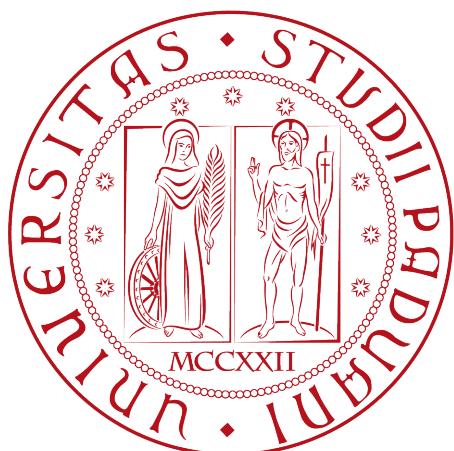


Università degli Studi di Padova
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
CORSO DI LAUREA IN INFORMATICA



**MoviORDER: una piattaforma mobile per la
creazione e l'invio di ordini**

Tesi di laurea triennale

Relatore

Prof. Armir Bujari

Laureando

Tommaso Carraro

ANNO ACCADEMICO 2017-2018

Tommaso Carraro: *MoviORDER: una piattaforma mobile per la creazione e l'invio di ordini*, Tesi di laurea triennale, © Dicembre 2018.

LOREM IPSUM dolor sit amet, consectetuer adipiscing elit.

— Oscar Wilde

Dedicato a ...

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di *stage*, della durata di 320 ore, dal laureando Tommaso Carraro presso l'azienda VisioneImpresa S.r.l. situata a Pernumia (PD).

Lo scopo principale del progetto era la realizzazione di un'applicazione *mobile* che permettesse ai clienti di una qualsiasi azienda di acquistare, tramite il proprio *smartphone*, dei prodotti venduti dalla stessa. Per raggiungere questo fine sono stati assegnati vari compiti.

Lo studente ha dovuto scegliere in autonomia l'ambiente di sviluppo ritenuto più opportuno. Poiché era richiesto che l'applicazione funzionasse sia in ambiente *Android* che in ambiente *iOS*, si è dovuto scegliere un *framework cross-platfrom_G* e, in particolare, il *framework_G PhoneGap*.

In seguito alla scelta del *framework* vi è stato un periodo di formazione, di circa 40 ore, sui *software gestionali_G* utilizzati in azienda e sul linguaggio *JavaScript*, in modo da facilitare lo sviluppo del progetto.

Si è poi potuto proseguire con la progettazione delle varie componenti della *piattaformag*, quali *servizio web_G*, *database* sottostanti, logica applicativa e interfaccia grafica. L'applicazione doveva funzionare interamente *online*, nessun dato doveva essere memorizzato in locale, per cui il servizio *web* e i *database* sono stati installati su un *server Azure_G* di proprietà dell'azienda. In particolare, è stato richiesto di progettare un *database* che permettesse la gestione dei dati di autenticazione e un *database* che contenesse i dati utili alla gestione degli ordini presso un'azienda cliente. Il *database* contenente i dati proprietari dell'azienda poteva essere locale al *server Azure* o all'interno di un *server cloud* dell'azienda stessa, a seconda delle scelte effettuate da quest'ultima.

In seguito alla progettazione delle varie parti si è iniziata l'implementazione della piattaforma. Il servizio doveva gestire le richieste *HTTP_G* (*HyperText Transfer Protocol*) provenienti dall'applicazione tramite oggetti *servlet_G Java* e rispondere a queste mediante stringhe in formato *JSON_G* (*JavaScript Object Notation*). La logica applicativa dell'applicazione doveva essere scritta in linguaggio *JavaScript*, questo perché lo stagista ha scelto il *framework PhoneGap*. Per permettere all'applicazione di comunicare con il servizio *web* tramite richieste *HTTP*, si è dovuta utilizzare la tecnica *AJAX_G* (*Asynchronous JavaScript And XML*). Il *design* dell'interfaccia doveva essere simile al *design* di un'altra applicazione sviluppata dall'azienda, chiamata *moviDOC_G*. Infine, per permettere all'applicazione di essere *usabile_G* dalla maggior parte dei dispositivi, si è dovuto rendere il *design* della stessa *responsive_G*.

“Life is really simple, but we insist on making it complicated”

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. NomeDelProfessore, relatore della mia tesi, per l'aiuto e il sostegno fornитоми durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Dicembre 2018

Tommaso Carraro

Indice

1	Introduzione	1
1.1	L'azienda	1
1.1.1	Core business	2
1.2	Offerta di stage	3
1.3	Obiettivi e pianificazione	5
1.4	Rischi	7
1.5	Organizzazione del testo	8
2	Processo di sviluppo	9
2.1	Modello incrementale	9
3	Background tecnologico	11
3.1	Framework	11
3.1.1	Motivazioni alla base dei framework cross-platform	11
3.1.2	Approcci alla base dei framework cross-platform	13
3.1.3	Xamarin	15
3.1.4	PhoneGap	15
3.1.5	La scelta di PhoneGap	17
3.2	Ambiente di sviluppo	17
3.2.1	Suite di PhoneGap	17
3.2.2	Editor e IDE	19
3.2.3	Gestione DBMS	20
3.2.4	Server web	22
3.2.5	Cloud computing	22
3.2.6	Strumenti di testing	22
3.2.7	Strumenti di versioning e ticketing	24
3.2.8	Strumenti di modellazione e documentazione	24
3.2.9	Linguaggi di programmazione e markup	25
3.2.10	DBMS	27
4	Analisi dei requisiti	29
4.1	Casi d'uso	29
4.1.1	Attori del sistema	30
4.1.2	UC1 - Azioni utente non autenticato	30
4.1.3	UC1.1 - Autenticazione	31
4.1.4	UC1.1.1 - Inserimento username	32
4.1.5	UC1.1.2 - Inserimento password	32
4.1.6	UC1.2 - Visualizzazione errore di autenticazione	32

4.1.7	UC2 - Azioni utente autenticato	33
4.1.8	UC2.1 - Logout	33
4.1.9	UC2.2 - Visualizzazione tutorial applicazione	34
4.1.10	UC2.3 - Gestione carrello	35
4.1.11	UC2.3.1 - Aggiunta articolo	37
4.1.12	UC2.3.1.1 - Inserimento codice articolo	37
4.1.13	UC2.3.1.2 - Inserimento manuale codice articolo	38
4.1.14	UC2.3.1.3 - Inserimento codice articolo con scansione codice a barre	38
4.1.15	UC2.3.1.4 - Inserimento quantità articolo	38
4.1.16	UC2.3.1.5 - Inserimento note articolo	39
4.1.17	UC2.3.2 - Visualizzazione errore di inserimento articolo	39
4.1.18	UC2.3.3 - Modifica articolo	40
4.1.19	UC2.3.3.1 - Modifica quantità articolo	40
4.1.20	UC2.3.3.2 - Modifica note articolo	41
4.1.21	UC2.3.4 - Visualizzazione errore di modifica articolo	41
4.1.22	UC2.3.5 - Selezione singolo articolo	41
4.1.23	UC2.3.6 - Deselezione singolo articolo	41
4.1.24	UC2.3.7 - Selezione totale di articoli	42
4.1.25	UC2.3.8 - Deselezione totale di articoli	42
4.1.26	UC2.3.9 - Rimozione articoli selezionati	42
4.1.27	UC2.3.10 - Invio ordine	43
4.1.28	UC2.3.10.1 - Modifica data ordine proposta	43
4.1.29	UC2.3.10.2 - Inserimento note ordine	44
4.1.30	UC2.3.11 - Visualizzazione errore di mancata selezione di articoli	44
4.2	Requisiti	45
4.2.1	Requisiti funzionali	46
4.2.2	Requisiti qualitativi	56
4.2.3	Requisiti di vincolo	57
4.2.4	Riepilogo requisiti	58
4.2.5	Validazione dei requisiti	58
5	Progettazione	59
5.1	Architettura generale	59
5.1.1	Architettura front end	61
5.1.2	Architettura back end	62
5.2	Progettazione servizio web	64
5.2.1	Package servlet	65
5.3	Progettazione database	65
6	Codifica	71
6.1	Servizio web	71
6.1.1	Servlet	71
6.1.2	JDBC	80
6.1.3	Classi utilità	82
6.2	Logica applicativa	84
6.2.1	Plugin di PhoneGap	84
6.2.2	Installazione dei plugin	85
6.2.3	Premesse all'utilizzo dei plugin	85
6.2.4	Plugin utilizzati	86

6.3	Interfaccia grafica	91
6.3.1	Schermata di login	91
6.3.2	Home page	92
6.3.3	Modal di aggiunta articolo	93
6.3.4	Pagina di aggiunta articolo	94
6.3.5	Pagina di modifica articolo	95
6.3.6	Modal di invio ordine	95
6.3.7	Considerazioni sullo sviluppo	96
7	Verifica e validazione	99
7.1	Verifica	99
7.1.1	Analisi statica	99
7.1.2	Analisi dinamica	100
7.2	Validazione	101
7.2.1	Validazione di moviORDER	101
8	Conclusioni	103
8.1	Consuntivo finale	103
8.2	Grado di soddisfacimento dei requisiti	104
8.3	Conoscenze acquisite	105
8.3.1	JavaScript	105
8.3.2	SQL Server	105
8.3.3	Apache Tomcat e oggetti servlet	105
8.3.4	PhoneGap	106
8.4	Sviluppi futuri	106
8.5	Valutazione personale	106
A	Glossario	109
B	Convenzioni	111
B.1	Casi d'uso	111
B.2	Requisiti	111
	Bibliografia	115

Elenco delle figure

1.1	Logo dell'azienda VisioneImpresa	2
1.2	Banner del <i>software VisionENTERPRISE</i>	2
1.3	Logo del marchio <i>movidat</i>	3
1.4	Logo dell'evento <i>Stage-IT</i>	3
1.5	Diagramma di <i>Gantt</i> della pianificazione delle attività di <i>stage</i>	6
2.1	Modello di ciclo di vita incrementale	10
3.1	Frammentazione SO del mercato italiano nel 2016	12
3.2	Significato di <i>cross-platform</i>	12
3.3	Architettura di un'applicazione ibrida	13
3.4	Architettura di un'applicazione interpretata	14
3.5	Architettura del <i>framework Xamarin</i>	15
3.6	Architettura del <i>framework PhoneGap</i>	16
3.7	Figura illustrativa di <i>PhoneGap Build</i>	16
3.8	<i>PhoneGap Desktop App</i>	18
3.9	<i>PhoneGap CLI e PhoneGap App</i>	19
3.10	Logo di <i>Sublime Text 3.0</i>	19
3.11	Logo di <i>Android Studio</i>	20
3.12	Logo di <i>XCode</i>	20
3.13	Logo di <i>Eclipse</i>	21
3.14	<i>Screenshot</i> di <i>SQL Server Management Studio</i>	21
3.15	Logo di <i>Apache Tomcat</i>	22
3.16	Logo di <i>Microsoft Azure</i>	22
3.17	Logo di <i>Postman</i>	23
3.18	<i>Screenshot</i> della console di <i>Google Chrome</i>	23
3.19	Logo di <i>GitHub</i>	24
3.20	Logo di <i>Asana</i>	24
3.21	Loghi di <i>Gantt Project</i> , <i>Astah UML</i> e <i>Visual Paradigm CE</i>	25
3.22	Loghi di <i>Lucidchart</i> e <i>TexMaker</i>	25
3.23	Loghi di <i>HTML5</i> , <i>CSS3</i> e <i>JavaScript</i>	26
3.24	Logo di <i>Java</i>	27
3.25	Logo di <i>LATEX</i>	27
3.26	Logo di <i>SQL Server</i>	28
4.1	Use Case - UC1: Azioni utente non autenticato	30
4.2	Use Case - UC1.1: Autenticazione	31
4.3	Use Case - UC2: Azioni utente autenticato	33

4.4	Use Case - UC2.3: Gestione carrello	35
4.5	Use Case - UC2.3.1: Aggiunta articolo	37
4.6	Use Case - UC2.3.3: Modifica articolo	40
4.7	Use Case - UC2.3.10: Invio ordine	43
5.1	Architettura generale di <i>moviORDER</i>	60
5.2	Differenze tra pattern <i>MVC</i> e <i>MVP</i>	61
5.3	Architettura <i>front end</i>	61
5.4	Diagramma di sequenza del pattern <i>MVP</i>	62
5.5	Architettura a strati del <i>back end</i>	63
5.6	Diagramma dei <i>package</i> del servizio <i>web</i>	64
5.7	Diagramma delle classi del <i>package servlet</i>	65
5.8	Diagramma <i>ER</i> del <i>database CommonDb</i>	66
5.9	Diagramma <i>ER</i> del <i>database mvo_aziendaNomeAzienda</i>	69
6.1	Metodo <i>doPost()</i> del <i>servlet</i> che gestisce l'autenticazione	72
6.2	Esempio di invio di una richiesta <i>HTTP</i> tramite <i>AJAX</i>	75
6.3	Metodo costruttore della classe <i>DatabaseConnection</i>	81
6.4	Metodo <i>connectToDb()</i> della classe <i>DatabaseConnection</i>	82
6.5	Metodo <i>sendMail()</i> della classe <i>MailUtility</i>	84
6.6	Esempio di codice <i>JavaScript</i> che attende l'evento <i>deviceready</i>	85
6.7	Esempio di codice <i>JavaScript</i> che non utilizza il <i>plugin dialogs</i>	86
6.8	Esempio di visualizzazione scorretta - <i>alert</i>	87
6.9	Esempio di codice <i>JavaScript</i> che utilizza il <i>plugin dialogs</i>	87
6.10	Esempio di visualizzazione corretta - <i>dialog</i>	87
6.11	Esempio di utilizzo della proprietà <i>type</i>	88
6.12	Esempio di utilizzo dell'evento <i>offline</i>	88
6.13	Codice <i>Objective-C++</i> per il settaggio del processo di scansione	90
6.14	Codice <i>JavaScript</i> che utilizza il <i>plugin barcode scanner</i>	90
6.15	Schermata di login	91
6.16	<i>Home page</i>	92
6.17	<i>Modal</i> di aggiunta articolo e modalità di aggiunta (scansione o inserimento manuale)	93
6.18	Pagina di aggiunta articolo	94
6.19	<i>Modal</i> di invio ordine	95
6.20	<i>Modal</i> di invio ordine	96
6.21	Codice <i>css</i> che utilizza unità di misura relative	97

Elenco delle tabelle

1.1	Pianificazione oraria del periodo di <i>stage</i>	6
1.2	Analisi dei rischi	7
4.1	Tabella del tracciamento dei requisiti funzionali	55
4.2	Tabella del tracciamento dei requisiti qualitativi	56
4.3	Tabella del tracciamento dei requisiti di vincolo	57
4.4	Riepilogo requisiti	58
8.1	Consuntivo finale	104
8.2	Grado di soddisfacimento dei requisiti	105

Capitolo 1

Introduzione

Al giorno d'oggi la tecnologia sta ricoprendo un ruolo importante nei *task* di tutti i giorni. In particolare, è indispensabile che un'azienda venditrice di un qualsiasi tipo di prodotti abbia una propria piattaforma *online* per la gestione degli ordini. Questo perché le persone sono sempre più abituate ad effettuare ordini *online*. Infatti, acquistare *online* risulta vantaggioso per molteplici motivi, quali il risparmio di tempo e denaro ma soprattutto la comodità di non doversi muovere da casa per acquistare un prodotto. Per molte aziende non cogliere questo cambiamento potrebbe essere fallimentare, infatti, in futuro, la maggior parte degli acquisti avverrà principalmente *online* per qualsiasi tipologia di prodotto.

MoviORDER nasce per rispondere a questa necessità, proponendosi come piattaforma universale per la registrazione e l'invio di ordini *online*. Una qualsiasi azienda interessata a vendere i propri prodotti *online* può contattare VisioneImpresa per ricevere *moviORDER* e, dopo un breve periodo di configurazione, l'applicazione sarà pronta a ricevere ordini dagli utenti registrati. *MoviORDER* risulta vantaggiosa per i seguenti motivi:

* per i clienti:

- possibilità di trovare i prodotti *online* e non solamente nel negozio fisico;
- possibilità di risparmiare tempo e denaro dovuti al raggiungimento del negozio fisico;
- possibilità di controllare in tempo reale la disponibilità dei prodotti.

* per le aziende:

- riduce il rischio di fallimento dovuto al cambiamento delle convenzioni degli utenti;
- aumenta il numero di clienti per l'azienda, permettendo a coloro che non riescono a raggiungere il negozio fisico di poter comunque effettuare ordini grazie alla piattaforma *online*.

1.1 L'azienda

VisioneImpresa è un'azienda che da 30 anni si occupa di informatica e più precisamente di quella parte dell'informatica dedicata alle applicazioni gestionali. Inizialmente l'attività di VisioneImpresa era rivolta ad aziende, enti pubblici, studi professionali e centri

di elaborazione dati, gestendo totalmente problematiche informatiche, progettazione di sistemi, *hardware*, reti, sistemi operativi e *software* applicativo. Oggi VisioneImpresa punta sulla specializzazione, dedicandosi in modo particolare allo sviluppo del *software* applicativo e dei relativi servizi di implementazione dello stesso nell'azienda. VisioneImpresa è un *team* di persone esperte e motivate che opera direttamente su gran parte del Nord Est e indirettamente sull'intero territorio nazionale. VisioneImpresa si rivolge a piccole e medie aziende italiane che intendono impostare sul sistema informatico non solo la semplice gestione amministrativa o di magazzino, ma la completa organizzazione aziendale per affrontare un futuro sempre più complesso e veloce, con il supporto di un sistema informatico che aiuti l'azienda a prendere decisioni sempre basate su dati precisi.



Figura 1.1: Logo dell'azienda VisioneImpresa

1.1.1 Core business

VisioneImpresa presenta principalmente due *core business*: *VisionENTERPRISE* e *movidat*. *VisionENTERPRISE* è un *software gestionale* ERP_G dedicato alle medie e piccole aziende industriali, commerciali e dei servizi, che gestiscono notevoli moli di dati e hanno la necessità di lavorare con grande velocità e stabilità. Il *software* è in grado di collegarsi a tutte le informazioni dell'azienda cliente e di interfacciarsi con tutti i *software* utilizzati al fine di gestire in modo ottimale l'intera organizzazione aziendale con la massima semplicità e velocità operativa. Il *software* è in grado di rendere disponibile in tempo reale, alla direzione o al titolare dell'azienda, tutte le informazioni di cui hanno bisogno per prendere decisioni sulla base di dati concreti e oggettivi. Altra qualità di *VisionENTERPRISE* è la sua completa copertura funzionale: dalla contabilità al magazzino, dall'area commerciale alla produzione, dal controllo di gestione all'*E-business*.



Figura 1.2: Banner del *software* *VisionENTERPRISE*

Movidat è un marchio di VisioneImpresa che progetta e sviluppa applicazioni per dispositivi *mobile* rivolte alle piccole e medie imprese che vogliono rendere i loro *processi_G* più semplici, veloci ed efficienti. Lo slogan di *movidat* è “ovunque tu sia, il tuo *business* a portata di mano!”, infatti le applicazioni *movidat* sono rivolte ai dipendenti che lavorano in movimento, i quali devono essere in grado di agire tempestivamente in caso di problematiche inaspettate. Le soluzioni *movidat* sono compatibili con la maggior parte dei *software* gestionali disponibili sul mercato. Tra le soluzioni di miglior successo vi sono:

- * ***moviCheck***: applicazione rivolta alle aziende che vogliono offrire alle proprie figure direzionali uno strumento in grado di analizzare in mobilità i dati di *business* più significativi;
- * ***moviSell***: applicazione specificatamente rivolta ai professionisti della vendita, ideata come supporto per la gestione dei rapporti con i clienti.



Figura 1.3: Logo del marchio *movidat*

MoviORDER rientra tra le applicazioni del marchio *movidat*, soddisfacendo però i bisogni del cliente finale dell'azienda. Infatti, tramite *MoviORDER*, gli utenti possono acquistare dei prodotti della propria azienda direttamente *online*, senza doversi recare al negozio fisico.

1.2 Offerta di stage

Il 10 Aprile 2018 si è tenuta a Padova la 15-esima edizione di *Stage-IT*, iniziativa che mira ad agevolare l'incontro tra aziende e studenti universitari che puntano ad entrare nel mondo del lavoro con specifico riferimento al settore *ICT_G*, favorendo un'occasione di conoscenza reciproca tramite colloqui individuali. Le aziende partecipanti propongono spesso progetti innovativi con ottime opportunità di accrescimento delle competenze individuali. Proprio per questo motivo, lo stagista ha deciso di partecipare attivamente all'evento, concludendo positivamente sette colloqui.



Figura 1.4: Logo dell'evento *Stage-IT*

Le aziende sono state selezionate dallo studente per ambito e tecnologie di sviluppo adottate nel progetto. In particolare, lo studente era in cerca di:

- * progetti di sviluppo in ambito *mobile* (preferibilmente *Android*): al giorno d'oggi i *task* si stanno spostando sempre di più dalle postazioni *desktop* ai dispositivi portatili;

- * progetti di sviluppo di *applicazioni web_G* (preferibilmente con utilizzo di *framework* e *librerie_G Javascript* moderne): le *skill* di programmazione in *Javascript* sono richieste da gran parte delle aziende che si occupano della realizzazione di applicazioni *web*.

Tra le varie offerte di *stage*, VisioneImpresa rientrava in entrambe le preferenze, infatti proponeva un progetto di sviluppo in ambito *mobile* con l'utilizzo di un *framework cross-platform*. Nonostante lo *stage* non prevedesse lo sviluppo in *codice nativo_G Android*, permetteva comunque la realizzazione di un'applicazione *mobile*. Inoltre, richiedendo il progetto l'utilizzo di un *framework cross-platform*, era pressoché implicito l'utilizzo di *tecnologie web_G*, tra le quali anche *Javascript*. Per cui, il buon compromesso tra tecnologie conosciute e tecnologie ritenute interessanti ha favorito VisioneImpresa tra le varie offerte analizzate.

1.3 Obiettivi e pianificazione

Il progetto prevedeva la realizzazione di un'applicazione *mobile*, principalmente *Android* e *iOS*, che consentisse ad un utente registrato, tramite la lettura di codici a barre o l'inserimento manuale di codici articolo, di inviare ordini di acquisto al proprio fornitore. L'azienda ha richiesto lo sviluppo delle seguenti funzionalità:

- * **login**: permette ad un utente di essere riconosciuto come cliente dell'azienda. Le credenziali di accesso vengono distribuite dall'azienda insieme all'applicazione;
- * **gestione del carrello**:
 - **aggiunta di un nuovo articolo**: permette all'utente autenticato di aggiungere un nuovo articolo al carrello. L'aggiunta di un nuovo articolo prevede l'inserimento di un codice articolo, tramite scansione di un codice a barre o inserimento manuale e, successivamente, l'inserimento di una quantità da ordinare;
 - **modifica di un articolo**: permette all'utente autenticato di modificare la quantità di un articolo in carrello;
 - **selezione articoli**: permette all'utente autenticato di selezionare uno o più articoli in carrello;
 - **deselezione articoli**: permette all'utente autenticato di deselectare uno o più articoli in carrello;
 - **rimozione articoli**: permette all'utente autenticato di rimuovere gli articoli selezionati dal carrello;
 - **invio ordine**: permette all'utente autenticato di inviare un ordine composto dagli articoli selezionati in carrello.
- * **invio di e-mail di conferma**: nel caso in cui un ordine venga inviato con successo, il sistema deve inviare all'azienda e all'utente autenticato una *mail* di conferma contenente un riepilogo dell'ordine.

I prodotti attesi per il termine dello *stage* erano:

- * **analisi dei requisiti** per l'applicazione da realizzare: partendo dalle specifiche e dalla microanalisi ricevuta, lo stagista doveva definire le funzionalità offerte dall'applicazione, i dettagli dei *web services* di comunicazione tra *database* e *front end*, e l'interfaccia grafica dell'applicazione;
- * **applicazione moviORDER**: sviluppo dell'applicazione in ambiente *multipiattaformaG* in seguito alla scelta del *framework* ritenuto più opportuno;
- * **servizio web**: sviluppo di un servizio *web* che permettesse all'applicazione di accedere ed interagire con un *database* su un *server cloud* di VisioneImpresa;
- * **manuali** utente e sviluppatore.

Per adempiere a questi obblighi lo stagista ha pianificato delle attività che sono state concordate con il *tutor* aziendale. Lo *stage* prevedeva 320 ore di lavoro che sono state distribuite nel periodo tra l'inizio e la fine dello *stage*, 2 Luglio 2018 e 7 Settembre 2018 rispettivamente. La pianificazione oraria delle attività all'inizio dello *stage* è indicata nella seguente tabella.

Attività	Ore
Formazione assistita sui <i>software gestionali</i> di VisioneImpresa e sulle applicazioni analoghe a <i>moviORDER</i>	40
Formazione individuale sui <i>framework cross-platform</i> e scelta del <i>framework</i> ritenuto più opportuno	40
Ridefinizione delle specifiche comprendente delle soluzioni da realizzare e delle metodologie per implementarle	40
Realizzazione dei <i>web services</i> per l'interazione con il <i>database</i> : servizio di autenticazione, servizio di lettura dati, servizio di scrittura dati e servizio di invio <i>e-mail</i>	40
Realizzazione della <i>business logic</i> dell'applicazione	40
Realizzazione delle interfacce grafiche dell'applicazione: <i>login</i> , gestione carrello, aggiunta/modifica articolo e invio ordine	40
Test di scambio dati tra <i>VisionENTERPRISE</i> e <i>moviORDER</i>	40
Documentazione del codice e stesura del manuale utente e sviluppatore	40

Tabella 1.1: Pianificazione oraria del periodo di *stage*

Viene di seguito riportato un *diagramma di Gantt* che illustra come le attività di *stage* sono state allocate nel tempo di calendario da parte dello stagista. Il diagramma mostra le dipendenze temporali tra le varie attività e il periodo di ferie che l'azienda ha concesso allo stagista, indicato in viola.



Figura 1.5: Diagramma di *Gantt* della pianificazione delle attività di *stage*

1.4 Rischi

Parallelamente allo studio dei documenti forniti dal *tutor* aziendale, è stata sviluppata un'analisi preventiva dei rischi che si sarebbero potuti verificare durante lo svolgimento delle attività di progetto. Al fine di attuare una corretta e quindi utile gestione dei rischi, essi sono stati identificati nel contesto di processo e di prodotto. In seguito all'identificazione dei rischi sono state analizzate le probabilità di occorrenza, imminenza e l'impatto di ciascun rischio, pianificando come evitarne o mitigarne gli effetti. Durante lo svolgimento del progetto è stato costantemente eseguito un controllo per rilevare eventuali indicatori di rischio e ogni nuovo rischio rilevato è stato aggiunto alla seguente tabella, insieme alle azioni atte a mitigarne gli effetti.

Rischio	Descrizione	Grado	Mitigazione
Tecnologie non conosciute	Il tempo richiesto per l'apprendimento delle nuove tecnologie da parte dello stagista potrebbe causare ritardi nello sviluppo	Occorrenza: Media Pericolosità: Media	Effettuare una buona analisi iniziale delle tecnologie richieste ed usufruire del tempo di formazione per prendere dimestichezza con le tecnologie non conosciute
Modifica dei requisiti	Nonostante i requisiti esposti inizialmente siano chiari, vi è la possibilità che questi vengano modificati dal <i>tutor</i> aziendale	Occorrenza: Bassa Pericolosità: Alta	Negoziare la modifica ai requisiti nel caso in cui siano richiesti cambiamenti eccessivi da parte del <i>tutor</i> aziendale
Stima dei tempi	Può accadere che lo stagista abbia pianificato erroneamente alcune attività e che durante lo sviluppo si sfiorino i tempi concordati	Occorrenza: Media Pericolosità: Alta	Agire tempestivamente sulla pianificazione, ove possibile, nei casi più gravi. Pianificare in maniera intelligente inserendo dei <i>periodi di slack</i> G tra le attività ritenute più critiche
Difficoltà nelle interazioni	Durante lo <i>stage</i> le interazioni avvengono spesso tramite scambio di <i>e-mail</i> tra lo stagista e il <i>tutor</i> aziendale. Può succedere che alcune risposte arrivino in tempi prolungati, causando tempi morti nello sviluppo dell'applicazione	Occorrenza: Bassa Pericolosità: Media	Se il problema dovesse diventare ingestibile, proporre al <i>tutor</i> aziendale l'utilizzo di uno strumento di comunicazione alternativo, altrimenti presentarsi fisicamente in ufficio per discutere delle problematiche di maggiore entità

Tabella 1.2: Analisi dei rischi

1.5 Organizzazione del testo

Il secondo capitolo descrive il modello di *ciclo di vita_G* adottato durante lo sviluppo del progetto.

