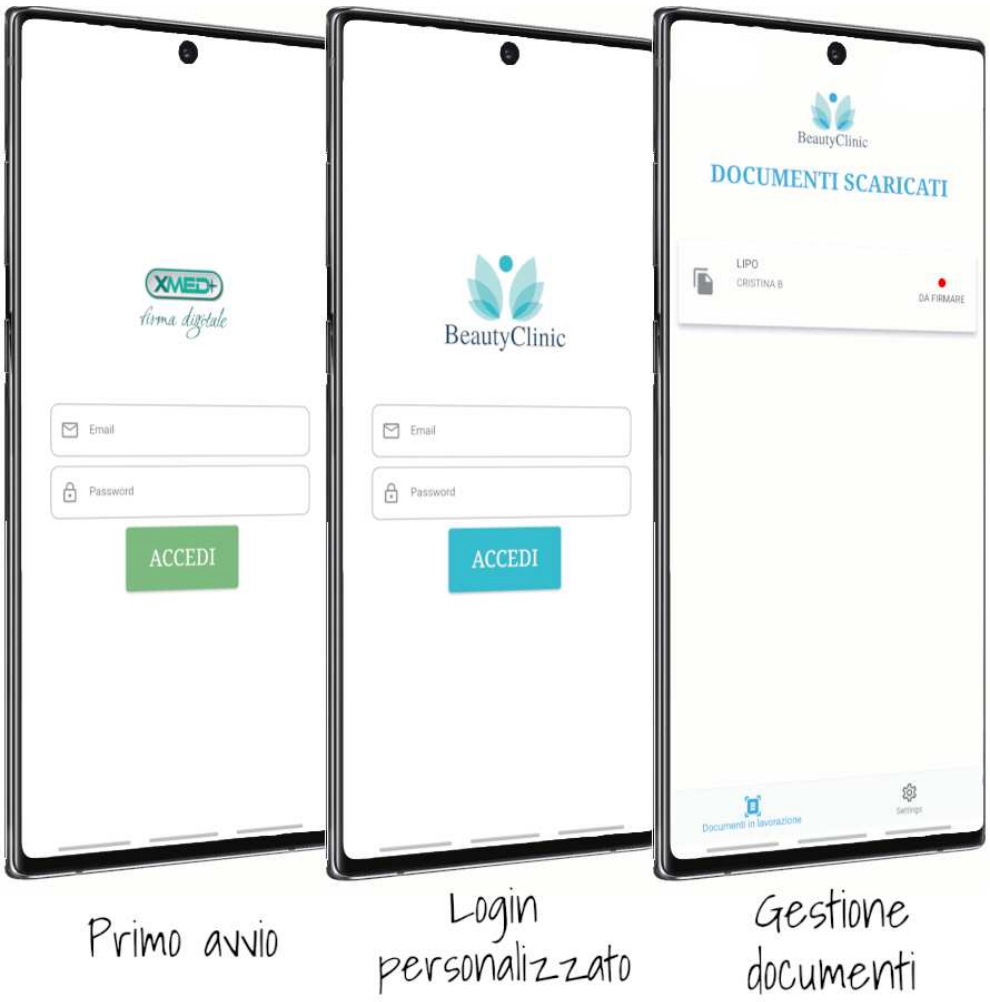


Fonte: www.codingsight.com

Figura 3.6: Struttura esemplificata del *repository pattern* utilizzata nell'applicazione prodotta, in linea con le indicazioni dell'architettura *Clean*.

Il *Repository Pattern* è utilizzato nello sviluppo *software* per separare la logica di accesso ai dati dalla logica di *business*. Consiste in un'interfaccia o una classe astratta che definisce operazioni standard di creazione, lettura, aggiornamento e cancellazione dei dati, e in una implementazione concreta ma separata dei metodi.

3.3.4 UI



Fonti utilizzate: www.pngitem.com

Figura 3.7: Raccolta di alcune interfacce che compongono l'applicazione consegnata che rappresentano la sua tipologia *white label*.

L'interfaccia delle applicazioni prodotte da CWBI è progettata da una figura del personale dedicata. Il *designer* utilizza il *software* Figma, che permette la creazione di design dinamici e supporta la collaborazione contemporanea sullo stesso progetto.

