Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Integrazione di Biometria Grafometrica in Applicazioni Mediche su Tablet

Tesi di laurea triennale

Relatore

Prof. Tullio Vardanega

Laureando

Alberto Angeloni

Anno Accademico 2022-2023

Sommario

La presente tesi si concentra sull'esperienza di stage dello studente Angeloni Alberto presso *Codice Web Banking Innovation* (CWBI), una *software house* specializzata nella progettazione e commercializzazione di applicazioni e servizi per il settore bancario. L'obiettivo durante il periodo di tirocinio è stato lo sviluppo di una componente *software* finalizzata alla gestione della registrazione di dispositivi mobile compatibili con firma grafometrica ad un servizio esterno all'azienda. Tale componente è stata integrata in un'applicazione mobile in un contesto che mirava a supportare gli studi medici nella raccolta digitale di firme per i documenti relativi al consenso informato dei pazienti.

L'integrazione della componente nell'applicazione mobile ha richiesto un modulo software in linguaggio Dart in grado di gestire in modo efficiente il ciclo di vita delle licenze, dalla loro concessione iniziale fino alla loro scadenza. Ciò implicava la necessità di sviluppare funzionalità per l'assegnazione delle licenze ai dispositivi, il monitoraggio dello stato di validità delle licenze e la gestione delle rinnovazioni annuali.

Realizzare, con mezzi informatici, l'automazione delle procedure e la digitalizzazione dei dati di tale processo offre numerosi vantaggi, tra cui un notevole aumento dell'efficienza nella raccolta e nell'archiviazione delle firme grafometriche, una maggiore sicurezza dei dati dei pazienti, una riduzione dei costi di archiviazione dei documenti cartacei e un accesso semplificato alle informazioni da parte dei professionisti medici, contribuendo complessivamente a migliorare l'efficacia e la convenienza del servizio.

Questa tesi offre un'analisi dettagliata delle soluzioni tecniche implementate e delle competenze acquisite nel campo dello sviluppo di applicazioni *web* e mobile apprese. La tesi è composta di quattro capitoli, dove descrivo,la realtà aziendale dove ho svolto l'esperienza di tirocinio, il progetto di *stage* nei dettagli e l'insieme di metodologie che ho adottato per portare a termine i lavori. Il quarto capitolo invece tratta la valutazione retrospettiva dell'esperienza. Ho adottato alcune convenzioni stilistiche all'interno della tesi, evidenziando in corsivo tutti i termini in lingua diversa dall'italiano e apponendo una didascalia descrittiva del contenuto alla base di ogni immagine e ogni tabella, riportando la fonte, nel caso in cui i dati provengano da fonti esterne al sottoscritto. Ho dedicato una sezione "Glossario" che contiene degli approfondimenti che ho reputato come necessari al lettore per comprendere nei dettagli quanto descritto dalla tesi, mentre la sezione "Acronimi e abbreviazioni" è totalmente dedicata alla raccolta e spiegazione di tutti gli acronimi che ho utilizzato nel testo.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Tullio Vardanega, relatore della mia tesi, per le conoscenze trasmesse durante il corso di studi, l'aiuto e il sostegno fornitomi durante la redazione della tesi. Ringrazio i miei genitori per avermi sostenuto durante il periodo di studi, non ce l'avrei mai fatta se non ci fossero stati loro in alcuni momenti. Ringrazio mio fratello per avermi sempre mostrato la via, per avermi dato sempre un esempio a cui mirare e con cui competere, mi ha sempre dimostrato che impegnandosi tutto sarebbe stato possibile. Ringrazio anche un'altra persona, la mia fidanzata Anna, grazie alla quale ho riscoperto me stesso, le mie passioni e con queste, il mio amore per l'informatica. Ha sempre creduto in me, amandomi e rassicurandomi ogni volta che mi presentavo dubbioso sul mio futuro, senza di lei infatti avrei rinunciato alla continuazione del mio percorso di studi, per immettermi subito nel mondo del lavoro, e ad oggi, posso dire di aver preso la scelta giusta per me, dedico anche a lei questo mio traguardo personale. Ringrazio mio nonno per aver dato delle solide radici ai miei valori, anche se non fisicamente, so che sei con me, spero di renderti fiero. Ringrazio i miei tre migliori amici Davide Luca e Filippo, che hanno sempre visto il meglio di me, dandomi in cambio il meglio di loro, mi hanno dato una "seconda famiglia" dove potermi sentire al sicuro, spero che nelle loro rispettive strade, riescano a sentirsi realizzati e soddisfatti. Ringrazio infine me stesso, in quanto, non mi sono mai permesso di mollare, ho sempre creduto ciecamente nella scelta fatta anni fa, e sono contento di aver raggiunto questo traguardo con le mie forze, questo percorso ha inciso indelebilmente la mia crescita personale e professionale.

Galzignano Terme, Dicembre 2022 Alberto Angeloni

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Realtà aziendale

1.1 Profilo aziendale

1.1.1 Presentazione

Codice Web Banking Innovation (CWBI) è un'azienda italiana che opera nel mercato dell'*Information Communication Technology*. Vene fondata nel 2013, con sede a Padova ed è specializzata nel settore bancario, creando principalmente applicazioni *web* e *mobile* per banche, enti assicurativi e industrie. Oltre alla produzione *software* l'azienda fornisce servizi di consulenza a terzi seguendo una procedura divenuta la loro norma, che va inizialmente a studiare il modello di *business* del cliente, per poi andare a ridefinire i processi organizzativi e progettando soluzioni *software*, al fine di implementare concretamente la strategia risolutiva prodotta dopo l'analisi iniziale. L'azienda collabora a stretto contatto con un'altra realtà: SEC Servizi, altro consorzio per la fornitura di servizi informativi in *outsourcing*, dove sono dislocati alcuni dipendenti che comunicano direttamente ad un sottoinsieme del personale CWBI che opera come un *team* di sviluppo indipendente.

1.1.2 Organizzazione del personale

Durante la mia esperienza, ho esplorato l'organizzazione interna di CWBI e identificato diverse figure professionali. Ognuno infatti andava a ricoprire mansioni diverse a seconda del progetto nel quale era coinvolto, poteva infatti, fare lo sviluppatore per un particolare prodotto mentre per un altro esserne l'analista. Trovo come le scelte fatte dal CEO sulla composizione dei gruppi di lavoro, tenessero sempre conto delle caratteristiche personali dei componenti. Tali scelte sono rese possibili dall'esperienza del *tutor* e permettono di migliorare l'efficienza e la produttività del *team* combinando il personale a disposizione correttamente. La tabella 1.1 esprime il ruolo e le mansioni di ogni figura professionale.

Ruolo	Mansioni
CEO	Il'CEO è l'amministratore delegato dell'azienda. Egli definisce le scelte operative, gestisce le relazioni con i clienti durante l'acquisizione e implementa strategie nello sviluppo aziendale.
UI/UX *Designer*	Questo professionista specializzato ottimizza l'aspetto visivo e l'usabilità delle applicazioni informatiche, assicurandosi che siano attraenti, intuitive e soddisfacenti per gli utenti. Inoltre, definisce l'esperienza utente complessiva.
Analista	Questo esperto valuta e analizza i progetti, assicurando che rispondano alle esigenze degli utenti e agli obiettivi aziendali. All'interno di CWBI, questa figura è sempre ricoperta dall'CEO o da sviluppatori.
Sviluppatore	Questo professionista utilizza linguaggi di programmazione e *frameworks* per scrivere codice informatico, creando, migliorando o mantenendo *software*, applicazioni o sistemi informatici. Gli sviluppatori sono divisi in due categorie basate sull'esperienza: *junior* e *senior*.

Tabella 1.1: Figure professionali all'interno di CWBI.

1.2 Tecnologie utilizzate

1.2.1 Spring MVC

Spring MVC è una parte del *framework* Spring, viene ampiamente utilizzato per la creazione di applicazioni *web* ed *enterprise* utilizzando l'architettura *Model View Controller* (MVC). Molte organizzazioni scelgono Spring MVC come *framework* operativo poichè offre un modo altamente personalizzabile e flessibile per sviluppare applicazioni *web* in Java. È ampiamente utilizzato nell'industria per la creazione di applicazioni *web* scalabili, sicure e manutenibili. Per tali motivi CWBI ha adottato questo *framework* per lo sviluppo della parte *front-end* delle applicazioni.

1.2.2 Spring MVC REST

Spring MVC REST è una parte del *framework* Spring che si concentra sulla creazione di servizi *web* che utilizzano un'architettura in linea con i principi *Representational State Transfer* (REST). L'obiettivo principale di Spring MVC REST è consentire agli sviluppatori di creare servizi *web* leggeri, scalabili e interoperabili tra *client* e *server*. Fornisce un insieme di funzionalità per gestire le richieste e le risposte REST, facilitando la creazione di servizi *web* con intefacce uniformi per l'accesso alle risorse e facilitandone il rilascio in ambienti distribuiti. Per tali motivi CWBI ha adottato questo *framework* per lo sviluppo della parte *back-end* delle applicazioni.

1.2.3 Java EE

Java EE (Jakarta EE) è una piattaforma di sviluppo *enterprise* per la creazione di applicazioni aziendali robuste e scalabili. Fornisce specifiche e tecnologie per affrontare

le sfide tipiche delle applicazioni *enterprise*, consentendo agli sviluppatori di concentrarsi sulla logica di *business* anziché sulla gestione delle complessità infrastrutturali.

1.2.4 Flutter

Flutter è un *framework* open-source sviluppato da Google per la creazione di applicazioni *cross-platform*, cioè applicazioni che possono essere eseguite su diverse piattaforme, come iOS, Android, Web e desktop, utilizzando una singola base di codice Dart. Esso non impone un'architettura specifica ma consente agli sviluppatori di scegliere tra diverse architetture, come BLoC (*Business Logic of Component*, nonchè fornisce un meccanismo d'accesso alle API native delle piattaforme iOS e Android fornendo così il supporto multi-piattaforma, ha una community molto attiva ed è diventato popolare per le prestazioni elevate e la sua stabilità, infine, la compilazione *Ahead of Time* consente di tradurre il codice Dart in codice nativo (relativamente *Swift* o *Kotlin* per le piattaforme di destinazione. Per tali motivi CWBI ha adottato questo *framework* per lo sviluppo delle appplicazioni mobile.

1.2.5 Bootstrap

Bootstrap è un *framework front-end* gratuito utilizzato per sviluppare siti *web* e applicazioni *web responsive*. Offre un sistema di griglia flessibile, componenti UI predefiniti e stili CSS di base. Dispone anche di componenti JavaScript opzionali e può essere altamente personalizzato. La documentazione completa e una comunità attiva lo rendono una scelta popolare per creare siti *web* moderni che funzionano su diverse piattaforme e *browser*. Per tali motivi CWBI ha adottato questo *framework* per arricchire lo stile della parte *front-end* delle loro applicazioni.

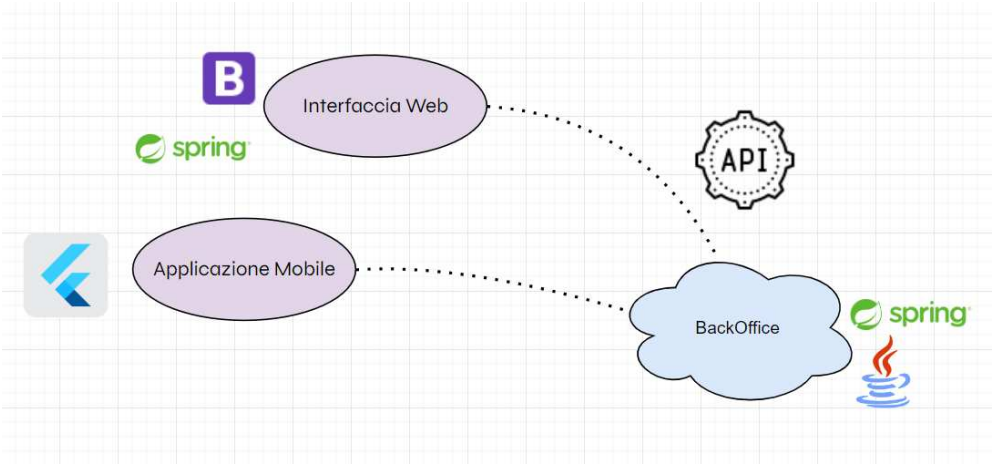


Figura 1.1: Relazioni tra le tecnologie utilizzate Fonte: flaticon.com.

1.3 Processi interni

1.3.1 Flusso di produzione

Tengo a precisare che quanto riportato in questa tesi di laurea, è soltanto quanto visto e appreso da me medesimo e non essendo a conoscenza di come avvenga il primo contatto tra cliente e CWBI, salterò la narrativa di tale avvenimento.

La tipologia di cliente che solitamente si rivolge a CWBI è variabile, ma in un dominio ben definito, esso comprende infatti: banche, istituti finanziari o fornitori di servizi bancari terzi. Nel corso del mio periodo di *stage*, ho avuto inoltre l'opportunità di collaborare alla realizzazione della *suite* di test d'accettazione di un prodotto destinato a tutte le pubbliche amministrazioni che vogliono integrare il servizio nazionale PagoPA all'interno dei loro applicativi.

