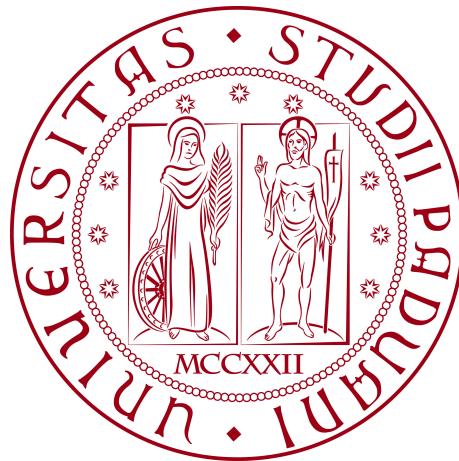


**Università degli Studi di Padova**

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**MoviORDER, modulo agenti per gestione  
clienti**

*Tesi di Laurea Triennale in Informatica*

*Relatore*

Prof. Vardanega Tullio

*Laureando*

Oseliero Antonio

Matricola 1226325



# Sommario

Il presente documento descrive il lavoro svolto dallo studente Oseliero Antonio durante il suo *stage* presso l’azienda VisioneImpresa Software House.

Lo scopo del tirocinio è stato studiare il codice dell’applicazione mobile MoviORDER e sviluppare un modulo di autenticazione di agenti aziendali in un’applicazione pensata per clienti terzi. Più precisamente era richiesto che, dopo l’autenticazione, l’agente possa scegliere un cliente da una lista e operare all’interno dell’applicazione come il cliente selezionato senza la necessità di autenticarsi come tale.

Raggiungere questo obiettivo ha richiesto lo studio del codice e dell’architettura dell’applicazione: ***front-end, back-end e base dati sottostante***, e un certo insieme di tecnologie e strumenti tra i quali **React Native e ASP.NET Core, Visual Studio, e Server Management Studio**. Il progetto ha incluso la realizzazione delle API e l’interfaccia grafica del modulo richiesto, insieme a una batteria di test automatici per la *Business Logic* di alcune di tali API.

Ho segnalato tutte le parole non italiane in *corsivo* all’interno del documento.

Utilizzo il carattere **monospaziato** per i nomi di tavole e colonne del *database*, classi, funzioni, *component*, *view* o altre parti del codice.

Evidenzio le parole del glossario con una G a pedice, in corsivo e in blu (ad esempio *CSR<sub>G</sub>*).

Uso il **grassetto** per enfatizzare la parola chiave di un punto elenco per migliorare la leggibilità e rendere immediatamente identificabili i concetti chiave.

Ecco come appare un elenco puntato:

---

"Elenco numeri primi:

- 1
  - 2
  - 3
- ..."

Ho diviso il documento in 4 macro sezioni:

**Capitolo 1 - VisioneImpresa:** Descrivo l'azienda dove ho svolto il tirocinio, riportando brevemente clienti, prodotti, organizzazione aziendale, strumenti e tecnologie utilizzate, e la propensione all'innovazione.

**Capitolo 2 - Descrizione del progetto:** Presento il progetto assegnatomi dall'azienda, specificando obiettivi, vincoli tecnologici e temporali. Approfondisco inoltre il rapporto dell'azienda con gli *stage* in generale e le motivazioni dietro la scelta di questo specifico progetto.

**Capitolo 3 - Stage:** Racconto la mia esperienza di *stage*.

**Capitolo 4 - Retrospettiva:** Offro un giudizio obiettivo sul raggiungimento degli obiettivi di *stage*, personali e aziendali. Faccio inoltre un resoconto delle conoscenze acquisite con questa esperienza e una valutazione personale del percorso di studi universitario.

# Indice

<b>1 VisioneImpresa</b>	<b>1</b>
1.1 L’azienda . . . . .	1
1.2 Clienti e servizi . . . . .	2
1.3 Organizzazione aziendale . . . . .	5
1.3.1 Aree di competenza . . . . .	5
1.3.2 Metodologie di sviluppo <i>software</i> . . . . .	7
1.4 Tecnologie . . . . .	9
1.4.1 Elenco delle tecnologie utilizzate . . . . .	9
1.4.2 Integrazione delle tecnologie con i processi aziendali . . . . .	12
1.5 Propensione all’innovazione . . . . .	15
<b>2 Descrizione e pianificazione del progetto di stage</b>	<b>16</b>
2.1 Strategia aziendale e rapporto con gli <i>stage</i> . . . . .	16
2.2 Descrizione progetto . . . . .	17
2.3 Scelta dell’attività di <i>stage</i> . . . . .	19
2.4 Vincoli . . . . .	20
2.4.1 Vincoli tecnologici . . . . .	20
2.4.2 Vincoli temporali . . . . .	23
2.5 Obiettivi . . . . .	24
2.5.1 Obiettivi aziendali . . . . .	24
2.5.2 Obiettivi personali . . . . .	24
<b>3 Sviluppo del modulo agenti</b>	<b>26</b>
3.1 Analisi . . . . .	26
3.1.1 Casi d’uso . . . . .	26

## INDICE

---

3.1.2	Requisiti funzionali e non funzionali . . . . .	28
3.2	Progettazione . . . . .	34
3.2.1	Struttura del <i>database</i> . . . . .	34
3.2.2	Architettura delle API . . . . .	35
3.2.2.1	<i>Data Access layer e Repository Pattern</i> . . . . .	37
3.2.2.2	<i>Business Logic layer e Service Pattern</i> . . . . .	38
3.2.2.3	DTO . . . . .	39
3.2.2.4	<i>Presentation layer e API Controller</i> . . . . .	40
3.2.3	<i>Front-end</i> . . . . .	42
3.2.3.1	<i>Root della repository e file di configurazione</i> . .	42
3.2.3.2	Architettura React . . . . .	44
3.3	Codifica . . . . .	46
3.3.1	<i>Database e Data Access layer</i> . . . . .	46
3.3.2	API e <i>Business Logic layer</i> . . . . .	47
3.3.3	<i>Front-end</i> . . . . .	48
3.4	Verifica e documentazione . . . . .	49
3.4.1	<i>Testing</i> . . . . .	49
3.4.2	Documentazione . . . . .	51
3.5	Risultati . . . . .	52

<b>Glossario</b>	<b>i</b>
------------------	----------

<b>Sitografia</b>	<b>iii</b>
-------------------	------------

# Elenco delle figure

1.1	Obiettivi delle società benefit. . . . .	2
1.2	Organizzazione di uno <i>sprint</i> con il <i>framework</i> Scrum . . . . .	8
1.3	Integrazione metodologie e tecnologie . . . . .	12
2.1	Alcune <i>view</i> di MoviORDER . . . . .	17
2.2	Tecnologie utilizzate per lo sviluppo di MoviORDER . . . . .	20
2.3	Pianificazione attività di <i>stage</i> . . . . .	23
3.1	<i>Use Cases</i> modellati per il progetto . . . . .	27
3.2	<i>Database</i> di MoviORDER. . . . .	34
3.3	Architettura API. . . . .	36
3.4	<i>Root</i> della <i>repository</i> che contiene il <i>front-end</i> . . . . .	42
3.5	Rappresentazione dell’architettura React per MoviORDER . . . . .	44
3.6	Versione finale delle <i>homepages</i> di MoviORDER . . . . .	48

# Elenco delle tavole

2.1	Tabella Obiettivi . . . . .	24
2.2	Tabella Obiettivi Personalisi . . . . .	25

## ELENCO DELLE TABELLE

---

3.1	Tabella del tracciamento dei requisiti funzionali e non funzionali.	33
3.2	Riepilogo requisti . . . . .	34



# Capitolo 1

## VisioneImpresa

### 1.1 L'azienda

VisioneImpresa è un'azienda con quarant'anni di esperienza nell'offrire a piccole e medie imprese soluzioni informatiche per la gestione e l'automazione dei processi aziendali. I suoi prodotti di punta sono infatti sistemi *ERP<sub>G</sub>* (*Enterprise Resource Planning*) ovvero sistemi che permettono di coordinare il flusso di dati tra i processi di un'azienda, fornendo un'unica fonte di informazioni e semplificandone le operazioni.

VisioneImpresa è situata a Pernumia (Padova) e opera in prevalenza nel Nord Italia, dal 2016 è entrata a far parte del gruppo Officegroup, azienda che riunisce diverse *software house* specializzate nella progettazione e sviluppo di sistemi gestionali evoluti. Dal 2023 inoltre è diventata una società *benefit*, ovvero è un'azienda che opera con l'obiettivo di generare un impatto positivo sulla società e sull'ambiente, oltre al profitto finanziario.



**Figura 1.1:** Obiettivi delle società benefit.

fonte: <https://www.vsh.it/azienda/societa-benefit/>

La figura 1.1 mostra le iniziative promosse da questo tipo di società, dalla dematerializzazione e digitalizzazione alla promozione di politiche a sostegno della conciliazione vita-lavoro.

Altri obiettivi delle società *benefit* possono essere: investire nelle energie rinnovabili e la sostenibilità, l'investimento in tecnologie ad alta efficienza energetica, rispetto della parità di genere, formazione professionale del lavoratore, progetti con scuole ed università, co-progettazione con associazioni e istituzioni del territorio con il duplice obiettivo di stimolare la partecipazione dei dipendenti a “buone cause” della comunità e valorizzare il lavoro di associazioni no-profit del territorio, generando così valore sociale.

## 1.2 Clienti e servizi

VisioneImpresa ha come clienti piccole e medie imprese situate in prevalenza in Veneto e in generale nel Nord Italia, possiamo trovare però anche clienti dal Centro Italia e dalla Sardegna.

Il gestionale che propone può adattarsi a qualsiasi tipo di azienda indipendentemente dal settore in cui operi (anche se come vedremo vengono venduti dei gestionali ad hoc per i settori: petrolifero, ittico, assistenza post-vendita, ortofrutticolo, antincendi e antinfortunistica, trasporti).

Una volta implementato il gestionale all'interno dell'azienda del cliente viene

offerta una formazione all'utilizzo del *software* per i dipendenti, che parteciperanno a delle riunioni tenute da un consulente tecnico che ne illustrerà le funzionalità e insegnerrà come sfruttarle al meglio.

VisioneImpresa propone due linee di prodotti principali: Vision e MoviDAT. La prima, Vision, è la linea di gestionali dell'azienda, con VisionENTERPRISE, che è il loro *ERP G* di punta, e poi una serie di soluzioni verticali per venire incontro alle specifiche esigenze delle varie aziende con cui VisioneImpresa opera. Ognuna delle soluzioni verticali offerte dall'azienda è una variazione di VisionENTERPRISE, che viene arricchita con funzionalità specifiche per adattarsi a specifici settori. In particolare quindi nella linea Vision abbiamo:

- **VisionENTERPRISE**, *ERP G* di punta dell'azienda e dedicato ad imprese che non hanno necessità di funzionalità specifiche.
- **VisionENERGY**, gestionale con specifiche funzionalità pensate per le aziende che lavorano nel settore petrolifero, come la possibilità di gestire la vendita di carburante, manutenzione valvole, ecc.;
- **VisionBLUE**, gestionale con specifiche funzionalità pensate per le aziende che lavorano nel settore ittico, come la possibilità di gestire lotti, prodotti e imballaggi;
- **VisionASSISTANCE**, gestionale con specifiche funzionalità pensate per le aziende specializzate nell'assistenza post-vendita, come la possibilità di gestire richieste di assistenza, contratti e assegnare gli ordini di intervento ai singoli tecnici;
- **VisionFRESH**, gestionale con specifiche funzionalità pensate per le aziende che lavorano nel settore ortofrutticolo, come la possibilità di gestire movimentazione merce, inserimento pesate, interfacciamento con bilance elettroniche, ecc.;
- **VisionANTINCENDI**, gestionale con specifiche funzionalità pensate per le aziende che lavorano nel settore antincendi e antinfortunistica, come la possibilità di gestire chiamate ed interventi straordinari, buoni di manutenzione e geolocalizzare gli interventi;

- **VisionTRASPORTI**, gestionale con specifiche funzionalità pensate per le aziende che lavorano nel settore trasporti, come la possibilità di gestire listini, anagrafiche, dotazioni, manutenzione, pianificazione viaggi, ecc.

Nella linea di prodotti MoviDAT invece troviamo una gamma di applicazioni sviluppate per i principali sistemi operativi per dispositivi *mobile*: Android e iOS. Queste applicazioni sono state sviluppate per integrarsi direttamente con i gestionali della linea Vision e permettono di semplificare il lavoro di dipendenti che operano in mobilità e non hanno a disposizione un *computer* con cui lavorare durante le trasferte (ed anche se ce lo avessero il suo utilizzo risulterebbe scomodo).

In questa linea dunque troviamo:

- **MoviDOC** è un *web appG* (ovvero un *app* a cui è possibile accedere direttamente da *browser* senza necessità di installarla sul dispositivo) che consente la gestione e condivisione dei documenti;
- **Handy** è un *app* per palmare che integrata a VisionENTERPRISE supporta la movimentazione della merce del magazzino o del punto vendita;
- **MoviSELL** è un *app* sviluppata per *tablet* iOS dedicata agli agenti aziendali, permette di: visualizzare i clienti su una mappa, avere visibilità dello stato contabile e inserire ordini clienti direttamente nel ciclo attivo dell'azienda.
- **MoviREP** è un *app* sviluppata per *tablet* iOS per la gestione digitalizzata dei rapportini da parte di operatori addetti alla manutenzione o all'assistenza post vendita.
- **MoviALERT** è una *web appG* che permette di inviare *mail* di notifica automatiche all'avvenire di specifici eventi nel gestionale;
- **MoviCHECK** è un *web appG*, scaricabile anche su dispositivi Android e iOS per consultare i dati di *business* in mobilità;
- **MoviEXPENSE** è un *app* per Android e iOS, per la registrazione automatica delle note spese;

- **MoviCHECKIN** è una *web app* per la registrazione dei visitatori in azienda;
- **MoviORDER** applicazione per *smartphone e tablet* iOS e Android che l’azienda può fornire ai propri clienti per l’invio di ordini e richieste di approvvigionamento.

