ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

ANALISI DEL FRAMEWORK .NET MAUI PER LO SVILUPPO DI APPLICAZIONI MULTIPIATTAFORMA E REALIZZAZIONE DI UN'APPLICAZIONE COME CASO DI STUDIO

Elaborato in
PROGRAMMAZIONE AD OGGETTI

Relatore
Prof. ALESSANDRO RICCI

Presentata da
ALESSANDRO VENTURINI

Anno Accademico 2022 – 2023

"In the world of cross-platform development, if the code compiles and runs on all platforms on the first try, you might be tempted to pinch yourself to see if you're dreaming."

Indice

Introduzione

Sviluppare un'applicazione in modo che questa possa essere eseguita su più piattaforme o tipologie di dispositivi è sempre stato un tema caldo nell'ambito dell'ingegneria informatica. Una delle caratteristiche che probabilmente ha contributo di più al diffondersi del linguaggio C è che fosse del tipo *write once - compile everywhere*[1]. Stessa cosa vale per il linguaggio Java che grazie alla sua *Java Virtual Machine* permette di compilare il codice in un linguaggio intermedio che poi può essere eseguito su qualsiasi piattaforma o macchina che supporti la *JVM*.

Tecnologie con lo scopo di ridurre il carico di lavoro dello sviluppatore nella produzione di applicativi multipiattaforma continuano a nascere anche oggi. Un esempio è .NET MAUI che, da poco sul mercato, offre la possibilità di sviluppare app mobile e desktop native che condividono un'unica *codebase*.

L'obiettivo di questa tesi è quello di andare ad approfondire il tema dello sviluppo multipiattaforma e valutare il framework .NET MAUI a seguito della realizzazione di un'applicazione per un'azienda (questo fa sì che lo standard realizzativo debba inevitabilmente essere alto).

Struttura della tesi

- **Capitolo 1**: analisi in dettaglio del tema dello sviluppo di applicazioni multipiattaforma, descrizione dei principali approcci ed enumerazione delle caratteristiche chiave che un framework per lo sviluppo multipiattaforma dovrebbe avere.
- **Capitolo 2**: descrizione specifica di cos'è .NET MAUI, delle sue fondamenta e di alcuni dei possibili framework concorrenti.
- **Capitolo 3**: presentazione del caso di studio.
- **Capitolo 4**: analisi di .NET MAUI alla luce del lavoro svolto.

[1] Con i giusti accorgimenti

Capitolo 1

Sviluppo di applicazioni multipiattaforma

1.1 Il problema dello sviluppo multipiattaforma

Per uno sviluppatore riuscire a portare su più piattaforme, e quindi su un numero maggiore di dispositivi, il prodotto che sta realizzando significa poter accedere ad una base di utenza più grande.

Raggiungere il maggior numero possibile di utenti è il motivo fondamentale per cui la possibilità di una applicazione di essere eseguita su più piattaforme è molto importante e, in certi casi, addirittura un requisito di progettazione. Si pensi ad esempio ad un'applicazione per la fruizione di un servizio pubblico: è giusto che tutti gli utenti, a prescindere dalla tipologia di dispositivo che possiedono, possano averne accesso.

I problemi che si possono incontrare quando si sviluppa un'applicazione che possa essere eseguita su più piattaforme dipendono strettamente dall'approccio scelto per lo sviluppo. Le principali metodologie sono elencate e spiegate di seguito.

1.2 Approccio *native*

Sviluppare un'applicazione con approccio *native* significa utilizzare gli strumenti e le tecnologie pensate appositamente per produrre un'applicazione che venga eseguita su una specifica piattaforma.

Questo approccio è in grado di portare ad un prodotto di massimo livello qualitativo.

Il problema di sviluppare la stessa applicazione in maniera *native* per più piattaforme risiede nel fatto che è **costoso**. Infatti uno sviluppo di questo tipo prevede:

- Lo sviluppo e il mantenimento di *codebase* separate
- Team che conoscano gli strumenti di sviluppo di ogni singola piattaforma

Si ha a che fare quindi con costi che dipendono da un maggiore tempo investito nello sviluppo e con costi diretti per la formazione o l'assunzione di sviluppatori con competenze specifiche.

Differenza tra *Approccio native* e *Applicazione native*

Un'applicazione nativa è tale in quanto utilizza in maniera diretta le funzionalià della piattaforma su cui viene eseguita. Questo dettaglio è importante perché aggiunge la possibilità di sviluppare applicazioni native anche con un approccio che non sia quello appena descritto. Un esempio è proprio quello di .NET MAUI (il framework oggetto della tesi) che pur avendo come output un'applicazione nativa non utilizza un approccio *native*

1.3 Approccio web

Data la natura particolarmente costosa dell'approccio *native* si è andati alla ricerca di soluzioni alternative che potessero risolvere questo problema.

Le applicazioni web sfruttano il fatto che i browser sono decisamente più uniformati e standardizzati rispetto ai sistemi operativi. Questo è particolarmente evidente, basta visitare lo stesso sito o applicativo web da dispositivi con sistema operativo differente e si potrà verificare che le differenze sono veramente minime.

Questo tipo di applicazioni quindi vengono progettate per essere fruite attraverso il browser e si basano su i classici stack di sviluppo web.

Le applicazioni web normalmente permettono di ottenere un'applicazione funzionante con una grande percentuale di riutilizzo del codice e una larga base di utenza. Per ottenere però questi benefici si deve scendere ai seguenti compromessi:

- Prestazioni ridotte
- Rinuncia al *look and feel* dell'applicazione nativa (spesso le piattaforme adottando internamente degli standard grafici o funzionali al quale l'utente si abitua)

- Necessità dell'app di essere eseguita nel browser
- Utilizzare le funzionalità specifiche (anche hardware) del dispositivo diventa più complicato e non sempre possibile

1.4 Approccio ibrido

Le applicazioni ibride sono una fusione dei due approcci presentati sopra. Si compongono infatti di un nucleo funzionale sviluppato con tecnologie web e di un contenitore nativo.

Accade infatti, in maniera abbastanza frequente, di entrare in contatto con applicazioni che vengono effettivamente installate sul dispositivo, ma che una volta aperte non fanno altro che renderizzare pagine web attraverso il browser di sistema. Questo permette di rendere più semplice l'utilizzo di funzionalita specifiche del sistema ma con i seguenti problemi:

- Percentuale di riutilizzo del codice inferiore rispetto alle applicazioni web
- Prestazioni ridotte
- Rinuncia al *look and feel* dell'applicazione nativa

1.5 Caratteristiche chiave di un framework per lo sviluppo multipiattaforma

In questa sezione si andranno ad analizzare quali sono le caratteristiche importanti per un framework per lo sviluppo multipiattaforma. Queste caratteristiche permettono anche quindi di valutare le differenze tra i framework e capire quale sia più adatto per una situazione piuttosto che per un'altra.

- **Piattaforme supportate**
- **Tipologie di piattaforme supportate** (le più importanti sono: *mobile*, *desktop* e *web*)
- **Performance**
- **Interfaccia grafica** (Mostrare all'utente un'interfaccia che rispetti le linee guida della piattaforma da lui adottata)
- **Accesso alle funzionalità del dispositivo**
- **Percentuale di riutilizzo del codice**

- **Strumenti di assistenza allo sviluppo** (ad esempio integrazione del supporto ai simulatori)

Queste caratteristiche non sono in ordine di importanza ed è fondamentale sottolineare che molto spesso sono in conflitto tra di loro. L'obiettivo di un framework che vuole imporsi in questo settore è quello di individuare il giusto compromesso tra queste qualità.

Il panorama di framework tra i quali scegliere è molto vasto, alcuni tra i framework più conosciuti sono:

- React Native
- Flutter
- Xamarin
- Apache Cordova
- Ionic
- Electron
- Qt
- NativeScript

In questa tesi si vuole andare ad analizzare .NET MAUI in quanto uno tra i più recenti e promettenti (date le sue fondamenta) comparsi sul mercato.

Capitolo 2

.NET MAUI

2.1 Cos'è .NET MAUI

.NET MAUI è un framework *open-source* multipiattaforma che permette di realizzare applicazioni interamente native. È possibile utilizzare una unica *codebase* sia per quanto riguarda la parte funzionale che per l'interfaccia grafica, ma viene comunque fornita la possibilità di scrivere codice *platform-specific*.

Le piattaforme supportate sono le seguenti:

- Android
- iOS
- Windows
- MacOS
- Tizen (anche se nella documentazione ufficiale viene nominato solo alcune volte)

2.1.1 Storia

.NET MAUI è l'evoluzione di Xamarin.Forms, un framework che, da ormai più di un decennio, permette di sviluppare applicazioni multipiattaforma **mobile** native.

I controlli dell'interfaccia grafica sono stati interamente ricostruiti e la struttura del progetto è cambiata in maniera drastica, queste sono le differenze principali tra .NET MAUI e Xamarin.Forms. Comunque uno sviluppatore con esperienza in Xamarin.Forms si sentirebbe decisamente a casa se decidesse di spostarsi su .NET MAUI.

2.1.2 Obiettivi

.NET MAUI si propone come soluzione ai più importanti problemi visti nel capitolo precedende. Infatti:

- La percentuale di riuso del codice dovrebbe essere molto alta
- Le prestazioni dovrebbero essere comparabili a quelle di un'applicazione *native*
- È possibile raggiungere un *look and feel* da applicazione nativa

2.2 Come funziona

.NET 6 e le versioni successive forniscono 4 framework specifici per creare applicazioni nelle diverse piattaforme (.NET for Android, .NET for iOS, .NET for Mac, WinUI 3). Sarebbe già possibile utilizzare direttamente questi framework per realizzare le singole applicazioni ma è proprio qui che entra in gioco .NET MAUI che, come è possibile vedere nella figura 2.1, si occupa di astrarre i sopra citati framework per fornire un'unica interfaccia alla creazione delle applicazioni.

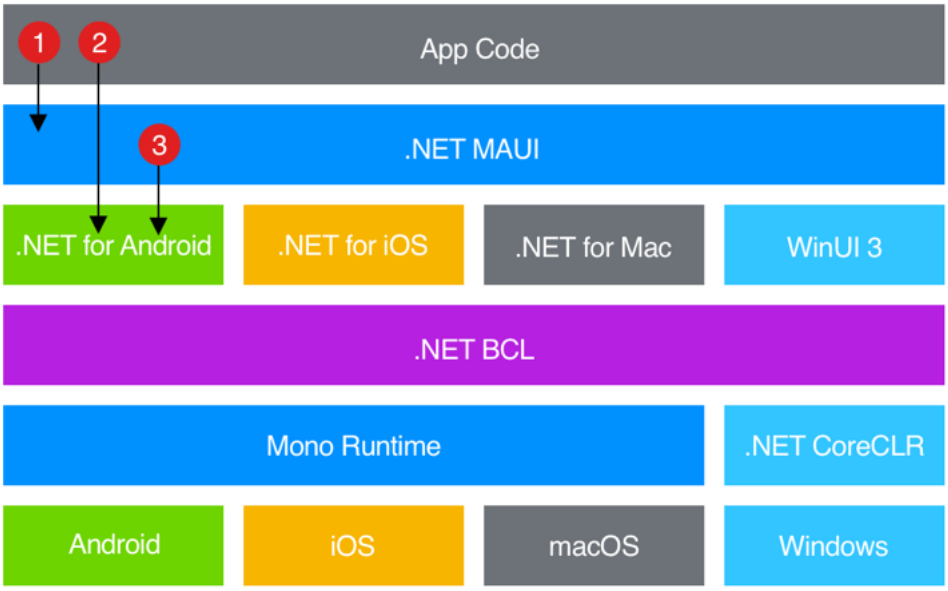


Figura 2.1: Diagramma dell'architettura di .NET MAUI.
`https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui\#how-net-maui-works`

Nel diagramma 2.1 è inoltre messo in evidenza il fatto che, il codice scritto dallo sviluppatore (*App Code*), può interfacciarsi non solo con .NET MAUI (1) ma anche direttamente con i framework *platform-specific* (2). È proprio questo dettaglio a garantire una enorme possibilità di personalizzazione del prodotto in base alla piattaforma.

Per quanto riguarda Android l'applicazione viene compilata in un linguaggio intermedio per la distribuzione e in un assembly nativo attraverso la compilazione *just-in-time* quando l'app viene lanciata sul dispositivo. La versione iOS dell'applicazione viene compilata direttamente in codice assembly. Per generare l'applicazione per MacOS viene sfruttato Mac Catalyst, una soluzione fornita da Apple che permette di portare su MacOS una qualsiasi applicazione iOS. Per quanto riguarda Windows l'applicazione viene compilata direttamente per essere eseguita su questo sistema operativo

2.3 Cosa deve imparare chi vuole sviluppare con .NET MAUI

A meno che uno sviluppatore non abbia avuto esperienza con Xamarin.Forms o WPF probabilmente prima di iniziare a programmare dovrà prendersi del tempo per studiare il framework e capire come funziona. In questa sezione vengono descritte le cose più importanti da imparare per poter iniziare a sviluppare con .NET MAUI.

2.3.1 Utilizzare la documentazione ufficiale

La documentazione ufficiale (`https://learn.microsoft.com/en-us/dotnet/maui/`) è davvero ben fatta e completa rimanendo comunque semplice grazie ai tanti esempi proposti.

Lo sviluppatore che intende imparare questo framework passerà parecchio tempo a leggere documentazione, non è tempo sprecato. Anche se si dovesse avere conoscenze pregresse in Xamarin.Forms leggendo la documentazione ci si potrà rendere conto di alcuni cambiamenti che aprono nuove possibilità.

2.3.2 Struttura della *solution*

La struttura della *solution* cambia radicalmente rispetto a Xamarin.Forms. L'antenato di .NET MAUI sfruttava 3 *projects*:

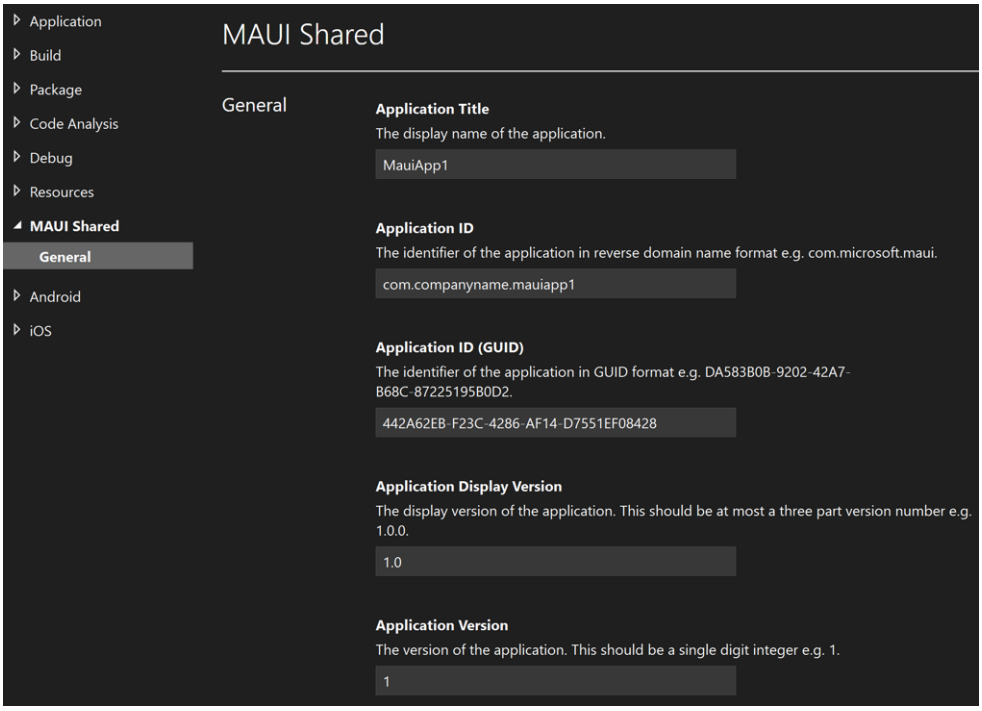
- Uno per iOS
- Uno per Android

- Uno condiviso

L'idea era che nel caso in cui non si volesse modificare il comportamento specifico per una piattaforma fosse possibile sviluppare l'intera app lavorando solo sul progetto condiviso. In realtà erano diverse le situazioni nelle quali era necessario andare a lavorare sul *project* della singola piattaforma, ad esempio modificare i file di configurazione **Info.plist** e **AndroidManifest.xml** oppure inserire risorse come le immagini che andavano gestite separatamente per le piattaforme.

