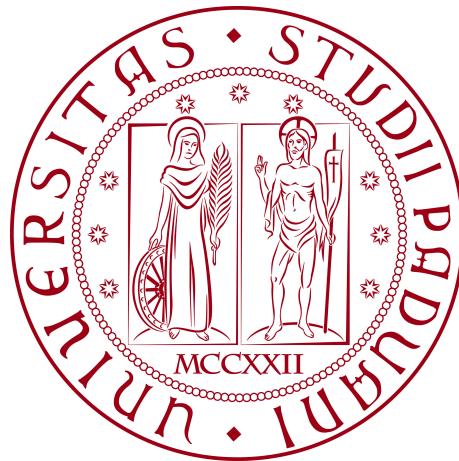


Università degli Studi di Padova

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_G*).

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

1 VisioneImpresa	1
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
2 Descrizione del progetto	16
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
3 Stage	26
3.1 Studio	26
3.1.1 Struttura del <i>database</i>	26

3.1.2	Architettura delle API	27
3.1.2.1	<i>Data Access layer e Repository Pattern</i>	29
3.1.2.2	<i>Business Logic layer e Service Pattern</i>	30
3.1.2.3	DTO	31
3.1.2.4	Presentation layer e API Controller	32
3.1.3	<i>Front-end</i>	34
3.1.3.1	<i>Root della repository e file di configurazione</i> . .	34
3.1.3.2	Architettura React	36
3.2	<i>Setup</i>	38
3.3	Analisi	40
3.3.1	Casi d'uso	40
3.3.1.1	UC 1 - <i>Login</i>	41
3.3.1.2	UC 2 - Operazioni disponibili nella Homepage Agenti	43
3.3.1.3	UC 2.1 - Interazioni con la lista clienti	45
3.3.1.4	UC 3 - Operazioni disponibili nella Homepage Agenti autenticati come clienti	49
3.3.1.5	UC 4 - Cambio tema	51
3.3.1.6	UC 5 - <i>Logout</i>	52
3.3.2	Requisiti funzionali e non funzionali	53
3.4	Progettazione	59
3.5	Sviluppo	60
3.5.1	Modifica del <i>database</i> e funzione <i>Login</i>	60
3.5.2	Sviluppo del modulo agenti	61
3.5.2.1	<i>API_G</i> <code>GetCustomerList</code>	61
3.5.2.2	<i>Homepages</i>	62
3.5.2.3	<i>API_G</i> <code>AdditionalLogin</code>	63
3.5.3	Modifica delle interfacce	65
3.6	Verifica	66
3.7	Risultati	67

INDICE

Sitografia

iii

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>Database</i> di MoviORDER.	26
3.2	Architettura API.	28
3.3	<i>Root</i> della <i>repository</i> che contiene il <i>front-end</i>	34
3.4	Rappresentazione dell'architettura React per MoviORDER	36
3.5	<i>Use Case</i> 1: <i>Login</i> Utente	41
3.6	<i>Use Case</i> 2: Operazioni disponibili nella Homepage Agenti	43
3.7	<i>Use Case</i> 2.1: Interazioni con la lista clienti	45
3.8	<i>Use Case</i> 3: Operazioni disponibili nella Homepage Agenti	49
3.9	<i>Use Case</i> 4: Cambio tema	51
3.10	<i>Use Case</i> 5: <i>Logout</i>	52
3.11	Versione finale delle <i>homepages</i> di MoviORDER	62
3.12	Versione <i>tablet</i> di Homepage Agenti e <i>search bar</i> per la ricerca clienti nella lista	65

Elenco delle tabelle

2.1	Tabella Obiettivi	24
2.2	Tabella Obiettivi Personali	25
3.1	Tabella del tracciamento dei requisiti funzionali e non funzionali.	59
3.2	Riepilogo requisti	59

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_G* (*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_G*), 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 ¹:

- 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.

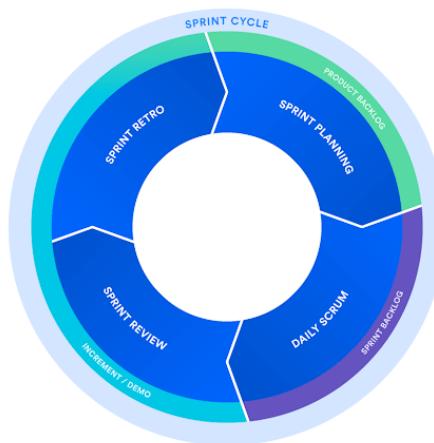


Figura 1.2: Organizzazione di uno *sprint* con il *framework* Scrum.
fonte: <https://www.atlassian.com/it/agile/scrum>

¹Manifesto Agile per lo sviluppo *software*, fonte riportata nella sitografia 3.7.

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

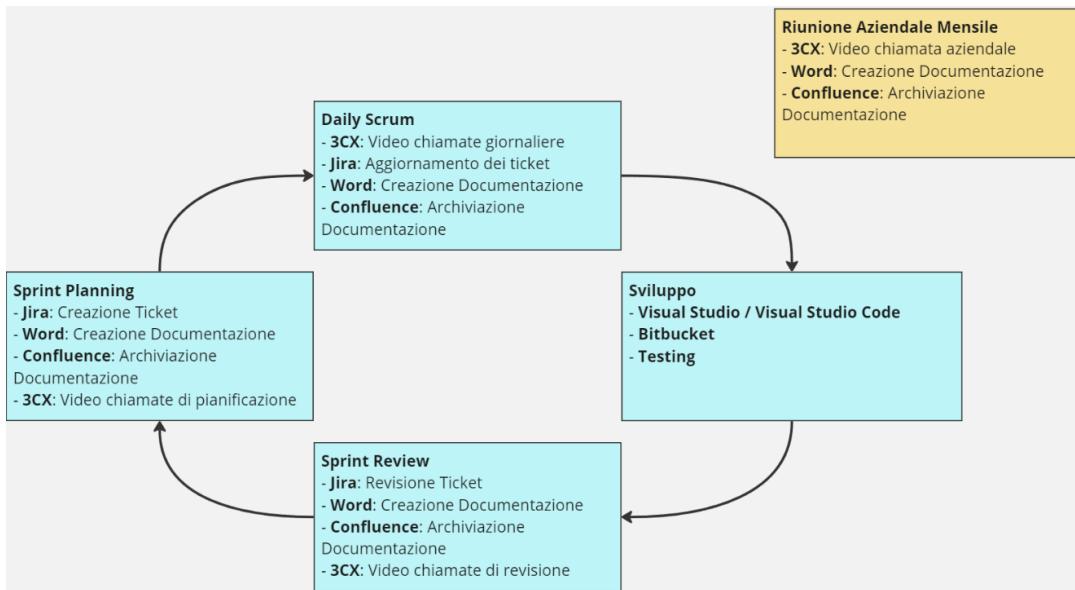


Figura 1.3: Integrazione metodologie e tecnologie.

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_G*, 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 del progetto

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.

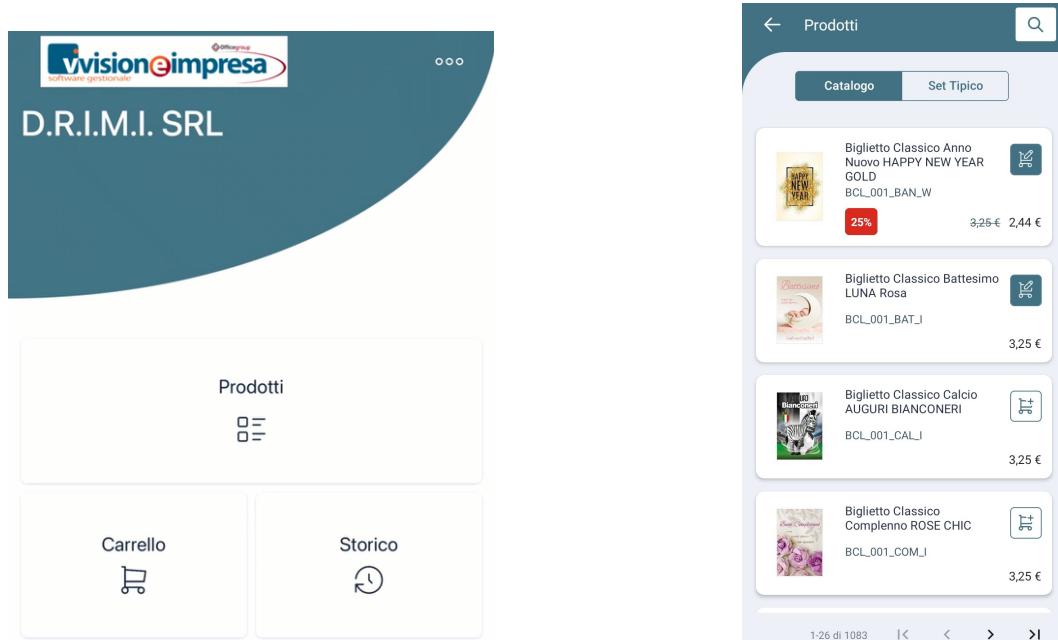
Al termine del lavoro, lo studente presenta il suo operato agli sviluppatori e al *project manager* del prodotto in questione. Questo *meeting* è cruciale per valutare la qualità del lavoro svolto e comprendere la logica di sviluppo adottata. Permette inoltre agli sviluppatori e al *project manager*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

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_G* 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 espliciti, eccetto l'uso di BitBucket come piattaforma per conservare il codice.

