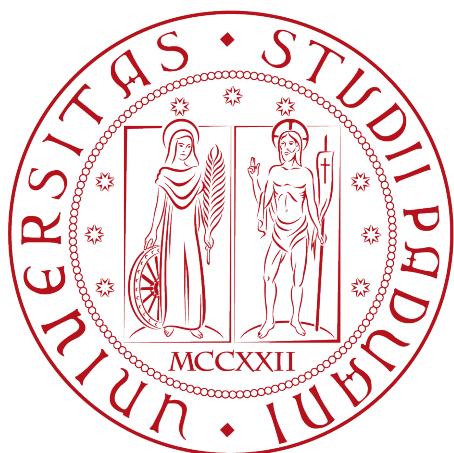


Università degli Studi di Padova
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
CORSO DI LAUREA IN INFORMATICA



**Migrazione a dispositivo mobile di una
applicazione web tramite tecnologia Ionic**

Tesi di laurea

Relatore

Prof.Tullio Vardanega

Laureando

Enrico Buratto

ANNO ACCADEMICO 2019-2020

Enrico Buratto: *Migrazione a dispositivo mobile di una applicazione web tramite tecnologia Ionic*, Tesi di laurea, © Dicembre 2020.

Sommario

Il seguente documento descrive il tirocinio che ho svolto presso la sede di Padova dell'azienda **Sync Lab s.r.l.**, d'ora in poi "*Sync Lab*" o "*l'azienda*", nel periodo che va dal 31/08/2020 al 23/10/2020. Lo stage ha avuto una durata complessiva di circa **300 ore**, ed è stato supervisionato dal mio tutor esterno, Andrea Giunta, e dal mio relatore, prof. Tullio Vardanega.

Lo scopo dello stage è stato quello di studiare e utilizzare il *framework Ionic*, tecnologia che permette la realizzazione di applicazioni *mobile* ibride a partire da codice *JavaScript* o *TypeScript*, per il **porting** di un applicativo web preesistente su dispositivi mobili. La migrazione verso dispositivi mobili che ho effettuato si è basata sulla componente *front-end* dell'applicativo *SyncTrace*, sviluppata da altri studenti durante i precedenti tirocini e completata ufficialmente il 28/08/2020; questa componente è stata sviluppata tramite il *framework Angular* in accoppiata al linguaggio *TypeScript*.

L'**obiettivo** di questo applicativo è inquadrabile nell'ambito del *contact tracing* per la prevenzione di infezione da *Sars-CoV2*: esso permette infatti il monitoraggio di persone infette, e a rischio infezione, da parte di medici ed esercenti.

Il presente documento è suddiviso nei seguenti capitoli:

- **Capitolo 1:** presentazione dell'azienda, comprendente i processi messi in atto dall'azienda per soddisfare il cliente, la metodologia di sviluppo, le tecnologie di interesse e un approfondimento sulla propensione dell'azienda all'innovazione;
- **Capitolo 2:** descrizione della proposta di *stage*, comprendente gli obiettivi e i vincoli a cui mi sono dovuto attenere e la motivazione della scelta. Describe inoltre la fase iniziale del mio tirocinio, corrispondente alla formazione sulle tecnologie e sul precedente progetto, punto di partenza del mio lavoro;
- **Capitolo 3:** descrizione dettagliata del progetto di *stage*, ossia delle attività di sviluppo, verifica, validazione e collaudo che ho effettuato per ottenere il prodotto finale;
- **Capitolo 4:** valutazione retrospettiva dello *stage*, contenente il soddisfacimento degli obiettivi e le competenze professionali che ho maturato.

In questo documento ho utilizzato le seguenti **convenzioni tipografiche**:

- I termini in lingua straniera e i termini propri sono evidenziati dall'utilizzo del corsivo;
- Piccole sezioni di testo possono essere riportate in grassetto, al fine di evidenziarne l'importanza nel discorso;
- Tutte le tavole e le immagini riportano un numero progressivo, una descrizione e, se provengono da una risorsa esterna, la fonte.

*“Boschi ed acque, venti ed alberi, saggezza, forza e cortesia, che il favore della giungla
ti accompagni.”*

— Rudyard Kipling

Ringraziamenti

Padova, Dicembre 2020

Enrico Buratto

Indice

1	Contesto aziendale	1
1.1	Introduzione	1
1.2	L'azienda	1
1.3	Processi aziendali	3
1.3.1	Processi	3
1.3.2	Metodologia Agile	4
1.4	Tecnologie utilizzate	5
1.5	Propensione all'innovazione	9
2	Il progetto nel contesto aziendale	11
2.1	Il rapporto tra stage e azienda	11
2.2	L'azienda in relazione al contesto attuale	12
2.3	Lo scopo dello stage	12
2.4	Vincoli e obiettivi dello stage	14
2.4.1	Vincoli temporali	14
2.4.2	Vincoli organizzativi	15
2.4.3	Vincoli tecnologici	16
2.4.4	Obiettivi	17
2.5	Motivazione della scelta	18
2.6	Formazione	19
2.6.1	Tecnologie	19
2.6.2	Progetto	20
3	Il progetto di stage	23
3.1	Analisi dei requisiti	23
3.1.1	Casi d'uso	23
3.1.2	Requisiti	26
3.2	Progettazione	27
3.3	Codifica	30
3.4	Verifica	34
3.4.1	Analisi statica	34
3.4.2	Test di unità	35
3.5	Validazione e collaudo	36
3.6	Risultati ottenuti	38
4	Valutazioni retrospettive	41
4.1	Soddisfacimento degli obiettivi	41
4.2	Bilancio formativo	42

4.2.1	Maturazione professionale	42
4.2.2	Rapporto tra università e lavoro	42

Bibliografia	43
---------------------	-----------

Elenco delle figure

1.1	Moduli di SynClinic.	2
1.2	Metodologia Scrum.	4
1.3	Panoramica delle tecnologie utilizzate da Sync Lab.	6
1.4	Architettura dello <i>Spring framework</i>	7
1.5	Pattern MVVM, adottato anche dal <i>framework Angular</i>	8
1.6	Funzionamento del <i>framework Electron</i> con <i>NodeJS</i>	8
1.7	Alcune università con le quali l'azienda collabora.	9
1.8	I progetti di ricerca e sviluppo a cui Sync Lab collabora.	10
2.1	Rappresentazione schematica del <i>contact tracing</i>	13
2.2	Immagine di cattura di schermo iniziale dell'applicazione <i>SyncTrace</i> . .	13
2.3	Immagine di cattura di schermo iniziale della <i>webapp</i> per, rispettivamente, medici ed esercenti.	14
2.4	Porzione del registro delle attività.	16
2.5	Schema rappresentante le relazioni che sussistono tra le tecnologie descritte in SyncTrace.	20
3.1	Attori principali.	24
3.2	Diagramma dei casi d'uso per un utente non autenticato.	24
3.3	Diagramma dei casi d'uso per un utente autenticato come medico. .	25
3.4	Diagramma dei casi d'uso per un utente autenticato come esercente. .	26
3.5	Esempio di <i>sketch</i> dell'applicazione realizzato con <i>WireframeSketcher</i> .	28
3.6	Esempio di funzionalità proprie di <i>Ionic</i>	29
3.7	Porzioni di <i>directory</i> contenente un progetto <i>Ionic</i>	31
3.8	In ordine: pagina di benvenuto, <i>login</i> e registrazione.	31
3.9	In ordine: lettura del QR a fotocamera spenta, lettura del QR a fotocamera accesa e inserimento manuale.	32
3.10	Maschera delle impostazioni dell'utente.	32
3.11	In ordine: statistiche del sistema, gestione degli infetti e gestione dei contatti.	33
3.12	Esempi di, rispettivamente, <i>toast</i> e <i>alert</i>	33
3.13	Risultato dell'esecuzione del <i>linter</i> sul codice da me prodotto.	34
3.14	Esempio di test di unità <i>TypeScript</i>	35
3.15	Porzione dell'output di <i>Jasmine</i> per il corrente progetto.	36
3.16	Esempio di utilizzo di <i>ionic serve</i> su <i>browser Chromium</i>	37
3.17	Porzione del <i>docker-compose</i> per i servizi del <i>back-end</i>	37
3.18	Esempio di commento a un metodo per la generazione automatica di documentazione con <i>Compodoc</i>	38

Elenco delle tabelle

2.1	Suddivisione settimanale delle attività	15
2.2	Obiettivi di progetto originali	18
2.3	Ulteriorie obiettivo, aggiunto in corso di progetto.	18
3.1	Totale dei requisiti identificati.	27
3.2	Numero indicativo di <i>SLOC</i> del prodotto che ho sviluppato.	33
3.3	Requisiti soddisfatti e non soddisfatti.	38
4.1	Grado di soddisfacimento degli obiettivi di <i>stage</i>	41

Capitolo 1

Contesto aziendale

1.1 Introduzione

In questo documento descrivo la mia esperienza di *stage* presso la sede di Padova di **Sync Lab**, un'azienda nata come *software house* e che, negli anni, si è affermata nel campo dell'*ICT (Information and Comunication Technology)*.

È di particolare rilievo, nella mia particolare esperienza, il periodo storico in cui questo tirocinio è avvenuto: ho infatti effettuato lo stage tra il mese di settembre e il mese di ottobre 2020, ossia durante l'emergenza sanitaria globale dovuta alla malattia *COVID-19*. La pandemia in atto ha influenzato radicalmente il lavoro di molte persone, me compreso: al fine di ridurre le possibilità di contagio, infatti, ho dovuto svolgere gran parte del tirocinio in regime di *smart working*, potendo accedere alla struttura solo una volta alla settimana, e alle infrastrutture quasi esclusivamente in maniera remota. Nonostante la situazione non favorevole, ho avuto comunque modo di interfacciarmi con i dipendenti dell'azienda; sono quindi riuscito anche a raccogliere le informazioni riguardanti l'azienda che, unite a delle ricerche online, ho riportato in questo capitolo.

1.2 L'azienda

Sync Lab nasce come *software house* a Napoli nel 2002; negli anni, l'azienda si espande a gran velocità, aprendo sedi a Roma, Milano, Padova e Verona. Al giorno d'oggi, l'azienda conta 5 sedi, per un totale di oltre 250 dipendenti e più di 150 clienti diretti e finali.

Dall'apertura ad oggi, Sync Lab si è tramutata in un *system integrator* grazie alla maturazione delle competenze tecniche e metodologiche in ambito software. Un tratto distintivo dell'azienda è la grande attenzione posta alla gestione delle **risorse umane**: testimonianza di ciò è il basso *turn-over*, segno che i collaboratori condividono un progetto comune e concreto.

Altro segno d'eccellenza sono le certificazioni di qualità che l'azienda ha conseguito; finora, infatti, l'azienda ha ottenuto le certificazioni per gli standard *ISO 9001*, *ISO 14001*, *ISO 27001* e *ISO 45001*.

Prodotti e servizi offerti

Una certa attenzione è posta dall'azienda alla diversificazione dei prodotti e dei servizi offerti; essi sono infatti inquadrabili in diverse aree tematiche, quali salute, *privacy*, sicurezza, telecomunicazioni, finanza, territorio e ambiente.

Alcuni dei prodotti che l'azienda offre al momento sono i seguenti:

- **DPS 4.0:** software per la gestione degli adempimenti alla *Privacy GDPR - General Data Protection Regulation*, utilizzato da svariate aziende per attuare correttamente quanto previsto da tale regolamento europeo; tale prodotto permette di censire, tracciare e controllare chi può trattare dati personali in azienda;
- **StreamLog:** sistema finalizzato al soddisfacimento dei requisiti fissati dal *Garante per la Protezione dei dati personali*, utilizzato dagli amministratori di sistema per controllare gli accessi agli utenti al fine di soddisfare i requisiti di *privacy* richiesti dal garante;
- **SynClinic:** sistema informativo per la gestione integrata dei processi, clinici e amministrativi, di ospedali, cliniche e case di cura. Questo applicativo fornisce svariate funzionalità organizzate in moduli, elencati nell'immagine seguente, che intersecano i bisogni del personale amministrativo e quelli del personale clinico delle strutture sanitarie, permettendo di gestire e monitorare tutte le fasi del percorso di cura del paziente. È utilizzabile sia in *cloud* che *on premises*;



Figura 1.1: Moduli di SynClinic.

Fonte: synclinic.it

- **StreamCrusher:** tecnologia che aiuta le aziende a effettuare corrette decisioni di *business*, a identificare tempestivamente eventuali criticità e a riorganizzare i processi in base a nuove esigenze. Questo software è in grado di raccogliere, indicizzare e interpretare i dati, siano essi *log* di applicazione o di sistema, *alert*, dati di configurazione o modifiche ai sistemi, al fine di estrarre informazioni utili all'*IT management*;
- **Wave:** software che si propone come integrazione tra il mondo della videosorveglianza e quello dei *Sistemi Informativi Territoriali*, permettendo di avere

una visione geo-referenziata della distribuzione delle telecamere installate sul territorio e consentendo così all'utilizzatore di avere un impatto visivo immediato sull'area di copertura di una data installazione reale, o di avere un'anteprima di tale area in fase di progettazione;

- **Seastream:** piattaforma pensata per migliorare l'efficienza, la sicurezza e il processo di innovazione del settore marittimo; per fare ciò l'azienda fornisce, attraverso questa piattaforma, un *Fleet Operation Center (FOC)*, ovvero un sistema di monitoraggio avanzato delle flotte armatoriali operative in tutto il mondo, e un *Harbor Operation Platform (HOC)*, ovvero una piattaforma di servizi per gli operatori portuali.

Clienti principali

Sync Lab collabora con numerose aziende italiane e multinazionali, sia pubbliche che private. Tra le aziende **private** più importanti possiamo trovare *Sky*, *Eni*, *Enel*, *Vodafone*, *Accenture*, *Fastweb*, *Tim*, *UniCredit* e *H&M*.

Tra le collaborazioni con **enti statali** e parastatali, invece, troviamo quelle con *Trenitalia*, *RAI*, *Poste Italiane*, la *Regione Lazio* e il *Ministero dell'Economia e delle Finanze*.