.NET MAUI invece utilizza una *solution* con un singolo *project* nel quale le risorse (come le immagini) vengono gestite automaticamente, i file di configurazione possono essere modificati indirettamente passando attraverso un'unica interfaccia (figura 2.2).

Figura 2.2: Interfaccia condivisa per modificare alcune delle informazioni necessarie nei *manifest* delle applicazioni.
`https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/single-project\#app-manifest`



2.3.3 C#

C# è un linguaggio oggi molto utilizzato e presenta diverse caratteristiche che sono fondamentali per rendere efficace lo sviluppo di applicazioni in .NET

MAUI. Ad esempio la programmazione ad eventi, uno sviluppato sistema di *reflection*, e la possibilità di usare proprietà e attributi nelle classi.

Inoltre il quantitativo di tutorial, informazioni e librerie che si possono trovare online per questo linguaggio sono spropositate. Questo lo rende veramente comodo per sviluppare tutta la *business logic* dell'applicazione.

Da questo punto di vista .NET MAUI non si differenzia da Xamarin.Forms, è infatti molto probabile che tutta la parte funzionale dell'applicazione possa essere spostata tra i due framework senza modifiche.

2.3.4 XAML

XAML sta per *eXtensible Application Markup Language* ed è un linguaggio di markup basato su XML che è stato utilizzato per descrivere l'interfaccia grafica in diversi framework (ad esempio WPF, Xamarin.Forms e adesso .NET MAUI).

In .NET MAUI normalmente un file .xaml è associato ad un secondo file .xaml.cs, questo perché consistono in due definizioni parziali della stessa classe C#. Questo significa che è possibile definire l'interfaccia in maniera dichiarativa (nel file xaml) esattamente come è possibile farlo in maniera imperativa (nel file .xaml.cs) in C#. Risulta conveniente descrivere l'interfaccia in XAML e utilizzare il file di *code-behind* solo per aggiungere funzionalità complesse come ad esempio degli *handler* per gli eventi generati dai componenti grafici.

XAML non è particolarmente leggibile ed è anche decisamente verboso, inoltre la sintassi per utilizzare alcune funzionalità non è molto intuitiva e bisogna spesso ricorrere alla documentazione ogni volta che si intende fare qualcosa di nuovo.

Questo linguaggio però si sposa benissimo con il pattern MVVM che verrà descritto nella sezione successiva.

2.3.5 MVVM

MVVM è l'acronimo di *Model View ViewModel* e consiste in un pattern architetturale per mettere insieme e gestire le parti fondamentali di un'applicazione grafica: la *UI* e la logica applicativa

MVVM è perfetto per ridurre la complessità della sincronizzazione tra i dati e l'intefaccia grafica rispetto ad esempio al pattern MVC (utilizzato in framework più datati tipo UIKit). Infatti per ogni *view* si ha un *view model* che al contrario del *controller* in MVC non deve controllare gli aggiornamenti dei dati di ogni componente ma solo definire quali sono i dati che la *view* deve utilizzare. L'aggiornamento della *view* viene poi effettuato grazie alla

sottoscrizione ad eventi eventi innescati automaticamente dalla modifica dei dati nel *view model* (questo meccanismo prende il nome di *data binding*).

Va però riconosciuto che, in .NET MAUI, l'utilizzo del pattern MVVM non è semplice ed è molto verboso se non si approfitta di librerie apposite (vedi la sezione 4.2.1).

2.4 Framework concorrenti/alternativi

Di seguito un breve elenco delle principali alternative a .NET MAUI mettendone in evidenza le differenze, i vantaggi e gli svantaggi. [1]

2.4.1 React Native

React Native è un framework sviluppato da Meta (ex Facebook) che permette di creare applicazioni native mappando le primitive di React (libreria JavaScript per interfacce grafiche) nei controlli nativi di Android e iOS.

Questo consente la creazione di applicazioni *native*. Bisogna però tenere in considerazione che il funzionamento è diverso rispetto a .NET MAUI. Infatti React Native sfrutta il motore JavaScript presente sulle piattaforme per interpretare il codice dell'applicazione che poi si interfaccia con le primitive della piattaforma. Questo fa sì che ci sia comunque uno strato in più rispetto ad una vera applicazione *native*.

Come si è spiegato nella sezione precedente .NET MAUI riesce a far eseguire sulle piattaforme direttamente codice assembly questo permette di ottenere un risultato veramente nativo.

Inoltre per quanto riguarda React Native il supporto per le piattaforme desktop (Windows e MacOS) non è fornito direttamente ma bensì attraverso delle estensioni (`https://microsoft.github.io/react-native-windows/`) manutenute da Microsoft. Mentre su .NET MAUI le due piattaforme desktop sono parte centrale del framework tanto quanto quelle mobile.

2.4.2 Flutter

Flutter è un framework sviluppato da Google che permette lo sviluppo di applicazioni multipiattaforma native. Come .NET MAUI il prodotto finito consiste in codice assembly quindi permette prestazioni paragonabili a quelle

[1] Per poter considerare un framework come valida alternativa si sono usati come criteri il fatto che le applicazioni prodotte fossero *native* (o almeno ibride) e che sia le piattaforme mobile che desktop fossero supportate.

native. Inoltre consente lo sviluppo su tutte le piattaforme supportate da .NET MAUI (in più anche Linux).

Una delle differenze più sostanziali tra le due piattaforme è che Flutter non utilizza i componenti grafici nativi della piattaforma ma bensì sfrutta un motore di rendering per disegnare direttamente i propri componenti. Questo ha il vantaggio di portare ad un aspetto consistente tra le diverse piattaforme ma allontana l'applicazione dal *look and feel* specifico della piattaforma.

Capitolo 3

Caso di studio

Al fine di effettuare una valutazione realistica e fondata di .NET MAUI si è deciso di utilizzarlo per realizzare un'applicazione, inoltre questa verrà veramente impiegata in ambito aziendale e questo fa sì che si debbano necessariamente analizzare tutti gli aspetti che portano alla realizzazione di un prodotto, senza poter fare semplificazioni o approssimazioni tipiche di un "progettino universitario".

Di seguito viene quindi presentata l'applicazione che si è realizzata per questa tesi. Questo capitolo è molto importante in quanto permette di contestualizzare i ragionamenti e le considerazioni descritte in questo documento in quanto sono tutte derivate dall'esperienza maturata durante la realizzazione di questo progetto.

Dato il fatto che la tipologia dell'applicazione prodotta non copre tutto lo spettro di applicazioni che è possibile realizzare in .NET MAUI, non è stato possibile valutare tutte le possibilità del framework e di conseguenza le considerazioni effettuate si riferiscono esclusivamente a quelle funzionalità che si sono toccate con mano.

Il progetto consiste nella riscrittura dell'applicazione Salus Carichi (iOS) in .NET MAUI per renderla multipiattaforma. Nella sezione seguente si va a presentare l'applicazione originale e in quelle successive si tratterà il processo di riscrittura in .NET MAUI.

3.1 L'applicazione Salus Carichi

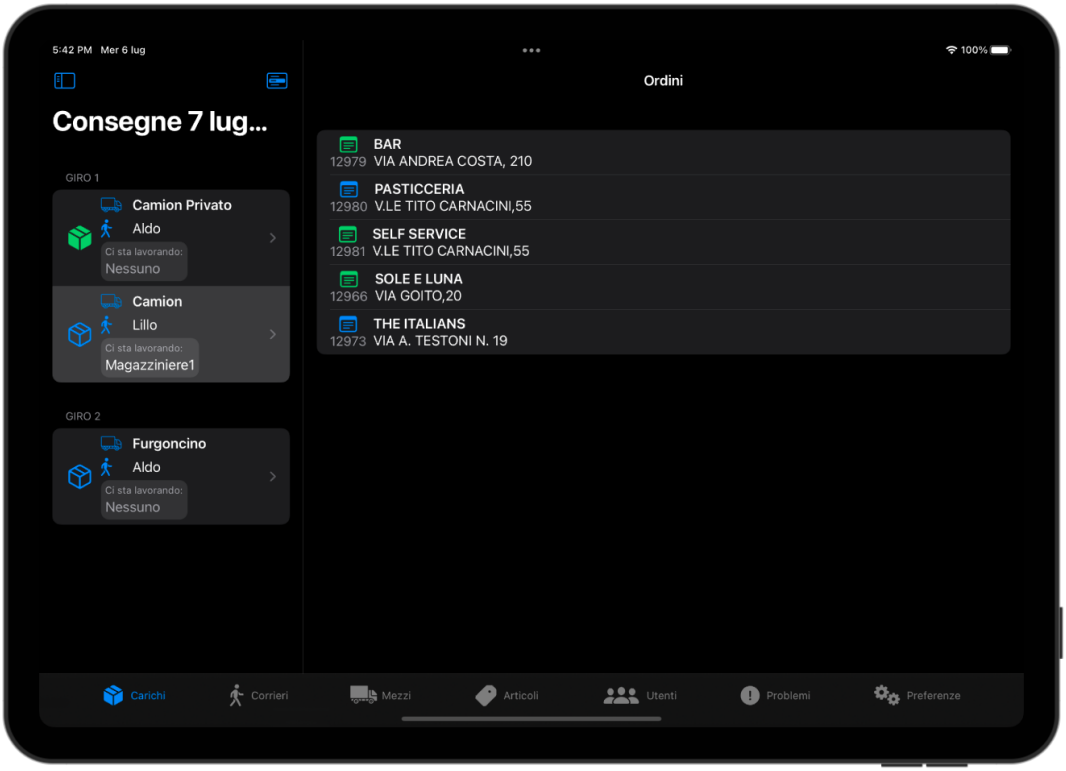
Salus Carichi è un progetto che è stato sviluppato su misura per l'azienda Salus s.r.l. dal 2020 al 2022, consiste in un sistema di due applicazioni iOS che permettono di velocizzare e migliorare la fase di pianificazione dei carichi da spedire e di ridurre le possibilità di errori umani durante la fase di caricamento dei mezzi di trasporto.

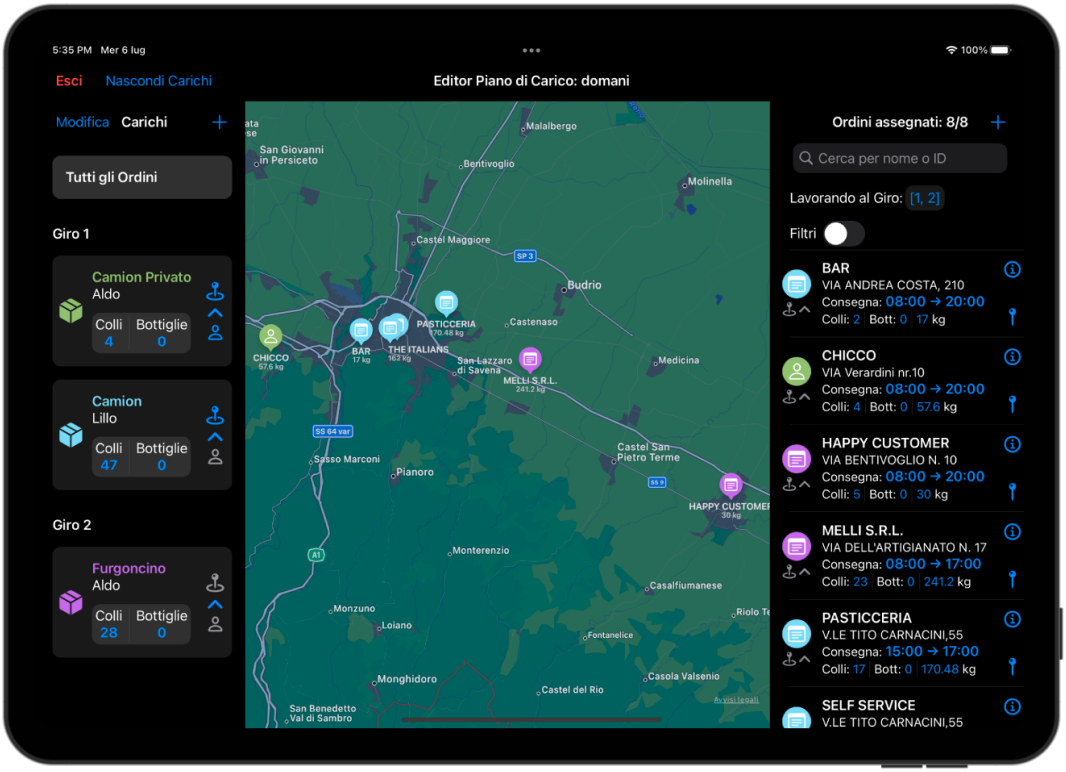
3.1.1 Salus Carichi Manager

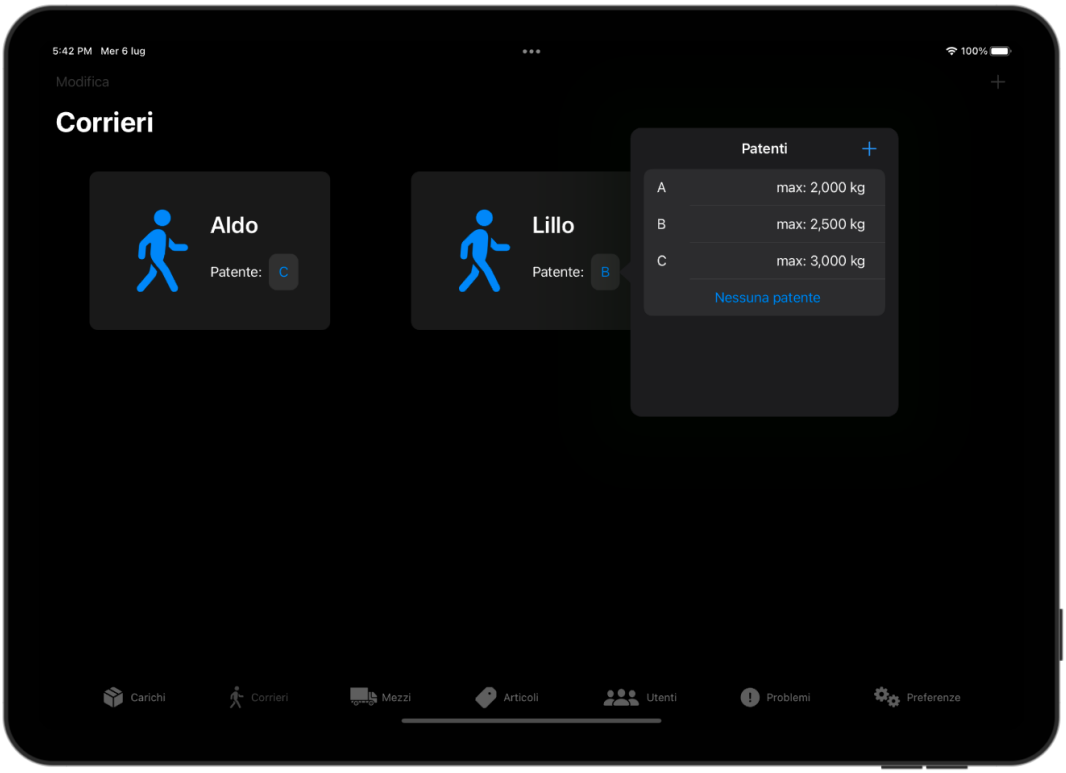
Salus Carichi Manager è l'applicazione più complessa, pensata per iPad e destinata ad essere utilizzata da colui che pianifica i carichi, che implementa le seguenti funzionalità:

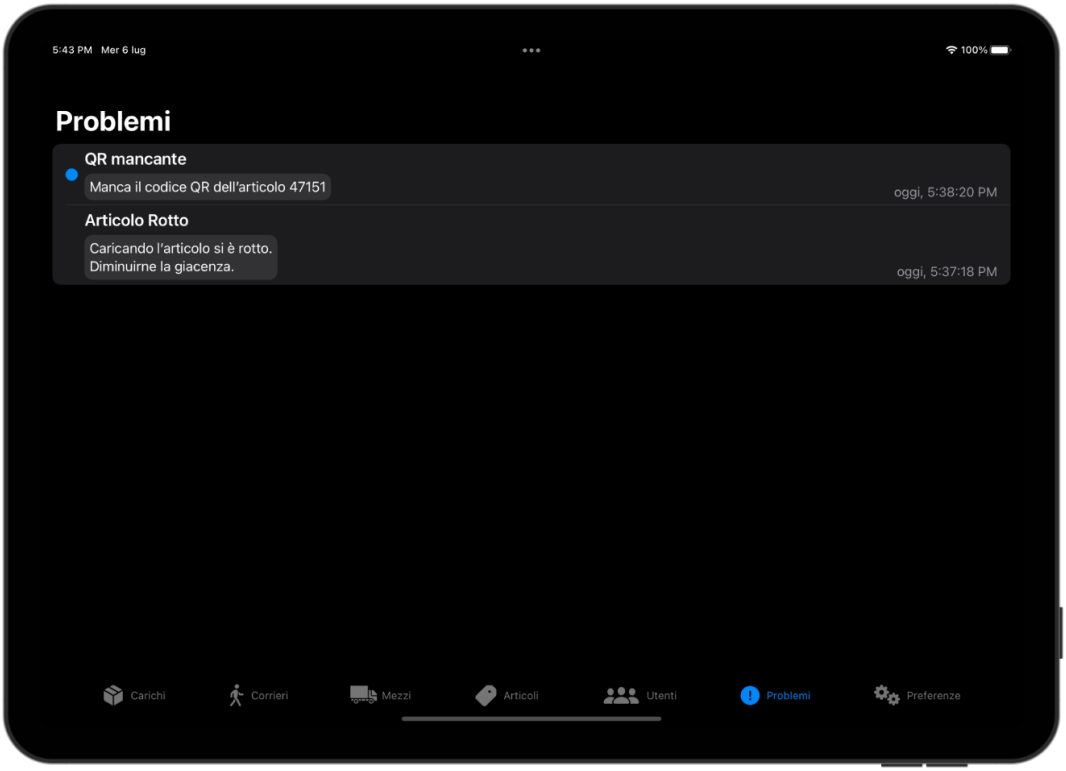
- Possibilità di importare gli ordini dal gestionale.
- L'editor di piani di carico permette di organizzare i carichi per i giorni a seguire.
- La visualizzazione su mappa consente di pianificare i carichi nel miglior modo possibile.
- Le informazioni sul peso calcolate automaticamente assicurano che i mezzi non siano mai sovraccarichi.
- Controllo in tempo reale di quanti e quali articoli siano stati caricati e quali no.
- Controllo in tempo reale di quali magazzinieri stanno lavorando su uno specifico carico.
- Ricezione di informazioni su eventuali problemi in magazzino.
- Possibilità di gestire la cache sul dispositivo.

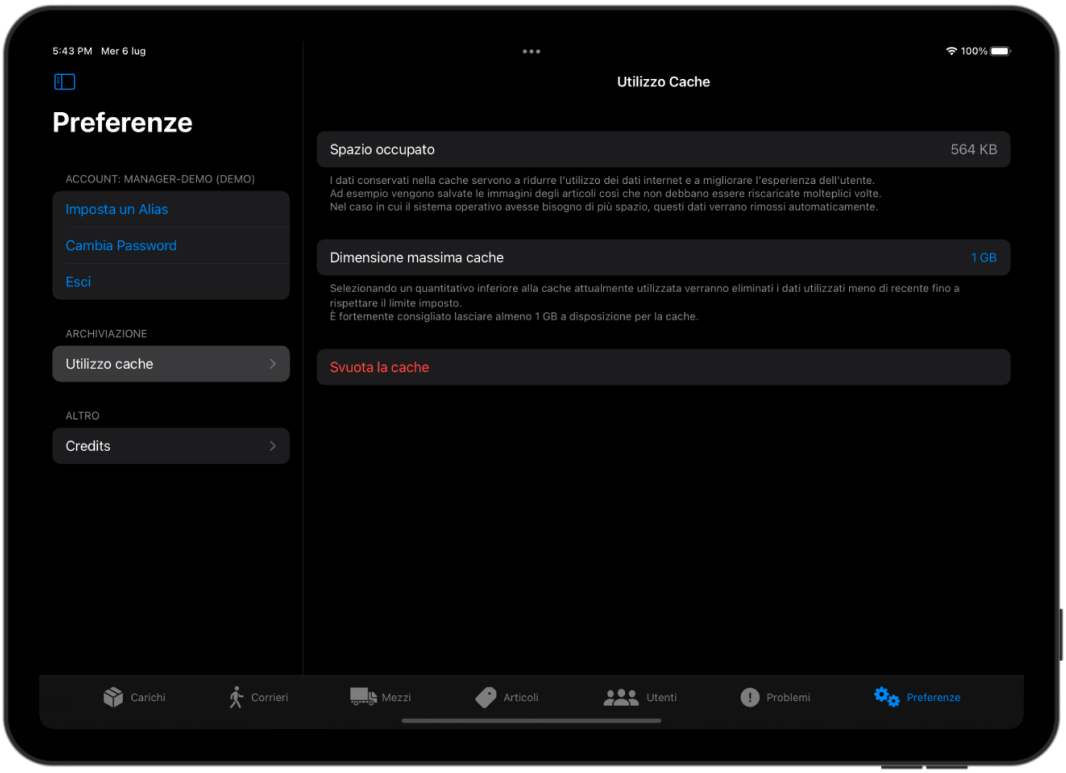
Figura 3.1: App Salus Carichi Manager per iOS






5:42 PM Mer 6 lug
100%
Modifica
Corrieri
Aldo
Patente: C
Lillo
Patente: B
Patenti
A max: 2,000 kg
B max: 2,500 kg
C max: 3,000 kg
Nessuna patente
Carichi
Corrieri
Mezzi
Articoli
Utenti
Problemi
Preferenze


5:43 PM Mer 6 lug
100%
Problemi
QR mancante
Manca il codice QR dell'articolo 47151
oggi, 5:38:20 PM
Articolo Rotto
Caricando l'articolo si è rotto.
Diminuirne la giacenza.
oggi, 5:37:18 PM
Carichi
Corrieri
Mezzi
Articoli
Utenti
Problemi
Preferenze

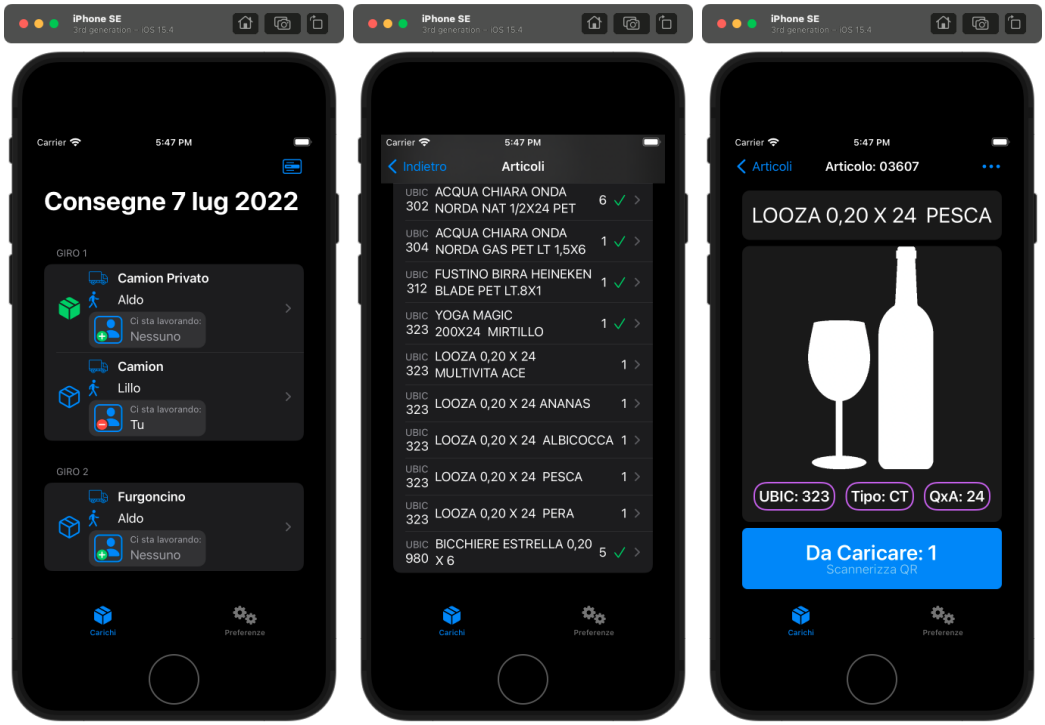


3.1.2 Salus Carichi

Salus Carichi è l'applicazione, pensata per iPhone e destinata ad essere usata dai magazzinieri, che implementa le seguenti funzionalità:

- E' possibile ricevere direttamente sul dispositivo una lista dei prodotti da caricare.
- Grazie ai codici QR è impossibile sbagliare articolo da caricare.
- L'assegnamento dei magazzinieri ai singoli carichi permette di organizzarsi meglio.
- Possibilità di monitorare lo stato di ogni carico in tempo reale.
- Possibilità di gestire la cache sul dispositivo.

Figura 3.4: App Salus Carichi per iOS



3.2 Perché in questo caso è utile lo sviluppo multipiattaforma

In questo specifico caso, la possibilità di sviluppare queste due applicazioni con un approccio multipiattaforma porta ai seguenti vantaggi:

- Per quanto riguarda l'applicazione per i magazzinieri il fatto di supportare anche Android significherebbe poter far utilizzare l'applicazione direttamente sui telefoni degli operatori senza la necessità, da parte dell'azienda, di acquistare dei dispositivi appositi. O nel caso in cui si decidesse comunque di acquistare dei dispositivi per lo scopo si potrebbe optare per dispositivi Android che mediamente hanno un prezzo più contenuto.

- Per quanto riguarda l'applicazione per l'addetto alla pianificazione sarebbe molto vantaggioso non dipendere da un unico dispositivo (l'iPad) ma poter utilizzare l'applicazione anche su desktop.

Sebbene l'applicazione iOS già esista è comunque più vantaggioso riscriverla in .NET MAUI piuttosto che andare solo a creare la versione Andorid nativa

in quanto la prima opzione consentirebbe nel lungo termine di manutenere una sola *code-base* invece di due (inoltre non si avrebbe accesso alle piattaforme desktop).

3.3 Piano di lavoro

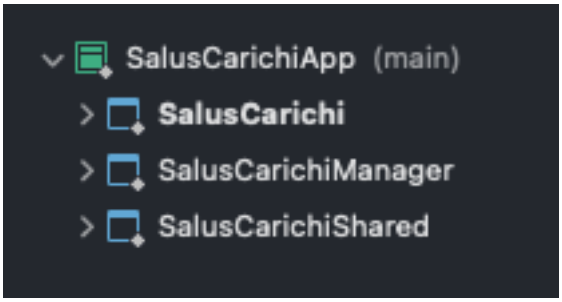
Inizialmente verrà portata solo l'applicazione per i magazzinieri (Salus Carichi), in quanto più urgente e più veloce da realizzare. Solo in futuro verrà portata anche l'applicazione per la pianificazione dei carichi (Salus Carichi Manager).

Bisogna comunque tenere conto che le due applicazioni condividono una buona percentuale di codice e questo significa che pur puntando a realizzare interamente una prima dell'altra si dovrà comunque progettare delle fondamenta che possano poi accogliere anche la seconda app in futuro.

3.4 Struttura della *solution*

.NET MAUI, a differenza di Xamarin.Forms imposta la *solution* di Visual Studio con un unico *project*. Siccome ad un *project* corrisponde un'applicazione in output, e dato il fatto che in questo caso dovranno essere prodotte due applicazioni che condivideranno parte del codice si è deciso di strutturare la *solution* nel seguente modo.

Figura 3.5: Struttura della solution



- 1 *project* di tipo *library* che contenga il codice condiviso da entrambe le applicazioni (Salus Carichi Shared).
- 2 *project* di tipo *executable* (dipendenti dal primo *project*) rispettivamente per l'applicazione Salus Carichi e per l'applicazione Salus Carichi Manager.