Il terzo capitolo descrive le tecnologie utilizzate durante lo sviluppo del progetto.

Il quarto capitolo descrive i *casi d'uso_G* e i corrispondenti requisiti rilevati durante la fase di analisi dei requisiti del progetto.

Il quinto capitolo descrive l'*architettura_G* dell'applicazione, comprensiva di contestualizzazione dei *design pattern_G* adottati.

Il sesto capitolo descrive i dettagli implementativi del progetto.

Il settimo capitolo descrive i processi di verifica e validazione attuati nel progetto.

L'ottavo capitolo descrive le conclusioni tratte dallo studente in merito al progetto realizzato.

Riguardo la stesura del testo, relativamente al documento, sono state adottate le seguenti convenzioni tipografiche:

- * gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- * per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *termine_G*;
- * i termini in lingua straniera, o facenti parte del gergo tecnico, sono evidenziati con il carattere *corsivo*.

Capitolo 2

Processo di sviluppo

Durante lo sviluppo di un progetto *software* è importante aderire ad un modello di ciclo di vita. La durata del ciclo di vita di un *software* inizia dalla sua concezione, ossia il momento in cui nasce il bisogno, passa poi per lo sviluppo e l'utilizzo, prolungato nel tempo e in cui è soggetto a manutenzione, per poi terminare con il ritiro del prodotto. L'avanzamento tra questi stati avviene tramite l'esecuzione di attività definite nel modello di ciclo di vita adottato. I progetti in cui non viene adottato un modello di ciclo di vita sono detti *code-n-fix_G* e l'insieme delle attività è priva di organizzazione e rende il progetto caotico e poco gestibile. Aderire ad un modello di ciclo di vita è quindi essenziale ma determina vincoli sulla pianificazione e sulla gestione di progetto, per cui è importante che la scelta del modello da adottare avvenga prima della pianificazione del progetto. In un modello di ciclo di vita le attività sono coese e raggruppate in processi e gli ingressi e le uscite di ciascuna attività sono identificati, al fine di permettere un ordinamento temporale tra esse.

2.1 Modello incrementale

Durante il corso di Ingegneria del *Software* sono stati studiati vari modelli di ciclo di vita. In particolare, viste le modalità di interazione e le richieste del *tutor* aziendale, per il progetto di *stage* poteva essere adottato un modello incrementale o una metodologia *Agile_G*. Non avendo l'azienda imposto un processo di sviluppo, lo stagista ha optato per il modello incrementale in quanto già utilizzato durante il progetto di Ingegneria del *Software*. Tale modello segue un approccio adattativo dove la realtà è considerata imprevedibile e per questo risulta utile nel caso in cui i requisiti possano cambiare in corso d'opera. Il modello presenta una fase iniziale di analisi e progettazione dove il problema viene compreso nel suo contorno fondamentale individuando i requisiti macroscopici e l'architettura del prodotto. Tale fase non viene ripetuta e risulta essenziale per la pianificazione dei cicli di *incremento_G* in cui si decide il numero di incrementi necessari a soddisfare i requisiti e si associano i requisiti ai vari incrementi pianificati. In seguito alla pianificazione è possibile transitare nella fase di realizzazione che comprende attività di progettazione in dettaglio e codifica. Tale fase è incrementale e al termine di ogni incremento si verifica che tutti i requisiti associati ad esso siano stati soddisfatti. Se la verifica va a buon fine è possibile integrare l'incremento con quanto già stato prodotto nelle fasi precedenti, costruendo in questa maniera una nuova *baseline_G* di prodotto. Viene presentata di seguito una figura illustrativa del modello incrementale.



Figura 2.1: Modello di ciclo di vita incrementale

Tra i principali vantaggi di tale modello vi sono:

- * ogni incremento di funzionalità permette un avvicinamento alle attese e una riduzione del rischio di fallimento;
- * le funzionalità più importanti sono le prime a raggiungere la stabilità poiché essendo inserite nei primi incrementi attraversano più cicli di verifica;
- * il fenomeno del *big-bang integration_G* viene evitato producendo valore ad ogni incremento;
- * il numero di incrementi è fissato in fase di pianificazione.

Nello specifico, nella pianificazione del progetto sono stati fissati i seguenti incrementi, tutti corrispondenti ad importanti *milestone_G* di progetto:

1. realizzazione del servizio *web*;
2. realizzazione della logica applicativa;
3. realizzazione delle interfacce grafiche;
4. stesura della documentazione annessa al progetto.

Durante il periodo di *stage* veniva pianificata una riunione con il *tutor* aziendale al raggiungimento di ogni *milestone*, al fine verificare che quanto prodotto nella corrispondente *baseline* fosse inerente alle attese.

Capitolo 3

Background tecnologico

In questo capitolo vengono presentate le tecnologie utilizzate durante lo sviluppo di *moviORDER*. La realizzazione dell'applicazione ha permesso l'apprendimento di nuove tecnologie e l'approfondimento di alcune già in parte conosciute. Alcune di queste sono state scelte dallo stagista in seguito al completamento dell'analisi dei requisiti, mentre la maggior parte sono state imposte dal *tutor* aziendale o dal dominio del problema. Le tecnologie scelte dallo stagista sono state concordate con il *team* di sviluppo di VisioneImpresa. Le prossime sezioni presentano le tecnologie in base al contesto in cui sono state utilizzate.

3.1 Framework

La presentazione delle tecnologie utilizzate per lo sviluppo di *moviORDER* comincia dalla scelta del *framework*, in quanto il *framework* scelto ha imposto l'utilizzo di alcuni linguaggi di programmazione usati durante il periodo di *stage*. Poiché il progetto richiedeva la realizzazione di un'applicazione che funzionasse in ambiente *Android* e *iOS*, il *tutor* aziendale ha consigliato l'utilizzo di un *framework cross-platform*. Questo perché data la diversità delle tecnologie richieste per lo sviluppo di codice nativo *Android* e *iOS*, e la limitata quantità di tempo a disposizione per la realizzazione del progetto, l'utilizzo di un *framework cross-platform* era la migliore soluzione per portare a termine il progetto nei tempi richiesti. Allo stagista è stato richiesto di decidere tra due *framework cross-platform*: *Xamarin* e *PhoneGap*.

Vengono di seguito descritti:

1. le motivazioni alla base dei *framework cross-platform*;
2. gli approcci alla base dei *framework cross-platform*;
3. il *framework Xamarin*;
4. il *framework PhoneGap*;
5. le motivazioni che hanno portato *PhoneGap* a prevalere su *Xamarin*.

3.1.1 Motivazioni alla base dei framework cross-platform

Al giorno d'oggi è impensabile realizzare un'applicazione *mobile* per una sola piattaforma perché il mercato è eccessivamente frammentato. Quindi, se si dovesse scegliere

di sviluppare un'applicazione per una sola piattaforma si perderebbe una potenziale parte di clienti. La seguente figura mostra, a scopi illustrativi, la frammentazione del mercato italiano nel 2016.



Figura 3.1: Frammentazione SO del mercato italiano nel 2016

Compresa la necessità di sviluppare più versioni della medesima applicazione in diverse piattaforme, il problema si sposta sulle risorse economiche e sul tempo che si ha a disposizione per lo sviluppo. Infatti sviluppare in diverse piattaforme comporta l'utilizzo di differenti linguaggi di programmazione e quindi la necessità di avere più programmatore esperti, precisamente almeno uno per piattaforma. Altre variabili di cui tener conto sono gli strumenti di sviluppo necessari, le *API* che si hanno a disposizione e fattori quali i sensori disponibili sui dispositivi, la dimensione degli schermi e le capacità di calcolo differenti.

L'obiettivo dei *framework cross-platform* è la risoluzione di tutti questi problemi in maniera efficiente ed efficace in termini di risorse utilizzate, quindi, più precisamente, la riduzione degli effetti negativi della frammentazione del mercato.



Figura 3.2: Significato di *cross-platform*

Per raggiungere questo obiettivo i *framework cross-platform* consentono di utilizzare un solo linguaggio di programmazione, o un insieme ristretto di linguaggi, per sviluppare un unico codice sorgente, che viene poi convertito nel codice nativo delle piattaforme sulle quali si desidera distribuire l'applicazione.

Per concludere, dati oggettivi dimostrano che durante il 2016 l'utilizzo dei *framework cross-platform* ha permesso un risparmio in termini di risorse economiche nell'80% dei casi ed un risparmio di tempo nell'83% dei casi.

3.1.2 Approcci alla base dei framework cross-platform

Per la scelta del *framework* più idoneo è stato richiesto di studiare gli approcci secondo i quali i *framework* permettono la distribuzione su varie piattaforme. Esistono principalmente quattro approcci in base ai quali è possibile classificare i *framework*:

- * approccio *web*;
- * approccio ibrido;
- * approccio interpretato;
- * approccio *cross-compiled*.

In questa tesi vengono presentati solamente l'approccio ibrido e quello interpretato, poiché utilizzati dai *framework* proposti dal *tutor* aziendale.

L'approccio ibrido si interpone tra la realizzazione di un'applicazione *web* e lo sviluppo di un'applicazione *mobile* in codice nativo. In questo tipo di approccio l'applicazione viene sviluppata utilizzando tecnologie *web* ed eseguita all'interno di un *container* nativo sul dispositivo *mobile*. Per eseguire l'applicazione viene utilizzato il motore di *rendering* del *browser* del dispositivo, il quale si occupa di interpretare e visualizzare il contenuto *HTML* dell'applicazione tramite una visualizzazione *web* a schermo intero. L'accesso alle funzionalità native offerte dal dispositivo è permesso grazie ad un livello astratto che si interpone tra l'applicazione ibrida e tali funzionalità. Questo livello astratto espone le funzionalità tramite *API Javascript*.

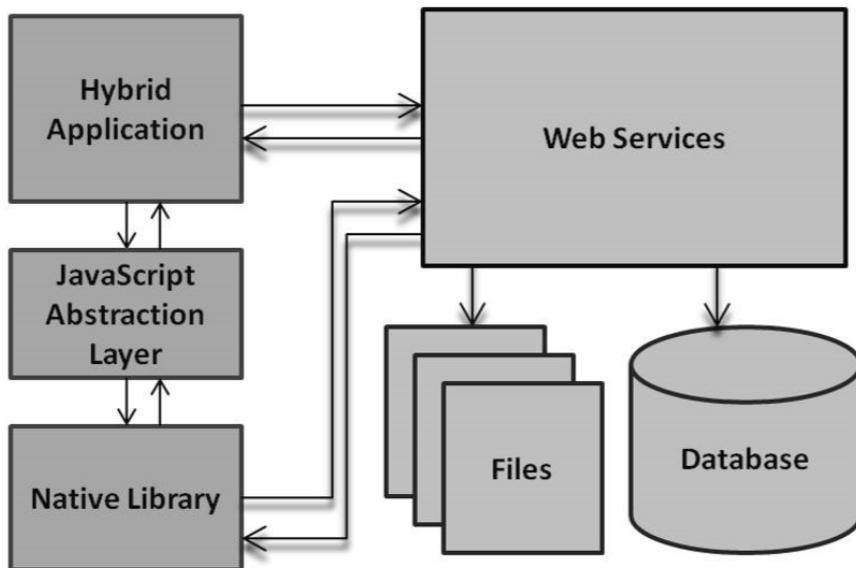


Figura 3.3: Architettura di un'applicazione ibrida

Nel caso delle applicazioni interpretate il codice sorgente dell'applicazione viene distribuito sul dispositivo *mobile*, dove viene successivamente interpretato da un interprete che si occupa di eseguire il codice a *run-time*. Lo sviluppo *cross-platform* è supportato dall'interprete, che permette di eseguire il codice sorgente su differenti piattaforme. L'applicazione interpretata interagisce con un livello astratto per accedere alle *API* native. Un vantaggio di questo approccio è che utilizza elementi delle specifiche interfacce grafiche native per l'interazione utente. Infine, la logica applicativa viene prelevata in maniera del tutto indipendente dalla piattaforma sulla quale l'applicazione viene eseguita.

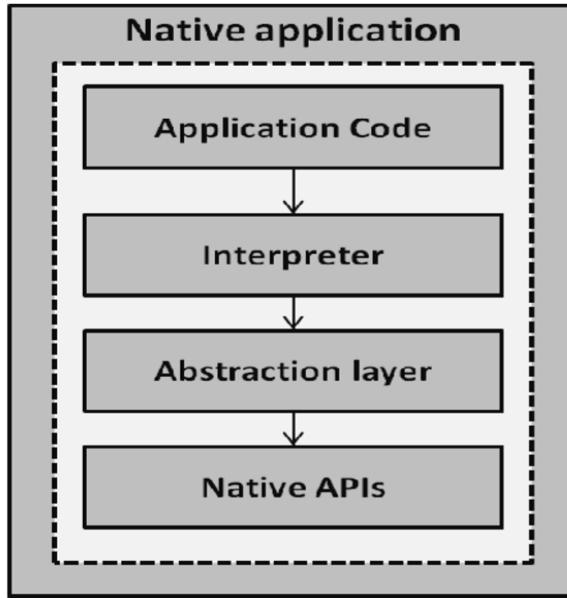


Figura 3.4: Architettura di un'applicazione interpretata

3.1.3 Xamarin

Xamarin è un *framework cross-platform* di proprietà dell'azienda *Microsoft*, il quale utilizza due approcci differenti: l'approccio interpretato per l'ambiente *Android* e *Windows*, e l'approccio *cross-compiled* per l'ambiente *iOS*. Più precisamente, per le piattaforme *Android* e *Windows* è possibile generare l'applicazione direttamente tramite i *tool* messi a disposizione dal *framework* e successivamente distribuirla sui rispettivi *store*, mentre per la piattaforma *iOS* è necessario un passo aggiuntivo. Infatti, per eseguire la compilazione dell'applicazione è richiesto il passaggio per una macchina *Apple* che abbia installato *XCode*. Infine, *Xamarin* richiede che per lo sviluppo dell'applicazione venga utilizzato il linguaggio *C#G*. Nella figura sottostante viene presentata l'architettura di *Xamarin*.

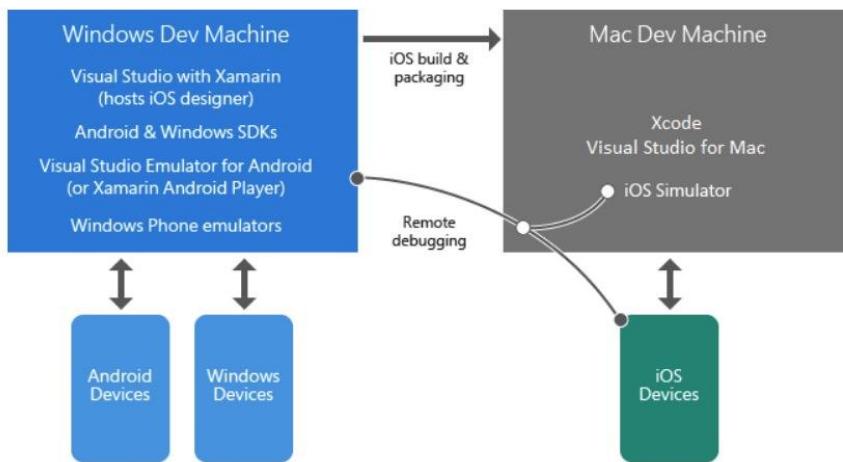


Figura 3.5: Architettura del *framework Xamarin*

3.1.4 PhoneGap

PhoneGap è un *framework cross-platform* di proprietà dell'azienda *Apache*, il quale utilizza un approccio ibrido. Quindi permette la realizzazione di applicazioni *mobile* tramite l'utilizzo di tecnologie *web*, che sono al giorno d'oggi strumenti conosciuti da tutti gli sviluppatori. L'accesso ai componenti hardware dei dispositivi *mobile* è permesso grazie all'utilizzo di *plugin* scaricabili dalla pagina ufficiale del *framework*. Vantaggi importanti del *framework* sono la presenza di documentazione completa per i *plugin* e l'esistenza di una *community* grande e disponibile. Infine, *PhoneGap* rende disponibili degli strumenti che facilitano lo sviluppo dell'applicazione: *PhoneGap Desktop App*, *PhoneGap CLI*, *PhoneGap App* e *PhoneGap Build*. I primi tre verranno descritti successivamente, in quanto fanno parte dell'ambiente di sviluppo utilizzato durante lo *stage*. *PhoneGap Build* è uno strumento che permette di eseguire la *build* dell'applicazione direttamente su un *server cloud Adobe*, a partire da un file zip contenente la cartella con il codice sorgente dell'applicazione. In seguito alla *build* è possibile generare e scaricare automaticamente l'applicazione per *Windows* o *Android*. Per *iOS* è necessario formire i certificati richiesti da *Apple* per la distribuzione dell'applicazione. Nella pagina successiva vengono presentate l'architettura di *PhoneGap* e una figura illustrativa di *PhoneGap Build*.

PhoneGap Architecture

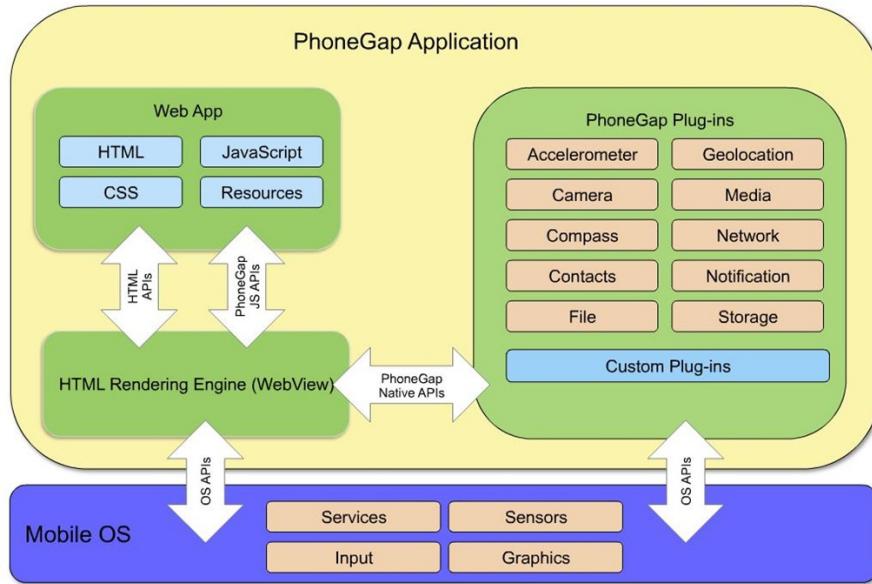


Figura 3.6: Architettura del framework PhoneGap

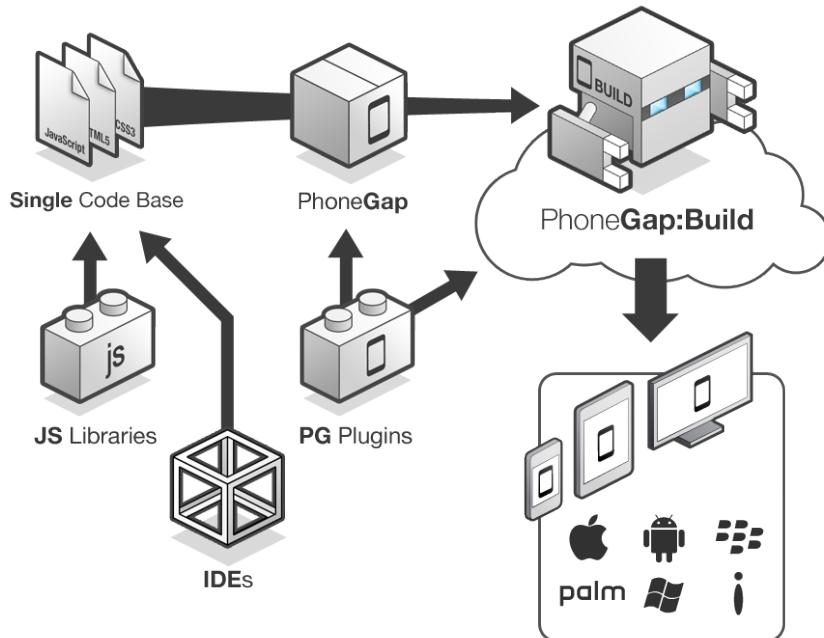


Figura 3.7: Figura illustrativa di PhoneGap Build

3.1.5 La scelta di PhoneGap

I motivi per i quali lo stagista ha optato per il *framework PhoneGap* sono i seguenti:

1. **linguaggio di programmazione:** *PhoneGap* richiedeva l'utilizzo di tecnologie *web* già conosciute e apprese dallo stagista all'Università. Lo studio del linguaggio *C#* imposto da *Xamarin* avrebbe richiesto un periodo di formazione maggiore delle 40 ore messe a disposizione per lo studio delle tecnologie;
2. **linguaggio Javascript:** come detto precedentemente, lo stagista era interessato ad approfondire il linguaggio *Javascript*, poiché richiesto al giorno d'oggi dalla maggior parte delle aziende che si occupano della realizzazione di applicazioni *web*;
3. **facilità nello sviluppo dell'interfaccia grafica:** utilizzando tecnologie *web* risultava semplice progettare e sviluppare un'interfaccia grafica responsive, in grado di adattarsi alla maggior parte dei dispositivi *mobile* presenti sul mercato.

3.2 Ambiente di sviluppo

Durante il periodo di *stage* è stato utilizzato uno specifico ambiente di lavoro, comprendente tecnologie imposte dal *tutor* aziendale e alcune scelte dallo stagista. La qualità delle tecnologie ha impatto diretto sulla qualità di processo e quindi su quella del prodotto, per cui è importante tenere l'ambiente di lavoro costantemente completo, ordinato e aggiornato. Per ottenere questo lo stagista ha dovuto analizzare le tecnologie scelte per assicurarsi che fossero le più adatte per il dominio del problema.

3.2.1 Suite di PhoneGap

La suite di *PhoneGap* ha costituito parte fondamentale dell'ambiente di lavoro. Sono stati utilizzati i seguenti *software*:

- * ***PhoneGap Desktop App*:** applicazione utilizzata inizialmente per prendere dimestichezza con il *framework*;
- * ***PhoneGap CLI*:** interfaccia a linea di comando utilizzata dopo aver preso dimestichezza con il *framework*;
- * ***PhoneGap App*:** applicazione *mobile* utilizzata inizialmente per testare l'applicazione generata dal *framework*.

PhoneGap Desktop App è disponibile su *Windows* o *Mac* e permette di iniziare ad utilizzare il *framework* con estrema facilità. Essa fornisce un'interfaccia grafica per creare, gestire e testare progetti *PhoneGap*. Nella pagina successiva viene presentata una figura che mostra l'interfaccia grafica di *PhoneGap Desktop App*.



Figura 3.8: PhoneGap Desktop App

PhoneGap CLI estende le funzionalità offerte da *PhoneGap Desktop App* tramite un’interfaccia a riga di comando. Prima del rilascio dell’app desktop, *PhoneGap CLI* veniva utilizzata come strumento principale per configurare e gestire il *framework*. *PhoneGap CLI* può essere utilizzata singolarmente oppure assieme a *PhoneGap Desktop App* e/o *PhoneGap Build*.

PhoneGap App è un’applicazione *mobile* che permette di testare l’applicazione *PhoneGap* senza generare ed installare nessun file applicazione. Per il test è sufficiente avviare l’esecuzione del progetto su *PhoneGap Desktop App* e collegare il computer di sviluppo con il dispositivo su cui è installata *PhoneGap App*. Dopo il collegamento sarà possibile testare completamente l’applicazione.



Figura 3.9: PhoneGap CLI e PhoneGap App

3.2.2 Editor e IDE

3.2.2.1 Sublime Text 3.0

Per lo sviluppo del progetto *PhoneGap* è stato utilizzato l'editor *Sublime Text 3.0*, ritenuto dallo stagista il più semplice e leggero per la realizzazione di applicazioni *web*.



Figura 3.10: Logo di Sublime Text 3.0

3.2.2.2 Android Studio

Durante lo *stage* l'applicazione *Android* generata tramite i *tool* offerti dal *framework PhoneGap* non era soddisfacente: spesso non funzionava o l'interfaccia grafica non rispecchiava quella desiderata. Per rendere l'app usabile è stato necessario installare *Android Studio*, il quale ha permesso di mettere mano sul codice nativo. *Android Studio* è un ambiente di sviluppo integrato (*IDE*) basato sul *software* di *JetBrains IntelliJ IDEA* e progettato specificamente per lo sviluppo di applicazioni *Android*.



Figura 3.11: Logo di *Android Studio*

3.2.2.3 XCode

Durante lo *stage* l'applicazione *iOS* generata tramite i *tool* offerti dal *framework PhoneGap* non era soddisfacente: spesso non funzionava o l'interfaccia grafica non rispecchiava quella desiderata. Per rendere l'*app* usabile è stato necessario installare *XCode*, il quale ha permesso di mettere mano sul codice nativo. *Xcode* è un ambiente di sviluppo integrato (*IDE*), sviluppato e mantenuto da *Apple*, che contiene una suite di strumenti utili allo sviluppo di *software* per i sistemi *macOS*, *iOS*, *watchOS* e *tvOS*.



Figura 3.12: Logo di *XCode*

3.2.2.4 Eclipse JEE

Durante lo *stage* è stata richiesta la realizzazione di un servizio *web* che si interponesse tra la logica applicativa di *moviORDER* e un *database* presente sul *server Azure* di VisioneImpresa. Il servizio *web* è stato realizzato in linguaggio *Java* tramite l'utilizzo di oggetti *servlet*. Per lo sviluppo degli oggetti *servlet* è stato utilizzato l'*IDE Eclipse JEE*, il quale offre buoni strumenti per lo sviluppo di applicazioni *web Java*. *Eclipse* è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma, ideato da un consorzio di grandi società quali *Ericsson*, *HP*, *IBM*, *Intel*, *MontaVista Software*, *QNX*, *SAP* e *Serena Software*, chiamato *Eclipse Foundation*.



Figura 3.13: Logo di *Eclipse*

3.2.3 Gestione DBMS

Durante il progetto è stato utilizzato il *DBMS_G* *Microsoft SQL Server* per gestire il *database* di *moviORDER*. Per una gestione veloce ed ottimale dello stesso si è deciso di usare il *software* *SQL Server Management Studio*. *SQL Server Management Studio (SSMS)* è un'applicazione *software* usata per configurare, gestire e amministrare *database*, in locale o su un *server cloud*, con il *DBMS Microsoft SQL Server*.

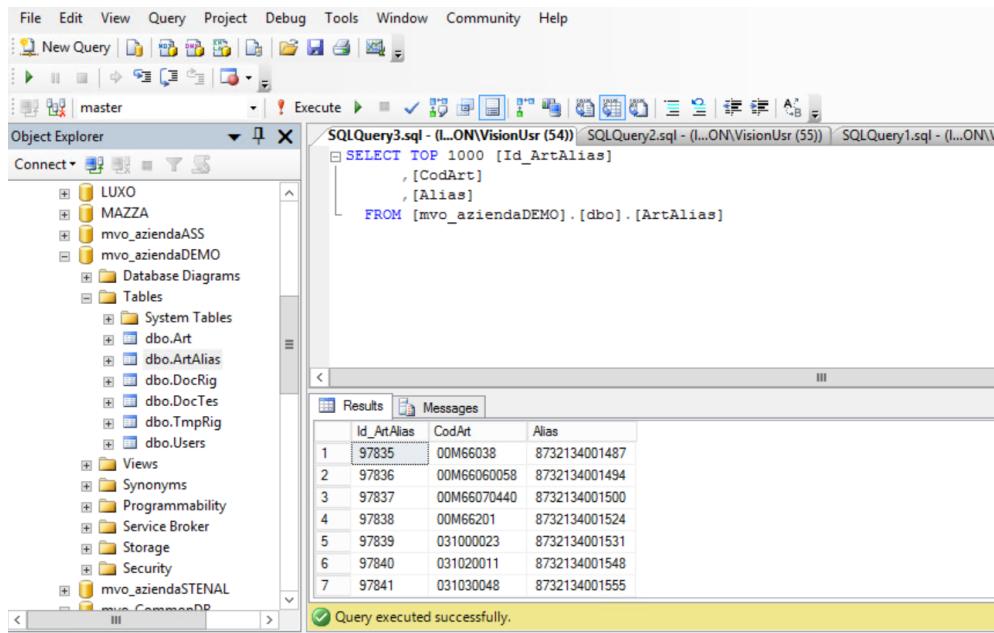


Figura 3.14: Screenshot di *SQL Server Management Studio*

3.2.4 Server web

Per permettere l'esecuzione del servizio *web* sul *server cloud* di VisioneImpresa si è utilizzato *Apache Tomcat*. *Apache Tomcat* (o semplicemente *Tomcat*) è un *web server* (nella forma di contenitore *servlet*) *open source* sviluppato dalla *Apache Software Foundation*. Implementa le specifiche *JavaServer Pages (JSP)* e *servlet*, fornendo quindi una piattaforma *software* per l'esecuzione di applicazioni *web* sviluppate in linguaggio *Java*.