Per quanto riguarda l'acquisizione dei clienti, a cadenza settimanale avveniva un *meeting* preliminare tra nuovi clienti e il CEO dell'azienda, durante il quale venivano discussi i requisiti funzionali e i vincoli di progetto da parte del cliente. Successivamente al *meeting* il CEO con gli analisti coinvolti elabora una strategia preliminare per produrre quanto richiesto dai clienti, traducendo i loro bisogni in requisiti, verranno prodotti inoltre uno o più preventivi e offerte commerciali che vengono presentati al cliente. Ho notato anche che i clienti tendono a rivolgersi a CWBI per adattamenti di soluzioni già esistenti sviluppate per altri clienti già fidelizzati, richiedendo cambiamenti allo stile (*whitelabeling* ovvero adattare lo stile di base in funzione dell'azienda rappresentante) o l'introduzione di nuove funzionalità specifiche. Una volta ricevuti i preventivi e le offerte commerciali, i possibili clienti sono liberi di scegliere l'opzione più adatta alle proprie esigenze e in caso di accettazione verrà eseguito propriamente il processo di analisi dei requisiti che prenderà quanto di fatto durante le trattative, per ampliarlo e completarlo nel dettaglio. Una volta completata quest'attività si susseguirano i processi di progettazione codifica e conseguente verifica prima del rilascio e del mantenimento.

La comunicazione con i clienti avviata la fase di sviluppo è quasi totalmente affidata a strumenti di supporto quali ITS (come JIRA e SVN), altri clienti invece utilizzavano messaggi di posta elettronica, altri ancora contatti telefonici o riunioni telematiche.

1.3.2 Descrizione della *BaseApp*

Ogni prodotto CWBI nasce come un adattamento, in base alle richieste del cliente, di un applicazione proprietaria nominata *BaseApp*, in quanto, fa da scheletro e da punto di partenza vero e proprio, rendendo più facile per gli sviluppatori, focalizzarsi sui requisiti cruciali. Durante la prima parte del mio periodo di stage ho avuto modo di conoscerne l'architettura affiancato dal *tutor* aziendale. Il core è un'applicazione scritta in JavaEE nella quale vengono utilizzati *Spring MVC* e *Bootstrap* per la componente *front-end*, mentre *Spring MVC Rest* e *Hybernate* per il *back-end*. La soluzione architetturale inoltre, utilizza elementi concettuali relativi alla *Model Driven Architecture* (MDA), metodo di approccio allo sviluppo che pone al centro della strategia la creazione di modelli UML, che, rappresentano astrattamente le componenti del sistema, il loro comportamento e altri aspetti. Una volta codificati i modelli, verranno utilizzati da base per generare, grazie a dei *tools* forniti dall'ambiente di sviluppo, il codice sorgente, la documentazione e altri artefatti del *software* in maniera automatica, tali trasformazioni vengono nominate infatti, *model-to-model e model-to-code*. Questo approccio alla progettazione *software* permette di sfruttare molti vantaggi quali la

possibilità di creare sia modelli specifici che indipendenti dalla piattaforma, rendendo il codice portabile ed interoperabile e grazie alla generazione automatica si riesce ad apportare modifiche ai modelli senza doversi preoccupare di eventuali effetti collaterali nell'implementazione. La figura 1.2 riassume l'architettura della *BaseApp*:

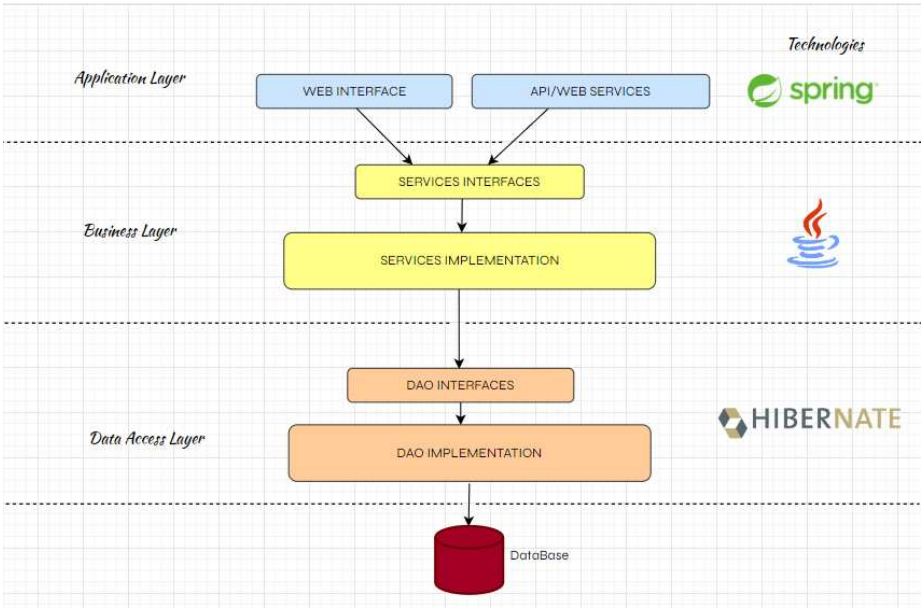


Figura 1.2: Struttura della BaseApp Fonte: flaticon.com.

1.3.3 Progettazione di un prodotto

Dopo la fase di analisi e definizione dei requisiti, la modalità di sviluppo può avvenire principalmente in due modi, a seconda della natura della richiesta del cliente. Se necessità di un' API da integrare in un suo prodotto preesistente allora viene clonata solo la componente di *backend* della *BaseApp*, se il cliente necessita anche di un interfaccia grafica allora si procede a clonare l'intero progetto.

Ovviamente il nuovo progetto creato viene adattato e incrementato sulla base delle richieste del cliente mediante un processo di sviluppo che utilizza elementi del modello di sviluppo a cascata con possibilità di ritorno in caso di variazione dei requisiti. I requisiti del progetto possono infatti subire variazioni sia sul contenuto, sia sul numero, il contatto con il cliente viene di conseguenza sempre mantenuto.

1.3.4 Sviluppo di un prodotto

Subito dopo la fase di progettazione inizia lo sviluppo vero e proprio, dove, uno o più *developer* prendono in carico il progetto e si impegnano formalmente a consegnare un prodotto che soddisfi i requisiti sia lato utente che lato*software*. Per favorire la comunicazione, CWBI adotta generalmente due metodi:

- gli sviluppatori coinvolti nel progetto si siedono in piccoli gruppi;
- gli sviluppatori coinvolti effettuano *meeting* di allineamento giornalieri.

Ovviamente, tutto il codice prodotto prima di andare in produzione, dovrà essere testato in un ambiente diverso da quello di sviluppo riuscendo così a simulare il più possibile l'ambiente di produzione per prevenire *bug* che non si presentavano in ambiente di sviluppo.

1.3.5 Manutenzione di un prodotto

Se la fase di sviluppo all'interno di CWBI ha delle norme unificate a prescindere dal prodotto offerto, non è cosi per la fase di *testing*, che talvolta, viene effettuato dal cliente utilizzando un set di dati più ampio e completo. Esistono delle figure professionali denominate *tester* che notificano, tramite una modalità concordata con CWBI (generalmente ITS), eventuali problemi cambiamenti o *bug* riscontrati nel loro ambiente di test. Questo non era il caso del progetto a cui ho lavorato nella fase finale dello stage, in quanto la batteria di test d'accettazione è stata eseguita, in un apposito ambiente di test, internamente all'azienda. La figura 1.3 rappresenta il ciclo di vita di ogni prodotto *software* a marchio CWBI.

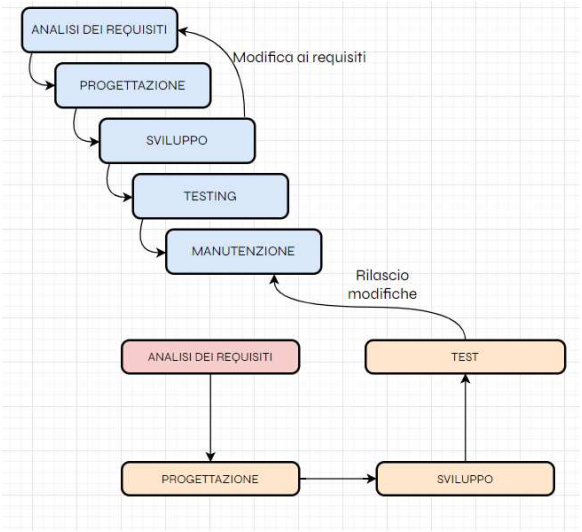


Figura 1.3: Modello di sviluppo aziendale.

1.4 Strumenti di supporto

1.4.1 JIRA

JIRA è un *software* di gestione dei progetti sviluppato da Atlassian. È ampiamente utilizzato come ITS (*Issue Tracking System*) poichè è in grado di fornire un alto grado di personalizzazione, per venire incontro alle esigenze del *team* di sviluppo aiutando a creare un ambiente di lavoro collaborativo. Fornisce inoltre funzionalità di tracciamento e pianificazioni (Diagrammi di Gantt) e riunisce tutto il lavoro programmato per periodi in un' interfaccia grafica intuitiva e anch'essa personalizzabile. All'interno di CWBI esso veniva utilizzato come ITS, VCS e gestore di progetto per i clienti CSE.

1.4.2 SVN

Apache Subversion (SVN) è un *Version Control System* utilizzato per tracciare e gestire le modifiche apportate ai progetti *software*. SVN consente di tenere traccia delle diverse versioni di un progetto, facilitando la collaborazione tra membri del *team*. Gli utenti possono eseguire il *commit* delle modifiche, tornare a versioni precedenti e risolvere conflitti in modo efficiente. È stato ampiamente utilizzato vista la facilità di apprendimento nell'utilizzo di tale strumento, la sua stabilità e il controllo centralizzato che esso offre. All'interno di CWBI veniva utilizzato appunto come VCS e ITS per i clienti non facenti parte del consorzio sopracitato.

1.4.3 VMWare Workstation

VMware Workstation è un *software* di virtualizzazione *desktop* che consente agli utenti di creare e gestire macchine virtuali VM su computer *desktop* o *laptop*. Tale applicativo consente di condividere in una rete più macchine virtuali utilizzando la stessa macchina fisica, isolando gli ambienti di esecuzione delle VM da quello del sistema operativo *host*. Consente inoltre alle macchine di comunicare tra loro fornendo un elevato supporto a vari standard di connettività di rete e una gestione delle macchine virtuali in esecuzione tramite un'interfaccia grafica intuitiva. L'adozione di tale strumento favorisce la collaborazione tra i membri del *team* di sviluppo, in quanto, permette di creare ambienti di test riservati a singoli progetti o a gruppi di macchine, che, possono essere adattate o clonate grazie a dei meccanismi forniti dall'applicativo, in quanto, progetti diversi potrebbero richiedere risorse *hardware* differenti aiutando di conseguenza l'efficienza generale.

1.4.4 EclipseIDE

Eclipse IDE è un ambiente di sviluppo altamente personalizzabile e estensibile che può essere adattato alle esigenze specifiche degli sviluppatori di diverse discipline esso infatti fornisce non solo supporto per il linguaggio Java ma anche per C/C++, Python, PHP, JavaScript, Ruby, e molti altri. La sua flessibilità e la vasta gamma di funzionalità lo rendono una scelta popolare tra gli sviluppatori *software* in molte discipline poichè è uno strumento che centralizza molte delle funzionalita che uno sviluppatore necessita. La figura 1.4 esprime le relazioni che intercorrono tra gli strumenti di supporto, le attività ove essi siano coinvolti e i relativi processi di appartenenza.

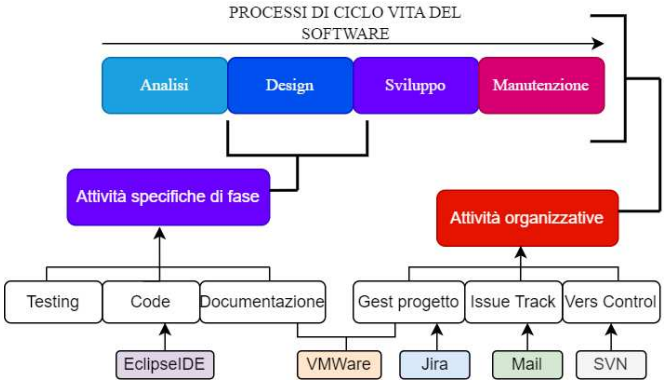


Figura 1.4: Strumenti di supporto e relativi processi.