1.3 Processi aziendali

1.3.1 Processi

L'azienda persegue i propri obiettivi attuando i processi qui elencati.

Consulenza

L'azienda fornisce servizi di consulenza informatica a svariate imprese, sia pubbliche che private; questi servizi hanno lo scopo di far evolvere, sia in termini di sviluppo che di competitività, i clienti dell'azienda. Per fare ciò, Sync Lab collabora con altre aziende di consulenza e con specialisti del settore.

Fornitura

Il processo di fornitura viene istanziato ognqualvolta un cliente assume Sync Lab per lo sviluppo e la realizzazione di un prodotto. Contemporaneamente alla realizzazione, l'azienda svolge delle attività che possano migliorare questo processo. In particolare:

- **Qualità del software:** il software viene controllato e ottimizzato, anche attraverso l'utilizzo delle *best practice*, al fine di aderire alle regole aziendali;
- **Verifica delle procedure:** le procedure vengono verificate, al fine di poter agire in maniera correttiva nel caso in cui si dovessero manifestare dei problemi;
- **Analisi e miglioramento degli standard:** gli standard aziendali vengono analizzati e, possibilmente, migliorati; conseguenza di ciò è un costante miglioramento anche dal punto di vista di qualità del software.

Sviluppo

Per quanto riguarda il processo di sviluppo, Sync Lab fa uso della metodologia *Agile*¹, descritta più dettagliatamente in seguito. Questa permette di coinvolgere il cliente durante tutto il processo, tenendolo aggiornato sull'evoluzione dello sviluppo del prodotto e ricevendo di ritorno le sue opinioni, al fine di poter agire sia correttivamente che migliorativamente.

Manutenzione

Una volta consegnato il prodotto, l'azienda assicura le attività di manutenzione per tutto il ciclo di vita del software. La manutenzione offerta dall'azienda è di tre tipi:

- **Correttiva**, corrispondente alla correzione di eventuali difetti;
- **Adattiva**, ossia il riadattamento del software a nuovi requisiti quali l'ambiente di produzione o l'architettura;
- **Evolutiva**, ossia l'aggiunta o l'aggiornamento in senso migliorativo di porzioni di software.

1.3.2 Metodologia Agile

Per il processo di sviluppo, l'azienda fa uso di una metodologia *Agile* che molto si avvicina al modello *Scrum*². Punto cardine del metodo di sviluppo di Sync Lab, infatti, è la continua interazione con gli *stakeholder*, ossia i clienti: questi vengono coinvolti durante tutto il processo, venendo aggiornati sull'evoluzione del prodotto; questo permette all'azienda di ricevere *feedback* che possono aiutare a migliorare il prodotto e ad adattarlo al meglio alle esigenze.

Come ho potuto constatare di persona durante il mio periodo di tirocinio, il modello adottato dall'azienda prevede uno sviluppo che procede per *sprint*, ossia un'unità di base di durata fissa compresa tra una e quattro settimane a seconda degli obiettivi posti. A ogni *sprint* corrisponde una funzionalità; questa viene verificata insieme al cliente, per tastarne la soddisfazione o ricevere consigli che possano migliorare tale nuova funzionalità.

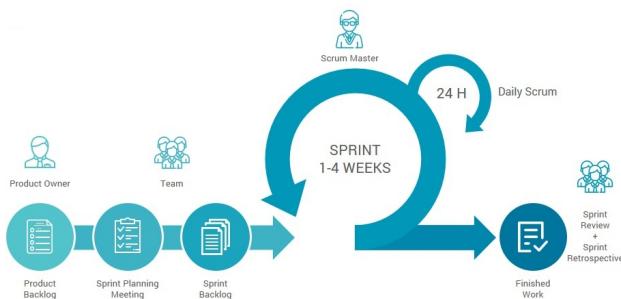


Figura 1.2: Metodologia Scrum.

Fonte: antevenio.com

¹Ken Schwaber et al. Kent Beck Jeff Sutherland. *Manifesto for Agile Software Development*. URL: <https://agilemanifesto.org/>.

²Metodologia SCRUM. URL: <https://www.scrum.org/>.

Lo sviluppo di un prodotto in Sync Lab passa attraverso cinque attività, inquadrabili tutte in quanto previsto dalla metodologia *Scrum* riassunta dalla precedente immagine. La prima attività che viene svolta è la redazione di una lista di cose da fare per portare a termine il progetto; nel caso del mio progetto di *stage*, ad esempio, ho definito insieme al tutor aziendale le *feature* che avrei dovuto mettere a disposizione, e abbiamo visionato assieme i *bug* presenti nel codice che avrei dovuto utilizzare come *baseline*, al fine di sapere cosa avrei dovuto correggere di quanto già fatto. Nella metodologia *Scrum*, questa lista assume il nome di **Product Backlog**.

Dopo aver redatto questa lista, solitamente viene realizzata una pianificazione degli *sprint* che è necessario effettuare per portare a termine il progetto. In ambito *Scrum* questa pianificazione viene chiamata **Sprint Planning**, e nel caso del mio tirocinio è corrisposta alla divisione in incrementi che avrei dovuto effettuare per completare il piano di lavoro. Successivamente a questo, vengono individuati dei sottoinsiemi di obiettivi per ogni *sprint* definito durante l'attività chiamata **Sprint Backlog**. Un esempio pratico di questa attività è la definizione degli incrementi che ho effettuato prima di cominciare lo *stage*.

Una volta definiti tutti gli obiettivi e aver correttamente diviso le tempistiche in **Sprint**, questi vengono ad uno ad uno eseguiti; nell'ambito del mio *stage*, questo è corrisposto allo sviluppo di quanto previsto dal piano di progetto. All'esecuzione di ogni incremento, inoltre, di questo è stata verificata l'aderenza agli obiettivi posti, che nella metodologia *Scrum* vengono chiamati **Sprint Goal**.

Durante il mio *stage* ho inoltre potuto sperimentare anche due degli eventi facenti parte della metodologia *Scrum*, ossia il *Daily Scrum* e lo *Sprint Review*:

- Il **Daily Scrum**, come definito dai creatori di questa metodologia, è un evento a cadenza giornaliera in cui i membri del gruppo di lavoro discutono dell'andamento dello *sprint*. Questo evento viene svolto dai diversi *team* di lavoro dell'azienda giornalmente; nel caso specifico del mio tirocinio, essendosi questo svolto in gran parte in *remote working*, questo evento non ha potuto avere cadenza giornaliera. Nonostante questo, ho effettuato svariati allineamenti con il tutor aziendale, e questo si può almeno in parte configurare con quanto previsto dalla metodologia;
- Lo **Sprint Review**, ossia la verifica di quanto effettuato durante uno *sprint* alla fine di questo, viene effettuato da tutti i gruppi di lavoro al raggiungimento degli obiettivi fissati dallo *sprint*. Nel caso specifico del mio tirocinio, questo evento ha avuto luogo quasi settimanalmente, con la verifica di quanto fatto durante l'incremento definito.

1.4 Tecnologie utilizzate

Come riportato sul sito aziendale, Sync Lab fa uso di diversi linguaggi di programmazione, *framework* e strumenti di supporto moderni e funzionali, al fine di soddisfare i clienti; oltre a questo, l'azienda è costantemente aggiornata sulle tecnologie di riferimento e pronta ad espandere le proprie conoscenze con le tecnologie più moderne.



Figura 1.3: Panoramica delle tecnologie utilizzate da Sync Lab.

Fonte: synclab.it

Tra i linguaggi di programmazione più utilizzati troviamo i seguenti:

- **Java³**: linguaggio di programmazione ad alto livello orientato agli oggetti, largamente utilizzato dalle aziende; in particolare, *Java* viene utilizzato da Sync Lab, in accoppiata al *framework Spring⁴*, per lo sviluppo dei servizi *REST* necessari alle componenti *back-end* di svariati applicativi, tra cui ad esempio *SynClinic* e *StreamCrusher*. Questo linguaggio, abbinato al *framework Spring*, è stato utilizzato anche per lo sviluppo della componente *back-end* dell'applicativo *SyncTrace*, oggetto del mio tirocinio;
- **JavaScript⁵**: linguaggio di programmazione orientato agli oggetti e agli eventi, originariamente pensato per la creazione di effetti dinamici interattivi per i siti web ma sempre più utilizzato come linguaggio *general purpose* per lo sviluppo di applicativi web e non web. Viene utilizzato dall'azienda per realizzare la componente logica delle *single page application*;
- **TypeScript⁶**: *super-set* di *JavaScript* sviluppato da *Microsoft*. Questo linguaggio estende la sintassi di *JavaScript* in modo che ogni programma scritto in *JavaScript* possa funzionare anche con *TypeScript*, e come il primo viene utilizzato dall'azienda per realizzare la componente logica delle *single page application*. Si può trovare questo linguaggio, abbinato al *framework Angular⁷*, in svariati prodotti dell'azienda, tra cui *SynClinic* e *DPS4.0*; è inoltre il linguaggio con il quale è stata scritta la componente *front-end* della *web application* di *SyncTrace*;
- **HTML5 e CSS3⁸**: linguaggi di *markup* utilizzati, anche dall'azienda, per modellare la componente visiva delle *single page application* e, più in generale, dei siti web. Questi linguaggi di *markup* sono usati dall'azienda principalmente nei progetti che coinvolgono il *framework Angular*, come ad esempio *SynClinic* e *DPS4.0*;

³Oracle Corporation. *Java*. URL: <https://docs.oracle.com/javase/7/docs/technotes/guides/language/>.

⁴*Spring Framework*. URL: <https://spring.io/>.

⁵Mozilla Foundation. *JavaScript*. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

⁶*TypeScript: Typed JavaScript at Any Scale*. URL: <https://www.typescriptlang.org/>.

⁷*Angular Framework*. URL: <https://angular.io/>.

⁸The W3 Consortium. *HTML& CSS*. URL: <https://www.w3.org/standards/webdesign/htmlcss.html>.

- **Kotlin**⁹: linguaggio di programmazione *general purpose* e multi-paradigma, sviluppato da *JetBrains*, utilizzato dall'azienda per sviluppare le applicazioni mobili per i dispositivi *Android*. Un esempio di utilizzo di questo linguaggio è l'applicazione mobile per il *contact tracing* del progetto *SyncTrace*;
- **Swift**¹⁰: linguaggio di programmazione orientato agli oggetti, sviluppato da *Apple* e utilizzato dall'azienda per sviluppare le applicazioni mobili per i dispositivi *iOS*.

L'azienda utilizza anche svariati *framework* a supporto della programmazione; alcuni tra i *framework* più utilizzati dall'azienda sono:

- **Spring**: *framework open-source* per lo sviluppo di applicazioni con linguaggio di programmazione *Java*; viene utilizzato, possibilmente combinando il *core* del *framework* con altri progetti quali *Spring Boot* e *Spring Data*, per sviluppare applicativi lato *server*. Esempio di utilizzo di questo *framework* da parte di Sync Lab sono svariate applicazioni web la cui componente *back-end* è sviluppata con l'ausilio di queste tecnologie, come ad esempio *SynClinic* e *StreamCrusher*. Di seguito viene riportata l'architettura generale di tale *framework*;

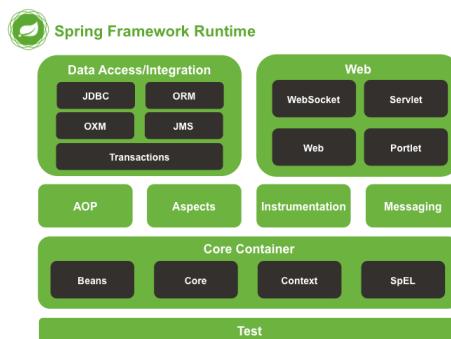


Figura 1.4: Architettura dello *Spring framework*.

Fonte: javaboss.it

- **Angular**: *framework open-source* per lo sviluppo di applicazioni web tramite linguaggio di programmazione *TypeScript*. L'utilizzo principale di questa tecnologia risiede nello sviluppo di *single page application* reattive e costruite su un *back-end* composto da servizi *REST*. Esempi di utilizzo di questo *framework* sono *SynClinic* e *DPS4.0*, già citati in precedenza.
Oltre alla possibilità di sviluppare applicativi veloci e funzionali, questo *framework* offre anche un *design pattern* di tipo *Model-View-ViewModel* nativo come riassunto dalla seguente immagine, fattore che facilita la progettazione e lo sviluppo delle applicazioni;

⁹ Kotlin Programming Language. URL: <https://kotlinlang.org/>.

¹⁰ Apple INC. Swift. URL: <https://developer.apple.com/swift/>.

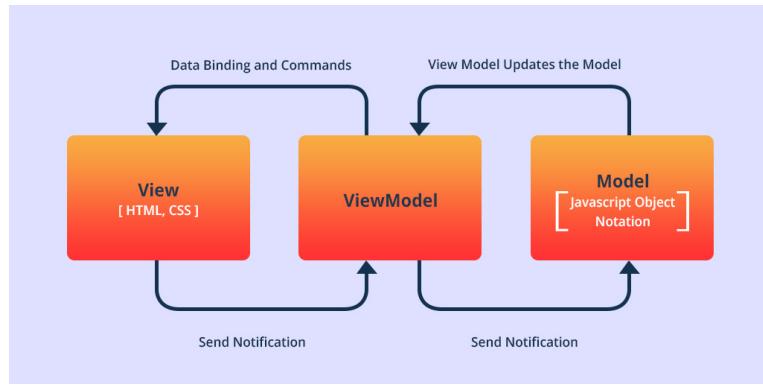


Figura 1.5: Pattern MVVM, adottato anche dal *framework Angular*.

Fonte: alphalogicinc.com

- **Electron¹¹**: *framework open-source* che consente lo sviluppo dell'interfaccia grafica di applicazioni desktop utilizzando tecnologie tipicamente pensate per il web, quali *HTML*, *CSS*, *JavaScript* e *TypeScript*; per fare ciò, questa tecnologia combina il motore di rendering *Chromium* e il *runtime NodeJS¹²*.