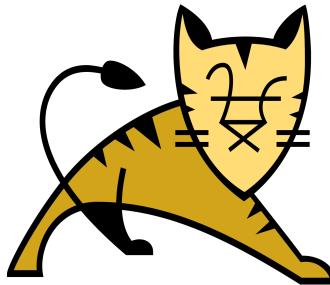


Figura 3.15: Logo di *Apache Tomcat*

3.2.5 Cloud computing

Il servizio *web* e il *database* con il quale l'applicazione interagisce risiedono su un *server Azure* di VisioneImpresa. *Microsoft Azure* (precedentemente nota come *Windows Azure*) è la piattaforma *cloud* pubblica di *Microsoft*, che offre servizi di *cloud computing*. Tramite Azure vengono erogati servizi appartenenti a diverse categorie, quali: risorse di elaborazione, archiviazione e memorizzazione dati, trasmissione dati e interconnessione di reti, analisi, *intelligence*, apprendimento automatico, sicurezza e gestione delle identità, monitoraggio e gestione, nonché servizi per lo sviluppo di applicazioni. Per accedere al *server* da remoto è stato utilizzato il *software* “Connessione Desktop Remoto di Windows”.



Figura 3.16: Logo di *Microsoft Azure*

3.2.6 Strumenti di testing

Durante lo sviluppo di *moviORDER* sono stati eseguiti dei test per verificare il corretto funzionamento del servizio *web* e dell'applicazione. Per il test dell'*API* del servizio *web* si è utilizzato *Postman*. *Postman* è uno strumento di *API testing* che permette di testare *API* direttamente, o come parte di test d'integrazione, per determinare se soddisfano criteri di funzionalità, affidabilità, performance e sicurezza. Nel caso di *moviORDER*, il test avviene tramite l'invio di una richiesta *HTTP POST* all'*API* sul *server cloud*. Successivamente il *software* visualizza la stringa in formato *JSON* restituita dal *server* e lo sviluppatore può verificare se la risposta del servizio è quella attesa.



Figura 3.17: Logo di *Postman*

Per testare il corretto funzionamento dell'applicazione si è utilizzata la *console* del browser *Google Chrome*, facente parte degli strumenti offerti dallo stesso per gli sviluppatori *web*. Tramite la *console* è stato possibile verificare il codice *JavaScript* che costituisce la logica applicativa di *moviORDER*.

 A screenshot of the Google Chrome developer tools console. The interface shows various tabs like Elements, Network, and Console. The Console tab is active, displaying a log of JavaScript errors. One error is highlighted with a red background: "Uncaught ReferenceError: stripe_setup is not defined". The stack trace for this error is shown below, indicating it occurred at line 12:1 of an anonymous script. Other logs show XHR requests and jQuery code execution.


```

<top frame> > Preset log
▶ XHR finished loading: POST
  "http://partners.dev/orders/set_shipments".
> stripe_setup()
✖ ▶ Uncaught ReferenceError: stripe_setup is not defined
  at <anonymous>:12:1
  at Object.InjectedScript._evaluateOn (<anonymous>:895:140)
  at Object.InjectedScript._evaluateAndWrap (<anonymous>:828:34)
  at Object.InjectedScript.evaluate (<anonymous>:694:21)
  
```

Figura 3.18: Screenshot della console di *Google Chrome*

3.2.7 Strumenti di versioning e ticketing

Per scelta dello stagista il progetto è stato sottoposto a controllo di versione in ogni sua parte: applicazione, servizio *web* e documentazione. Questo ha permesso, principalmente nel caso dell'applicazione, di tornare a baseline sicure nel caso di sovrascritture, perdite accidentali e *commit* di codice con errori di programmazione. Per il controllo di versione si è utilizzato lo strumento *Git* e in particolare il servizio di hosting *GitHub*. Lo stagista ha scelto tali *software* poiché già utilizzati in progetti didattici durante l'Università.

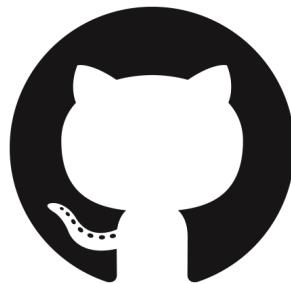


Figura 3.19: Logo di *GitHub*

Lo stagista ha scelto inoltre di utilizzare lo strumento di *ticketing Asana*, in modo da facilitare la pianificazione del progetto. *Asana* è un'applicazione *web* e *mobile* progettata per aiutare i *team* ad organizzare, tracciare e gestire il loro lavoro. In particolare, lo strumento è stato utilizzato per dare una scadenza ai *task* assegnati dal *tutor* aziendale.



Figura 3.20: Logo di *Asana*

3.2.8 Strumenti di modellazione e documentazione

Il progetto ha richiesto lo sviluppo di diagrammi di *Gantt* in fase di pianificazione e di diagrammi *UML* in fase di analisi dei requisiti e progettazione. Per la realizzazione dei diagrammi di *Gantt* si è utilizzato *Gantt Project*, per i diagrammi *UML* dei casi d'uso

si è utilizzato *Astah UML*, mentre per i diagrammi *UML* dei *package* e delle classi si è utilizzato *Visual Paradigm CE*. I *software* sono stati scelti perché già appresi durante il progetto di Ingegneria del *Software* all'Università.



Figura 3.21: Loghi di *Gantt Project*, *Astah UML* e *Visual Paradigm CE*

Per la realizzazione dei diagrammi *ER* e delle figure illustrative dell'architettura del prodotto si è utilizzato il *software* *Lucidchart*. Per la stesura della documentazione si è utilizzato l'*editor* *TexMaker*, anch'esso appreso durante il progetto di Ingegneria del *Software*. *TexMaker* permette l'integrazione con un dizionario per il controllo ortografico e la compilazione e visione del *pdf* prodotto.



Figura 3.22: Loghi di *Lucidchart* e *TexMaker*

3.2.9 Linguaggi di programmazione e markup

3.2.9.1 HTML5, CSS3 e JavaScript

Siccome è stato scelto il *framework* *PhoneGap*, lo stagista ha dovuto utilizzare i linguaggi *HTML5*, *CSS3* e *JavaScript*, in quanto *PhoneGap* richiede di sviluppare l'applicazione tramite l'utilizzo di tecnologie *web*. In particolare, *HTML5* è stato scelto perché include un insieme di funzionalità che permettono di valorizzare le interfacce *mobile*. Alcune di queste evidenziano come *HTML5* sia già per sua natura orientato al *mobile*. In particolare *HTML5* fornisce *API* per:

- * **geolocalizzazione:** con la scrittura di poco codice, forniscono la posizione del dispositivo in coordinate terrestri. Quindi, la stessa funzionalità su uno *smartphone* o un *tablet* fornisce la posizione dell'utente stesso;
- * **eventi touch:** mentre i meccanismi di input nei PC consistono per lo più nella tastiera e nel mouse, nei dispositivi mobili quasi tutto passa per il *touch screen*, e avere funzionalità comode per gestire questo strumento consente un'interazione più ricca e senza limitazioni per l'utente. Le gestualità da attuare su un *display*, nel mondo *mobile*, costituiscono un vero e proprio linguaggio fondamentale nella *user experience*;
- * **controllo batteria:** considerata l'importanza rivestita dalle risorse energetiche, l'esistenza stessa di questa libreria nel linguaggio dimostra come il suo impiego sia particolarmente mirato al panorama *mobile*.

Ciò che ha favorito la scelta di *CSS3* sono state le *media queries*_G. Esse permettono di definire regole stilistiche in base alla tipologia del mezzo di visualizzazione, delle sue dimensioni e della sua attuale disposizione (*portrait* o *landscape*). Ciò influisce non solo sull'aspetto esteriore degli elementi ma anche sul loro posizionamento e quindi sulla struttura stessa dell'interfaccia.

Per quanto riguarda il linguaggio *JavaScript* si è utilizzato *JavaScript* puro, senza l'utilizzo di *framework* o *JQuery*_G. Una particolarità del linguaggio, detta *AJAX*, ha reso possibile eseguire chiamate all'*API* del servizio *web* di *moviORDER*. *AJAX*, acronimo di *Asynchronous JavaScript and XML*, è una tecnica di sviluppo *software* per la realizzazione di applicazioni *web* interattive (*Rich Internet Application*). Lo sviluppo di applicazioni *HTML* con *AJAX* si basa su uno scambio di dati in *background* fra *web browser* e *server*, che consente l'aggiornamento dinamico di una pagina *web* senza esplicito ricaricamento da parte dell'utente.



Figura 3.23: Loghi di *HTML5*, *CSS3* e *JavaScript*

I linguaggi scelti, grazie alle loro caratteristiche che li rendono orientati allo sviluppo *mobile*, insieme ai meccanismi che il *framework* *PhoneGap* utilizza per convertire una *web application* in un'applicazione *mobile*, permettono di effettuare meno modifiche in seguito per perfezionare l'applicazione su *Android* e *iOS*.

3.2.9.2 Java

Per la realizzazione del servizio *web* si è utilizzato il linguaggio *Java*. La scelta poteva ricadere su *Java* o *PHP*. È stato scelto *Java* in quanto possiede un compilatore, risulta più facilmente “*debuggabile*” rispetto a *PHP* e permette l'utilizzo di oggetti *servlet*. I

servlet sono oggetti *Java* che operano all'interno di un *server web*, oppure un *server* per applicazioni, che permettono la creazione di un'applicazione *web*. Nel caso del progetto, i *servlet* hanno permesso lo sviluppo dell'*API* che costituisce il servizio *web*. Una descrizione dell'*API*, di come i *servlet* sono stati utilizzati nel progetto e di come l'*API* viene interrogata dall'applicazione, è presente in sezione §6.1.1.3.



Figura 3.24: Logo di *Java*

3.2.9.3 L^AT_EX

Per la stesura della documentazione annessa a *moviORDER* è stato utilizzato il linguaggio L^AT_EX. La scelta è ricaduta su L^AT_EX poiché già appreso e utilizzato durante il progetto di Ingegneria del *Software*. L^AT_EX è un linguaggio di *markup* usato per la preparazione di testi, basato sul programma di composizione tipografica T_EX. Fornisce funzioni di *desktop publishing* programmabili e mezzi per l'automazione della composizione tipografica, inclusa la numerazione, i riferimenti incrociati, tabelle e figure, organizzazione delle pagine, bibliografie e molto altro. Infine la particolarità più utile del linguaggio è l'esistenza di *community* che rendono disponibili *template* riutilizzabili, come quello che è stato utilizzato per la stesura di questa tesi.



Figura 3.25: Logo di L^AT_EX

3.2.10 DBMS

Per la creazione, gestione e amministrazione di *database* è stato utilizzato il *DBMS Microsoft SQL Server*. È stato scelto questo *software* poiché già ampiamente utilizzato all'interno di VisioneImpresa. In questo modo gli sviluppatori dell'azienda potranno occuparsi della manutenzione del *database* di *moviORDER* con un *DBMS* conosciuto. *Microsoft SQL Server* usa una variante del linguaggio *SQL* standard chiamata *Transact-SQL* (*T-SQL*). *Transact-SQL* espande le prestazioni di *SQL* aggiungendo:

- * funzioni per controllo di flusso;
- * possibilità di definire variabili locali;
- * varie funzioni per la manipolazione di stringhe, date ed espressioni matematiche;
- * miglioramento delle istruzioni *DELETE* e *UPDATE*.



Figura 3.26: Logo di *SQL Server*

Capitolo 4

Analisi dei requisiti

In questo capitolo vengono descritti i casi d'uso e i requisiti della piattaforma *moviORDER*, individuati e classificati per definire nel dettaglio obiettivi e funzionalità del sistema. I casi d'uso e i requisiti sono stati dedotti da un'analisi preliminare eseguita dal *tutor* aziendale, la quale è stata perfezionata dallo stagista per perseguire massima efficienza ed efficacia del sistema. Le convenzioni adottate per il codice univoco di casi d'uso e requisiti sono presentate in Appendice §B.

4.1 Casi d'uso

La descrizione di ogni caso d'uso si compone dei seguenti punti:

1. **attori**: lista di attori principali e secondari coinvolti nel caso d'uso;
2. **descrizione**: breve descrizione del caso d'uso;
3. **pre-condizioni**: condizioni che devono essere verificate prima dell'esecuzione del caso d'uso;
4. **post-condizioni**: condizioni che devono essere verificate dopo dell'esecuzione del caso d'uso;
5. **scenario principale**: descrizione composta dal flusso dei sottocasi d'uso del caso d'uso principale;
6. **scenari alternativi**: descrizione composta dai casi d'uso che non appartengono al flusso principale di esecuzione del caso d'uso (estensioni), se presenti;
7. **specializzazioni**: indica quali sono tutte le specializzazioni del caso d'uso, se presenti;
8. **generalizzazioni**: indica quali sono tutte le generalizzazioni del caso d'uso, se presenti.

Per lo studio dei casi di utilizzo più complessi della piattaforma, sono stati creati dei diagrammi dei casi d'uso che meglio descrivono funzioni e/o servizi offerti dal sistema, così come sono percepiti e utilizzati dagli attori che interagiscono con lo stesso. Per la definizione dei diagrammi *UML* dei casi d'uso è stato utilizzato lo standard *UML 2.0_G*.

4.1.1 Attori del sistema

Lo scopo di *moviORDER* è permettere alle aziende che forniscono prodotti di vendere gli stessi ai propri clienti tramite un'applicazione multiplataforma. *MoviORDER* viene distribuita da VisioneImpresa alle aziende che forniscono prodotti, la quale viene distribuita dalle aziende stesse ai propri clienti. Gli utilizzatori finali di *moviORDER* sono quindi i clienti delle singole aziende clienti di VisioneImpresa. L'accesso all'applicazione è consentito solamente agli utenti provvisti di credenziali di accesso, le quali vengono distribuite, insieme all'applicazione, dal fornitore. Non è prevista quindi una funzionalità di registrazione. Nel contesto di *moviORDER* vi sono quindi due tipologie di attori:

1. **Utente non autenticato:** è un utente che non ha effettuato l'accesso al sistema, al quale viene offerta la sola funzionalità di autenticazione. Una volta che un utente non autenticato viene riconosciuto accedendo al sistema, diventa un utente autenticato;
2. **Utente autenticato:** è un utente che ha effettuato l'accesso al sistema e che può usufruire di tutte le sue funzionalità. Le funzionalità offerte all'utente autenticato sono:
 - * possibilità di effettuare il *logout*;
 - * possibilità di aggiungere articoli al proprio carrello;
 - * possibilità di modificare gli articoli nel proprio carrello;
 - * possibilità di rimuovere articoli dal proprio carrello;
 - * possibilità di inviare un ordine alla propria azienda.

4.1.2 UC1 - Azioni utente non autenticato

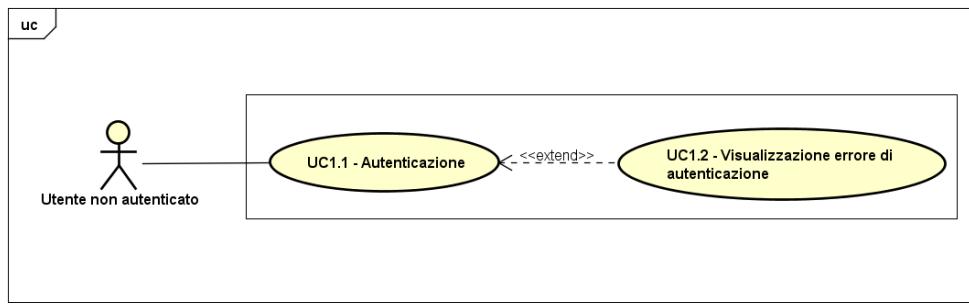


Figura 4.1: Use Case - UC1: Azioni utente non autenticato

- * **Attore:** Utente non autenticato;
- * **Descrizione:** L'attore può eseguire l'operazione di autenticazione alla piattaforma *moviORDER*;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione e non è ancora stato riconosciuto dal sistema;

- * **Post-condizioni:** L'attore ha eseguito l'operazione che desiderava compiere da utente non autenticato;
- * **Scenario principale:** UC1.1 - Autenticazione;
- * **Scenario alternativo:** L'attore ha fornito credenziali di accesso non corrispondenti a nessun utente registrato dall'azienda, oppure non riesce ad accedere al sistema perché è stato bloccato dall'azienda stessa: UC1.2 - Visualizzazione errore di autenticazione.

4.1.3 UC1.1 - Autenticazione

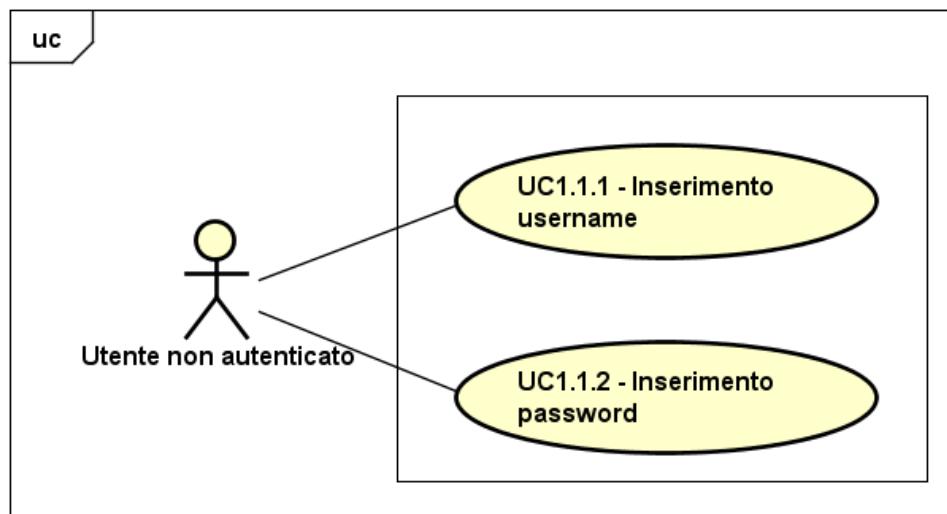


Figura 4.2: Use Case - UC1.1: Autenticazione

- * **Attore:** Utente non autenticato;
- * **Descrizione:** L'attore può eseguire l'operazione di autenticazione;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, non è ancora riconosciuto dal sistema e ha espresso la volontà di effettuare l'autenticazione a *moviORDER*;
- * **Post-condizioni:** L'attore ha eseguito l'operazione di accesso al sistema ed è quindi ora riconosciuto come utente autenticato;
- * **Scenario principale:**
 1. UC1.1.1 - Inserimento *username*;
 2. UC1.1.2 - Inserimento *password*.

4.1.4 UC1.1.1 - Inserimento username

- * **Attore:** Utente non autenticato;
- * **Descrizione:** L'attore deve inserire una *username* per l'operazione di autenticazione;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, non è ancora riconosciuto dal sistema e il sistema richiede l'inserimento di una *username* per l'operazione di autenticazione;
- * **Post-condizioni:** L'attore ha inserito la *username* per l'operazione di autenticazione;
- * **Scenario principale:** L'attore inserisce la *username* tramite una text box.

4.1.5 UC1.1.2 - Inserimento password

- * **Attore:** Utente non autenticato;
- * **Descrizione:** L'attore deve inserire una *password* per l'operazione di autenticazione;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, non è ancora riconosciuto dal sistema e il sistema richiede l'inserimento di una *password* per l'operazione di autenticazione;
- * **Post-condizioni:** L'attore ha inserito la *password* per l'operazione di autenticazione;
- * **Scenario principale:** L'attore inserisce la *password* tramite una text box.

4.1.6 UC1.2 - Visualizzazione errore di autenticazione

- * **Attore:** Utente non autenticato;
- * **Descrizione:** L'attore inserisce credenziali di accesso non corrispondenti a nessun utente registrato dall'azienda, oppure l'utente è stato bloccato dall'azienda stessa;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, non è ancora riconosciuto dal sistema e inserisce credenziali di accesso non corrispondenti a nessun utente registrato dall'azienda, oppure è stato bloccato;
- * **Post-condizioni:** L'attore ha visualizzato un messaggio d'errore relativo all'impossibilità di eseguire l'autenticazione per inserimento di credenziali errate o perché è stato bloccato;
- * **Scenario principale:** L'attore visualizza un messaggio d'errore relativo all'impossibilità di eseguire l'autenticazione per inserimento di credenziali errate o perché è stato bloccato.

4.1.7 UC2 - Azioni utente autenticato

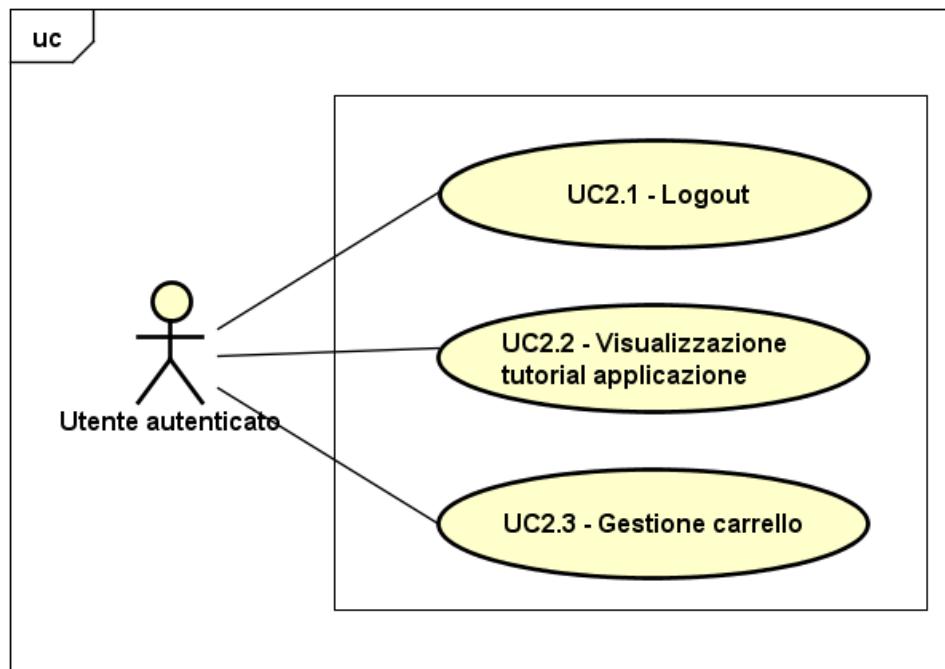


Figura 4.3: Use Case - UC2: Azioni utente autenticato

* **Attore:** Utente autenticato;

* **Descrizione:** L'attore può:

1. eseguire l'operazione di *logout*;
2. visualizzare il *tutorial* dell'applicazione;
3. gestire il proprio carrello.

* **Pre-condizioni:** L'attore ha avviato l'applicazione ed è riconosciuto dal sistema;

* **Post-condizioni:** L'attore ha eseguito le azioni che desiderava compiere da utente autenticato;

* **Scenario principale:**

1. UC2.1 - *Logout*;
2. UC2.2 - Visualizzazione *tutorial* applicazione;
3. UC2.3 - Gestione carrello.

4.1.8 UC2.1 - Logout

* **Attore:** Utente autenticato;

- * **Descrizione:** L'attore può eseguire l'operazione di *logout*;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha espresso la volontà di effettuare il *logout* da *moviORDER*;
- * **Post-condizioni:** L'attore ha eseguito l'operazione di *logout* da *moviORDER* ed è quindi uscito dal sistema tornando ad essere un utente non autenticato;
- * **Scenario principale:** L'attore esegue l'operazione di *logout* da *moviORDER* premendo sul relativo bottone.

4.1.9 UC2.2 - Visualizzazione tutorial applicazione

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può visualizzare il *tutorial* dell'applicazione;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto la visualizzazione del *tutorial* dell'applicazione;
- * **Post-condizioni:** L'attore ha visualizzato il *tutorial* dell'applicazione;
- * **Scenario principale:** L'attore visualizza il *tutorial* dell'applicazione premendo sul relativo bottone.

4.1.10 UC2.3 - Gestione carrello



Figura 4.4: Use Case - UC2.3: Gestione carrello

* **Attore:** Utente autenticato;

* **Descrizione:** L'attore può:

1. aggiungere un articolo al carrello;
2. modificare un articolo in carrello;
3. selezionare un singolo articolo in carrello;
4. deselectionare un singolo articolo in carrello;
5. selezionare tutti gli articoli in carrello;
6. deselectionare tutti gli articoli in carrello;
7. rimuovere articoli dal carrello;
8. inviare un ordine.

* **Pre-condizioni:** L'attore ha avviato l'applicazione ed è riconosciuto dal sistema;

* **Post-condizioni:** L'attore ha eseguito le operazioni che desiderava compiere sul proprio carrello;

* **Scenario principale:**

1. UC2.3.1 - Aggiunta articolo;
2. UC2.3.3 - Modifica articolo;
3. UC2.3.5 - Selezione singolo articolo;
4. UC2.3.6 - Deselezione singolo articolo;
5. UC2.3.7 - Selezione totale di articoli;
6. UC2.3.8 - Deselezione totale di articoli;
7. UC2.3.9 - Rimozione articoli selezionati;
8. UC2.3.10 - Invio ordine.

* **Scenari alternativi:**

- Durante l'operazione di aggiunta articolo, l'attore ha inserito un codice articolo non corrispondente a nessun articolo venduto dall'azienda, oppure ha inserito una quantità non permessa per l'articolo, oppure la scansione del codice a barre per la ricerca del codice articolo è fallita: UC2.3.2 - Visualizzazione errore di inserimento articolo;
- Durante l'operazione di modifica articolo, l'attore ha inserito una quantità non permessa per l'articolo: UC2.3.4 - Visualizzazione errore di modifica articolo;
- L'attore ha premuto il bottone di rimozione articoli selezionati senza aver selezionato alcun articolo dal carrello: UC2.3.11 - Visualizzazione errore di mancata selezione di articoli;
- L'attore ha premuto il bottone di invio ordine senza aver selezionato alcun articolo dal carrello: UC2.3.12 - Visualizzazione errore di mancata selezione di articoli.

4.1.11 UC2.3.1 - Aggiunta articolo



Figura 4.5: Use Case - UC2.3.1: Aggiunta articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può aggiungere un articolo al carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di aggiungere un articolo al carrello;
- * **Post-condizioni:** L'attore ha aggiunto l'articolo al carrello;
- * **Scenario principale:**
 1. UC2.3.1.1 - Inserimento codice articolo;
 2. UC2.3.1.4 - Inserimento quantità articolo;
 3. UC2.3.1.5 - Inserimento note articolo.

4.1.12 UC2.3.1.1 - Inserimento codice articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore deve inserire un codice articolo per l'operazione di aggiunta articolo;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di aggiungere un articolo in carrello e il sistema richiede l'inserimento di un codice articolo per l'operazione di aggiunta articolo;
- * **Post-condizioni:** L'attore ha inserito il codice articolo;
- * **Scenario principale:** L'attore inserisce il codice articolo manualmente (UC2.3.1.2) o tramite la scansione di un codice a barre (UC2.3.1.3);

* **Specializzazioni:**

1. UC2.3.1.2 - Inserimento manuale codice articolo;
2. UC2.3.1.3 - Inserimento codice articolo con scansione codice a barre.

4.1.13 UC2.3.1.2 - Inserimento manuale codice articolo

* **Attore:** Utente autenticato;

* **Descrizione:** L'attore deve inserire un codice articolo per l'operazione di aggiunta articolo;

* **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di aggiungere un articolo in carrello e quando il sistema richiede l'inserimento di un codice articolo per l'operazione di aggiunta articolo, l'attore esprime la volontà di voler inserire il codice articolo manualmente;

* **Post-condizioni:** L'attore ha inserito il codice articolo manualmente;

* **Scenario principale:** L'attore inserisce il codice articolo manualmente tramite una text box;

* **Generalizzazione:** UC2.3.1.1 - Inserimento codice articolo.