Nel caso in cui un’azienda richieda funzionalità specifiche per uno dei *software* sopra elencati, VisioneImpresa offre la possibilità di creare una versione modificata dei propri prodotti. Per evitare di avere troppe variazioni della stesso prodotto il codice delle personalizzazioni (così vengono chiamate le funzioni in più richieste dal cliente) vengono inserite direttamente nel codice del *software* principale, e "attivate" da specifici parametri controllati all'avvio del sistema. Nel caso di MoviORDER, che ho avuto la possibilità di esaminare per questo progetto, a seconda del valore del campo *Company* ottenuto a seguito dell'autenticazione del cliente venivano apportate alcune variazioni grafiche (loghi, tema). Questo si può ottenere grazie ad un'attenta progettazione e appropriate scelte architettoniche.

## 1.3 Organizzazione aziendale

### 1.3.1 Aree di competenza

VisioneImpresa è strutturata in tre aree di competenza, ognuna con ruoli e responsabilità specifiche.

Reparto Assistenza: qui operano i consulenti tecnici gestionali, il cui compito è assistere l’azienda nell’implementazione dei nuovi gestionali e nella gestione del cambiamento assicurandosi che il personale aziendale sia formato sull’uso delle nuove tecnologie. Ogni consulente è responsabile di uno o più *software* di cui hanno un’ampia conoscenza operativa. Inoltre, forniscono assistenza ai clienti, aiutandoli nella risoluzione dei problemi e, se necessario, segnalando le problematiche al reparto sviluppo che aprirà quindi un *ticket* all’interno della piattaforma Jira (vedi capitolo 1.4).

Area Amministrazione Commerciale e *Marketing*: In quest'area si trovano diverse competenze, tra cui:

- **Responsabile marketing:** ha il compito di realizzare strategie per promuovere l'azienda e i suoi prodotti ai potenziali clienti;
- **Risorse umane:** ha il compito di amministrare stipendi, pensioni e *bene-fit*, nonché di assicurarsi il rispetto da parte dell'azienda delle normative sul lavoro;
- **Contabilità:** ha il compito di gestire e registrare le transazioni finanziarie, garantendo che tutte le attività economiche siano documentate in modo accurato e trasparente;
- **Segreteria generale:** ha il compito di gestire e indirizzare le chiamate in entrata, gestire la posta elettronica e la corrispondenza, pianificare eventi aziendali;
- **Segreteria commerciale:** ha il compito di mantenere le comunicazioni con i clienti fornendo informazioni su prodotti e servizi e preparando offerte commerciali, contratti di vendita e documenti correlati;
- **Amministrazione ciclo attivo:** ha il compito di garantire una gestione efficiente delle vendite e della riscossione dei pagamenti;
- **Commerciale rete diretta:** ha il compito di occuparsi della vendita dei prodotti direttamente ai clienti finali;
- **Commerciale rete indiretta:** ha il compito di gestire le vendite attraverso intermediari come distributori, rivenditori, agenti o partner commerciali;
- **Responsabile d'impatto:** si occupa della valutazione, pianificazione e promozione delle misure di responsabilità sociale d'impresa (*CSR<sub>G</sub>*), ovvero di tutte le iniziative attuate dall'azienda in ambito sociale e di transizione ecologica.

- **Amministratore** è responsabile di dirigere e gestire l'azienda nel suo complesso, assicurando che tutti i dipartimenti e le attività lavorino insieme per raggiungere gli obiettivi strategici e operativi.
- **Product manager** ha il compito di assicurare che i prodotti siano sviluppati in linea con le esigenze del mercato, lanciati con successo e gestiti efficacemente durante il loro ciclo di vita.

Reparto Sviluppo *Software*: In quest'area troviamo:

- **Sviluppatori**: possono essere *front-end*, specializzati nello sviluppo di interfacce e della gestione dell'interazione uomo-macchina, *back-end* specializzati nello sviluppo della logica del *software* e nella manipolazione dei dati, o *full-stack*, in grado di operare sia come sviluppatore *front-end* che *back-end*.
- **Project manager**: ha il compito di assicurarsi che vengano rispettati obiettivi, tempi, costi e vengano soddisfatti i parametri di qualità;
- **Direttore dello sviluppo**: ha il compito di prendere le decisioni implementative e scegliere l'architettura del *software*, si occupa inoltre di dirigere il team e di pianificare e assegnare i lavori da svolgere;
- **Analista**: si occupa di interagire con i clienti per delineare i requisiti del progetto *software* e documentarli in un documento di analisi.

### 1.3.2 Metodologie di sviluppo *software*

Ho potuto notare, durante la mia esperienza di tirocinio, che gli sviluppatori utilizzano metodologie Agile per la gestione dei loro progetti.

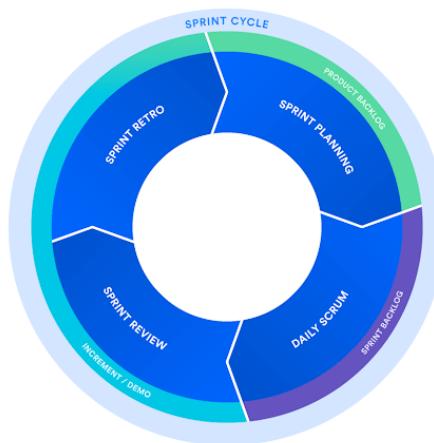
Le metodologie Agile sono un'approccio alla gestione dei progetti che prevede la suddivisione del progetto in fasi e del lavoro in cicli brevi, al termine dei quali verranno introdotti cambiamenti che avvicinano il progetto sempre di più al soddisfacimento di tutti i requisiti. Questo approccio è particolarmente adattabile agli imprevisti, permettendo di reagire velocemente e riducendo al minimo i danni, come lo slittamento della data di completamento e conseguentemente

l'aumento di denaro da destinare al progetto. Il manifesto Agile riporta i punti principali punti di questa filosofia <sup>1</sup>:

- Gli **individui e le interazioni** più che i processi e gli strumenti;
- Il **software funzionante** più che la documentazione esaustiva;
- La **collaborazione col cliente** più che la negoziazione del contratto;
- **Rispondere al cambiamento** più che seguire un piano.

Sebbene il manifesto Agile, pubblicato nel 2001, abbia posto le basi di questo approccio, le pratiche Agile si sono notevolmente evolute nel corso degli anni. Attualmente, esistono diverse interpretazioni e implementazioni delle metodologie Agile, adattate alle specifiche esigenze delle organizzazioni e dei *team* di sviluppo.

In particolare, VisioneImpresa adotta il *framework* Agile Scrum, che definisce una serie di principi, pratiche e ceremonie per riuscire ad assimilare nel proprio metodo di lavoro la metodologia Agile. Scrum richiede di suddividere il lavoro in *sprint* dalla durata variabile di una fino a quattro settimane. VisioneImpresa pianifica *sprint* di una settima in modo da rispondere tempestivamente a gli imprevisti ed effettuare una pianificazione più efficace.



**Figura 1.2:** Organizzazione di uno *sprint* con il *framework* Scrum.  
fonte: <https://www.atlassian.com/it/agile/scrum>

---

<sup>1</sup>Manifesto Agile per lo sviluppo *software*, fonte riportata nella sitografia 3.5.

La figura 1.2 mostra come ogni *sprint* è strutturato in una serie di incontri che avvengono solitamente in video chiamata usando 3CX (vedi capitolo 1.4).

Si comincia il lunedì, all'inizio dello *sprint*, quando programmati, direttore dello sviluppo e *project manager* partecipano ad un *meeting* chiamato *sprint planning* dove si pianifica il lavoro da svolgere per lo *sprint* in corso. Quindi ogni giorno si tiene un breve *meeting* prima della pausa pranzo chiamato *daily scrum* dove si discute dello stato dei lavori ed eventuali problemi emersi. Il venerdì si tiene l'ultimo *meeting* dello *sprint* chiamato *sprint review* dove si discute dello stato dei lavori rispetto alle aspettative e discutendo dei problemi emersi durante lo *sprint* si cercano modi per migliorare.

VisioneImpresa organizza inoltre un ulteriore *meeting* a cadenza mensile dove non solo le persone interessate al progetto, ma tutti i dipendenti dell'azienda si riuniscono per discutere dello stato dei lavori di ogni settore: evoluzione dei prodotti, vendite, *feedback* dei clienti, aggiornare il reparto *marketing* e commerciale sulle nuove funzionalità dei *software* ecc.. Questo incontro ha lo scopo di dare a tutti i dipendenti dell'azienda una visione d'insieme evitando il cosiddetto "effetto sottomarino", ovvero quando una persona o un gruppo si focalizzano soltanto in uno specifico ambito, favorendo l'isolamento rispetto al resto dell'azienda, che ha invece bisogno di lavorare coordinando i vari settori.

## 1.4 Tecnologie

### 1.4.1 Elenco delle tecnologie utilizzate

L'azienda utilizza diversi strumenti sia per lo sviluppo, che per lo svolgimento dei normali processi aziendali.

- **Portatili:** ad ogni impiegato viene messo a disposizione un portatile con Windows 10 o 11, il sistema operativo di Microsoft o all'occorrenza un Mac con macOS, il portatile sviluppato da Apple con il suo sistema operativo proprietario;

- **Dispositivi mobile:** all'interno dell'azienda troviamo molti dispositivi *mobile* con diversi sistemi operativi e dimensioni dello schermo, usati per il *testing* delle applicazioni Android e iOS;
- **Microsoft Office 365:** servizio in abbonamento di Microsoft che include diversi *software* come Word e PowerPoint;
- **Zimbra:** sistema di posta elettronica utilizzato dall'azienda, durante il mio *stage* è stato cambiato in favore di un'integrazione di Zimbra con Outlook;
- **Bitbucket:** strumento per la gestione della versione Git basato sul *web*, che consente di creare *repository* pubbliche o private per caricare il proprio codice e gestirlo in modo collaborativo con il proprio *team*;
- **Jira:** *suite* di *software* proprietari per il tracciamento delle segnalazioni sviluppato da Atlassian, che consente il *bug tracking* e la gestione dei progetti;
- **Confluence:** strumento che permette ai *team* di condividere e organizzare documenti e contenuti in un ambiente centralizzato e strutturato;
- **3CX:** centralino telefonico PBX (*Private Branch Exchange*), ovvero una rete telefonica privata utilizzata all'interno di un'azienda o organizzazione. Gli utenti del sistema telefonico PBX possono comunicare internamente ed esternamente, tramite il classico telefono fisso o chat da *smartphone*. Questo sistema permette anche di effettuare video chiamate e di scambiarsi messaggi all'interno di *chat*;
- **Visual Studio:** Si tratta di un ambiente di sviluppo integrato completo (*IDE*) che è possibile usare per scrivere, modificare, eseguire il *debug* e compilare codice. Visual Studio include compilatori, strumenti di completamento del codice, controllo del codice sorgente, estensioni e molte altre funzionalità per migliorare ogni fase del processo di sviluppo *software*;

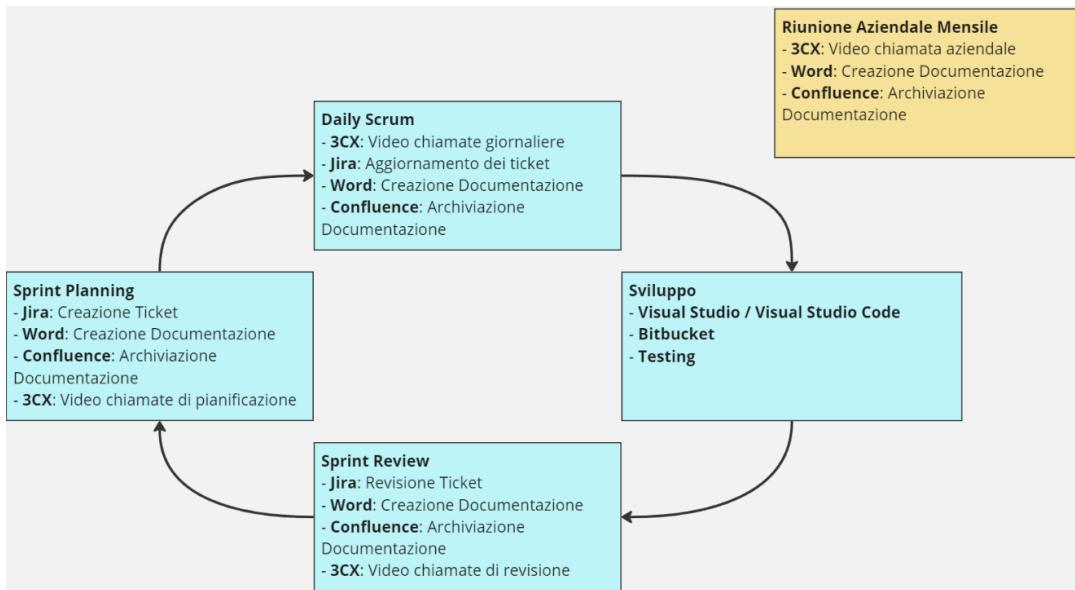
- **Visual Studio Code:** *editor* di codice sorgente particolarmente leggero ed estensibile grazie ad una gamma di estensioni che è possibile integrargli. È inoltre *open source* e compatibile con una vasta gamma di sistemi operativi;
- **SQL Server Management Studio (SSMS):** è un ambiente integrato per la configurazione, la gestione e l'amministrazione di tutti i componenti, le istanze e i *database* all'interno di Microsoft SQL Server. SSMS include sia *editor* di *script* che strumenti grafici che lavorano con oggetti e funzionalità del *server*.
- **Swagger:** un insieme di strumenti e tecnologie per la progettazione, la costruzione e la documentazione delle *API* RESTful, ovvero delle interfacce che permettono a i vari *software* di comunicare tra loro.