Capitolo 2

La proposta di stage

2.1 Gli *stage* per l'azienda

L'azienda risulta essere aperta alla collaborazione, in quanto, fornisce un collegamento diretto con l'Università degli Studi di Padova e con un altro istituto superiore di secondo grado convenzionato. L'azienda ha mantenuto un approccio all'avanguardia e orientato all'innovazione nel dominio applicativo di riferimento, assegnando ogni aspetto del problema a una tecnologia o a un *framework* dedicato, l'integrazione alle volte richiede una conoscenza profonda delle varie tecnologie coinvolte, ma per quello che concerne la mia esperienza, non ha mai rappresentato un ostacolo bloccante nel lavoro svolto, c'è da dire però che l'azienda generalmente immette sul mercato prodotti già esistenti il che non è vera e propria innovazione. Ho notato che l'azienda non cercava attivamente nuove alternative tecnologiche, nonostante la discussione critica fosse ben accolta, si tendeva a cercare il massimo livello di esperienza per ogni tecnologia, rimanendo nei confini tracciati dalla *BaseApp*. In generale l'azienda si affida molto all'aiuto di studenti tirocinanti, e mira ad un'integrazione totale del tirocinante nel contesto aziendale, con il fine ultimo dell'assunzione, fornendo da subito formazione sulle metodologie di lavoro e corsi o seminari per permettere agli interessati di sviluppare nuove competenze. Durante la mia esperienza ho potuto notare come i profili selezionati provenissero tutti da un istituto di secondo grado convenzionato, dall'Università degli Studi di Padova oppure studenti neodiplomati, offrendo in tal caso un contratto di apprendistato. È evidente che l'azienda è costantemente alla ricerca di nuove opportunità, consapevole del ritorno che può derivare da esperienze di questo tipo, a prova di ciò infatti ho notato come il *tutor* stesso incoraggiasse gli studenti a prendere parte a progetti con il solo fine di allargare le competenze, incoraggiando lo studente ad essere più autonomo ma anche istruendolo ad una corretta comunicazione con il personale. L'insieme di metodologie di lavoro adottate da CWBI influenza la propria innovazione spingendo le varie parti interessate, attraverso diverse tecniche, a cercare sempre soluzioni fornendo uno spazio strutturato che permetta il proliferare di nuove idee. Promuove inoltre la collaborazione tra i componenti del *team* e gli *stakeholders* interessati, con riunioni o contatti frequenti e strumenti di gestione che permettano lo scambio di informazioni tra le parti. Per quanto riguarda la gestione dei processi invece si adottano strategie che rendono lo sviluppo flessibile e reattivo ai cambiamenti proposti mettendo al primo posto l'attenzione ai bisogni del cliente finale.

2.2 Firma digitale

2.2.1 Firma digitale semplice FES

Prima di addentrarmi nella narrazione volevo fare una digressione riguardo la firma digitale e qual'è il valore ad esso attribuibile, in quanto trovo che siano elementi necessari per capire nel profondo i bisogni che l'applicativo avrebbe dovuto soddisfare. La firma digitale costituisce un metodo elettronico per autenticare un documento informatico, confermando la sua creazione o approvazione da parte di una persona specifica. Spesso si utilizzano firme digitali per garantire l'integrità e l'autenticità dei documenti digitali, facilitando transazioni sicure *online*.

Il processo per acquisire una firma digitale varia a seconda del tipo di firma adottata, ma generalmente segue questi passaggi:

- generazione delle chiavi crittografiche;
- creazione della firma digitale;
- allegazione della firma sul documento;
- verifica della firma.

La firma digitale semplificata (o semplice) è costituita da una serie di dati elettronici, secondo quanto stabilito dal regolamento eIDAS, in cui gli utenti intenzionati a firmare sono "acclusi o connessi tramite associazione logica a dati elettronici ed utilizzati per firmare". Si definisce semplice perché, il suo ruolo è solo dare un'autenticazione informatica a un documento elettronico.

Il suo valore legale è strettamente legato al documento informatico associato, di conseguenza, il valore giuridico della firma coincide con il valore giuridico del documento informatico, ma in generale non fornisce garanzie sull'identità del firmatario, né ha valore probatorio in sede legale.

2.2.2 Firma digitale avanzata FEA

La firma digitale avanzata garantisce maggiore sicurezza rispetto alla firma semplice, in quanto il processo di acquisizione di tale firma tiene conto di dati aggiuntivi per garantire l'identità del firmatario, tale processo generalmente necessiterà di strumenti appositi (quali penne o *tablet* specifici) per poter essere completati correttamente.

Apponendo una firma di questo tipo su un documento implica che i dati aggiuntivi acquisiti siano collegati alla firma stessa, rendendo così identificabile una firma anche dopo successive modifiche al documento, un esempio di firma digitale avanzata è la grafometrica, firma che è stata adottata nel progetto di *stage*. Essendo un tipo di firma avanzato la firma grafometrica è in grado di acquisire tratti biometrici unici della firma manuale di una persona come la velocità, la pressione, l'angolo e la direzione del tratto. I sistemi che presidiano la sicurezza di una soluzione di questo tipo compiono delle verifiche di corrispondenza tra la firma sottoscritta e la firma depositata per poi procedere alla sua validazione.

Il suo valore giuridico è più alto di una FES e generalmente può considerarsi autentica per una vasta gamma di documenti e transazioni, tuttavia il suo valore può comunque dipendendere dalle normative vigenti e dall'auditabilità del processo di acquisizione. In generale viene considerata valida ma potrebbe non esserlo per scopi più critici quali contratti legali o se usata in contesti giuridici specifici.

2.2.3 Firma digitale qualificata FEQ

La Firma Elettronica Qualificata (FEQ) è la modalità di firma digitale che garantisce maggior rigore e sicurezza nella sua implementazione. Queste qualità sono garantite poiché la procedura con cui un utente ottiene una firma qualificata presso un Ente di Certificazione Qualificato (ECQ) implica specifici controlli sull'identità prima di rilasciare un certificato informatico contenente le informazioni necessarie per creare la firma.

La firma stessa poi, non può essere generata in qualsiasi momento ma c'è bisogno di un secondo supporto, che può essere *hardware* o *software*, utilizzato per autenticarsi, generare la firma e successivamente applicarla sul documento designato. Utilizzando un dispositivo fisico si guadagna in sicurezza in quanto la chiave privata per generare la firma è fisicamente contenuta nel dispositivo ma si perde portabilità nel caso in cui non ci fosse accesso diretto al dispositivo fisico, mentre con una soluzione *software* è possibile utilizzare la stessa firma su dispositivi diversi in quanto basta che l'applicativo di gestione della firma sia installato sui dispositivi.

La figura 2.1 presenta le qualità offerte da ogni tipo di firma digitale:

Figura 2.1: Tipologie di Firma Digitale Fonte: intesa.it.

2.3 Problemi affrontati

2.3.1 Soluzioni progettate

L'obiettivo principale consiste nella corretta raccolta di firme grafometriche per ottenere il consenso informato sui documenti medici.

In questa sezione discuto nel dettaglio come è stato possibile determinare quando un dispositivo fosse compatibile con la firma grafometrica, come l'applicativo sia in grado di fornire funzionalità per la gestione del ciclo di vita di una licenza e come tutto

questo sia integrato all'interno dell'appplicazione *mobile* finale, la quale anch'essa ha delle funzionalità cruciali da dover soddisfare.

2.3.1.1 Compatibilità con firma grafometrica

Durante un *brainstorming* tra me, il collega Jacopo Angeli e il *tutor* che faceva da mediatore, abbiamo identificato le caratteristiche fondamentali che un dispositivo *mobile* deve possedere per essere considerato compatibile con la firma grafometrica::

- *touchscreen* di alta qualità: Il dispositivo *mobile* deve avere un *display* di alta qualità che possa catturare con precisione gli *input* della penna o del dito.
- *software* di riconoscimento dell'*handwriting*: Deve essere disponibile un *software* di riconoscimento dell'*handwriting* che possa interpretare la firma identificandone i tratti unici e convertirla in dati digitali.
- penna attiva: solo una penna attiva (in assenza di *display* avanzati) può registrare tutti gli *input* per la firma grafometrica.

2.3.1.2 Funzionalità per gestione delle licenze

Durante il periodo di *stage*, ho implementato le seguenti funzionalità per la gestione delle licenze: la creazione di una nuova licenza, l'identificazione degli utenti per interfacciarsi alla firma e la gestione delle licenze prossime alla scadenza. PPrima di procedere con l'implementazione, ho sviluppato una logica per gestire gli stati delle varie componenti nell'architettura progettata (per i dettagli sull'architettura, fare riferimento al capitolo 3). Ciò ha permesso di assegnare uno stato a ciascuna licenza e, grazie a dei controllori, di fornire meccanismi per l'accesso all'applicazione (o registrazione in caso di nuovo utente), permettendo agli utenti di interfacciarsi con il pacchetto di sviluppo offerto da Namirial per la firma dei documenti.

Oltre a queste funzionalità grazie ad una corretta gestione degli stati è stato facilmente implementabile un meccanismo che avvisava gli utenti tramite *mail* ad un mese dalla scadenza, in maniera che essi potessero rinnovare la licenza qualora lo volessero.

2.3.1.3 Integrazione contesto mobile

L'obiettivo principale fornito dal capitolato era produrre la componente *software* che gestisse la registrazione degli utenti e la gestione delle licenze. Analizzando il problema con una visione d'insieme poi capiremmo facilmente che tale componente presa singolarmente avrebbe avuto poco senso, ecco spiegato il motivo che ha portato l'azienda a suggerire l'integrazione di tale componente in una applicazione *mobile* a tutti gli effetti, che avrebbe dovuto attenersi a degli specifici vincoli tecnologici e che avrebbe dovuto soddisfare alcune caratteristiche per potersi considerare completa, ora andremo ad analizzare tali funzionalità.

Applicazione senza etichetta Tra le funzionalità cruciali richieste per l'applicazione c'era la *White label software*, ovvero un *software* senza etichetta, che consente agli utenti di personalizzare l'interfaccia grafica scegliendo colori di base, *font* del testo e loghi da caricare. Quando in un'applicazione viene usato questo modello stilistico, generalmente viene prima caricata un'interfaccia grafica di base, la quale viene aggiornata sulla base

di scelte effettuate dal signolo utente, così facendo sono andato a produrre (collaborando con il collega Jacopo Angeli) del *software* che si adattasse in base alle varie scelte che ogni utente poteva effettuare.

Gestione autenticazione utenti Ho collaborato nello sviluppo dell'applicazione *mobile*, concentrandomi sulla gestione del flusso di autenticazione degli utenti.

La realizzazione di tale funzionalità era cruciale per poter poi andare a progettare la logica che permettesse l'implementazione del *whitelabeling* Per implementare concretamente queste funzionalità abbiamo pensato di adottare la stessa strategia vista nella progettazione della componente di *enrollment* ovvero inserire una logica che permettesse la gestione degli stati delle componenti, in particolare, il modulo di *login*. Così facendo infatti riuscivamo a determinare esattamente quando un utente stava tentando l'operazione di *login*, se l'operazione va a buon fine il sistema caricherà l'interfaccia grafica con delle caratteristiche specifiche scelte dall'utente. Ad ogni sessione di *login* il sistema poi controlla automaticamente lo stato corrente della licenza del cliente, andando ad interfacciarsi con il pacchetto Namirial. La figura 2.2 fornisce una panoramica generale delle componenti progettate per implementare le soluzioni ai requisiti fondamentali del progetto.

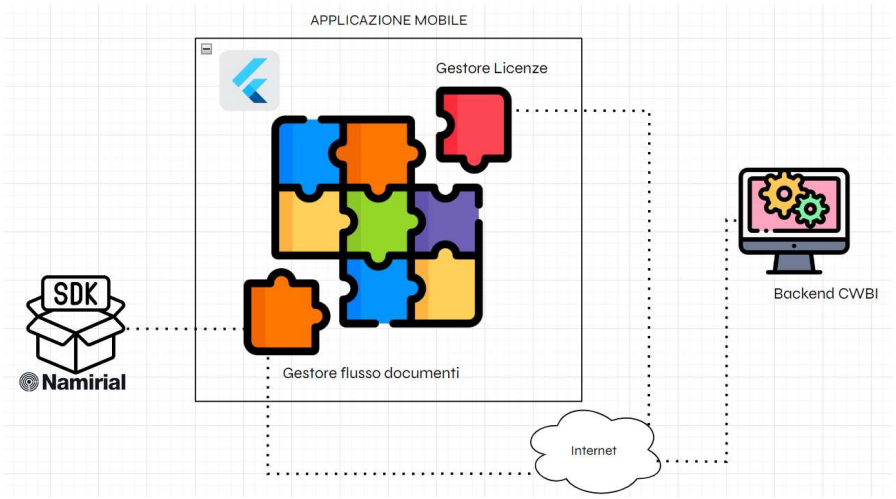


Figura 2.2: Panoramica delle componenti progettate Fonte: flaticon.com.