4.1.14 UC2.3.1.3 - Inserimento codice articolo con scansione codice a barre

* **Attore:** Utente autenticato;

* **Descrizione:** L'attore deve inserire un codice articolo per l'operazione di aggiunta articolo;

* **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di aggiungere un articolo in carrello e quando il sistema richiede l'inserimento di un codice articolo per l'operazione di aggiunta articolo, l'attore esprime la volontà di voler inserire il codice articolo mediante la scansione del codice a barre di un articolo;

* **Post-condizioni:** L'attore ha inserito il codice articolo mediante la scansione del codice a barre dell'articolo;

* **Scenario principale:** L'attore inserisce il codice articolo mediante la scansione del codice a barre dell'articolo;

* **Generalizzazione:** UC2.3.1.1 - Inserimento codice articolo.

4.1.15 UC2.3.1.4 - Inserimento quantità articolo

* **Attore:** Utente autenticato;

* **Descrizione:** L'attore deve inserire una quantità per l'operazione di aggiunta articolo;

- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di aggiungere un articolo in carrello e il sistema richiede l'inserimento di una quantità per l'operazione di aggiunta articolo;
- * **Post-condizioni:** L'attore ha inserito la quantità dell'articolo;
- * **Scenario principale:** L'attore inserisce la quantità dell'articolo tramite una text box.

4.1.16 UC2.3.1.5 - Inserimento note articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può inserire delle note durante l'operazione di aggiunta articolo;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di aggiungere un articolo in carrello e il sistema permette l'inserimento di note durante l'operazione di aggiunta articolo;
- * **Post-condizioni:** L'attore ha inserito delle note per l'articolo;
- * **Scenario principale:** L'attore inserisce delle note per l'articolo tramite una text area.

4.1.17 UC2.3.2 - Visualizzazione errore di inserimento articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore inserisce un codice articolo non corrispondente a nessun articolo venduto dall'azienda, oppure inserisce una quantità non permessa per l'articolo, oppure la scansione del codice a barre per la ricerca del codice articolo è fallita;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di aggiungere un articolo in carrello e ha inserito un codice articolo che non corrisponde a nessun articolo venduto dall'azienda, oppure ha inserito una quantità non permessa per l'articolo, oppure la scansione del codice a barre per la ricerca del codice articolo è fallita;
- * **Post-condizioni:** L'attore ha visualizzato un messaggio d'errore relativo all'impossibilità di aggiungere l'articolo in carrello;
- * **Scenario principale:** L'attore visualizza un messaggio d'errore relativo all'impossibilità di aggiungere l'articolo in carrello.

4.1.18 UC2.3.3 - Modifica articolo

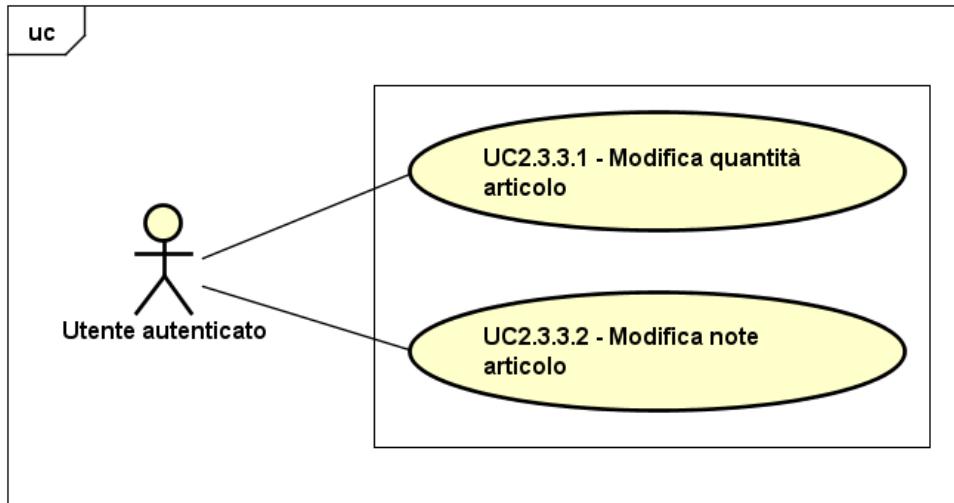


Figura 4.6: Use Case - UC2.3.3: Modifica articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può modificare un articolo in carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di modificare un articolo in carrello;
- * **Post-condizioni:** L'attore ha modificato l'articolo in carrello;
- * **Scenario principale:**
 1. UC2.3.3.1 - Modifica quantità articolo;
 2. UC2.3.3.2 - Modifica note articolo.

4.1.19 UC2.3.3.1 - Modifica quantità articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può modificare la quantità dell'articolo selezionato;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di modificare un articolo in carrello e il sistema permette la modifica della quantità dell'articolo selezionato;
- * **Post-condizioni:** L'attore ha modificato la quantità dell'articolo selezionato;
- * **Scenario principale:** L'attore modifica la quantità dell'articolo selezionato tramite una text box.

4.1.20 UC2.3.3.2 - Modifica note articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può modificare le note dell'articolo selezionato;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di modificare un articolo in carrello e il sistema permette la modifica della note dell'articolo selezionato;
- * **Post-condizioni:** L'attore ha modificato le note dell'articolo selezionato;
- * **Scenario principale:** L'attore modifica le note dell'articolo selezionato tramite una text box.

4.1.21 UC2.3.4 - Visualizzazione errore di modifica articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore inserisce una quantità non permessa per l'articolo selezionato;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di modificare un articolo in carrello e ha inserito una quantità non permessa per l'articolo selezionato;
- * **Post-condizioni:** L'attore ha visualizzato un messaggio d'errore relativo all'impossibilità di modificare l'articolo selezionato;
- * **Scenario principale:** L'attore visualizza un messaggio d'errore relativo all'impossibilità di modificare l'articolo selezionato.

4.1.22 UC2.3.5 - Selezione singolo articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può selezionare un articolo non ancora selezionato in carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di selezionare un articolo non ancora selezionato in carrello;
- * **Post-condizioni:** L'attore ha selezionato l'articolo in carrello;
- * **Scenario principale:** L'attore seleziona l'articolo in carrello tramite una checkbox.

4.1.23 UC2.3.6 - Deselezione singolo articolo

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può deselectare un articolo selezionato in carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di deselectare un articolo selezionato in carrello;

- * **Post-condizioni:** L'attore ha deselectato l'articolo in carrello;
- * **Scenario principale:** L'attore deselecta l'articolo in carrello tramite una check-box.

4.1.24 UC2.3.7 - Selezione totale di articoli

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può selezionare tutti gli articoli in carrello non ancora selezionati;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di selezionare tutti gli articoli in carrello non ancora selezionati;
- * **Post-condizioni:** L'attore ha selezionato tutti gli articoli in carrello non ancora selezionati;
- * **Scenario principale:** L'attore seleziona tutti gli articoli in carrello non ancora selezionati premendo su un bottone.

4.1.25 UC2.3.8 - Deselezione totale di articoli

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può deselectare tutti gli articoli in carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di deselectare tutti gli articoli in carrello;
- * **Post-condizioni:** L'attore ha deselectato tutti gli articoli in carrello;
- * **Scenario principale:** L'attore deselecta tutti gli articoli in carrello premendo su un bottone.

4.1.26 UC2.3.9 - Rimozione articoli selezionati

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può rimuovere gli articoli selezionati dal carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di rimuovere gli articoli selezionati dal carrello;
- * **Post-condizioni:** L'attore ha rimosso gli articoli selezionati dal carrello;
- * **Scenario principale:** L'attore rimuove gli articoli selezionati dal carrello premendo su un bottone.

4.1.27 UC2.3.10 - Invio ordine

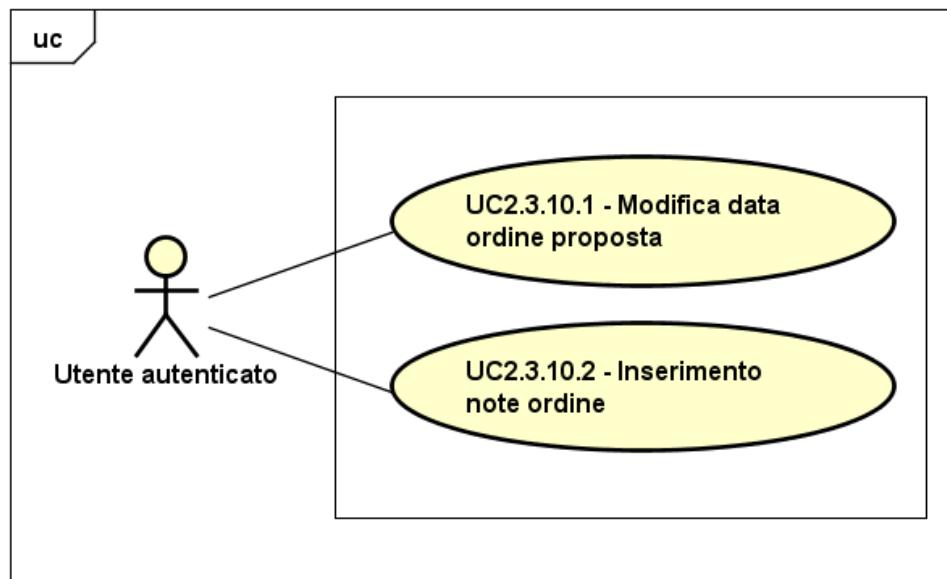


Figura 4.7: Use Case - UC2.3.10: Invio ordine

- * **Attore:** Utente autenticato;
 - * **Descrizione:** L'attore può inviare un ordine composto dagli articoli selezionati in carrello;
 - * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema e ha richiesto di inviare un ordine composto dagli articoli selezionati in carrello;
 - * **Post-condizioni:** L'attore ha inviato un ordine composto dagli articoli selezionati in carrello;
 - * **Scenario principale:**
 1. UC2.3.10.1 - Modifica data ordine proposta;
 2. UC2.3.10.2 - Inserimento note ordine.

4.1.28 UC2.3.10.1 - Modifica data ordine proposta

- * **Attore:** Utente autenticato;
 - * **Descrizione:** L'attore può modificare la data d'ordine proposta;
 - * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di inviare un ordine e il sistema permette la modifica della data d'ordine proposta;
 - * **Post-condizioni:** L'attore ha modificato la data d'ordine proposta;
 - * **Scenario principale:** L'attore modifica la data d'ordine proposta tramite una text box.

4.1.29 UC2.3.10.2 - Inserimento note ordine

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore può delle note per l'ordine;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di inviare un ordine e il sistema permette l'inserimento di note per l'ordine;
- * **Post-condizioni:** L'attore ha inserito le note per l'ordine;
- * **Scenario principale:** L'attore inserisce le note per l'ordine tramite una text area.

4.1.30 UC2.3.11 - Visualizzazione errore di mancata selezione di articoli

- * **Attore:** Utente autenticato;
- * **Descrizione:** L'attore preme sul bottone di rimozione articoli o invio ordine senza aver selezionato degli articoli in carrello;
- * **Pre-condizioni:** L'attore ha avviato l'applicazione, è riconosciuto dal sistema, ha richiesto di rimuovere degli articoli dal carrello o di inviare un ordine, senza aver effettivamente selezionato degli articoli in carrello;
- * **Post-condizioni:** L'attore ha visualizzato un messaggio d'errore relativo all'impossibilità di rimuovere gli articoli selezionati o inviare un ordine composto dagli articoli selezionati;
- * **Scenario principale:** L'attore visualizza un messaggio d'errore relativo all'impossibilità di rimuovere gli articoli selezionati o inviare un ordine composto dagli articoli selezionati.

4.2 Requisiti

In questa sezione sono presentati i requisiti della piattaforma *moviORDER*, dedotti dai diagrammi dei casi d'uso, dal capitolato fornito dal *tutor* aziendale e da raffinamenti di questo, ottenuti in seguito a brevi interazioni avvenute con il *tutor* aziendale. I requisiti sono stati classificati dallo stagista in base alla loro tipologia e importanza strategica. Sono stati individuati:

1. **requisiti funzionali**: descrivono funzionalità del sistema;
2. **requisiti qualitativi**: descrivono qualità del prodotto finale;
3. **requisiti di vincolo**: descrivono vincoli imposti dal *tutor* aziendale o dal dominio del problema.

L'importanza strategica dei requisiti individuati è stata negoziata con il *tutor* aziendale e si è deciso che ciascun requisito deve appartenere ad una delle seguenti categorie:

- * **obbligatorio**: non più negoziabile e irrinunciabile per il *tutor* aziendale;
- * **desiderabile**: rappresenta un valore aggiunto d'interesse riconosciuto da parte del *tutor* aziendale;
- * **facoltativo**: relativamente utile per gli scopi dell'applicazione.

Nelle seguenti sezioni vengono dettagliati i requisiti di *moviORDER*. In ogni sezione è presente una tabella contenente per ogni requisito:

- * **codice identificativo**: codice che identifica univocamente il requisito, utile a fini di tracciabilità;
- * **descrizione**: breve descrizione testuale del requisito;
- * **fonte**: fonte da cui è stato dedotto il requisito, utile a fini di tracciabilità. La fonte di un requisito può essere:
 - **caso d'uso**: se il requisito è stato dedotto da un caso d'uso specifico;
 - **capitolato**: se il requisito è stato dedotto dal capitolato d'appalto;
 - **stagista**: se il requisito è stato introdotto dallo stagista durante il periodo di *stage*;
 - **tutor**: se il requisito è stato introdotto dal *tutor* aziendale durante il periodo di *stage*.

4.2.1 Requisiti funzionali

Per facilitare la lettura della tabella, il codice dei requisiti macroscopici viene evidenziato con una dicitura in grassetto. Per requisiti macroscopici si intendono i requisiti che presentano sotto-requisiti.

Id Requisito	Descrizione	Fonti
RFO1	Se l'applicazione viene lanciata in assenza di connettività, deve essere visualizzato un messaggio d'errore di connettività assente e l'applicazione deve essere chiusa	Stagista
RFO2	L'utente non autenticato deve poter accedere all'applicazione	Capitolato UC1.1
RFO2.1	Per accedere, l'utente non autenticato deve poter inserire una <i>username</i>	Capitolato UC1.1.1
RFO2.2	Per accedere, l'utente non autenticato deve poter inserire una <i>password</i>	Capitolato UC1.1.2
RFO2.3	L'utente autenticato deve poter tentare l'accesso premendo su un bottone di login	Capitolato
RFO2.4	Se l'utente non autenticato inserisce una <i>username</i> non esistente e preme sul bottone di login, deve essere visualizzato un messaggio d'errore di <i>username</i> inesistente	UC1.2
RFO2.5	Se l'utente non autenticato inserisce una <i>password</i> non corrispondente alla <i>username</i> inserita e preme sul bottone di login, deve essere visualizzato un messaggio d'errore di <i>password</i> errata	UC1.2
RFO2.6	Se l'utente non autenticato preme sul bottone di login senza aver inserito le credenziali di accesso, deve essere visualizzato un messaggio che inviti l'utente ad inserire delle credenziali	UC1.2
RFO2.7	Se il <i>server</i> non risponde in fase di login, deve essere visualizzato un messaggio d'errore che notifichi l'utente autenticato che il <i>server</i> è down	Stagista
RFO3	Se l'utente non autenticato riesce ad accedere al sistema, deve essere visualizzata la <i>home page</i> dell'applicazione	Capitolato

Id Requisito	Descrizione	Fonti
RFO3.1	L'utente autenticato deve visualizzare sulla <i>home page</i> la propria ragione sociale	Capitolato
RFO3.2	L'utente autenticato deve poter visualizzare sulla <i>home page</i> il <i>tutorial</i> dell'applicazione	UC2.2
RFO3.3	L'utente autenticato deve poter effettuare il <i>logout</i> dall'applicazione premendo su un bottone	UC2.1
RFO3.4	L'utente autenticato deve poter visualizzare sulla <i>home page</i> i propri articoli nel carrello	Capitolato
RFO3.4.1	Se l'utente autenticato non presenta articoli in carrello, deve essere visualizzato un messaggio che notifichi l'utente che il carrello è vuoto	Stagista
RFO3.4.2	La visualizzazione di un articolo in carrello deve comprendere una check-box per la selezione dell'articolo	Capitolato
RFO3.4.3	La visualizzazione di un articolo in carrello deve comprendere la quantità dell'articolo	Capitolato
RFO3.4.4	La visualizzazione di un articolo in carrello deve comprendere il codice dell'articolo	Capitolato
RFO3.4.5	La visualizzazione di un articolo in carrello deve comprendere la descrizione dell'articolo	Capitolato
RFO3.5	L'utente autenticato deve poter aggiungere un nuovo articolo al carrello	Capitolato UC2.3.1
RFO3.5.1	Per aggiungere un nuovo articolo al carrello, l'utente autenticato deve inserire un codice articolo	Capitolato UC2.3.1.1
RFO3.5.2	L'utente autenticato deve poter decidere se inserire il codice articolo con la scansione di un codice a barre o manualmente	Capitolato UC2.3.1.1
RFO3.5.3	Se l'utente autenticato decide di inserire il codice articolo con la scansione di un codice a barre, l'applicazione deve accedere alla fotocamera e permettere la cattura del codice a barre	Capitolato UC2.3.1.3

Id Requisito	Descrizione	Fonti
RFO3.5.3.1	L'applicazione deve permettere la cattura di un codice a barre di tipo <i>EAN13</i>	Capitolato
RFO3.5.3.2	L'applicazione deve permettere la cattura di un codice a barre di tipo <i>EAN8</i>	Capitolato
RFO3.5.3.3	L'applicazione deve permettere la cattura di un codice a barre di tipo <i>CODE39</i>	Capitolato
RFO3.5.3.4	L'applicazione deve permettere la cattura di un codice a barre di tipo <i>UPC</i>	Capitolato
RFO3.5.3.5	L'applicazione deve permettere la cattura di un codice a barre di tipo <i>QR</i>	Capitolato
RFO3.5.3.6	L'utente autenticato deve poter annullare la scansione del codice a barre con il tasto indietro del sistema operativo	Stagista
RFO3.5.3.7	Se l'utente autenticato annulla la scansione del codice a barre, deve essere visualizzato un messaggio di annullamento della scansione da parte dell'utente	UC2.3.2
RFO3.5.3.8	Se la scansione del codice a barre fallisce, la fotocamera deve essere chiusa e deve essere visualizzato un messaggio d'errore di scansione fallita	UC2.3.2
RFO3.5.3.9	Se la scansione del codice a barre va a buon fine ma il codice ottenuto non corrisponde ad un articolo venduto dall'azienda, deve essere visualizzato un messaggio d'errore che notifichi l'utente autenticato del problema	UC2.3.2
RFO3.5.3.10	Se la scansione del codice a barre va a buon fine e il codice ottenuto corrisponde ad un articolo venduto dall'azienda ma l'articolo è già presente in carrello, deve essere visualizzato un messaggio che inviti l'utente autenticato a cambiare la quantità dell'articolo in carrello	UC2.3.2
RFO3.5.3.11	Se la scansione del codice a barre va a buon fine e il codice ottenuto corrisponde ad un articolo venduto dall'azienda, deve essere visualizzata la pagina di aggiunta articolo	Capitolato

Id Requisito	Descrizione	Fonti
RFO3.5.4	Se l'utente autenticato decide di inserire il codice articolo manualmente, deve essere visualizzata una text box per l'inserimento del codice	Capitolato UC2.3.1.2
RFO3.5.4.1	Se l'utente autenticato inserisce un codice articolo che non corrisponde ad un articolo venduto dall'azienda, deve essere visualizzato un messaggio d'errore che notifichi l'utente autenticato del problema	UC2.3.2
RFO3.5.4.2	Se l'utente autenticato inserisce un codice articolo che corrisponde ad un articolo venduto dall'azienda ma l'articolo è già presente in carrello, deve essere visualizzato un messaggio che inviti l'utente autenticato a cambiare la quantità dell'articolo in carrello	UC2.3.2
RFO3.5.4.3	Se l'utente autenticato inserisce un codice articolo che corrisponde ad un articolo venduto dall'azienda, deve essere visualizzata la pagina di aggiunta articolo	Capitolato
RFO3.5.5	Nella pagina di aggiunta articolo, l'utente autenticato deve poter visualizzare il codice articolo	Capitolato
RFO3.5.6	Nella pagina di aggiunta articolo, l'utente autenticato deve poter visualizzare la descrizione dell'articolo	Capitolato
RFO3.5.7	Nella pagina di aggiunta articolo, l'utente autenticato deve poter visualizzare le informazioni dell'articolo	Capitolato
RFO3.5.8	La pagina di aggiunta articolo deve proporre all'utente autenticato una quantità minima per l'articolo da ordinare	Capitolato
RFO3.5.9	Nella pagina di aggiunta articolo, l'utente autenticato deve poter inserire una quantità per l'articolo da ordinare	Capitolato UC2.3.1.4
RFO3.5.10	Nella pagina di aggiunta articolo, l'utente autenticato deve poter inserire delle note per l'articolo da ordine	Capitolato UC2.3.1.5

Id Requisito	Descrizione	Fonti
RFO3.5.11	Nella pagina di aggiunta articolo, l'utente autenticato deve poter annullare l'inserimento dell'articolo	Capitolato
RFO3.5.12	Nella pagina di aggiunta articolo, l'utente autenticato deve poter confermare l'inserimento dell'articolo	Capitolato
RFO3.5.13	Se l'utente autenticato conferma l'inserimento dell'articolo ed ha inserito una quantità errata (troppo bassa, troppo alta o che non soddisfa uno step impostato dall'azienda), deve essere visualizzato un messaggio d'errore di errata quantità inserita	UC2.3.2
RFO3.5.14	Se l'utente autenticato conferma l'inserimento dell'articolo e i dati inseriti sono corretti, deve essere visualizzata la <i>home page</i> dell'applicazione e il nuovo articolo deve essere presente in carrello	Capitolato
RFO3.5.15	Se l'utente autenticato annulla l'inserimento dell'articolo, deve essere visualizzata la <i>home page</i> dell'applicazione e il carrello non deve aver subito cambiamenti	Capitolato
RFO3.5.16	Se viene persa la connettività nella pagina di aggiunta articolo, deve essere visualizzato un messaggio che invita l'utente a chiudere l'applicazione per connettività assente	Stagista
RFO3.6	L'utente autenticato deve poter selezionare un articolo in carrello premendo sulla relativa check-box	Capitolato UC2.3.5
RFO3.7	L'utente autenticato deve poter deselezionare un articolo in carrello selezionato premendo sulla relativa check-box	Capitolato UC2.3.6
RFO3.8	L'utente autenticato deve poter selezionare tutti gli articoli in carrello premendo su un bottone di seleziona/deseleziona tutti presente sulla <i>home page</i>	Capitolato UC2.3.7

Id Requisito	Descrizione	Fonti
RFO3.9	L'utente autenticato deve poter deselezionare tutti gli articoli in carrello premendo su un bottone di seleziona/deseleziona tutti presente sulla <i>home page</i>	Capitolato UC2.3.8
RFO3.10	L'utente autenticato deve poter modificare un articolo in carrello premendo sopra la visualizzazione dell'articolo sul carrello	Capitolato UC2.3.3
RFO3.10.1	Se l'utente autenticato preme sulla visualizzazione di un articolo in carrello, deve essere richiesta la conferma di modifica tramite la visualizzazione di un <i>dialog</i>	Stagista
RFO3.10.2	Se l'utente autenticato accetta di modificare l'articolo sul <i>dialog</i> di conferma, deve essere aperta la pagina di modifica articolo	Stagista
RFO3.10.3	Se l'utente autenticato rifiuta di modificare l'articolo sul <i>dialog</i> di conferma, deve essere visualizzata la <i>home page</i> senza cambiamenti sugli articoli in carrello	Stagista
RFO3.10.4	Nella pagina di modifica articolo, l'utente autenticato deve poter visualizzare il codice articolo dell'articolo selezionato	Capitolato
RFO3.10.5	Nella pagina di modifica articolo, l'utente autenticato deve poter visualizzare la descrizione dell'articolo selezionato	Capitolato
RFO3.10.6	Nella pagina di modifica articolo, l'utente autenticato deve poter visualizzare le informazioni dell'articolo selezionato	Capitolato
RFO3.10.7	La pagina di modifica articolo deve proporre all'utente autenticato la quantità precedentemente inserita per l'articolo selezionato	Capitolato
RFO3.10.8	Nella pagina di modifica articolo, l'utente autenticato deve poter cambiare la quantità per l'articolo selezionato	Capitolato UC2.3.3.1
RFO3.10.9	La pagina di modifica articolo deve proporre all'utente autenticato le note precedentemente inserite per l'articolo selezionato	Capitolato

Id Requisito	Descrizione	Fonti
RFO3.10.10	Nella pagina di modifica articolo, l'utente autenticato deve poter modificare le note per l'articolo selezionato	Capitolato UC2.3.3.1
RFO3.10.11	Nella pagina di modifica articolo, l'utente autenticato deve poter annullare la modifica dell'articolo	Capitolato
RFO3.10.12	Nella pagina di modifica articolo, l'utente autenticato deve poter confermare la modifica dell'articolo	Capitolato
RFO3.10.13	Se l'utente autenticato conferma la modifica dell'articolo ed ha inserito una quantità errata (troppo bassa, troppo alta o che non soddisfa uno step impostato dall'azienda), deve essere visualizzato un messaggio d'errore di errata quantità inserita	UC2.3.4
RFO3.10.14	Se l'utente autenticato conferma la modifica dell'articolo e i dati modificati sono corretti, deve essere visualizzata la <i>home page</i> dell'applicazione e l'articolo modificato deve essere presente in carrello con le modifiche apportate	Capitolato
RFO3.10.15	Se l'utente autenticato annulla la modifica dell'articolo, deve essere visualizzata la <i>home page</i> dell'applicazione e l'articolo selezionato non deve essere stato modificato	Capitolato
RFO3.10.16	Se viene persa la connettività nella pagina di modifica, deve essere visualizzato un messaggio che inviti l'utente a chiudere l'applicazione per connettività assente	Stagista
RFO3.11	L'utente autenticato deve poter rimuovere gli articoli selezionati sul carrello premendo su un bottone presente sulla <i>home page</i>	Capitolato UC2.3.9
RFO3.11.1	Se l'utente autenticato preme sul bottone di rimozione articoli senza aver selezionato alcun articolo, deve essere visualizzato un messaggio che inviti l'utente a selezionare degli articoli	UC2.3.11

Id Requisito	Descrizione	Fonti
RFO3.11.2	Se l'utente autenticato preme sul bottone di rimozione articoli avendo selezionato degli articoli sul carrello, deve essere richiesta la conferma di eliminazione tramite la visualizzazione di un <i>dialog</i>	Stagista
RFO3.11.3	Se l'utente autenticato conferma l'eliminazione sul <i>dialog</i> di conferma, gli articoli precedentemente selezionati devono essere rimossi dal carrello	Capitolato
RFO3.11.4	Se l'utente autenticato rifiuta l'eliminazione sul <i>dialog</i> di conferma, il carrello non deve subire cambiamenti	Capitolato
RFO3.12	L'utente autenticato deve poter inviare un ordine composto dagli articoli selezionati sul carrello premendo su un bottone presente sulla <i>home page</i>	Capitolato UC2.3.10
RFO3.12.1	Se l'utente autenticato preme sul bottone di invio ordine senza aver selezionato alcun articolo, deve essere visualizzato un messaggio che inviti l'utente a selezionare degli articoli	UC2.3.11
RFO3.12.2	Se l'utente autenticato preme sul bottone di invio ordine avendo selezionato degli articoli sul carrello, deve essere visualizzato il <i>modal</i> di invio ordine	Stagista
RFO3.12.3	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter visualizzare il codice cliente	Capitolato
RFO3.12.4	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter visualizzare la descrizione cliente	Capitolato
RFO3.12.5	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter visualizzare il codice del documento da generare per l'ordine	Capitolato
RFO3.12.6	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter visualizzare la descrizione del documento da generare per l'ordine	Capitolato