L'azienda utilizza una vasta gamma di linguaggi di programmazione, *framework* e librerie per diversi motivi. Alcuni di questi includono l'acquisizione e l'adattamento di codice sorgente da altre aziende, il fatto che il codice sia stato scritto molti anni fa con tecnologie ormai obsolete, e la necessità di utilizzare linguaggi specifici per soddisfare esigenze particolari. Tuttavia, l'azienda si impegna a uniformare quanto più possibile i linguaggi e a ridurre il numero di tecnologie in uso, al fine di semplificare e rendere più efficiente la gestione delle risorse tecnologiche. Oltre a quelle che ho usato per il mio progetto (che sono discusse più approfonditamente nel capitolo 2.4.1) queste sono alcune delle tecnologie che l'azienda usa per lo sviluppo dei suoi prodotti.

- **Angular:** *framework* per lo sviluppo di *web app* basato su TypeScript, sviluppato e mantenuto da Google;
- **Librerie Dev Express:** Dev Express è un'azienda incentrata sulla creazione di librerie di componenti grafici di cui le più famose sono Blazor e MAUI. Queste librerie sono supportate per lo sviluppo in React, Angular e Vue;

- **FoxPro:** un sistema di gestione di *database* e un linguaggio di programmazione procedurale orientato agli oggetti. Originariamente sviluppato da Fox Software e successivamente acquisito da Microsoft.

### 1.4.2 Integrazione delle tecnologie con i processi aziendali



**Figura 1.3:** Integrazione metodologie e tecnologie.

La figura 1.3 mostra come l'applicazione del *framework* Scrum richieda l'impiego di diverse tecnologie:

- **Documentazione:** a seguito dei *meeting* aziendali viene prodotto un documento che può avere diverse finalità: trascrivere gli argomenti dell'incontro, le difficoltà incontrate e le soluzioni da applicare per correggerle o semplici appunti. Gli sviluppatori usano quindi Word per redarre la documentazione data la sua estrema semplicità e la velocità con la quale si possono produrre tali documenti;
- **Archiviazione di documenti:** eccezion fatta per gli appunti personali, la documentazione richiede di essere condivisa e organizzata. Nonostante in VisioneImpresa esista una serie di cartelle di rete accedibili tramite il *file manager* di Windows, questa soluzione risulta inadeguata allo scopo

in quanto povera di funzionalità e personalizzazioni. Confluence permette non solo di organizzare la documentazione in una serie di *directory online* in modo da rendere i documenti sempre consultabili, ma permette anche di definire una serie di privilegi per consentire o meno l'accesso alla documentazione.

- **Comunicazione:** 3CX agisce come canale di comunicazione per la maggior parte delle comunicazioni interne. È il luogo dove si svolgono i *meeting*, fondamentali per mantenere il *team* coordinato, ma permette anche di effettuare chiamate e scrivere in *chat* confinando la maggior parte delle comunicazioni in un unico luogo. Ovviamente anche le *mail* sono un valido strumento di comunicazione, ed infatti viene usato Zimbra per comunicare con persone esterne all'azienda o per alcune *mail* interne (comunicazioni di servizio, *reminder*, ecc.). Quindi 3CX viene usato maggiormente per i *meeting* o per un tipo di comunicazione veloce ed informale, mentre il servizio di posta elettronica viene usato principalmente per comunicazione con persone esterne all'azienda. Ovviamente i clienti che hanno necessità di assistenza possono chiamare l'assistenza tramite il numero di telefono che viene loro fornito per mettersi in contatto con uno dei tecnici del reparto assistenza;
- **Sviluppo:** Per quanto riguarda lo sviluppo gli strumenti utilizzati e le tecnologie impiegate variano a seconda del progetto a cui lo sviluppatore sta lavorando o da preferenze personali (come nel caso del sistema operativo utilizzato nel proprio *computer*). Anche per quanto riguarda l'*editor* viene lasciata libertà di scelta, personalmente per lo sviluppo di *API* in .NET (le tecnologie che ho impiegato per lo sviluppo del progetto sono discusse in un capitolo a parte [2.4.1](#)) ho preferito utilizzare Visual Studio perché offre molti strumenti di supporto e *debug* integrati. Per quanto riguarda lo sviluppo del *front-end* ho preferito utilizzare Visual Studio Code perché più leggero, personalizzabile.
- **Testing:** anche qui gli strumenti utilizzati dipendono dal progetto che si sta sviluppando. Nel mio caso ho utilizzato uno *smartphone* e *tablet*

Android per testare l'applicazione nel suo insieme e la piattaforma Swagger per testare manualmente le *API* (*G*) (le tecnologie che ho impiegato per lo sviluppo del progetto sono discusse in un capitolo a parte [2.4.1](#), mentre per la descrizione accurata di come ho effettuato il testing del codice consulta il capitolo TODO 3.6). Riporto per completezza che per quanto riguarda il *testing* di applicazioni Android è possibile emulare un dispositivo con gli strumenti offerti dallo strumento di sviluppo Android Studio e installare l'applicazione su questo *device* virtuale. Il problema di questo strumento è che richiede *computer* con prestazioni molto alte per funzionare in maniera fluida altrimenti rischia di paralizzare l'elaboratore. Non sono sicuro che sia utilizzato dagli sviluppatori di VisioneImpresa quindi ho evitato di includerlo nelle tecnologie utilizzate;

- **Collaborazione e gestione del codice:** Bitbucket viene utilizzato come piattaforma per depositare il codice sorgente e gestire lo sviluppo collaborativo. Qui gli sviluppatori caricano il loro codice suddiviso in *repository* per ogni progetto. Ogni *repository* vede diversi *branch* attivi: *main* che contiene l'ultima versione rilasciata al pubblico del *software*, *develop* ovvero il *branch* di lavoro dove nascono e confluiscono tutti i *feature branch* prima di effettuare il rilascio in *main*, i *feature branch* che viene creato dal programmatore per sviluppare una specifica funzione del programma che sarà, una volta terminata e testata, aggiunta in *develop*;
- **Gestione dei progetti:** Jira è una piattaforma particolarmente utile per pianificare i vari compiti da svolgere nello *sprint* e assegnarli ai vari componenti del *team* di sviluppatori. Questi compiti sono chiamati *ticket* e possono essere di diverso tipo: *epic* che rappresentano grosse porzioni di lavoro e sono quindi usate come raccolte di *ticket*, *task* il singolo compito che deve essere completato e *bug* che rappresenta una problematica da risolvere. I *bug* possono essere avere diverse origini: gli sviluppatori stessi nel caso in cui si accorgano di un difetto di programmazione o da i clienti che telefonando all'assistenza riportano il problema, quindi il tecnico riporterà la problematica al *project manager* che creerà la *task*. Jira offre

inoltre molti altri strumenti per la gestione di metodologie Agile come la possibilità di creare diagrammi di Gantt, che permettono di avere una rappresentazione visiva delle attività programmate nel tempo, aiutando il *team* a comprendere meglio la sequenza delle attività, o la definizione di un *backlog*, ovvero una lista con le *task* rimaste incompiute durante gli *sprint* precedenti;

## 1.5 Propensione all’innovazione

VisioneImpresa non dispone di un ufficio specificamente dedicato alla ricerca e sviluppo, ma questo non significa che non vengano effettuati aggiornamenti costanti delle tecnologie e degli strumenti utilizzati. Ad esempio, l’azienda ha in programma di migrare i propri sistemi *ERP<sub>G</sub>*, attualmente scritti in FoxPro (un linguaggio il cui supporto da parte di Microsoft è terminato nel 2015), verso tecnologie più moderne. Questo progetto, data la grandezza e complessità dei *software* coinvolti, richiederà anni per essere completato.

Inoltre, l’attività di *stage* rappresenta un’opportunità per l’azienda di innovare. Durante il mio tirocinio, ho osservato un apprezzamento particolare per l’indipendenza degli stagisti nel cercare e implementare soluzioni o tecnologie originali. Per ulteriori dettagli sul rapporto dell’azienda con gli *stage*, consulta il capitolo [2.1](#).

# Capitolo 2

## Descrizione e pianificazione del progetto di stage

### 2.1 Strategia aziendale e rapporto con gli *stage*

Come ho descritto nel capitolo 1.5, gli *stage* rappresentano per VisioneImpresa un'opportunità di innovazione e crescita.

Gli stagisti, prossimi alla conclusione del corso triennale di laurea in informatica, possiedono una buona conoscenza di quali sono le nuove tecnologie e hanno la curiosità di studiarle e implementarle.

L'azienda decide quindi di sfruttare l'occasione offerta dai tirocini per innovare e rinnovare i propri prodotti.

I progetti assegnati agli stagisti devono essere formativi, richiedendo uno studio approfondito del codice e delle tecnologie. Questi compiti non devono essere banali e devono mirare a stimolare la crescita e l'apprendimento dello studente. Inoltre, devono basarsi su necessità reali, identificate mediante colloqui con i clienti, e sono pertanto di interesse per VisioneImpresa nell'ottica di una futura implementazione dei progetti nei prodotti reali.

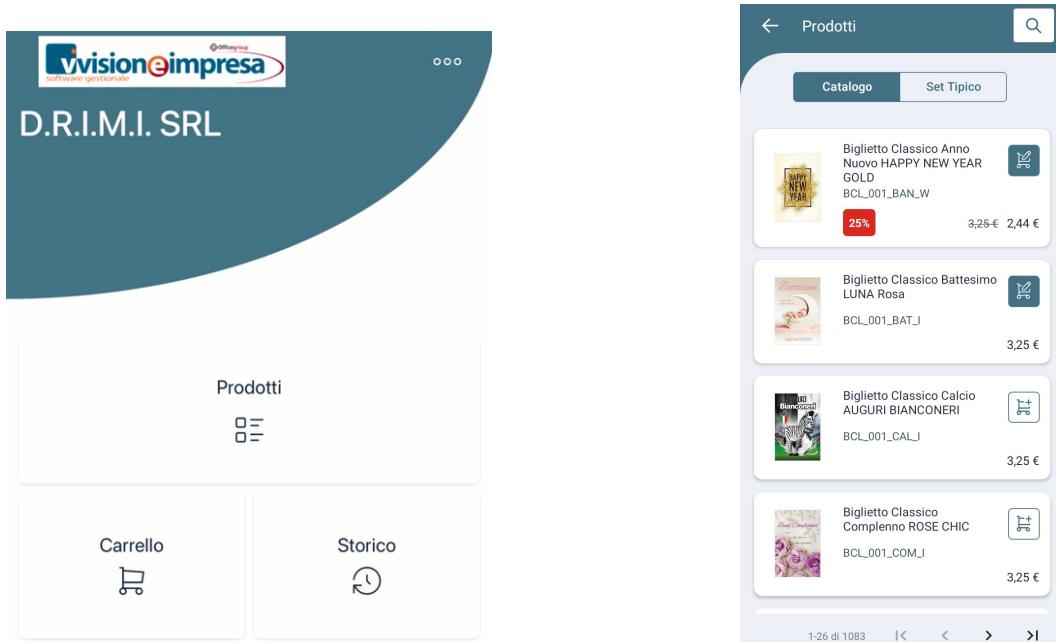
Al termine del lavoro, lo studente presenta il suo operato agli sviluppatori e al *project manager* del prodotto in questione. Questo *meeting* è cruciale per valutare la qualità del lavoro svolto e comprendere la logica di sviluppo adottata. Permette inoltre agli sviluppatori e al *project manager* di esaminare il progetto

in funzione e valutare i benefici di una sua futura implementazione.

Successivamente, si richiede allo stagista di consegnare il codice sorgente e la documentazione tecnica, che serviranno da guida per l'eventuale implementazione futura.

Gli *stage* per VisioneImpresa sono inoltre un modo per mettersi in contatto con studenti che si sono distinti in azienda e che hanno intenzione di interrompere il loro percorso universitario dopo la laurea, in modo da poter proporre loro un colloquio dove poterne valutare l'assunzione.

## 2.2 Descrizione progetto



(a) Homepage della versione *tablet* di MoviORDER

(b) Pagina Prodotti della versione *smartphone* di MoviORDER

**Figura 2.1:** Alcune view di MoviORDER

MoviORDER è un'applicazione fornita dalle aziende ai loro clienti per gestire gli ordini ed effettuare il *restock* della merce. L'*app* permette inoltre di: visualizzare lo storico degli ordini precedenti, accedere a un *set* tipico (ovvero un ordine predefinito con i prodotti solitamente acquistati), visualizzare il carrello per il riepilogo dell'ordine e uno storico dei documenti generati dopo la conferma

dell'ordine.

La sua Homepage è visibile in figura 2.1a, mentre il catalogo prodotti è mostrato in figura 2.1b.

Il progetto assegnatomi da VisioneImpresa richiede di sviluppare un modulo per MoviORDER chiamato "Modulo Agenti", ovvero l'insieme di interfacce, funzioni, *API<sub>G</sub>*, ecc. che permettono l'autenticazione di un nuovo tipo di utente, gli agenti aziendali, i quali, selezionando da una lista uno dei clienti a loro assegnati, possono operare nell'*app* come il cliente selezionato.