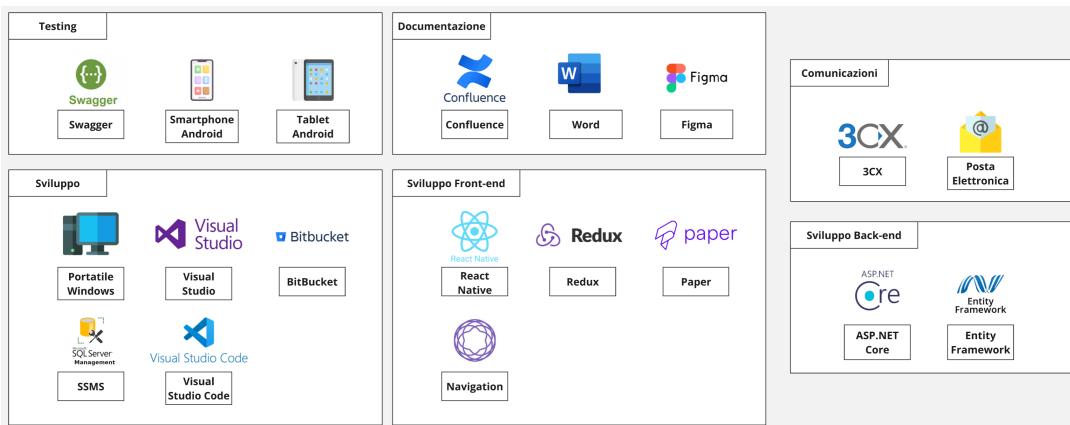


Figura 2.2: Tecnologie utilizzate per lo sviluppo di MoviORDER

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

- **Visual Studio:** per lo sviluppo delle *API_G*. 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 G* e valutarne il corretto funzionamento.

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

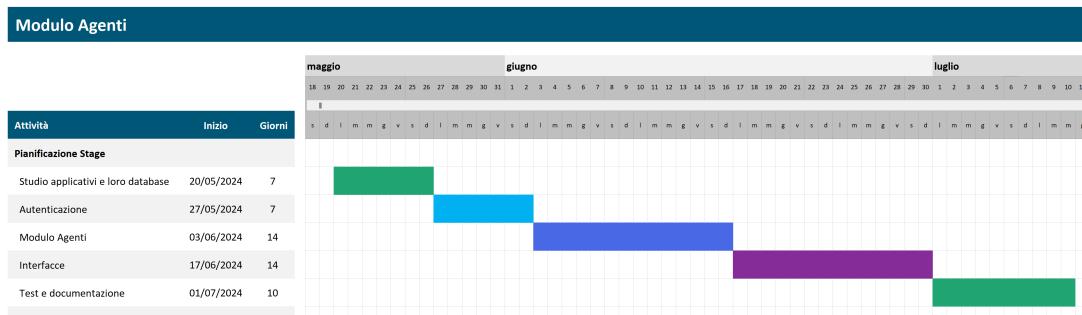


Figura 2.3: Pianificazione attività di *stage*

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 fasi 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_G* di autenticazione** (una settimana): l'obiettivo è modificare la *API_G* di *login* per il nuovo tipo di utente agente;
3. **Sviluppo del modulo agenti** (due settimane): prevede la creazione delle *API_G* 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

Tipo	Obiettivo
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

Stage

3.1 Studio

3.1.1 Struttura del *database*

La prima fase chiamata "Studio degli applicativi e del *database*", come introdotto dal capitolo 2.4.2, vede come obiettivo lo studio delle tecnologie, del codice e del *database* utilizzato dall'applicazione. Ho cominciato quindi dallo studio dell'architettura del *database*.

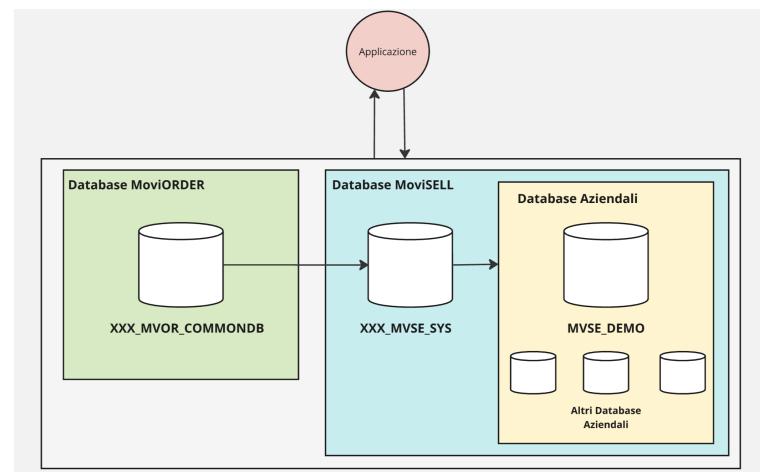


Figura 3.1: *Database* di MoviORDER.

Come illustrato dalla figura 3.1, 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 a gli 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.

3.1.2 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 *repository* BitBucket a sè 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.

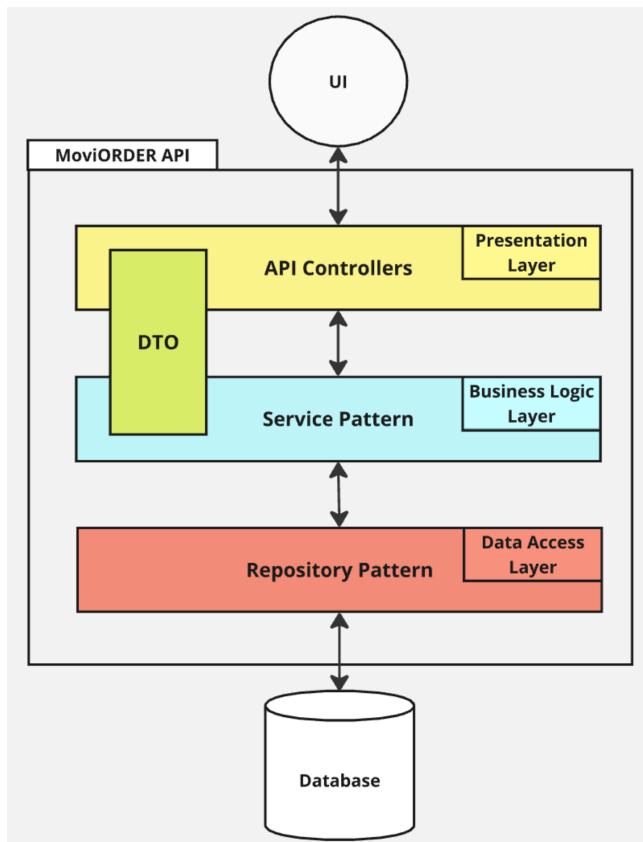


Figura 3.2: Architettura API.

Le *API* di MoviORDER adottano un'architettura a livelli, combinando il *Pattern Repository* e il *Pattern Service*, come mostra la figura 3.2, 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*.

3.1.2.1 *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*.

3.1.2.2 *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 3.1.2.3), 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.

3.1.2.3 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. Questo disaccoppiamento facilita la manutenzione e l'evoluzione del codice, 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 in figura 3.2 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.