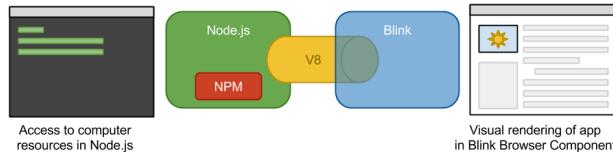


Figura 1.6: Funzionamento del *framework Electron* con *NodeJS*.

Fonte: freecontent.manning.com

Sync Lab utilizza strumenti tecnologici anche per gestire il lavoro da remoto; in particolare, durante l'emergenza sanitaria ancora in corso, l'azienda fa uso di diverse tecnologie per organizzare il lavoro non in presenza, permettendo a tutti i collaboratori di rimanere correttamente aggiornati sulle attività e i compiti di cui sono responsabili. Alcune di queste tecnologie sono le seguenti:

- **Google Meet**: servizio di *Google* per effettuare videoconferenze online. Questa piattaforma viene usata dall'azienda, e in particolare è stata utilizzata anche durante il mio *stage*, per comunicare con gli altri collaboratori e poter rimanere aggiornati sui progressi effettuati;
- **Discord**: applicazione *VoIP* per la comunicazione vocale e testuale. Uno dei punti di forza di questa piattaforma, sfruttato anche dall'azienda, è la possibilità di poter avere più canali, sia testuali che vocali, all'interno dello stesso *server*; questo permette una comunicazione più ordinata e metodica, riducendo il rischio di incomprensioni;

¹¹ *Electron - Build cross-platform desktop apps with JavaScript, HTML, and CSS*. URL: <https://www.electronjs.org/>.

¹² *Runtime NodeJS*. URL: <https://nodejs.org/>.

- **Google Docs:** servizio di *Google* per la condivisione di documenti online. Questa piattaforma è stata utilizzata in particolare durante il mio tirocinio per tenere traccia degli incrementi giornalieri che ho svolto;
- **Trello:** software gestionale in stile *Kanban*, utilizzato per coordinare il proprio *workflow* e visualizzare quello degli altri collaboratori. Questa piattaforma si sposa bene con la metodologia *Agile* che Sync Lab utilizza, in quanto può essere utilizzato come una *Scrum board*.

1.5 Propensione all'innovazione

Sync Lab presta grande attenzione anche all'innovazione e allo sviluppo, sia in senso tecnologico che industriale. L'azienda, infatti, conta tre dipartimenti ideati per sperimentare e innovare, i quali sono **Research and Development**, nato con lo scopo di promuovere nuovi prodotti nati da ricerche in svariati settori, **Lab**, in cui l'azienda sviluppa soluzioni a quanto studiato nel precedente dipartimento, e **Start-up**, il cui scopo è quello di promuovere le *start-up* di maggiore rilevanza per quanto riguarda l'innovazione; per fare ciò, Sync Lab collabora con svariati enti privati e università, sia italiane che estere.



Figura 1.7: Alcune università con le quali l'azienda collabora.

Fonte: synclab.it

Alcuni dei progetti di ricerca fondati e mantenuti da Sync Lab sono i seguenti:

- **BIG-ASC:** acronimo di *BIG Data and Advanced Analytics for Secure Mobile Commerce*, è un progetto che punta a creare una piattaforma *Big Data* che sappia rispondere a requisiti stringenti delle piattaforme di *Mobile Commerce*, come la scalabilità, l'autonomia e le *performance*, attraverso l'analisi continua e in tempo reale dei dati d'utilizzo. Per fare questo, Sync Lab collabora con l'azienda CeRICT e l'*Università degli studi Parthenope* di Napoli;
- **eHealthNet:** ecosistema software per la Sanità Elettronica, che si propone di intervenire su quattro aree tematiche riguardanti la sanità, ossia interoperabilità, pervasività, sostenibilità e preventivabilità. Per lo sviluppo di questo progetto è stato avviato un laboratorio che vede come collaboratori svariate aziende private, l'*Istituto Italiano di Tecnologia*, l'*Istituto Nazionale Tumori*, l'*Università degli studi Federico II* di Napoli e l'*Università degli studi di Salerno*;
- **BDA4PHR:** piattaforma *open-source*, scalabile, estendibile e manutenibile che offre servizi di *storing* e *Big Data Analytics* dedicati ad informazioni di tipo medico-sanitario. Lo scopo di questo progetto è la creazione di un *repository* sicuro, distribuito e affidabile per la gestione, la condivisione e l'analisi di dati eterogenei. Tra i finanziatori di questo progetto ci sono l'*Unione Europea* e il *Ministero dello Sviluppo Economico* italiano.

I progetti a cui l'azienda prende parte sono molteplici e in continuo aumento; alcuni dei progetti ora attivi sono riassunti dal sito aziendale, del quale l'immagine che segue è una cattura di schermo.



Figura 1.8: I progetti di ricerca e sviluppo a cui Sync Lab collabora.

Fonte: synclab.it

Sempre nell'ambito dell'innovazione, posso dire che l'azienda ha un atteggiamento molto propositivo e aperto alle nuove tecnologie: non è raro, infatti, che i collaboratori propongano l'utilizzo di nuovi linguaggi, *framework* e strumenti per completare i servizi e i prodotti offerti dall'azienda.

Ho potuto respirare questo clima di apertura anche nell'ambito del mio tirocinio: avendo avuto accesso alla piattaforma *Discord* aziendale, ho potuto notare che il *server* è diviso in più sottocanali, ognuno dedicato a un ambito di sviluppo, in cui i dipendenti inoltrano articoli e documentazione riguardanti nuove tecnologie, aprendo così a un dibattito costruttivo.

Capitolo 2

Il progetto nel contesto aziendale

2.1 Il rapporto tra stage e azienda

Lo *stage* è un momento fondamentale nella carriera di uno studente universitario; esso infatti rappresenta un passaggio dal mondo accademico al mondo del lavoro, permettendo di passare gradualmente da un'ottica di studio prettamente teorico, per quanto corredato da esercizi pratici, a un'ottica di lavoro con tecnologie, strumenti e metodologie utilizzate quotidianamente dalle aziende.

Sync Lab, avendo sedi in città universitarie ed essendo sempre tesa verso l'innovazione, prende con gran serietà l'opportunità di ospitare tirocini al suo interno. Questo si può vedere dalla grande **esperienza** che questa azienda ha in fatto di *stage*: essi sono infatti organizzati in modo da conformarsi al meglio alle esigenze dello studente, in quanto molto flessibili in termini di durata complessiva. In Sync Lab, inoltre, lo studente non solo è seguito da un tutor esterno, che si occupa di guidare il tirocinante sia organizzativamente che pragmaticamente durante lo sviluppo, ma può trovare aiuto e consiglio anche negli altri collaboratori dell'ufficio.

Per quanto riguarda le motivazioni che ha Sync Lab ad ospitare *stage* universitari, ho individuato le seguenti:

- Prima tra tutte è la tendenza all'**innovazione**: uno studente universitario, infatti, per quanto inesperto della vita lavorativa è mediamente più aggiornato sulle nuove tecnologie, e questo porta l'azienda ad aprirsi a nuove tematiche;
- C'è poi un'ottica di **assunzione futura**: le università sono infatti per Sync Lab il principale bacino di nuovi lavoratori. È molto frequente, infatti, che dopo un tirocinio ben riuscito l'azienda offra un contratto di lavoro allo studente, se questi è disponibile al lavoro;
- Infine, un aspetto da prendere in considerazione è quello **economico**: l'azienda, infatti, non è obbligata a pagare un rimborso spese al tirocinante, e la maggior parte delle spese assicurative sono a carico dell'università; la presenza di uno o più tirocinanti, infine, non rappresenta un costo rilevante in termini di risorse.

2.2 L'azienda in relazione al contesto attuale

Il 31 dicembre 2019, le autorità cinesi riferiscono all'*Organizzazione Mondiale della Sanità* di 41 casi di polmonite anomala che si sono verificati a Wuhan, capitale della provincia di Hubei, in Cina; pochi giorni dopo, gli scienziati cinesi identificano il nuovo virus con il nome di *2019-nCov*. Inizialmente tutto il mondo sottovaluta questa situazione, pensando che fosse un problema riguardante solo la Cina e pochi altri paesi del sud-est asiatico; il 31 gennaio 2020, però, vengono confermati i primi due casi di contagio in Italia.¹ Da quel momento, la sensazione che potesse diventare un problema globale comincia a farsi strada nei pensieri di molte persone.

I giorni a seguire vedono una rapida successione di eventi: a inizio febbraio il virus viene rinominato dall'*OMS* in *SARS-CoV-2*, a fine febbraio scattano le cosiddette **zone rosse** in Italia, zone da cui diventa impossibile uscire e in cui è impossibile entrare, il 7 marzo la Lombardia diventa totalmente zona rossa e, due giorni dopo, scatta il *lockdown* in tutta Italia². Da questo momento, la maggior parte dei luoghi di lavoro viene chiusa, e questo *shutdown* dura fino all'introduzione della *fase 2*, il 4 maggio, giorno in cui alcune attività vengono riavviate, e ancora parzialmente fino al 15 giugno, giorno in cui scatta la *fase 3* che durerà fino al 13 ottobre.

Durante questo periodo di emergenza sanitaria, molte aziende cominciano ad adottare le tecniche del *remote working* e dello *smart working*; tra queste aziende c'è anche Sync Lab, che si attrezza per permettere ai suoi dipendenti e ai tirocinanti di lavorare da casa, interrompendo inizialmente e riducendo di molto successivamente la presenza fisica negli uffici.

Questo contesto di attualità è di grande importanza anche per quanto riguarda il mio *stage* in particolare: temporalmente parlando, infatti, il mio tirocinio è coinciso con la cosiddetta fase 3. Sebbene fosse possibile effettuare incontri fisici, l'azienda ha infatti deciso di ridurre tali incontri per una questione di sicurezza; da questo ne deriva che ho svolto gran parte del mio tirocinio da casa, senza poter accedere direttamente alle strutture e alle infrastrutture se non una volta alla settimana.

Oltre a questo, il periodo è di vitale importanza anche per descrivere lo **scopo** del mio tirocinio: come descrivo più approfonditamente nella sezione seguente, infatti, il progetto di *stage* si configura all'interno del progetto *SyncTrace*, un insieme di applicativi concernenti il tema del *contact tracing*.

2.3 Lo scopo dello stage

In risposta alla situazione sanitaria, molte aziende del settore *ICT* si sono attivate per sviluppare applicazioni e sistemi di *contact tracing*. Il *contact tracing* consiste nell'identificazione delle persone che potrebbero essere venute a contatto con una persona infetta, e nella successiva raccolta di ulteriori informazioni, quali vicinanza e tempo di esposizione, su tali contatti; il fine di questa pratica è aiutare il sistema sanitario a individuare i possibili contagi, arrivando a costruire un albero dei contatti per poter agire tempestivamente sul problema.

¹Il Sole 24 Ore. *Cronistoria del Coronavirus*. URL: <https://lab24.ilsole24ore.com/storia-coronavirus/>.

²Ibid.



Figura 2.1: Rappresentazione schematica del *contact tracing*.

Fonte: mashable.com

Tra le aziende che si sono mobilitate per questa causa c'è anche Sync Lab, che ha avviato un suo progetto in tale ambito chiamato *SyncTrace*. Questo sistema ha come scopo lo sviluppo di due prodotti differenti ma in relazione tra loro: un'applicazione di *contact tracing* puro e un applicativo di gestione e organizzazione per i medici e per gli esercenti di attività commerciali.

Il **primo** applicativo, pensato per l'utilizzo da parte di tutta la popolazione, ha come unico scopo il tracciamento dei contatti delle persone infette; esso è in forma di applicazione per dispositivi mobili, quali *smartphone* e *tablet*, ed è pensato in modo da tracciare i contatti e i livelli di rischio di positività basandosi su parametri oggettivi come la vicinanza e il tempo di esposizione.

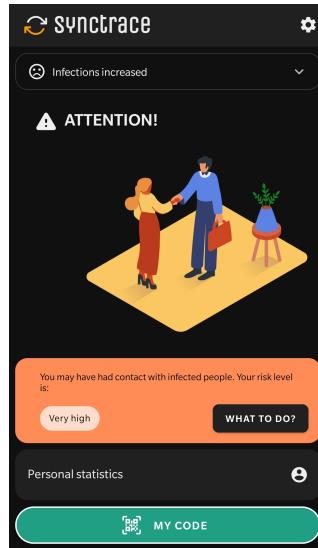


Figura 2.2: Immagine di cattura di schermo iniziale dell'applicazione *SyncTrace*.

Il **secondo** applicativo, consistente in una *web application*, è pensato per fornire un'interfaccia di gestione del sistema ai medici e come applicazione di controllo per gli esercenti. Esso ha quindi un doppio utilizzo:

- I **medici**, una volta identificati come tali, possono usarlo per registrare nuovi contatti infetti, eliminare eventuali pazienti non infetti e verificarne la presenza e/o il livello di rischio di contagio nel sistema condiviso;

- Gli **esercenti**, una volta registrati come tali, possono controllare l'eventuale positività o il livello di rischio di contagio dei clienti che entrano nelle proprie attività.

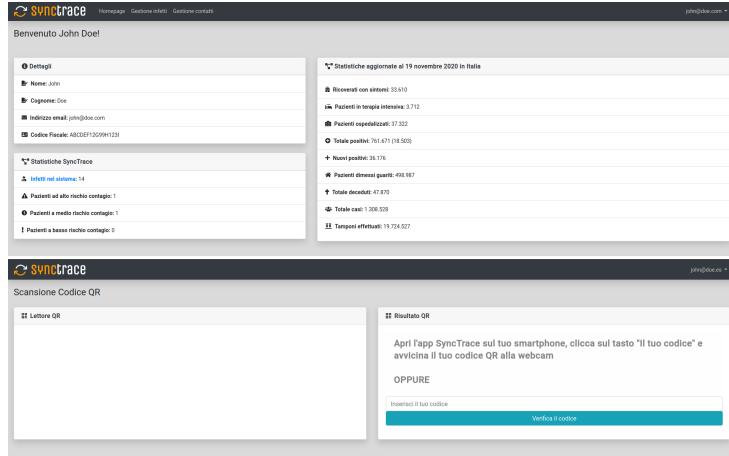


Figura 2.3: Immagine di cattura di schermo iniziale della *webapp* per, rispettivamente, medici ed esercenti.