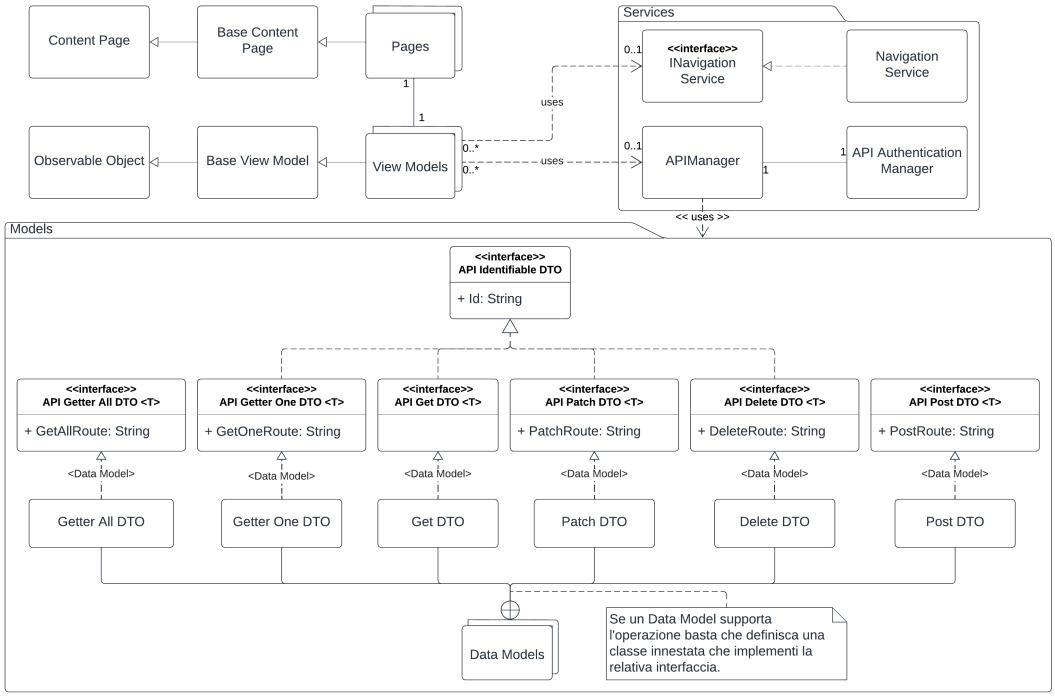
In questo modo è possibile condividere il codice nel progetto e andare a configurare le singole applicazioni (ad esempio i *bundle identifiers*, i dispositivi supportati, ecc...). [1]

3.5 Architettura dell'applicazione

Si sono definite delle classi con funzionalità base (tipo *callback* di navigazione e indicatori di attività) come *BaseContentPage* e *BaseViewModel* in modo che poi potessero essere ereditate.

È inoltre possibile vedere il servizio di navigazione del quale si parla nella sezione 3.7 e l'*APIManager* che si occupa della comunicazione con l'API web del sistema. Entrambi questi servizi vengono iniettati nell'applicazione all'avvio, questo diminuisce l'accoppiamento e inoltre favorisce il testing.

Figura 3.6: Diagramma UML dell'architettura dell'applicazione.
Il diagramma è semplificato e sono presenti solo le classi più importanti, inoltre quelle simili sono raggruppate (Views, View Models e Data Models)



[1] È evidente che dover produrre due applicazioni diverse che condividano buona parte del codice non sia uno dei casi più comuni e quindi questo tipo di soluzione non viene proposto nella documentazione ufficiale. Questo ha fatto sì che non fosse così banale riuscire ad identificare questo tipo di struttura come soluzione al requisito in questione.

3.6 Networking e API

Sono state implementate subito le funzionalità di autenticazione, registrazione e gestione dell'account.

Poi si è dedicato del tempo ad implementare un sistema di interfacce generico (vedi il package *Models* in figura 3.6) in maniera tale da permettere all'*APIManager* di definire funzioni tipo:

Figura 3.7: Alcuni metodi della classe *APIManager* che permettono di interfacciarsi con l'API web in modo generico rispetto al tipo di entità sulla quale si vuole eseguire l'operazione.

```
public async Task<TGetDTO> GetOne<T, TGetterOneDTO, TGetDTO>(TGetterOneDTO getterDTO)
    where TGetterOneDTO : APIGetterOneDTO<T> where TGetDTO : APIGetDTO<T> {
    HttpRequestMessage req = new(HttpMethod.Get, $"{getterDTO.GetOneRoute}/{getterDTO.id}");

    var response = await AuthenticateAndSend(req);

    return (await response.Content.ReadFromJsonAsync<TGetDTO>())!;
}

public async Task<IList<TGetDTO>> GetAll<T, TGetterAllDTO, TGetDTO>(TGetterAllDTO getterAllDTO)
    where TGetterAllDTO : APIGetterAllDTO<T> where TGetDTO : APIGetDTO<T> {
    HttpRequestMessage req = new(HttpMethod.Get, $"{getterAllDTO.GetAllRoute}");

    var response = await AuthenticateAndSend(req);

    return (await response.Content.ReadFromJsonAsync<List<TGetDTO>>())!;
}

public async Task<TGetDTO> PatchOne<T, TPatchDTO, TGetDTO>(TPatchDTO patchDTO)
    where TPatchDTO : APIPatchDTO<T> where TGetDTO : APIGetDTO<T> {
    HttpRequestMessage req = new(HttpMethod.Patch, $"{patchDTO.PatchRoute}");
    req.Content = new StringContent(JsonSerializer.Serialize(patchDTO), Encoding.UTF8, "application/json");

    var response = await AuthenticateAndSend(req);

    return (await response.Content.ReadFromJsonAsync<TGetDTO>())!;
}
```

La comodità di un approccio simile è che ogni entità (vedi figura 3.8) può definire i propri dati per le varie operazioni HTTP, i propri *endpoint* e può non supportare un'operazione semplicemente non implementandone l'interfaccia. È sicuramente un po' verboso ma decisamente flessibile e quindi perfetto per un'applicazione che deve poter evolvere velocemente.

Questo è l'approccio che si era adottato anche nella vecchia applicazione iOS, con alcuni adattamenti dovuti all'impossibilità in C# di richiedere in un'interfaccia che la classe implementante definisca una variabile statica. Questo avrebbe permesso, ad esempio, di non passare nessun parametro al metodo *GetAll* in quanto per conoscere l'*endpoint* si sarebbe potuto utilizzare *TGetDTO.GetOneRoute*

Figura 3.8: Esempio di una classe *Data Model* che implementa selettivamente le interfacce per eseguire le operazioni attraverso l'API web. Si possono inoltre notare gli attributi del *namespace* "System.Text.Json.Serialization" che permettono di personalizzare le fasi di codifica e decodifica JSON.

```
using System.Text.Json.Serialization;

namespace SalusCarichiShared.Models {
    public class LoadingPlan {

        public const string APIBaseRoute = "loadingPlans";

        public class GetterOneDTO : APIGetterOneDTO<LoadingPlan> {

            public string GetOneRoute => APIBaseRoute;

            string APIIdentifiableDTO.id => Id;

            [JsonPropertyName("id")]
            public string Id { get; }

            public GetterOneDTO(string id) ...
        }

        public class GetterAllDTO : APIGetterAllDTO<LoadingPlan> {

            public string GetAllRoute => APIBaseRoute;
        }

        public class GetDTO : APIGetDTO<LoadingPlan> {

            string APIIdentifiableDTO.id => Id;


            [JsonPropertyName("id")]
            public string Id { get; }

            [JsonPropertyName("createdAt")]
            public DateTime CreatedAt { get; }

            [JsonPropertyName("lastModifiedAt")]
            public DateTime LastModifiedAt { get; }

            [JsonPropertyName("deliveryDate")]
            public DateTime DeliveryDate { get; }

            public GetDTO(string id, DateTime createdAt, DateTime lastModifiedAt, DateTime deliveryDate) ...
        }
            public class PostDTO : APIPostDTO<LoadingPlan> {

                public string PostRoute => APIBaseRoute;

                [JsonPropertyName("deliveryDate")]
                public DateTime DeliveryDate { get; }

                public PostDTO(DateTime deliveryDate) {
                    DeliveryDate = deliveryDate;
                }
            }

            public class DeleteDTO : TrackedAPIDeleteDTO<LoadingPlan> {
                string APIDeleteDTO<LoadingPlan>.DeleteRoute => APIBaseRoute;
                string APIIdentifiableDTO.id => Id;
                DateTime TrackedAPIDeleteDTO<LoadingPlan>.lastModifiedAt => LastModifiedAt;


                [JsonPropertyName("id")]
                public string Id { get; }

                [JsonPropertyName("lastModifiedAt")]
                public DateTime LastModifiedAt { get; }

                public DeleteDTO(string id, DateTime lastModifiedAt) ...
            }
        }
    }
```

Per farlo sono state utilizzate le librerie di .NET *System.Net.Http* e *System.Text.Json* che si sono rivelate semplici e abbastanza intuitive da utilizzare. Inoltre C# offrendo il costrutto *async-await* elimina quasi totalmente il problema del rapporto tra funzioni asincrone e *UI/main thread*, rendendo il codice veramente semplice e leggibile.

3.6.1 Considerazioni

Quando è possibile progettare un sistema disaccoppiando la logica applicativa dall'interfaccia (ad esempio attraverso un'API *rest*) diventa veramente semplice andare a costruire nuovi *frontend*.

Inoltre il fatto di utilizzare un linguaggio tanto largamente utilizzato quanto C# rende più falice trovare strumenti che permettano di velocizzare o automatizzare la scrittura del codice per interfacciarsi alle API, come ad esempio **Swagger Codegen**, anche se in questo caso specifico non è stato utilizzato.

3.7 Struttura della UI e modalità di navigazione tra le pagine

Per quanto riguarda l'interfaccia grafica si è cercato di rimanere vicini all'applicazione già esistente in modo da non cambiare eccessivamente l'esperienza dell'utente.

La documentazione di .NET MAUI è ben fatta e guida abbastanza bene nell'impostazione iniziale dell'applicazione.

La classe *Shell* fornisce diverse funzionalità che normalmente si vogliono all'interno di una applicazione, come ad esempio la suddivisione in *tabs* e la navigazione tra pagine.

La navigazione tra pagine fornita dalla classe *Shell* è anche migliorata rispetto a quella di Xamarin.Forms, infatti prima era possibile passare parametri di navigazione solo sotto forma di stringa (cosa abbastanza limitante) mentre ora si possono anche passare degli oggetti. Continua ad esistere invece il problema che rendeva molto scomodo intercettare gli eventi di navigazione, infatti per farlo è necessario effettuare l'*overriding* di un metodo nella classe *Shell* e poi gestire uno ad uno i casi in base alla sorgente e alla destinazione della navigazione. Per questo si è deciso di implementare un sistema di navigazione personalizzato in modo da permettere ad ogni pagina di gestire i propri eventi di navigazione.

3.8 Creazione delle pagine e implementazione delle funzionalità

Una volta buttate le fondamenta che si sono appena descritte, la costruzione delle pagine diventa abbastanza semplice e diretta.

Pagina per pagina quindi si è proseguito a integrare il codice per interfacciarsi con l'API web, a creare l'interfaccia grafica e ad implementare la parte funzionale.

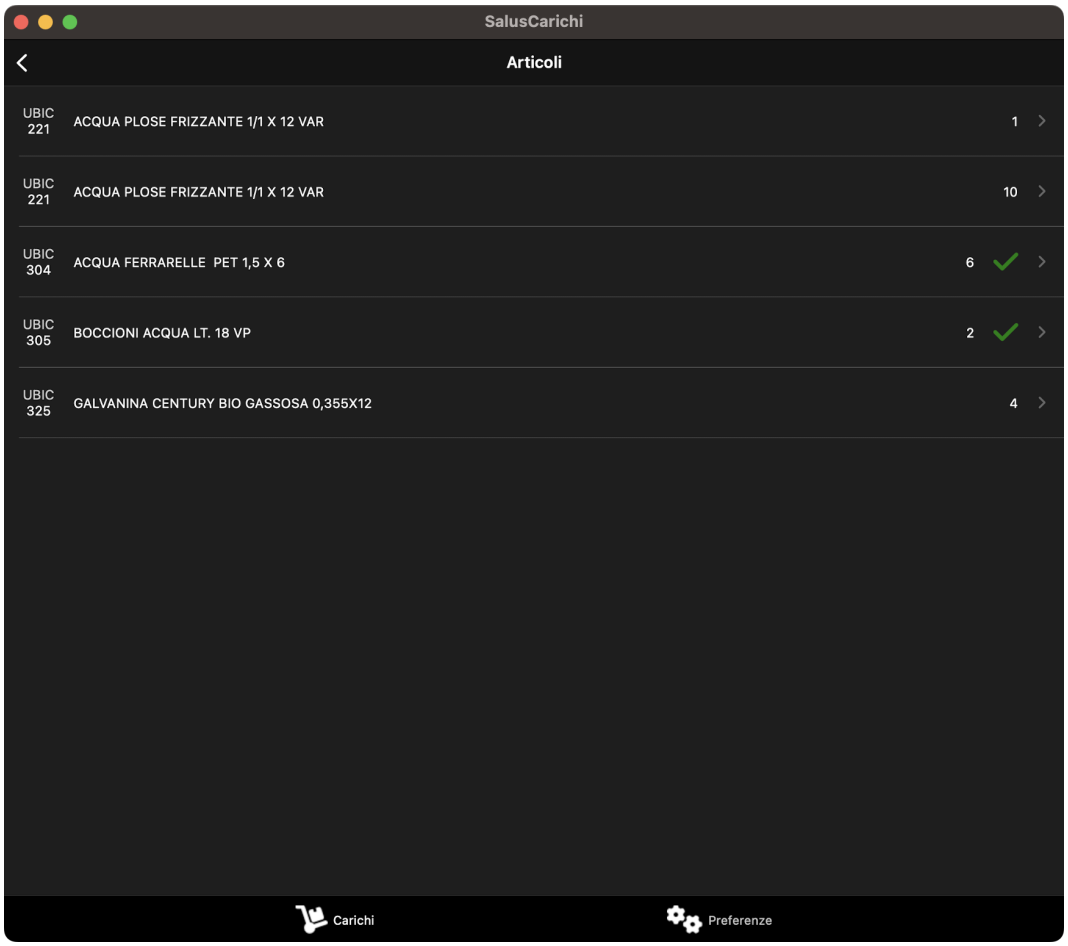
Qui sotto è mostrato come esempio il codice XAML che definisce la pagina incaricata di mostrare gli articoli di un ordine (è possibile vedere il risultato in figura 3.9)

```
<?xml version="1.0" encoding="utf-8" ?>
<shared_views:BaseContentPage
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    xmlns:models="clr-namespace:SalusCarichiShared.Models"
    xmlns:view_models="clr-namespace:SalusCarichiShared.ViewModels"
    xmlns:shared_views="clr-namespace:SalusCarichiShared.Views"
    xmlns:icons="clr-namespace:SalusCarichiShared.Resources.Fonts"
    xmlns:custom_comp="clr-namespace:SalusCarichiShared.CustomComponents"
    x:DataType="view_models:QuantifiedStockItemsPageViewModel"
    x:Class="SalusCarichiShared.Views.QuantifiedStockItemsPage"
    Title="Articoli">

    <shared_views:BaseContentPage.PageContent>

        <ListView
            x:Name="listView"
            ItemsSource="{Binding QuantifiedStockItems}"
            SelectedItem="{Binding SelectedQuantifiedStockItem}"
            IsPullToRefreshEnabled="True"
            IsRefreshing="{Binding IsRefreshing}"
            RefreshCommand="{Binding RefreshQuantifiedStockItemsCommand}"
            HasUnevenRows="True">

            <ListView.Behaviors>
                <!-- The x:Name etc.. fixes a bug where
                the toolkit is not included by the linker
                (https://stackoverflow.com/a/76452761) -->
                <toolkit:EventToCommandBehavior
                x:Name="EventToCommandBehavior"
                EventName="ItemSelected"
                Command="{Binding QuantifiedStockItemSelectedCommand}"/>
            </ListView.Behaviors>
```

```
<ListView.ItemTemplate>
    <DataTemplate
        x:DataType="{Type models:QuantifiedStockItem+GetDTO}">
        <ViewCell>
            <Grid
                ColumnDefinitions="Auto, *, Auto, Auto, Auto">

                <VerticalStackLayout
                    Grid.Column="0"
                    HorizontalOptions="Center"
                    Padding="20">

                    <Label
                        Text="UBIC"
                        TextColor="LightGray"
                        Padding="0"/>

                    <Label
                        Text="{Binding StockItem.Ubic}"
                        Padding="0"
                        HorizontalTextAlignment="Center"/>

                </VerticalStackLayout>

                <Label
                    Grid.Column="1"
                    Text="{Binding StockItem.Name}"
                    VerticalOptions="Center"/>

                <Label
                    Grid.Column="2"
                    Text="{Binding Quantity}"
                    VerticalOptions="Center"/>

                <Label
                    Grid.Column="3"
                    Style="{StaticResource IconLabel}"
                    Text="{Static icons:FASolidIcon.Check}"
                    VerticalOptions="Center"
                    IsVisible="{Binding Completed}"
                    TextColor="Green"
                    Margin="20, 20, 0, 20"/>

                <custom_comp:NavigationChevron
                    Grid.Column="4"/>

            </Grid>
        </ViewCell>
    </DataTemplate>
```

```
            </ListView.ItemTemplate>
        </ListView>
    </shared_views:BaseContentPage.PageContent>

</shared_views:BaseContentPage>
```

Figura 3.9: Pagina degli aricoli di un ordine



3.9 Ambiente di sviluppo

L'applicazione è stata realizzata interamente con Visual Studio for Mac.