3.1.2.4 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*.
- **Validazione input:** Verificano la correttezza e la sicurezza dei dati in ingresso.
- **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.

3.1.3 *Front-end*

3.1.3.1 *Root* della *repository* e *file* di configurazione

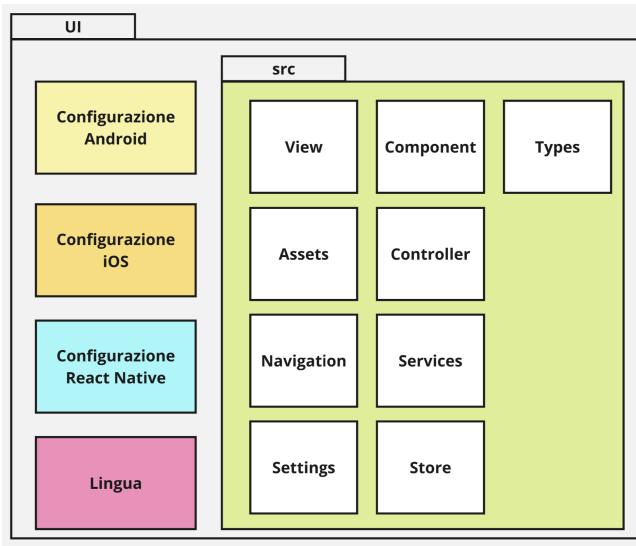


Figura 3.3: *Root* della *repository* che contiene il *front-end*

Infine ho studiato React Native e l’architettura utilizzata per il *front-end*.

Prima di descrivere l’architettura soffermiamoci ad analizzare il contenuto del *root* della *repository* mostrata dalla figura 3.3. 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.

3.1.3.2 Architettura React

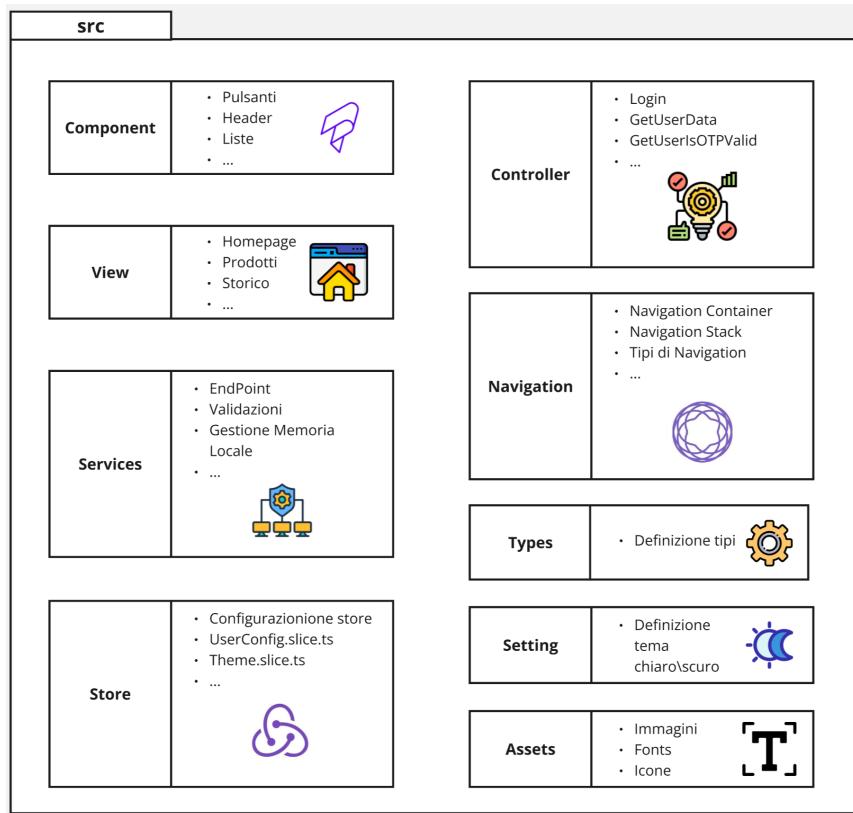


Figura 3.4: Rappresentazione dell’architettura React per MoviORDER

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 3.4, 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 centrale.

L’architettura React offre numerosi vantaggi nello sviluppo *web*. Favorisce un’architettura basata su componenti che migliora la manutenibilità e il riutilizzo del codice, consente una gestione efficace dello stato globale tramite librerie come Redux e la sua natura modulare facilita la scalabilità del progetto, permettendo

una crescita organica dell'applicazione. Infine, la struttura basata su componenti semplifica il processo di *unit testing*, aumentando l'affidabilità complessiva del *software*.

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 efficacemente 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).

3.2 *Setup*

Sempre durante la prima fase di studio, ho avviato il *setup* dell'ambiente di sviluppo. Il mio obiettivo era di riuscire a configurare i diversi *editor* ed eseguire il progetto in locale con il *database* di prova assegnatomi.

La configurazione del *database* in locale e la creazione di un utente di accesso per poter modificare le tabelle con SSMS si sono rivelate relativamente semplici, grazie l'aiuto degli sviluppatori. In meno di due ore il *database* era disponibile in locale sulla porta 2022.

L'avvio delle *API_G* in locale ha richiesto più tempo, ma sempre grazie al supporto del *team* di VisioneImpresa sono riuscito in poco tempo a completare il *setup* in tempi ragionevoli. Inizialmente ho configurato Visual Studio e risolto alcuni problemi di compatibilità dei pacchetti e versione di .NET. Quindi ho modificato gli *endpoint* di connessione al *database* e quelli in cui esporre le *API_G*, impostando come porta locale la 4048.

Questo compito è stato leggermente più complesso del precedente, poiché per evitare errori imprevisti ho dovuto limitare la possibilità di ricercare il *company database*. Come ho descritto nel capitolo 3.1.1 il *company database*, a differenza del *common e system database*, non è statico ma scelto dinamicamente a seconda del valore dell'attributo `CompanyCode` dell'utente durante la fase di *login*.

Con le modifiche apportate la ricerca del *company database* restituisce un unico risultato: `localhost:2022`.

Quando si avviano le *API_G* e si digita `localhost:4048` nella barra degli indi-

rizzi del *browser*, viene visualizzata l’interfaccia di Swagger. Questa permette di testare il corretto funzionamento delle *API_G* e la loro integrazione con il *database*.

La parte che ha richiesto più tempo, ritardando di alcuni giorni la fase di modifica dell’*API_G* di *login*, è stata l’avvio in locale del *front-end*. Questa attività ha richiesto diverso tempo a causa di vari errori che occorrevano durante l’avvio e del conseguente *troubleshooting* necessario per risolverli.

I problemi principali erano dovuti all’Android SDK, il *kit* di sviluppo *software* per Android. Nonostante fosse già installato sul *computer*, le variabili d’ambiente non erano configurate. Dopo aver sistemato le variabili d’ambiente, ho scoperto che ADB (*Android Debug Bridge*), uno strumento fondamentale di Android SDK, non risultava installato o utilizzabile, e ho dovuto quindi installarlo e configuralo separatamente.

Successivamente ho cambiato gli *endpoint* per consentire la connessione in locale alle *API_G* ed ho provato ad installare l’*app* nel AVD (*Android Virtual Device*). AVD è lo strumento offerto da Android Studio per simulare un dispositivo Android su *computer*, senza utilizzare un dispositivo reale. I limiti di questo approccio sono subito evidenti a causa dell’eccessiva pesantezza dello strumento che richiede prestazioni molto elevate. Ho quindi optato per l’utilizzo di uno *smartphone* fornito dall’azienda, sostituendo Android Studio con Visual Studio Code.

Per poter lavorare in locale con lo *smartphone*, ho dovuto collegarlo al *computer* e utilizzare il comando `abd reverse` dell’Android SDK. Questo comando permette di utilizzare le porte locali del dispositivo come se fossero quelle del *computer*. Ad esempio lanciando il comando `adb -s [nome del dispositivo] reverse tcp:4048 tcp:4048` e avviando l’applicazione dallo *smartphone*, al momento della connessione alle *API_G* locali (esposte sulla porta locale 4048 del *computer*), il dispositivo utilizzerà la porta del *pc* anziché la sua porta locale 4048.

Quest’ultima attività ha richiesto qualche giorno di lavoro, posticipando l’inizio della seconda fase. Tuttavia, al termine di questo processo, l’applicazione è stata completamente configurata per operare in locale, permettendo così l’inizio

dello sviluppo vero e proprio.