La necessità di implementare questo modulo nasce da alcuni bisogni segnalati dai suoi clienti a VisioneImpresa e che mi sono stati descritti durante il primo incontro in azienda con il *tutor* aziendale. I principali motivi sono:

- **MoviSELL, l'*app* pensata per gli agenti aziendali, è disponibile solo per *tablet* iOS.** Questo può costituire un problema per alcune aziende che, per dotare i propri agenti dell'*app*, sono obbligate ad acquistare questi *tablet* per i propri agenti. Per aziende con agenti plurimandatari, cioè che rappresentano più aziende contemporaneamente, questo requisito si rivela essere particolarmente oneroso da soddisfare, dato che si richiede di fornire i *tablet* non ai propri dipendenti, ma a professionisti esterni;
- **Non tutti gli agenti si trovano a loro agio ad usare il *tablet*,** trovandolo ingombrante e scomodo, soprattutto per chi lavora molto in mobilità. Pertanto, avere un'alternativa per smartphone risulta preferibile.
- MoviSELL è un'*app* ricca di funzionalità, ma può risultare di difficile utilizzo per chi non ha dimestichezza con gli strumenti digitali. **MoviORDER risulta molto più semplice e intuitiva, rimuovendo la barriera tecnologica per alcuni agenti** e permettendo loro di svolgere il loro lavoro;
- Alcuni clienti preferiscono contattare direttamente gli agenti per effettuare i loro ordini, invece di usare MoviORDER. **Il modulo agenti semplifica l'operazione di creazione dell'ordine per gli agenti**, evitando loro di

dover appuntare la merce da ordinare e poi effettuare l'ordine dal *computer* una volta rientrati in ufficio.

## 2.3 Scelta dell'attività di *stage*

L'incontro con VisioneImpresa è avvenuto durante l'evento StageIT 2024, organizzato dall'Università di Padova.

Questo evento offre alle aziende l'opportunità di incontrare gli studenti e condurre brevi colloqui, illustrando le caratteristiche dell'azienda e i progetti offerti per lo *stage*.

In questa occasione, ho avuto l'opportunità di esplorare diverse realtà operanti in settori distinti, dallo sviluppo *web* a progetti in ambito *cyber security*. Non mi sono limitato a cercare progetti allineati alle mie conoscenze pregresse, ma ho esplorato diverse opzioni.

Durante l'incontro con VisioneImpresa, ho approfondito la conoscenza dell'azienda e mi sono state fornite ulteriori informazioni riguardo i loro progetti di *stage*, descritti in un elenco diffuso prima dell'evento per tutte le aziende coinvolte.

Successivamente ho selezionato i progetti più interessanti e fissato ulteriori colloqui con le aziende per discutere in maniera più approfondita delle proposte. I colloqui si sono focalizzati principalmente su: il progetto in dettaglio, le tecnologie utilizzate, l'azienda e una discussione ad alto livello sulle possibili implementazioni del progetto.

Durante l'incontro con VisioneImpresa ho inoltre potuto fare un *tour* dell'azienda, che mi ha permesso di conoscere i dipendenti e osservare il loro ambiente di lavoro e le interazioni con i clienti.

La scelta di lavorare al progetto "Modulo agenti" è stata motivata da diversi fattori: innanzitutto, mi ha consentito di lavorare ad un'applicazione *Android*, ambito di grande interesse personale, utilizzando tecnologie moderne e ricercate come React Native. Questa tecnologia non mi è del tutto estranea, grazie all'esperienza pregressa con React, sebbene non includa le funzionalità specifiche per applicazioni *mobile*. Il progetto ha previsto anche l'utilizzo di ASP.NET

Core per lo sviluppo di *API<sub>G</sub>*.

In secondo luogo, l'opportunità offerta da VisioneImpresa mi ha permesso di lavorare sia come sviluppatore *front-end*, creando le interfacce per l'*app*, che come sviluppatore *back-end*, creando e modificando le *API<sub>G</sub>* dell'applicazione.

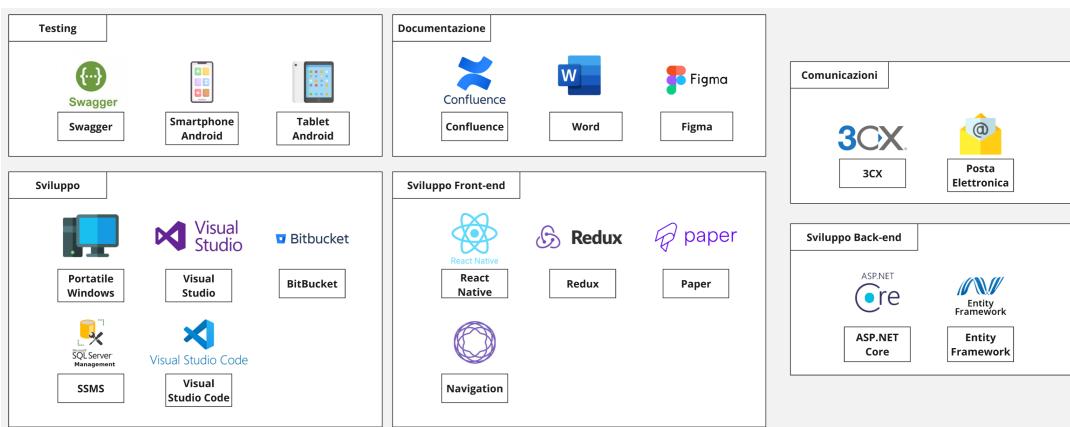
## 2.4 Vincoli

### 2.4.1 Vincoli tecnologici

VisioneImpresa non impone vincoli specifici sulle tecnologie da utilizzare. Tuttavia, essendo lo scopo del progetto lo sviluppo di un modulo per un'applicazione esistente, emergono vincoli tecnologici impliciti. È necessario utilizzare gli stessi *framework* già impiegati per lo sviluppo, per poter integrare il modulo nell'*app* ed eventualmente riportarlo nel *software* commercializzato.

Non vi sono divieti circa l'introduzione di nuove tecnologie. Come riporto nel capitolo 2.1, uno degli obiettivi aziendali per il progetto di *stage* è valutare il beneficio di nuove librerie, *framework* e tecnologie per favorire innovazione e crescita.

Anche per gli strumenti da utilizzare non ho ricevuto vincoli esplicativi, eccetto l'uso di BitBucket come piattaforma per conservare il codice.



**Figura 2.2:** Tecnologie utilizzate per lo sviluppo di MoviORDER

Le tecnologie che ho utilizzato per sviluppare il modulo agenti, come mostra la figura 2.2, sono:

- **Visual Studio:** per lo sviluppo delle *API<sub>G</sub>*. Offre numerosi strumenti per il *debug* del codice e facilita la gestione dei pacchetti necessari. È particolarmente efficace per lo sviluppo in C#, grazie alle funzioni di auto completamento e all'aggiornamento automatico degli *import*;
- **Visual Studio Code:** per lo sviluppo del *front-end*, grazie alle numerose estensioni disponibili per lo sviluppo in React (come Prettier, che assicura una formattazione uniforme del codice);
- **Computer con Windows 10:** preferito al Mac per la mia esperienza pregressa con Windows, evitando così l'apprendimento di un nuovo sistema operativo parallelamente alle nuove tecnologie necessarie;
- **Smartphone e tablet Android:** per il *testing* dell'applicazione;
- **BitBucket:** per conservare il codice all'interno di un *branch* del progetto e usufruire delle funzionalità di Git;
- **Confluence:** per conservare la documentazione creata;
- **Word:** per creare la documentazione tecnica, poiché consente di creare rapidamente e agevolmente un documento facilmente consultabile e modificabile;
- **Figma:** per creare il manuale utente. Permette una gestione dello stile più precisa e personalizzabile rispetto a Word, ma richiede più tempo per la creazione di un documento. Il risultato è un impatto grafico migliore, meno rilevante per un documento tecnico, ma apprezzabile per un documento destinato all'utente finale;
- **Servizio di posta elettronica:** per ricevere gli annunci aziendali;
- **SQL Server Management Studio (SSMS):** per operare sul *database* usato per lo sviluppo dell' applicazione;
- **Swagger:** per testare manualmente le *API<sub>G</sub>* e valutarne il corretto funzionamento.

## CAPITOLO 2. DESCRIZIONE E PIANIFICAZIONE DEL PROGETTO DI STAGE

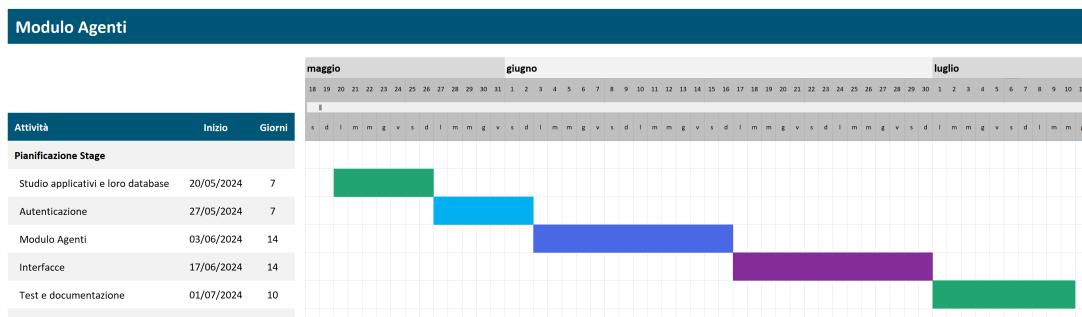
---

Per completezza, menziono anche 3CX tra le tecnologie che mi sono state fornite. Tuttavia, non ho mai avuto necessità di utilizzarlo, avendo avuto la possibilità di confrontarmi personalmente con gli sviluppatori e il *tutor* aziendale. Inoltre, durante il mio *stage*, il *meeting* mensile generale è stato annullato, impedendomi di partecipare alla video chiamata.

I *framework* e le librerie utilizzate sono:

- **React Native:** *framework* che consente lo sviluppo di applicazioni Android e iOS utilizzando il *framework* React. Include funzionalità specifiche per dispositivi *mobile*, distinguendosi da React, principalmente utilizzato per lo sviluppo di siti *web*.  
Permette di scrivere e mantenere un unico codice funzionante per entrambi i sistemi operativi, eliminando la necessità di sviluppatori specializzati separatamente nello sviluppo Android e iOS;
- **React Native Paper:** libreria di componenti per le interfacce;
- **React Native Navigation:** libreria per la gestione della navigazione tra le *view*;
- **React Native Redux:** libreria per la gestione dello stato dell'applicazione in React Native;
- **ASP.NET Core:** *framework open source* moderno per lo sviluppo di applicazioni connesse a *internet*, utilizzato in questo caso per lo sviluppo delle *API G* necessarie;
- **Entity Framework Core:** *ORM G* (*Object-Relational Mapping*) per .NET, un *mapper* che semplifica l'accesso e la gestione dei dati nel *database*, permettendo di lavorare con oggetti .NET invece di *query SQL*.

## 2.4.2 Vincoli temporali



**Figura 2.3:** Pianificazione attività di *stage*

La figura 2.3 illustra la pianificazione delle mie attività di *stage*, come riportata nel piano di lavoro concordato con il *tutor* aziendale e approvato dal relatore. Lo stage si divide in cinque attività principali:

1. **Studio degli applicativi e del *database*** (una settimana): si concentra sull'apprendimento delle tecnologie utilizzate e sull'analisi del codice sorgente;
2. **Modifica della *API<sub>G</sub>* di autenticazione** (una settimana): l'obiettivo è modificare la *API<sub>G</sub>* di *login* per il nuovo tipo di utente agente;
3. **Sviluppo del modulo agenti** (due settimane): prevede la creazione delle *API<sub>G</sub>* necessarie e la modifica del *front-end*;
4. **Modifica delle interfacce** (due settimane): prevede la creazione dei componenti, definizione dello stile della UI (*User Interface*) e l'ottimizzazione per *tablet*;
5. **Documentazione e *testing*** (dieci giorni): prevede la creazione della documentazione tecnica , operativa e il *testing* dell'applicazione.

## 2.5 Obiettivi

### 2.5.1 Obiettivi aziendali

La tabella 2.1 illustra gli obiettivi del progetto di *stage* richiesti dall'azienda. Questi mirano a produrre un prodotto usabile e funzionale, costituendo una solida base di studio per l'implementazione reale del modulo. Gli obiettivi sono categorizzati in obbligatori, contrassegnati dalla lettera M (*mandatory*), e opzionali, contrassegnati dalla lettera O (*optional*).

Tipo	Obiettivo
M	Modifica del sistema di autenticazione
M	Aggiornamento delle interfacce per adeguamento al nuovo modulo
M	Sviluppo del modulo agenti
M	Stesura documentazione tecnica e operativa
M	<i>Testing</i> delle nuove funzionalità
O	Ottimizzazione per <i>tablet</i>

**Tabella 2.1:** Tabella Obiettivi

### 2.5.2 Obiettivi personali

Per quanto riguarda gli obiettivi personali che ho stabilito per questo *stage*, sono elencati dalla tabella 2.2, nello stesso modo descritto per la tabella precedente.

Tipo	Obiettivo
M	Raggiungere una buona conoscenza delle principali tecnologie utilizzate (React Native e .NET)
	Continua nella prossima pagina...

**Tabella 2.2 – Continuo della tabella**

<b>Tipo</b>	<b>Obiettivo</b>
M	Analizzare e comprendere l'architettura <i>software</i> di MoviORDER
M	Apprendere ed implementare le metodologie di sviluppo aziendali
M	Migliorare nel <i>problem solving</i> trovando soluzioni efficienti ai problemi incontrati durante lo sviluppo
M	Collaborare con il <i>team</i> di VisioneImpresa per la realizzazione del progetto

**Tabella 2.2:** Tabella Obiettivi Personalni

# Capitolo 3

## Sviluppo del modulo agenti

### 3.1 Analisi

L’analisi del progetto è un’attività cruciale nello sviluppo di qualsiasi sistema *software*. Essa fornisce le fondamenta su cui si basa l’intero processo di sviluppo, definendo chiaramente i requisiti e le aspettative del sistema. Questa fase non è un processo statico, ma un’attività dinamica che evolve e si raffina continuamente durante l’intero ciclo di vita del progetto.