2.4 Strategia adottata

2.4.1 Pre-*stage*

Il Piano di Studi e, di conseguenza, l'ateneo fissano dei limiti in termini di tempo per il periodo di *stage*, pur essendo un'esperienza formativa. La strategia adottata dall'azienda quindi, dovrà predisporre dei comportamenti per agevolare il tirocinante ad apprendere quanto più possibile dall'esperienza. Non è stato previsto infatti che il sottoscritto seguisse dall'inizio alla fine il progetto, l'idea era concentrarsi sui momenti cruciali (definizione dei requisiti obbligatori, sviluppo e rilascio delle funzionalità, chiusura di progetto). Prima di iniziare effettivamente il periodo di tirocinio, ho

partecipato a due incontri preliminari con colui che sarebbe diventato il mio *tutor* aziendale. Durante questi incontri, sono stati affrontati vari temi come:

Primo incontro preliminare Questo incontro era prevalentemente conoscitivo. Sono stato coinvolto in una lunga conversazione con il CEO, il quale cercava di capire i miei interessi principali (accademici e non) e in quale area di sviluppo mi sarebbe piaciuto operare. Successivamente a questa conversazione, mi è stato somministrato un *test* di cultura generale, il quale viene somministrato a qualsiasi tirocinante voglia prendere parte alle attività di CWBI, grazie al quale il *tutor* aziendale è riuscito a capire quali potessero essere le aree operative dove avrei potuto dare il meglio o imparare di più, sempre con l'obiettivo di un'ipotetica assunzione.

Secondo incontro preliminare Il carattere di questo incontro è stato più formale del primo, è stato svolto a distanza tramite l'ausilio dell'applicativo *Google Meet*, l'obiettivo era proprio definire i vincoli e i requisiti cruciali da dover sviluppare per poter portare avanti il progetto di *stage*. Dopo una discussione approfondita sui vari vincoli e requisiti del progetto, il *tutor* mi ha proposto di prenderne parte, considerando anche la partecipazione del collega e il mio entusiasmo nel dominio applicativo presentato. Successivamente, è stata eseguita un'analisi dettagliata dei requisiti. Dopo questa serie di incontri è iniziato il vero e proprio periodo di *stage*.

2.4.2 Post-*stage*

Al termine del periodo di tirocinio, CWBI ha posto come unica condizione risolvere quanto richiesto dal capitolato e fornire un'architettura funzionante. Per quanto riguarda il rilascio, il progetto è stato purtroppo messo in pausa per consentire al *team* di sviluppo di concentrarsi su altri progetti con maggiore priorità. Ad ogni modo seguendo il flusso di produzione interamente, la fase di rilascio, erogazione e gestione delle licenze sarebbe stato gestito internamente all'azienda, seguendo le norme aziendali. Ho potuto osservare tale attività durante gli ultimi giorni di tirocinio, in quanto, ho preso parte all'attività di *test* di un prodotto precedentemente rilasciato da CWBI che andava revisionato in seguito ad un cambio di versione di alcune API, tale attività verrà descritta nel dettaglio nel capitolo 4. La figura 2.3 riassume la strategia adottata dall'azienda.

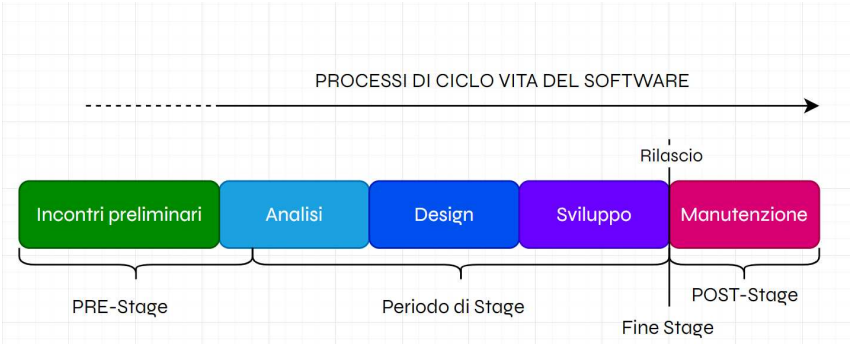


Figura 2.3: Strategia aziendale pianificata per il tirocinio.

2.5 Il progetto di *stage*

2.5.1 Obiettivi

Obiettivi aziendali Gli obiettivi che l'azienda aveva posto per il progetto di tesi sono il riflesso di quanto descritto nella sezione 2.2. Dopo una prima decomposizione, i cossiddetti *needs* del capitolato sono stati trasformati in requisiti, che possono esseres oddisfatti, producendo delle implementazioni, esse vengono prodotte con l'esecuzione di una serie di attività ben progettate e programmate. La tabella 2.1 espone gli obiettivi aziendali e la loro rilevanza nel progetto di *stage*.

Obiettivo	Descrizione	Rilevanza
Accesso al sistema	L'utente deve poter accedere al sistema	Obbligatorio.
Enrollment dispositivo	L'utente deve poter attivare la propria licenza	Obbligatorio.
Download licenza	L'utente deve poter effettuare il *download* della sua licenza	Obbligatorio.
Rinnovo licenza	L'utente deve poter rinnovare la propria licenza d'utilizzo	Obbligatorio.
Errore connessione	L'utente deve visualizzare un messaggio d'errore in caso stesse cercando di utilizzare un servizio finchè *offline*	Opzionale.
Integrazione Namirial SDK	Il sistema deve integrare Namirial FEA SDK	Obbligatorio.
Distribuzione applicativo	Il sistema mobile completo deve essere distribuito su tutti gli store disponibili	Obbligatorio.
Vincolo tecnologico	La tecnologia utilizzata dovrà essere Flutter	Obbligatorio.
Whitelabeling	L'interfaccia grafica dovrà rispettare i principi del *white label software*	Obbligatorio.
Condivisione licenza	La licenza può essere condivisa da più dispositivi	Desiderabile.
Storico licenze	Il sistema mette a disposizione lo storico delle licenze acquistate dall'utente	Desiderabile.
Temi preimpostati	L'utente può scegliere tra una serie di temi preimpostati	Desiderabile.

Tabella 2.1: Obiettivi aziendali da raggiungere al termine del tirocinio.

Obiettivi personali Oltre agli obiettivi aziendali, mi sono posto anche degli obiettivi personali che avrei sperato di raggiungere al termine dell'esperienza, riassunti in:

- inserimento in un contesto produttivo;
- apprendimento di nuove metodologie di lavoro e sviluppo;
- apprendimento di nuove conoscenze tecnologiche.

2.5.2 Vincoli

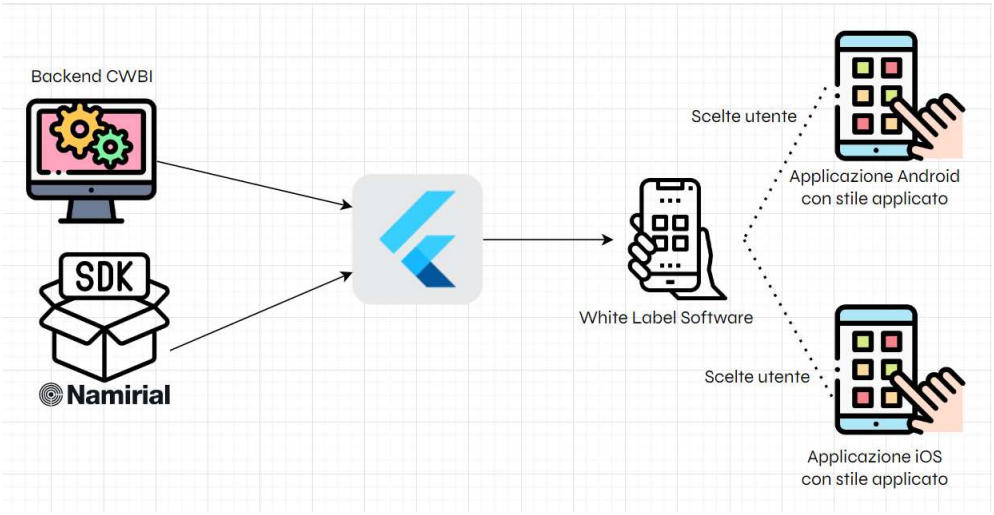


Figura 2.4: Vincoli imposti e loro impatto sul prodotto Fonte: flaticon.com.

2.5.3 Tecnologie utilizzate per la realizzazione

2.5.3.1 Flutter

Vista la grande versatilità del *framework* in questione, durante il secondo incontro il *tutor* aziendale esponendomi il progetto ha suggerito l'utilizzo di tale tecnologia. Non avevo mai avuto occasione di utilizzare tale tecnologia, infatti, durante la prima parte del tirocinio ho dovuto impegnarmi per capire a fondo il suo funzionamento interno così da poter capire a pieno come poter risolvere i problemi richiesti. Trovo che tale scelta sia ottimale visto il contesto produttivo nel quale il capitolato è inserito.

2.5.3.2 Namirial SDK

Vista la necessità di integrare la tecnologia di firma grafometrica nell'applicativo CWBI ha deciso di affidarsi ad un'azienda terza che offrisse un pacchetto di sviluppo indipendente che consente di risolvere autonomamente tale problema. Ecco come è nata la collaborazione con Namirial, multinazionale specializzata nella distribuzione di *software* e soluzioni per la digitalizzazione di compagnie private e di enti della pubblica amministrazione. Namirial infatti ha fornito un *Software Development Kit* (SDK) ovvero un'insieme di strumenti atti all'integrazione della tecnologia firma grafometrica all'interno delle proprie applicazioni *mobile*. Tale pacchetto infatti permette all'utente di aprire un'interfaccia grafica dove, nei punti delineati, poter apporre la propria firma

che viene generata seguendo l'algoritmo descritto in 2.1 tramite una chiamata alle API dei servi interni a Namirial. Una volta firmato il documento segue il flusso di gestione descritto in 2.2.1.3

2.5.3.3 Backend CWBI

La componente designata alla raccolta dei documenti firmati era già presente all'inizio del mio periodo di tirocinio, è stato infatti vincolante doversi interfacciare con essa per poter implementare soluzione proposta per la gestione del flusso dei documenti.

2.5.4 Relazione con la strategia di sviluppo

Il rientro che l'azienda ha per ogni progetto affidato a uno o più tirocinanti è estremamente variabile e strettamente correlato al progetto stesso. In termini più pratici il rientro economico non è sempre assicurato, in quanto, non è detto che il lavoro portato avanti dal tirocinante corrisponda perfettamente a quanto ci si aspetta sul mercato attuale, oppure come nel mio caso, se il progetto viene messo in pausa il rientro economico non è certo. Un altro tipo di rientro che possiamo analizzare è il rientro sociale, in quanto un candidato potrebbe suggerire nuove tecnologie per affrontare un problema, o proporre una nuova metodologia da adottare internamente che permetterebbe di guadagnare in efficienza, tale rientro dipende esclusivamente dalle caratteristiche del candidato e quindi difficilmente valutabile al di fuori della specifica esperienza. Il punto del discorso è che non sempre l'azienda ha garanzie di ritorno quando si prende a carico uno *stagista*, ma credo anche che sia innegabile come un esperienza del genere offra enormi opportunità di crescita sia per l'azienda che per lo studente, credo inoltre, che anche questo possa essere considerato un rientro per l'azienda.

2.5.5 Motivazione della scelta

Grazie Stage-IT ovvero l'evento promosso da Confindustria Veneto Est con i Dipartimenti di Matematica e Scienze Statistiche dell'Università di Padova sono venuto a conoscenza di CWBI in quanto, molte aziende che operano in campo ICT hanno preso parte all'evento, presentando uno o più progetti per la quale cercavano tirocinanti che se ne occupassero. Una volta saputo della cosa, mi sono subito documentato, stillando una lista di quelli che potevano essere i progetti più interessanti e le relative aziende. All'evento poi, ogni azienda aveva uno *stand* dove effettuava colloqui conoscitivi con i possibili candidati, andando ad esporre nel dettaglio i progetti e rispondendo ad eventuali domande poste dai candidati stessi. Attirato dal colui che sarebbe stato il futuro *tutor* di *stage*, in quanto, l'azienda non era nella lista delle proposte che avevo reputato prioritarie, mi sono seduto allo *stand* di CWBI attirato dal carisma e dalla passione che trasmettevano le parole del rappresentante aziendale, dopo aver appreso le informazioni generali dell'azienda ho valutato attentamente le offerte ricevute. La tabella 2.2 esprime i motivi che mi hanno portato a scegliere CWBI come realtà per la mia esperienza di tirocinio.

Impatto iniziale	L'impatto iniziale è stato fondamentale, in quanto, CWBI è stata l'unica realtà in grado di affascinarmi ed invogliarmi alla partecipazione effettiva.
Ambito aziendale	L'ambito aziendale e la tipologia di clientela, ovvero istituti bancari, mi ha dato l'idea di inserirmi in un contesto lavorativo di prestigio, dove ogni scelta viene pianificata e ponderata con precisione potendo contare inoltre su un organico ricco di esperienza,
Comparto tencologico	Il comparto tecnologico mi ha affascinato sin da subito in quanto si adatta perfettamente a quelle che sono le richeiste del mercato odierno del mondo *enterprise*, ho pensato quindi che avrei potuto otttenere conoscenze fondamentali per il mio futuro da informatico.
Posizione geografica	La posizione dell'azienda è stato un altro aspetto che ha condizionato la mia scelta, in quanto, ho trovato molto più stimolante optare per una realtà distante per spingermi ad uscire dalla mia zona di *comfort* e in secondo luogo per potermi inserire in un contesto urbano totalmente diverso.

Tabella 2.2: Motivazioni che mi hanno portato a scegliere CWBI.

La tabella 2.3 espone gli obiettivi che mi ero personalmente proposto di raggiungere durante l'esperienza di *stage*.