3.3 Analisi

3.3.1 Casi d'uso

I casi d'uso rappresentano uno strumento fondamentale nell'analisi dei requisiti di un sistema *software*. 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*.

Ogni caso d'uso riporta:

- **Attore:** attori coinvolti nel caso d'uso;
- **Descrizione:** descrizione del caso d'uso;
- **Pre-condizione:** condizioni vere prima del verificarsi del caso d'uso;
- **Post-condizione:** condizioni vere dopo il verificarsi del caso d'uso;
- **Scenario:** sequenza specifica di eventi che si verifica quando un attore interagisce con il sistema per raggiungere un obiettivo;

Gli attori che ho identificato sono:

- **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.

Nelle sezioni seguenti, esamineremo in dettaglio i casi d'uso, che ho identificato durante lo studio del capitolato del progetto.

3.3.1.1 UC 1 - *Login*

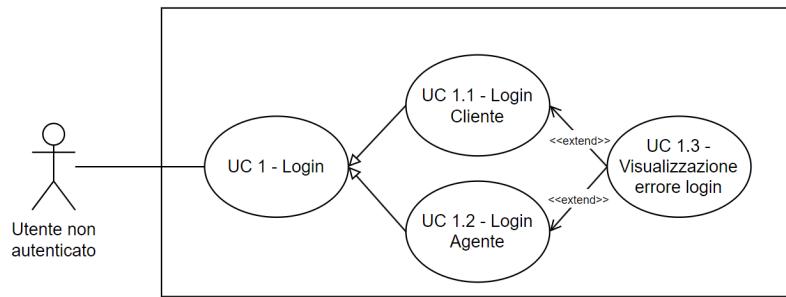


Figura 3.5: Use Case 1: *Login* Utente

UC 1: *Login* utente

Attori Principali: Utente non autenticato.

Descrizione: L'utente ha inserito *username* e *password* per autenticarsi nell'*app*.

Precondizioni: L'utente ha avviato l'*app*.

Postcondizioni: L'utente è autenticato nel sistema o ha ricevuto un errore.

Scenario:

- L'utente visualizza la schermata di *login* di MoviORDER;
- L'utente inserisce lo *username*;
- L'utente inserisce la *password*;
- L'utente preme il pulsante per effettuare il *login*;
- L'utente viene autenticato ed entra nell'applicazione.

UC 1.1: *Login* cliente

Attori Principali: Utente non autenticato.

Descrizione: L'utente è un cliente dell'azienda e vuole autenticarsi nell'*app*.

Precondizioni: L'utente ha avviato l'*app*.

Postcondizioni: L'utente è autenticato come cliente nel sistema o ha ricevuto un errore. Viene portato nella **Homepage** e nulla cambia rispetto la vecchia versione dell'*app*.

Scenario:

- L'utente visualizza la schermata di *login* di MoviORDER;
- L'utente inserisce lo *username*;
- L'utente inserisce la *password*;
- L'utente preme il pulsante per effettuare il *login*;
- L'utente viene autenticato e visualizza la **Homepage** dell'applicazione.

UC 1.2: *Login* agente

Attori Principali: Utente non autenticato.

Descrizione: L'utente è un agente dell'azienda e vuole autenticarsi nell'*app*.

Precondizioni: L'utente ha avviato l'*app*.

Postcondizioni: L'utente è autenticato come agente nel sistema o ha ricevuto un errore. Viene portato nella **Homepage Agenti**.

Scenario:

- L'utente visualizza la schermata di *login* di MoviORDER;
- L'utente inserisce lo *username*;
- L'utente inserisce la *password*;
- L'utente preme il pulsante per effettuare il *login*;
- L'utente viene autenticato e visualizza la **Homepage Agenti** dell'applicazione.

UC 1.3: Visualizzazione errore *login*

Attori Principali: Utente non autenticato.

Descrizione: L'utente vuole autenticarsi nell'app ma ha inserito *username* o *password* non validi.

Precondizioni: L'utente ha avviato l'operazione di *login*.

Postcondizioni: L'utente visualizza un messaggio che lo avvisa del fallimento dell'operazione di *login*.

Scenario:

- L'utente visualizza la schermata di *login* di MoviORDER;
- L'utente inserisce lo *username*;
- L'utente inserisce la *password*;
- L'utente preme il pulsante per effettuare il *login*;
- L'utente visualizza un errore.

3.3.1.2 UC 2 - Operazioni disponibili nella Homepage Agenti

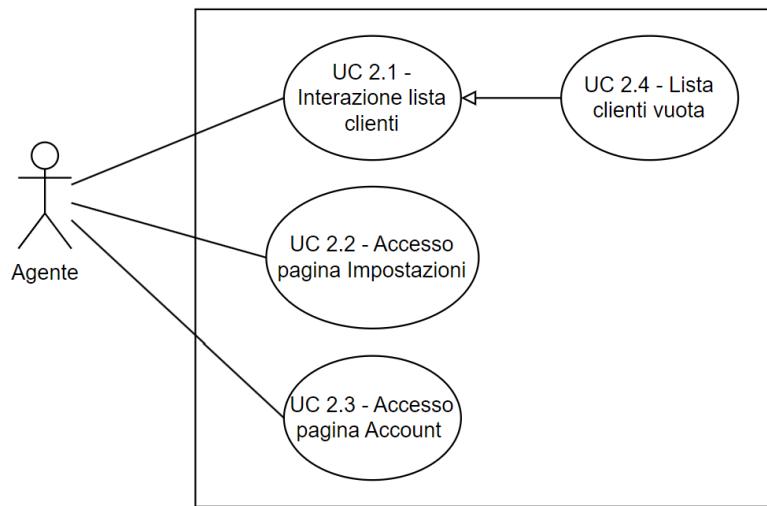


Figura 3.6: Use Case 2: Operazioni disponibili nella Homepage Agenti

UC 2.2: Accesso pagina Impostazioni

Attori Principali: Agente.

Descrizione: L'agente, cliccando nell'apposito menu il pulsante "Impostazioni", viene spostato nella pagina **Impostazioni**.

Precondizioni: L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la **Homepage Agenti**.

Postcondizioni: L'agente viene spostato nella pagina **Impostazioni**.

Scenario:

- L'agente visualizza la **Homepage Agenti**;
- L'agente visualizza il menu;
- L'agente clicca il pulsante "Impostazioni";
- L'agente viene spostato nella pagina **Impostazioni**.

UC 2.3: Accesso pagina Account

Attori Principali: Agente.

Descrizione: L'agente, cliccando nell'apposito menu il pulsante "*Account*", viene spostato nella pagina **Account**.

Precondizioni: L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la **Homepage Agenti**.

Postcondizioni: L'agente viene spostato nella pagina **Account**.

Scenario:

- L'agente visualizza la **Homepage Agenti**;
- L'agente visualizza il menu;
- L'agente clicca il pulsante "*Account*";
- L'agente si sposta nella pagina **Account**.

UC 2.4: Lista clienti vuota

Attori Principali: Agente.

Descrizione: L'agente che ha una lista clienti vuota visualizza il messaggio "nessun cliente trovato".

Precondizioni:

- L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la **Homepage Agenti**
- La lista clienti è vuota.

Postcondizioni: L'agente visualizza il messaggio "nessun cliente trovato".

Scenario:

- L'agente visualizza la **Homepage Agenti**
- La lista clienti è vuota;
- L'agente visualizza il messaggio "nessun cliente trovato" al posto della lista.

3.3.1.3 UC 2.1 - Interazioni con la lista clienti

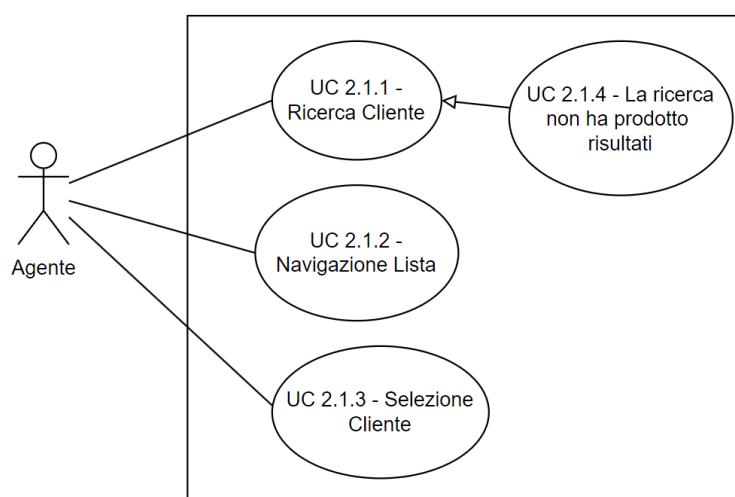


Figura 3.7: Use Case 2.1: Interazioni con la lista clienti

UC 2.1.1: Ricerca cliente

Attori Principali: Agente.