È fondamentale sottolineare che l’analisi non si conclude con l’inizio dell’attività di codifica. Al contrario, è un processo iterativo che viene costantemente raffinato man mano che emergono nuove informazioni o si incontrano sfide impreviste durante lo sviluppo. Questo approccio permette di adattarsi alle mutevoli esigenze del progetto e garantisce che il prodotto finale soddisfi le aspettative del cliente.

#### 3.1.1 Casi d’uso

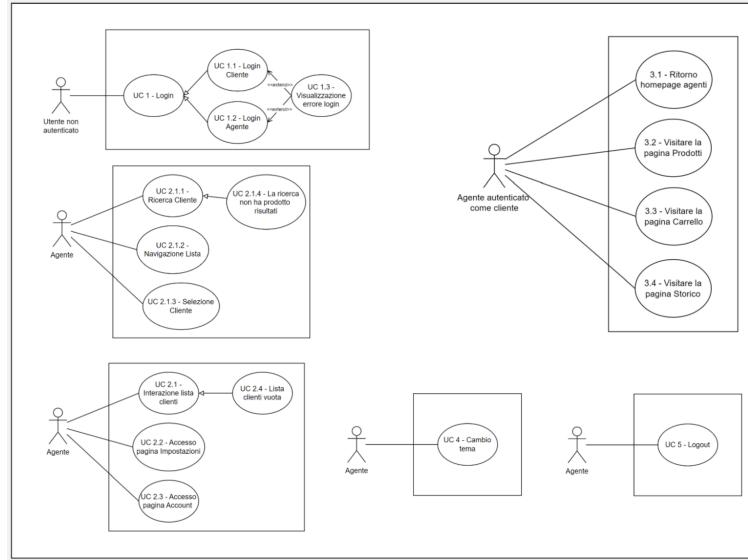
Un’attività fondamentale durante l’analisi del progetto è la definizione dei casi d’uso. Essi descrivono le interazioni tra gli utenti (o altri sistemi esterni) e il sistema in esame, illustrando come questo debba comportarsi per soddisfare le esigenze degli *stakeholder*.

Per definire i casi d’uso, ho condotto un’attenta lettura del capitolo e svolto

## CAPITOLO 3. SVILUPPO DEL MODULO AGENTI

---

molteplici interviste con il *tutor* aziendale. Il risultato di questo processo è una serie di casi d'uso, illustrati dalla figura 3.1.



**Figura 3.1:** Use Cases modellati per il progetto

Questa attività di modellazione mi ha permesso di comprendere meglio le funzionalità richieste per il modulo agenti e di individuare necessità implicite.

Nella modellazione dei casi d'uso vengono riportate diverse informazioni, come le condizioni richieste prima e dopo il caso d'uso, una descrizione, scenario e attori coinvolti.

Per comprendere i requisiti che ho riportato nel prossimo capitolo, riporto una descrizione dei tre attori principali, ovvero le entità (umani o altri sistemi) che interagiscono il sistema:

- **Utente non autenticato:** utente che non ha completato la procedura di *login*, può essere un cliente o un agente;
- **Agente:** utente autenticato e riconosciuto dal sistema come agente aziendale;
- **Agente autenticato come cliente:** agente che ha selezionato un cliente e vuole operare all'interno dell'*app* come il cliente selezionato.

### 3.1.2 Requisiti funzionali e non funzionali

L'analisi dei casi d'uso permette l'identificazione e la definizione dei requisiti funzionali e non funzionali del progetto.

I requisiti funzionali descrivono le funzionalità specifiche che il sistema deve offrire, dettagliando il comportamento atteso in risposta alle diverse interazioni degli utenti. I requisiti non funzionali, d'altra parte, definiscono le caratteristiche qualitative del sistema, come prestazioni, sicurezza, usabilità e scalabilità. Sebbene non sempre esplicitamente evidenti nei casi d'uso, questi requisiti sono spesso impliciti nelle aspettative degli utenti. In alcuni casi le necessità che portano a descrivere questi requisiti non sono proprio modellabili come casi d'uso (come la necessità di fornire la documentazione sul progetto), pertanto la definizione dei requisiti si rivela essere un lavoro complesso quanto essenziale.

Il numero di requisiti soddisfatti rispetto a quelli concordati permette di avere una misura della qualità del lavoro svolto.

Ho assegnato ad ogni requisito un codice identificativo, che riporta:

- **F, V, Q, U:** F rappresenta i requisiti funzionali, V i requisiti non funzionali di vincolo, Q i requisiti non funzionali qualitativi e U i requisiti non funzionali di usabilità;
- **O, D:** O per i requisiti obbligatori, D per quelli desiderabili;
- **id:** numero identificativo del requisito;

Per ogni requisito viene fornita anche una descrizione e la fonte da cui è tratto. Le fonti che ho riportato nella tabella indicano da dove ho ricavato il requisito descritto, troviamo:

- **UC:** il requisito descritto è stato ricavato da un caso d'uso;
- **Capitolato:** il requisito descritto è stato ricavato dal capitolato del progetto;
- **Tutor:** il requisito descritto è stato ricavato da colloqui avuti con il *tutor* aziendale durante il tirocinio;

- **Studente:** il requisito descritto è stato ricavato da personali considerazioni sul progetto.

La tabella 3.1 riporta l'elenco dei requisiti che ho individuato e che ho concordato con il *tutor*:

Requisito	Descrizione	Fonte
FO1	Un utente deve poter effettuare il <i>login</i> ed essere automaticamente riconosciuto come cliente	UC, Capitolo
FO1.1	Un cliente deve essere spostato nella Homepage in seguito al <i>login</i>	lato
FO1.2	Un cliente non deve poter notare nessuna differenza rispetto alla precedente versione dell' <i>app</i>	Capitolato
FO2	Un utente deve poter effettuare il <i>login</i> ed essere automaticamente riconosciuto come agente	UC, Capitolo
FO2.1	Un agente deve essere spostato nella Homepage Agenti in seguito al <i>login</i>	UC, Studente
FO3	Un utente deve visualizzare un messaggio d'errore se le credenziali sono errate	UC, Studente
FO4	Un agente deve poter visualizzare una lista con i suoi clienti	Capitolato
FD4.1	Un agente deve poter ricercare un cliente all'interno della lista dei suoi clienti	UC, Tutor
FO4.2	Un agente deve poter navigare la lista dei suoi clienti	UC, Capitolo
FO4.3	Un agente deve poter selezionare dalla lista uno dei suoi clienti	lato

Continua nella prossima pagina...

**Tabella 3.1 – Continuo della tabella**

<b>Requisito</b>	<b>Descrizione</b>	<b>Fonte</b>
FO4.4	Un agente deve visualizzare un messaggio d'errore che riporta la frase "nessun cliente trovato" se la lista clienti è vuota	UC, <i>Tutor</i>
FD4.5	Un agente deve visualizzare un messaggio d'errore che riporta la frase "nessun cliente trovato" se la ricerca clienti non ha prodotto risultati	UC, <i>Tutor</i>
FO4.6	Un agente deve essere spostato nella pagina <b>Homepage</b> dopo aver selezionato un cliente dalla lista	UC, Studente
FO4.7	Un agente deve essere autenticato come il cliente selezionato dopo aver selezionato un cliente dalla lista	UC, Capitolo
FD5	Un agente deve poter visualizzare un menu nella <b>Homepage Agenti</b>	Studente
FD5.1	Un agente deve poter premere il pulsante "Impostazioni" dal menu nella <b>Homepage Agenti</b>	UC, Studente
FD5.2	Un agente deve essere spostato nella pagina <b>Impostazioni</b> dopo aver premuto il pulsante "Impostazioni"	UC, Studente
FD5.3	Un agente deve poter premere il pulsante "Account" dal menu nella <b>Homepage Agenti</b>	UC, Studente
FD5.4	Un agente deve essere spostato nella pagina <b>Account</b> dopo aver premuto il pulsante "Account"	UC, Studente
Continua nella prossima pagina...		

**Tabella 3.1 – Continuo della tabella**

<b>Requisito</b>	<b>Descrizione</b>	<b>Fonte</b>
FO6	Un agente autenticato come cliente deve poter visualizzare un menu nella <b>Homepage</b>	Studente
FD6.1	Un agente autenticato come cliente deve poter ritornare alla <b>Homepage Agenti</b> premendo il pulsante " <i>Homepage Agenti</i> " nel menu della <b>Homepage</b>	UC, Studente
FD6.2	Un agente autenticato come cliente deve essere spostato nella pagina <b>Prodotti</b> premendo il pulsante " <b>Prodotti</b> " nel menu della <b>Homepage</b>	UC, Capitolo
FD6.2.1	Un agente autenticato come cliente deve poter operare come il cliente selezionato nella pagina <b>Prodotti</b>	UC, Capitolo
FD6.3	Un agente autenticato come cliente deve essere spostato nella pagina <b>Carrello</b> premendo il pulsante " <b>Carrello</b> " nel menu della <b>Homepage</b>	UC, Capitolo
FD6.3.1	Un agente autenticato come cliente deve poter operare come il cliente selezionato nella pagina <b>Carrello</b>	UC, Capitolo
FD6.4	Un agente autenticato come cliente deve essere spostato nella pagina <b>Storico</b> premendo il pulsante " <b>Storico</b> " nel menu della <b>Homepage</b>	UC, Capitolo
FD6.4.1	Un agente autenticato come cliente deve poter operare come il cliente selezionato nella pagina <b>Storico</b>	UC, Capitolo
Continua nella prossima pagina...		

**Tabella 3.1 – Continuo della tabella**

<b>Requisito</b>	<b>Descrizione</b>	<b>Fonte</b>
FD7.1	Un agente deve poter modificare il tema impostando il tema chiaro dalla pagina <b>Impostazioni</b>	UC, <i>Tutor</i>
FD7.2	Un agente deve poter modificare il tema impostando il tema scuro dalla pagina <b>Impostazioni</b>	UC, <i>Tutor</i>
FO7.3	Un agente non deve poter modificare la propria schermata d'avvio dalla pagina <b>Impostazioni</b>	Studente
FD8	Un agente deve poter effettuare il <i>logout</i> dalla pagina <b>Account</b>	UC, <i>Tutor</i>
QO1	Insieme al modulo deve essere consegnato anche un manuale tecnico	Capitolato, <i>Tutor</i>
QO2	Insieme al modulo deve essere consegnato anche un manuale utente	Capitolato, <i>Tutor</i>
UO1	Un agente deve visualizzare il suo nome all'interno della schermata <b>Homepage Agenti</b>	Capitolato, Studente
UO2	Un agente deve visualizzare il nome del cliente selezionato all'interno della schermata <b>Homepage</b>	Capitolato, Studente
UO3	Ogni voce della lista deve riportare alcune informazioni del cliente	<i>Tutor</i>
UO3.1	Ogni voce della lista deve riportare il nome del cliente	<i>Tutor</i>
UO3.2	Ogni voce della lista deve riportare l'indirizzo del cliente	<i>Tutor</i>
UD4	L'applicazione deve essere disponibile nella lingua italiana	<i>Tutor</i>
Continua nella prossima pagina...		

**Tabella 3.1 – Continuo della tabella**

Requisito	Descrizione	Fonte
UD5	L'applicazione deve essere disponibile nella lingua inglese	<i>Tutor</i>
UD6	Deve essere possibile ricercare un cliente nelle lista attraverso l'uso di una <i>search bar</i>	<i>Tutor</i>
UD7	Durante la digitazione del parametro di ricerca i risultati devono essere filtrati anche per i parametri parziali	<i>Tutor</i>
UD8	Durante la ricerca i clienti devono essere filtrati secondo il parametro digitato dall'agente	<i>Tutor</i>
UD9	Lo stile dell'applicazione deve essere compatibile con dispositivi <i>tablet</i>	Capitolato, <i>Tutor</i>
VO1	Il modulo deve utilizzare i <i>databases</i> di MoViORDER apportando eventuali modifiche alla struttura	Capitolato, <i>Tutor</i>
VO2	Il modulo deve estendere le <i>API G</i> esistenti sviluppate in .NET	Capitolato, <i>Tutor</i>
VO3	Il modulo deve estendere le interfacce esistenti sviluppate in React Native	Capitolato, <i>Tutor</i>

**Tabella 3.1:** Tabella del tracciamento dei requisiti funzionali e non funzionali.

Di seguito riporto una tabella che riepiloga i requisiti precedentemente descritti.

Tipologia	Obbligatorio	Desiderabile	Totale
<b>Funzionale</b>	13	17	30
<b>Di vincolo</b>	3	0	3
<b>Qualitativi</b>	2	0	2
Continua nella prossima pagina...			

**Tabella 3.2 – Continuo della tabella**

Tipologia	Obbligatorio	Desiderabile	Totale
Usabilità	5	6	11
<b>Totale</b>	<b>23</b>	<b>23</b>	<b>46</b>

**Tabella 3.2:** Riepilogo requisti

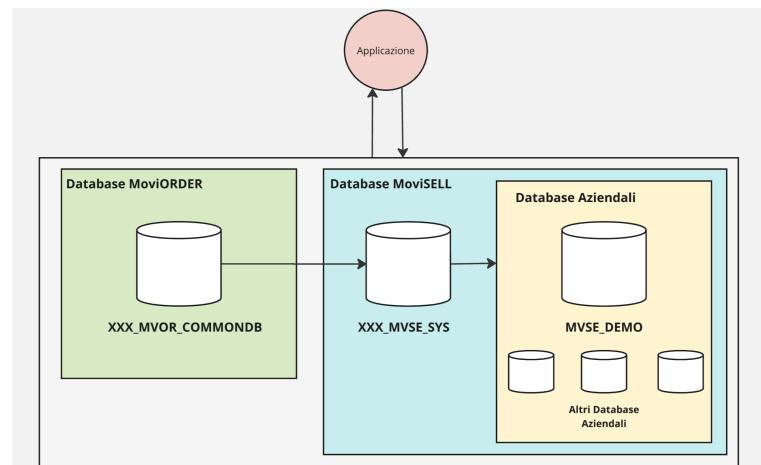
## 3.2 Progettazione

La progettazione di un sistema *software* è un’attività che traduce i requisiti in una struttura concreta per l’implementazione. Essa coinvolge la definizione dell’architettura, la definizione di *design pattern* e la pianificazione delle interazioni tra le componenti del sistema.