Id Requisito	Descrizione	Fonti
RFO3.12.7	Il <i>modal</i> di invio ordine deve proporre all'utente autenticato la data corrente come data d'ordine	Capitolato
RFO3.12.8	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter modificare la data d'ordine proposta	Capitolato UC2.3.10.1
RFO3.12.9	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter inserire delle note per l'ordine	Capitolato UC2.3.10.2
RFO3.12.10	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter confermare l'invio dell'ordine	Capitolato
RFO3.12.11	Nel <i>modal</i> di invio ordine, l'utente autenticato deve poter annullare l'invio dell'ordine	Capitolato
RFO3.12.12	Se l'utente autenticato decide di annullare l'invio dell'ordine, il <i>modal</i> di invio ordine deve essere chiuso	Stagista
RFO3.12.13	Se l'utente autenticato decide di confermare l'invio dell'ordine, deve essere richiesta la conferma di invio ordine tramite un <i>dialog</i>	Stagista
RFO3.12.14	Se l'utente autenticato conferma l'invio dell'ordine nel <i>dialog</i> di conferma e l'ordine è andato a buon fine, deve essere visualizzato un messaggio di ordine inviato con successo e una mail di conferma deve essere inviata sia all'utente che all'azienda	Capitolato
RFO3.12.14.1	La mail inviata dal sistema al momento dell'invio dell'ordine deve contenere in oggetto la data di registrazione dell'ordine	Capitolato
RFO3.12.14.2	La mail inviata dal sistema al momento dell'invio dell'ordine deve contenere in oggetto il codice azienda dell'utente che ha effettuato l'ordine	Capitolato
RFO3.12.14.3	La mail inviata dal sistema al momento dell'invio dell'ordine deve contenere in oggetto la tipologia di documento generato con la registrazione dell'ordine	Capitolato

Id Requisito	Descrizione	Fonti
RFO3.12.14.4	La mail inviata dal sistema al momento dell'invio dell'ordine deve contenere nel corpo il codice cliente del cliente che ha effettuato l'ordine	Capitolato
RFO3.12.14.5	La mail inviata dal sistema al momento dell'invio dell'ordine deve contenere nel corpo una tabella contenente i dati degli articoli ordinati	Capitolato
RFO3.12.14.6	Per ogni articolo in tabella deve essere indicata la quantità ordinata	Capitolato
RFO3.12.14.7	Per ogni articolo in tabella deve essere indicato il codice articolo	Capitolato
RFO3.12.14.8	Per ogni articolo in tabella deve essere indicata la descrizione dell'articolo	Capitolato
RFO3.12.14.9	La mail inviata dal sistema al momento dell'invio dell'ordine deve contenere nel corpo l'avviso di non rispondere alla mail	Capitolato
RFO3.12.15	Se l'utente autenticato conferma l'invio dell'ordine nel <i>dialog</i> di conferma e l'ordine non è andato a buon fine, deve essere visualizzato un messaggio d'errore di mancato invio dell'ordine	Capitolato
RFO3.12.16	Se l'utente autenticato annulla l'invio dell'ordine nel <i>dialog</i> di conferma, il <i>dialog</i> di conferma deve essere chiuso	Stagista
RFO3.13	Se viene persa la connettività nella <i>home page</i> , deve essere visualizzato un messaggio che invita l'utente a chiudere l'applicazione per connettività assente	Stagista

Tabella 4.1: Tabella del tracciamento dei requisiti funzionali

4.2.2 Requisiti qualitativi

Id Requisito	Descrizione	Fonti
RQO1	L'applicazione deve essere fornita di manuale utente	<i>Tutor</i>
RQD2	L'applicazione deve essere fornita di manuale sviluppatore	<i>Tutor</i>

Tabella 4.2: Tabella del tracciamento dei requisiti qualitativi

4.2.3 Requisiti di vincolo

Id Requisito	Descrizione	Fonti
RVO1	L'applicazione deve essere sviluppata con ambiente multipiattaforma (preferibilmente <i>PhoneGap</i> o <i>Xamarin</i>)	Capitolato
RVO2	L'applicazione deve utilizzare il <i>database</i> standard di <i>VisionENTERPRISE</i> , senza modifica alla sua struttura	Capitolato
RVO3	L'applicazione deve interfacciarsi con la fotocamera per la lettura di codici a barre	Capitolato
RVO4	L'applicazione deve funzionare completamente <i>online</i> , nessun dato deve essere memorizzato in locale e nessun <i>database</i> deve essere sincronizzato	Capitolato
RVD5	L'applicazione deve utilizzare un <i>database SQL Server</i> lato <i>web</i>	Capitolato
RVD6	L'applicazione deve supportare codici a barre bidimensionali (<i>QRcode</i>)	Capitolato
RVF7	L'applicazione deve condividere i dati di login con l'applicazione <i>moviDOC</i>	Capitolato
RVF8	L'applicazione deve avere lo stesso stile di interfaccia di <i>moviDOC</i>	Capitolato
RVF9	L'applicazione deve permettere la firma digitale del cliente a fine ordine	Capitolato

Tabella 4.3: Tabella del tracciamento dei requisiti di vincolo

4.2.4 Riepilogo requisiti

I requisiti sono di seguito raggruppati in una tabella riepilogativa, in modo da comprendere la grandezza del progetto in termini di requisiti.

Tipologia	Obbligatorio	Desiderabile	Facoltativo	Totale
Funzionale	102	0	0	102
Qualitativo	1	1	0	2
Di vincolo	4	2	3	9
Totale	107	3	3	113

Tabella 4.4: Riepilogo requisiti

4.2.5 Validazione dei requisiti

Al termine dell'analisi dei requisiti, lo stagista ha validato i requisiti tramite delle interazioni con il *tutor* aziendale. Nello specifico ci si è assicurati che i requisiti individuati rappresentassero il sistema richiesto dal *tutor* aziendale. Lo stagista ha fatto in modo aderire allo standard *IEEE 830-1998* per la scrittura dei requisiti. Lo standard *IEEE 830-1998* individua otto qualità essenziali per i requisiti:

- * assenza di ambiguità;
- * correttezza;
- * completezza;
- * verificabilità;
- * consistenza;
- * modificabilità;
- * tracciabilità;
- * ordinamento per rilevanza.

Capitolo 5

Progettazione

In questo capitolo vengono presentati gli aspetti più interessanti della progettazione di *moviORDER*. Il capitolo inizia con la descrizione dell’architettura generale della piattaforma per poi entrare nel dettaglio delle varie componenti che la costituiscono.

5.1 Architettura generale

L’obiettivo di una buona progettazione è il soddisfacimento dei requisiti mediante un sistema di qualità, ottenibile tramite la definizione di una buona architettura logica del prodotto, che presenti componenti dalle specifiche chiare e coese, che sia realizzabile con risorse e costi fissati e che abbia una struttura che faciliti i cambiamenti futuri. In quest’ottica *moviORDER* presenta un’architettura *client-server*, dove il *client* è l’applicazione installata sul dispositivo (*Android* o *iOS*) dell’utente finale, e il *server* è un *server web Apache Tomcat* installato su un *server Azure* di proprietà di VisioneImpresa. L’applicazione si connette al *server* per la fruizione di un’*API* che permette l’accesso a *database* contenenti i dati di *moviORDER*. Viene di seguito fornita una figura illustrativa dell’architettura generale di *moviORDER*.

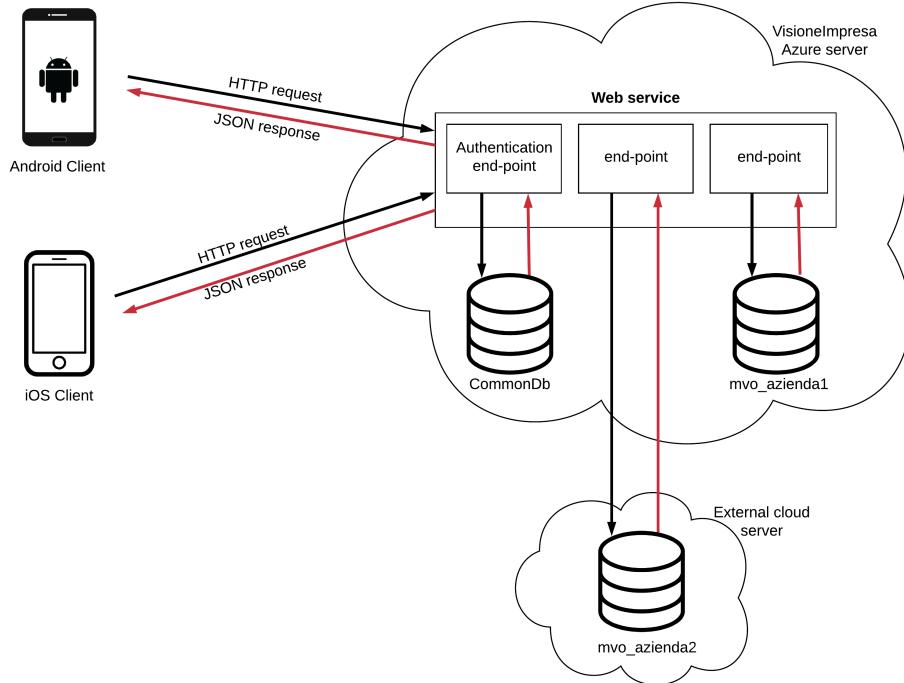


Figura 5.1: Architettura generale di *moviORDER*

L'applicazione comunica con il *server* tramite l'invio di richieste *HTTP*. Sul *server* è installato un servizio *web* che si occupa di elaborare tali richieste tramite oggetti *servlet Java*, i quali eseguono delle query sul *database* e rispondono, sulla base dei risultati ottenuti, tramite stringhe codificate in formato *JSON*. Per la costruzione delle risposte, il servizio *web* interroga i seguenti *database SQL Server*:

- * *CommonDb*: *database* locale al *server Azure* di VisioneImpresa e contenente i dati di autenticazione degli utenti di *moviORDER*. Ogni qualvolta che un'azienda acquista il servizio, vengono inserite nel *CommonDb* le credenziali di accesso di tutti gli utenti dell'azienda che desiderano utilizzare l'applicazione;
- * *mvo_aziendaNomeAzienda*: *database* contenente i dati utili alla gestione degli ordini presso l'azienda *NomeAzienda*. All'interno del *server Azure* è presente un *database* di questo tipo per ogni azienda che usufruisce del servizio e che ha deciso di affidare la gestione completa dei propri ordini a VisioneImpresa. Per le aziende che preferiscono invece gestire internamente i propri ordini, tale *database* è contenuto nei *server cloud* delle rispettive aziende. Ne consegue che il servizio *web* deve essere in grado di connettersi a *database* locali o remoti.

Una descrizione più approfondita dei dati contenuti in questi *database* è presente in sezione 5.3.

5.1.1 Architettura front end

Sul *client*, ovvero l'applicazione installata sul dispositivo dell'utente utilizzatore, è presente il pattern architettonale *MVP* (Model View Presenter). Tale pattern presenta componenti distribuite, infatti la *view* e il *presenter* si trovano sul dispositivo, mentre il *model* si trova sul *server Azure* di VisioneImpresa o sul *server cloud* di un'azienda cliente. Nello specifico, la *view* è la *GUI* dell'applicazione, il *presenter* è la logica applicativa della stessa e il *model* è il *database* contenente i dati utili alla gestione degli ordini. È stato scelto un *MVP* in quanto il *model* interagisce solamente con il *presenter* e non può modificare la *view* come invece accade per il pattern *MVC* (*Model View Controller*). Nella seguente figura è possibile notare le differenze concettuali tra i due pattern.

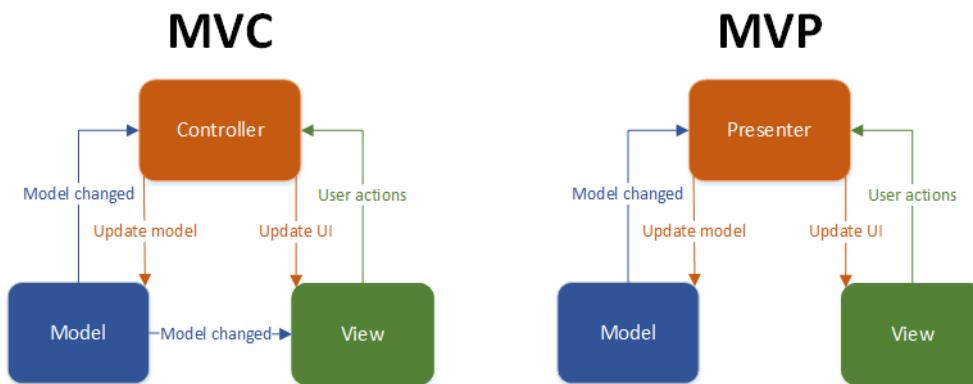


Figura 5.2: Differenze tra pattern *MVC* e *MVP*

Viene di seguito presentata una figura illustrativa di come il design pattern *MVP* è stato istanziato nell'architettura del *front end* di *moviORDER*.

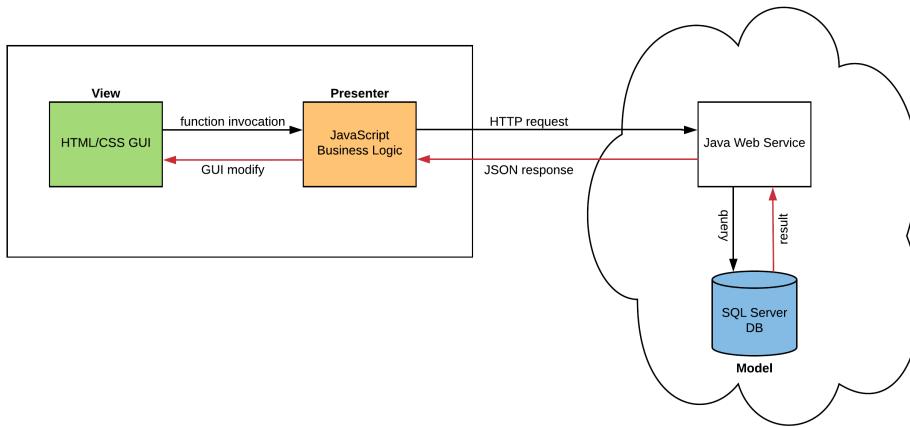


Figura 5.3: Architettura *front end*

Nello specifico, il flusso del pattern è il seguente:

- * l'utente interagisce con la *view* eseguendo delle operazioni sull'interfaccia dell'applicazione;
- * il *presenter* capta le interazioni e, sulla base di queste, richiede la lettura/scrittura di dati sul *model* tramite l'invio di richieste *HTTP* ad un servizio *web*;
- * il servizio *web* legge/scrive sul *model* a seconda della richiesta ricevuta e prepara ed invia una risposta al *presenter*;
- * il *presenter* riceve la risposta, la elabora e modifica la *view* di conseguenza.

Viene di seguito presentato un diagramma di sequenza esemplificativo del pattern MVP istanziato nel contesto del progetto.

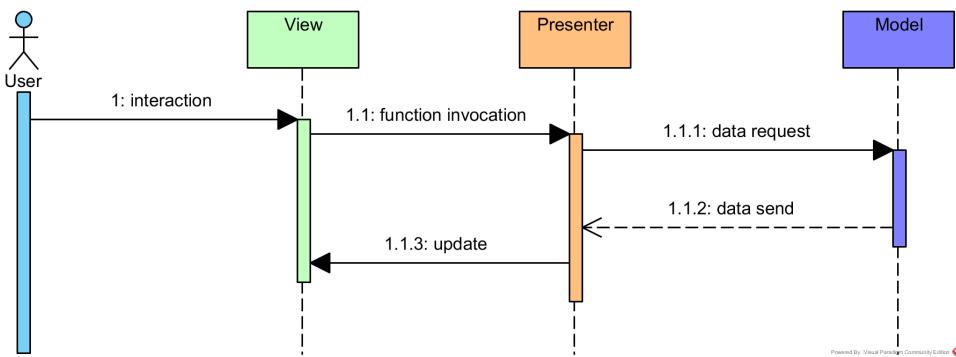


Figura 5.4: Diagramma di sequenza del pattern MVP

Importanti vantaggi nell'utilizzo del pattern *MVP* sono:

- * possibilità di utilizzare lo stesso *model* da parte di *view* differenti;
- * semplicità nell'aggiunta di nuovi tipi di *client*, e quindi di applicazioni: è sufficiente scrivere un *presenter* e una *view* per ognuna delle nuove applicazioni. *MVP* permette quindi un disaccoppiamento tra logica applicativa e *database* sottostante.

5.1.2 Architettura back end

Il *server* presenta un'architettura a strati. In questo pattern i componenti sono organizzati in strati orizzontali e ogni strato possiede specifici ruoli e responsabilità nel contesto dell'applicazione. Nel caso di *moviORDER*, il pattern è stato diviso nei seguenti strati:

- * ***business layer***: contiene gli oggetti *servlet* del servizio *web*, i quali si occupano di captare le richieste *HTTP* provenienti dal client, di leggere o scrivere sul *database* di conseguenza, e di fornire delle risposte;
- * ***persistence layer***: contiene le classi del servizio *web* che permettono agli oggetti *servlet* di accedere al *database* dell'applicazione;
- * ***database layer***: contiene i *database* dell'applicazione.

Viene di seguito presentata una figura illustrativa di come il design pattern *layered architecture* è stato istanziato nell'architettura del *back end* di *moviORDER*.

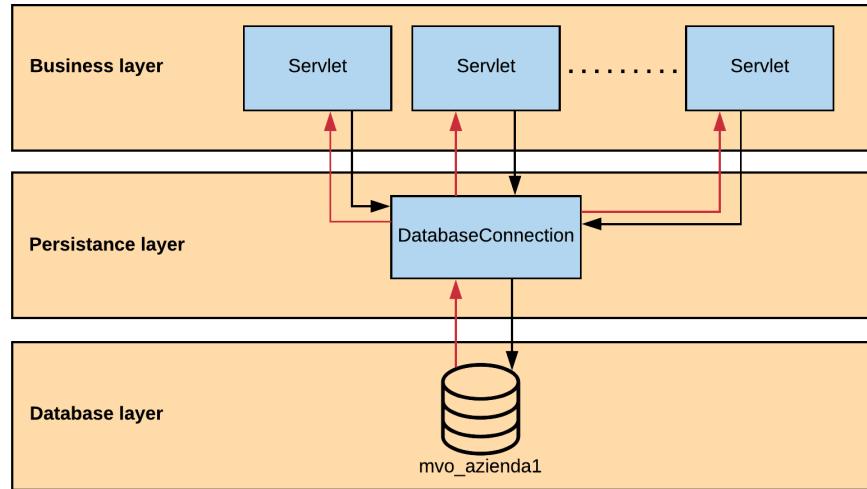


Figura 5.5: Architettura a strati del *back end*

Uno dei vantaggi più importanti dell'architettura a strati è la separazione delle responsabilità tra i componenti. Un componente all'interno di uno specifico strato può eseguire solamente compiti che spettano a tale strato. Questo tipo di classificazione facilita lo sviluppo, il *testing* e la manutenzione del *back end* di *moviORDER*.

5.2 Progettazione servizio web

Il seguente diagramma dei *package* rappresenta la struttura del servizio *web* di *moviORDER*.

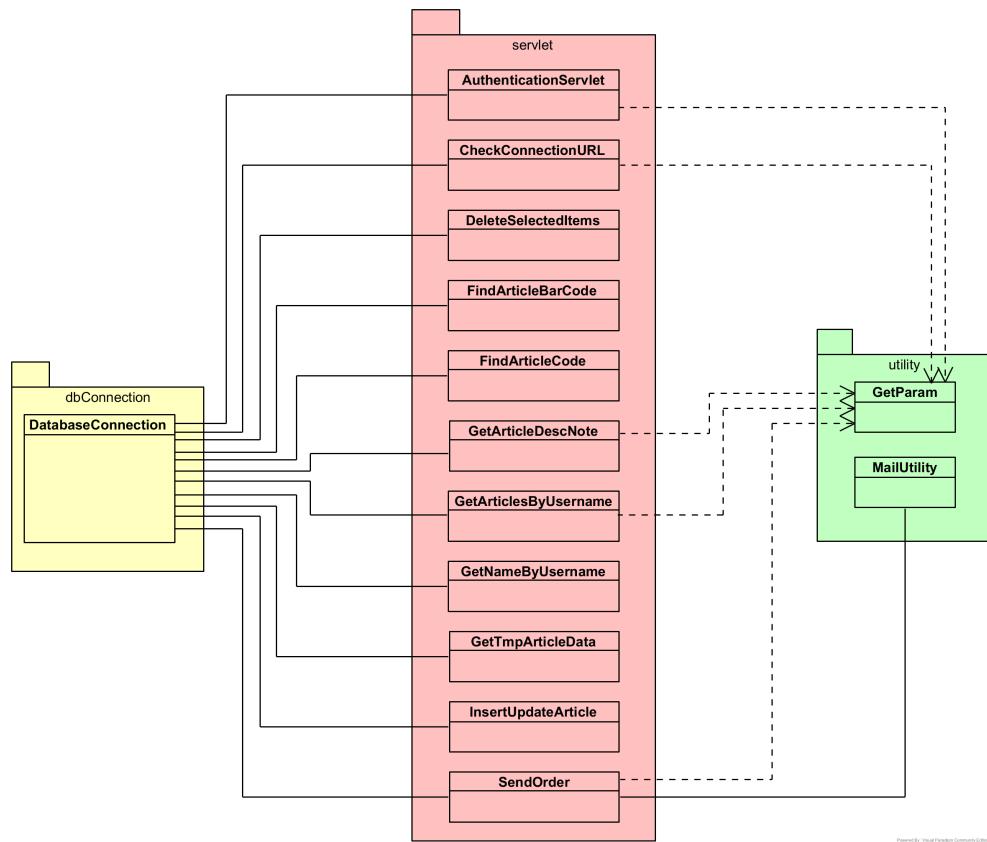


Figura 5.6: Diagramma dei *package* del servizio *web*

Come si può vedere dal diagramma, il servizio è costituito da tre *package*:

- * *dbConnection*: contiene classi atte alla gestione della connessione con un *database*. Il *package* viene utilizzato per permettere agli oggetti *servlet* di connettersi a *database SQL Server* locali o remoti;
- * *servlet*: contiene le classi che definiscono gli oggetti *servlet* del servizio. Questi oggetti si occupano di captare le richieste *HTTP* provenienti dal *client* e di rispondere a queste tramite stringhe in formato *JSON*;
- * *utility*: contiene le classi utilità del servizio. Queste classi facilitano i compiti che gli oggetti *servlet* devono eseguire.

Una descrizione più approfondita di come tali *package* sono stati utilizzati nella codifica del servizio *web* è presente in sezione 6.1.

5.2.1 Package servlet

Essendo il *package servlet* il più articolato, merita una descrizione più approfondita. Il seguente diagramma delle classi rappresenta la struttura del *package servlet*.

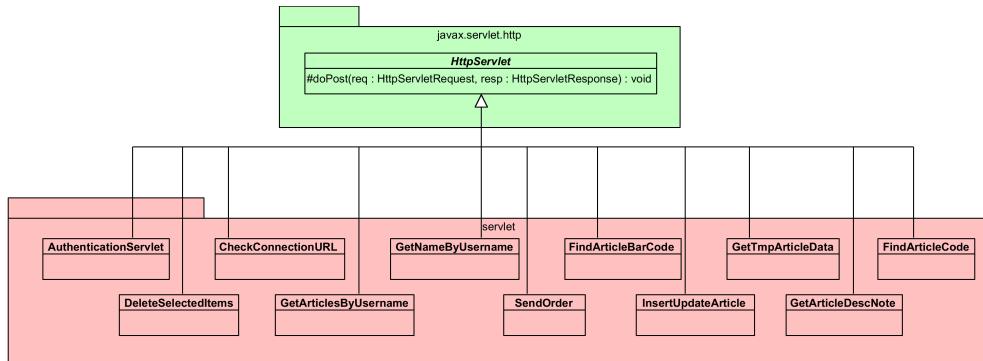


Figura 5.7: Diagramma delle classi del *package servlet*

Come si può vedere dal diagramma, ogni classe *servlet* concreta eredita dalla classe astratta *HttpServlet*. Inoltre, ogni *servlet* concreto definisce il metodo *doPost()*, il quale permette di definire il comportamento del *servlet* alla ricezione di richieste *HTTP POST*. Una descrizione di come queste classi sono state implementate nella pratica è presente in sezione 6.1.

5.3 Progettazione database

Come detto precedentemente, i dati di *moviORDER* sono raggruppati all'interno di due *database*. Il *database* *CommonDb* contiene i dati di autenticazione degli utenti di *moviORDER* e le stringhe di connessione ai rispettivi *database* aziendali (ogni azienda possiede un proprio *database*), mentre il *database* *mvo_aziendaNomeAzienda* contiene tutti i dati utili alla gestione degli ordini presso l'azienda *NomeAzienda*. Più precisamente, in fase di autenticazione le credenziali dell'utente vengono cercate all'interno del *CommonDb* e, in caso di corrispondenza, viene prelevata la stringa di connessione al *database* dell'azienda presso cui l'utente è cliente. In seguito questa stringa viene utilizzata per collegare l'applicazione al *database* corretto e quindi permettere all'utente di visualizzare solamente i dati sugli articoli venduti dalla propria azienda. Il *database* *CommonDb* presenta le seguenti tabelle:

- * **Users:** tabella contenente i dati di autenticazione degli utenti di *moviORDER*. La tabella presenta i seguenti campi:
 - *UserName* (chiave primaria): è il nome utente per accedere all'applicazione. Viene reso univoco poiché costituito dalla concatenazione della partita IVA dell'azienda presso cui l'utente è cliente e un codice auto-incrementante assegnato al cliente al momento della consegna dell'applicazione;
 - *Password*: è la *password* per accedere all'applicazione. La coppia *username/password* viene assegnata all'utente al momento della consegna dell'applicazione;

- *CodAzienda* (chiave esterna): è l’identificativo univoco dell’azienda presso cui l’utente è cliente. Questo campo presenta un vincolo d’integrità referenziale con il campo *CodAzienda* della tabella **Aziende**;
 - *EmailU*: è l’indirizzo e-mail dell’utente;
 - *Bloccato*: è un flag che vale 1 se l’utente è stato bloccato dall’azienda, oppure 0 se l’utente è operativo e quindi può utilizzare l’applicazione.
- * **Aziende**: tabella contenente le stringhe di connessione ai *database* aziendali di tutte le aziende registrate al servizio *moviORDER*. Sono presenti inoltre i parametri di configurazione del *server SMTP* di ogni azienda. Questo *server* viene utilizzato da *moviORDER* per inviare le mail di conferma dei vari ordini registrati. La tabella presenta i seguenti campi:
- *CodAzienda* (chiave primaria): è l’identificativo univoco dell’azienda;
 - *Path*: è la stringa di connessione al *database* aziendale dell’azienda;
 - *EmailA*: è l’indirizzo e-mail aziendale dell’azienda;
 - *Host*: è l’indirizzo della macchina dove è installato il *server SMTP* dell’azienda;
 - *Post*: è la porta su cui è installato il *server SMTP* dell’azienda;
 - *Username*: è il nome utente per accedere al *server SMTP* dell’azienda;
 - *Password*: è la *password* per accedere al *server SMTP* dell’azienda.

Viene di seguito presentato il diagramma *Entity Relationship* del *database CommonDb*. Le chiavi primarie sono state sottolineate, mentre le chiavi esterne sono state scritte in corsivo.

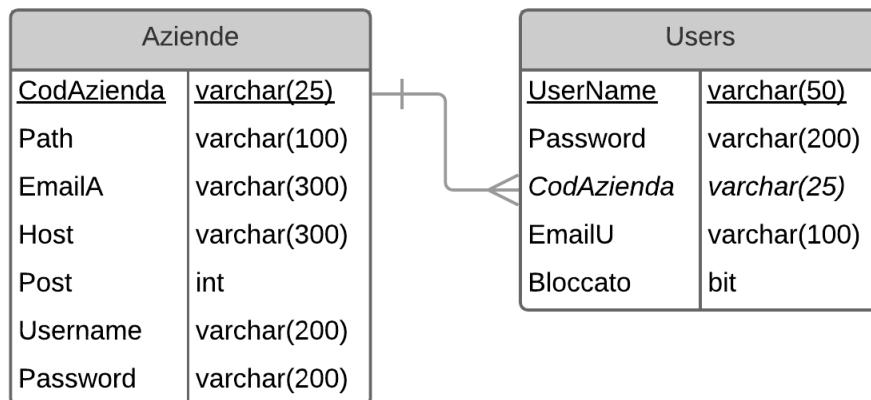


Figura 5.8: Diagramma ER del *database CommonDb*

Il database *mvo_aziendaNomeAzienda* presenta le seguenti tabelle:

- * **Art:** tabella contenente i dati sugli articoli venduti dall'azienda *NomeAzienda*. La tabella presenta i seguenti campi:

- *Id_Art*: è un codice auto-incrementante assegnato al record;
- *CodArt* (chiave primaria): è il codice articolo dell'articolo;
- *DesArt*: è il nome dell'articolo;
- *Note*: sono le note presenti in *database* per l'articolo;
- *QtaMin*: è la minima quantità ordinabile per l'articolo;
- *QtaMul*: è lo step di quantità ordinabile per l'articolo. Ad esempio se *QtaMul* è 2, significa che è possibile ordinare 2, 4, 6 pezzi dell'articolo, e così via.