Contesto produttivo	Possedere la capacità di inserirmi completamente in un contesto produttivo in autonomia, assumendo un buon controllo delle *best practises* aziendali.
Competenze tecnologiche	Raggiungere un buon livello di conoscenza del linguaggio Java e del *framework* Flutter, imparando nel dettaglio come progettare un'applicazione professionale.
Competenze relazionali	Imparare a lavorare come parte di un *team*, imparando a riconoscere quando cercare di risolvere i problemi in autonomia e quando invece chiedere aiuto

Tabella 2.3: Obiettivi personali da raggiungere al termine del tirocinio.

Capitolo 3

Flusso di lavoro

3.1 Metodo di lavoro

3.1.1 Pianificazione

Durante lo stage, abbiamo riflettuto l'insieme di criteri che motiva ogni scelta di pianificazione eseguita, rispecchiando le metodologie aziendali, in quanto, la mia inesperienza avrebbe potuto causare ritardi nelle attività e compromettere il completamento dei vincoli iniziali tra azienda e tirocinante. Inizialmente, il *tutor* ha elaborato una pianificazione generale contenuta all'interno del Piano di Lavoro, presentato prima di iniziare le attività di tirocinio. All'interno, abbiamo diviso le attività per ogni periodo individuato, con una durata generalmente fissa, con possibilità di allungare ulteriormente la disponibilità di tempo in caso di compiti particolarmente complessi, e dopo aver definito un "coefficiente di gravosità" per ogni attività prevista dalla pianificazione, abbiamo assegnato i periodi di tempo disponibili alle attività da completare. Questo ci ha permesso di definire una pianificazione più rigorosa, con pre e post condizioni da rispettare e scadenze ben definite, così facendo l'avanzamento dei lavori è stato notevolmente agevolato, in quanto, ho capito cos'avrei dovuto fare in ogni periodo lavorativo grazie a questa attività di pianificazione, questo mi ha consentito di anticipare i passi necessari per soddisfare le richieste e formulare una strategia a monte dello svolgimento delle attività. La pianificazione ha anche svolto una funzione retrospettiva, in quanto, dopo le attività svolte, ho svolto delle analisi retrospettive. Queste analisi mi hanno di individuare quali sono stati i motivi che hanno portato ad un rallentamento nelle attività, così facendo sono stato in grado di predisporre meccanismi per evitare che questo tipo di situazioni sgradevoli si verifichino nuovamente.

3.1.2 Interazioni con *tutor* aziendale

Come spiegato in precedenza, ho diviso le attività per settimana, andando a riflettere quelli che sarebbero stati i processi di ciclo di vita del *software* istanziati nel periodo preso in esame. Ogni qual volta iniziavo un nuovo periodo, sistematicamente, avveniva una riunione con il *tutor* aziendale per andare a fare un allineamento e successiva retrospezione riguardo il periodo appena passato, in particolare in caso di gravi problemi o in caso necessitassi di chiarimenti avveniva una sessione di *brainstorming* nella quale il *tutor* stesso cercava di farmi arrivare alle soluzioni dei problemi presentati, stimolandomi a pensare in maniera innovativa e non pensando a quanto già visto in passato.

3.2 Analisi dei requisiti

3.2.1 Descrizione del prodotto

La richiesta del cliente è stata posta sin da subito con molta chiarezza, egli infatti aveva richiesto un applicazione *mobile* dove poter permettere ad un generico utente di autenticarsi e subito dopo, riuscire a fargli firmare un documento medico di consenso informato, tale applicazione deve inoltre prevedere un modulo che permetta la gestione delle licenze, tale componente dovrà occuparsi di una serie di controlli che permettono all'utente di autenticarsi e di conseguenza gestire le operazioni che vengono rese disponibili all'utente sulla base dello stato della licenza stessa. Il tutto doveva essere svolto tenendo in conto che operazioni come la firma del documento (lato paziente e lato medico) e il relativo *upload* nel *database* aziendale erano operazioni che non potevano essere svolte in concorrenza ma gestendo il flusso tramite meccanismi di sincronizzazione. Tale richiesta che sembra ben strutturata però, per poter essere compresa nella sua totalità ha avuto bisogno di una successiva decomposizione, fino ad ottenere compiti più semplici in maniera tale da poterli aggregare con più facilità per andare a definire i requisiti di progetto. A tal proposito sono state svolte diverse sessioni di *brainstorming* per andare a sviscerare il problema in ogni sua sfaccettatura, andando, per ogni nuovo requisito individuato, a definire una procedura per poter testare e tracciare la validità di tale requisito.

3.2.2 Attori individuati

In prima battuta, il lavoro principale che ho svolto prima di andare a decomporre quanto riportato dal capitolato è stato individuare gli attori coinvolti nell'utilizzo del sistema, così facendo sono riuscito successivamente a capire dove focalizzarmi per andare a cercare i requisiti che non sono esplicitati propriamente dal capitolato, ma che sicuramente serviranno per permettere al sistema nella sua totalità, di funzionare come previsto e quindi correttamente. Gli attori individuati sono riassunti nella tabella 3.1.

Attore	**Tipologia**	**Scopo nel sistema**
Utente comune	Interno	Utente generico appena atterrato nell'applicazione.
Utente autenticato	Interno	Utente generico che ha superato la fase di autenticazione, può utilizzare tutte le funzionalità del sistema.
Namirial FEA SDK	Esterno	Pacchetto di sviluppo offerto dall'azienda Namirial, esterno poichè separato dal sistema principale.

Tabella 3.1: Attori individuati dopo l'attività di analisi dei requisiti.

3.2.3 Casi d'uso modellati

Una volta che gli attori sono stati definiti, dopo aver chiesto conferma di correttezza al *tutor*, sono partito con la modellazione dei casi d'uso, andando prima a decomporre le funzionalità utente definite dal capitolato in oggetti via via più semplici, fino ad arrivare a delle unità fondamentali. Tali unità vengono così chiamate appunto perchè indivisibili in compiti più semplici e sono state la base di partenza attraverso le quali sono andato a modellare i casi d'uso del sistema. Tali unità poi si rifletteranno nei casi d'uso, e nel momento in cui verranno aggregati per formare un requisito esse si rifletteranno anche in loro per costruzione. I casi d'uso individuati e modellati sono schematizzati nella figura 3.1.

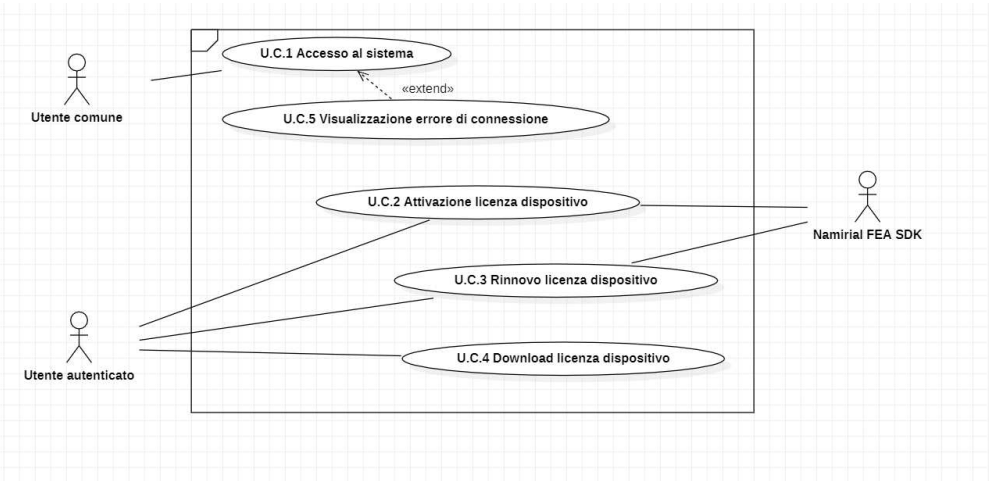


Figura 3.1: Casi d'uso modellati durante l'attività di analisi dei requisiti.

I casi d'uso possono essere riassunti dalla tabella 3.2:

Nome	Attori	Descrizione
Accesso al sistema	Utente comune	L'utente tenta l'accesso al sistema.
Attivazione licenza dispositivo	Utente comune, Namirial FEA SDK	L'accesso si conclude eseguendo la registrazione del dispositivo e validando la sessione tramite l'SDK.
Rinnovo licenza dispositivo	Utente autenticato, Namirial FEA SDK	L'utente rinnova la sua licenza d'utilizzo.
Download licenza dispositivo	Utente autenticato	L'utente scarica la propria licenza dal *server*.

Tabella 3.2: Descrizione dei casi d'uso individuati durante l'attività di analisi.

Ho anche individuato un caso d'uso che prevedeva l'eventualità di un errore di connessione esposto in tabella 3.3.

Nome	Attori	Descrizione
Visualizzazione errore di connessione	Utente comune	L'utente visualizza un messaggio d'errore in seguito ad un errore di connessione.

Tabella 3.3: Descrizione del caso d'uso d'errore.

3.2.4 Requisiti individuati

Requisiti funzionali Partendo dai casi d'uso individuati dopo una prima analisi e dopo varie consultazioni con gli *stakeholders* del progetto, ho prodotto una lista di requisiti funzionali che viene esposta dalla tabella 3.4:

Descrizione	Importanza	Fonte	Tipologia
L'utente deve poter accedere al sistema	Obbligatorio	Casi d'uso	Funzionale.
L'utente deve poter attivare la propria licenza	Obbligatorio	Casi d'uso	Funzionale.
L'utente deve poter effettuare il *download* della sua licenza	Obbligatorio	Casi d'uso	Funzionale.
L'utente deve poter rinnovare la propria licenza d'utilizzo	Obbligatorio	Casi d'uso	Funzionale.
L'utente deve visualizzare un messaggio d'errore in caso stesse cercando di utilizzare un servizio finchè *offline*	Opzionale	Casi d'uso	Funzionale.
Il sistema deve integrare Namirial FEA SDK	Obbligatorio	Richiesta del cliente	Vincolo.
Il sistema mobile completo deve essere distribuito su tutti gli store disponibili	Obbligatorio	Richiesta del cliente	Vincolo.

Tabella 3.4: Descrizione dei requisiti funzionali.

I requisiti di vincolo invece, come descritto parzialmente nella sottosezione 2.5.2 sono tali perchè derivano da vincoli espliciti esposti dal cliente o condizionati dalla natura del problema, essi quindi saranno obbligatori e necessariamente soddisfatti altrimenti la soluzione proposta non avrebbe senso di esistere. La tabella 3.5 descrive i requisiti di vincolo che ho individuato durante l'attività di analisi.

Descrizione	**Importanza**	**Fonte**	**Tipologia**
La tecnologia utilizzata dovrà essere Flutter	Obbligatorio	Natura del problema	Vincolo
L'interfaccia grafica *white label*	Obbligatorio	Richiesta del cliente	Vincolo.

Tabella 3.5: Descrizione dei requisiti di vincolo.

3.3 Progettazione

3.3.1 Architettura

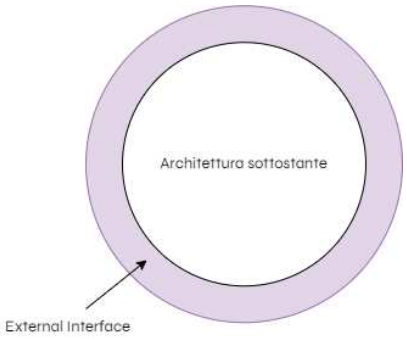
Dopo aver terminato l'analisi dei requisiti, ho affrontato una scelta fondamentale per far progredire il ciclo di vita del *software* di progetto, ovvero la scelta di un'architettura che andasse a "restringere" il dominio del progetto evidenziato dalla precedente attività progettuale e principalmente le strade possibili erano due:

- far si che l'architettura del prodotto rispecchiasse completamente la *BaseApp*;
- cercare cercare una soluzione architetturale totalmente diversa.

Dopo uno studio approfondito del *framework* Flutter, ho scoperto un'architettura molto utilizzata nelle applicazioni *mobile* professionali, e l'ho implementata con il collega per risolvere le sfide emerse durante l'analisi. L'architettura implementata prende il nome di *Clean Architecture* che come si può intuire, mira ad ottenere applicazioni solide separando quanto più possibile la *business logic* dalla implementazione vera e propria, rendendo così l'applicazione più facile da testare e manutenere e cambiare nel tempo. Essa si basa sul principio fondante della *Layered Architecture*, che organizza il lavoro in strati concentrici, al centro dei quali risiede la logica di *business* aziendale. Ora fornisco una definizione formale dello scopo preciso di ogni strato.

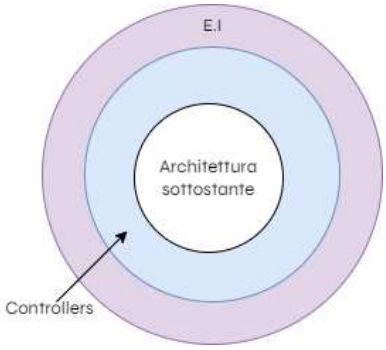
External interfaces

Questo livello contiene le implementazioni di interfacce esterne o *frameworks*, componendo l'interfaccia utente e consentendo all'applicazione di comunicare con l'infrastruttura sottostante.