Il progetto che mi è stato proposto da VisioneImpresa ha come obiettivo l’espansione del sistema esistente e non si è reso necessario espandere o modificare l’architettura esistente.

In questo capitolo descrivo l’architettura di MoviORDER che ho studiato al fine di ampliare le funzionalità del sistema.

### 3.2.1 Struttura del *database*

**Figura 3.2:** Database di MoviORDER.

Come illustrato dalla figura 3.2, l’infrastruttura dati dell’applicazione MoviORDER si articola su tre *database* distinti: XXX\_MVOR\_COMMONDB, e i *database* condivisi con l’applicazione MoviSELL, ovvero XXX\_MVSE\_SYS e MVSE\_DEMO.

XXX\_MVOR\_COMMONDB (che per brevità chiameremo *common database*), contiene i dati relativi agli utenti di MoviORDER. Esso conserva informazioni necessarie all’applicazione come: credenziali di accesso, stato di validità della licenza, configurazioni dell’interfaccia utente, dettagli del dispositivo, indirizzi *email* e altri parametri necessari al corretto funzionamento del *software*. Questi dati sono fondamentali per i processi di autenticazione e inizializzazione dell’applicazione. Ogni azienda ha il proprio *database* dedicato, che chiameremo *company database*, in cui vengono conservati tutti i dati a lei inerenti come: anagrafica clienti, catalogo prodotti, politiche di sconto personalizzate, profili degli agenti aziendali ecc. Per ogni *company database* viene assegnato un nome del tipo MVSE\_[nome azienda], nel mio caso per lavorare in locale mi è stato fornito un *database* di prova chiamato MVSE\_DEMO.

Quindi abbiamo XXX\_MVSE\_SYS (che per brevità chiameremo *system database*), il cui scopo è quello di fare da *router*, infatti il *common database* contiene un campo nella tabella User chiamato CompanyCode che agisce da chiave esterna assegnando ad ogni utente di MoviORDER un *company database* di riferimento. Questo CompanyCode viene utilizzato dalle API\_G come chiave per il *system database* per ottenere la stringa di connessione al *company database* durante la fase di autenticazione dell’utente.

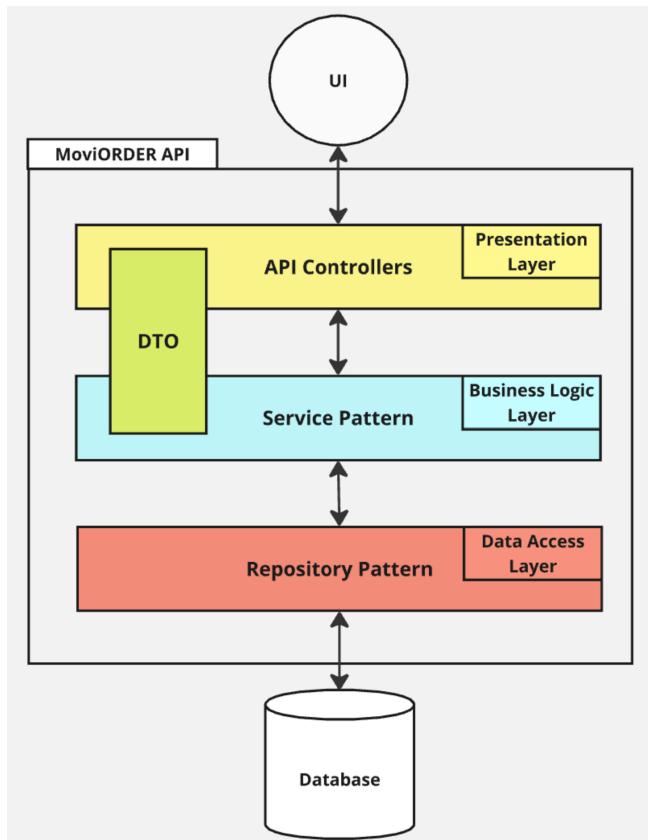
Questa architettura garantisce una gestione efficiente e scalabile dei dati, consentendo una separazione netta tra informazioni riguardanti l’applicazione e i dati specifici delle aziende, oltre a fornire un meccanismo flessibile per l’indirizzamento delle connessioni ai *database* aziendali.

### 3.2.2 Architettura delle API

Ho continuato quindi il mio studio passando all’analisi dell’architettura delle API\_G, che come ho accennato nel capitolo 2.4.1 sono sviluppate utilizzando il framework ASP.NET Core. Il *back-end* di MoviORDER si trova in una *reposit-*

*tory* BitBucket a sé stante rispetto al *front-end* per consentire una gestione più efficiente e modulare del progetto. Questo permette:

- **Manutenibilità:** facilita la manutenzione e l'aggiornamento di ciascuna componente senza impattare l'altra;
- **Flessibilità:** permette l'utilizzo di tecnologie e *framework* diversi per *back-end* e *front-end*, ottimizzando ciascuno per il proprio scopo e facilitando eventuali cambiamenti di tecnologie futuri;
- **Versionamento:** consente una versionamento separato per le due parti dell'applicazione e semplifica il tracciamento delle modifiche del codice;
- **Riutilizzo del codice:** Facilita il riutilizzo del *back-end* per diverse interfacce o applicazioni.



**Figura 3.3:** Architettura API.

Le *API<sub>G</sub>* di MoviORDER adottano un'architettura a livelli, combinando il *Pattern Repository* e il *Pattern Service*, come mostra la figura 3.3, garantendo

scalabilità, manutenibilità e sicurezza. Questa architettura stratifica il codice in livelli di astrazione crescente, partendo dallo strato più "concreto" in diretta interazione con il *database*, fino a quello più "astratto" che si interfaccia con il *front-end*.

### 3.2.2.1 *Data Access layer e Repository Pattern*

Il *Data Access layer* di MoviORDER è implementato seguendo il *Repository Pattern*, un modello di progettazione che separa la logica di accesso ai dati dal resto dell'applicazione. Questo *pattern* crea un'astrazione tra il livello di accesso ai dati e la logica di *business*, consentendo una gestione più flessibile e manutenibile dei dati.

Il *Repository Pattern* è implementato attraverso le classi:

- **ReadOnlyRepository<TContext>**: Questa classe astratta fornisce metodi per operazioni di sola lettura sul database.
- **Repository<TContext>**: Estende `ReadOnlyRepository` aggiungendo metodi per operazioni di scrittura.

In questo caso il *template* `TContext` delle classi `Repository` e `ReadOnlyRepository` rappresenta `DbContext`, una classe fondamentale in Entity Framework che rappresenta una sessione con il *database*. Essa permette di:

- **Eseguire query sul database;**
- **Tracciare le modifiche apportate alle entità;**
- **Persistere i cambiamenti nel database.**

Qui vengono definiti anche i modelli e le *migration* generate tramite Entity Framework, un *mapper* ad alto livello integrabile a .NET che permette la trasposizione delle tabelle del *database* in classi del dominio applicativo.

Entity Framework, originariamente parte del *framework* .NET ma ora distribuito come pacchetto indipendente installabile attraverso l'*installer* di Visual Studio, promuove un approccio di sviluppo *code first*. Entity Framework permette infatti di gestire il *database* attraverso i modelli, ovvero delle particolari

classi che descrivono la forma delle tabelle e i loro attributi.

Creando o modificando questi modelli è possibile creare o modificare la tabella della base dati senza la necessità di interventi manuali diretti, ma attraverso le *migration* generate dal *framework* automaticamente. Quando si apportano modifiche al modello, Entity Framework compara le *migration* esistenti, determinando così lo stato attuale del *database*. Questo processo permette di identificare precisamente le modifiche necessarie alla struttura del *database*. Se il processo di analisi e generazione va a buon fine, Entity Framework crea una nuova *migration* che riporta tutte le modifiche applicate. Questo meccanismo assicura una gestione coerente e tracciabile dell’evoluzione della struttura del *database*, mantenendo sincronizzati il modello dei dati nell’applicazione e la struttura effettiva del *database*.

### 3.2.2.2 *Business Logic layer e Service Pattern*

Il *Business Logic layer* in MoviORDER implementa il *Service Pattern*, un modello di progettazione *software* che separa la logica di *business* dal resto del sistema. Questo *patter* funge da intermediario tra il *layer* di presentazione (*API Controllers*) e il *Data Access layer*.

Il *Service Pattern* è implementato principalmente attraverso un componente chiave: `BaseService<TContext, TEntity>` che gestisce il flusso di dati, definisce le operazioni CRUD e incapsula la logica dei servizi.

I servizi sono classi concrete che estendono `BaseService` ed implementano la logica di *business* che viene richiamata dagli *API Controllers* del livello superiore.

Questo approccio offre diversi vantaggi:

- **Separazione delle responsabilità:** la logica di *business* è chiaramente distinta dalla presentazione e dall’accesso ai dati;
- **Riusabilità:** le funzionalità incapsulate nei servizi sono facilmente riutilizzabili in diverse parti dell’applicazione;

- **Manutenibilità:** la struttura modulare facilita la manutenzione e l'estensione del codice;

I servizi gestiscono anche gli errori e mappano i dati in DTO (vedi capitolo 3.2.2.3), aumentando la modularizzazione.

In questo *layer* sono implementati meccanismi di sicurezza basati su *token*. Generato all'autenticazione dell'utente, il *token* contiene informazioni criptate utilizzate per verificare identità e permessi ad ogni richiesta successiva, garantendo un accesso sicuro alle risorse dell'applicazione.

In conclusione, il *Business layer* e il *Service Pattern* in MoviORDER forniscono una struttura robusta per l'implementazione della logica di *business*, fungendo da ponte efficace tra presentazione e accesso ai dati, e assicurando modularità, riusabilità e manutenibilità del codice.

### 3.2.2.3 DTO

L'utilizzo diretto dei modelli come classi *standard* presenta due criticità:

- **Vulnerabilità nella sicurezza dei dati:** la creazione di istanze dirette dei modelli può esporre involontariamente informazioni sensibili. Un esempio è la tabella **User** del *common database*, dove tra gli attributi troviamo la **password** dell'utente. L'utilizzo di queste istanze potrebbe portare alla divulgazione accidentale di dati riservati in parti dell'applicazione dove non sono necessari;
- **Limitazioni nella flessibilità strutturale:** i modelli generati rispecchiano fedelmente la struttura del *database*, ma spesso sono richieste rappresentazioni dei dati più sofisticate o personalizzate. In molti casi, è preferibile definire classi che aggregano o rielaborano dati provenienti da più modelli, offrendo una rappresentazione più adatta alle esigenze funzionali dell'applicazione.

Ecco perché vengono introdotti i DTO (*Data Transfer Object*).

Essenzialmente, sono contenitori di dati privi di logica di *business*, progettati

per trasportare informazioni tra i componenti del sistema. Tipicamente contengono solo proprietà pubbliche, senza implementare comportamenti complessi, permettendo di controllare precisamente quali dati vengono esposti e trasferiti, migliorando significativamente la sicurezza del sistema.

Questo è particolarmente importante per proteggere informazioni sensibili, come *password* o altri dati riservati, che possono essere omessi o mascherati.

L'utilizzo dei DTO incrementa anche la manutenibilità del codice, introducendo un livello di astrazione tra la struttura del *database* e la logica applicativa, permettendo modifiche alla struttura del *database* o alla *Business Logic* senza impattare direttamente le interfacce esposte.

Quello dei DTO non è un vero e proprio *layer*, ma come mostrato nella figura 3.3 agisce da ponte tra il *Presentation layer* e il *Business layer*, in quest'ultimo infatti viene gestita la mappatura dei DTO al corrispondente modello in modo da poter convertire un modello in DTO e viceversa. È possibile anche mappare tra loro i DTO in modo da avere una gestione profonda del passaggio delle informazioni.

#### 3.2.2.4 *Presentation layer* e *API Controller*

Il *Presentation layer* rappresenta lo strato più esterno dell'architettura API di MoviORDER, fungendo da interfaccia tra il sistema *back-end* e il *front-end*. Questo livello è implementato principalmente attraverso gli *API Controller*, che sono responsabili della gestione delle richieste HTTP in ingresso e della formattazione delle risposte.

Questi *controller* agiscono come punto di ingresso per le richieste *client*, orchestrando il flusso di dati e le operazioni tra il *client* e il *Business Logic layer*.

Le principali responsabilità degli *API Controller* includono:

- **Gestione delle richieste:** Ricevono e interpretano le richieste HTTP in arrivo.
- **Routing:** Indirizzano le richieste alle appropriate funzioni del *Business Logic layer*.

- **Formattazione risposte:** Preparano e inviano risposte HTTP appropriate utilizzando i DTO.

Gli *API Controller* interagiscono direttamente con il *Business Logic layer*, in particolare con i servizi che vengono richiamati direttamente nel corpo del *controller*.

Tutte le *API G* a parte quella per il *login* non possono essere interrogate da un utente non autenticato (ovvero che non ha completato la procedura di *login* e che non possiede un *token* valido), in modo da concederne l'utilizzo solo a gli utenti di MoviORDER.