Descrizione: L'agente ricerca un cliente specifico all'interno della lista clienti.

Precondizioni:

- L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la Homepage Agenti;
- La lista clienti non è vuota.

Postcondizioni: L'agente visualizza il cliente ricercato.

Scenario:

- L'agente visualizza la Homepage Agenti;
- L'agente visualizza la lista clienti;
- L'agente ricerca un cliente nella lista clienti;
- L'agente visualizza il cliente ricercato.

UC 2.1.2: Navigazione lista

Attori Principali: Agente.

Descrizione: L'agente naviga la lista che contiene tutti i clienti dell'agente.

Precondizioni:

- L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la Homepage Agenti;
- La lista clienti non è vuota.

Postcondizioni: L'agente è riuscito a navigare nella lista.

Scenario:

- L'agente visualizza la Homepage Agenti;

- L'agente naviga la lista visualizzando i clienti contenuti.

UC 2.1.3: Selezione cliente

Attori Principali: Agente.

Descrizione: L'agente seleziona un cliente per operare nell'*app* come il cliente selezionato.

Precondizioni:

- L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la **Homepage Agenti**;
- La lista clienti non è vuota.

Postcondizioni: L'agente viene spostato nella **Homepage** e può operare nell'*app* come il cliente selezionato.

Scenario:

- L'agente visualizza la **Homepage Agenti**;
- L'agente visualizza la lista clienti;
- L'agente seleziona un cliente;
- L'agente viene spostato nella **Homepage**;
- L'agente può operare nell'*app* come il cliente selezionato.

UC 2.1.4: La ricerca non ha prodotto risultati

Attori Principali: Agente.

Descrizione: La ricerca di un cliente specifico all'interno della lista clienti non ha prodotto risultati.

Precondizioni:

CAPITOLO 3. STAGE

- L'utente si è autenticato con successo ed è stato riconosciuto come agente, quindi visualizza la **Homepage Agenti**;
- La lista clienti non è vuota;
- L'agente ricerca un cliente.

Postcondizioni: L'agente visualizza il messaggio "nessun cliente trovato".

Scenario:

- L'agente visualizza la **Homepage Agenti**;
- L'agente visualizza la lista clienti;
- L'agente ricerca un cliente nella lista clienti;
- La ricerca non produce risultati;
- L'agente visualizza il messaggio "nessun cliente trovato".

3.3.1.4 UC 3 - Operazioni disponibili nella Homepage Agenti autenticati come clienti

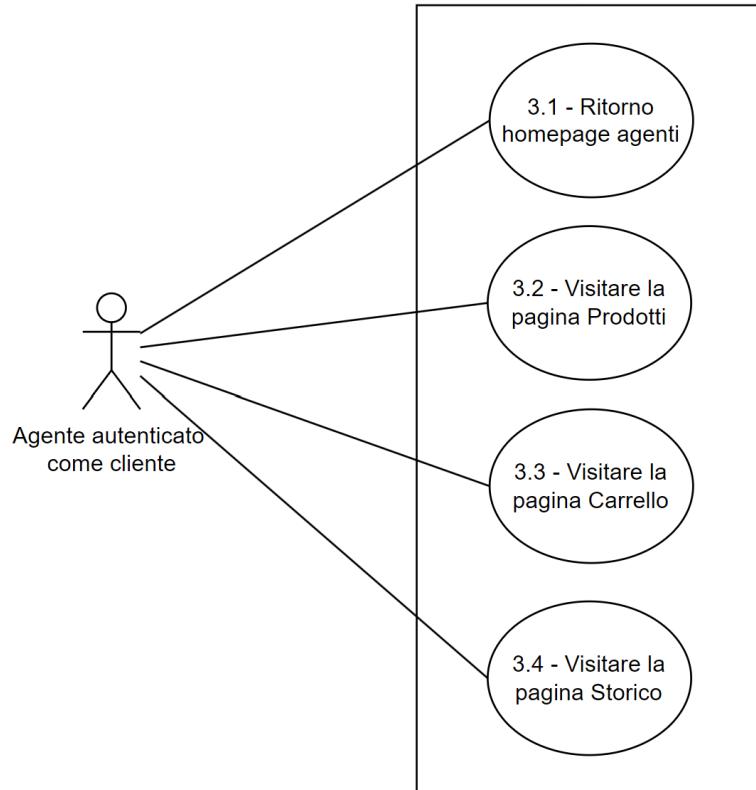


Figura 3.8: Use Case 3: Operazioni disponibili nella Homepage Agenti

UC 3.1: Ritorno Homepage Agenti

Attori Principali: Agente.

Descrizione: L'agente, cliccando nell'apposito menu il pulsante "*Homepage Agenti*", ritorna alla Homepage Agenti.

Precondizioni: L'agente ha selezionato un cliente dalla lista.

Postcondizioni: L'agente si è spostato nella Homepage Agenti.

Scenario:

- L'agente seleziona un cliente della lista;
- L'agente viene spostato nella Homepage;

- L'agente può operare nell'*app* come il cliente selezionato;
- L'agente preme il pulsante "*Homepage Agenti*";
- L'agente ritorna alla *Homepage Agenti*.

UC 3.2: Visitare la pagina Prodotti

Attori Principali: Agente.

Descrizione: L'agente, cliccando nell'apposito menu il pulsante "Prodotti", può operare nella pagina **Prodotti** come il cliente selezionato.

Precondizioni: L'agente ha selezionato un cliente dalla lista.

Postcondizioni: L'agente si è spostato nella pagina **Prodotti**.

Scenario:

- L'agente seleziona un cliente della lista;
- L'agente viene spostato nella *Homepage*;
- L'agente preme il pulsante "Prodotti";
- L'agente viene spostato nella pagina **Prodotti** dove può operare come il cliente selezionato.

UC 3.3: Visitare la pagina Carrello

Attori Principali: Agente.

Descrizione: L'agente, cliccando nell'apposito menu il pulsante "Carrello", può operare nella pagina **Carrello** come il cliente selezionato.

Precondizioni: L'agente ha selezionato un cliente dalla lista.

Postcondizioni: L'agente si è spostato nella pagina **Carrello**.

Scenario:

- L'agente seleziona un cliente della lista;
- L'agente viene spostato nella *Homepage*;

- L'agente preme il pulsante "Carrello";
- L'agente viene spostato nella pagina **Carrello** dove può operare come il cliente selezionato.

UC 3.4: Visitare la pagina **Storico**

Attori Principali: Agente.

Descrizione: L'agente, cliccando nell'apposito menu il pulsante "Storico", può operare nella pagina **Storico** come il cliente selezionato.

Precondizioni: L'agente ha selezionato un cliente dalla lista.

Postcondizioni: L'agente si è spostato nella pagina **Storico**.

Scenario:

- L'agente seleziona un cliente della lista;
- L'agente viene spostato nella **Homepage**;
- L'agente preme il pulsante "Storico";
- L'agente viene spostato nella pagina **Storico** dove può operare come il cliente selezionato.

3.3.1.5 UC 4 - Cambio tema

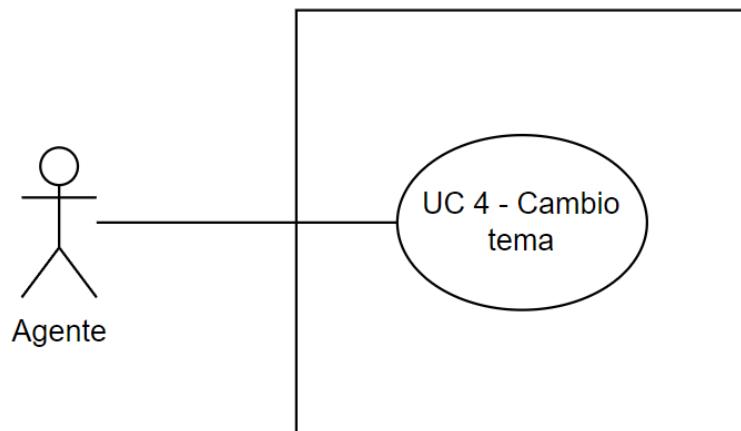


Figura 3.9: Use Case 4: Cambio tema

UC 4: Cambio tema

Attori Principali: Agente.