Il mio progetto di *stage* si inserisce nel contesto di quest'ultimo applicativo descritto. Lo scopo principale del mio tirocinio è, infatti, effettuare il *porting* dell'applicativo utilizzando il *framework Ionic*³, per poter riutilizzare parte della logica già sviluppata secondo il principio del *write once, run anywhere*.

2.4 Vincoli e obiettivi dello stage

Prima di cominciare il tirocinio ho definito, insieme al tutor aziendale, gli obiettivi e i vincoli che il mio lavoro avrebbe dovuto rispettare. Questo ha portato alla stesura di un **piano di lavoro**, riportante lo scopo dello stage, i prodotti attesi, la pianificazione settimanale del lavoro e gli obiettivi del progetto; durante il progetto, però, questo piano di lavoro ha subito due modifiche per meglio adattarsi a esigenze personali e per occupare al meglio il tempo in termini di produttività.

2.4.1 Vincoli temporali

Lo *stage* aveva una durata prevista di 8 settimane, di cui 7 da 40 ore l'una e l'ultima da 20 ore, per un totale di 300 ore. Come già detto, la maggior parte del progetto si è svolto in *smart working*: ho quindi lavorato una media di 8 ore giornaliere da casa. I giorni in cui mi sono recato in ufficio, generalmente il venerdì di ogni settimana, l'orario di lavoro è stato dalle 09.00 alle 18.00, con un'ora di pausa pranzo. Di seguito viene riportata una tabella riassuntiva delle attività svolte, divise per ore e settimane.

³Ionic - Cross-Platform Mobile App Development. URL: <https://ionicframework.com/>.

Settimana	Ore	Attività
1	8	Formazione sul linguaggio <i>Java SE</i>
	16	Formazione sul linguaggio <i>Java EE</i>
	16	Formazione sul <i>framework Spring</i>
2	40	Formazione sul <i>framework Spring</i>
	24	Formazione sul linguaggio <i>JavaScript</i>
3	8	Formazione sul linguaggio <i>TypeScript</i>
	8	Formazione sul <i>runtime NodeJS</i>
4	40	Formazione sul <i>framework Angular</i>
5	20	Formazione sul <i>framework Ionic</i> e su quanto fatto nel progetto precedente
	4	Configurazione dell'ambiente di produzione
	16	Codifica delle maschere previste dal progetto
6	24	Codifica delle maschere previste dal progetto
	16	Configurazione del <i>server</i> di <i>back-end</i> tramite containerizzazione con piattaforma <i>Docker</i>
7	8	Configurazione del <i>server</i> di <i>back-end</i> tramite containerizzazione con piattaforma <i>Docker</i>
8	16	Codifica delle ultime maschere e di componenti aggiuntive
	24	Verifica del codice scritto
	8	Collaudo dell'applicativo prodotto
9	24	Documentazione dell'intero progetto
	8	Codifica e verifica di ultimi <i>fix</i> a <i>bug</i> trovati tramite il collaudo

Tabella 2.1: Suddivisione settimanale delle attività

2.4.2 Vincoli organizzativi

Prima di cominciare lo *stage* ho inoltre pattuito i vincoli organizzativi con il tutor aziendale. Come già detto, per quanto riguarda questo frangente è stato deciso di lavorare in *smart working* per la maggior parte della durata del progetto; abbiamo inoltre deciso di incontrarci periodicamente, solitamente il venerdì, per fare il punto della situazione e potermi allineare con il tutor aziendale.

Ho rendicontato il lavoro che ho effettuato, sia in remoto che in presenza, in un foglio di calcolo online riportante la data, le attività svolte in tale data e l'approvazione o meno da parte del tutor aziendale; ho aggiornato questo documento quotidianamente, e settimanalmente ho ricevuto il *feedback* da parte del tutor. Quella che segue è una porzione di tale documento.

	A	B	C
1	Data	Argomenti	Verificato
2	31/08/2020	Ripasso del linguaggio Java SE: sintassi, costrutti, buone pratiche di programmazione	OK
3	01/09/2020	Studio pratico del linguaggio Java SE [traduzione di un programma esistente da C++ a Java per esercizio], breve overview di Java EE	OK
4	02/09/2020	Studio teorico di Java EE (videolezioni) + primo approccio pratico allo sviluppo di servlet con Java EE e glassfish	OK
5		Conclusione studio generale di Java EE con annessi piccoli esercizi pratici riguardanti JPA. Inizio studio teorico del framework Spring e configurazione del sistema per i futuri esercizi pratici su questo [installazione e configurazione di ide, framework, database myql, server tomcat], e per il futuro sviluppo dell'applicativo richiesto da piano di lavoro	OK
6	03/09/2020	Fine della configurazione del sistema per i futuri esercizi (risolte problematiche con mysql), inizio dello studio dello spring framework (panoramica dell'architettura e del funzionamento generale, creazione del primo progetto di test)	OK
7	04/09/2020	Inizio del secondo progetto di test (da cui si espande poi tutto il resto); introduzione allo spring mvc, creazione di un template con apache tiles e inizio dello studio dello strato di persistenza (connessione dell'applicativo di prova con il database)	OK
8	07/09/2020	Continuazione dello studio dello spring framework, in particolare creazione dello strato di persistenza e dello strato di servizio dell'applicativo, con precedente configurazione della fonte dati (database mysql). Creazione delle prime query, con anche utilizzo delle stored procedures, e inizio delle prime view	OK
9	08/09/2020	Continuazione dello studio dello spring framework, in particolare continuazione della vista con dati prelevati dal database e form di inserimento dati; studio delle diverse notazioni del controller nello spring framework; panoramica sulla sicurezza (prevalentemente prevenzione di sql injection)	ok
10	09/09/2020	Continuazione dello studio dello spring framework, in particolare internazionalizzazione, perfezionamento della vista, JPA2	ok
11	10/09/2020	Studio di spring Boot e dei web services; studio del formato json e suo utilizzo nell'ambito di architetture rest con Spring Boot; full stack development con il framework spring (e spring boot) e angularjs. Conclusione dello studio di Spring	ok
	11/09/2020		

Figura 2.4: Porzione del registro delle attività.

Per quanto riguarda il salvataggio e il versionamento del codice e della documentazione prodotti, inoltre, ho fatto uso della *repository GitLab* aziendale.

2.4.3 Vincoli tecnologici

Per quanto riguarda i vincoli tecnologici, posso dire di essere stato abbastanza libero nella scelta di quali strumenti utilizzare; i vincoli a cui sono sottostato sono quindi stati decisi principalmente da me, in accordo con il tutor aziendale. Questi sono qui riportati.

Sistema operativo

Essendo il prodotto a cui ho lavorato non coperto da segreto industriale, ho svolto l'intero progetto sulla mia macchina senza bisogno di software esterni per rispettare le norme di sicurezza aziendali. Il sistema operativo che ho utilizzato per lo sviluppo è stato *GNU/Linux*, in particolare la distribuzione *Arch Linux* a 64 bit.

PostgreSQL

Sebbene il mio lavoro non comprendesse l'utilizzo diretto di un *database*, la componente *back-end* dell'applicativo fa uso del *database* relazionale *PostgreSQL*⁴; ho avuto modo di utilizzare questa tecnologia durante il processo di collaudo, per verificare che i dati inseriti dall'applicazione venissero correttamente immagazzinati dal *database*.

Docker

Per effettuare il *deploy* della componente *back-end* dell'applicativo, necessaria al processo di collaudo su dispositivo mobile, ho fatto uso dello strumento di containerizzazione

⁴ *PostgreSQL: The world's most advanced open source database.* URL: <https://www.postgresql.org/>.

*Docker*⁵. Questo mi ha permesso di rendere la suddetta componente installabile facilmente su qualsiasi sistema.

Android Studio

Per effettuare il collaudo di quanto sviluppato su dispositivo mobile, sono stato vincolato all'ambiente di sviluppo integrato *Android Studio*, fornito gratuitamente dall'azienda *JetBrains*, per la compilazione dell'applicazione per dispositivi *Android*; in accoppiata a questo ho utilizzato il *tool ADB* (*Android Debug Bridge*) per l'installazione dell'applicazione sul dispositivo.

Documentazione

Per quanto riguarda la documentazione, dopo i consigli del mio tutor aziendale la mia scelta è ricaduta sul *tool Compodoc*⁶, uno strumento simile a *Javadoc* per applicazioni sviluppate tramite il *framework Angular*; questo strumento ha permesso la generazione automatica di documentazione a partire da commenti strutturati al codice.

2.4.4 Obiettivi

Insieme al tutor aziendale ho delineato gli obiettivi dello *stage* e li ho riportati in un piano di lavoro validato dal tutor interno, il prof. Tullio Vardanega.

Ogni obiettivo possiede un nome formato nel seguente modo:

0- [tipologia] [numero]

dove:

- [tipologia] indica la tipologia di obiettivo. Questa può essere:
 - 0: obbligatorio, ossia un obiettivo vincolante e necessario da soddisfare;
 - D: desiderabile, ossia un obiettivo non vincolante ma dal riconoscibile valore aggiunto;
 - F: facoltativo, ossia un obiettivo non vincolante e rappresentante un valore aggiunto non competitivo.
- [numero] consiste in un numero, progressivo per ogni tipologia.

Gli obiettivi inizialmente individuati sono riportati nella seguente tabella.

⁵ Docker: Empowering App Development for Developers. URL: <https://www.docker.com/>.

⁶ Compodoc - The missing documentation tool for your Angular application. URL: <https://compodoc.app/>.

Obiettivo	Descrizione
0-01	Acquisizione delle competenze su linguaggi di programmazione, <i>framework</i> e strumenti di supporto
0-02	Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma
0-03	Portare a termine le implementazioni previste con una percentuale di superamento pari all'80%
0-D1	Portare a termine le implementazioni previste con una percentuale di superamento pari al 100%
0-F1	Dare un contributo importante nelle fasi di progettazione delle maschere con l'obiettivo di realizzare una app usabile, <i>responsive</i> e <i>adaptive</i>
0-F2	Implementare la lettura del <i>QR Code</i> dirattamente dalla fotocamera dello <i>smartphone</i> , evitando quindi inserimenti manuali

Tabella 2.2: Obiettivi di progetto originali.

In corso di progetto il piano di lavoro è stato rimodulato, in quanto ho svolto parte del lavoro in un tempo minore del previsto; ai precedenti obiettivi si è quindi aggiunto il seguente:

Obiettivo	Descrizione
0-D2	Containerizzazione della componente <i>back-end</i> del progetto <i>SyncTrace</i>

Tabella 2.3: Ulteriorie obiettivo, aggiunto in corso di progetto.

Il piano di lavoro identifica inoltre i prodotti attesi; questi sono:

- Un **documento tecnico** che descriva le maschere utilizzate;
- Il **codice** rilasciato sul *repository* indicato dall'azienda.

2.5 Motivazione della scelta

Sono venuto a conoscenza di Sync Lab durante l'evento **StageIT 2020**. Questo evento, organizzato dal professor Tullio Vardanega dell'*Università degli Studi di Padova* in collaborazione con l'associazione di imprenditori *Assindustria Venetocentro*, è un'occasione in cui gli studenti di diverse facoltà dell'università di Padova vengono messi in contatto con le aziende presenti sul territorio.

L'evento viene organizzato annualmente; quest'anno, però, a causa dell'emergenza sanitaria non si è potuto svolgere in presenza. Nonostante questo, l'adesione da parte delle aziende e degli studenti è stato comunque soddisfacente, anche grazie al fatto che, seguendo gli incontri con le aziende per via telematica, è stato possibile contattarne più di una contemporaneamente.

Durante l'evento *StageIT* ho avuto occasione di parlare con i rappresentanti di quattro aziende operanti nel comune di Padova, tra cui l'ing. Fabio Pallaro di Sync Lab.

Gli ambiti di *stage* a cui ero interessato erano principalmente tre: sviluppo software *mobile*, *data science* e *cybersecurity*. Uno dei motivi per cui ho optato per Sync Lab

come azienda è stato che l'ing. Pallaro mi ha offerto due diversi ambiti di *stage*: uno riguardante lo sviluppo di un'applicazione per dispositivi mobili tramite *framework Ionic*, e uno riguardante l'aspetto di sicurezza dei prodotti che l'azienda stava e avrebbe sviluppato con altri tirocinanti, entrambi ambiti a cui ero particolarmente interessato. L'azienda, inoltre, mi è sembrata molto attenta agli studenti interessati a un tirocinio presso di essa, segno a mia opinione di serietà, professionalità e apertura a nuove esperienze.

Per quanto riguarda il progetto, ho infine optato per lo sviluppo di un applicativo mobile per svariati motivi, anzitutto l'ambito: la tematica del *contact tracing* mi è infatti fin da subito interessata per l'utilità che può avere in situazioni come quella attuale. Oltre a questo, ho trovato che le tecnologie che sarei poi andato ad utilizzare fossero interessanti da un punto di vista personale e, soprattutto, professionale: i linguaggi *JavaScript* e *TypeScript* e i *framework* web e mobile come *Angular* e *Ionic*, infatti, sono utilizzati sempre di più frequentemente dalle aziende operanti nel settore *IT*.

2.6 Formazione

Ho dedicato il primo mese del mio tirocinio allo studio e all'approfondimento di tutte le conoscenze che mi sarebbero servite, o mi sarebbero state utili, a sviluppare il progetto di *stage* vero e proprio. Gli **strumenti** che ho utilizzato a tale scopo sono molteplici: anzitutto, ho fatto uso della piattaforma *Udemy* per acquisire le informazioni teoriche riguardanti le tecnologie oggetto di studio; a questo ho associato anche l'utilizzo della piattaforma *YouTube*, sulla quale ho trovato svariati video di particolare utilità. Per quanto riguarda gli esercizi pratici, invece, ho fatto uso sia della piattaforma *Udemy* che di *tool* per l'esercitazione, quali ad esempio il pacchetto *npm "Learn YouNode"*⁷ per quanto riguarda lo studio del runtime *NodeJS*.