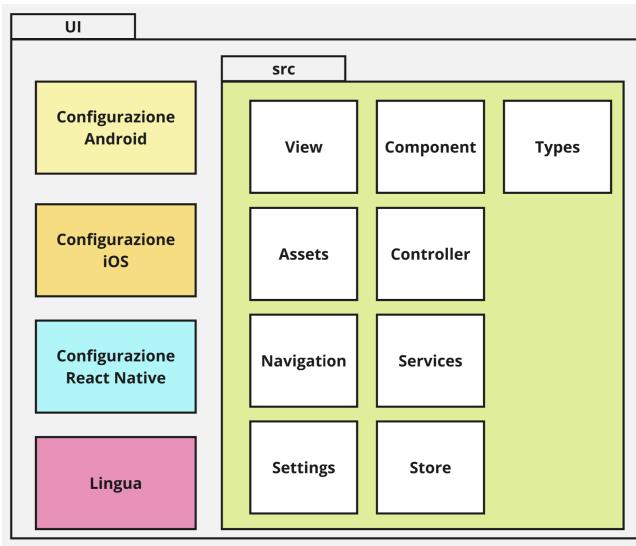
All'interno della cartella dove sono definiti i *controller* troviamo inoltre:

- **Program.cs:** questo file è il punto di ingresso dell'applicazione, dove viene configurato e avviato il *server web* che ospita le *API G*;
- **Configurazione di Swagger:** ovvero lo strumento usato per il *testing* delle *API G*;
- **Definizione degli endpoint:** ovvero gli indirizzi di connessione ai vari *database* e l'indirizzo in cui le *API G* vengono esposte.

In conclusione, il *Presentation layer* e gli *API Controller* in MoviORDER fungono da interfaccia tra il mondo esterno e la logica interna dell'applicazione. Attraverso una progettazione attenta e l'uso di DTO, questo *layer* garantisce una comunicazione efficiente, sicura e flessibile con il *front-end*, mantenendo al contempo una chiara separazione delle responsabilità all'interno dell'architettura complessiva del sistema.

### 3.2.3 *Front-end*

#### 3.2.3.1 *Root della repository e file di configurazione*



**Figura 3.4:** Root della repository che contiene il *front-end*

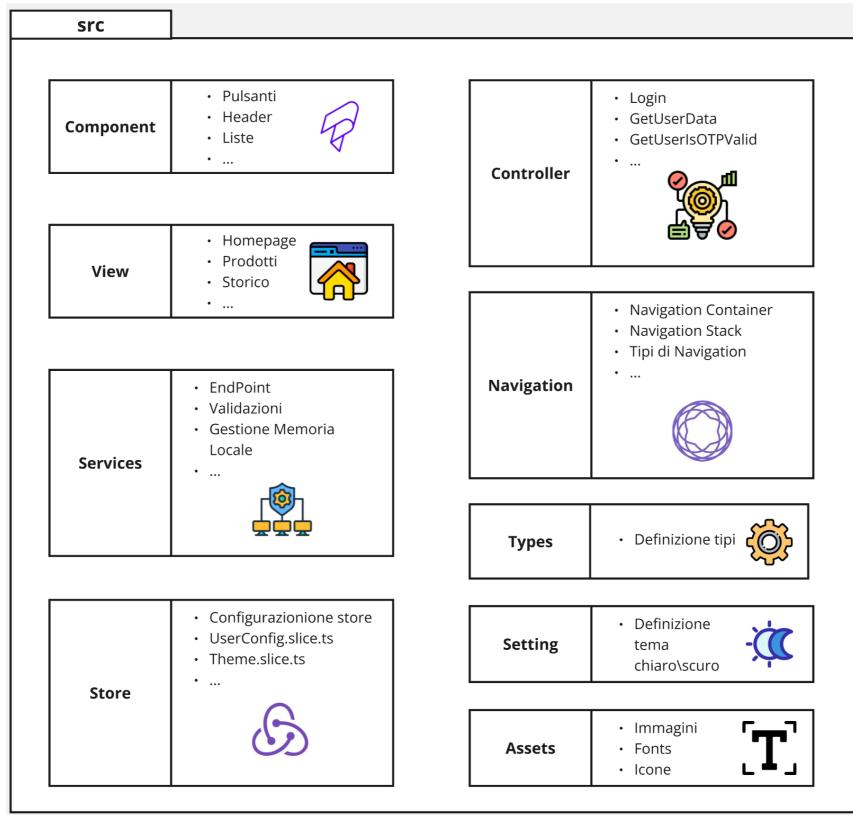
Prima di descrivere l'architettura del *front-end* soffermiamoci ad analizzare il contenuto del root della *repository* mostrata dalla figura 3.4. Qui sono contenuti i file di configurazione di React Native e le impostazioni dei compilatori.

La *repository* è suddivisa nelle seguenti cartelle:

- **Android:** specifica per React Native, contiene i file di configurazione per la versione Android dell'*app* e file come `build.gradle` e `AndroidManifest.xml` per la configurazione del compilatore Android;
- **iOS:** specifica per React Native, contiene il progetto Xcode e i file di configurazione per la versione iOS dell'*app* e file per la configurazione del compilatore di iOS;
- **lang:** contiene i file `en.json` e `it.json` che definiscono il testo usato nell'applicazione nelle lingue italiano e inglese;
- **src:** contiene tutto il codice del *front-end* e realizza l'architettura;

- **Altri *file* di configurazione:** questi *file* sono fondamentali per garantire un corretto funzionamento e una gestione efficiente del progetto, qui riporto quelli più importanti:
  - **package.json**: contiene le informazioni sul progetto come il nome del progetto, versione, *script* di comando e dipendenze. È il *file* principale per gestire i pacchetti Node.js utilizzati nel progetto;
  - **yarn.lock**: questo *file* viene generato automaticamente quando si genera il progetto e permette di gestire ed installare le dipendenze con le loro versioni esatte;
  - **metro.config.js**: configura Metro, il *bundler* di JavaScript predefinito per React Native. Un *bundler* è uno strumento di sviluppo *software* che combina diversi *file* di codice sorgente e le loro dipendenze in uno o più *file* ottimizzati, pronti per essere distribuiti o eseguiti in un ambiente di produzione. Questo *file* permette di personalizzare il comportamento di Metro, come l'aggiunta di *alias*, *path* personalizzati, o l'esclusione di determinati *file* dalla *build*.
  - **babel.config.js**: configura Babel, il *transpiler* di JavaScript. Definisce come il codice deve essere trasformato per essere compatibile con le varie versioni di JavaScript e i diversi ambienti in cui verrà eseguito.
  - **index.js**: punto d'ingresso principale dell'applicazione React Native. Qui viene avviato il *rendering* del componente radice dell'*app*;
  - **app.tsx**: contiene il componente radice dell'applicazione. Qui vengono definiti l'interfaccia utente principale e la logica di base dell'*app*.
  - **tsconfig.json**: configura il compilatore TypeScript, specificando le opzioni di compilazione e il comportamento del *transpiling* del codice TypeScript in JavaScript.

### 3.2.3.2 Architettura React



**Figura 3.5:** Rappresentazione dell’architettura React per MoviORDER

Una caratteristica distintiva di React è la sua flessibilità nell’implementazione dell’architettura dell’applicazione, distinguendosi da altri *framework* JavaScript che impongono modelli predefiniti. Questa libertà consente agli sviluppatori di strutturare l’applicazione in base alle specifiche esigenze del progetto.

L’architettura React, come illustrata la figura 3.5, si configura come un insieme di componenti responsabili della costruzione dell’interfaccia utente (UI) del *software*. Questa architettura può essere concepita come un’organizzazione del codice che facilita la realizzazione di progetti personalizzati, includendo vari elementi UI quali pulsanti, moduli, servizi *API* e sistemi di gestione dello stato. L’architettura si articola nelle seguenti directory principali:

- **View**: contiene le pagine complete dell’applicazione, composizioni di componenti più piccoli che formano l’interfaccia utente. Le viste gestiscono

la logica e l'organizzazione dei componenti per creare l'esperienza utente complessiva;

- **Component**: ospita componenti UI riutilizzabili e modulari. Questi possono essere elementi come pulsanti, *form*, barre di navigazione o qualsiasi altro elemento dell'interfaccia che può essere utilizzato in più parti dell'applicazione. I componenti in questa cartella sono generalmente più piccoli e più specifici rispetto alle viste;
- **Controller**: contiene la logica di *business* dell'applicazione. Qui si trovano le funzioni e le classi che gestiscono la logica applicativa, elaborano i dati e coordinano le interazioni tra i vari componenti e servizi;
- **Services**: gestisce le interazioni con risorse esterne, come chiamate *API G* o la memoria del dispositivo. Questa cartella contiene il codice per la comunicazione con il *back-end*, la gestione delle richieste HTTP e l'elaborazione delle risposte. Qui sono definiti inoltre gli *endpoint* per la connessione alle *API G*;
- **Store**: centralizza la gestione dello stato dell'applicazione utilizzando Redux. Questa cartella contiene la configurazione dello *store* Redux e i *reducer* che definiscono come lo stato dell'applicazione cambia in risposta alle azioni e dei selettori per accedere allo stato;
- **Navigation**: implementa la logica di *routing* e navigazione dell'applicazione utilizzando React Navigation. Include la mappatura tra i nomi dei *routers* (componenti che definiscono lo stato della navigazione) e le *view* corrispondenti, nonché opzioni di configurazione per ogni schermata come titoli e animazioni di transizione;
- **Types**: questa cartella contiene le definizioni dei tipi personalizzati utilizzati in tutta l'applicazione. Ciò include interfacce, tipi e enumerazioni che aiutano a mantenere il codice tipizzato e più robusto;

- **Assets**: contiene risorse statiche come immagini, icone, *font* e altri *file* multimediali utilizzati nell'applicazione. Queste risorse sono accessibili e utilizzabili in tutto il progetto;
- **Setting**: definisce i *file* di configurazione per temi (chiaro/scuro).

### 3.3 Codifica

Una volta compresa l'architettura *software* dell'*app* ho iniziato l'attività di codifica, dove ho tradotto i requisiti riportati dalla tabella 3.1 in codice.

Questa attività ha richiesto una particolare attenzione all'integrazione del nuovo codice con quello esistente, in modo da non modificare il comportamento del *software* per i clienti. Il modulo agenti mi ha richiesto l'estensione o la modifica di quasi ogni componente dell'architettura precedentemente descritta e mi ha permesso di esplorare il funzionamento dell'applicazione nella sua interezza.

#### 3.3.1 *Database e Data Access layer*

Sono partito modificando la struttura del *database* aggiungendo una colonna **IdUser** alla tabella **User** del *common database*. Utilizzando le funzionalità di Entity Framework mi è bastato modificare il modello dei dati e lanciare i comandi per rendere effettive le modifiche anche nel *database*. Con questa nuova colonna ora posso identificare i clienti dal fatto che la colonna **BPCode** risulta nulla e la colonna **IdUser** non nulla, viceversa i clienti.

La seconda modifica è stata l'importazione della tabella **Bp** del *company database* che contiene i dati inerenti ai clienti aziendali. Il *company database* è stato creato dall'*app* MoviSELL, e MoviORDER ne utilizza qualche tabella necessaria per svolgere le sue funzioni. Per questo motivo per integrare **Bp** nel modello di MoviORDER ho lavorato al contrario utilizzando il comando **scaffold** di Entity Framework che permette di integrare una tabella già presente nel *database* al modello.

### 3.3.2 API e *Business Logic layer*

Per permettere il funzionamento del modulo agenti ho dovuto modificare una funzione del *Business layer* e creare due nuove *API<sub>G</sub>*.

La funzione `Login`, richiamata dall'omonimo *API<sub>G</sub> Controller*, è la funzione che si occupa di autenticare l'utente all'interno dell'*app*. Le modifiche che ho apportato riguardano una serie di controlli aggiuntivi per permettere l'autenticazione degli agenti. Ho inoltre modificato le informazioni contenute nel *token* restituito a gli agenti per ottimizzare la funzione `GetAgentCustomers` che verrà discussa in questo capitolo.

Una delle due *API<sub>G</sub>* che ho creato per il modulo agenti è `AdditionalLogin`, richiamata dall'omonimo *API<sub>G</sub> Controller*. Questa funzione, definita nello stesso servizio di `Login`, permette di effettuare una seconda *login* nell'*app* saltando i controlli sulla *password* e restituendo un nuovo *token*. `AdditionalLogin` è fondamentale per permettere all'agente di autenticarsi nell'*app* come il cliente selezionato grazie al sistema di gestione del doppio *token* che ho implementato nel *front-end*. La sicurezza di questa *API<sub>G</sub>* è assicurata dal fatto che può essere richiamata solo da un utente autenticato, e necessita quindi che l'utente che richiama l'*API<sub>G</sub>* sia un utente MoviORDER.

L'ultima funzione che ho creato è `GetAgentCustomers`, richiamata dall'omonimo *API<sub>G</sub> Controller* e sviluppata all'interno del nuovo servizio `BpService`. Questa funzione recupera tutte le informazioni inerenti ai clienti dell'agente e ritorna una lista di `BpCustomerDto`, un nuovo DTO che contiene una serie di informazioni ricavate sul cliente.

### 3.3.3 *Front-end*



**Figura 3.6:** Versione finale delle *homepages* di MoviORDER

Lo sviluppo del modulo ha richiesto diverse aggiunte e modifiche anche alla parte di *front-end*.