- * **ArtAlias:** tabella contenente i codici a barre degli articoli venduti dall'azienda. La tabella presenta i seguenti campi:

- *Id_ArtAlias*: è un codice auto-incrementante assegnato al record;
- *CodArt* (chiave primaria, chiave esterna): è il codice articolo dell'articolo. Questo codice presenta un vincolo d'integrità referenziale con il campo *CodArt* della tabella **Art**;
- *Alias* (chiave primaria): è il codice a barre dell'articolo. Fa parte della chiave primaria, insieme a *CodArt*, poiché è possibile che un articolo abbia più codici a barre.

- * **DocRig:** tabella contenente i dati d'ordine degli articoli che sono stati ordinati presso l'azienda. La tabella presenta i seguenti campi:

- *Id_DocRig* (chiave primaria): è un codice auto-incrementante assegnato al record;
- *Id_DocTes* (chiave esterna): è il codice della fattura a cui il prodotto ordinato appartiene. Questo campo presenta un vincolo di integrità referenziale con il campo *Id_DocTes* della tabella **DocTes**;
- *Username* (chiave esterna): è il nome utente dell'utente che ha ordinato l'articolo. Questo campo presenta un vincolo d'integrità referenziale con il campo *UserID* della tabella **Users**;
- *CodArt* (chiave esterna): è il codice articolo dell'articolo ordinato. Questo campo presenta un vincolo d'integrità referenziale con il campo *CodArt* della tabella **Art**;
- *Quantita*: è la quantità ordinata dell'articolo;
- *Note*: sono le note che l'utente ha inserito per l'articolo quando l'ha aggiunto al carrello.

- * **DocTes:** tabella contenente i dati delle fatture degli ordini che sono stati registrati prezzo l'azienda. La tabella presenta i seguenti campi:

- *Id_DocTes* (chiave primaria): è un codice auto-incrementante assegnato al record;

- *CodDoc* (chiave esterna): è un codice che rappresenta la tipologia di fattura emessa dall’azienda. Questo campo presenta un vincolo d’integrità referenziale con il campo *CodDoc* della tabella **Users**;
- *CodCliFor*: è il nome utente dell’utente che ha eseguito l’ordine presso l’azienda;
- *DataDoc*: è la data dell’ordine;
- *Note*: sono le note inserite dall’utente in fase di invio ordine;
- *Status*: è un flag che vale 0 nel momento in cui il record viene memorizzato in tabella, 1 nel momento in cui inizia l’importazione del record nel gestionale di VisioneImpresa e 2 quando l’importazione è terminata.

* **TmpRig**: tabella contenente i dati d’ordine degli articoli non ancora ordinati presso l’azienda. Si tratta di articoli inseriti nel carrello degli utenti, ma che non sono ancora stati ordinati. La tabella presenta i seguenti campi:

- *Id_TmpRig* (chiave primaria): è un codice auto-incrementante assegnato al record;
- *Username* (chiave esterna): è il nome utente dell’utente che presenta l’articolo in carrello. Questo campo presenta un vincolo d’integrità referenziale con il campo *UserID* della tabella **Users**;
- *CodArt* (chiave esterna): è il codice articolo dell’articolo. Questo campo presenta un vincolo d’integrità referenziale con il campo *CodArt* della tabella **Art**;
- *Quantita*: è la quantità da ordinare inserita dall’utente;
- *Note*: sono le note che l’utente ha inserito per l’articolo quando l’ha aggiunto al carrello.

* **Users**: tabella contenente le informazioni anagrafiche degli utenti clienti dell’azienda. La tabella presenta i seguenti campi:

- *UserID* (chiave primaria): è il nome utente dell’utente;
- *DesCliFor*: è una breve descrizione dell’utente;
- *Indirizzo*: è l’indirizzo di residenza dell’utente;
- *Localita*: è la località di residenza dell’utente;
- *CodProv*: è il codice della provincia di residenza dell’utente;
- *CodNazione*: è il codice della nazione di residenza dell’utente;
- *CodDoc*: è il codice della fattura che deve essere emessa quando l’utente effettua un ordine;
- *DesDoc*: è una descrizione della fattura che deve essere emessa quando l’utente effettua un ordine.

Viene di seguito presentato il diagramma *Entity Relationship* del database *mvo_aziendaNomeAzienda*. Le chiavi primarie sono state sottolineate, mentre le chiavi esterne sono state scritte in corsivo.

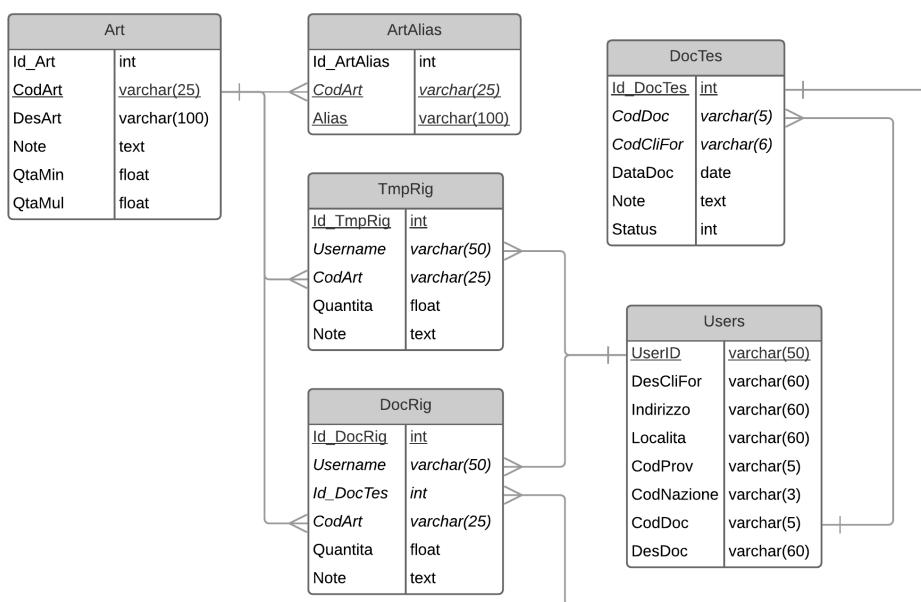


Figura 5.9: Diagramma ER del database `mvo_aziendaNomeAzienda`

Capitolo 6

Codifica

In questo capitolo vengono trattati gli aspetti più interessanti della codifica dell'applicazione *moviORDER*. In particolare, il capitolo è stato diviso in sezioni che trattano la codifica di:

1. **servizio web;**
2. **logica applicativa;**
3. **interfaccia grafica.**

6.1 Servizio web

Per lo sviluppo del servizio *web* che permette all'applicazione di accedere al *database* presente sul *server cloud* di VisioneImpresa si è utilizzato il linguaggio *Java* e, nello specifico, gli oggetti *servlet*. Per permettere agli oggetti *servlet* di interagire con il *database* si sono dovuti utilizzare i driver *JDBC* per *SQL Server*, in quanto *moviORDER* è basata su tale *database*.

6.1.1 Servlet

In questa sezione viene presentato l'utilizzo degli oggetti *servlet* nella realizzazione del servizio *web*. Le classi che implementano gli oggetti *servlet* appartengono al *package* *servlet*.

6.1.1.1 Struttura di un oggetto servlet

Un oggetto *servlet* è una classe *Java* che eredita dalla classe *HttpServlet*, appartenente al *package* *javax.servlet.http*. *HttpServlet* è una classe astratta che può essere estesa per creare un *servlet* *HTTP* utilizzabile per un sito *web*. Nel progetto realizzato, il *servlet* è stato utilizzato per acquisire richieste *HTTP POST*, per elaborarle interrogando il *database* sul *server cloud* di VisioneImpresa, e per rispondere ad esse tramite una stringa in formato *JSON*. Ogni *servlet* del servizio *web* implementa il metodo *protected void doPost(HttpServletRequest req, HttpServletResponse resp)*. Tale metodo viene chiamato dal *server* per permettere all'oggetto *servlet* di acquisire una richiesta *POST* da parte di un *client*. Nel caso del progetto, il *client* è la logica applicativa dell'applicazione *moviORDER* e il *server* è *Apache Tomcat*.

HttpServletRequest è la classe Java che rappresenta le richieste *HTTP* che possono essere inviate all'oggetto *servlet*. Tramite opportuni metodi è possibile accedere alle informazioni contenute nella specifica richiesta. Tramite il metodo *String getParameter(String name)* è possibile, data una stringa rappresentante un parametro della richiesta *HTTP*, ottenere una stringa contenente il valore associato a tale parametro, oppure il valore *null* se il parametro non esiste.

HttpServletResponse è la classe Java che rappresenta la risposta dell'oggetto *servlet* alla richiesta del *client*. Tramite opportuni metodi è possibile configurare la risposta:

- * *void setContentType(String type)*: permette di impostare la tipologia di contenuto della risposta. Nel progetto la risposta è una stringa in formato *JSON*, quindi è stato impostato il content type *application/json*;
- * *PrintWriter getWriter()*: restituisce un oggetto *PrintWriter* che può essere utilizzato per inviare caratteri di testo al *client*. Nel progetto, l'oggetto *PrintWriter* è stato utilizzato per inviare la stringa di risposta in formato *JSON*.

Viene di seguito fornita, a titolo d'esempio, l'implementazione del metodo *doPost()* del *servlet* del servizio *web* che si occupa di controllare che le credenziali inserite dall'utente in fase di login siano corrette. Nella prossima sezione viene fornito un esempio di come la logica applicativa di *moviORDER* effettua una richiesta a tale *servlet* e di come utilizza la risposta per modificare lo stato dell'applicazione. Nell'esempio, la classe *DatabaseConnection* fornisce un'interfaccia per l'interrogazione di un *database SQL Server*. Una spiegazione di tale classe è presente in sezione 6.1.2.2.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    //prelievo parametri richiesta
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    //connessione al database
    DatabaseConnection dbConnect=new DatabaseConnection(GetParam.getUrlCommonDB());
    try {
        dbConnect.connectToDB();
    }catch(SQLException e) {
        e.printStackTrace();
    }

    //esecuzione query
    ResultSet rs=dbConnect.doQuery("select * from Users");

    //generazione stringa JSON di risposta
    String json ="";
    json = generateResponse(rs,username,password);

    //invio risposta
    response.setContentType("application/json");
    response.getWriter().write(json);

    //chiusura connessione con database
    dbConnect.closeConnection();
    try {
        rs.close();
    }catch(SQLException e) {
        e.printStackTrace();
    }
}
```

Figura 6.1: Metodo *doPost()* del *servlet* che gestisce l'autenticazione

6.1.1.2 Interrogazione del servizio web

Nel momento in cui viene implementato un *servlet* concreto, *eclipse* gli associa un *end-point* in automatico. Tramite l'*end-point* è possibile raggiungere il *servlet* sul *server* per effettuare richieste *HTTP*. L'*end-point* associato da *eclipse* è della forma */NomeClasseServlet* e può essere cambiato dalle impostazioni dell'*IDE*. Per rendere possibile il raggiungimento del servizio *web* da parte della logica applicativa di *moviORDER*, è necessario che venga effettuato il *deploy* del servizio su *Apache Tomcat* e che quest'ultimo venga fatto girare sul *server cloud* di *VisioneImpresa*. Una volta che il servizio *web* è raggiungibile tramite la rete, è possibile iniziare ad effettuare richieste *HTTP*. In *moviORDER*, la struttura dell'*URL* di una richiesta *HTTP* è la seguente: *http://indirizzo:porta/moviORDER/NomeServlet*, dove:

- * *indirizzo*: è l'indirizzo del *server* dove il servizio *web* viene fatto girare;
- * *porta*: è la porta del *server* dove il servizio *web* viene fatto girare;
- * *NomeServlet*: è l'*end-point* (*servlet*) a cui si vuole inviare la richiesta *HTTP*.

MoviORDER effettua richieste *HTTP* tramite l'utilizzo di *AJAX* (*Asynchronous JavaScript And XML*). È importante far notare che *AJAX* non è un linguaggio di programmazione, bensì una tecnica per accedere ad un *server web* da una pagina *web*. Tramite *AJAX* è possibile leggere dati da un *server web* dopo che una pagina è stata caricata, aggiornare una pagina *web* senza il bisogno di dover ricaricare la stessa e inviare dati ad un *server web* in maniera del tutto trasparente all'utente.

Nel progetto, *AJAX* è stato implementato mediante *JavaScript* con l'utilizzo dell'oggetto *XMLHttpRequest*. Tale oggetto è supportato da tutti i *browser* moderni e può essere utilizzato per scambiare dati con un *server web* in maniera trasparente, ovvero senza il bisogno di dover ricaricare la pagina *web* per cambiarne lo stato. La sintassi per la creazione di un oggetto *XMLHttpRequest* è la seguente: *var xhttp = new XMLHttpRequest();*

Per l'invio di richieste *HTTP* al servizio *web* si utilizzano i metodi *open()* e *send()* di *XMLHttpRequest*. In particolare, il metodo *open()* permette di specificare la tipologia di richiesta tramite il passaggio di tre parametri:

- * **metodo**: specifica il metodo utilizzato per inviare la richiesta *HTTP*: *GET* o *POST*;
- * **url**: specifica l'indirizzo del *server* a cui inviare la richiesta *HTTP*. Nel caso del progetto l'indirizzo comprende l'*end-point* presso il quale la richiesta deve essere gestita;
- * **asincrona/sincrona**: specifica se la richiesta è asincrona (*true*) o sincrona (*false*).

Il metodo *send(string)* permette invece l'invio di una richiesta *HTTP* al servizio *web*. Esso richiede il passaggio di una stringa contenente i parametri da inviare al *server*. Poiché alcuni parametri possono contenere caratteri accentati, è stato necessario utilizzare il metodo *setRequestHeader()* per specificare la codifica dei caratteri, aggiungendo *header HTTP* alla richiesta. Facendo questo, è stato possibile evitare errori di lettura/scrittura di stringhe con caratteri accentati sul *database* di *moviORDER*.

È importante far notare che tutte le richieste *HTTP* inviate da *moviORDER* al servizio *web* sono asincrone, questo perché:

- * il codice sincrono non è raccomandato poiché *JavaScript* stopparebbe l'esecuzione fino all'arrivo di una risposta da parte del *server*. Se il *server* è occupato o lento, l'applicazione potrebbe aspettare per un tempo prolungato;
- * le richieste *AJAX* di tipo sincrono saranno rimosse dallo standard *web* nei prossimi anni. Scegliendo di utilizzare solamente richieste asincrone si permette a *moviORDER* di essere robusta a questo cambiamento futuro.

Per la gestione della risposta ricevuta dal *server* si sono utilizzate le seguenti proprietà dell'oggetto *XMLHttpRequest*:

- * *readyState*: contiene lo stato dell'oggetto. In particolare, per lo scopo del progetto, è interessante sapere che il valore 4 corrisponde ad una richiesta la cui risposta è pronta;
- * *status*: contiene il messaggio sullo stato della richiesta. In particolare, per lo scopo del progetto, è interessante sapere che il valore 200 corrisponde al messaggio *OK*, che nello standard *web* rappresenta una richiesta *HTTP* andata a buon fine;
- * *onreadystatechange*: definisce una funzione che deve essere eseguita quando la proprietà *readyState* cambia valore;
- * *responseText*: incapsula la stringa di risposta ricevuta dal servizio *web*.

Poiché la risposta ricevuta dal servizio *web* è una stringa in formato *JSON*, per effettuare il parsing di tale stringa è stato necessario convertirla in un oggetto *JavaScript*, tramite l'utilizzo del metodo *JSON.parse()*.

Viene di seguito fornito, a titolo d'esempio, il codice *JavaScript* della logica applicativa di *moviORDER* che effettua una chiamata *HTTP* all'*end-point* (*servlet*) che si occupa di gestire l'autenticazione. Il codice si occupa anche di gestire la risposta ricevuta dal servizio *web*. La funzione *tryLogin()* viene eseguita nel momento in cui l'utente preme sul pulsante di login presente nella schermata di login dell'applicazione.

```

function tryLogin(){
    //prelievo username e password da text-boxes
    var usern = document.getElementById("username").value;
    var password = document.getElementById("password").value;

    //creazione oggetto XMLHttpRequest
    var xhttp = new XMLHttpRequest();
    if(usern!="" && password!=""){ //caso credenziali inserite
        document.getElementById("loading").style.display = "block";

        //invio richiesta al servlet di controllo delle credenziali inserite
        xhttp.open("POST",host+"/moviORDER/AuthenticationServlet",true);
        xhttp.onreadystatechange = function() { //funzione anonima da eseguire quando readyState cambia stato
            if (this.readyState == 4 && this.status == 200) { //richiesta terminata e risposta pronta

                //creazione oggetto JS da stringa JSON di risposta
                var risp=JSON.parse(this.responseText);
                if(risp.messaggio=="OK"){ //caso credenziali corrette
                    document.getElementById("loading").style.display = "none";
                    location.replace("visualizzazioneArticoli.html?codAz="+risp.codAz+
                        "&username="+risp.username);
                }else{ //caso credenziali non corrette
                    document.getElementById("loading").style.display = "none";
                    showSomething(this); //visualizzazione errore riscontrato
                }
            }
        };
    }

    //configurazione header HTTP e invio richiesta
    xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded; charset=UTF-8");
    xhttp.send("username="+usern+"&password="+password);
} else{ //caso credenziali non inserite
    var errore=document.getElementById("errorMessage");
    errore.innerHTML="Inserire username e password";
    errore.style.backgroundColor="#2196F3";
    errore.style.display="block";
}
}
}

```

Figura 6.2: Esempio di invio di una richiesta *HTTP* tramite *AJAX*

6.1.1.3 API del servizio web

In questa sezione viene presentata la *API* del servizio *web* di *moviORDER*. In particolare, per ogni *end-point* (*servlet*) vengono specificati:

- * **indirizzo:** indirizzo tramite il quale è possibile raggiungere l'*end-point*;
- * **input:** costituito dai parametri inviati al servizio *web* tramite una richiesta *HTTP*;
- * **output:** costituito da una stringa in formato *JSON* che presenta struttura diversa a seconda dell'*end-point* che gestisce la richiesta;
- * **descrizione.**

6.1.1.3.1 Servizio di autenticazione

- * **Indirizzo:** /AuthenticationServlet;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *username*: è la *username* inserita dall'utente in fase di login;
 - *password*: è la *password* inserita dall'utente in fase di login.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:

- codice azienda e *username* dell’utente nel caso in cui *username* e *password* passati come parametro corrispondono ad un utente presente in *database*;
 - un messaggio d’errore nel caso in cui le credenziali sono scorrette o l’utente è stato bloccato.
- * **Descrizione:** questo *servlet* rappresenta il servizio di autenticazione di *mo-viORDER*. Ricevuti i parametri, il *servlet* cerca la *username* dell’utente nel *database* e, nel caso in cui questa fosse presente, procede nel cercare la *password* corrispondente. Nel caso in cui *username* e *password* passati siano quelli corretti, il *servlet* restituisce una stringa *JSON* contenente il codice azienda e la *username* dell’utente che ha cercando di loggarsi. Nel caso in cui le credenziali non siano corrette, oppure l’utente è stato bloccato, viene restituito un messaggio d’errore.

6.1.1.3.2 Servizio di verifica di connessione con il *database*

- * **Indirizzo:** /CheckConnectionURL;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *codice azienda*: è il codice azienda dell’azienda cliente di VisioneImpresa della quale si vuole verificare la presenza del *database*.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - un messaggio positivo nel caso in cui il *database* dell’azienda corrispondente al codice inserito è raggiungibile;
 - un messaggio negativo nel caso in cui il *database* dell’azienda corrispondente al codice inserito non è raggiungibile.
- * **Descrizione:** questo *servlet* si occupa di controllare se un *database* aziendale è raggiungibile provando ad effettuare una *query* su tale *database*. Risponde positivamente nel caso in cui il *database* dell’azienda corrispondente al codice inserito è raggiungibile (*query* con un ritorno), mentre risponde negativamente nel caso in cui il *database* non è raggiungibile (*query* che ha sollevato un eccezione).

6.1.1.3.3 Servizio di rimozione articoli dal carrello

- * **Indirizzo:** /DeleteSelectedItems;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *lista di codici articolo*: è una lista dei codici articolo degli articoli che devono essere eliminati;
 - *username*: è la *username* dell’utente che ha richiesto l’eliminazione degli articoli;
 - *path*: è la stringa di connessione al *database* aziendale dell’utente autenticato.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - un messaggio positivo nel caso in cui la cancellazione è andata a buon fine;
 - un messaggio negativo nel caso in cui la cancellazione non è andata a buon fine.

- * **Descrizione:** questo *servlet* si occupa di eliminare dal *database* aziendale dell’utente autenticato gli articoli che quest’ultimo ha richiesto di rimuovere dal carrello. Risponde positivamente nel caso in cui gli articoli sono stati rimossi correttamente dal *database*, mentre risponde negativamente nel caso in cui gli articoli non sono stati rimossi dal *database*.

6.1.1.3.4 Servizio di ricerca di un codice a barre

- * **Indirizzo:** /FindArticleBarCode;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *codice a barre*: è il codice a barre di un articolo del quale si vuole conoscere il codice articolo;
 - *path*: è la stringa di connessione al *database* aziendale dell’utente autenticato.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - codice articolo corrispondente al *barcode* passato come parametro nel caso in cui il *barcode* corrisponde a quello di un articolo venduto dall’azienda;
 - un messaggio negativo nel caso in cui il *barcode* passato come parametro non corrisponde ad un articolo venduto dall’azienda.
- * **Descrizione:** questo *servlet* si occupa di fornire il codice articolo dell’articolo corrispondente al codice a barre ricevuto come parametro, effettuando la ricerca del *barcode* all’interno del *database*. Nel caso in cui il codice a barre è presente all’interno del *database* viene restituito il codice articolo corrispondente, mentre nel caso in cui non è presente viene restituito un messaggio negativo.

6.1.1.3.5 Servizio di ricerca di un codice articolo

- * **Indirizzo:** /FindArticleCode;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *codice articolo*: è il codice articolo o il *barcode* di un articolo del quale si vuole verificare la presenza in *database*;
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - codice articolo nel caso in cui l’articolo è presente in *database*;
 - un messaggio negativo nel caso in cui l’articolo non è presente in *database*.
- * **Descrizione:** questo *servlet* si occupa di fornire il codice articolo dell’articolo corrispondente al codice articolo o al *barcode* ricevuto come parametro, effettuando la ricerca del codice articolo o del *barcode* all’interno del *database*. Nel caso in cui il codice articolo o il *barcode* passato come parametro corrisponde ad un articolo presente in *database* viene restituito il codice articolo, mentre nel caso in cui non corrisponde ad un articolo presente in *database* viene restituito un messaggio negativo.

6.1.1.3.6 Servizio di prelievo informazioni di un articolo

- * **Indirizzo:** /GetArticleDescNote;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *codice azienda*: è il codice azienda della quale l’utente autenticato è cliente;
 - *codice articolo*: è il codice articolo dell’articolo del quale si vogliono ottenere informazioni.
- * **Output:** questo *servlet* può produrre un solo output poiché l’applicazione è stata implementata in modo che non ci siano casi negativi:
 - descrizione, note, quantità minima ordinabile e step di incremento quantità per l’articolo corrispondente al codice articolo passato come parametro.
- * **Descrizione:** questo *servlet* si occupa di fornire informazioni sull’articolo corrispondente al codice articolo passato come parametro.

6.1.1.3.7 Servizio di prelievo degli articoli in carrello

- * **Indirizzo:** /GetArticlesByUsername;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *codice azienda*: è il codice azienda della quale l’utente autenticato è cliente;
 - *username*: è il nome utente dell’utente autenticato.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - array degli articoli nel carrello dell’utente autenticato, dove per ogni articolo vengono restituiti la quantità ordinata, il codice articolo e la descrizione dell’articolo;
 - array vuoto nel caso in cui l’utente autenticato non presenta articoli in carrello.
- * **Descrizione:** questo *servlet* si occupa di fornire la lista degli articoli in carrello per l’utente la quale *username* è stata passata come parametro. Nel caso in cui l’utente presenta articoli in carrello viene restituita la lista degli articoli in carrello, mentre nel caso in cui non presenta articoli in carrello viene restituita una lista vuota.

6.1.1.3.8 Servizio di prelievo di informazioni di un utente

- * **Indirizzo:** /GetNameByUsername;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *path*: è la stringa di connessione al *database* aziendale dell’utente autenticato;
 - *username*: è il nome utente dell’utente autenticato.
- * **Output:** questo *servlet* può produrre un solo output poiché l’applicazione è stata implementata in modo che non ci siano casi negativi:

- nome e ragione sociale dell’utente autenticato e codice documento e descrizione del documento da generare nel caso in cui l’utente invii un ordine.
- * **Descrizione:** questo *servlet* si occupa di fornire informazioni sull’utente la cui *username* è stata passata come parametro.

6.1.1.3.9 Servizio di prelievo informazioni di un articolo in carrello

- * **Indirizzo:** /GetTmpArticleData;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *path*: è la stringa di connessione al *database* aziendale dell’utente autenticato;
 - *codice articolo*: è il codice articolo dell’articolo del quale si vogliono ottenere informazioni;
 - *username*: è il nome utente dell’utente autenticato.
- * **Output:** questo *servlet* può produrre un solo output poiché l’applicazione è stata implementata in modo che non ci siano casi negativi:
 - quantità e note dell’articolo nel carrello dell’utente autenticato corrispondente al codice articolo passato come parametro.
- * **Descrizione:** questo *servlet* si occupa di fornire informazioni riguardanti uno specifico articolo nel carrello dell’utente la quale *username* è stata passata come parametro.

6.1.1.3.10 Servizio di inserimento/modifica articolo

- * **Indirizzo:** /InsertUpdateArticle;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - *path*: è la stringa di connessione al *database* aziendale dell’utente autenticato;
 - *query*: è una stringa contenente la *query* di inserimento/modifica di un articolo.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - un messaggio positivo nel caso in cui la *query* di inserimento/modifica articolo è andata a buon fine;
 - un messaggio negativo nel caso in cui la *query* di inserimento/modifica articolo non è andata a buon fine.
- * **Descrizione:** questo *servlet* si occupa di inserire o modificare un articolo nel carrello dell’utente autenticato. L’inserimento e la modifica avvengono mediante l’utilizzo dello stesso *servlet* poiché la *query* da eseguire sul *database*, che può essere di tipo *INSERT* o *UPDATE*, viene passata come parametro. Nel caso in cui la *query* viene eseguita in modo corretto sul *database* viene restituito un messaggio positivo, mentre nel caso in cui non viene eseguita viene restituito un messaggio negativo.

6.1.1.3.11 Servizio di invio di un ordine

- * **Indirizzo:** /SendOrder;
- * **Input:** il *servlet* richiede i seguenti parametri:
 - path: è la stringa di connessione al *database* aziendale dell’utente autenticato;
 - codici articolo: è la lista dei codici degli articoli che l’utente autenticato ha ordinato;
 - *username*: è il nome utente dell’utente autenticato;
 - ragione sociale: è la ragione sociale dell’utente autenticato;
 - nome: è il nome dell’utente autenticato;
 - codice documento: è il codice del documento che deve essere generato con l’ordine;
 - data: è la data d’invio dell’ordine;
 - note: sono le note inserite dall’utente in fase di invio dell’ordine;
 - codice azienda: è il codice azienda della quale l’utente autenticato è cliente.
- * **Output:** le possibili risposte del *servlet* sono le seguenti:
 - un messaggio positivo nel caso in cui l’ordine è stato inviato con successo;
 - un messaggio negativo nel caso in cui l’ordine non è stato inviato con successo.
- * **Descrizione:** questo *servlet* si occupa di registrare sul *database* un ordine contenente gli articoli passati come parametro. Il resto dei parametri passati viene utilizzato per inviare una mail di conferma all’utente autenticato e all’azienda presso cui l’utente è cliente. Nel caso in cui la registrazione dell’ordine sul *database* avvenga correttamente viene restituito un messaggio positivo, mentre nel caso in cui non avvenga correttamente viene restituito un messaggio negativo.

6.1.2 JDBC

In questa sezione viene presentato l’utilizzo della *Java Database Connectivity API* nella realizzazione del servizio *web*.

6.1.2.1 Driver JDBC

Un driver *JDBC* è una componente *software* che permette ad un’applicazione *Java* di interagire con un *database*. Per supportare la connessione a singoli *database*, *JDBC* (*Java Database Connectivity API*) richiede i driver per ogni *database*. Il driver permette la connessione con il *database* e implementa il protocollo di trasferimento di *query* e risultati tra il *client* e il *database*. Poiché per l’implementazione del *database* è stato utilizzato *SQL Server* si sono dovuti utilizzare i driver *JDBC* per tale *database*.

6.1.2.2 Classe DatabaseConnection

Per lo sviluppo del servizio *web* è stata realizzata la classe *DatabaseConnection* appartenente al package *dbConnection*. Tale classe fornisce un’interfaccia utilizzabile per la gestione dell’interazione con un *database* di tipo *SQL Server*. In questa sezione vengono presentati i metodi che costituiscono tale interfaccia.

6.1.2.2.1 Costruttori

La classe presenta due metodi costruttori:

- * *public DatabaseConnection(String u, String user, String psw, String db)*: costruisce un oggetto *DatabaseConnection* a partire dall'*URL* del *server* in cui è presente il *database*, la *username* e la *password* di accesso al *database*, e il nome del *database* al quale si desidera connettersi;
- * *public DatabaseConnection(String dbConnectionString)*: costruisce un oggetto *DatabaseConnection* a partire dalla stringa di connessione ad uno specifico *database*.

Il formato della stringa di connessione ad un *database* è il seguente: *indirizzoServer;databaseName=nomeDb;user=u;password=psw*, dove:

- * *indirizzoServer*: è l'indirizzo IP pubblico del *server* contenente il *database* al quale si desidera connettersi. Nel caso del progetto potrebbe essere l'indirizzo del *server* di un'azienda esterna che desidera utilizzare il proprio *database* per l'interazione;
- * *nomeDb*: è il nome del *database* al quale si desidera connettersi;
- * *u*: è il nome utente per l'accesso al *database*;
- * *psw*: è la *password* per l'accesso al *database*.

Viene di seguito presentato, a titolo d'esempio, il codice Java che implementa il metodo *public DatabaseConnection(String dbConnectionString)*. Il metodo si occupa di splittare la stringa di connessione passata come parametro per ottenere i dati utili alla costruzione dell'oggetto *DatabaseConnection*.