La creazione dell'interfaccia avviene in contemporanea con la progettazione dell'applicazione, e prima di avviare la fase di codifica, *designer* e *developer* si confrontano sul prodotto del primo. Si analizza se l'interfaccia contiene tutte le funzionalità progettate, e vengono posti alcuni vincoli specifici che il cliente ha espresso.

Ricevuti i *file* e le istruzioni dal *designer*, ho quindi proseguito con la codifica dell'applicazione nei termini delle indicazioni ricevute. Di seguito ho inserito alcune immagini raccolte con permesso del *tutor* durante lo sviluppo dell'applicazione, per dare l'idea dello stile dell'applicazione consegnata.

3.4 Codifica

3.4.1 Integrazione di NamirialSDK

Quando uno sviluppatore utilizza Flutter, scrive del codice comune a tutte le piattaforme, ma in alcuni casi è necessario del codice più specifico, ad esempio per applicazioni con funzionamento diverso a seconda del sistema operativo. Questo è il caso dell'applicazione sviluppata, il pacchetto di sviluppo *NamrialSDK* infatti non viene distribuito per il *framework* Flutter, ma come libreria da utilizzare progetti Kotlin e Java per quanto riguarda il sistema operativo Android, e i progetti Swift per sistema operativo iOS.

Per quanto riguarda lo sviluppo della parte nativa, assieme al pacchetto di sviluppo l'azienda fornisce previo contatto diretto e scambio di credenziali, una pagina web contente tutta la documentazione necessaria per l'integrazione dei loro prodotti, sia su sistema operativo Android, sia su iOS.

Per l'integrazione della parte nativa invece, ho dovuto formarmi in autonomia utilizzando risorse online, come documentazione ufficiali e video corsi. Il *framework* Flutter fornisce degli oggetti specifici chiamati *MethodChannel*, per chiamare e gestire delle funzioni definite in linguaggio nativo, e apprendere il modo corretto per utilizzarli non è stato banale. Quest'ultimi sono essenzialmente dei flussi di comunicazione tra codice scritto in Dart (Flutter) e codice nativo, dove entrambe le parti chiamano funzioni e inviano dati.

3.4.2 Interazione con *backend*

Ho sviluppato le interazioni con il *backend* seguendo le linee guida fornite dal *tutor* aziendale, basate su altri progetti dello stesso tipo ed è stata una delle parti più formative, in quanto ho appreso il concetto di non ripudiabilità.

La non ripudiabilità è la proprietà di un'azione o una transazione effettuata da un utente o da un sistema, di non poter essere negata o respinta successivamente. In altre parole, una volta che un'azione è eseguita o una richiesta è inviata, l'entità che l'ha effettuata non può negare le sue azioni. È un principio fondamentale in ambito informatico per molti aspetti, nel particolare:

Conformità legale

In molti settori, come quello finanziario o governativo, ci sono regolamenti e leggi che richiedono la registrazione e la conservazione di determinate azioni e comunicazioni.

Audit e tracciabilità

La capacità di tracciare le azioni degli utenti o dei sistemi è fondamentale per rilevare abusi, frodi o comportamenti non autorizzati. La non ripudiabilità risolve il problema consentendo di tenere traccia delle azioni e di utilizzare queste informazioni in un eventuale processo.

Contratti digitali

In contesti come i contratti digitali o le firme digitali, la non ripudiabilità è fondamentale per garantire che le parti coinvolte non possano negare la validità o l'autenticità di un documento o di una transazione firmata digitalmente.

Ogni richiesta ricevuta dal *backend* viene quindi prima verificata e poi eventualmente gestita. Ho creato quindi un un oggetto specifico per l'invio delle richieste di tipo HTTP, che contengono nel loro campo *header* due valori chiamati *signature* e *thumbprint*. Il primo è generato dal processo di firma del contenuto della richiesta mediante una chiave privata utilizzato per validare il contenuto della richiesta, il secondo invece contiene un certificato che identifica l'applicazione.

Tutte gli *end-point* ovvero i punti ai quali effettuare le richieste erano ben documentati, grazie allo strumento Swagger utilizzato dall'azienda nei loro prodotti. Comunque per informazioni aggiuntive avevo il supporto sia del *tutor* aziendale che dei dipendenti già esperti nella produzioni di applicazioni del genere.