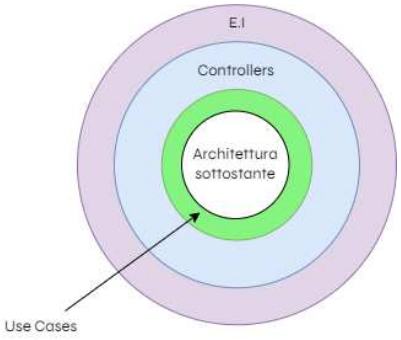
Controllers

Questo livello gestisce la comunicazione tra gli strati interni ed esterni, fornendo interfacce e metodi di conversione dei dati per rendere indipendenti i livelli dalle tecnologie adottate, fornenendo ulteriore supporto ai cambi di tecnologie in propensione all'innovazione, esso fondamentalmente converte i dati dagli *Use cases* e li traduce per comunicare con i livelli esterni.

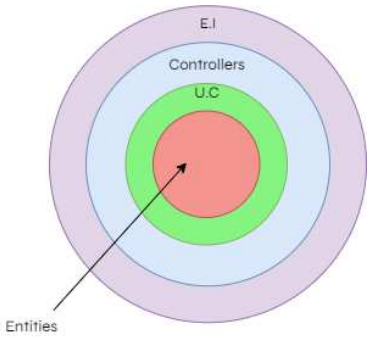


Use cases

Questo livello traduce la logica di *business* generale dell'applicazione in regole specifiche che modellano il flusso dei casi d'uso individuati, come l'attivazione o il rinnovo di una licenza.



Entities



Questo livello contiene la logica di *business* dell'applicazione, ovvero le regole che rimangono costanti tra le varie applicazioni prodotte dall'azienda. Rimangono tali in quanto fondamento della logica d'impresa dell'azienda, come le regole d'accesso all'applicazione o i principi regolanti l'utilizzo delle licenze Namirial.

Come indicato dalla direzione delle frecce la dipendenza è inversa, ovvero gli strati inferiori non dipendono funzionalmente dai superiori, ecco come ogni componente quindi può essere facilmente testata individualmente simulando l'esecuzione del *software* appartenente ai livelli inferiori. La natura sistematica dell'architettura poi rende difficile fare errori in quanto ogni compito è suddiviso ai minimi terimini, di conseguenza questo si tradurrà in componenti meno complesse e quindi meno soggette ad errori. Ho implementato la soluzione architetturale nell'applicazione, creando un modulo per ogni funzionalità specifica e risolvendo i requisiti:

- ***license***: modulo dedicato alla gestione delle licenze;
- ***whitelabeling***: modulo dedicato alla gestione della grafica *whitelabel*.

Ho anche lavorato in parte al modulo ***login***, ovvero il modulo dedicato alla gestione degli accessi all'applicazione, in collaborazione con un altro collega in quanto era "terreno comune" nei relativi progetti di *stage*. Successivamente, ho implementato la suddivisione in strati, dividendo ogni modulo in tre sezioni che riflettono l'architettura come in figura 3.2:

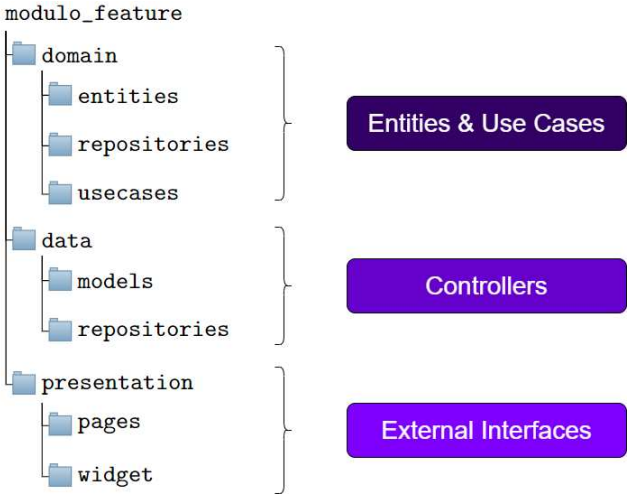


Figura 3.2: Struttura della *repository* e relazione con *Clean Architecture*.

- il *domain layer* contiene la definizione della logica di *business*, dei modelli di dati e *repositories* utilizzate per accedervi e le definizioni dei casi d'uso del modulo;

- il *data layer* contiene l'implementazione delle *repositories*, i DTO necessari per scambiare dati e metodi di conversione per permettere la comunicazione tra livelli;
- il *presentation layer* contiene l'interfaccia grafica del modulo e le direttive finali per gestire il *whitelabeling*.

3.3.2 *Design patterns*

I *design patterns* o soluzioni architetturali forniscono soluzioni progettuali a problemi ricorrenti, garantendo la certezza di risolvere problemi ben definiti con un'implementazione corretta e coerente. Dopo l'analisi dei requisiti, ho studiato l'argomento in dettaglio e ho sviluppato una lista di soluzioni architetturali necessarie per implementare le funzionalità che avrebbero soddisfatto i relativi requisiti. Tali *design patterns* sono:

***Data Transfer Object* (DTO)** Il *design pattern* DTO è uno strumento utile per gestire il trasferimento efficiente dei dati tra le diverse parti di un'applicazione distribuita o tra sistemi differenti. Esso separa logica di *business* e dati stessi ridcendo il numero di chiamate remote e la quantità di dati trasmessi. Il suo scopo è rappresentare un oggetto utilizzato per il trasferimento dei dati e simularne il comportamento, fornendo metodi di serializzazione e deserializzazione per garantire che la richiesta sia in forma corretta per la parte che deve riceverla. Ulteriori vantaggi derivano dalla riduzione del traffico di rete, limitando le chiamate a quelle essenziali e dall'alta scalabilità che permette una migliore gestione dei dati in sistemi distribuiti, fornendo oggetti semplici e facilmente estendibili. Nel progetto di *stage*, ho utilizzato questo *pattern* ogni qual volta era necessario il trasferimento di dati tra il *backend* CWBI e l'applicazione *mobile*, ad esempio, la richiesta di trasferimento di una licenza nel *server* o la richiesta di *download* di un documento. Il funzionamento della soluzione può essere descritto come in figura 3.3.

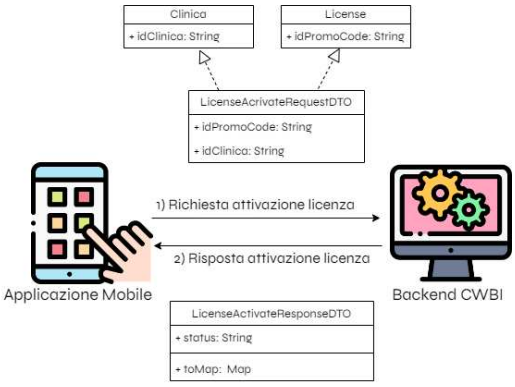


Figura 3.3: Funzionamento del *design pattern* DTO Fonte: flaticon.com

***Business Logic of Components* (BLoC)** Il *design pattern* BLoC è una soluzione architetturale principalmente utilizzata in applicazioni Flutter, e ha come scopo principale gestire lo stato e la relativa logica di *business* in modo organizzato e separato dalla parte dell'interfaccia utente, mantenendola reattiva ai cambiamenti. Uno schema

di come il flusso di lavoro è organizzato per gestire l'interazione tra *UI* e *backend* implementando la soluzione BLoC è espresso in figura 3.4.

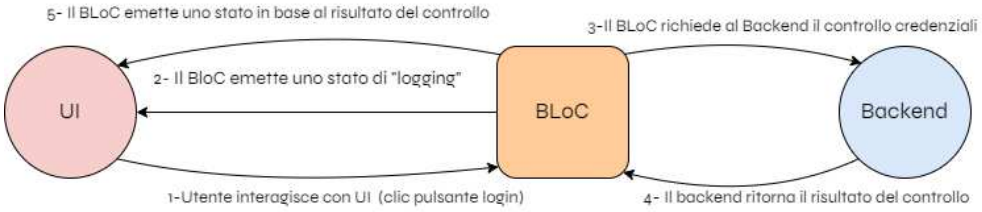


Figura 3.4: Funzionamento del *design pattern* BLoC

Repository pattern Il *design pattern Repository* è un modello ampiamente utilizzato nell'ambito del *software engineering*, fornendo un'astrazione tra il codice che accede ai dati e la sorgente dei dati stessi, consentendo una maggiore separazione tra la logica di business e l'accesso ai dati. In sostanza è composto da un'interfaccia che ne definisce le operazioni, tipicamente *Create Read Update Delete* (CRUD) e da una sua implementazione concreta dove si va ad implementare di quanto definito nell'interfaccia. L'ho utilizzato in quanto fortemente consigliato dalla comunità Flutter e dalla documentazione ufficiale riguardante la *Clean Architecture*.

3.3.3 *User Interface* (UI)

Ho studiato l'interfaccia utente e ho preso, assieme al collega, decisioni operative per migliorare l'esperienza utente durante l'utilizzo dell'applicazione prodotta dopo l'attività di progettazione. La modalità narrativa che ho scelto di adottare per descrivere la UI è cercare di focalizzare l'attenzione sulle motivazioni che hanno portato a compiere determinate scelte operative. descrivendo quindi, perchè ogni componente adottata è stata inserita nell'interfaccia e a quale scopo essa serva.

3.3.3.1 *Whitelabeling* e *login*

Per implementare la proprietà *whitelabel* dell'interfaccia grafica ho dovuto progettare e di conseguenza gestire, una regola facente parte della logica di *business* ovvero l'accesso all'applicazione, questo lavoro è stato svolto come citato in precedenza, in collaborazione con un altro tirocinante, e insieme abbiamo deciso che al primo avvio, l'applicazione caricherà un tema di *default*, una volta che l'utente avrà effettuato l'operazione di *login* per la prima voltà, sarà in grado di decidere le varie proprietà del tema personale (logo, colore primario, colore dello sfondo e colore di contrasto), che verranno salvate nella memoria del dispositivo e caricate all'avvio dell'applicazione sostituendo le proprietà del tema di *default*, conferendo al prodotto quindi la proprietà desiderata. La figura 3.5 esprime tale processo.

Figura 3.5: Processo di avvio dell'app con tema caricato da utente implementazione della proprietà *whitelabel*.

3.3.3.2 Sezionamento applicazione

Una volta effettuata l'operazione di *login* abbiamo diviso nettamente le funzionalità che ogni pagina doveva offrire, arrivando a progettare una schermata principale denominata **documenti** dove l'utente è in grado di utilizzare le funzionalità principali dell'applicativo (*step* di firma) e una sezione secondaria denominata ***settings*** dove poter gestire le funzionalità ausiliarie (proprietà del tema, gestione *account*, gestione licenza). La figura 3.6 mostra tali sezioni.

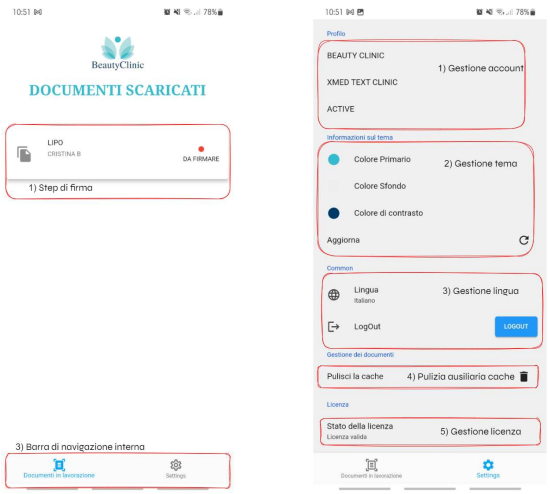


Figura 3.6: Pagine componenti dell'interfaccia grafica dell'applicazione.

3.4 Codifica

3.4.1 Ambiente di sviluppo e modalità d'integrazione

Ho scelto *Visual Studio Code* come ambiente di sviluppo, in quanto, è l' *Integrated Development Environment* con cui ho maggior esperienza. Ho sviluppato la componente per la gestione delle licenze in linguaggio Dart e mi sono integrato con il *backend* CWBI (scritto in Spring MVC) in un unico ambiente, rendendo il prodotto finale più *testabile* e affidabile. Il codice prodotto è composto da circa duemila righe di codice composte per l'85,4% da codice Dart, per il 14% da codice Kotlin e per il restante 0.6% da file di configurazione e codice Swift. Il Piano di Lavoro prevedeva come tempo totale da dedicare alle attività di codifica circa di centoventi ore. Sono risucito a completare l'attività nei tempi previsti e dalle percentuali riportate si evince chiaramente come la maggior parte del tempo è stato utilizzato per produrre la parte comune del sistema e la componente di gestione delle licenze in linguaggio Dart. Ho utilizzato il resto del tempo deidicato alla codifica sviluppando le componenti atte all'integrazione tra la componente Flutter e la componente nativa Android e iOS.