Descrizione: L'agente cambia il tema dell'applicazione in chiaro o scuro.

Precondizioni: L'agente, dalla Homepage Agenti, cliccando nell'apposito menu il pulsante "Impostazioni", si sposta nella pagina Impostazioni.

Postcondizioni: L'agente ha cambiato il tema dell'app.

Scenario:

- L'agente si trova nella Homepage Agenti;
- L'agente preme il pulsante "Impostazioni" dal menu;
- L'agente viene spostato nella pagina Impostazioni;
- L'agente cambia il tema dell'applicazione.

3.3.1.6 UC 5 - *Logout*

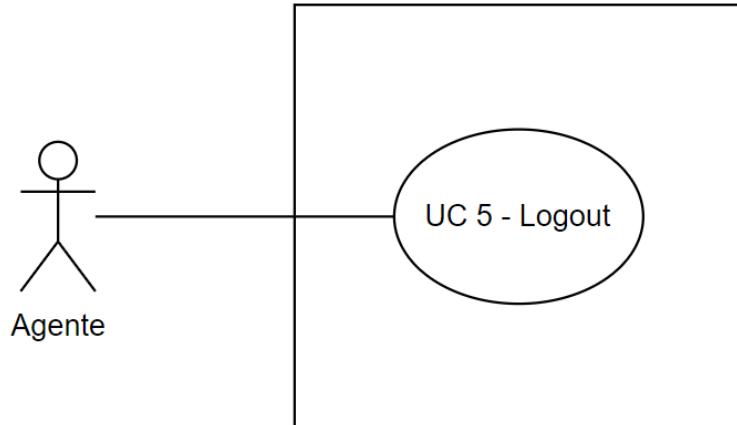


Figura 3.10: Use Case 5: *Logout*

UC 5: *Logout*

Attori Principali: Agente.

Descrizione: L'agente effettua la procedura di *logout*.

Precondizioni: L'agente, dalla **Homepage Agenti**, cliccando nell'apposito menu il pulsante "*Account*", si sposta nella pagina **Account**.

Postcondizioni: L'agente ha effettuato il *logout* e torna un utente non autenticato. L'agente viene spostato alla pagina di *login*.

Scenario:

- L'agente si trova nella **Homepage Agenti**;
- L'agente preme il pulsante "*Account*" dal menu;
- L'agente viene spostato nella pagina **Account**;
- L'agente effettua il *logout*;
- L'agente torna un utente non autenticato;
- L'agente viene spostato alla pagina di *login*

3.3.2 Requisiti funzionali e non funzionali

L'analisi dei casi d'uso rappresenta un punto di partenza fondamentale per 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 e nelle condizioni operative del sistema. 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 vengono riportate nella tabella indicano da dove ho ricavato il requisito descritto, principalmente troviamo:

- **UC:** i casi d'uso precedentemente descritti hanno portato alla definizione del requisito descritto;
- **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.

Requisito	Descrizione	Fonte
FO1	Un utente deve poter effettuare il <i>login</i> ed essere automaticamente riconosciuto come cliente	UC 1.1, Capitolato
FO1.1	Un cliente deve essere spostato nella Homepage in seguito al <i>login</i>	UC 1.1, Capitolato
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 1.2, Capitolato
FO2.1	Un agente deve essere spostato nella Homepage Agenti in seguito al <i>login</i>	UC 1.2, Studente
Continua nella prossima pagina...		

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Fonte
FO3	Un utente deve visualizzare un messaggio d'errore se le credenziali sono errate	UC 1.3, 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 2.1.1, <i>Tutor</i>
FO4.2	Un agente deve poter navigare la lista dei suoi clienti	UC 2.1.2, Capitolato
FO4.3	Un agente deve poter selezionare dalla lista uno dei suoi clienti	UC 2.1.3, Capitolato
FO4.4	Un agente deve visualizzare un messaggio d'errore che riporta la frase "nessun cliente trovato" se la lista clienti è vuota	UC 2.4, <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 2.1.4, <i>Tutor</i>
FO4.6	Un agente deve essere spostato nella pagina Homepage dopo aver selezionato un cliente dalla lista	UC 2.1.3, Studente
FO4.7	Un agente deve essere autenticato come il cliente selezionato dopo aver selezionato un cliente dalla lista	UC 2.1.3, Capitolato
FD5	Un agente deve poter visualizzare un menu nella Homepage Agenti	Studente
FD5.1	Un agente deve poter premere il pulsante "Impostazioni" dal menu nella Homepage Agenti	UC 2.2, Studente

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Fonte
FD5.2	Un agente deve essere spostato nella pagina Impostazioni dopo aver premuto il pulsante "Impostazioni"	UC 2.2, Studente
FD5.3	Un agente deve poter premere il pulsante " <i>Account</i> " dal menu nella Homepage Agenti	UC 2.3, Studente
FD5.4	Un agente deve essere spostato nella pagina Account dopo aver premuto il pulsante " <i>Account</i> "	UC 2.3, Studente
FO6	Un agente autenticato come cliente deve poter visualizzare un menu nella Homepage	Studente
FD6.1	Un agente autenticato come cliente deve poter ritornare alla Homepage Agenti premendo il pulsante " <i>Homepage Agenti</i> " nel menu della Homepage	3.1, Studente
FD6.2	Un agente autenticato come cliente deve essere spostato nella pagina Prodotti premendo il pulsante " <i>Prodotti</i> " nel menu della Homepage	3.2, Capitolo
FD6.2.1	Un agente autenticato come cliente deve poter operare come il cliente selezionato nella pagina Prodotti	3.2, Capitolo
FD6.3	Un agente autenticato come cliente deve essere spostato nella pagina Carrello premendo il pulsante " <i>Carrello</i> " nel menu della Homepage	3.3, Capitolo
Continua nella prossima pagina...		

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Fonte
FD6.3.1	Un agente autenticato come cliente deve poter operare come il cliente selezionato nella pagina Carrello	3.3, Capitolo
FD6.4	Un agente autenticato come cliente deve essere spostato nella pagina Storico premendo il pulsante "Storico" nel menu della Homepage	3.4, Capitolo
FD6.4.1	Un agente autenticato come cliente deve poter operare come il cliente selezionato nella pagina Storico	3.4, Capitolo
FD7.1	Un agente deve poter modificare il tema impostando il tema chiaro dalla pagina Impostazioni	UC 4, <i>Tutor</i>
FD7.2	Un agente deve poter modificare il tema impostando il tema scuro dalla pagina Impostazioni	UC 4, <i>Tutor</i>
FO7.3	Un agente non deve poter modificare la propria schermata d'avvio dalla pagina Impostazioni	Studente
FD8	Un agente deve poter effettuare il <i>logout</i> dalla pagina Account	UC 5, <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 Homepage Agenti	Capitolato, Studente
Continua nella prossima pagina...		

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Fonte
UO2	Un agente deve visualizzare il nome del cliente selezionato all'interno della schermata Homepage	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>
UD5	L'applicazione deve essere disponibile nella lingua inglese	<i>Tutor</i>
UD6	Deve essere possibile ricercare un cliente nelle liste 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</i> esistenti sviluppate in .NET	Capitolato, <i>Tutor</i>
Continua nella prossima pagina...		

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Fonte
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.

Ho qui riportato una tabella che riepiloga i requisiti precedentemente descritti.

Tipologia	Obbligatorio	Desiderabile	Totale
Funzionale	13	17	30
Di vincolo	3	0	3
Qualitativi	2	0	2
Usabilità	5	6	11
Totale	23	23	46

Tabella 3.2: Riepilogo requisti

3.4 Progettazione

Data la natura intrinseca del progetto, concepito come un'estensione del codice, non si è resa necessaria una fase di progettazione. L'approccio che ho adottato ha privilegiato una metodologia pragmatica, impegnandomi a rispettare le architettura preesistenti.