3.5 Risultati raggiunti

3.5.1 Copertura dei requisiti

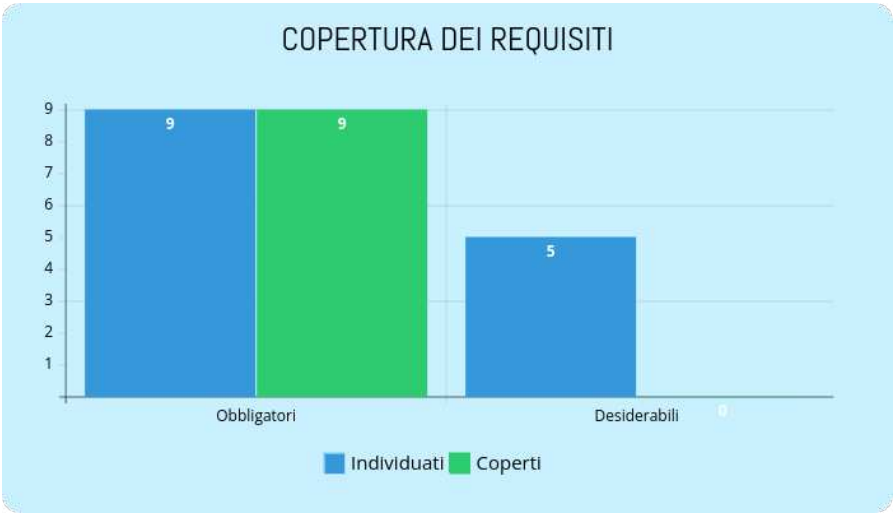


Figura 3.8: Grafico a barre sulla copertura dei requisiti individuati.

La versione dell'applicazione che ho consegnato al termine del tirocinio rispettava il 100% dei requisiti obbligatori, i requisiti opzionali invece non sono riuscito a portarli a termine, a causa della quantità di tempo che ho dovuto trascorrere in attività formative. Nonostante questo l'architettura del prodotto che ho consegnato facilità altamente lo sviluppo dei requisiti mancanti e la documentazione che ho prodotto permette facilmente l'integrazione di un nuovo tirocinante o sviluppatore.

3.5.2 Documentazione e natura del codice

Uno dei principi fondamentali che ho applicato durante lo sviluppo del prodotto è stato quello di renderlo completamente documentato. Sulla base di esperienze precedenti e di altri prodotti CWBI, ho ritenuto più corretto accompagnare tutto il codice con la sua corretta documentazione, sotto forma di commenti della seguente tipologia:

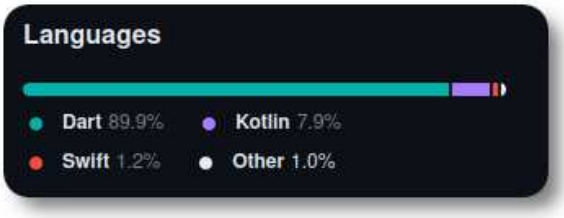
```
/// Configura e prepara un'istanza di [HttpCustomClient] per effettuare
/// richieste HTTP.
///
/// Questo metodo accetta una [Map] di chiavi [String] e valori dinamici
/// che rappresentano i parametri del corpo della richiesta. Inizializza
/// un client [Dio] con i timeout di connessione e ricezione specificati,
/// l'URL di base e gli header personalizzati.
///
/// Restituisce un [Future] che si completa con l'istanza di
/// [HttpCustomClient] configurata per permettere le chiamate a catena.
/// Solleva un'eccezione se si verifica un problema durante la
/// generazione della firma della richiesta o dell'impronta digitale
/// nell'ambito del processo di creazione dell'header.
```

Figura 3.9: Esempio di commenti con i quali ho documentato il codice dell'applicazione.

Dove i commenti descrivono una funzione e sono scritti in un formato specifico che permette, tramite il comando `dart doc` fornito di base da Dart, la generazione automatica di documentazione del codice.

Oltre alla documentazione relativa al codice, durante tutto il progetto ho prodotto e mantenuto della documentazione aggiuntiva, quale SRS (*Software Requirements Specification*) dove ho definito le funzionalità che l'applicazione doveva fornire alla consegna e il documento SDS (*System Design Specification*) dove invece ho descritto e giustificato tutti i suoi dettagli implementativi.