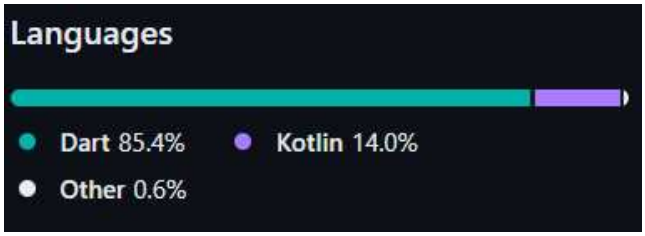


Figura 3.7: Codice prodotto durante l'attività di codifica Fonte: GitHub.

3.4.2 Metodologia di codifica

Durante l'attività di progettazione architetturale, abbiamo definito i confini generali e le macro funzionalità necessarie per l'intera applicazione, suddividendo il problema in componenti specifiche per risolvere parti specifiche del problema principale. Ho applicato pratiche apprese durante il corso di studi, che riflettono i principi del modello di sviluppo a V durante l'attività di progettazione architetturale. Il funzionamento interno delle componenti è stato definito in un'attività antecedenete alla codifica, chiamata progettazione di dettaglio. Tali pratiche mi hanno da un lato "forzato" a dover progettare oltre alle componenti dell'architettura (generale) e il loro funzionamento interno (dettaglio) anche delle operazioni che potessero verificare la correttezza, di quanto implementato. L'utilizzo di tale metodologia mi ha permesso di ottenere la correttezza per costruzione dei requisiti implementati e ha agevolato notevolmente l'attività di tracciamento dei requisiti. Ho implementato le varie funzionalità in modo sistematico, aprendo diverse *issues* nell'*Issue Tracking System* che, una volta completate, portavano a operazioni di integrazione nel *Version Control System* tramite il *Feature Branch*, mentre eventuali *bug* riscontrati sarebbero stati oggetto di discussione durante i successivi *meeting* d'allineamento con il *tutor* aziendale.

3.4.3 Integrazione con Namirial

Dopo aver sviluppato il substrato di codice comune ad entrambi, io e il secondo tirocinante ci siamo separati per lavorare indipendentemente ai relativi progetti di

stage e quindi il *focus* principale per me era le gestione delle licenze tramite l'integrazione di Namirial SDK all'interno della componente progettata. Dopo lo sviluppo del codice comune, io e il secondo tirocinante ci siamo separati per lavorare indipendentemente sui rispettivi progetti di *stage*. Il mio *focus* principale era la gestione delle licenze tramite l'integrazione di Namirial FEA SDK nella componente progettata. Grazie alla documentazione fornita da Namirial, non consultabile autonomamente poiché protetta da credenziali d'accesso, ho compreso il funzionamento interno dei servizi offerti dall'azienda. Ho tradotto questo flusso in logica interna nella mia componente, definendo tutti gli *end-point* necessari per interfacciarsi con i servizi esterni, consentendo l'interazione tra la nostra tecnologia e Namirial FEA SDK per la validazione delle licenze, utilizzando lo stesso procedimento usato per validare le firme sui documenti La figura 3.8 esprime il processo di codifica.

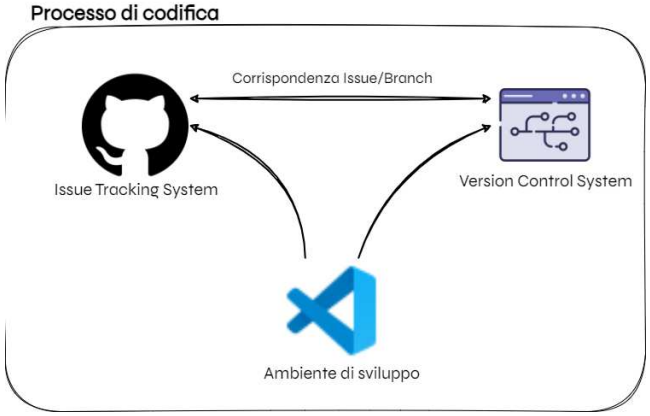


Figura 3.8: Flusso del processo di codifica.

3.5 Risultati ottenuti

3.5.1 *Performance* e gestione degli errori

Una volta terminata l'attività di codifica della componente atta alla gestione delle licenze sono passato all'attività di integrazione all'interno dell'applicazione *mobile* sviluppata in precedenza. Durante questa fase ho posto l'attenzione alle *performance* del sistema, ogni qual volta la componente *license* veniva chiamata. Il sistema infatti doveva rispondere in maniera sistematica alle richieste dell'utente, assicurandosi però un adeguato livello di sicurezza, ecco spiegata la presenza di una fase di autenticazione preliminare. Ho dovuto anche affrontare il problema dell'utilizzo efficiente delle risorse del sistema, cercando di ridurre al minimo il numero di chiamate che il sistema effettuava alla componente *backend*. Focalizzandomi su questi aspetti, sono riuscito ad ottimizzare il sistema, garantendo un livello di performance accettabile per applicazioni *enterprise* professionali. Avendo adottato una gestione degli stati sia all'interno dell'applicazione *mobile* sia nella componente *license*, la gestione degli errori è stata notevolmente facilitata, in quanto, ho modellato ogni possibile stato d'errore, cercando di rendere il sistema reattivo a tali situazioni mostrando all'utente opportune notifiche attraverso l'interfaccia grafica e adottando strategia risolutive interne all'applicazione.

3.5.2 Sicurezza

La sicurezza è stata un altro aspetto fondamentale del progetto di *stage*, in quanto lo scopo principale della componente che ho sviluppato è quello di verificare lo stato della licenza di un'utente, autorizzandolo in caso di esito positivo, ad effettuare gli *step* di firma del documento di consenso informato. Per garantire la non ripudiabilità delle firme apposte ai documenti infatti abbiamo prima implementato un meccanismo di autenticazione di primo livello degli utenti, ovvero tramite *username* e *password*, e solo in secondo luogo, dopo un controllo dello stato della licenza dell'utente, le funzionalità del sistema vengono rese disponibili. Le funzionalità messe a disposizione dal sistema sono: la firma dei documenti, il *download* e *upload* di essi e le funzionalità di *download* e rinnovo della licenza, esse rispecchiano quanto descritto in 2.3.1.

3.5.3 Documentazione

Durante ogni attività di progetto, ho dedicato delle ore lavorative alla redazione della documentazione del prodotto, producendo i documenti di Analisi dei Requisiti e Specifica Architetturale. Ho completamente commentato il codice prodotto, descrivendo il flusso di esecuzione e fornendo una traduzione letterale del comportamento di ogni istruzione, questa pratica ha reso il codice altamente leggibile e comprensibile. Tutta la documentazione prodotta ha seguito le metodologie e pratiche adottate da CWBI per la redazione dei propri documenti. La figura 3.9 schematizza il rapporto tra le ore spese in attività di sviluppo e le ore dedicate alla redazione della documentazione di progetto.

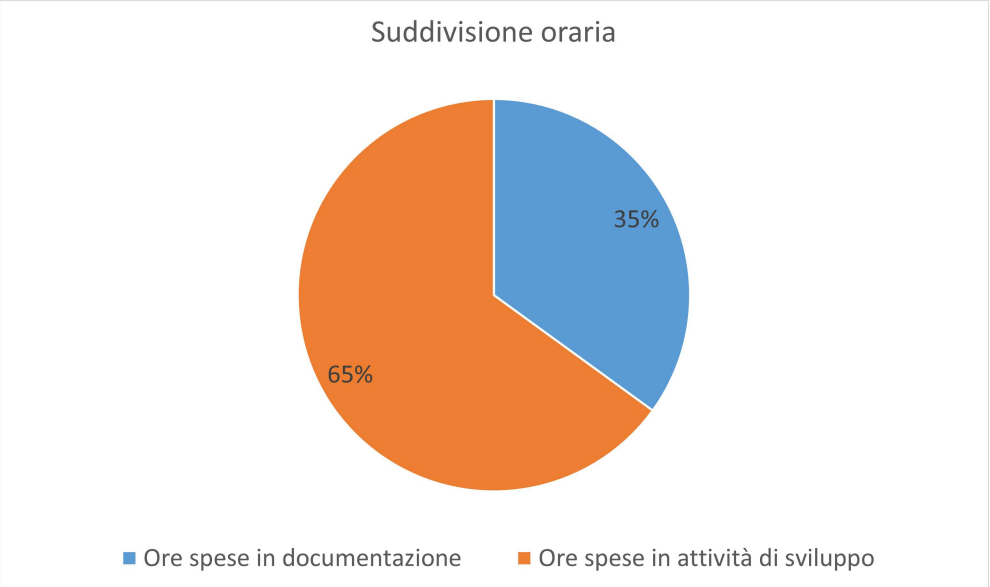


Figura 3.9: Rapporto tra attività di sviluppo e documentazione.

3.5.4 Requisiti soddisfatti

Al termine del percorso di *stage*, ho soddisfatto tutti i requisiti obbligatori e opzionali individuati durante l'attività di analisi attraverso relative implementazioni, mentre per questioni di tempo non sono riuscito a portare a termine gli obiettivi in 2.5.1 con rilevanza desiderabile. Ho eseguito manualmente i *test* progettati per verificare le implementazioni, poiché non esiste un ambiente di *test* automatico per il linguaggio Dart. L'adozione di questa pratica ha facilitato notevolmente il tracciamento dei requisiti. La figura 3.10 esprime la copertura dei requisiti attraverso un grafico.

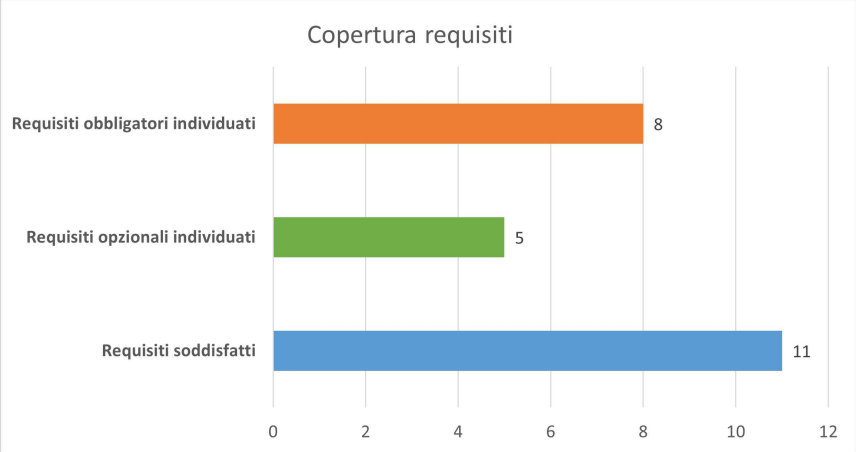


Figura 3.10: Grafico della copertura dei requisiti.

Capitolo 4

Valutazione retrospettiva

4.1 Raggiungimento degli obiettivi aziendali

4.1.1 Obiettivi dell'esperienza

Nella sezione 2.5.1, ho esposto gli obiettivi che CWBI aveva posto riguardo l'esperienza di *stage*. In questa sezione, discuto dello stato di avanzamento di tali obiettivi al termine dell'esperienza di tirocinio.

Obiettivo	Descrizione	Stato
Accesso al sistema	L'utente deve poter accedere al sistema	Soddisfatto.
Enrollment dispositivo	L'utente deve poter attivare la propria licenza	Soddisfatto.
Download licenza	L'utente deve poter effettuare il *download* della sua licenza	Soddisfatto.
Rinnovo licenza	L'utente deve poter rinnovare la propria licenza d'utilizzo	Soddisfatto.
Errore connessione	L'utente deve visualizzare un messaggio d'errore in caso stesse cercando di utilizzare un servizio finchè *offline*	Soddisfatto.
Integrazione Namirial SDK	Il sistema deve integrare Namirial FEA SDK	Soddisfatto.
Distribuzione app	Il sistema deve essere distribuito su tutti gli store disponibili	Soddisfatto.
Whitelabeling	L'interfaccia grafica dovrà rispettare i principi del *white label software*	Soddisfatto.

Tabella 4.1: Obiettivi aziendali raggiunti al termine del tirocinio.

La tabella 4.2 esprime gli obiettivi aziendali classificati come 'desiderabili', che non ho soddisfatto al termine dell'esperienza di tirocinio. Tali obiettivi non sono stati raggiunti in parte a causa del poco tempo disponibile e poichè il *tutor* ha preferito assegnarmi altre attività post-progetto.

Obiettivo	Descrizione	Stato
Storico licenze	Il sistema mette a disposizione lo storico delle licenze acquistate dall'utente	Non soddisfatto.
Temi preimpostati	L'utente può scegliere tra una serie di temi preimpostati	Non soddisfatto.

Tabella 4.2: Obiettivi aziendali non raggiunti al termine del tirocinio.