2.6.1 Tecnologie

Ho cominciato lo studio delle tecnologie con il linguaggio di programmazione **Java**, del quale ho appreso principalmente la sintassi, i costrutti di base e, soprattutto, l'implementazione di *servlet* e le *JPA* (*Java Persistence API*) del linguaggio *Java Enterprise Edition*. Sebbene il mio progetto non prevedesse lo sviluppo in codice *Java*, la componente *back-end* del prodotto cui il mio progetto appartiene è scritto con questo linguaggio in accoppiata al *framework Spring*; ho quindi ritenuto opportuno, insieme al tutor aziendale, approfondire questo argomento per poter comprendere almeno a grandi linee il funzionamento del *back-end*. Successivamente ho quindi studiato il *framework Spring*: di questo ho studiato a grandi linee il funzionamento generale, comprendente i diversi strati da cui è composto un applicativo sviluppato con questa tecnologia (*database*, persistenza, *business* e presentazione).

Una volta terminato lo studio delle tecnologie riguardanti la componente *back-end* del prodotto, sono passato allo studio di quelle riguardanti la componente *front-end*. Ho quindi cominciato con lo studio di **JavaScript**, linguaggio del quale avevo già delle conoscenze abbastanza radicate; di questo ho quindi approfondito aspetti avanzati, quali *promise*, *fetch api* e le funzioni asincrone. Sebbene anche questo linguaggio non fosse previsto dal mio progetto, è stato comunque utile da studiare come base per lo

⁷ Il pacchetto *Learn YouNode*. URL: <https://www.npmjs.com/package/learnyounode>.

studio di **TypeScript**, che è stato quello principalmente utilizzato per lo sviluppo dell'applicativo del mio progetto.

Una volta concluso lo studio di questi linguaggi di programmazione ho quindi approfondito i *framework* che li utilizzano nel contesto del mio progetto di *stage*. Per primo ho studiato **Angular**, che è la tecnologia utilizzata per lo sviluppo dell'applicativo di cui il mio progetto prevedeva il *porting*; è stato quindi necessario studiarlo per capire al meglio come fosse strutturato tale applicativo. Ho infine approfondito il *framework* **Ionic** che è stato quello che più ho utilizzato per lo sviluppo.

Per completare le mie conoscenze in ambito *back-end JavaScript*, inoltre, ho studiato il *runtime* **NodeJS**. Nessun componente del prodotto a cui il mio progetto appartiene è sviluppato interamente con *NodeJS*; alcune componenti di esso vengono utilizzate, però, per la gestione del collegamento tra la componente *front-end* e quella *back-end* dell'applicativo di cui ho eseguito il *porting*.

Ho infine approfondito **Docker**, piattaforma che automatizza il *deployment* di applicazioni all'interno di macchine virtuali *linux* minimali. Ho approfondito questo argomento a sviluppo già cominciato, poiché per il processo di collaudo si è reso necessario per me avere la componente *back-end* del prodotto su un *server* accessibile via *http*, e mi è stato proposto di containerizzare tale componente per poter effettuare un *deployment* più efficiente. Di seguito ho riportato uno schema che mostra come le diverse tecnologie interagiscono tra loro nel caso del progetto oggetto del mio *stage*.

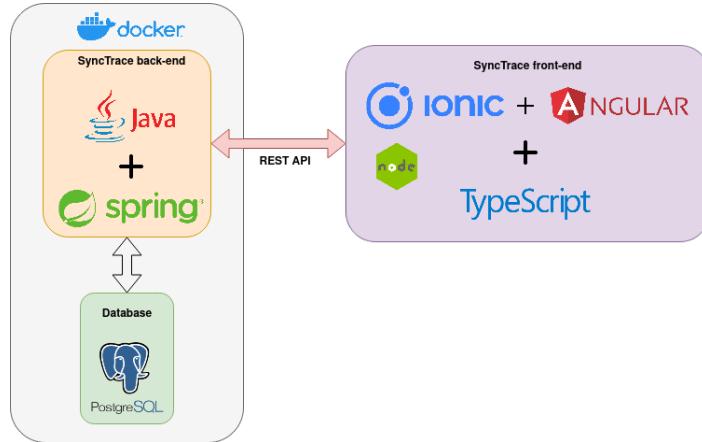


Figura 2.5: Schema rappresentante le relazioni che sussistono tra le tecnologie descritte in SyncTrace.

2.6.2 Progetto

Come precedentemente riportato, il mio progetto è consistito nell'effettuare il *porting* di una *web application* su dispositivi mobili tramite il *framework* *Ionic*. L'applicazione web, consistente in una *single page application* sviluppata in *TypeScript* tramite il *framework* *Angular*, offre le stesse funzionalità dell'applicativo sviluppato da me; molte di queste, però, non sono traslabilmente direttamente su *mobile*, poiché consistenti di elementi non nativi dei sistemi operativi mobili. Esempio di ciò sono sia piccolezze estetiche, quali ad esempio i bottoni, i *form* e gli *alert*, che l'utilizzo delle risorse hardware del sistema, quali ad esempio la fotocamera, i bottoni fisici e gli strumenti di misura come

l'accelerometro. Ho quindi effettuato anche uno studio su quanti e quali componenti software avrei dovuto cambiare per sviluppare correttamente il mio prodotto.

Capitolo 3

Il progetto di stage

3.1 Analisi dei requisiti

Ho effettuato l'analisi dei requisiti aiutandomi con quanto fatto durante il progetto precedente, di cui il mio prodotto è un *porting*: le funzionalità offerte dal sistema che ho sviluppato, infatti, devono essere e sono le medesime di tale progetto. Prima di cominciare le attività di progettazione e codifica ho quindi delineato i requisiti richiesti dal progetto al fine di avere uno strumento concreto per verificare l'avanzamento del prodotto e poter validare infine quanto fatto.

In questa sezione riporto i casi d'uso dell'applicativo che ho sviluppato; questi non sono parte della documentazione che ho prodotto durante il progetto, ma solo uno strumento a me congeniale per descrivere sinteticamente le funzionalità offerte dal prodotto che ho sviluppato.

3.1.1 Casi d'uso

I casi d'uso consistono in diagrammi che descrivono le possibili interazioni tra l'utente e il sistema. Ogni caso d'uso riportato in questa sezione possiede un codice identificativo, formato nel seguente modo:

UC[codicepadre].[codicefiglio]

dove:

- UC significa Use Case;
- [codicepadre] è un numero progressivo che inizia da 1;
- [codicefiglio] è un numero progressivo che inizia da 1 ad ogni incremento di [codicepadre].

Ho qui riportato solamente i casi d'uso principali riguardanti gli attori primari, al fine di esporre chiaramente solo le funzionalità richieste dal sistema oggetto del mio *stage*.

Attori primari

Gli attori primari corrispondono agli utenti che interagiscono con il sistema.

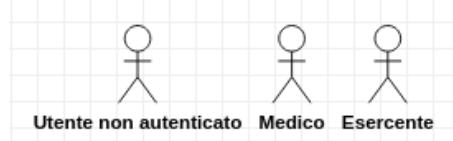


Figura 3.1: Attori principali.

Nel sistema del prodotto che ho sviluppato, gli attori principali sono i seguenti:

- **Utente non autenticato:** utente, medico o esercente, che non si è ancora autenticato sulla piattaforma;
- **Medico:** utente che si è autenticato come medico sulla piattaforma;
- **Esercente:** utente che si è autenticato come esercente sulla piattaforma.

Diagrammi dei casi d'uso

Utente non autenticato

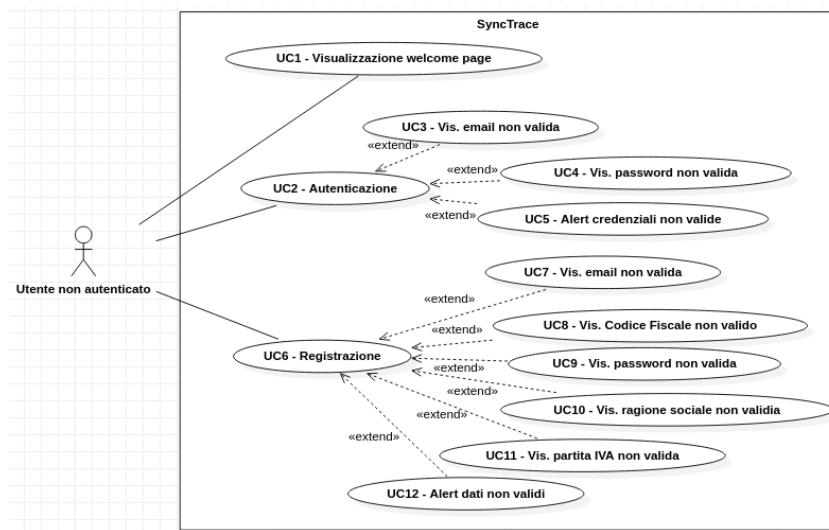


Figura 3.2: Diagramma dei casi d'uso per un utente non autenticato.

- **Attore primario:** l'utente non autenticato;
- **Descrizione:** l'utente non autenticato non può accedere al resto dell'applicazione. Per poterlo fare deve autenticarsi o registrarsi, e questo deve venire offerto dall'applicazione;
- **Precondizione:** l'applicazione offre la possibilità di autenticarsi o registrarsi;
- **Scenario principale:** l'utente non autenticato visualizza la pagina di benvenuto. Dopo aver cliccato sul pulsante *Inizia* può autenticarsi (UC2) o registrarsi (UC6);
- **Postcondizione:** l'utente si è autenticato;
- **Estensioni:**

- Se l’utente in autenticazione inserisce una mail o una password mal formattata viene visualizzato un errore sul campo errato (UC3 e UC4);
- Se l’utente tenta di autenticarsi con credenziali non valide viene visualizzato un *alert* (UC5);
- Se l’utente in registrazione inserisce mail, codice fiscale, password, ragione sociale o partita IVA mal formattati viene visualizzato un errore sul campo errato (UC7, UC8, UC9, UC10 e UC11);
- Se l’utente in registrazione inserisce dati non validi viene visualizzato un *alert* (UC12).

Medico

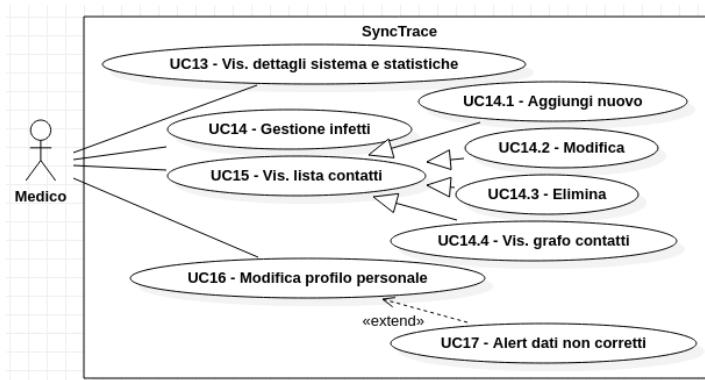


Figura 3.3: Diagramma dei casi d’uso per un utente autenticato come medico.

- **Attore primario:** l’utente autenticato come medico;
- **Descrizione:** l’utente medico può accedere alle funzionalità offerte dall’applicazione per il suo ruolo;
- **Precondizione:** l’applicazione permette di accedere alle funzionalità offerte ai medici;
- **Scenario principale:** il medico può visualizzare i dettagli del sistema e le statistiche riguardanti COVID-19 in Italia (UC13), gestire gli infetti (UC14 e sottocasi), visualizzare la lista dei contatti presenti nel sistema (UC15) e modificare il profilo personale (UC16);
- **Postcondizione:** l’utente ha utilizzato le funzionalità a lui riservate;
- **Estensioni:** se l’utente modifica i suoi dati inserendo dati errati viene visualizzato un *alert* (UC17).

Esercente

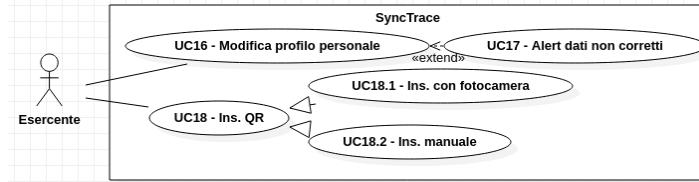


Figura 3.4: Diagramma dei casi d'uso per un utente autenticato come esercente.

- **Attore primario:** l'utente autenticato come esercente;
- **Descrizione:** l'utente esercente può accedere alle funzionalità offerte dall'applicazione per il suo ruolo;
- **Precondizione:** l'applicazione permette di accedere alle funzionalità offerte agli esercenti;
- **Scenario principale:** l'esercente può modificare il profilo personale (UC16) e inserire il codice QR (UC18) tramite fotocamera o manualmente (sottocasi di UC18);
- **Postcondizione:** l'utente ha utilizzato le funzionalità a lui riservate;
- **Estensioni:** se l'utente modifica i suoi dati inserendo dati errati viene visualizzato un *alert* (UC17).

3.1.2 Requisiti

Una volta definiti i casi d'uso ho delineato i requisiti del progetto che sarei andato a sviluppare; per fare ciò sono partito dai requisiti precedentemente definiti e riportati in documentazione, e li ho espansi con i requisiti dell'applicativo oggetto del mio *stage*. Ogni requisito possiede un codice identificativo così definito:

R[**tipo**] [**importanza**] - [**numero**]

dove:

- R indica che è un requisito;
- [**tipo**] identifica il tipo di requisito. Questo può essere:
 - F: requisito funzionale;
 - Q: requisito qualitativo;
 - V: requisito di vincolo;
- [**importanza**] identifica l'importanza del requisito. Questa può essere:
 - O: requisito obbligatorio;
 - D: requisito desiderabile;
 - Z: requisito opzionale;
- [**numero**] identifica il numero del requisito. Esso è composto in tal modo:

a . b . c

dove:

- a: numero progressivo che parte da 1 e viene incrementato a ogni nuovo requisito dello stesso tipo e importanza;
- b: numero progressivo che parte da 1 e viene incrementato a ogni incremento di a;
- c: numero progressivo che parte da 1 e viene incrementato a ogni incremento di b.