Al termine del progetto l'applicazione consisteva di circa diecimila linee di codice scritte per il 90% in linguaggio Dart, per l'8% in linguaggio Kotlin, il restante 2% in linguaggio Swift e altri linguaggi per file di configurazione.



Fonte: www.github.com

Figura 3.10: Distribuzione dei linguaggi di programmazione utilizzati durante lo sviluppo dell'applicazione.

Capitolo 4

Retrospettiva sull'esperienza

4.1 Raggiungimento degli obiettivi personali

Nel Capitolo 2 ho presentato alcuni obiettivi personali che mi ero posto all'inizio dell'esperienza di stage, e in questa sezione tratto di come sono riuscito a raggiungerli nel corso dei due mesi passati all'interno dell'azienda.

Competenze tecniche

Durante il mio percorso ho acquisito delle basi sul linguaggio di programmazione Java, in quanto ho avuto l'opportunità di lavorare in modo indiretto con alcune applicazioni sviluppate all'interno dell'azienda. Ho potuto studiare le *best practices* riguardanti l'utilizzo del *framework* Spring e del linguaggio Java. Inoltre sono riuscito ad approfondire le mie competenze nello sviluppo con Flutter, una tecnologia alla quale ero molto interessato anche prima del tirocinio personale.

Competenze trasversali

Durante il mio percorso ho lavorato molto sulla capacità di risoluzione dei problemi, riuscendo a portare a termine il progetto in autonomia. Ho lavorato sulla capacità di identificare la figura competente a cui rivolgermi alla presenza di un problema specifico, e alla mia abilità di comunicare in modo chiaro e conciso grazie alla preziosa guida di alcuni colleghi che mi hanno offerto istruzioni e *feedback* professionali.

Ambiente aziendale

Ho passato circa due mesi all'interno dell'azienda e ho partecipare attivamente alle attività dell'ufficio, adattandomi nelle situazioni alle quali non avevo mai avuto occasione di trovarmi. Ho partecipato anche ad alcune video chiamate conoscitive con alcune aziende riguardo progetti in fase di partenza osservando la modalità di interazione tra due rappresentanti aziendali. Infine ho dovuto apprendere un linguaggio tecnico utilizzato nel contesto delle applicazioni sulle quali ho lavorato.

4.2 Raggiungimento degli obiettivi aziendali

Nella Sezione 2.2.4 ho riportato una serie di obiettivi aziendali in riferimento all'applicazione da sviluppare durante il periodo di *stage*, nel seguito riporto gli stessi obiettivi con il loro stato al termine del tirocinio, all'interno della Tabella 4.1.

Obiettivo	**Soddisfacimento**
Implementazione della caratteristica *white label* per la personalizzazione dell'interfaccia grafica	Soddisfatto
Implementazione della funzione di *login* di primo livello (richiesti solo *email* e *password*).	Soddisfatto
Implementazione del flusso di *download* e *upload* dei documenti da e nel *server*.	Soddisfatto
Implementazione della visualizzazione dei documenti scaricati e modificati in formato lista.	Soddisfatto
Implementazione della funzione di firma grafometrica parziale di un documento in formato PDF.	Soddisfatto
Implementazione della funzione di personalizzazione dell'ordine della lista di documenti.	Non soddisfatto
Implementazione della funzione di *auto-login* all'avvio dell'applicazione.	Soddisfatto

Tabella 4.1: Soddisfacimento degli obiettivi aziendali al termine dell'esperienza di *stage*.

4.3 Valutazione dell'esperienza

Nel corso dei mesi che ho passato all'interno dell'azienda ho osservato alcuni aspetti positivi che ho apprezzato molto:

Esperienza

L'abilità dei miei colleghi di sviluppare e progettare soluzioni complesse su misura per i clienti che scelgono di affidarsi ai loro servizi mi ha dato l'occasione di confrontare le mie competenze con le loro. Ho potuto constatare personalmente come il *team* sia in grado di affrontare sfide complesse con creatività, competenza e approccio strategico.

Livello dei prodotti

Ogni prodotto *software* sul quale ho lavorato all'interno di CWBI ha origine da un nucleo centrale che è stato sviluppato e costantemente migliorato nel corso degli anni. Questo nucleo ha attraversato un lungo e rigoroso processo di test, e sono positivo di averlo potuto osservare e studiare.