```
public DatabaseConnection(String dbConnectionString) {
    //split su ";"
    String[] splitted=dbConnectionString.split(";");
    //split su "=" coppia database
    String[] db=splitted[1].split("=");
    //split su "=" coppia username
    String[] user2=splitted[2].split("=");
    //split su "=" coppia password
    String[] psw2=splitted[3].split("=");
    //prelievo url
    url=splitted[0];
    //prelievo nome db
    dbName=db[1];
    //prelievo username
    username=user2[1];
    //prelievo password
    password=psw2[1];
}
```

Figura 6.3: Metodo costruttore della classe *DatabaseConnection*

6.1.2.2.2 Metodo `connectToDb()`

Tale metodo si occupa di instaurare la connessione con il *database* desiderato. Per far questo, configura i driver *JDBC* e costruisce l'*URL* di connessione al *database* tramite i dati contenuti nell'oggetto d'invocazione. Il metodo potrebbe sollevare un'eccezione di tipo *ClassNotFoundException* nel caso in cui la classe utilizzata per i driver *JDBC* sia inesistente o non sia stata importata all'interno del progetto. Viene di seguito fornito, a titolo d'esempio, il codice del metodo `connectToDb()`.

```
public void connectToDB() throws SQLException{
    try {
        //configurazione driver JDBC
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

        //costruzione url di connessione al database
        String connectionUrl="jdbc:sqlserver://"+url+";databaseName="+
            dbName+";user=username;password=password;

        //connessione al database
        connection=DriverManager.getConnection(connectionUrl);
    }catch(ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

Figura 6.4: Metodo `connectToDb()` della classe *DatabaseConnection*

6.1.2.2.3 Altri metodi

Sono stati implementati altri metodi di complessità inferiore. Per questo motivo ne viene presentata solamente una breve descrizione:

- * *doQuery(query)*: questo metodo permette di eseguire una *query* di tipo *SELECT* sul *database*. Restituisce il *ResultSet* contenente il risultato della *query*;
- * *doUpdateQuery(query)*: questo metodo permette di eseguire una *query* di tipo *INSERT*, *UPDATE* o *DELETE* sul *database*. Restituisce il numero di righe inserite, modificate o cancellate;
- * *closeConnection()*: questo metodo permette di eseguire il processo di disconnessione dal *database*.

6.1.3 Classi utilità

Per lo sviluppo del servizio *web* è stato necessario implementare due classi utilità. In questa sezione ne viene presentata brevemente l'implementazione. Tali classi appartengono al *package utility*.

6.1.3.1 Classe `GetParam`

La classe *GetParam* si occupa di restituisc un campo statico contenente la stringa di connessione al *database* *CommonDb* dell'applicazione *moviORDER*. Come detto precedentemente, questo *database* contiene tutti i dati di autenticazione degli utenti

di *moviORDER*. Poiché tale stringa di connessione viene utilizzata frequentemente all'interno del servizio *web*, si è deciso di inserirla in un unico punto del servizio, in modo da evitare la modifica di più file nel caso in cui essa cambi.

6.1.3.2 Classe MailUtility

La classe *MailUtility* fornisce un'interfaccia per l'invio di e-mail. È stato necessario implementare tale classe poiché è stato richiesto di inviare una mail di conferma all'utente autenticato e all'azienda presso cui l'utente è cliente nel momento in cui un ordine viene registrato. La classe presenta un costruttore che richiede i parametri per la configurazione di un *server SMTP*:

- * **host**: è l'indirizzo IP dell'*host* su cui è installato il *server SMTP*;
- * **post**: è il numero di porta dell'*host* su cui è installato il *server SMTP*;
- * **username**: è il nome utente di accesso al *server SMTP*;
- * **password**: è la *password* di accesso al *server SMTP*.

Il metodo *sendMail()* si occupa di configurare e inviare una mail all'utente che ha effettuato l'ordine e all'azienda presso cui l'utente è cliente. Per far questo, il metodo richiede:

- * indirizzo e-mail dell'utente;
- * indirizzo e-mail dell'azienda;
- * indirizzo e-mail del mittente;
- * oggetto della e-mail;
- * testo della e-mail: è una tabella scritta in codice *HTML* contenente i dati degli articoli ordinati.

Viene di seguito fornito, a titolo d'esempio, il codice *Java* che implementa il metodo *sendMail()*.

```

public void sendMail (String dest, String az, String mitt, String oggetto, String testoEmail)
throws MessagingException {

    // Creazione di una mail session
    Properties props = new Properties();

    //settaggio del server smtp
    props.put("mail.smtp.host", host);
    props.put("mail.smtp.port", port);
    props.put("mail.smtp.auth", "true");
    Session session = Session.getDefaultInstance(props,new Authenticator() {

        @Override
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username,password);
        }
    });

    // Creazione del messaggio da inviare
    MimeMessage message = new MimeMessage(session);
    message.setSubject(oggetto,"UTF-8");
    Multipart mp = new MimeMultipart();
    MimeBodyPart mbp = new MimeBodyPart();
    mbp.setContent(testoEmail, "text/html; charset=utf-8");
    mp.addBodyPart(mbp);
    message.setContent(mp);
    message.setSentDate(new java.util.Date());

    // Aggiunta degli indirizzi del mittente e dei destinatari
    InternetAddress fromAddress = new InternetAddress(mitt);
    InternetAddress toAddress = new InternetAddress(dest);
    InternetAddress toAz = new InternetAddress(az);
    message.setFrom(fromAddress);
    message.setRecipient(Message.RecipientType.TO, toAddress);
    message.setRecipient(Message.RecipientType.CC, toAz);

    // Invio del messaggio
    Transport.send(message);
}

```

Figura 6.5: Metodo *sendMail()* della classe *MailUtility*

6.2 Logica applicativa

In questa sezione vengono presentati gli aspetti più interessanti riguardanti la codifica della logica applicativa di *moviORDER*. La sezione si incentra sui meccanismi di integrazione dei *plugin PhoneGap* con il codice *Javascript* della logica applicativa.

6.2.1 Plugin di PhoneGap

Un *plugin* è un pacchetto di codice che permette al visualizzatore *web* di *Cordova*, il quale renderizza l'applicazione, di comunicare con la piattaforma nativa sulla quale viene eseguito. I *plugin* forniscono accesso alle funzionalità del dispositivo che normalmente non sono disponibili per le applicazioni *web-based*. Tutte le *API* di *Cordova* sono implementate tramite *plugin*. Essi sono composti da un'unica interfaccia *JavaScript* che astrae le differenti implementazioni in codice nativo delle funzionalità fornite dal *plugin*. In fase di *build* dell'applicazione il codice *JavaScript* dei *plugin* viene quindi convertito nel codice nativo della piattaforma sulla quale si sta distribuendo l'applicazione.

6.2.2 Installazione dei plugin

Tramite *PhoneGap CLI*, interfaccia a linea di comando precedentemente descritta, è possibile aggiungere *plugin* alla configurazione del progetto *PhoneGap*. È sufficiente lanciare la CLI dalla cartella principale del progetto ed eseguire il comando *phonegap plugin add nomePlugin*, dove *nomePlugin* è il nome del *plugin* che si desidera scaricare ed installare (es. *cordova-plugin-whitelist*).

6.2.3 Premesse all'utilizzo dei plugin

Per poter utilizzare efficacemente i *plugin* è necessario predisporre il codice *JavaScript* al loro utilizzo. In particolare sono necessari due accorgimenti che vengono di seguito presentati.

6.2.3.1 Inclusione di *cordova.js*

Ogni file della logica applicativa di *moviORDER* che utilizza dei *plugin* deve includere il file *cordova.js*. Questo file *JavaScript* permette il funzionamento dei *plugin* utilizzati all'interno della logica. Se si utilizzasse un *plugin* senza aver importato tale file, la console darebbe il seguente errore: “*Cordova* is not available, Make sure to include *cordova.js*”.

6.2.3.2 Evento *deviceready*

L'evento *deviceready* è essenziale in ogni applicazione *PhoneGap* poiché segnala il corretto caricamento delle *API* di *Cordova* sul dispositivo. Se l'evento non venisse atteso prima di utilizzare un *plugin*, si rischierebbe che l'applicazione effettui una chiamata ad una funzione *JavaScript* di *Cordova* prima che il corrispondente codice nativo per il dispositivo diventi disponibile. L'evento *deviceready* viene lanciato nel momento in cui *Cordova* è stato completamente caricato. Quindi dopo aver atteso l'evento, è possibile effettuare chiamate alle *API* di *Cordova* in completa sicurezza. Per attendere l'evento è sufficiente inserire un listener nel codice *JavaScript*. Viene di seguito fornito, a scopo illustrativo, un esempio di codice *JavaScript* che implementa l'attesa di tale evento.

```
//attesa di caricamento API Cordova
document.addEventListener("deviceready",checkConnection,false);

function checkConnection(){ //controllo presenza connessione
    if(navigator.connection.type == Connection.NONE) { //utilizzo plugin network information
        navigator.notification.alert("Non sei connesso ad Internet. L'applicazione è inutilizzabile.",
            function(){
                navigator.app.exitApp(); //chiusura app
            }, "Errore");
    }else{
        document.addEventListener("offline", off, false);
        disableBack();
    }
}
```

Figura 6.6: Esempio di codice *JavaScript* che attende l'evento *deviceready*

6.2.4 Plugin utilizzati

In questa sezione vengono presentati i *plugin PhoneGap* utilizzati nella realizzazione della logica applicativa di *moviORDER*. Per ogni *plugin* vengono messi in evidenza vantaggi e svantaggi (se presenti) nell'utilizzo e un esempio di utilizzo.

6.2.4.1 Dialogs plugin

Il *plugin dialogs* fornisce accesso all'interfaccia grafica nativa degli elementi *dialog* tramite l'utilizzo dell'oggetto *navigator.notification*. Questo *plugin* è stato utilizzato per convertire gli *alert box* delle usuali pagine *web* in *dialog* nativi per l'ambiente *mobile*. Sono stati utilizzati i seguenti metodi:

* *alert()*: visualizza un *dialog* con un messaggio di allerta preimpostato. Il metodo richiede i seguenti parametri:

- *message*: è il messaggio visualizzato nel *dialog*;
- *callback*: è una funzione anonima di *callback* da eseguire quando viene premuto il pulsante nel *dialog*;
- *title*: è il titolo del *dialog*;
- *buttonName*: è l'etichetta del pulsante nel *dialog*.

* *confirm()*: visualizza un *dialog* per chiedere conferma di un'azione. Il metodo richiede i seguenti parametri:

- *message*: è il messaggio visualizzato nel *dialog* di conferma;
- *callback*: è una funzione anonima di *callback* da eseguire nel caso in cui il bottone di conferma viene premuto;
- *title*: è il titolo del *dialog* di conferma;
- *buttonLabels*: sono le etichette dei vari bottoni presenti nel *dialog* di conferma. Solitamente sono “OK” e “Annulla”.

Senza l'utilizzo di questo *plugin* non sarebbe stato possibile modificare il titolo e i bottoni del *dialog* poiché per motivi di sicurezza *JavaScript* non permette tale modifica. Inoltre la visualizzazione del *dialog* non sarebbe stata quella desiderata, ovvero la visualizzazione nativa.

Vengono di seguito forniti, a scopo illustrativo, un esempio di codice *JavaScript* che non utilizza il *plugin* e un esempio che lo utilizza. Per ognuno degli esempi viene mostrato uno *screenshot* dell'output risultante che, nel primo caso sarà un comune *alert* e nel secondo un *dialog* nativo.

```
alert("Scansione errata o codice a barre non corrispondente ad un articolo del fornitore.");
```

Figura 6.7: Esempio di codice *JavaScript* che non utilizza il *plugin dialogs*

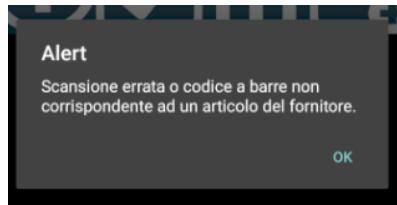


Figura 6.8: Esempio di visualizzazione scorretta - *alert*

```
navigator.notification.alert("Scansione errata o codice a barre non corrispondente ad un articolo del fornitore.",
    deleteModal, //funzione di callback da eseguire
    "Info" //titolo del dialog
);
```

Figura 6.9: Esempio di codice JavaScript che utilizza il *plugin dialogs*

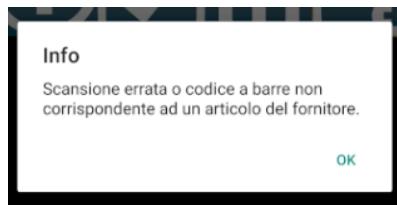


Figura 6.10: Esempio di visualizzazione corretta - *dialog*

6.2.4.2 Network information plugin

Il *plugin network information* fornisce una moderna implementazione della *Network Information API*. Essa fornisce informazioni riguardanti la rete cellulare e Wi-Fi del dispositivo e permette di capire se il dispositivo presenta una connessione ad *Internet*. Più precisamente, il *plugin* fornisce un’interfaccia *JavaScript* che astrae il codice nativo utilizzato per monitorare la rete del dispositivo. *Navigator.connection* è l’oggetto che permette di acquisire le informazioni appena descritte. La proprietà *type* è stata utilizzata per comprendere in modo veloce lo stato della connessione del dispositivo e la tipologia di connessione attiva. Essa può assumere i seguenti valori:

- * *Connection.UNKNOWN*: tipologia di rete sconosciuta;
- * *Connection.ETHERNET*: dispositivo connesso alla rete via cavo *ethernet*;
- * *Connection.WIFI*: dispositivo connesso ad una rete *Wi-Fi*;
- * *Connection.CELL_2G*: dispositivo connesso ad una rete cellulare di tipo *2G*;
- * *Connection.CELL_3G*: dispositivo connesso ad una rete cellulare di tipo *3G*;
- * *Connection.CELL_4G*: dispositivo connesso ad una rete cellulare di tipo *4G*;
- * *Connection.CELL*: dispositivo connesso ad una rete cellulare la cui tipologia non è identificabile;
- * *Connection.NONE*: dispositivo non connesso alla rete.

Un limite di questa proprietà è presente in ambiente *iOS*, infatti non è possibile identificare nessun tipo di rete cellulare alla quale il dispositivo è connesso. Per questo, in ambiente *iOS*, si è dovuta utilizzare la proprietà *onLine* dell'oggetto *navigator*.

All'oggetto *navigator.connection* sono associate due tipologie di evento:

- * *offline*: evento lanciato quando un dispositivo precedentemente collegato ad *Internet* perde la connessione e l'applicazione non è più in grado di accedere alla rete. In particolare, viene lanciato esattamente quando il valore della proprietà *type* diventa *Connection.NONE*;
- * *online*: evento lanciato quando un dispositivo precedentemente scollegato dalla rete riceve la connessione permettendo all'applicazione di accedere ad *Internet*. In particolare, viene lanciato esattamente quando il valore della proprietà *type* cambia da *NONE* ad un altro valore.

Il *plugin network information* è stato utilizzato per chiudere *moviORDER* nel caso in cui venisse aperta mentre il dispositivo è *offline*. L'evento *offline* ha permesso di visualizzare messaggi relativi allo stato della connessione. Più precisamente, nel caso in cui il dispositivo perdesse la connessione durante l'utilizzo dell'applicazione viene visualizzato un messaggio che notifica all'utente che l'applicazione è inutilizzabile.

Vengono di seguito forniti, a scopo illustrativo, degli esempi di codice *JavaScript* che utilizzano l'oggetto *network.connection* ed in particolare la proprietà *type* e l'evento *offline*.

```
function checkConnection(){ //funzione di verifica connettività
    if(navigator.connection.type == Connection.NONE) { //caso assenza connettività
        navigator.notification.alert("Non sei connesso ad Internet. L'applicazione è inutilizzabile.",
            function(){
                navigator.app.exitApp(); //chiusura applicazione
            }, "Errore");
    }else{
        document.addEventListener("offline", off, false);
        disableBack();
    }
}
```

Figura 6.11: Esempio di utilizzo della proprietà *type*

```
document.addEventListener("offline", off, false); //settaggio listener su evento offline
function off(){ //funzione da eseguire al lancio dell'evento offline
    navigator.notification.alert("Non sei più connesso ad Internet! L'applicazione è inutilizzabile. Attendere la connessione o chiudere l'app.",
        null,
        "Errore"); //visualizzazione di un messaggio d'errore
}
```

Figura 6.12: Esempio di utilizzo dell'evento *offline*

6.2.4.3 Barcode scanner plugin

Il *plugin barcode scanner* fornisce un'interfaccia *JavaScript* che astrae il codice nativo che permette di effettuare la scansione di un codice a barre da un qualsiasi dispositivo dotato di fotocamera. Il *plugin* crea l'oggetto *cordova.plugin.barcodeScanner* che presenta il metodo *scan(success, fail, settings)*. *Success* è una funzione di *callback* che viene eseguita quando la scansione del codice a barre va a buon fine. *Fail* è una funzione di *callback* che viene eseguita quando la scansione del codice a barre non va a buon fine. *Settings* è una variabile contenente un insieme di impostazioni per l'utilizzo del *plugin*, quali:

- * *preferFrontCamera*: permette di preferire la fotocamera frontale a quella posteriore per la scansione del codice a barre;
- * *showFlipCameraButton*: permette di visualizzare il bottone per il cambio della fotocamera;
- * *showTorchButton*: permette di visualizzare il bottone per l'attivazione del *flash*;
- * *torchOn*: permette di attivare il *flash* di *default*;
- * *saveHistory*: permette il salvataggio della cronologia dei codici scansionati;
- * *prompt*: permette la visualizzazione di un messaggio per aiutare l'utente nell'esecuzione della scansione;
- * *resultDisplayDuration*: permette la visualizzazione di un testo per un determinato numero di secondi nel caso in cui un codice a barre viene captato;
- * *formats*: permette di impostare la tipologia di codici a barre che devono essere captati;
- * *orientation*: permette di impostare l'orientamento del dispositivo durante la scansione del codice a barre;
- * *disableAnimations*: permette la disattivazione di ogni tipo di animazione durante la scansione del codice a barre;
- * *disableSuccessBeep*: permette di disabilitare l'emissione del suono acustico nel caso in cui un codice a barre viene captato.

In ambiente iOS, per poter utilizzare il *plugin*, è necessario aggiungere una *NSCameraUsageDescription* al file *Info.plist*. *NSCameraUsageDescription* descrive la ragione per la quale l'applicazione accede alla fotocamera dell'utente. Quando il sistema operativo richiede all'utente di permettere l'accesso, la stringa inserita nella *NSCameraUsageDescription* viene visualizzata sul *dialog*. Se non viene fornita la descrizione di utilizzo, l'applicazione crescerà prima della visualizzazione del *dialog*. Inoltre, Apple rifiuta le applicazioni che accedono a dati privati senza fornire una descrizione di utilizzo.

Il *plugin* ha funzionato perfettamente in ambiente *Android* ma ha presentato alcuni problemi in ambiente *iOS*. Nei dispositivi con una fotocamera mediocre le scansioni richiedevano più tempo del previsto, alcune volte anche minuti. Per risolvere il problema si è dovuto modificare il codice nativo della versione *iOS* dell'applicazione. Il problema veniva riscontrato perché il codice differenziava il processo di scansione a seconda della qualità rilevata della fotocamera del dispositivo. Per cellulari con fotocamera mediocre veniva fatta una valutazione troppo ottimistica e per questo si è dovuto modificare il codice in modo da rendere il processo di scansione uguale per tutte le fotocamere. Dopo vari test si è potuto osservare che la soluzione migliore era l'utilizzo del processo di scansione per fotocamere di qualità media. Viene di seguito fornito, a scopo illustrativo, il codice *Objective-C++* che si occupa di settare il processo di scansione.

```

if ([captureSession canSetSessionPreset:AVCaptureSessionPresetHigh]) { //se la fotocamera è buona
    captureSession.sessionPreset = AVCaptureSessionPresetMedium; //setting del processo di media qualità
} else if ([captureSession canSetSessionPreset:AVCaptureSessionPresetMedium]) { //se la fotocamera è mediocre
    captureSession.sessionPreset = AVCaptureSessionPresetMedium; //setting del processo di media qualità
} else {
    return @"unable to preset high nor medium quality video capture";
}

```

Figura 6.13: Codice *Objective-C++* per il settaggio del processo di scansione

Viene di seguito fornito, a scopo illustrativo, il codice *JavaScript* della logica di *moviORDER* che utilizza il *plugin barcode scanner*.

```

function openCamera(){
    cordova.plugins.barcodeScanner.scan( //definizione parametri metodo scan
        function (result) { //funzione di callback da eseguire in caso di scansione andata a buon fine
            if(result.cancelled == true) { //caso di scansione cancellata dall'utente -> ritorno alla home page
                navigator.notification.alert("Scansione cancellata.");
                location.replace("visualizzazioneArticoli.html?codAzienda="+codAzienda+"&username="+username), "Info");
            }else{ //caso scansione andata a buon fine
                var xhttp = new XMLHttpRequest();
                xhttp.open("POST",host+"/moviORDER/FindArticleBarcode",true); //ricerca codice articolo
                xhttp.onreadystatechange = function() {
                    if (this.readyState == 4 && this.status == 200) { //risposta pronta
                        var resp = JSON.parse(this.responseText);
                        if(resp.messaggio == "no"){ //codice non trovato in database -> messaggio d'errore
                            alert("Scansione errata o codice a barre non corrispondente ad un articolo del fornitore.", deleteModal, "Info");
                        }else{ //codice trovato in database
                            compute(resp.codArt);
                        }
                    }
                };
                xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded; charset=UTF-8");
                xhttp.send("codArtAlias="+result.text+"&path="+connectionUrl);
            }
        },
        function (error) { //funzione di callback da eseguire in caso di scansione non andata a buon fine
            navigator.notification.alert("Scansione fallita: " + error,deleteModal,"Errore");
        },
        {
            //settings del processo di scansione
            preferFrontCamera : false, // iOS and Android
            showFlipCameraButton : false, // iOS and Android
            showTorchButton : true, // iOS and Android
            torchOn: false, // Android, launch with the torch switched on (if available)
            saveHistory: true, // Android, save scan history (default false)
            prompt : "Central il codice a barre o il QR dentro l'area di rilevamento.", // Android
            resultDisplayDuration: 500, // Android, display scanned text for X ms. 0 suppresses it entirely, default 1500
            formats : "EAN_13,QR_CODE,EAN_8,CODE_39,UPC_A,UPC_E", // default: all but PDF_417 and RSS_EXPANDED
            orientation : "portrait", // Android only (portrait/landscape), default unset so it rotates with the device
            disableAnimations : true, // iOS
            disableSuccessBeep: false // iOS and Android
        }
    );
}

```

Figura 6.14: Codice *JavaScript* che utilizza il *plugin barcode scanner*

6.3 Interfaccia grafica

In questa sezione vengono presentati gli aspetti più interessanti riguardanti la codifica dell’interfaccia grafica di *moviORDER*. In particolare, vengono presentate le possibili interazioni dell’utente con le interfacce. Nell’ultima sezione vengono fatte delle considerazioni sullo sviluppo dell’interfaccia.

6.3.1 Schermata di login

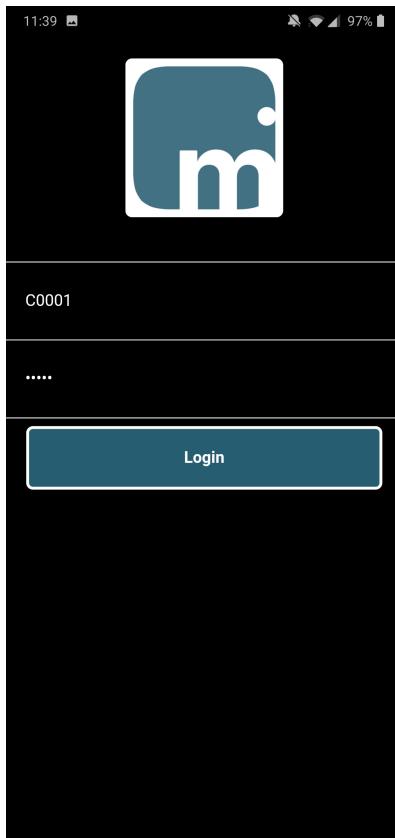


Figura 6.15: Schermata di login

Tramite la schermata di login, un qualsiasi utente in possesso di credenziali di accesso può accedere a *moviORDER*. Per tentare l’accesso, è necessario inserire *username* e *password* e premere sul pulsante di login. Se l’utente ha inserito credenziali corrette viene aperta la *home page* dell’applicazione, mentre se le credenziali non dovessero essere corrette viene visualizzato un messaggio d’errore. Il messaggio d’errore è esplicativo dell’errore riscontrato e i possibili errori sono:

- * le credenziali non sono state inserite;
- * la *username* inserita è inesistente;
- * la *password* inserita non è corretta;
- * le credenziali inserite sono corrette ma l’utente è stato bloccato dall’azienda.