Nell'ambito del precedente *stage* sono stati definiti solo i requisiti funzionali obbligatori riguardanti l'utente non autenticato e l'utente autenticato come medico; a questi ho quindi aggiunto i requisiti funzionali obbligatori riguardanti l'utente autenticato come esercente.

Ho inoltre aggiunto altri sei requisiti **funzionali**, di cui quattro desiderabili e due opzionali, riguardanti l'usabilità dell'applicativo su dispositivi mobili. Per quanto riguarda i requisiti di **vincolo** ho invece individuato tre requisiti, rifacendomi ai vincoli definiti dal mio *stage*, due dei quali sono individuabili fin da inizio *stage* e riguardano l'utilizzo di determinate tecnologie, e il terzo riguarda l'utilizzo di *Docker* per la containerizzazione del *back-end*, ed è stato quindi aggiunto in corso di sviluppo. Tra i requisiti **qualitativi**, infine, ho inserito l'utilizzo di analisi statica per la verifica del codice e l'utilizzo di test di unità per la verifica delle funzioni e dei metodi realizzati. Il totale dei requisiti che ho identificato è quindi riassunto dalla seguente tabella:

	Obbligatori	Desiderabili	Opzionali	Totale
Funzionali	20	4	2	26
Qualitativi	2	0	0	2
Di vincolo	3	0	0	3
Totali	25	4	2	31

Tabella 3.1: Totale dei requisiti identificati.

Considerando che molti dei requisiti funzionali individuati implicano anche numerosi sottorequisiti, il numero totale è certamente importante; nonostante questo, ritengo sia equilibrato rispetto al tempo e alle risorse che ho avuto a disposizione. La ragione di ciò è da ricercarsi nel fatto che 18 di questi requisiti provengono dal precedente progetto: nonostante io abbia dovuto comunque soddisfarli sono infatti stato agevolato nel farlo, non dovendo realizzare da zero le funzionalità ma potendo semplicemente trasporre il codice da una tecnologia a un'altra.

3.2 Progettazione

Anche per quanto riguarda le scelte progettuali, essendo il mio progetto la continuazione di un progetto già concluso non ho avuto molto spazio di manovra. Ho però potuto riprogettare l'intera interfaccia grafica dell'applicativo per meglio conformarsi all'utilizzo *mobile* di questo.

Prima di procedere alla progettazione vera e propria e alla codifica, ho sviluppato un

mockup dell'applicazione; poiché le funzionalità offerte dal sistema erano già state definite durante il precedente *stage* su tale progetto, la mia attenzione in questo momento si è rivolta unicamente all'aspetto grafico dell'applicazione che avrei dovuto sviluppare. Per fare ciò ho quindi usato un programma di *sketching*, chiamato *WireframeSketcher*¹, per abbozzare l'idea di come avrebbe dovuto essere l'applicazione. Questo programma permette di disegnare *mockup* di siti web e applicazioni *mobile* e *desktop*, mettendo a disposizione numerosi *asset*, quali ad esempio bottoni, campi di testo e *label*, che possono essere facilmente utilizzati per disegnare bozze di applicativi utili ad avere una base di partenza per il successivo sviluppo.

Con *WireframeSketcher* ho disegnato le bozze di quasi tutte le maschere che poi ho implementato: ho infatti realizzato 12 maschere, e ho sviluppato la bozza di 10 di queste; questa disuguaglianza trova ragione in una iniziale malinterpretazione dell'applicativo da cui sono partito per lo sviluppo, riguardante le maschere di modifica e visualizzazione degli utenti infatti nel sistema.

Non ho riportato questi *mockup* nella documentazione ufficiale fornita come parte del prodotto all'azienda, ma mi sono stati utili per procedere senza indugi nella successiva attività di codifica. Seguono alcuni *sketch* d'esempio.

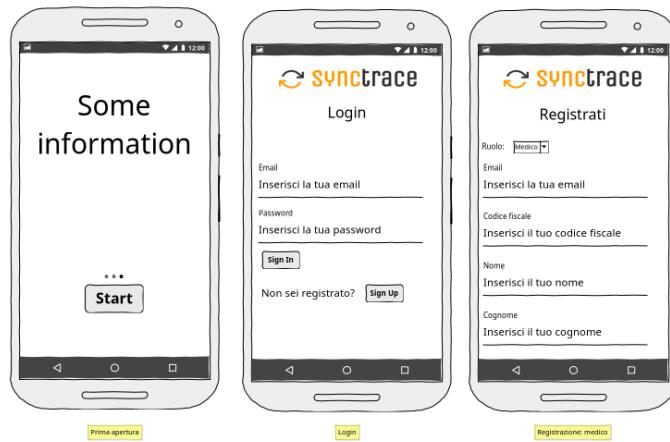


Figura 3.5: Esempio di *sketch* dell'applicazione realizzato con *WireframeSketcher*.

Come tratterò più approfonditamente nella prossima sezione, sebbene il mio progetto consistesse nella migrazione di una *webapp Angular* su dispositivi mobili è vero anche che il *framework* principale che ho utilizzato è diverso: la *webapp*, infatti, è stata sviluppata totalmente tramite il *framework Angular*, mentre l'applicazione oggetto del mio *stage* è sviluppata tramite il *framework Ionic* che utilizza alcune funzionalità del *framework Angular*, ma è in sostanza una tecnologia diversa.

La differenza più evidente tra *Angular* e *Ionic* è che il primo è pensato per lo sviluppo di *single page application*, mentre il secondo è stato creato appositamente per lo sviluppo di applicazioni multiplattaforma, principalmente *mobile*. *Ionic* fa uso del *framework Angular*, del *framework React*² o del *framework Vue*³ per quanto riguarda l'impostazione del progetto, la suddivisione in componenti e la gestione delle *route* tra questi; a questo però aggiunge un sovrastato per lo sviluppo di una componente *front-end* ottimizzata per dispositivi mobili, *reactive* e *responsive*, composto da numerosi

¹ *WireframeSketcher*: Wireframing tool for professionals. URL: <https://wireframesketcher.com/>.

² *React* - Una libreria JavaScript per creare interfacce utente. URL: <https://it.reactjs.org/>.

³ *Vue Framework*. URL: <https://vuejs.org/>.

componenti grafici di cui l'immagine successiva è un esempio.

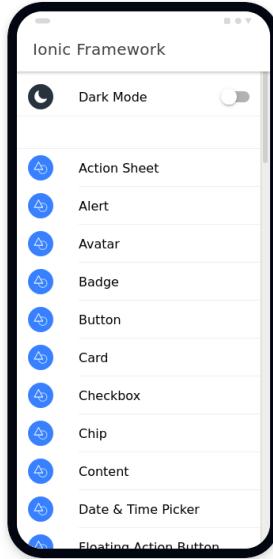


Figura 3.6: Esempio di funzionalità proprie di *Ionic*.

Fonte: ionicframework.com

Essendo quindi i *framework* utilizzati per lo sviluppo della *webapp* e dell'applicazione *mobile* differenti, ho dovuto compiere delle scelte progettuali riguardanti la **sostituzione** di determinati componenti con altri pensati appositamente per l'utilizzo su dispositivi mobili e l'**aggiunta** del supporto a funzionalità non presenti su dispositivi *desktop*. L'esempio più lampante del primo caso è l'utilizzo della fotocamera: la *webapp*, infatti, fa uso della libreria ZXingScannerModule⁴, libreria che non funziona su dispositivi mobili poiché non in grado di richiedere l'autorizzazione all'utilizzo della fotocamera; al suo posto ho utilizzato la libreria qr-scanner di *Ionic Native*⁵, che permette di accedere alle componenti hardware richieste.

Un esempio di aggiunta di supporto a funzionalità non presenti su dispositivi *desktop* è l'utilizzo di tasti fisici: i dispositivi di input di un computer *desktop* sono infatti principalmente mouse e tastiera, mentre in un dispositivo mobile sono principalmente *touchscreen* e tasti fisici (soprattutto *soft-touch*). Ho quindi dovuto utilizzare il componente backButton della libreria Platform di *Ionic* per garantire il corretto utilizzo del tasto fisico di *back* dei dispositivi mobili.

Combinando le attività di analisi dei requisiti e di progettazione ho individuato un totale di quattro importanti componenti da sostituire e sei funzionalità da aggiungere al fine di ottenere un'applicazione *mobile* funzionante, usabile e rasembrante un'applicazione nativa per dispositivi *Android* o *iOS*. A questi si sono poi aggiunti numerosi componenti minoritari di *Ionic* che sono andati a sostituire i rispettivi componenti di *Angular*; esempio di ciò sono i bottoni, i campi dati e le etichette, che su dispositivi *mobile* hanno dicitura e aspetto differenti rispetto a tali componenti implementati su una *webapp Angular*.

⁴Libreria Angular ZXingScannerModule. URL: <https://www.npmjs.com/package/@zxing/ngx-scanner>.

⁵Libreria Ionic Native. URL: <https://ionicframework.com/docs/native>.

3.3 Codifica

Una volta conclusa la progettazione sono passato subito all'attività di codifica. Come precedentemente ribadito, il *framework* utilizzato durante il progetto di *stage* precedente al mio e quello che ho invece utilizzato io sono diversi; un'ulteriore differenza, rispetto a quanto riportato, è che anche gli strumenti a supporto dello sviluppo sono differenti: per la generazione automatica del codice ripetitivo da cui partire con lo sviluppo del codice originale (codice *boilerplate*) con *framework Angular*, infatti, viene utilizzato il *tool AngularCLI*⁶, mentre con *Ionic* viene utilizzato il *tool IonicCLI*⁷. Questo strumento permette la generazione automatica dei componenti, divisi in opportune cartelle come mostrato nell'immagine che segue, da cui partire per sviluppare l'applicativo di interesse. In particolare, durante il progetto ho fatto uso di tale strumento per generare i seguenti componenti:

- **Component:** in *Angular*, un *component* è un componente consistente in vista e modello uniti; esso modella quindi sia l'interfaccia grafica che la manipolazione dei dati. In *Ionic*, un *component* è invece concepito come una **porzione** di applicazione che può essere usata o meno più volte all'interno delle *page*. Un esempio di questo componente in *Ionic* è, nel mio caso, il logo dell'applicazione, che essendo definito come *component* può essere inserito in diverse *page* usando solo la sua dicitura *HTML*;
- **Service:** i *service* di *Ionic* coincidono con quelli di *Angular*; essi corrispondono alla *business logic* del progetto, e racchiudono quindi le funzionalità che possono essere utilizzate dagli altri componenti. Nel caso del prodotto sviluppato da me e dagli studenti il cui *stage* ha preceduto il mio, un *service* è ad esempio la componente che si occupa di gestire l'autenticazione degli utenti all'interno del sistema, comunicando con i servizi offerti dal *back-end*;
- **Page:** dal punto di vista di *Angular*, una *page* consiste solamente e totalmente in un *component*; in *Ionic*, invece, consiste in un *component* che si comporta come un'intera *view*. Al suo interno, quindi, si possono trovare i *component* precedentemente definiti. Nel mio progetto, ad esempio, una *page* è la maschera di *login*, o una singola maschera visualizzata da un medico o un esercente;
- **Module:** un *module* consiste in un modulo *Angular* che effettua il *bootstrap* in un'applicazione *Ionic*. Lo scopo di tale componente è assicurare che tutti i componenti, le direttive e i *provider* del *framework Ionic* siano importati. Esempi di moduli nel mio progetto sono i *routing module*, che si occupano di gestire le *route* di *Angular* all'interno dell'applicazione.

⁶ Il *tool AngularCLI*. URL: <https://cli.angular.io/>.

⁷ Il *tool IonicCLI*. URL: <https://ionicframework.com/docs/cli>.

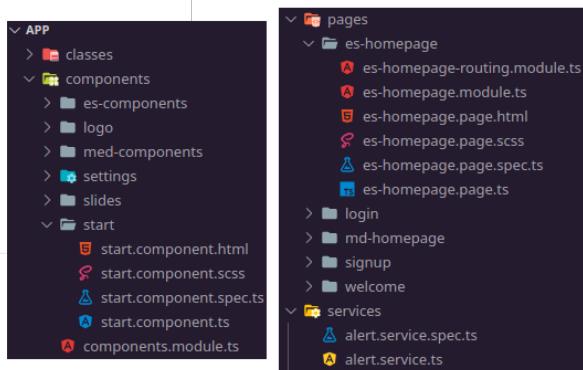


Figura 3.7: Porzioni di *directory* contenente un progetto *Ionic*.

Per quanto riguarda lo sviluppo del codice dell'applicazione, ho anzitutto cominciato con le maschere di *login* e *signup*, per permettermi fin da subito di autenticarmi correttamente all'interno del sistema e poter quindi sviluppare le funzionalità previste da entrambi i ruoli. Per la codifica di queste ho fatto un riutilizzo quasi completo dei servizi precedentemente sviluppati, andando a modificare solamente le *route* per permettermi di sviluppare anche la pagina di benvenuto, che viene presentata al primo avvio assoluto dell'applicazione e fintanto che l'utente non si autentica per la prima volta.