Per quanto riguarda l’interfaccia ho sviluppato la *view Homepage Agenti* e tutti i *component* utilizzati dalla pagina. Questa *view* è il luogo in cui viene portato l’utente dopo aver completato l’autenticazione con successo ed esser stato riconosciuto come agente aziendale. Qui l’agente può accedere alla pagina **Impostazioni** e **Account**, oppure consultare la lista dei propri clienti. Una volta selezionato un cliente dalla lista, l’agente viene portato nella *Homepage* dove può accedere a tutte le funzionalità dell’*app* e operare come il cliente selezionato.

Ovviamente si è visto necessaria la modifica della *Homepage*, aggiungendo un pulsante che permettesse all’agente di tornare alla selezione del cliente (componente nascosto ai clienti) e la rimozione della possibilità per gli agenti di modificare la propria schermata di avvio.

Entrambe le viste descritte sono mostrate dall’immagine [3.6](#).

Ovviamente per integrare queste nuove *view* nel sistema ho dovuto modificare lo **Store Redux** e lo *stack* di navigazione di **Navigation** in modo da rendere visualizzabili le modifiche solo ai clienti e per permettere una corretta visualizzazione della pagina e i suoi componenti sia nel caso l'agente effettui l'operazione di *login*, sia nel caso in cui l'agente abbia salvato le proprie credenziali in modo da rendere più rapido l'accesso al sistema.

La seconda modifica importante al *front-end* è la possibilità per il sistema di gestire due *token*. Il secondo *token* viene restituito da **AdditionalLogin** in seguito alla selezione da parte dell'agente di un cliente dalla lista. Questo *token* è fondamentale per richiamare alcune **API<sub>G</sub>** all'interno delle pagine dell'*app*. Per questo, controllando una variabile booleana nello stato, la funzione `getToken` decide quale dei due *token* ritornare permettendo la corretta integrazione del modulo al sistema.

Il lavoro inerente il lato estetico e di usabilità dell'*app* ha compreso:

- **Ottimizzazione per *tablet*:** ho adeguato il CSS della *view* **Homepage Agenti** in modo da renderla utilizzabile anche da *tablet*;
- **Modifiche generali:** leggere modifiche allo stile di *component* e *view* preesistenti;
- **Adeguamento di *Homepage Agenti* al tema selezionato:** possibilità di selezionare il tema chiaro o scuro anche per *Homepage Agenti*;
- **Adeguamento del modulo agenti alla lingua di sistema:** possibilità di vedere le nuove frasi introdotte nell'*app* con il modulo agenti in italiano e inglese.

## 3.4 Verifica e documentazione

### 3.4.1 *Testing*

Un'attività fondamentale del ciclo di sviluppo è il *testing* delle modifiche apportate al sistema.

VisioneImpresa non utilizza strumenti per la verifica automatica del codice, e ho quindi testato manualmente tutte le novità introdotte e la corretta integrazione del modulo agenti.

Tuttavia, per mia iniziativa e in accordo con il *tutor* aziendale, ho deciso di implementare una *suite* di *test* automatici per alcune funzioni della *Business Logic* del *back-end* utilizzando il *framework* Moq e xUnit, un *tool* per creare *unit test* in .NET.

Moq è uno strumento per la creazione di *mock* e *stub* in *unit test* e *integration test*. Permette di simulare il comportamento di oggetti e componenti esterni al codice che si sta testando, sostituendoli con implementazioni fintizie ma controllabili. Nello specifico, Moq consente di:

- **Creare *mock* di interfacce e classi concrete**, in modo da poter testare il codice in modo isolato senza dipendere dalle reali implementazioni.
- **Definire aspettative sui metodi testati**, verificando che il codice che si sta testando interagisca correttamente con i *mock* forniti.

Con l'utilizzo di Moq ho potuto simulare le chiamate a *database* e ad altre funzioni, restituendo oggetti creati *ad hoc* per testare diverse casistiche e verificare il corretto comportamento delle funzioni testate, indipendentemente dall'integrazione con il resto del sistema.

La prima difficoltà che ho incontrato in questa fase è stata la creazione di oggetti *mock* particolarmente complessi. A volte infatti i servizi richiedevano l'utilizzo di oggetti con molti attributi spesso non banali.

Inoltre, la versione gratuita di Moq non mi permetteva di simulare il comportamento di metodi non pubblici e virtuali, costringendomi a modificare la firma di alcune funzioni.

Sebbene questo non rappresenti l'approccio ideale per l'implementazione di *test* unitari, ho comunque deciso di cambiare la firma delle funzioni. La mia motivazione principale era quella di dimostrare l'efficacia dei *test* automatici nell'individuare e prevenire alcuni problemi dati dalla continua evoluzione del codice, come l'introduzione di regressione nelle funzionalità del sistema. Si parla di regressione quando una funzionalità del sistema presenta un comportamento

differente e non previsto con l'introduzione di novità nel codice.

Ho anche provato ad implementare una serie di *test* automatici per le funzioni del *front-end*, ma per mancanza di tempo ho preferito dare priorità alla stesura della documentazione, e ho quindi rinunciato.

### 3.4.2 Documentazione

Un'attività fondamentale dello sviluppo è quello della stesura della documentazione.

La documentazione è una componente cruciale dello sviluppo *software*, non solo perché facilita la manutenzione e l'evoluzione del *software*, ma anche perché aiuta i nuovi sviluppatori a comprendere rapidamente il sistema e supporta gli utenti finali nell'utilizzo efficace del prodotto.

Come richiesto dal *tutor* aziendale, ho prodotto due documenti inerenti il modulo agenti: un documento di specifica tecnica e un manuale utente.

Il manuale utente illustra in maniera semplice e precisa le funzionalità offerte dalla nuova versione di MoviORDER, senza utilizzare linguaggio tecnico. Per scrivere questo documento ho utilizzato il *software* Figma, come riportato nel capitolo 2.4.1, in modo da avere maggiori strumenti per curare l'estetica del documento. Abbondano anche le immagini, che forniscono un supporto visivo al testo e aiutano l'utente nell'utilizzo delle nuove funzionalità.

Viceversa il documento di specifica tecnica risulta essenziale dal punto di vista estetico e utilizza un linguaggio tecnico e settoriale descrivendo in maniera precisa tutte le modifiche introdotte e la logica di funzionamento del modulo agenti. Questo documento è destinato a sviluppatori e addetti ai lavori, in modo che possa servire in futuro a guidare l'implementazione del modulo nel prodotto reale.

Nella specifica tecnica, nell'apposita sezione, ho descritto una serie di *test* manuali, sia per l'interfaccia che per le *API*, in modo da dimostrare la corretta integrazione del nuovo modulo con il sistema esistente.

## 3.5 Risultati

Alla fine delle 300 ore di *stage* sono riuscito con successo nell'implementazione del modulo agenti all'interno di MoviORDER. Questo è confermato dal *tutor* e dal *team* di sviluppatori assegnati all'*app* che confermano il soddisfacimento di tutti i requisiti riportati nella tabella 3.1.

Per la realizzazione di questo progetto ho:

- **Modificato una tabella del *database*;**
- **Introdotto una nuova tabella nel modello delle  $\textcolor{blue}{API}_G$ ;**
- **Modificato l' $\textcolor{blue}{API}_G$  di *login*;**
- **Creato due nuove  $\textcolor{blue}{API}_G$ :** funzioni, interfacce, servizi e  $\textcolor{blue}{API}_G$  *Controller*;
- **Creato una nuova *view*;**
- **Modificato sette *view* pre esistenti per integrarle al nuovo modulo;**
- **Modificato due *slice* dello *store Redux*:** introdotte nuove variabili, funzioni e *reducer* per la gestione dello stato. Alcuni *slice* hanno invece richiesto delle modifiche per integrare correttamente il modulo agenti;
- **Modificato lo *stack* di *Navigation*, creando il *router* per gestire Homepage Agenti e inserendolo nello *stack*;**
- **Creati gli *endpoint* per permettere al *front-end* di interrogare le  $\textcolor{blue}{API}_G$ ;**
- **Modificate le due funzioni che gestiscono il *token* per implementare il sistema a due *token*;**
- **Creato un nuovo **DTO** per ritornare le informazioni riguardo i clienti;**
- **Creato un nuovo tipo in React per recuperare le informazioni inerenti i clienti;**

- **Create due funzioni in *Controller*** per gestire la logica di *business* di React, molte altre pre esistenti sono state modificate;
- **Creati cinque nuovi *component*,** molti altri pre esistenti sono stati modificati;
- **Modificato lo stile della nuova *view* e di quelle pre esistenti:** integrazione con il tema scuro, possibilità di utilizzare l'applicazione in inglese o italiano, adeguamento dello stile per il *tablet*;
- **Creato diciotto *unit test* per cinque funzioni** della *business logic* del *back-end*.
- **Creato una *suite* di test manuali** per il *front-end* e per il *back-end*;
- **Creato la documentazione richiesta:** manuale utente e specifica tecnica.

# Glossario

**API** *Application Programming Interface* (interfaccia di programmazione delle applicazioni) sono un insieme di definizioni e protocolli con i quali vengono realizzati e integrati *software applicativi*. Le API stabiliscono il contenuto e la forma dei dati necessari per la chiamata e quelli restituiti in risposta. [fonte riportata in sitografia]. [11](#), [13](#), [14](#), [18](#), [20–23](#), [33](#), [35](#), [36](#), [41](#), [44](#), [45](#), [47](#), [49](#), [51](#), [52](#)

**CSR** Per Responsabilità Sociale delle Imprese (e delle organizzazioni) o secondo l'acronimo inglese CSR, *Corporate Social Responsibility*, si intende l'integrazione su base volontaria, da parte delle imprese, delle preoccupazioni sociali e ambientali nelle loro operazioni interessate. [fonte riportata in sitografia]. [iii](#), [6](#)

**ERP** *Enterprise Resource Planning*, è un tipo di sistema *software* che aiuta le organizzazioni ad automatizzare e gestire i processi aziendali principali per ottenere le prestazioni ottimali. Il *software* ERP coordina il flusso di dati tra i processi di un'azienda, fornendo un'unica fonte di informazioni e semplificando le operazioni nell'azienda. È in grado di collegare le attività finanziarie, della catena di approvvigionamento, delle operazioni, del commercio, dei *report*, della produzione e delle risorse umane di un'azienda in una sola piattaforma.

[fonte della definizione inglese riportata in sitografia]. [1](#), [3](#), [15](#)

**IDE** *Integrated Development Environment* o ambiente di sviluppo integrato, è un *software* progettato per la realizzazione di applicazioni che aggrega strumenti di sviluppo comuni in un'unica interfaccia utente grafica. In

genere è costituito da: *editor* del codice sorgente, strumenti che consentono di automatizzare la *build* locale, un *debugger* e strumenti per l'esecuzione di test automatici.

[fonte riportata in *sitografia*]. [10](#)

**ORM** *Object-Relational Mapping*, è uno strumento che facilita l'interazione tra il codice orientato agli oggetti e i *databases* relazionali. Esso permette agli sviluppatori di manipolare i dati del *database* usando oggetti e metodi del linguaggio di programmazione, anziché scrivere *query SQL* dirette. Questo approccio aumenta la produttività, migliora la manutenibilità del codice e riduce il rischio di errori legati alla gestione diretta del database . [22](#)

**Web App** L'applicazione *web*, o abbreviato *web app*, nell'ambito dell'informatica e della programmazione, si riferisce alle applicazioni accessibili e fruibili attraverso il *web*, quindi accessibili dall'utente tramite un *browser web* con una connessione attiva . [4](#), [5](#), [11](#)

# Sitografia

*Definizione API.* URL: <https://www.redhat.com/it/topics/api/what-is-a-rest-api> (visitato il 04/08/2024).

*Definizione CSR.* URL: <https://www.lavoro.gov.it/temi-e-priorita/terzo-settore-e-responsabilita-sociale-imprese/focus-on/responsabilita-sociale-imprese-e-organizzazioni/pagine/default> (visitato il 11/08/2024).

*Definizione ERP.* URL: <https://www.microsoft.com/en-us/dynamics-365/topics/erp/what-is-erp> (visitato il 11/08/2024).

*Definizione IDE.* URL: <https://www.redhat.com/it/topics/middleware/what-is-ide> (visitato il 11/08/2024).

*Funzionalità di 3CX.* URL: <https://www.3cx.it/> (visitato il 28/07/2024).

*Funzionalità di SSMS.* URL: <https://learn.microsoft.com/it-it/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16> (visitato il 28/07/2024).

*Manifesto Agile.* URL: <https://www.atlassian.com/it/agile/manifesto> (visitato il 27/07/2024).

*Metro.* URL: <https://reactnative.dev/docs/metro> (visitato il 24/08/2024).

*Moq.* URL: <https://github.com/devlooped/moq> (visitato il 24/08/2024).

*React Architecture.* URL: <https://www.geeksforgeeks.org/react-architecture-pattern-and-best-practices/> (visitato il 24/08/2024).

*React Native Navigation.* URL: <https://reactnavigation.org/docs/getting-started> (visitato il 24/08/2024).

## CAPITOLO 3. SITOGRAFIA

---

*Repository Pattern.* URL: <https://learn.microsoft.com/it-it/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design> (visitato il 24/08/2024).

*Scrum.* URL: <https://www.atlassian.com/it/agile/scrum> (visitato il 27/07/2024).

*Service Pattern.* URL: <https://medium.com/@ankitpal181/service-repository-pattern-802540254019> (visitato il 24/08/2024).

*Sito di VisioneImpresa.* URL: <https://www.vsh.it/> (visitato il 27/07/2024).

*Swagger.* URL: <https://www.geekandjob.com/wiki/swagger> (visitato il 04/08/2024).

*Tipologia di ticket Jira.* URL: <https://www.atlassian.com/it/software/jira/guides/issues/overview#what-are-issue-types> (visitato il 27/07/2024).

*xUnit.* URL: <https://xunit.net/> (visitato il 24/08/2024).