Flessibilità

L'ambiente di lavoro nel quale ho lavorato è dinamico e offre regole flessibili, consentendo ai dipendenti di gestire il proprio tempo in modo più personale. L'approccio adattabile facilita a mio parere il raggiungimento degli obiettivi aziendali, e contribuisce in parte a creare un ambiente stimolante e motivante.

Ho anche osservato però alcune pratiche non in linea con quanto studiato durante il percorso universitario, che ho provato a seguire in autonomia durante lo sviluppo del progetto al quale ho lavorato. Nello specifico:

Automazione

I prodotti a cui ho avuto l'opportunità di lavorare erano altamente complessi e, in uno in particolare, l'intero processo di compilazione richiedeva diversi minuti, fino anche a cinque. Anche l'avvio della compilazione aveva circa la stessa durata, ed era composto da operazioni meccaniche che venivano effettuate esclusivamente a mano. Durante il percorso di studi ho avuto modo di studiare alcuni strumenti *software* dedicati alla soluzione di questo problema, e ho sperimentato quanto siano utili in casi come questo.

Documentazione

Ho notato che la produzione della documentazione è un'attività alla quale non viene dedicato lo stesso sforzo che viene invece dedicato allo sviluppo, nei due progetti a cui ho lavorato ad esempio era totalmente assente. Da quanto studiato durante il mio percorso accademico, se risulta vantaggioso dal punto di vista dello sviluppo non produrre documentazione adeguata, durante la manutenzione può invece rivelarsi un grande svantaggio, soprattutto in progetti complessi come in quelli ai quali ho lavorato.

4.4 Studio e lavoro: confronto sulle competenze

Nel contesto lavorativo, ho avuto l'occasione di confrontare le mie capacità con quelle di alcuni colleghi, e ho notato fin da subito una serie di differenze nel nostro approccio alle diverse attività che compongono la giornata lavorativa. E la più profonda differenza l'ho notata nella ricerca di una soluzione ad un problema. Quando personalmente cercavo una soluzione utilizzando le nozioni di teoria che ho appreso durante gli studi, i miei colleghi più esperti avevano quasi sempre già affrontato il problema e utilizzavano soluzioni utilizzate in precedenza rivelatesi corrette, rendendoli di conseguenza molto più veloci nella soluzione del problema.

In conclusione, tra quelle che sono le competenze sviluppate durante gli studi e quelle richieste dal tirocinio, e più in generale suppongo dal mondo del lavoro, c'è un grosso

divario. Questo non significa che reputo il percorso di studi inutile, neanche se finalizzato all'ingresso in un ambiente lavorativo. Piuttosto ritengo che oltre che a delle conoscenze tecniche alle quali non avrei avuto accesso altrimenti, la frequenza del Corso di Laurea Informatica mi abbia insegnato un approccio diretto, efficace ed efficiente all'apprendimento di nuove tecnologie, rendendo quindi prezioso il tempo passato a frequentare le lezioni.

Acronimi e Abbreviazioni

SDK *Software Development Kit*. 15

CEO *Chief Executive Officer*. 1–3, 10, 16

CRUD *Create, Read, Update o Delete*. 8

CWBI *Codice Web Banking Innovation*. 1–9, 11, 12, 17, 28, 31, 34

FEA Firma Elettronica Avanzata. 12, 20, 22

ICT *Information Comunication Technology*. 1, 17, 18

MVC *Model View Controller*. 4, 8, 9

pdf *Portable Document Format*. 15, 16

SVN *Subversion*. 6

UI *User Interface*. 1, 2

UX *User Experience*. 1, 2

Glossario

design pattern Un *design pattern* (modello di progettazione) è una soluzione generale e riusabile a un problema comune nell'ambito dello sviluppo del *software*. Questi modelli sono schemi o strutture concettuali che possono essere applicati per risolvere situazioni ricorrenti durante la progettazione e l'implementazione del *software*.. 26

framework È un insieme strutturato di librerie, componenti e regole di sviluppo che fornisce un'infrastruttura comune per la creazione di *software* o applicazioni. 8, 9, 15–17, 23, 24, 29, 32

information hiding In ambito di programmazione, è un principio di progettazione orientato agli oggetti che promuove l'incapsulamento delle informazioni all'interno di classi e oggetti. Questo significa che i dettagli interni di un oggetto (come variabili e metodi privati) sono nascosti all'esterno del proprio ambito, consentendo di limitare l'accesso e di proteggere le informazioni sensibili. Viene utilizzato sopratutto nella fornitura di servizi web su licenza.. 20