L'IDE mette a disposizione veramente tanti strumenti, specifici per lo sviluppo multipiattaforma, che permettono di velocizzare di molto lo sviluppo, alcuni esempi sono:

- Gestore di emulatori Android

- Integrazione con Xcode per utilizzare gli emulatori iOS
- Strumenti automatici per la firma degli eseguibili e la gestione dei certificati di sviluppo

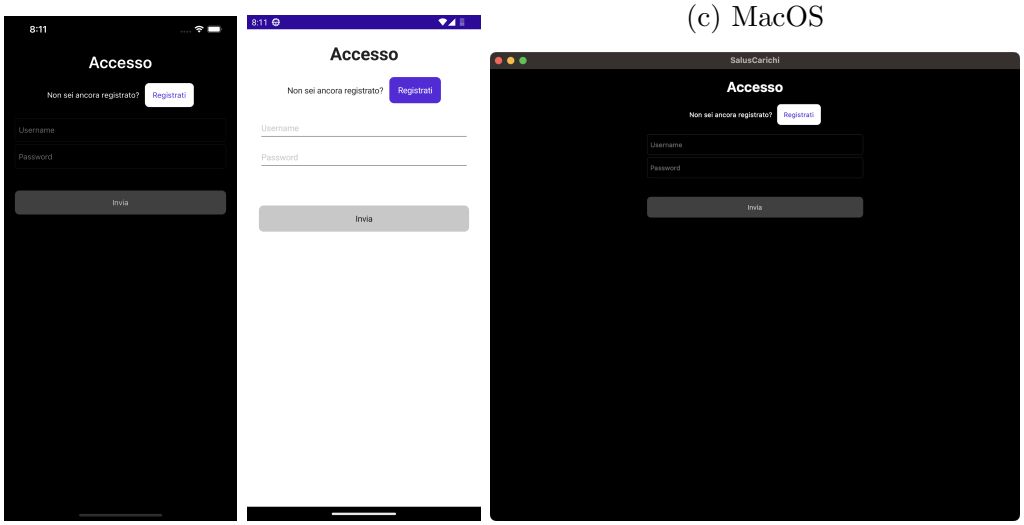
Questo IDE ha però mostrato anche diversi bug, alcuni risolvibili con un semplice riavvio, altri cambiando delle impostazioni (anche se poi potevano essere tranquillamente riportate al default senza che il bug si verificasse nuovamente). Alcuni bug impedivano la compilazione anche in caso di codice corretto, altri rendevano difficile effettuare il debug dell'applicazione in quanto alcune righe venivano saltate inaspettatamente.

Va anche aggiunto che da non molti giorni è stato annunciato che Microsoft smetterà di manutenere Visual Studio for Mac, al suo posto gli sviluppatori potranno spostarsi su Visual Studio Code utilizzando le estensioni per lo sviluppo .NET e .NET MAUI.

3.10 Immagini dell'applicazione prodotta

Figura 3.10: Pagina di accesso

(a) iOS (b) Android (c) MacOS



Nella pagina di accesso (figura 3.10) l'utente può effettuare il login (se già registrato) o andare alla pagina di registrazione. In questa vista si può notare il bottone "invia" disattivato in quanto nel *viewmodel* sono implementate delle *regular expressions* che permettono di controllare la validità dei dati in input.

Figura 3.11: Pagina di registrazione

(a) iOS (b) Android (c) MacOS

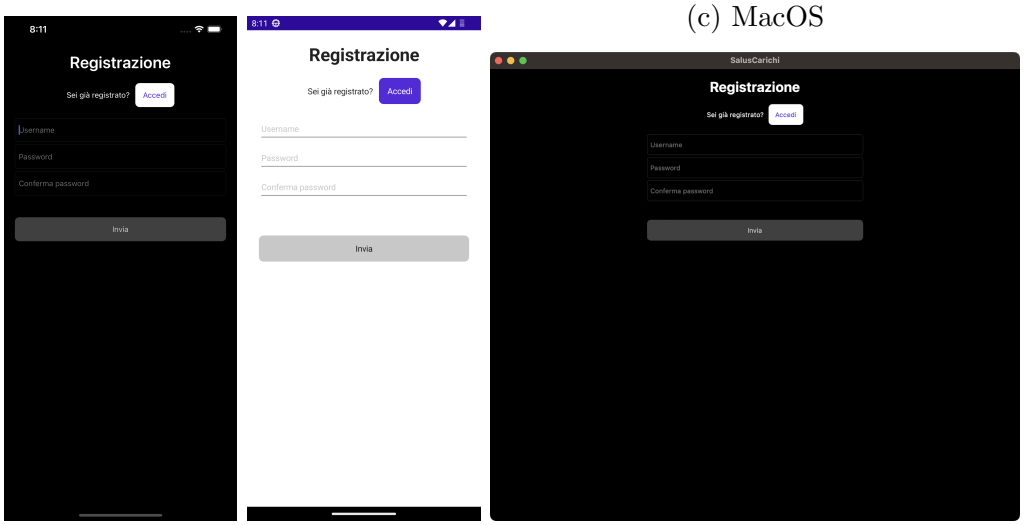
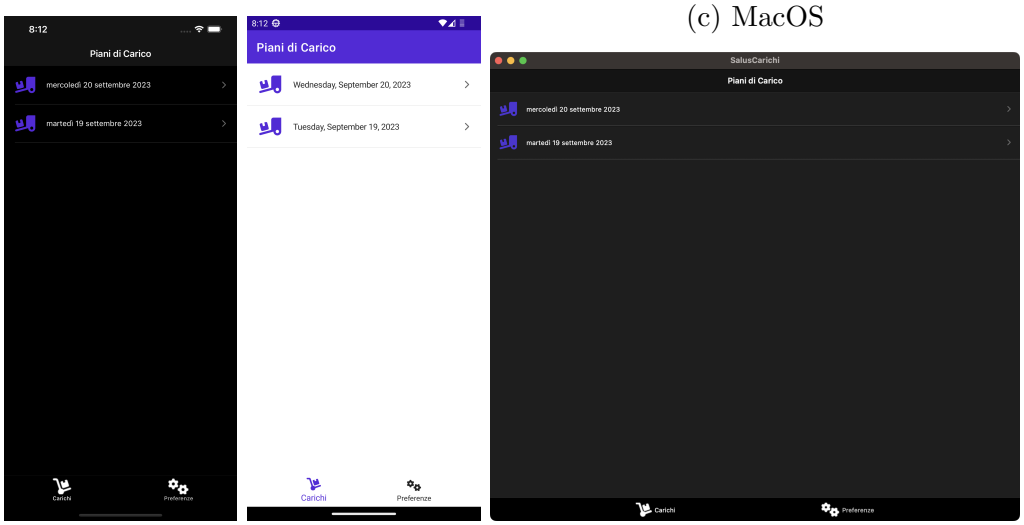


Figura 3.12: Pagina dei piani di carico

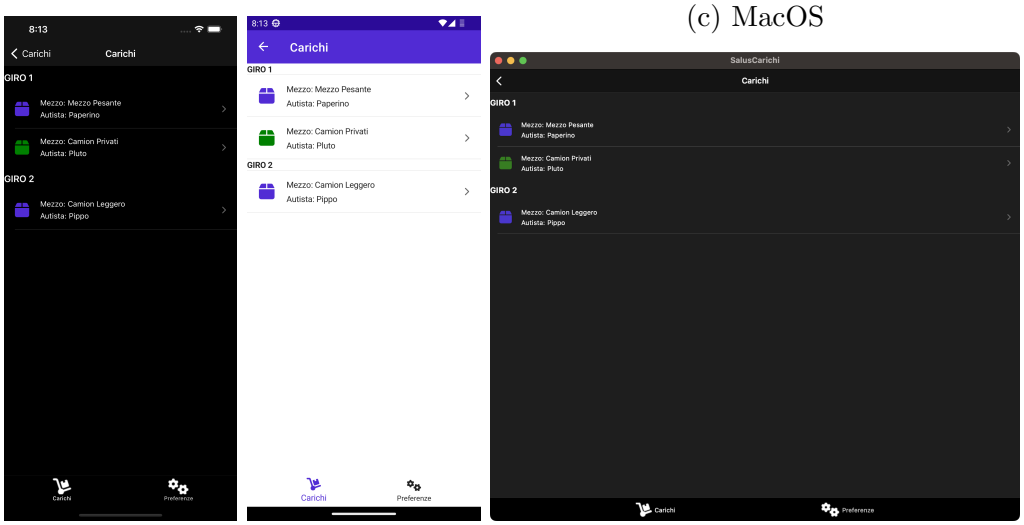
(a) iOS (b) Android (c) MacOS



Nella pagina dei piani di carico (figura 3.12) l'utente può scegliere il piano di carico da seguire.

Figura 3.13: Pagina dei carichi

(a) iOS (b) Android (c) MacOS



Nella pagina dei carichi (figura 3.13) l'utente (il magazziniere) sceglie quale carico andare a realizzare ovvero su quale mezzo andare a porre gli articoli.

Figura 3.14: Pagina degli ordini

(a) iOS (b) Android (c) MacOS

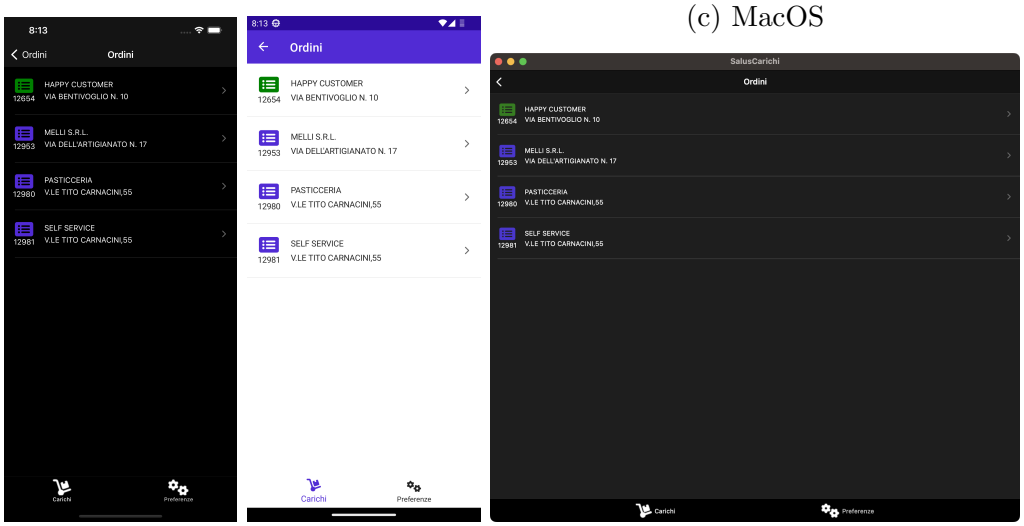


Figura 3.15: Pagina degli articoli

(a) iOS (b) Android (c) MacOS

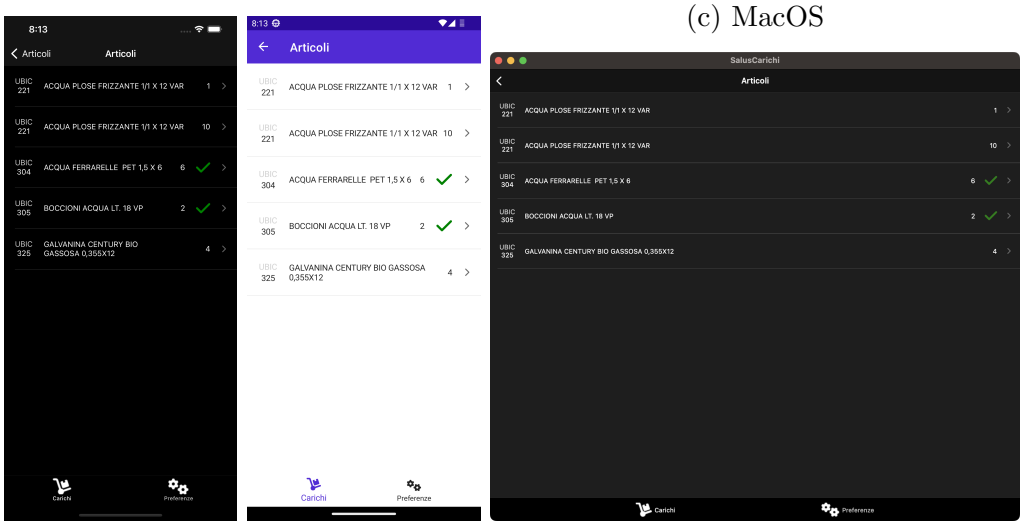
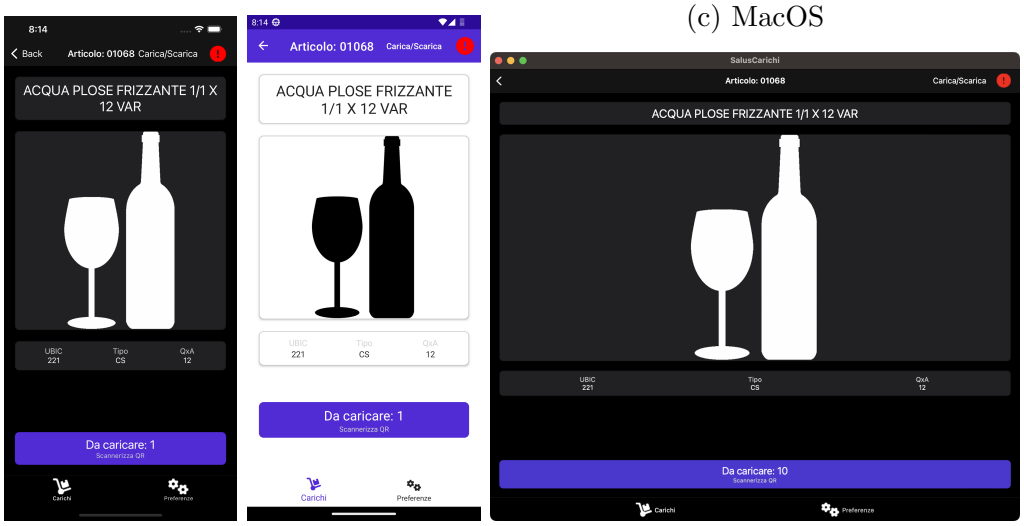


Figura 3.16: Pagina di un articolo

(a) iOS (b) Android (c) MacOS



Nella pagina dell'articolo (figura 3.16) è possibile attivare la fotocamera e se viene inquadrato il codice QR dell'articolo corretto questo viene segnato come caricato. Sono anche disponibili i bottoni sebbene ancora non funzionanti per andare a segnalare un problema o modificare l'immagine dell'articolo.