6.3.2 Home page



Figura 6.16: Home page

La *home page* racchiude tutte le funzionalità di *moviORDER*. A partire dall’alto e proseguendo verso il basso, l’interfaccia presenta le seguenti parti:

- * messaggio di benvenuto per l’utente autenticato: questo messaggio visualizza la ragione sociale dell’utente;
- * pulsante *tutorial*: premendo su questo pulsante è possibile visualizzare il *tutorial* di *moviORDER*;
- * pulsante *logout*: premendo su questo pulsante è possibile effettuare il logout da *moviORDER*;
- * pulsante di aggiunta articolo: premendo su questo pulsante è possibile accedere al *modal* per l’aggiunta di un articolo in carrello;
- * pulsante di selezione/deselezione multipla di articoli: premendo su questo pulsante è possibile selezionare/deselezionare tutti gli articoli in carrello;
- * pulsante di eliminazione articoli: premendo su questo pulsante è possibile rimuovere gli articoli selezionati dal carrello;

- * pulsante di invio ordine: premendo su questo pulsante è possibile accedere al *modal* di invio ordine;
- * carrello: ogni articolo in carrello presenta:
 - una check-box per la selezione/deselezione dell'articolo;
 - un'indicazione sulla quantità di pezzi ordinati;
 - il codice dell'articolo;
 - una breve descrizione dell'articolo.

6.3.3 Modal di aggiunta articolo

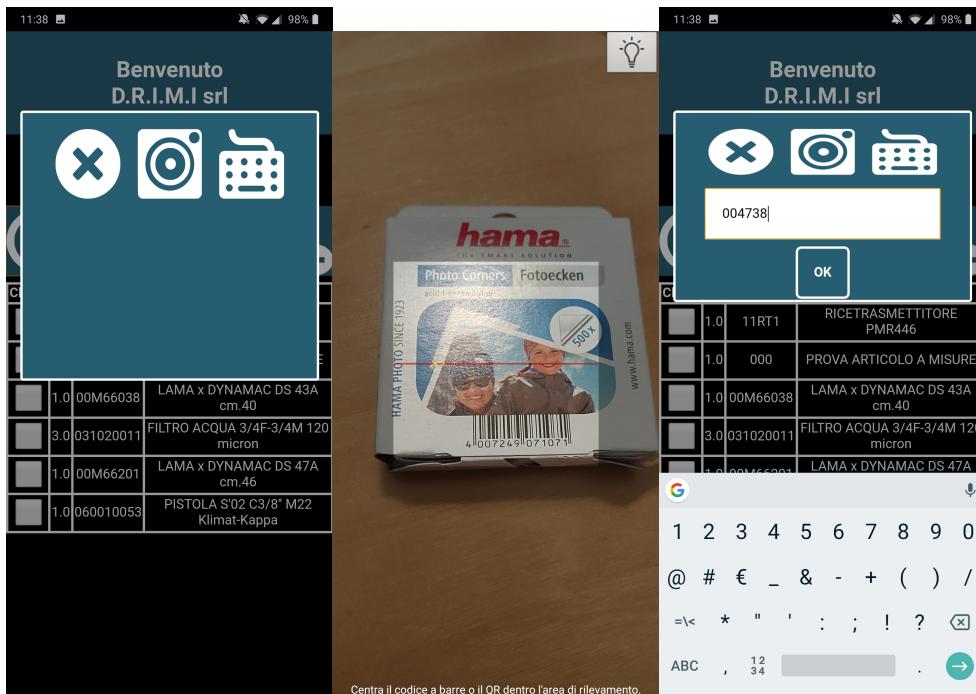


Figura 6.17: *Modal* di aggiunta articolo e modalità di aggiunta (scansione o inserimento manuale)

Il *modal* di aggiunta articolo permette di decidere la modalità di inserimento dell'articolo in carrello. Il *modal* presenta i seguenti pulsanti:

- * annullamento aggiunta: premendo su questo pulsante è possibile tornare alla *home page* di *moviORDER*;
- * scansione codice a barre: premendo su questo pulsante è possibile aggiungere un nuovo articolo scansionando il codice a barre dello stesso. Per la scansione del codice a barre viene aperta la fotocamera del dispositivo;
- * inserimento manuale: premendo su questo pulsante è possibile aggiungere un nuovo articolo inserendo manualmente il codice dello stesso.

Se la scansione del codice a barre o l'inserimento del codice articolo vanno a buon fine, viene aperta la pagina per l'aggiunta dell'articolo corrispondente.

6.3.4 Pagina di aggiunta articolo

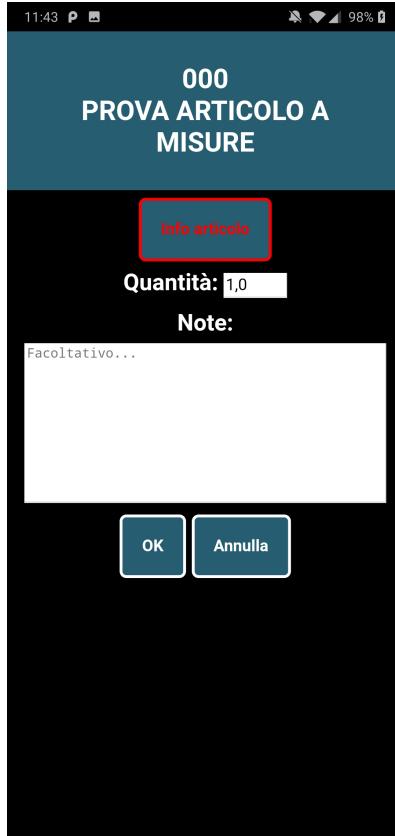


Figura 6.18: Pagina di aggiunta articolo

La pagina di aggiunta articolo permette di aggiungere un nuovo articolo al carrello e presenta le seguenti parti:

- * codice e nome articolo: in base al codice precedentemente scansionato o inserito, vengono visualizzate le informazioni relative al codice articolo e al nome dell'articolo;
- * pulsante di visualizzazione informazioni dell'articolo: premendo questo pulsante è possibile visualizzare le informazioni dell'articolo che si sta aggiungendo al carrello. Se il pulsante presenta bordo rosso significa che per l'articolo non sono presenti informazioni. Questo permette all'utente di evitare di pressare inutilmente sul pulsante;
- * *text-box* per l'inserimento della quantità: tramite questa *text-box* è possibile inserire la quantità da ordinare per l'articolo che si sta aggiungendo al carrello;
- * *text-area* per l'inserimento delle note: tramite questa *text-area* è possibile inserire delle note facoltative per l'articolo che si sta aggiungendo al carrello;

- * pulsante di conferma: tramite questo pulsante è possibile confermare l'aggiunta dell'articolo in carrello;
- * pulsante di annullamento: tramite questo pulsante è possibile annullare l'aggiunta dell'articolo in carrello e tornare alla *home page* di *moviORDER*.

6.3.5 Pagina di modifica articolo

Dalla *home page* di *moviORDER*, premendo su un qualsiasi articolo in carrello, è possibile modificare i dati d'ordine di tale articolo. Alla pressione dell'articolo viene aperta la pagina di modifica articolo. Questa pagina è identica alla pagina di aggiunta articolo ma contiene i dati inseriti nel momento in cui si è aggiunto l'articolo in carrello. È possibile modificare tali dati eseguendo le stesse interazioni previste per la pagina di aggiunta articolo.

6.3.6 Modal di invio ordine

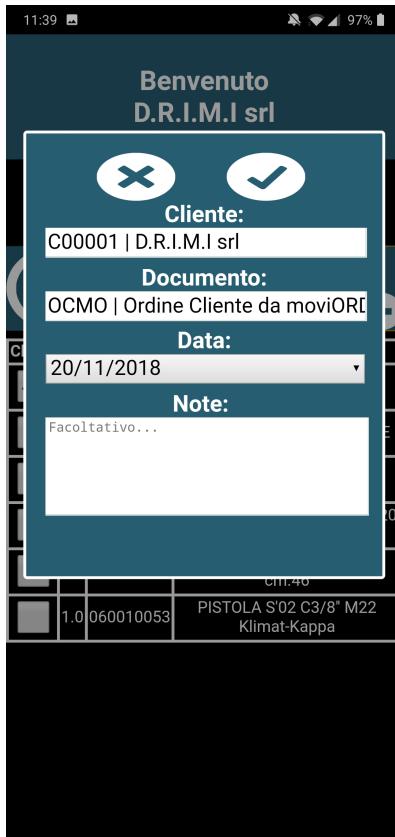


Figura 6.19: *Modal* di invio ordine

Il *modal* di invio ordine permette di inviare un ordine alla propria azienda. Tale ordine contiene tutti gli articoli precedentemente selezionati dal carrello. Il *modal* presenta le seguenti parti:

- * pulsante di annullamento: premendo questo pulsante è possibile annullare l'invio dell'ordine e tornare alla *home page* di *moviORDER*;
- * pulsante di conferma: premendo su questo pulsante è possibile confermare l'invio dell'ordine;
- * informazioni sul cliente: *text-box* contenente il codice e la ragione sociale del cliente che sta effettuando l'ordine;
- * informazioni sul documento: *text-box* contenente il codice e la descrizione del documento che deve essere generato una volta inviato l'ordine;
- * *select* per l'inserimento della data: questa *select* permette l'inserimento della data d'ordine. Di *default* viene proposta la data corrente;
- * *text-area* per l'inserimento delle note: questa *text-area* permette di inserire delle note facoltative per l'ordine che si sta inviando.

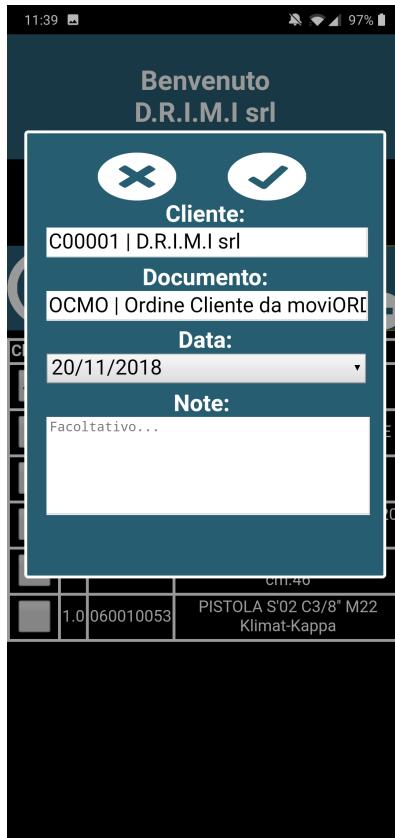


Figura 6.20: *Modal* di invio ordine

6.3.7 Considerazioni sullo sviluppo

L'applicazione *moviORDER* è stata realizzata tramite un framework cross-platform e quindi tramite la realizzazione di un'applicazione *web*. Per rendere l'applicazione usabile

su tutti gli smartphone è stato necessario progettare l’interfaccia dell’applicazione con un approccio *mobile first* e quindi cercando di realizzare un’interfaccia grafica che fosse più responsive possibile. Per raggiungere questo obiettivo ogni pagina di *moviORDER* presenta un layout elastico che utilizza unità di misura relative (*em* e *%*) le quali dipendono dalle preferenze dell’utente. Grazie a questo layout, le pagine si adattano correttamente ad ogni dimensione di schermo. Viene di seguito fornito, a scopo illustrativo, un esempio di codice *css* che utilizza unità di misura relative per la rappresentazione degli elementi dell’interfaccia.

```
label {
    color: white;
}

#selectQta {
    width: 15%;
}

#articleDesc {
    text-align:center;
    color: white;
    padding: 1.17em;
    background-color: #275D71;
    margin: 0;
}

#qta {
    width: 100%;
    text-align: center;
    margin-bottom: 0.5em;
}

#not {
    width: 100%;
    text-align: center;
    margin-bottom: 0.5em;
}
```

Figura 6.21: Codice *css* che utilizza unità di misura relative

Capitolo 7

Verifica e validazione

In questo capitolo vengono presentate le tecniche di verifica e validazione utilizzate durante il progetto.

7.1 Verifica

La verifica è il processo che si occupa di fornire prove oggettive che quanto prodotto in una fase di ciclo di vita di sviluppo del *software* soddisfi tutti i requisiti specificati per tale fase. Essa deve essere eseguita durante lo sviluppo, al fine di garantire consistenza, correttezza e completezza del prodotto *software* e della documentazione ad esso associata. È importante verificare ogni qualvolta che viene raggiunto un risultato intermedio tangibile, al fine di accertare che le attività di processo eseguite per raggiungere tale risultato non abbiano introdotto errori.

7.1.1 Analisi statica

L'analisi statica è una forma di verifica che studia le caratteristiche del codice sorgente e della documentazione ad esso associata senza richiedere l'esecuzione del prodotto in alcuna sua parte. Essa permette di verificare la conformità rispetto a norme progettuali, l'assenza di difetti e la presenza di proprietà positive nel codice. L'analisi statica risulta essenziale finché il sistema non raggiunge la sua completa disponibilità e deve essere applicata ad ogni prodotto di processo, al fine di verificare che quanto prodotto raggiunga la qualità attesa. Poiché il costo di rilevazione e di correzione degli errori cresce con l'avanzare del progetto, bisogna prestare attenzione nella scelta dell'approccio di verifica corretto. Nel caso del progetto si è cercato di utilizzare un approccio costruttivo, il quale consiste nel far accompagnare lo sviluppo con cicli di verifica, ottenendo la correttezza per costruzione.

7.1.1.1 Analisi statica del codice

7.1.1.1.1 Code reading

Per la logica applicativa dell'applicazione, essendo scritta in *JavaScript*, e quindi non compilata, si è utilizzata la tecnica del *code reading*. Essa consiste nell'esecuzione di un'attenta lettura individuale del codice per l'individuazione di errori e/o discrepanze con il progetto. Il lettore effettua mentalmente una pseudo-esecuzione del codice che lo conduce a verificarne la correttezza rispetto alle specifiche e agli standard adottati.

Questa tecnica è risultata utile nell'analisi della sintassi *JavaScript* poco conosciuta, ad esempio la sintassi dei *plugin PhoneGap* o la sintassi per l'utilizzo di *AJAX*.

7.1.1.1.2 Analisi statica in compilazione

Per il servizio *web*, essendo scritto il linguaggio *Java*, si è potuta utilizzare l'analisi statica in compilazione. I compilatori, per poter generare il *codice oggetto_G*, effettuano un'analisi statica del codice per verificare che un programma soddisfi particolari caratteristiche di correttezza statica. Tipici errori scovati a *compile-time* sono:

- * presenza di nomi di identificatori non dichiarati;
- * incoerenza tra tipi di variabili utilizzate in un'istruzione;
- * incoerenza tra parametri formali e parametri attuali nelle chiamate a metodi e/o funzioni: l'incoerenza potrebbe essere sul tipo e/o il numero di parametri.

7.1.1.2 Analisi statica della documentazione

Per la verifica della documentazione, essendo lo stagista da solo, si è dovuta utilizzare la tecnica *walkthrough*. Essa mira a rilevare la presenza di difetti eseguendo una lettura critica a largo spettro del prodotto in esame, questo perché vi è il sospetto che i difetti possano essere ovunque. Per l'esecuzione di un buon *walkthrough* è necessario che esso venga pianificato e che le attività svolte vengano documentate. Si è scelta questa tecnica perché nonostante abbia tempi di verifica molto lunghi, produce spesso documenti corretti.

7.1.2 Analisi dinamica

L'analisi dinamica è una forma di verifica che consiste nell'effettuare test sul prodotto in esecuzione. Un test è una verifica dinamica del comportamento del programma, effettuata su un insieme finito di casi, selezionati dal dominio di tutte le esecuzioni possibili. Poiché il dominio di tutte le esecuzioni possibili è potenzialmente infinito, i test non garantiscono l'esaustività, quindi l'analisi dinamica può rilevare la presenza di difetti ma non è in grado di dimostrarne l'assenza.

7.1.2.1 Test della logica applicativa

Per il test della logica applicativa si è utilizzata la tecnica del *debugging*. Essa consiste nella ricerca e nella correzione di difetti che sono causa di malfunzionamenti rilevati a *run-time*. Per il *debugging* del codice *JavaScript* si è dovuta utilizzare la *forza bruta*, ossia disseminare il codice di istruzioni di stampa per estrapolare quante più informazioni possibili sulla semantica del codice eseguito. Poiché *JavaScript* risulta essere un linguaggio difficilmente *debuggabile*, il *debugging* della logica applicativa è stato inefficiente e dispendioso in termini di tempo.

7.1.2.2 Test dell'API

Per il test dell'*API* di *moviORDER* si è utilizzato il *software Postman*, già precedentemente menzionato in sezione 3. Tale *software* permette di costruire una richiesta *HTTP*, di inviarla ad un *server* e di analizzare la risposta ricevuta. Nel caso del progetto sono state create richieste *HTTP POST* con i parametri richiesti dall'*API* e si sono successivamente analizzate le risposte in formato *JSON* ricevute dal *server*. Quindi

il *software* ha permesso di testare il comportamento del servizio *web* alla ricezione di richieste *HTTP* corrette e scorrette.

7.1.2.3 Test dell'applicazione

Per il test dell'applicazione, e quindi dell'usabilità e del funzionamento dell'interfaccia grafica, si è utilizzato l'emulatore dell'*IDE Android Studio*. Per il test della logica dell'applicazione si è considerato:

- * funzionamento dei buttoni: si è verificato che alla pressione di un bottone venisse riscontrato il comportamento atteso;
- * funzionamento del *plugin* per la scansione dei codici a barre: si sono testate tutte le tipologie di codici a barre richieste dal *tutor* aziendale.

Per il test del layout responsive dell'applicazione si sono utilizzati gli strumenti per gli sviluppatori del *browser Google Chrome*. In particolare si è testato il comportamento del layout su tutte le fasce di dispositivi, ovvero su schermi piccoli, medi e grandi ($phablet_G$).

7.2 Validazione

La validazione è il processo che si occupa di confermare, mediante esame e fornitura di prove oggettive, che le specifiche del *software* siano conformi alle esigenze dell'utente e all'uso previsto, e che i requisiti che il *software* implementa siano soddisfatti in maniera consistente. La validazione, a differenza della verifica, viene eseguita al termine del progetto e, affinché il prodotto venga accettato, è necessario eseguire continue verifiche durante il ciclo di vita dello stesso.

7.2.1 Validazione di moviORDER

La validazione di *moviORDER* è stata eseguita dal *tutor* aziendale tramite test dell'applicazione installata sul proprio *smartphone*. Il *tutor* ha analizzato i requisiti dell'applicazione e ha verificato che fossero tutti soddisfatti.

Capitolo 8

Conclusioni

In questo capitolo vengono presentate le conclusioni in merito al periodo di *stage* tenutosi presso l'azienda VisioneImpresa. In particolare vengono presentati:

- * un consuntivo finale;
- * il grado di soddisfacimento dei requisiti al termine del progetto;
- * le conoscenze acquisite;
- * una valutazione personale sullo *stage*.

8.1 Consuntivo finale

Le tempistiche e le modalità di svolgimento delle attività inizialmente concordate con il *tutor* aziendale sono state rispettate. Lo sviluppo di *moviORDER* ha avuto una durata di 260 ore e le restanti 60 ore sono state impiegate per accorgimenti correttivi sull'applicazione, per la documentazione del codice e per la stesura del manuale utente e sviluppatore. Lo studente ha allungato di un mese il periodo di *stage* per l'ottenimento di una borsa di studio e durante questo periodo è stato possibile rendere *moviORDER* un'applicazione pronta alla vendita. Viene di seguito presentata una tabella che mostra la differenza tra i tempi di sviluppo preventivati e quelli effettivi.

Attività	Ore pianificate	Ore effettive
Formazione assistita sui <i>software</i> gestionali di VisioneImpresa e sulle applicazioni analoghe a <i>moviORDER</i>	40	35
Formazione individuale sui <i>framework cross-platform</i> e scelta del <i>framework</i> ritenuto più opportuno	40	30

Attività	Ore pianificate	Ore effettive
Ridefinizione delle specifiche comprendente delle soluzioni da realizzare e delle metodologie per implementarle	40	40
Realizzazione dei <i>web services</i> per l'interazione con il <i>database</i> : servizio di autenticazione, servizio di lettura dati, servizio di scrittura dati e servizio di invio e-mail	40	45
Realizzazione della <i>business logic</i> dell'applicazione	40	35
Realizzazione delle interfacce grafiche dell'applicazione: login, gestione carrello, aggiunta/-modifica articolo e invio ordine	40	35
Test di scambio dati tra <i>VisionENTERPRISE</i> e <i>moviORDER</i>	40	40
Documentazione del codice e stesura del manuale utente e sviluppatore	40	60

Tabella 8.1: Consuntivo finale

Come si può osservare dal consuntivo, durante lo sviluppo dell'applicazione sono state risparmiate 20 ore in totale tra le diverse attività. Questo perché nelle prime settimane di *stage* lo stagista si è impegnato fortemente nel soddisfacimento dei requisiti funzionali obbligatori e quindi nello sviluppo del servizio, della logica applicativa e delle interfacce grafiche. Ciò ha permesso il rilascio dell'applicazione prima delle ferie estive di VisioneImpresa in modo da consentire al *tutor* aziendale di testare l'applicazione in serenità. Al rientro delle ferie lo stagista ha impegnato le ore risparmiate per la correzione dei bug riscontrati, per la documentazione del codice e per la stesura dei manuali dell'applicazione.

8.2 Grado di soddisfacimento dei requisiti

Al termine dello *stage* sono stati soddisfatti tutti i requisiti richiesti tranne uno (RVF9), poiché ritenuto dall'azienda di importanza irrilevante. Tale requisito richiedeva che l'applicazione permettesse la firma digitale del cliente a fine ordine. Viene di seguito fornita la tabella che illustra il grado di soddisfacimento dei requisiti.

Tipologia	Numero requisiti	Soddisfatti
Funzionale	102	102
Qualitativo	2	2
Di vincolo	9	8
Totale	113	112

Tabella 8.2: Grado di soddisfacimento dei requisiti

8.3 Conoscenze acquisite

Il periodo di *stage* ha permesso allo stagista di apprendere diverse nuove tecnologie e di approfondire l'utilizzo di tecnologie già in parte conosciute. Vengono di seguito presentate le tecnologie apprese durante lo *stage*.

8.3.1 JavaScript

Uno dei criteri per i quali lo stagista aveva scelto VisioneImpresa era l'utilizzo di *JavaScript* nello sviluppo del progetto. Questo perché il linguaggio *JavaScript* è sempre più richiesto dalle aziende e lo stagista ne aveva solamente una conoscenza basilare. Durante il periodo di formazione è stato possibile imparare ogni dettaglio di *JavaScript* puro seguendo i tutorial presenti sulla guida del sito *web W3C*. Questo ha permesso di prendere confidenza con il linguaggio e di realizzare la logica applicativa di *moviORDER* velocemente. In particolare, lo studente ha appreso l'utilizzo di *AJAX* come efficace strategia per inviare richieste *HTTP* tramite *JavaScript*.

8.3.2 SQL Server

Prima dello *stage* lo stagista conosceva solamente il *DBMS mySQL* perché imparato durante il corso di Tecnologie *Web*. Il periodo di *stage* ha permesso la comprensione di *SQL Server*, ampliando la conoscenza dello studente in termini di gestione di *database*.

8.3.3 Apache Tomcat e oggetti servlet

Prima dello *stage* lo studente non aveva mai realizzato un servizio *web* e non aveva idea di cosa fossero gli oggetti *servlet*, di quali fossero le loro applicazioni e di come rendere un servizio operativo. Lo *stage* ha permesso la comprensione delle tecniche per:

- * la realizzazione di un servizio *web* scritto in *Java* tramite l'utilizzo di oggetti *servlet*;
- * il deploy di un servizio sul *server web Apache Tomcat*.

8.3.4 PhoneGap

Prima dello *stage* lo studente non aveva mai realizzato un'applicazione *mobile*. Lo *stage* ha permesso di apprendere il *framework cross-platform PhoneGap* e di utilizzarlo per la realizzazione di applicazioni *mobile* su più piattaforme, in particolare *iOS* e *Android*. Si è appreso inoltre l'utilizzo delle funzionalità di base dei *software Android Studio* e *XCode*.

8.4 Sviluppi futuri

Lo stagista ritiene che l'interfaccia grafica realizzata sia attualmente utilizzabile esclusivamente come prototipo, per la preparazione di demo da mostrare a dei potenziali clienti. Si ritiene necessario migliorare il design dell'interfaccia, sia in termini di usabilità che in termini di resa grafica. Si osserva inoltre che la scelta di un *framework cross-platform* per lo sviluppo di un'applicazione che utilizza i sensori del dispositivo non è del tutto corretta. Astrarre il codice nativo per la gestione della fotocamera tramite un'interfaccia *JavaScript* diminuisce le performance dell'applicazione, sia in termini di memoria che in termini di batteria. In questo caso si sarebbe dovuto scrivere tale codice in linguaggio nativo. Si consiglia quindi, come sviluppo futuro, di effettuare il *refactoring* di tale parte dell'applicazione in codice nativo. Infine, avendo utilizzato il *framework PhoneGap*, si sono dovute utilizzare tecnologie *web* per lo sviluppo dell'applicazione. Si può quindi osservare che tramite piccoli accorgimenti sul codice del progetto è possibile realizzare una versione *web* di *moviORDER*, andando a completare l'intera piattaforma.

8.5 Valutazione personale

Lo stagista ritiene che l'esperienza di *stage* sia un'attività essenziale per comprendere il significato del lavoro. Non avendo mai lavorato era difficile capire le differenze tra il mondo universitario e quello lavorativo. Lo *stage* ha permesso di cogliere queste differenze e di cimentarsi in un'esperienza completamente nuova. Inoltre, per un laureato in informatica è importante avere esperienza lavorativa nel proprio bagaglio personale e lo *stage* obbligatorio ha permesso l'introduzione di tale esperienza.

Il periodo di *stage* presso VisioneImpresa ha permesso allo studente di calarsi nel mondo lavorativo e di cimentarsi nella realizzazione di un vero progetto *software* il quale prodotto verrà utilizzato quotidianamente dai clienti dell'azienda. Questo ha permesso di lavorare all'interno di un team di sviluppo e di comprendere il significato di responsabilità. Il progetto del corso di Ingegneria del *Software* aveva solamente fornito le basi di cosa significhi lavorare in modo responsabile. Grazie allo *stage* è stato possibile avere dei veri e propri task, assegnati da un titolare, e da completare entro una certa scadenza.

La preparazione del team di VisioneImpresa e la serenità dell'ambiente di lavoro hanno permesso fin da subito di abbattere ansie e insicurezze antecedenti al periodo di *stage* e di iniziare a lavorare in modo disciplinato e responsabile fin dall'inizio. Questo è stato possibile anche perché l'offerta di *stage* era d'interesse per lo studente e questo ha permesso di svolgere al meglio le attività e quindi di non vedere il lavoro con un'ottica di impegno universitario ma con un'ottica di opportunità per crescere tecnicamente ma anche a livello emotivo.

Le relazioni instaurate con il team di sviluppo e il *tutor* aziendale, gli impegni presi, il rispetto delle scadenze e degli orari lavorativi hanno incrementato le capacità organizzative e collaborative dello studente che saranno sicuramente utili nelle prossime esperienze lavorative.

Appendice A

Glossario

Agile
AJAX
API
Applicazione web
Architettura
Azure
Baseline
Big-bang-integration
Caso d'uso
Ciclo di vita
Code-n-fix
Codice nativo
Design pattern
Diagramma di Gantt
ERP
Framework
Framework cross-platform
HTTP
ICT
IDE
Incremento
JSON
Libreria
Media query
Milestone
MoviDOC
Multipiattaforma
Periodo di slack
Piattaforma
Processo
Responsive
Servizio web
Servlet
Software gestionale
Tecnologie web

Usabilità

Appendice B

Convenzioni

In questo capitolo vengono presentate le convenzioni utilizzate per la classificazione di casi d'uso e requisiti.

B.1 Casi d'uso

Ogni caso d'uso è classificato tramite il seguente formalismo:

$$\text{UC}\{\text{codice_identificativo}\}$$

dove:

- * **codice _ identificativo:** è un codice composto da una serie di numeri separati tramite punto che identificano il caso d'uso in maniera univoca e lo esprimono gerarchicamente.

B.2 Requisiti

Ogni requisito è classificato tramite il seguente formalismo:

$$\text{R}\{\text{X}\}\{\text{Y}\}\{\text{codice_identificativo}\}$$

dove:

- * **X** specifica la tipologia di requisito:
 - *F*: requisito funzionale;
 - *Q*: requisito qualitativo;
 - *V*: requisito di vincolo.
- * **Y** indica uno dei seguenti gradi di necessità:
 - *O*: requisito obbligatorio;
 - *D*: requisito desiderabile;
 - *F*: requisito facoltativo.
- * **codice _ identificativo:** è un codice composto da una serie di numeri separati tramite punto che identificano il requisito in maniera univoca e lo esprimono gerarchicamente.

Bibliografia