4.1.2 Attività post progetto

Una volta ultimati i lavori di progetto, il *tutor* stesso mi ha proposto di affiancare un *team* di sviluppo nella fase di progettazione e codifica e conseguente esecuzione di una *suite* di *test* d'accettazione per un progetto che prevedeva l'integrazione dei servizi PagoPA all'interno delle applicazioni aziendali. Ho anche avuto la possibilità di partecipare a un *meeting* con gli impiegati dell'azienda che ha realizzato il servizio. Il progetto era già stato realizzato, tali *test* d'accettazione andavano eseguiti nuovamente a seguito di un cambio di versione delle API interne del servizio PagoPA, ho trovato che tale scelta fosse finalizzata allo sviluppo di capacità che esulano dalle competenze tecnico informatiche necessarie a progettare la batteria di *test* in se, piuttosto il loro fine era quello di sviluppare capacità analitiche per riuscire a decifrare quando un bisogno dalla prospettiva utente, diventi necessariamente un requisito funzionale (a seguito di manipolazioni) e di come esso possa interagire con lo strato tecnologico, andando a creare i requisiti di vincolo, il tutto finalizzato alla progettazione e codifica di una *suite* di *test* che verificassero l'implementazione di tali requisiti.

4.2 Raggiungimento degli obiettivi personali

4.2.1 Obiettivi dell'esperienza

Voglio soffermarmi sono gli obiettivi di *stage* dal mio punto di vista, in quanto, come mostrato dalla sezione 3.5, il progetto si è concluso con successo con parecchia gratificazione da parte del *tutor* aziendale. Trovo però che gli obiettivi dell'esperienza in generale siano più ampi e debbano essere dimostrati sulla base di dati di fatto e non opinioni personali. Essi possono comprendere l'esplorazione di una possibile carriera futura e lo sviluppo di una mentalità più aperta e indirizzata verso il mondo del lavoro, oltre che ovviamente, l'acquisizione di competenze intrinseche al campo d'azione dell'azienda ospitante. Nella tabella 2.2, ho riportato una serie di obiettivi che mi ero prefissato prima di cominciare l'esperienza di tirocinio. Ora, facendo una retrospezione, valuteremo il loro grado di soddisfacimento.

Contesto produttivo

Al termine dell'esperienza di *stage*, posso dire di essermi inserito con successo nell'ambiente aziendale, anche grazie alla partecipazione a diversi *meeting* con clienti e aziende coinvolte nello sviluppo di altri progetti all'interno di CWBI. Avendo collaborato anche con un altro *team* oltre a quello costruito per il progetto di *stage*, posso dire di aver appreso come comunicare efficientemente sia la chiave per una risoluzione dei problemi mirata e sistematica, ricevendo apprezzamenti e riconoscimenti da vari sviluppatori all'interno dell'azienda stessa.

Competenze tecnologiche e abilità

Dopo aver vissuto l'esperienza di *stage* nella sua totalità, posso affermare di aver acquisito le competenze che mi ero prefissato. Trovo infatti che le mie abilità di analisi e progettazione siano notevolmente migliorate, avendo poi imparato un nuovo linguaggio di programmazione utilizzando il *framework* Flutter posso considerare il mio bagaglio culturale ampiamente arricchito. Ho avuto modo poi di esplorare il linguaggio Java e il suo utilizzo per un altro prodotto a marchio CWBI, permettendomi di studiarne la conformazione e le motivazioni dietro le scelte architetturali, supportati dalle varie *best practises* adottate. Trovo che tali abilità siano notevolmente migliorate, in quanto, tutto il lavoro svolto durante e dopo il progetto era finalizzato alla presa di decisioni consapevoli, andando poi a valutare attentamente quali potevano essere le ripercussioni sulle *performance* o sul funzionamento del sistema, date dalle scelte prese in precedenza.

Autonomia e responsabilità

Personalmente credo che il *tutor* abbia dimostrato il valore effettivo del lavoro svolto dal sottoscritto richiedendo la mia presenza in un progetto totalmente differente da quello di *stage*, permettendomi inoltre di prendere visione della documentazione ufficiale e partecipare ai *meeting* aziendali. Ho interpretato questo aumento di responsabilità come uno stimolo che il *tutor* mi ha voluto somministrare, per poter maturare ulteriormente in vista di un futuro impiego nel mondo dell'informatica, andando a simulare la dinamicità davanti alla quale ci si può trovare nella vita di tutti i giorni, tale decisione rappresenta per me un segno di riconoscenza nei miei confronti e nei confronti del mio lavoro.

Mentoring

Sin dall'inizio del periodo di tirocinio, abbiamo imparato a interagire principalmente con i colleghi adiacenti, piuttosto che rivolgerci al *tutor* in persona per i problemi minori, qualora non riuscissimo a risolverli in autonomia. Non trovo sbagliata tale decisione in quanto permette al tirocinante di relazionarsi con più persone, avere conversazioni con multipli scambi di idee e nel mio caso, essendo il personale vicino alla mia età, fanno legare maggiormente le persone. Mi sono sentito quasi un membro dell'azienda, soprattutto durante l'incarico successivo al progetto, dove mi sono inserito in un vero e proprio *team* di sviluppo, gli sviluppatori hanno condiviso con me la loro conoscenza, mi hanno dato molti consigli strategici e attraverso la loro esperienza di vita ho potuto avere degli elementi in più per la scelta del mio futuro.

Coinvolgimento e interesse

È chiaro dalla scelta di effettuare il tirocinio presso CWBI sia sintomo di un forte interesse verso la realtà e l'ambito aziendale, il punto su cui voglio focalizzarmi ora è, a posteriori, io ho gradito molto l'esperienza lavorativa, sono molto contento del lavoro svolto e sono sicuro che in un futuro, userò quanto appreso durante questa esperienza, il che mi porta a dire che il coinvolgimento è stato soddisfacente durante quasi tutta l'esperienza di *stage*. A valle dell'esperienza vissuta posso dire che il mondo *enterprise* mi continua ad affascinare molto e sono soddisfatto della scelta effettuata.

Competenze tecniche

Visto dal lato aziendale il tirocinio non è altro che un'attività di formazione a lungo termine che l'azienda si prende in carico di effettuare con un possibile ritorno futuro sull'investimento di tempo, andando ad assumere il tirocinante o fargli svolgere incarichi utili all'azienda in se. Trovo che CWBI, nell'esperienza di tirocinio generale (quindi esulando dall'esperienza personale) investa molto tempo nella formazione dei propri tirocinanti, offrendo corsi per sviluppare negli attendenti le competenze tecniche necessarie e seminari di più giorni per cercare di rendere quanto più affiatati i gruppi coinvolti per poter far sviluppare più velocemente le *soft skills* ed una mentalità adatta all'ambiente lavorativo nei candidati.

Competenze trasversali

Credo che la parte più importante dell'esperienza di tirocinio sia quella di far apprendere allo stagista le cosiddette competenze trasversali, le quali appunto sono quelle che permettono di unire i rapporti sociali nell'ambiente aziendale al lavoro vero e proprio, essi possono esprimersi in vari modi all'interno dell'azienda e spesso, l'organizzazione dell'azienda è riflessa nei processi organizzativi che governano la gestione del lavoro. Dopo aver effettuato l'esperienza posso dire che le mie abilità comunicative sono migliorate molto, avendo partecipato ad alcuni *meeting* con altre aziende durante l'incarico post progetto comunicando con altri *team* tramite *mail* o *Issue Tracking System*. La mia abilità di *problem solving* è stata notevolmente affinata anche grazie all'intervento di qualche collega che non appena perdevo un attimo il *focus* del problema mi riportava alla giusta distanza per poter avere un quadro chiaro di cosa si dovesse risolvere, una volta fatta questa operazione sono riuscito a portare a termine i lavori abbastanza autonomamente.

4.3 Distanza teoria-pratica

L'esperienza di *stage* è stato anche il luogo dove ho messo in pratica quanto appreso durante il corso di studi in Informatica, di conseguenza, ho potuto constatare quanto diversa sia l'esperienza richiesta in un possibile posto di lavoro. Devo ammettere che inizialmente ho trovato una curva di difficoltà non indifferente, in quanto, è stata la mia prima esperienza lavorativa in campo informatico. Una volta fatta l'abitudine con gli orari lavorativi e aver appreso le conoscenze fondamentali per l'avanzamento del progetto, la curva di difficoltà è scesa presentandomi ogni giorno nuove "sfide" minori da risolvere, sono riuscito, sfide che tramite le conoscenze apprese sono riuscito a portare a termine con successo. Il discorso cambia totalmente quando si tratta delle attività post-progetto, in quanto in questo caso, lo studio approfondito della

documentazione e del codice prodotto è stato fondamentale, tali attività hanno richiesto capacità che avevo già appreso grazie al corso di Ingegneria del *software* del corso di studi. Trovo che quanto studiato durante il percorso di laurea sia totalmente in linea, con quanto richiesto dal mondo del lavoro odierno, sebbene l'ambito di studi cambi ripetutamente, il corpo docente è sempre in grado di stare all'avanguardia con i propri insegnamenti. In conclusione, ho notato come ci sia un evidente *gap* tra la specificità delle conoscenze richieste dal mondo del lavoro e quelle erogate dal corso di studi, nonostante ciò, ho notato come l'obiettivo principale del corso di studi fosse fornire agli studenti un "metodo di lavoro" che segua le *best practises* in ogni attività che riguarda un progetto. Ho notato come tale approccio non sempre fosse la prassi per alcuni componenti del *team* di sviluppo, e trovo che qui sia palese il valore aggiunto dato dall'aver frequentato l'Università.

Acronimi e Abbreviazioni

BLoC *Business Logic of Components*. vi, 25, 26

CEO *Central Executive Officer*. 1, 2, 4, 13

CRUD *Create Read Update Delete*. 26

CWBI *Codice Web Banking Innovation*. ii, v–vii, 1–8, 13, 15–17, 25, 28, 30, 32, 34, 35

DTO *Data Transfer Object*. vi, 25

ECQ Ente di Certificazione Qualificato. 10

eIDAS Regolamento Europeo per l'Identificazione Elettronica. 9

FEA Firma Elettronica Avanzata. iv, 9, 14, 19–21, 29, 32

FEQ Firma Elettronica Qualificata. iv, 10

FES Firma Elettronica Semplice. iv, 9

MDA *Model Driven Architecture*. 4

MVC *Model View Controller*. 2, 28

REST *Representational State Transfer*. 2

SDK *Software Development Kit*. 14, 15, 19–21, 29, 32

UI *User Interface*. v, 26

UML *Unified Modeling Language*. 4

VM *Virtual Machine*. 7

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.. 3, 5, 13, 16, 33, 38

Feature Branch approccio di sviluppo *software* in cui ogni nuova funzionalità viene sviluppata in un *branch* (ramo) separato all'interno del VCS. Questo permette agli sviluppatori di lavorare in isolamento sulla nuova funzionalità senza influenzare il codice del ramo principale. Una volta completata la feature, il nuovo *branch* viene integrato nel principale tramite una *pull request* o un *merge*.. 28, 38

Integrated Development Environment ambiente *software* completo utilizzato dai programmatori per scrivere, testare e sviluppare applicazioni *software*. Fornisce un insieme di strumenti integrati che comprendono un *editor* di codice, un compilatore, un *debugger* e spesso funzionalità aggiuntive come la gestione dei progetti, il controllo di versione e suggerimenti di codice. Disporre di un ambiente del genere semplifica il processo di sviluppo, migliorando la produttività e fornendo un ambiente coeso per la scrittura del codice e la gestione del ciclo di vita dell'applicazione.. 28, 38

Issue Tracking System applicazione *software* che consente alle organizzazioni di registrare, monitorare e gestire in modo efficiente i problemi, i *bug*, le richieste di assistenza e le attività correlate ai progetti. Questi sistemi forniscono un meccanismo per la registrazione delle problematiche, l'assegnazione a responsabili, il monitoraggio dello stato di avanzamento e la generazione di *report* per garantire una gestione efficace delle questioni aperte.. 28, 35, 38

Layered Architecture approccio di progettazione del *software* che organizza l'applicazione in diversi livelli o strati, ognuno dei quali ha responsabilità specifiche e comunica con i livelli adiacenti attraverso interfacce ben definite. L'obiettivo principale è separare le preoccupazioni, migliorare la modularità e facilitare la manutenibilità del sistema.. 22, 38

modello di sviluppo a V approccio al ciclo di sviluppo del software che enfatizza la parallelità tra le attività di sviluppo e di test, la forma a "V" rappresenta la struttura del modello, in cui le attività di sviluppo procedono dalla parte sinistra della "V" (analisi, progettazione, codifica) e le attività di *test* avanzano dalla parte destra (verifica, validazione). Questo modello mira a garantire che ogni

fase di sviluppo abbia una corrispondente fase di *test*, consentendo il rilevamento tempestivo e relativa correzione degli errori.. 28, 38

PagoPA PagoPA è un servizio di pagamento elettronico introdotto in Italia per semplificare le transazioni finanziarie tra cittadini, imprese e enti pubblici. Consente di effettuare pagamenti online in modo sicuro e conveniente per servizi pubblici, tasse, multe e altre spese governative, contribuendo a ridurre la burocrazia e migliorare l'efficienza delle transazioni finanziarie in tutto il paese.. 33, 39

Version Control System *software* utilizzato dagli sviluppatori per gestire e tracciare le modifiche apportate al codice sorgente di un progetto. Consente di tenere traccia delle diverse versioni dei *file*, monitorare le modifiche, consentire il lavoro collaborativo da parte di più sviluppatori e, in caso di problemi, ripristinare le versioni precedenti del codice o risolvere i conflitti.. 28, 39