Figura 3.17: Pagina delle impostazioni

(a) iOS (b) Android (c) MacOS

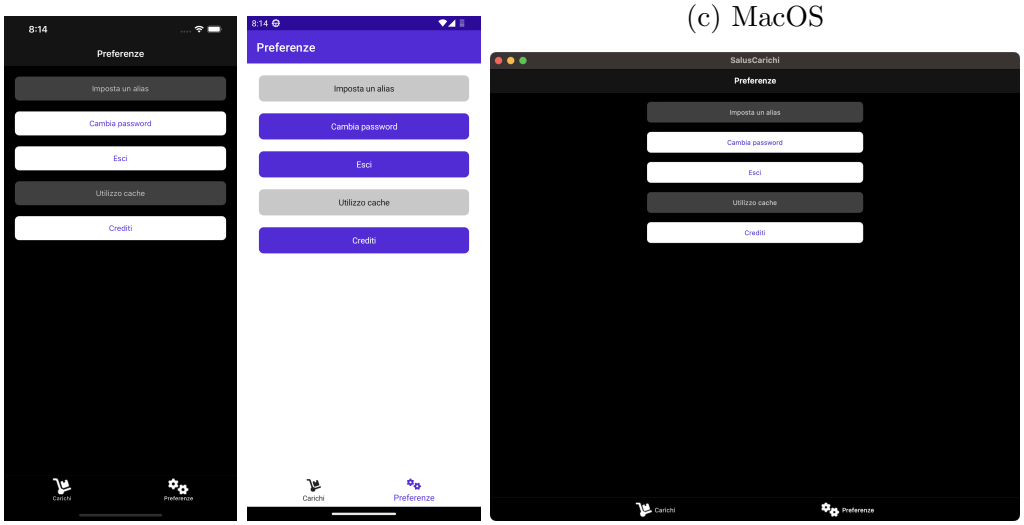
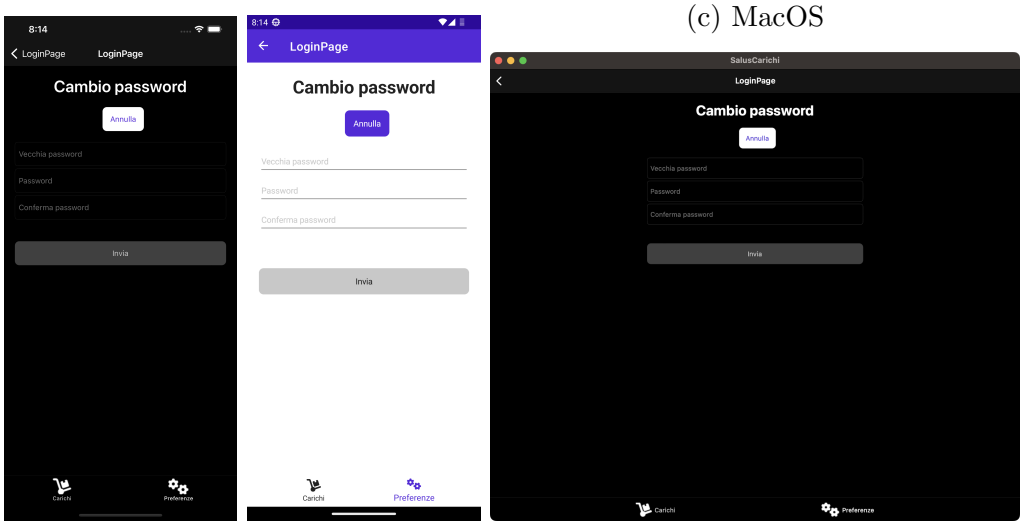


Figura 3.18: Pagina di cambio password

(a) iOS (b) Android (c) MacOS



Capitolo 4

Analisi e valutazione di .NET MAUI

4.1 Valutazione delle caratteristiche chiave per lo sviluppo multipiattaforma

In questa sezione si andrà a valutare .NET MAUI considerando i punti descritti nella sezione 1.5.

4.1.1 Supporto a più piattaforme

La varietà di piattaforme supportate è sicuramente un punto di forza di .NET MAUI.

Il fatto di poter produrre in output applicazioni *native* per 5 piattaforme diverse e contando anche che viene coperto sia il panorama *mobile* che quello *desktop* è una caratteristica quasi senza rivali.

4.1.2 Prestazioni

Le prestazioni sono molto buone, se messe a confronto con quelle dell'applicazione iOS originale l'unica differenza si nota sul tempo di avvio dell'applicazione. Va anche però notato che l'applicazione non è particolarmente pensate di suo e quindi un risultato simile era facilmente prevedibile e probabilmente non particolarmente significativo.

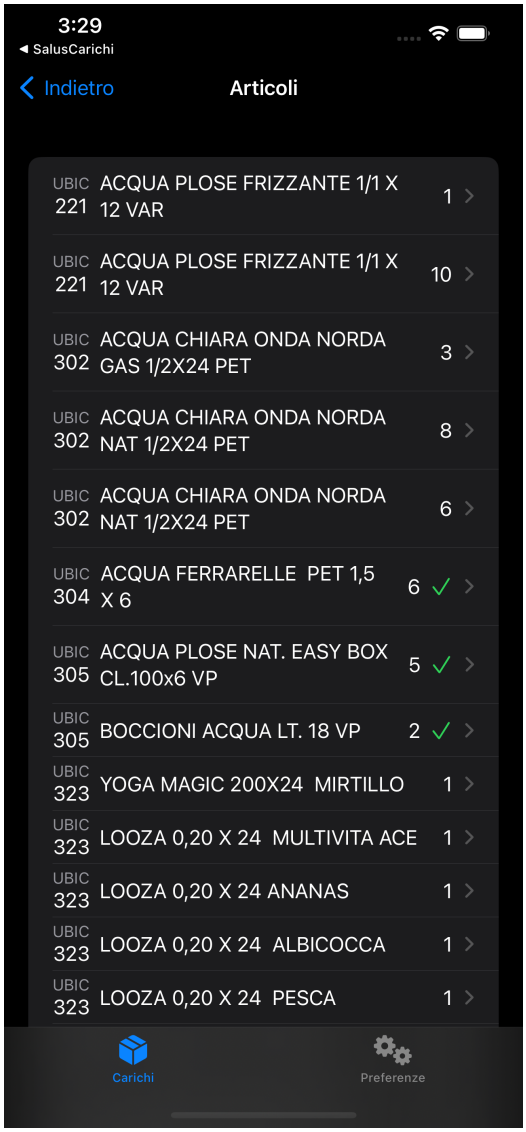
4.1.3 Interfaccia grafica

.NET MAUI utilizza i componenti nativi delle piattaforme per renderizzare a schermo i controlli, però sembra puntare ad uniformare il più possibile il

risultato visivo sulle diverse piattaforme ed incoraggia lo sviluppatore verso questa direzione.

Figura 4.1: Differenza tra l'aspetto di default di una lista su iOS rispetto a quello di default di un'app generata con .NET MAUI (notare il contorno grigio e il la spaziatura esterna)

(a) Default iOS



(b) Default .NET MAUI

Questo in base alla prospettiva può essere visto come un punto di forza o come una debolezza del framework.

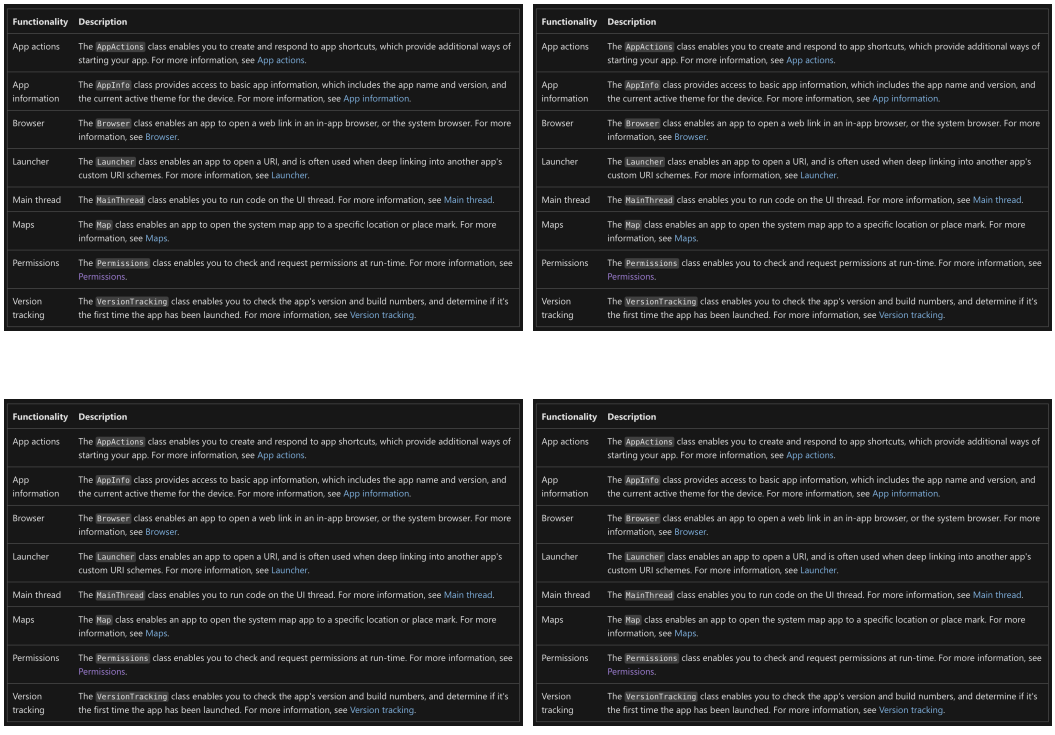
Un risultato uniforme sulle piattaforme è gradevole in quanto ha senso che la stessa applicazione abbia lo stesso aspetto per ogni utente e inoltre permette allo sviluppatore di lavorare all'interfaccia senza preoccuparsi di eventuali differenze *platform-specific*.

L'altra faccia della stessa medaglia però consiste nel fatto che l'utente non avrà il *look and feel* della piattaforma alla quale è abituato. In verità grazie alla possibilità di scrivere codice *platform-specific* lo sviluppatore potrebbe decidere di risolvere questo problema, andrebbe però chiaramente a diminuire la percentuale di riutilizzo del codice e quindi i costi di mantenimento della *codebase*. Riguardo a questo punto è possibile trovare ulteriori considerazioni più avanti nel testo (sezioni 4.5.1 e 4.5.2).

4.1.4 Accesso alle funzionalità del dispositivo

.NET MAUI non solo permettere allo sviluppatore di accedere a tantissime funzionalità di sistema ma inoltre semplifica parecchio le API nascondendo egregiamente le differenze tra le piattaforme.

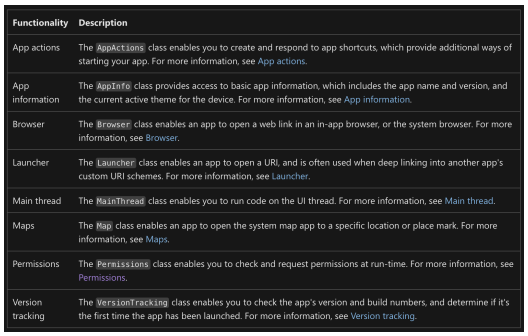
Figura 4.2: Lista API messe a disposizione da .NET MAUI per utilizzare le funzionalità di sistema.
`https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/`



Functionality	Description
App actions	The `AppActions` class enables you to create and respond to app shortcuts, which provide additional ways of starting your app. For more information, see App actions.
App information	The `AppInfo` class provides access to basic app information, which includes the app name and version, and the current active theme for the device. For more information, see App information.
Browser	The `Browser` class enables an app to open a web link in an in-app browser, or the system browser. For more information, see Browser.
Launcher	The `Launcher` class enables an app to open a URI, and is often used when deep linking into another app's custom URI schemes. For more information, see Launcher.
Main thread	The `MainThread` class enables you to run code on the UI thread. For more information, see Main thread.
Maps	The `Map` class enables an app to open the system map app to a specific location or place mark. For more information, see Maps.
Permissions	The `Permissions` class enables you to check and request permissions at run-time. For more information, see Permissions.
Version tracking	The `VersionTracking` class enables you to check the app's version and build numbers, and determine if it's the first time the app has been launched. For more information, see Version tracking.

Functionality	Description
App actions	The `AppActions` class enables you to create and respond to app shortcuts, which provide additional ways of starting your app. For more information, see App actions.
App information	The `AppInfo` class provides access to basic app information, which includes the app name and version, and the current active theme for the device. For more information, see App information.
Browser	The `Browser` class enables an app to open a web link in an in-app browser, or the system browser. For more information, see Browser.
Launcher	The `Launcher` class enables an app to open a URI, and is often used when deep linking into another app's custom URI schemes. For more information, see Launcher.
Main thread	The `MainThread` class enables you to run code on the UI thread. For more information, see Main thread.
Maps	The `Map` class enables an app to open the system map app to a specific location or place mark. For more information, see Maps.
Permissions	The `Permissions` class enables you to check and request permissions at run-time. For more information, see Permissions.
Version tracking	The `VersionTracking` class enables you to check the app's version and build numbers, and determine if it's the first time the app has been launched. For more information, see Version tracking.