Per fare questo mi sono quindi impegnato per studiare a fondo l'architettura del sistema e i *pattern* che ho riportato nel capitolo 3.1 e le tecnologie per realizzare il modulo.

3.5 Sviluppo

3.5.1 Modifica del *database* e funzione Login

Terminato lo studio delle tecnologie e del codice di MoviORDER e completato il *setup* dell’ambiente di sviluppo, ho quindi iniziato la seconda fase dello *stage*, la modifica della *API_G* di autenticazione.

Innanzitutto, è stato necessario aggiungere una colonna alla tabella `User` del *common database*, che permetesse di distinguere i clienti dagli agenti. Nella tabella esisteva già un attributo chiamato `BPCode` (o *Business Partner Code*), ovvero la chiave che rappresenta un cliente all’interno della propria tabella del *company database*. Ho sfruttato il fatto che gli agenti possedessero un *id* nella loro tabella dedicata all’interno del *company database* chiamato `IdUser`, aggiungendo questa colonna anche in `User` come chiave esterna. In questo modo ho potuto identificare come clienti gli utenti con `BPCode` non nullo e `IdUser` nullo, e viceversa identificare gli utenti come agenti.

Per modificare il *database* ho utilizzato le potenzialità offerte da Entity Framework. In questo modo, mi è bastato modificare il modello delle *API_G* e applicare le modifiche lanciando gli appositi comandi da Visual Studio. Aggiunta la nuova colonna alla tabella, ho provveduto a modificare un utente del *database*, assegnandogli un `IdUser` compatibile con quelli riportati nel *company database*, in modo da avere un agente con cui testare le nuove funzionalità.

Il passaggio successivo è stato modificare il servizio `UserService` al cui interno è definita la logica utilizzata dal *API_G controller* che si occupa del *login*. Inizialmente, ho nuovamente modificato il modello dei dati rendendo il `BPCode` della tabella `User` *nullable* (cioè in grado di ammettere valori nulli). Successivamente, ho modificato la funzione `Login` e le relative funzioni di supporto, aggiungendo una serie di controlli di validità. Come ultima cosa ho testato manualmente il corretto funzionamento della *API_G* e la corretta integrazione delle modifiche nel *database* utilizzando Swagger.

3.5.2 Sviluppo del modulo agenti

3.5.2.1 *API_G* GetCustomerList

In questa terza fase mi sono impegnato a raggiungere due obiettivi: lo sviluppo dell'*API_G* per recuperare i clienti degli agenti e la creazione di una *view* che chiamerò *Homepage Agenti*.

La *API_G* che ho sviluppato per il modulo agenti si chiama `GetCustomerList` e richiama all'interno del suo *controller* la funzione chiamata `GetAgentCustomers` definita nel servizio `BpService.cs`. Il suo funzionamento è semplice ed è il risultato finale di un processo di miglioramento continuo, in cui ho affinato il codice man mano che acquisivo maggiore familiarità con esso. Pertanto, descriverò solo il risultato finale, senza entrare nei dettagli dell'evoluzione e del ragionamento dietro lo sviluppo.

Una particolarità da considerare è la gestione del *database* di MoviORDER: il *company database* è in realtà condiviso con un'altra applicazione, MoviSELL.

La gestione e la modifica di quel *database* sono quindi responsabilità di MoviSELL, che li ha creati e li gestisce grazie agli strumenti di Entity Framework.

Per introdurre una tabella del *company database* nel modello di MoviORDER, ho dovuto lavorare al contrario, utilizzando il comando `scaffold` di Entity Framework per costruire il modello a partire da una tabella esistente.

La tabella che avevo necessità di introdurre è `Bp`, ovvero la tabella che contiene le informazioni relative ai clienti aziendali. Ho utilizzato le informazioni di questa tabella anche nella funzione `Login` per controllare la validità del `BPCode` del cliente. Per rendere più efficiente la funzione, ho aggiunto un ulteriore valore solo per gli agenti al *token* di autenticazione. Questo contiene al suo interno, critte, due informazioni utili per le funzioni della *Business Logic*: `CompanyCode` e `Username` a cui ho aggiunto il nuovo valore, l'identificativo degli agenti `IdUser`, evitando di doverlo recuperare nella funzione.

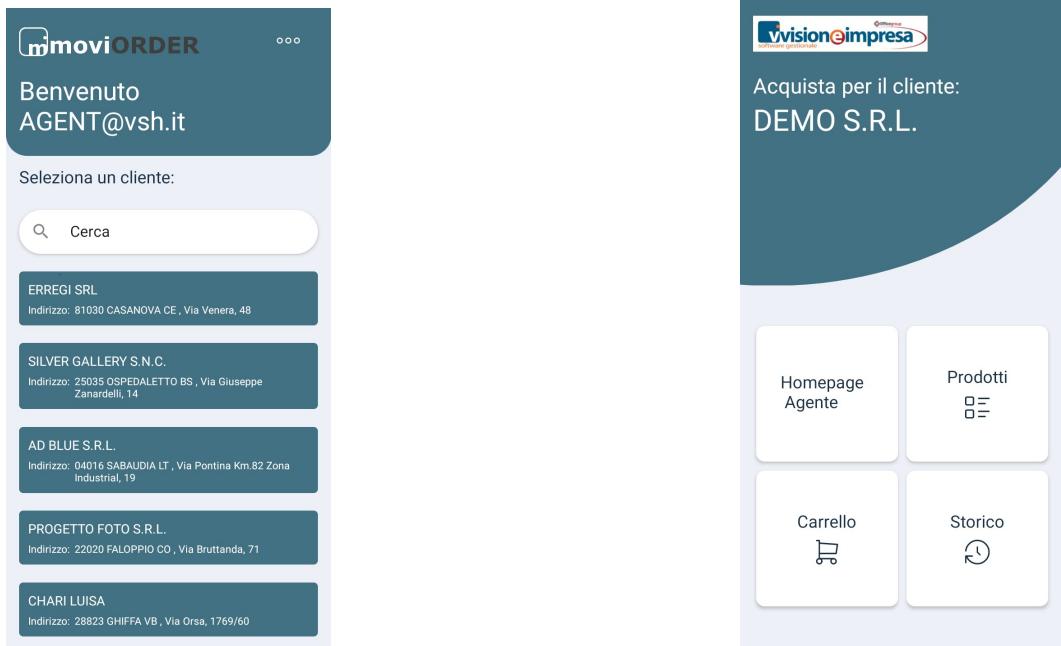
Infine, ho creato un DTO chiamato `BpCustomerDto` per ritornare i dati al *controller* della *API_G*.

La funzione `GetAgentCustomers` recupera due liste: quella degli utenti con

CAPITOLO 3. STAGE

CompanyCode uguale a quello dell'agente e IdUser nullo, e quella dei clienti con campo IdUser uguale a quello dell'agente (che significa in questo caso che il cliente è assegnato all'agente). Quindi, creo una terza lista, di BpCustomerDto, che contiene tutti i clienti appartenenti ad entrambe le liste. Questo serve ad evitare che clienti non ancora registrati nell'applicazione appaiano nella lista dei clienti selezionabili, oltre al fatto che sono necessari i dati presenti in entrambe le tabelle per il corretto funzionamento del modulo (lo *username* è contenuto in User, mentre le informazioni come l'indirizzo del cliente sono contenute in Bp).

3.5.2.2 *Homepages*



(a) Homepage Agenti definitiva di MoviORDER

(b) Homepage definitiva di MoviORDER

Figura 3.11: Versione finale delle *homepages* di MoviORDER

La nuova *view* che ho creato, chiamata *Homepage Agenti*, permette agli agenti di visualizzare e selezionare i propri clienti. Questa *view* è stata sviluppata per richiamare l'*API G GetCustomerList* precedentemente descritta, che consente di recuperare la lista dei clienti associati a un determinato agente.

A differenza dei clienti, che vengono indirizzati direttamente alla *Homepage* dopo il *login*, gli agenti vengono invece reindirizzati alla *Homepage Agenti*. Questa

view agisce come una "camera stagna", in cui l'agente rimane confinato fino a quando non seleziona un cliente con cui operare all'interno dell'*app*. Questo approccio è stato adottato per evitare che gli agenti possano cambiare la schermata senza aver selezionato un cliente e ridurre quindi la possibilità di comportamenti anomali. Per questo, a differenza dei clienti, ho inabilitato la possibilità di cambiare schermata di avvio dell'*app* dalla pagina **Impostazioni**. La figura 3.11 mostra le due *homepage* nella loro versione finale.