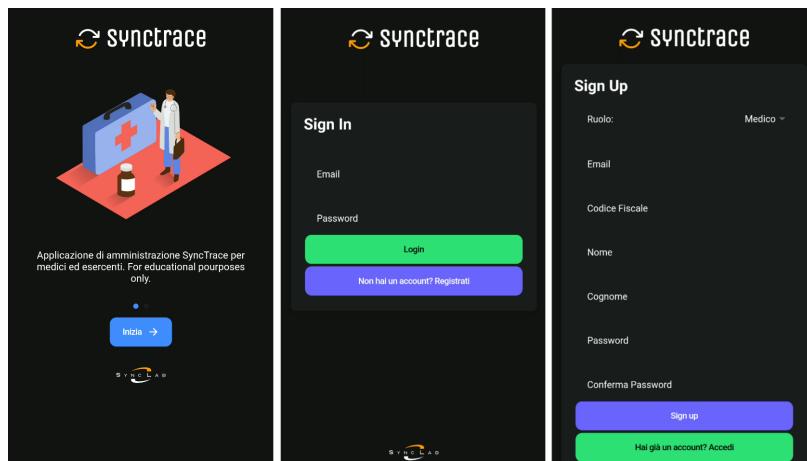


Figura 3.8: In ordine: pagina di benvenuto, *login* e registrazione.

Dopo aver realizzato queste prime tre maschere sono passato alla codifica delle maschere riguardanti l'utente con ruolo esercente. Come anticipato precedentemente, l'esercente utilizza l'applicazione per controllare se un cliente può o meno entrare nella sua attività commerciale; per fare ciò, l'esercente legge il codice identificativo del cliente tramite l'applicazione, che comunicherà se questo è o meno infetto o con quale rischio lo è. Per leggere il codice identificativo l'utente ha due possibilità: utilizzare la fotocamera del dispositivo per leggere un codice QR o inserire il codice manualmente. Ho quindi realizzato due maschere, una per modalità di inserimento del codice.

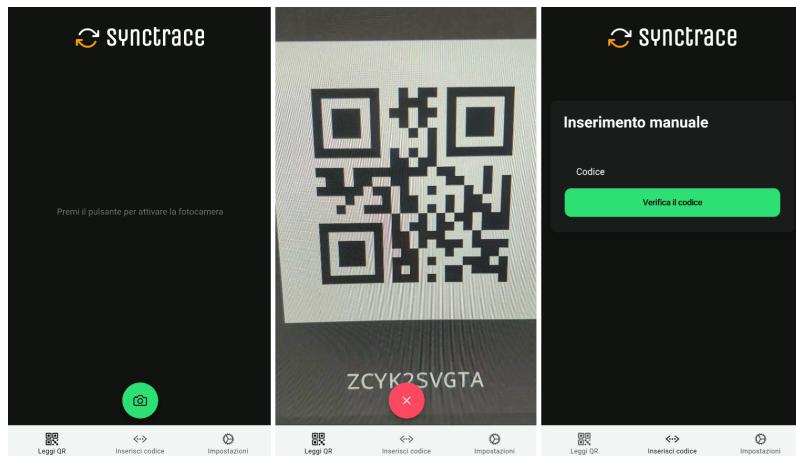


Figura 3.9: In ordine: lettura del QR a fotocamera spenta, lettura del QR a fotocamera accesa e inserimento manuale.

A cavallo tra lo sviluppo delle maschere riguardanti l'esercente e quelle riguardanti il medico, ho codificato la maschera delle impostazioni; da questa visuale è possibile visualizzare e modificare le informazioni personali e la password.

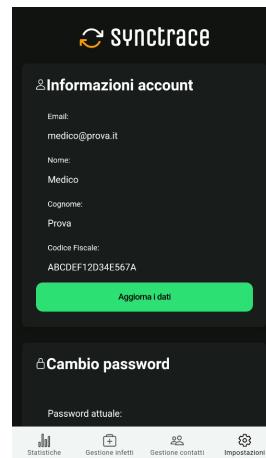


Figura 3.10: Maschera delle impostazioni dell'utente.

Una volta realizzata questa maschera ho proceduto alla realizzazione delle maschere riguardanti il medico. Come descritto brevemente in precedenza, l'utente medico utilizza l'applicazione per visualizzare le informazioni riguardanti gli infetti e i contatti presenti il tutto il sistema, per gestire gli utenti infetti e visualizzare i contatti tra utenti. Ho quindi realizzato sei maschere per visualizzare i dati, visualizzare, modificare e aggiungere gli utenti infetti e visualizzare i contatti tra utenti.

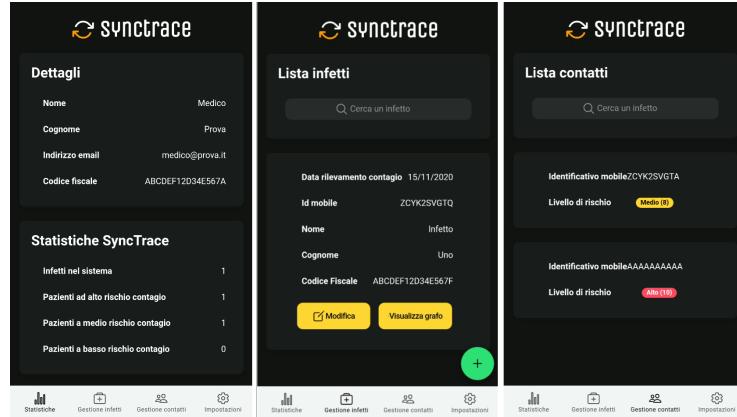


Figura 3.11: In ordine: statistiche del sistema, gestione degli infetti e gestione dei contatti.

Durante lo sviluppo delle maschere ho apportato diverse modifiche ai *service* precedentemente sviluppati, al fine di renderli maggiormente compatibili con i dispositivi mobili. In particolare, ho completamente riscritto il servizio di *alert*, poiché quelli nativi del *framework Angular* non sono completamente funzionanti su dispositivi mobili. Al suo posto ho realizzato un servizio che fa uso delle librerie *Ionic Native* per mostrare messaggi di *alert* nativi per dispositivi *Android* e *iOS*. Ho inoltre realizzato un servizio di *toast*, che permette di visualizzare messaggi da parte dell'applicazione sottoforma di *toast*. Entrambi i servizi di avviso sono quindi visualizzati come elementi nativi del sistema operativo mobile, come mostrato dalla seguente figura.



Figura 3.12: Esempi di, rispettivamente, *toast* e *alert*.

Riassumendo, durante l'attività di codifica ho sviluppato 12 maschere, arrivando così a coprire tutti i casi d'uso previsti dal mio *stage* e da quello precedente. Ho inoltre realizzato 8 servizi, 2 dei quali completamente originali e 6 trasposti dall'applicativo *web* di partenza. La seguente tabella illustra una stima indicativa, precisa nell'ordine delle centinaia, del numero di *SLOC* (*single lines of code*) per ogni linguaggio utilizzato del prodotto che ho sviluppato, indicando anche quante sono originali e quante provengono dal precedente progetto di *stage*.

	Originali	Progetto precedente	Totale
TypeScript	2400	1200	3600
HTML	700	0	700
CSS	600	0	600
Commenti	1200	0	1200
Totale	5200	1200	6100

Tabella 3.2: Numero indicativo di *SLOC* del prodotto che ho sviluppato.

3.4 Verifica

Sempre a causa del fatto che il mio progetto di *stage* si è basato in gran parte su un prodotto software già completo, ho iniziato il processo di verifica verso la fine dell’attività di codifica: gran parte delle funzioni cruciali, per le quali un *testing* precoce è di indubbia utilità, appartengono infatti ai servizi già precedentemente sviluppati. Ho quindi reputato di scarsa utilità attuare il processo di verifica per la prima parte di sviluppo, poiché in tale periodo mi sono occupato principalmente di codificare in *HTML* e *CSS*; sebbene in questo periodo io abbia creato la maggior parte dei *component* e delle *page*, sviluppate in *TypeScript*, è inoltre vero che il *tool IonicCLI* si occupa di creare i test di unità per il codice automaticamente generato.

Quando mi sono avvicinato alla fine dell’attività di codifica ho quindi iniziato il processo di verifica: a questo fine ho svolto due attività, che sono state l’analisi statica e la scrittura di test di unità.

3.4.1 Analisi statica

L’analisi statica è un’attività che prevede la valutazione di un sistema nella sua interezza o di un suo sottoinsieme in base alla sua forma senza la necessità di eseguire il codice. Per quanto riguarda questa attività ho fatto uso principalmente dei metodi *walkthrough* e *inspection*:

- **Walkthrough:** lettura a largo spettro e senza assunzione del codice, mirata a rilevare la presenza di difetti. Ho utilizzato questa tecnica nel momento in cui ho percorso manualmente il codice prodotto simulando possibili esecuzioni;
- **Inspection:** lettura mirata di determinati punti critici del codice. Per utilizzare questa tecnica ho redatto una lista di controllo informale, basata sui punti da me valutati come cruciali anche in seguito all’utilizzo del metodo *walkthrough*.

Ho inoltre fatto uso di un *linter*, ossia di un programma di utilità che permette di verificare staticamente il codice in base a parametri predefiniti; nel mio caso ho utilizzato *ESLint*⁸, ossia il *linter* di riferimento per codice *JavaScript* e *TypeScript*. L’utilizzo di questo strumento, correttamente configurato, mi ha permesso di rendere tutto il codice *TypeScript* che ho prodotto aderente agli standard di qualità più stringenti. *ESLint*, inoltre, non si occupa solo della formattazione del codice, ma è anche utile a individuare porzioni di codice duplicato o inutilizzato: ciò mi ha permesso di rilasciare sui *server* di produzione un codice pulito e minimale.

```
enrico@enrico-thinkpad ~/.../webapp/ionic-webapp feature/ionic $ npm run lint
> ionic-webapp@0.0.1 lint /home/enrico/Repository/synctrace/webapp/ionic-webapp
> ng lint

Linting "app"...
All files pass linting.
```

Figura 3.13: Risultato dell’esecuzione del *linter* sul codice da me prodotto.

Ho attuato il metodo *walkthrough* e utilizzato il *linter* per tutti i *file TypeScript* dell’applicazione contenenti logica, compresi i *file* contenenti gli *unit test*, per un totale di 53 *file* esaminati; come dimostrato dalla precedente immagine, tutti i *file* analizzati hanno superato i controlli di qualità.

⁸ *ESLint - Pluggable JavaScript Linter*. URL: <https://eslint.org/>.

3.4.2 Test di unità

Parallelamente all'analisi statica ho sviluppato anche dei test di unità; questi sono test che permettono di verificare le singole unità (*i.e.* funzioni, metodi e classi) al fine di determinare se tali unità si comportano nella maniera desiderata.

Per realizzare i test di unità ho utilizzato gli strumenti nativi di *Angular*. Questo *framework*, e di conseguenza anche *Ionic*, al momento della generazione di un componente genera anche il codice *boilerplate* per l'implementazione di questi test, di cui la seguente immagine è un esempio; nello specifico, per ogni *file* contenente codice *TypeScript* viene generato parallelamente un altro *file* *TypeScript* con lo stesso nome, a cui viene aggiunto il termine *spec* a indicare che si tratta di un *file* contenente test.

```

1 import { async, ComponentFixture, TestBed } from '@angular/core/testing';
2 import { IonicModule } from '@ionic/angular';
3 import { RouterTestingModule } from '@angular/router/testing';
4 import { WelcomePage } from './welcome.page';
5
6 describe('WelcomePage', () => {
7   let component: WelcomePage;
8   let fixture: ComponentFixture<WelcomePage>;
9
10  beforeEach(async(() => {
11    TestBed.configureTestingModule({
12      declarations: [ WelcomePage ],
13      imports: [RouterTestingModule, IonicModule.forRoot()]
14    }).compileComponents();
15
16    fixture = TestBed.createComponent(WelcomePage);
17    component = fixture.componentInstance;
18    fixture.detectChanges();
19  }));
20
21  it('should create', () => {
22    expect(component).toBeTruthy();
23  });
24});
```

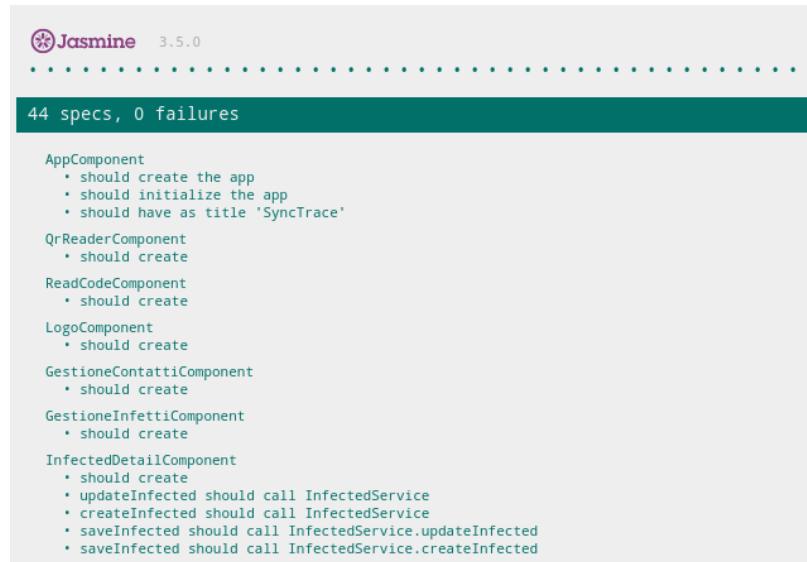
Figura 3.14: Esempio di test di unità *TypeScript*.

Lo strumento che si occupa del *testing* in *Angular*, e quindi in *Ionic*, è **Jasmine**⁹, un software consistente in un *framework* di test per codice *JavaScript* e *TypeScript*; questo viene eseguito tramite il motore *Karma*¹⁰, che si occupa di eseguire i test e di mostrarli su un *browser* tramite un *server* locale, come si può evincere dall'immagine che segue.

In totale ho realizzato 44 test di unità, alcuni dei quali (principalmente delle funzioni riguardanti i *service*) corrispondenti a quelli precedentemente sviluppati.

⁹ Jasmine Framework per il testing in JavaScript. URL: <https://jasmine.github.io/>.

¹⁰ Karma - Spectacular Test Runner for JavaScript. URL: <https://karma-runner.github.io/latest/index.html>.