API Acronimo di *Application Programming Interface* (ing. interfaccia di programmazione di un'applicazione), indica un insieme di regole e protocolli che consentono a diversi programmi *software* di comunicare tra loro, permettendo loro di scambiare dati e funzionalità in modo standardizzato e efficiente. 5, 8

CSS *Cascading Style Sheets* è un linguaggio utilizzato per definire lo stile e la formattazione delle pagine *web*. Con CSS, è possibile controllare l'aspetto di testo, immagini, *layout* e altri elementi HTML. Mediante la definizione di proprietà e relativi valori vengono applicati alle pagine stili specifici. 8

DAO Un *Data Access Object* è un *pattern* di progettazione che separa la logica di accesso ai dati da altre parti dell'applicazione. Fornisce un'interfaccia per accedere ai dati in modo indipendente dalla loro archiviazione sottostante, come un database. 5

digitalizzatore È una componente *hardware* che, posizionato sopra o sotto il pannello LCD, permette al dispositivo che la monta di riconoscere la posizione della penna o delle dita sullo schermo, abilitando la possibilità di acquisire un segnale di *input* utilizzando il tatto. Ne esistono di diverse tecnologie: quelle più semplici permettono solamente di riconoscere la posizione della penna o di un qualunque oggetto, mentre quelle più complesse oltre alla posizione forniscono una serie di informazioni aggiuntive. 12, 13

HTML *HyperText Markup Language* un linguaggio di *markup* utilizzato per creare pagine *web*. Consiste in elementi chiamati *tag* racchiusi tra parentesi angolari che definiscono la struttura e il contenuto della pagina. Gli elementi possono avere attributi per fornire informazioni aggiuntive. HTML è spesso combinato con CSS per lo stile e JavaScript per l'interattività delle pagine web. 8

HTTP *Hypertext Transfer Protocol* è un protocollo di comunicazione utilizzato su Internet per il trasferimento di dati tra *client* (come un browser web) e *server* web. HTTP opera attraverso richieste e risposte, con il *client* che invia richieste al *server* per ottenere risorse e il server che risponde con dati e informazioni di stato. 8, 30

indirizzo IP Un *Internet Protocol Address* è una serie numerica univoca assegnata a un dispositivo connesso a una rete, consentendo il suo identificativo e la comunicazione con altri dispositivi su Internet. 6

meta dati I metadati sono informazioni descrittive o informative associate a un determinato oggetto, documento o risorsa, che forniscono dettagli sul suo contenuto, origine, struttura o altri aspetti rilevanti. Sono utilizzati per organizzare, categorizzare e recuperare dati in modo efficiente.. 19

SDK Abbreviazione di *Software Development Kit*, comprendono una varietà di risorse come librerie, documentazione, campioni di codice, procedure e istruzioni per l'implementazione, tutti a disposizione degli sviluppatori per l'integrazione nelle proprie applicazioni. Questi kit sono progettati per essere utilizzati su piattaforme o linguaggi di programmazione specifici.. 15, 20, 22, 36

UML Acronimo di *Unified Modeling Language* (Linguaggio di Modellazione Unificato), è un linguaggio di modellazione grafica e un insieme di convenzioni standardizzato utilizzato nell'ambito dell'ingegneria del *software* per descrivere, progettare e documentare sistemi *software*. È uno strumento chiave per la comunicazione e la comprensione tra i membri del *team* di sviluppo, consentendo loro di rappresentare in modo chiaro e conciso i vari aspetti di un sistema *software*. 21

XML *eXtensible Markup Language* è un linguaggio di *markup* che viene utilizzato per strutturare e organizzare dati in un formato leggibile sia da esseri umani sia da *computer*. È ampiamente utilizzato per lo scambio di dati tra applicazioni diverse su Internet ed è noto per la sua flessibilità e adattabilità. 8