Ho iniziato lo sviluppo costruendo la *view* e i *component* per la lista dei clienti, in modo da poter testare il corretto funzionamento della **API_G**. In questa fase iniziale, l'aspetto della *view* era piuttosto grezzo e poco curato dal punto di vista estetico. Successivamente, ho modificato lo *stack* di Navigation per rendere la *view* visitabile, e ho aggiunto una serie di attributi specifici per gli agenti nello *store* (senza tuttavia approfondirli in questa sede, in quanto subiranno modifiche più avanti nel processo di sviluppo).

Infine, ho: creato l'*endpoint* per richiamare la **API_G** `GetCustomerList`, definito un nuovo tipo chiamato **CustomerModel** per rappresentare i clienti che costituiranno la lista, la funzione definita nel modulo **Controllers** per effettuare la chiamata all'**API_G** e i *reducer* dello **Store**, ovvero le specifiche funzioni di Redux utilizzate per modificare lo stato dell'*app*, in modo da richiamare la funzione del *controller* in specifici momenti e in specifiche condizioni (solo se l'utente è un agente e solo dopo il *login* o dopo l'apertura dell'*app*).

3.5.2.3 **API_G** AdditionalLogin

La prima sfida che ho dovuto affrontare è stata quella di autenticare l'agente come il cliente selezionato. Inizialmente avevo pensato di gestire questo aspetto direttamente dal *front-end*, salvando il **BPCode** del cliente nello *store* e utilizzandolo come parametro da passare alle **API_G**. Tuttavia, questo approccio avrebbe comportato la modifica di tutte le **API_G** che recuperano il valore del **BPCode** usando lo **Username** contenuto nel *token*.

Confrontandomi con gli sviluppatori di VisioneImpresa a riguardo, siamo giunti alla conclusione che questa sarebbe stata una scelta implementativa pessima, in quanto avrebbe compromesso l'integrità e la coerenza dell'intera architettura

dell'applicazione.

Abbiamo quindi ideato una soluzione molto più semplice e pulita per risolvere il problema. Questa soluzione si basa sul fatto di effettuare un nuovo *login* ogni volta che l'agente seleziona un cliente, utilizzando però una *API_G* differente rispetto a *Login* (quella normalmente utilizzata per l'autenticazione) chiamata *AdditionalLogin*.

Questa *API_G* funziona in maniera analoga a *Login*, ma salta tutti i controlli sulla *password*. La sicurezza di questa *API_G* è garantita dal fatto che per essere richiamata richiede il *token* di autenticazione, al contrario di *Login*, garantendo che l'utente che la richiama sia un utente autenticato di MoviORDER.

Questa funzione è stata sviluppata all'interno del servizio *UserService*, dove è definita anche *Login*, e anche per lei è stato definito un *API_G Controller*. Analogamente a *GetCustomerList*, nel *front-end* ho definito la funzione che richiama l'*API_G* all'interno del modulo *Controllers* e gli *endpoint* nel modulo *Services*.

Questo approccio funziona in quanto l'*API_G AdditionalLogin* ritorna gli stessi attributi restituiti da *Login*, incluso un nuovo *token* di autenticazione. Creando le apposite variabili di supporto nello *store* posso quindi salvare i dati del cliente e utilizzarli nell'applicazione quando necessario.

Per quanto riguarda la gestione dei *token*, ho adottato un approccio che prevede il salvataggio di due *token* distinti: il *token* utente, che viene salvato nella memoria del dispositivo in seguito alla *login*, e il nuovo *token* ottenuto dalla chiamata a *AdditionalLogin*, che anch'esso viene conservato nella memoria del dispositivo.

Ho modificato la funzione che restituisce il *token* in modo che, finché l'agente non seleziona un cliente, essa utilizzi il *token* dell'agente. Una volta selezionato un cliente, la funzione restituirà invece il secondo *token*, ovvero quello ottenuto dalla chiamata a *AdditionalLogin* e associato al cliente selezionato.

In questo modo, quando verranno chiamate le *API_G*, verrà fornito il *token* creato utilizzando le credenziali del cliente selezionato, consentendo all'agente di utilizzare l'*app* come se fosse il cliente stesso. Questa soluzione garantisce la coerenza e l'integrità dell'applicazione, evitando di dover modificare le *API_G*.

esistenti.

3.5.3 Modifica delle interfacce



(a) Homepage Agenti definitiva
di MoviORDER per *tablet*



(b) Ricerca clienti con la
search bar

In questa quarta fase, mi sono concentrato sulla parte grafica dell'applicazione, creando tutti i *component* necessari (come la *search bar* mostrata dalla figura 3.12b) e definendo lo stile mediante l'utilizzo di CSS.

Per rendere i *component* modulari ed indipendenti, facilitandone il riutilizzo in altre parti dell'applicazione, ho spostato tutta la logica di funzionamento nella *view Homepage Agenti*. In questo modo, la responsabilità di ciascun *component* è ben definita e separata, migliorando la manutenibilità e la scalabilità del codice.

Dopo aver completato questa ristrutturazione, ho apportato le ultime migliorie all'applicazione, come la possibilità di cambiare il tema della *Homepage Agenti* in accordo con il resto dell'applicazione. Successivamente ho adeguato il CSS della *view* e dei *component* per garantire una corretta visualizzazione anche in modalità *tablet*, come mostrato dalla figura 3.12a.

Infine, ho aggiornato i file .json che contengono la definizione delle frasi in inglese e italiano, in modo da rendere il modulo agenti disponibile in entrambe le lingue. Questa modifica assicura una maggiore accessibilità e inclusività dell'applicazione per gli utenti di diverse nazionalità.

3.6 Verifica

Nella penultima fase del tirocinio mi sono concentrato nel *testing* dell'applicazione. VisioneImpresa non utilizza strumenti per la verifica automatica del codice, e ho quindi testato manualmente tutte le componenti del sistema 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 le funzioni della *business logic* delle *API* utilizzando il *framework* Moq.

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 chiamati sui *mock***, verificando che il codice sotto *test* interagisca correttamente con i componenti dipendenti.
- **Restituire valori predefiniti o simulare comportamenti complessi sui *mock***, in modo da poter testare diverse casistiche.

Con l'utilizzo di Moq, ho potuto creare *mock* dei servizi e dei *repository* utilizzati dalle *API*, simulando il comportamento dei componenti dipendenti senza dover effettivamente invocarli. Questo mi ha permesso di testare in modo isolato le singole funzioni della *Business Logic*, verificandone il corretto funzionamento 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, dovuta alla natura complessa degli oggetti

utilizzati dalla logica di *business* delle *API* che stavo testando. 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 problemi di funzionamento, nonostante i vincoli tecnici incontrati.

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

L'adozione di questa pratica di *testing*, sebbene non fosse inizialmente prevista dal processo di sviluppo dell'azienda, è stata molto apprezzata dal *tutor* aziendale. Egli ha riconosciuto il valore aggiunto che questa scelta ha comportato in termini di qualità e robustezza del prodotto.

3.7 Risultati

Alla conclusione del mio periodo di tirocinio, ho presentato il mio lavoro al *tutor* aziendale e al *team* di sviluppatori di VisioneImpresa durante un *meeting* finale. Ho descritto nel dettaglio tutte le componenti del modulo e come queste si integrassero tra loro, mostrando l'applicazione funzionante. Infine, ho presentato gli *unit test* che avevo implementato utilizzando il *framework* Moq, come ho descritto nel capitolo precedente. Al termine della presentazione, il *tutor* aziendale e il *team* di sviluppatori hanno espresso il loro pieno soddisfacimento per il lavoro svolto, confermando che tutti gli obiettivi che ho riportato in 3.1 erano stati raggiunti con successo.

MoviORDER sta venendo completamente riscritta per arrivare in futuro ad unire le due applicazioni MoviORDER e MoviSELL in un solo *software*, per questo il *team* di VisioneImpresa ha sottolineato l'importanza del mio contributo, in quanto costituirà una solida base per le future evoluzioni del sistema.

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]. [vi](#), [11](#), [13](#), [14](#), [18–23](#), [27](#), [28](#), [33](#), [36–39](#), [58](#), [60–64](#), [67](#)

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).

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

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).

CAPITOLO 3. SITOGRAFIA

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