The screenshot shows the Jasmine test runner interface. At the top, it displays the logo and version "Jasmine 3.5.0". Below this is a green header bar with the text "44 specs, 0 failures". The main area contains a list of component names, each followed by a bullet-pointed list of test cases:

- AppComponent
 - should create the app
 - should initialize the app
 - should have as title 'SyncTrace'
- QrReaderComponent
 - should create
- ReadCodeComponent
 - should create
- LogoComponent
 - should create
- GestioneContattiComponent
 - should create
- GestioneInfettiComponent
 - should create
- InfectedDetailComponent
 - should create
 - updateInfected should call InfectedService
 - createInfected should call InfectedService
 - saveInfected should call InfectedService.updateInfected
 - saveInfected should call InfectedService.createInfected

Figura 3.15: Porzione dell'output di *Jasmine* per il corrente progetto.

3.5 Validazione e collaudo

Contemporaneamente all'avvio del processo di verifica ho cominciato anche il processo di collaudo, al fine di verificare il funzionamento di tutte le funzionalità anche su dispositivi mobili. Una volta concluso il collaudo sono passato alla validazione, momento in cui ho analizzato insieme al tutor esterno tutti i requisiti che avevo precedentemente definito.

Collaudo

Come anticipato nel paragrafo precedente, parallelamente alla realizzazione dei test di unità e all'analisi statica del codice ho anche collaudato l'applicazione su un dispositivo *mobile*. Vale la pena puntualizzare che non ho effettuato la codifica dell'applicazione a *black-box*, ossia senza controllare periodicamente i risultati di quanto sviluppato; il *framework Ionic*, infatti, possiede una funzionalità di *serve*, che permette la ricarica dinamica dell'applicativo ogniqualvolta venga fatta una modifica al codice. Questo applicativo, però, viene visualizzato in locale nella macchina sul quale sta venendo eseguito lo sviluppo, e più nello specifico viene visualizzato e utilizzato tramite *browser web*, come mostrato in foto. Questo è un ottimo strumento di supporto all'attività di codifica, ma non permette di usare l'applicazione su un dispositivo mobile; alcune funzionalità native, infatti, non possono essere utilizzate.

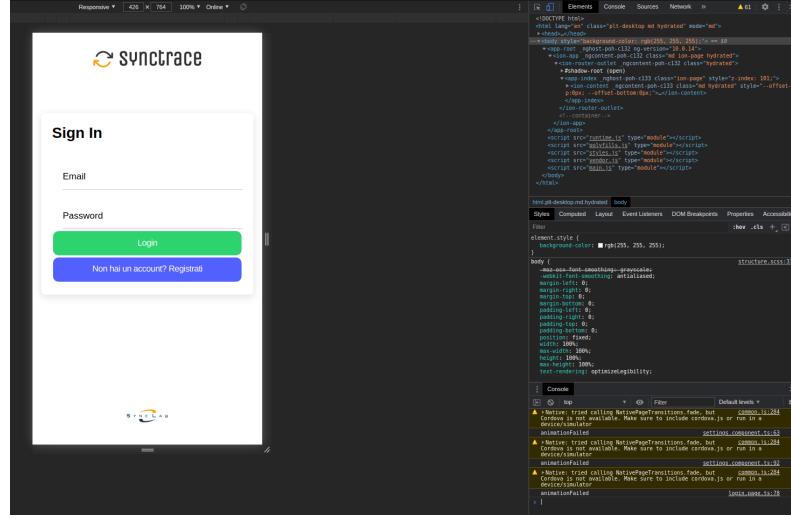


Figura 3.16: Esempio di utilizzo di *ionic serve* su browser Chromium.

Verso la fine dell'attività di codifica ho quindi dovuto poter testare l'applicazione su uno *smartphone*. Nel fare ciò, però, è sorto un problema: fino a quel momento ho eseguito la componente *back-end*, necessaria al funzionamento della maggior parte delle funzionalità dell'applicazione, in locale sulla mia macchina, e questo mi ha reso impossibile il collaudo su un dispositivo mobile. Al fine di poter eseguire questo collaudo, quindi, ho dovuto effettuare il *deploy* del *server back-end* su un *server aziendale*. Per fare ciò, mi è stato proposto dal tutor aziendale di containerizzare tale componente tramite la piattaforma *Docker*. Prima di procedere al collaudo, ho quindi interrotto le attività di codifica e verifica per dedicarmi alla trasposizione su *docker machine* del *back-end*. Ho quindi scritto due *file yml*, di cui l'immagine sottostante è un esempio, necessari al comando *docker-compose* di *Docker* per costruire le corrette macchine *docker* e i rispettivi volumi per la persistenza dei dati.

```

1 version: "3.7"
2
3 services:
4
5   postgres-db:
6     image: postgres:12.4-alpine
7
8     expose:
9       - 5432
10
11    restart: always
12
13    environment:
14      POSTGRES_USER: postgres
15      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:?err}
16      POSTGRES_DB: synctrace
17
18    volumes:
19      - type: volume
20        source: postgres-db-data
21        target: /var/lib/postgresql/data
22
23 discovery-server:
24   image: discovery-server:0.0.1-SNAPSHOT
25
26   restart: always
27
28   expose:
29     - 8761
30
31   environment:
32     SPRING_APPLICATION_JSON: '{
33       "server.port": "8761"
34     }'
35
36   depends_on:
37     - postgres-db
38

```

Figura 3.17: Porzione del *docker-compose* per i servizi del *back-end*.

Una volta effettuato il *deploy* su un *server aziendale*, quindi, sono passato al collaudo

vero e proprio dell'applicazione. Ho fatto uso dello strumento *Capacitor*¹¹, ossia un *runtime* che permette la trasposizione di codice *TypeScript* utilizzato all'interno di un progetto *Ionic* in un'applicazione *Android*. Questa applicazione consiste in un *wrapper* che effettua il *render* di una *webview* su dispositivi mobili *Android*, e per l'installazione di questa sul mio dispositivo ho utilizzato *adb* (*Android Debug Bridge*), strumento che mi ha permesso di effettuare anche il *debug* dell'applicazione.

Ho quindi redatto una lista di funzionalità di cui controllare il comportamento e l'ho seguita, determinando gli accorgimenti da intraprendere per rendere l'applicazione pienamente funzionante e usabile. Ho infine ripreso l'attività di codifica per correggere gli errori e i malfunzionamenti che sono risultati da questo collaudo.

Validazione

Alla conclusione delle precedenti attività ho analizzato quanto svolto insieme al tutor esterno, in funzione del soddisfacimento dei requisiti definiti. Il lavoro che ho svolto è stato apprezzabile, poiché ho assolto la maggior parte dei requisiti previsti; nella seguente tabella riassumo i risultati ottenuti:

	Total	Soddisfatti	Non soddisfatti	Percentuale
Funzionali	26	25	1	96%
Qualitativi	2	2	0	100%
Di vincolo	3	3	0	100%

Tabella 3.3: Requisiti soddisfatti e non soddisfatti.

Da questa tabella si può quindi vedere che la maggior parte dei requisiti è stata soddisfatta. L'unico requisito che non ho assolto è il requisito funzionale e opzionale riguardante la *localization*, ossia la traduzione dell'applicazione in lingue diverse da quella italiana; il motivo di ciò è da ricercarsi nella mancanza di tempo che ho avuto all'approssimarsi della fine del mio *stage*.

3.6 Risultati ottenuti

Una volta terminato lo sviluppo e validato il prodotto da me creato ho prodotto la documentazione tecnica, al fine di rendere il più possibile esplicite le scelte progettuali che ho effettuato e di rendere trasparente il codice all'azienda e a chi, possibilmente, effettuerà uno *stage* basato sulla porzione di prodotto sviluppata da me. Nel fare questo mi sono affidato allo strumento *Compodoc*; questo software, che si presenta come libreria di *npm*, provvede a creare in automatico la documentazione a partire da commenti strutturati e standardizzati al codice, dei quali quello che segue è un esempio.

```

178  /**
179   * Funzione che gestisce la corretta lettura di un codice QR. Viene controllata la presenza dell'utente scannerizzato:
180   * - all'interno dell'array degli utenti infetti tramite il servizio 'infectedService';
181   * - all'interno dell'array degli utenti a rischio tramite il servizio 'riskService'.
182   * Viene quindi mostrato un alert di conseguenza.
183   * @param result Stringa letta dal codice QR
184   * @return Promise
185 */

```

Figura 3.18: Esempio di commento a un metodo per la generazione automatica di documentazione con *Compodoc*.

¹¹ Capacitor - Cross-platform native runtime for web apps. URL: <https://capacitorjs.com/>.

Ho quindi documentato tutte le classi, i parametri, i metodi e le funzioni dell'applicativo, compresi quelli non sviluppati da me, arrivando a raggiungere una *documentation coverage* del **100%**.

Al concludersi del mio progetto di *stage* ho quindi integrato quanto ho sviluppato nell'intero sistema *SyncTrace*. La somma del mio e degli altri progetti di *stage* ha quindi dato vita a un prodotto completo, che copre i bisogni dettati dal tracciamento dei contatti a tutto tondo, soddisfacendo sia la necessità di tracciare i contatti tra persone che quella di gestire tali dati, nel caso dei medici, e di utilizzarli per mettere in sicurezza la propria attività come nel caso degli esercenti. La porzione di prodotto sviluppato da me, in particolare, semplifica a queste ultime due figure lo svolgimento dei rispettivi ruoli.

Capitolo 4

Valutazioni retrospettive

4.1 Soddisfacimento degli obiettivi

Una volta conclusi tutti i processi e le attività di sviluppo e documentazione ho analizzato insieme al tutor aziendale gli obiettivi fissati ad inizio *stage*, al fine di individuare il grado di soddisfacimento di questi e poter effettuare un’analisi a posteriori di cosa avrei potuto gestire meglio. In questa tabella sono riportati gli obiettivi fissati, il loro grado di soddisfacimento e una breve nota che sintetizza questo soddisfacimento.

Obiettivo	Risultato	Note
0-01	Soddisfatto	ABC
0-02	Soddisfatto	DEF
0-03	Soddisfatto	GHI
0-D1	Soddisfatto	LMN
0-D2	Soddisfatto	OPQ
0-F1	Soddisfatto	RST
0-F2	Soddisfatto	UVZ

Tabella 4.1: Grado di soddisfacimento degli obiettivi di *stage*.

Mi sono inoltre impegnato a consegnare tutti i prodotti richiesti dall’azienda; questi sono riassunti dalla seguente tabella.

Prodotto

Codice	Durante tutto il processo di sviluppo ho utilizzato il repository aziendale riservato agli stage.
Documentazione	La documentazione che ho consegnato consiste nella documentazione tecnica. Questa è interattiva e permette di visualizzare i dati in modo dinamico.
Containerizzazione	Per quanto concerne la containerizzazione, ho rilasciato sul repository precedentemente citato.

4.2 Bilancio formativo**4.2.1 Maturazione professionale****4.2.2 Rapporto tra università e lavoro**

Bibliografia

Siti web consultati

Angular Framework. URL: <https://angular.io/> (cit. a p. 6).

Capacitor - Cross-platform native runtime for web apps. URL: <https://capacitorjs.com/> (cit. a p. 38).

Compodoc - The missing documentation tool for your Angular application. URL: <https://compodoc.app/> (cit. a p. 17).

Consortium, The W3. HTML& CSS. URL: <https://www.w3.org/standards/webdesign/htmlcss.html> (cit. a p. 6).

Corporation, Oracle. Java. URL: <https://docs.oracle.com/javase/7/docs/technotes/guides/language/> (cit. a p. 6).

Docker: Empowering App Development for Developers. URL: <https://www.docker.com/> (cit. a p. 17).

Electron - Build cross-platform desktop apps with JavaScript, HTML, and CSS. URL: <https://www.electronjs.org/> (cit. a p. 8).

ESLint - Pluggable JavaScript Linter. URL: <https://eslint.org/> (cit. a p. 34).

Foundation, Mozilla. JavaScript. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (cit. a p. 6).

Il pacchetto LearnYouNode. URL: <https://www.npmjs.com/package/learnyounode> (cit. a p. 19).

Il tool AngularCLI. URL: <https://cli.angular.io/> (cit. a p. 30).

Il tool IonicCLI. URL: <https://ionicframework.com/docs/cli> (cit. a p. 30).

INC., Apple. Swift. URL: <https://developer.apple.com/swift/> (cit. a p. 7).

Ionic - Cross-Platform Mobile App Development. URL: <https://ionicframework.com/> (cit. a p. 14).

Jasmine Framework per il testing in JavaScript. URL: <https://jasmine.github.io/> (cit. a p. 35).

Karma - Spectacular Test Runner for JavaScript. URL: <https://karma-runner.github.io/latest/index.html> (cit. a p. 35).

Kent Beck Jeff Sutherland, Ken Schwaber et al. *Manifesto for Agile Software Development.* URL: <https://agilemanifesto.org/> (cit. a p. 4).

- Kotlin Programming Language.* URL: <https://kotlinlang.org/> (cit. a p. 7).
- Libreria Angular ZXingScannerModule.* URL: <https://www.npmjs.com/package/@zxing/ngx-scanner> (cit. a p. 29).
- Libreria Ionic Native.* URL: <https://ionicframework.com/docs/native> (cit. a p. 29).
- Metodologia SCRUM.* URL: <https://www.scrum.org/> (cit. a p. 4).
- Ore, Il Sole 24. *Cronistoria del Coronavirus.* URL: <https://lab24.ilsole24ore.com/storia-coronavirus/> (cit. a p. 12).
- PostgreSQL: The world's most advanced open source database.* URL: <https://www.postgresql.org/> (cit. a p. 16).
- React - Una libreria JavaScript per creare interfacce utente.* URL: <https://it.reactjs.org/> (cit. a p. 28).
- Runtime Node.JS.* URL: <https://nodejs.org/> (cit. a p. 8).
- Spring Framework.* URL: <https://spring.io/> (cit. a p. 6).
- TypeScript: Typed JavaScript at Any Scale.* URL: <https://www.typescriptlang.org/> (cit. a p. 6).
- Vue Framework.* URL: <https://vuejs.org/> (cit. a p. 28).
- WireframeSketcher: Wireframing tool for professionals.* URL: <https://wireframesketcher.com/> (cit. a p. 28).