Functionality	Description
App actions	The `AppActions` class enables you to create and respond to app shortcuts, which provide additional ways of starting your app. For more information, see App actions.
App information	The `AppInfo` class provides access to basic app information, which includes the app name and version, and the current active theme for the device. For more information, see App information.
Browser	The `Browser` class enables an app to open a web link in an in-app browser, or the system browser. For more information, see Browser.
Launcher	The `Launcher` class enables an app to open a URI, and is often used when deep linking into another app's custom URI schemes. For more information, see Launcher.
Main thread	The `MainThread` class enables you to run code on the UI thread. For more information, see Main thread.
Maps	The `Map` class enables an app to open the system map app to a specific location or place mark. For more information, see Maps.
Permissions	The `Permissions` class enables you to check and request permissions at run-time. For more information, see Permissions.
Version tracking	The `VersionTracking` class enables you to check the app's version and build numbers, and determine if it's the first time the app has been launched. For more information, see Version tracking.



Functionality	Description
App actions	The `AppActions` class enables you to create and respond to app shortcuts, which provide additional ways of starting your app. For more information, see App actions.
App information	The `AppInfo` class provides access to basic app information, which includes the app name and version, and the current active theme for the device. For more information, see App information.
Browser	The `Browser` class enables an app to open a web link in an in-app browser, or the system browser. For more information, see Browser.
Launcher	The `Launcher` class enables an app to open a URI, and is often used when deep linking into another app's custom URI schemes. For more information, see Launcher.
Main thread	The `MainThread` class enables you to run code on the UI thread. For more information, see Main thread.
Maps	The `Map` class enables an app to open the system map app to a specific location or place mark. For more information, see Maps.
Permissions	The `Permissions` class enables you to check and request permissions at run-time. For more information, see Permissions.
Version tracking	The `VersionTracking` class enables you to check the app's version and build numbers, and determine if it's the first time the app has been launched. For more information, see Version tracking.

Durante la realizzazione del progetto ad esempio sono state utilizzate queste API per:

- Richiedere e valutare i permessi di sistema riguardanti la fotocamera
- Utilizzare della vibrazione del dispositivo
- Salvare preferenze o token

In ognuno di questi casi sono bastate pochissime righe e l'esperienza di sviluppo è stata ottima.

4.1.5 Percentuale di riutilizzo del codice

Il modo in cui .NET MAUI si sviluppa attorno ad un singolo *project* è ottimo per favorire il riutilizzo di codice. Durante l'intero sviluppo del progetto non è stato necessario scrivere una sola linea di codice nei framework *native*. Inoltre la possibilità di inserire una risorsa ed affidare al framework il compito di elaborarla nel modo necessario per ogni piattaforma risulta davvero semplice e veloce.

Bisogna però tenere conto del fatto che questo può variare da applicazione ad applicazione. In particolare il fatto che .NET MAUI metta a disposizione delle API multipiattaforma rende inevitabile che le funzionalità disponibili su una piattaforma ma non su altre dovranno essere sfruttate attraverso il suo framework *platform-specific*. Un esempio consiste nella geolocalizzazione in background e ad applicazione chiusa che su Android e iOS funziona in maniera decisamente diversa e quindi può essere solo implementata direttamente.

4.1.6 Strumenti di assistenza allo sviluppo

Di questo si è già parlato nella sezione 3.9. È necessario però ricodare che le valutazioni sono state effettuate solo su Visual Studio for Mac e andrebbero intergrate con un po' di sperimentazione in Visual Studio e Visual Studio Code.

Inoltre i problemi principali risontrati sono dovuti a *bug* e non a mancanze funzionali e quindi decisamente meno gravi.

Fatte queste considerazioni il livello degli strumenti di assistenza allo sviluppo è molto buono.

4.2 Disponibilità di materiale online

Uno dei più grandi punti di forza di .NET MAUI è che seppur in produzione da poco più di un anno il quantitativo di risorse (oltre alla documentazione) online è enorme, sia per quanto riguarda la fase di apprendimento del framework che per quanto riguarda il *troubleshooting* di eventuali problemi.

Inoltre è possibile sfruttare il *package manager* **NuGet** per aggiungere delle librerie alla *solution*, ad esempio durante la realizzazione del progetto si è utilizzata una libreria *open source* per l'utilizzo della fotocamera e il riconoscimento dei codici QR. Il fatto che C# sia un linguaggio largamente utilizzato permette di trovare spesso pacchetti in grado di velocizzare lo sviluppo del proprio prodotto in maniera drastica. Un esempio è quello di MAUI community toolkit (vedi la sezione 4.2.1)

4.2.1 MAUI community toolkit

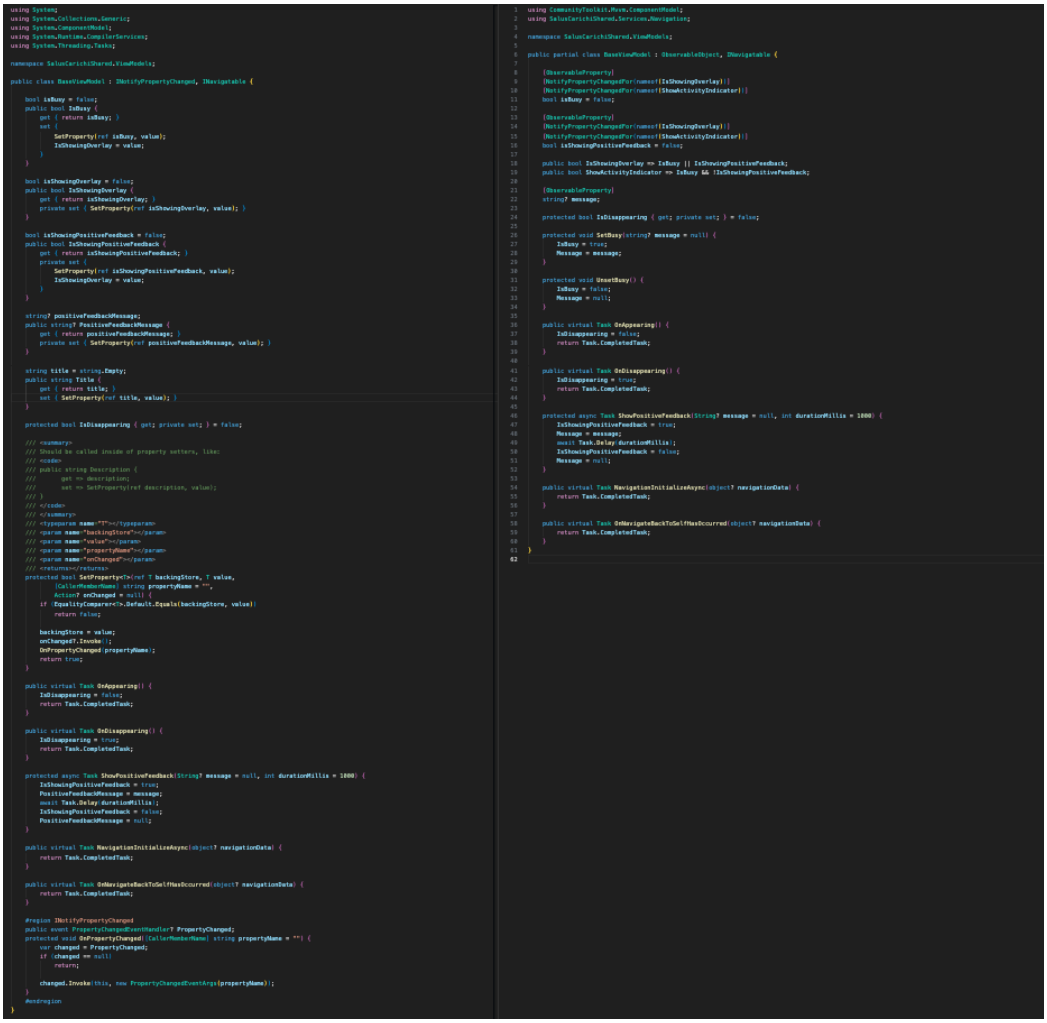
Il **community toolkit** è una raccolta di librerie per .NET sviluppato in gran parte dalla comunità *open soruce*. In partiolare i pacchetti **CommunityToolkit.Maui** e **CommunityToolkit.Mvvm** sono incredibilmente utili per velocizzare lo sviluppo con questo framework.

Il primo pacchetto offre diverse funzionalità che sono comunemente utilizzate come ad esempio *popup*, animazioni, *media player*, controlli per il *layout* e tante altre.

Il secondo pacchetto è sostanzialmente fondamentale in quanto rende l'implementazione del pattern MVVM decisamente più semplice e concisa grazie all'utilizzo di attributi che, una volta applicati ai *view model*, permettono di generare automaticamente del codice (nascosto) che normalmente sarebbe molto ripetitivo. Qui sotto nella figura 4.5 è viene mostrata la differenza tra una classe prima e dopo l'utilizzo di **CommunityToolkit.Mvvm**.[1]

[1]Si tenga inoltre conto che la classe in questione è un *view model* base ed è quindi pensato non per fornire funzionalità specifiche ma bensì solo funzionalità che potranno essere ereditate da altri *view model* nei quali la differenza sarebbe quindi ancora più marcata.

Figura 4.5: La stessa classe senza l'utilizzo di **CommunityToolkit.Mvvm** (sinistra) e con l'utilizzo della libreria (destra).



Il codice generato automaticamente è comodo non solo per il fatto che velocizza l'implementazione ma anche perché riduce la probabilità di errori dovuti a frequenti copia ed incolla che sarebbero la norma in assenza della libreria.

4.3 Grande possibilità di personalizzazione

Si nota subito che, al prezzo di una discreta complessità e verbosità, .NET MAUI offre un'enorme possibilità di personalizzazione. Il fatto di **poter** accedere a tutte le funzionalità delle singole piattaforme è fondamentale in quanto

fornisce allo sviluppatore gli strumenti per **poter** fare quello che desidera in modo che sia poi lui stesso a decidere se privilegiare il riutilizzo di codice o la qualità e specificità del risultato finale.

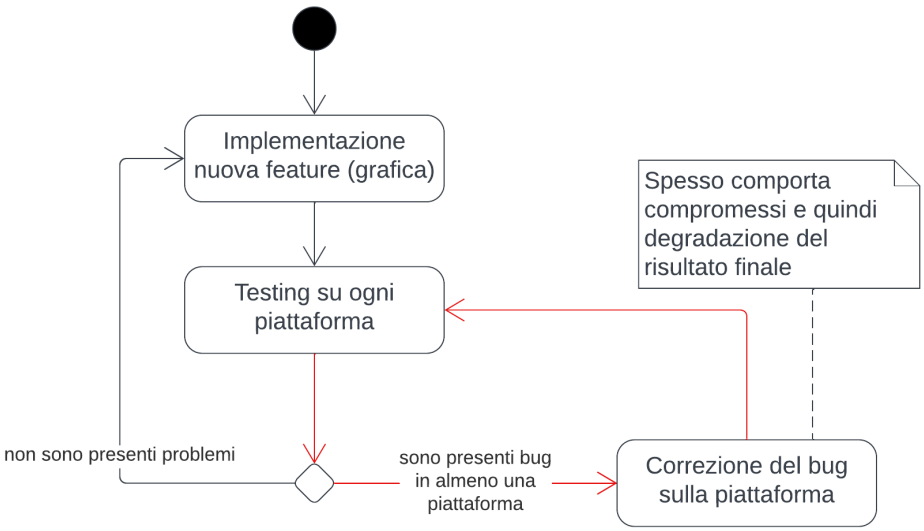
4.4 L'elefante nella stanza

.NET MAUI attualmente soffre di un enorme problema, l'incredibile mole di *bug*. [2]

È risaputo che software senza *bug* non esistono, ed è vero che questo problema è decisamente più risolvibile rispetto a carenze architetturali o di progettazione, ma in questo caso la situazione è talmente estrema da mettere in discussione il vantaggio prinipale che si ottiene dallo sviluppo multipiattaforma, la riduzione del tempo di sviluppo (e quindi dei costi).

Questa situazione si viene a creare in quanto lo sviluppatore si trova a seguire un'iterazione come quella evidenziata in rosso nella figura 4.6:

Figura 4.6: Processo di implementazione (parte grafica) delle feature. Evidenziato in rosso il ciclo dovuto alla correzione di *bug*



Tralasciando la possibilità di andare a risolvere i *bug* utilizzando codice *platform-specific* che non è scalabile dato il numero imponente di *bug*. Lo svi-

[2] Si parla di *bug* grafici o di comportamenti anomali dei componenti grafici, ma questa non è un'attenuante, il framework dopotutto si chiama .NET MA**UI** e *UI* sta per *User Interface*.

luppatore è costretto a trovare una nuova soluzione, e probabilmente anche a scendere a compromessi rispetto alla sua idea originale. Inoltre (dato che *XAML Hot Reload* consente di vedere in tempo reale le modifiche sull'interfaccia solo per un dispositivo alla volta) ad ogni cambiamento è necessario eseguire un numero di compilazioni del progetto pari al numero di piattaforme che si intende supportare e questo richiede parecchio tempo (per non parlare del fatto che si dovrebbe anche cambiare dispositivo in quanto per eseguire l'app Windows bisogna lavorare su Windows e per le app iOS/MacOS su MacOS).

I *bug* in questione sono parecchio evidenti all'utente, e questo li rende inaccettabili. Qui sotto alcuni esempi riscontrati durante la produzione del progetto.

Figura 4.7: Effettuando cambio di tema chiaro/scuro, gli elementi delle liste diventano trasparenti.

(a) Stato iniziale (b) Primo cambio tema (c) Secondo cambio tema

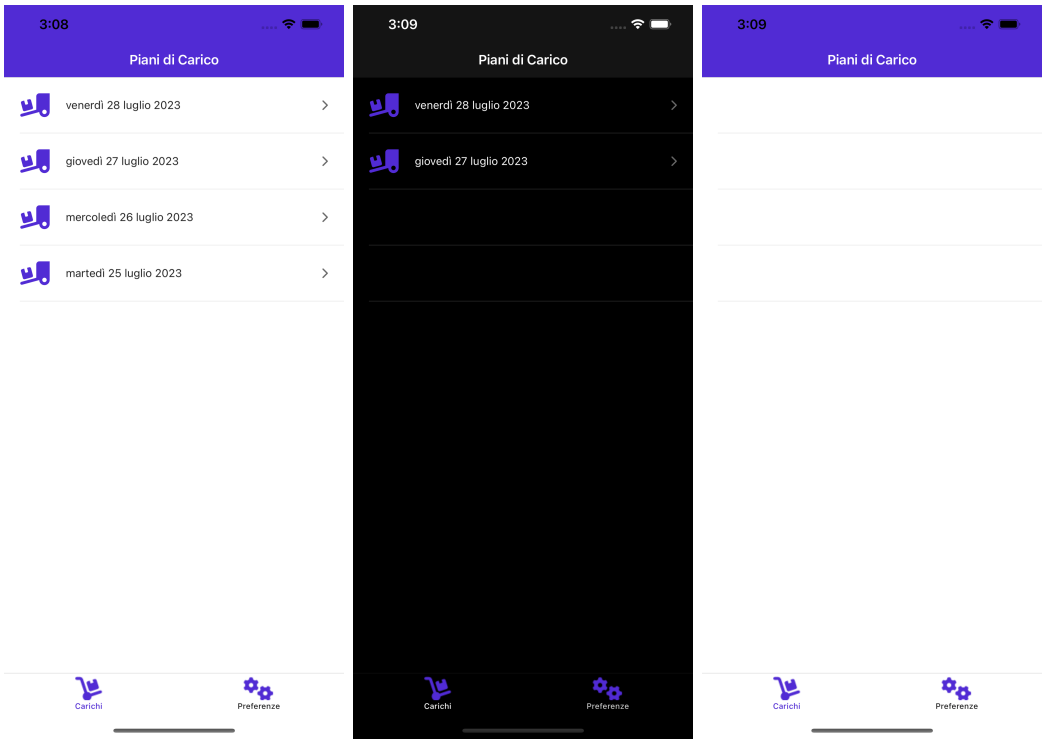
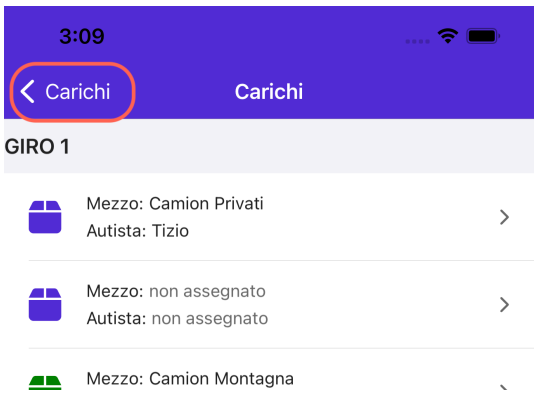


Figura 4.8: Il testo del bottone per navigare all'indietro coincide col titolo della pagina stessa e non con quello di quella precedente



Esempi di alcuni *bug* importanti che si sono incontrati:

- Su MacOS il gesto di trascinamento verso il basso per aggiornare una lista non viene riconosciuto. (Per risolvere questo problema lo sviluppatore dovrebbe implementare un metodo *custom* per l'aggiornamento)

- Il colore delle icone di tipo font, seppur impostato correttamente, non si comporta come previsto cambiando il tema chiaro/scuro.

- Durante la presentazione di tipo *modal* di una pagina gli avvisi non vengono mostrati a schermo.

- Su iOS le icone impostate come *FontImageSource* vengono renderizzate sfocate.

Parecchi di questi *bug* sono inolte noti alla community e descritti in diversi *issue* su GitHub, ma spesso vengono chiusi per inattività o in seguito alla scoperta di *workaround* e non di veri e propri *fix*. (Riguardo a questo sembra esserci anche un po' di malcontento riguardo alla gestione di .NET MAUI da parte di Microsoft.)

> *"MAUI is a powerful tool for building great apps and user interfaces, but the project is currently being held back by its maintainers. The documentation is lacking and the build toolset is immature, with a lot of bugs that the maintainers are not effectively addressing. Additionally, the maintainers are passive and not providing enough support to the community. These issues are causing frustration for users and making it difficult for the project to continue.*
> *One of the most frustrating issues is the way the maintainers handle issues that are reported to them. Instead of addressing the issue directly, they often "ping-pong" it between the MAUI and Xamarin repositories, leaving users confused and unsure of where to go for support. This is not only unproductive but also demotivating for users.*
> *Furthermore, when users report an issue and provide a workaround, the maintainers often close the issue without actually fixing the underlying problem. This is not a sustainable approach and it does not address the root cause of the problem. Users should not be expected to constantly come up with workarounds for issues that should be fixed by the maintainers."* [1]

Ricapitolando:

Questo genere di *bug* non è accettabile in un prodotto e inoltre causa grandi rallentamenti nello sviluppo. Il problema è assolutamente risolvibile da parte di Microsoft o della comunità *open source* ma probabilmente richiederà tempo.

4.5 Possibili problemi a lungo termine

In questa sezione si vuole analizzare quei problemi che potrebbero essere radicati nelle fondamenta di .NET MAUI e quindi potenzialmente non risolvibili nel breve periodo.

4.5.1 Si può veramente utilizzare la stessa interfaccia per tutte le piattaforme?

Fino a questo punto della tesi si è sempre assunto che avere la stessa interfaccia per ogni piattaforma fosse un vantaggio, ma è veramente così?

Interfacce *mobile* e *desktop*

Il dubbio principale si manifesta non sulla piattaforma in sè ma sulla tipologia di piattaforma, in questo caso *mobile* o *desktop*. Infatti queste due categorie sono fondamentalmente differenti, in particolare per quanto riguarda il metodo di fruizione (mouse e tastiera vs. schermo touch) e la grandezza dello schermo.

Non è un caso che ad esempio nell'ambito dello sviluppo di interfacce web si parli di *progressive enhancement* e *graceful degradation*, queste tecniche rendono evidente il fatto che su *desktop* è possibile avere un'interfaccia più ricca e funzionale, non sfruttarla sarebbe un peccato.

È chiaro che quello presentato non è un problema specifico di .NET MAUI in quanto colpisce in generale framework che offrono supporto per piattaforme *mobile* e *desktop*. Questo comunque implica che uno dei punti di forza di .NET MAUI non è necessariamente così applicabile.

La conclusione che si trae è che sicuramente lo sviluppatore può riutilizzare completamente il codice riguardante la logica applicativa ma, in base al suo caso specifico, dovrebbe almeno considerare l'opzione di sviluppare due interfacce differenti per i dispositivi *mobile* e *desktop*.

4.5.2 .NET MAUI può permettere lo sviluppo di un'applicazione senza troppi compromessi?

Sottolineando che anche in questo caso si parla di compromessi dal punto di vista della *UI*, bisogna capire se il framework è in grado di mappare l'idea dello sviluppatore su tutte le piattaforme in maniera abbastanza fedele.

Caso di studio:

Durante la realizzazione dell'applicazione Salus Carichi si voleva replicare la funzionalità di un bottone così come nell'applicazione iOS originale.

Figura 4.9: Bottone in alto a destra (tre puntini) che permette di modificare l'immagine del prodotto, segnalare un problema o cambiare lo stato del prodotto (opzione che deve essere disponibile solo nella versione di debug).

Il primo passaggio da effettuare è individuare il controllo per inserire un bottone nella barra di navigazione. Dalla documentazione si scopre che una *ContentPage* può ospitare dei bottoni nella barra di navigazione, inserendoli nella collezione di *ToolbarItems*. È però possibile aggiungere solamente dei *ToolbarItem* e non altri tipi di bottoni, questo implica che si debba **rinunciare** ad avere un unico bottone con *dropdown*.

Siccome è ovvio che tutti i bottoni non possono essere affiancati in così poco spazio una soluzione sarebbe di nascondere oltre a disabilitare i bottoni che non possono essere utilizzati (come quello per eliminare un'immagine quando non è presente). Si nota però che i *ToolbarItem*, al contrario di molti altri controlli, non hanno una proprietà *IsVisible* che permetterebbe di nasconderli. Sarebbe possibile aggiungerli e rimuoverli sfruttando il *code behind* della *view* ma questo porterebbe a **rinunciare** alla programmazione dichiarativa dell'interfaccia in XAML (vedi qui sotto).

```
public QuantifiedStockItemPage() {
    ...
#if DEBUG
    ToolbarItems.Add(
        new("Carica/Scarica", null, () => {
            ...
            vm.LoadUnloadCommand.Execute(null);
        })
    );
#endif
```

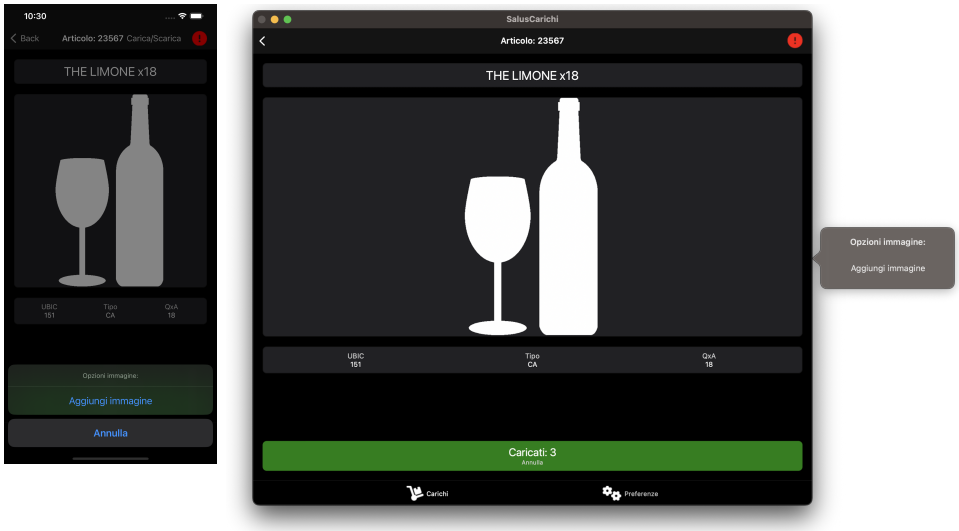
```
    ...
}
```

Diventa quindi necessario **scendere ad un compromesso** e abbandonare l'idea di avere tutti i bottoni nella barra di navigazione decidendo quindi di mostrare i bottoni per l'immagine dopo un click su di essa e gli altri due nella barra di navigazione.

A questo punto si decide di sfruttare un *ActionSheet* per mostrare più opzioni all'utente quando clicca sull'immagine, questo viene renderizzato bene su mobile ma ad esempio su MacOS avviene in una posizione poco gradevole **degradando quindi il prodotto finale** (figura 4.10).

Figura 4.10: Differenza di rendering del controllo *ActionSheet* tra iOS e MacOS

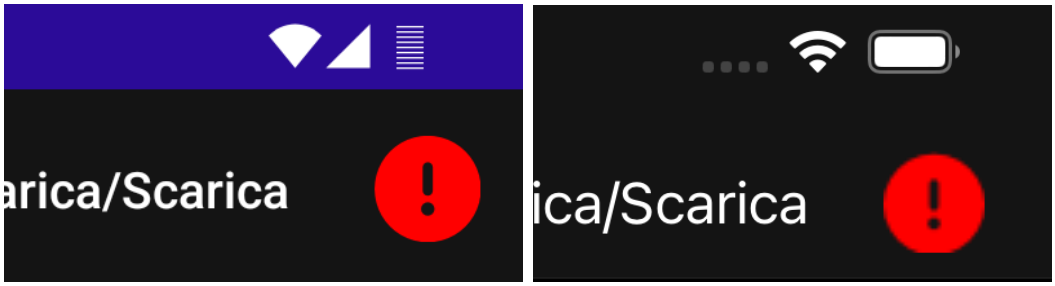
(a) iOS (b) MacOS



Per il bottone di segnalazione di un problema si decide di utilizzare un'icona. Su iOS e MacOS le icone settate attraverso *FontImageSource* (unico modo possibile per settare l'icona di un *ToolbarItem*) risultano sfocate e, in questo caso, anche leggermente tagliate (figura 4.11) Anche qui allora si deve scegliere se **rinunciare** all'icona o accettare che su iOS l'**utente noterà un problema**.

Figura 4.11: Bug di rendering di *FontImageSource*

(a) Android (b) iOS (notare sfocatura e clipping)



Per concludere

Questo è solo un esempio di diversi che si sono incontrati durante la realizzazione del progetto.

Più che un problema da risolvere (in quanto strettamente legato al fatto che .NET MAUI deve unificare 5 piattaforme diverse, compito non falice) questo è un fattore che lo sviluppatore di .NET MAUI deve tenere in considerazione e sapere che dovrà scendere a compromessi sulla qualità del prodotto finale oppure investire tempo in soluzioni *platform-specific*.

Conclusioni

Lo sviluppo di applicazioni multipiattaforma è una grande sfida, per riuscire a generare un prodotto di qualità e allo stesso tempo ridurre i costi è necessario fornire agli sviluppatori strumenti che siano affidabili e flessibili. Quando si parla di affidabilità si intende che il framework dovrebbe riuscire a comportarsi in un modo prevedibile dallo sviluppatore, permettergli di non dover iterare attraverso troppe soluzioni e di non dover scenere a troppi compromessi per realizzare il proprio prodotto.

Per quelle realtà che necessitano di realizzare un prodotto di massima qualità, nel quale la *user experience* sia un punto centrale, questo approccio non è applicabile. Invece le migliori possibilità di applicazione si avranno ad esempio in prodotti per utilizzo interno alle aziende o personale, per i quali qualche imperfezione grafica sarebbe assolutamente accettabile a fronte di una maggiore velocità di sviluppo o disponibilità di *features*.

.NET MAUI è un framework con un obiettivo ambizioso, e vanta delle fondamenta tecniche decisamente solide e collaudate, al momento è però carente dal punto di vista dell'affidabilità. Nel breve futuro può sicuramente però migliorare e diventare una valida opzione per lo sviluppo multipiattaforma.

Una possibilità che andrebbe sicuramente esplorata consiste nell'utilizzo di .NET MAUI come stack di tecnologie mantenendo però una suddivisione delle interfacce. Anche banalmente utilizzare un file xaml diverso per ogni piattaforma o per le due tipologie *mobile**desktop* permetterebbe di ridurre di gran lunga il numero di bug e problemi durante il ciclo di sviluppo e inoltre permetterebbe interfacce più aderenti alle piattaforme. Questo tipo di approccio sarebbe vantaggioso anche in quanto uno dei costi dello sviluppo *native* viene dal fatto che lo sviluppatore deve essere formato su tecnologie diverse, in questo caso il linguaggio per l'interfaccia sarebbe sempre lo stesso XAML (o C#).

Ringraziamenti

Questa tesi e la conseguente laurea sono solo l'atto conclusivo di uno dei percorsi più belli e importanti della mia vita. Durante questi anni di università ho lavorato sodo e mi sono impegnato davvero tanto. È giusto però sottolineare la fortuna di aver sempre avuto, dall'inizio alla fine di questa esperienza, delle persone fantastiche al mio fianco senza le quali non penso che tutto ciò sarebbe stato possibile.

Vorrei quindi ringraziare tanto tutti gli amici con i quali ho trascorso momenti di spensieratezza così come quelli con i quali ho portato avanti dei progetti ambiziosi.

Desidero ringraziare tanto la mia famiglia, che mi ha sempre spronato e soprattutto appoggiato nelle diverse scelte che ho preso durante questo percorso.

Dulcis in fundo, vorrei ringraziare tanto la mia ragazza. Ogni successo che ho ottenuto durante questi anni è stato ancora più bello una volta festeggiato con lei. La cosa però per la quale non potrò mai ringraziarla abbastanza è il fatto che ho sempre potuto contare su di lei nei momenti di difficoltà, nei quali mi è stata vicina, e mi ha aiutato a ritrovare la fiducia in me stesso.

Bibliografia

[1] u/Jazzlike_Apricot_254. Maui's potential hindered by inadequate maintainership, 2023. https://www.reddit.com/r/dotnet/comments/10g734a/mauis_potential_hindered_by_inadequate/.