

Università degli Studi di Camerino

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica (Classe L-31)



INTEGRAZIONE MICROSOFT DIAL SU DASHBOARD LOCCIONI

Laureandi

Daniele Moschini

Matricola 101439

Relatore

Prof. Michele Loreti

Correlatore

Luca Mazzuferi

A.A. 2020/2021

Indice

1	Introduzione	11
1.1	Obiettivo	11
1.2	Struttura della Tesi	11
2	Analisi e Studio di Fattibilità	13
2.1	Ricerca	14
2.2	Problematiche riscontrate	15
3	Metodologie Di Sviluppo	17
3.1	Scrum	17
3.2	Rilascio Incrementale	19
4	Tecnologie e strumenti	21
4.1	Aulos	21
4.2	Angular	21
4.3	TypeScript	22
4.4	HTML	22
4.5	KendoUI	23
4.6	Microsoft UWP	23
4.7	C#	24
4.8	Libreria RadialController	24
4.9	Microsoft Dial	25
4.10	Git	26
4.11	GitLab	26
4.12	Miro	26
4.13	Teams	26
4.14	Trello	27
4.15	Visual Studio	27
4.16	Remote Debugger	27
4.17	Visual Studio Code	27
5	Fasi Di Sviluppo	29
5.1	Prima Fase	29
5.2	Seconda Fase	29
5.3	Terza Fase	30

5.4	Quarta Fase	30
6	Architettura	33
6.1	Servizio Angular	33
6.1.1	Singleton services	34
6.1.2	DialService	34
6.1.3	DialProxy	35
6.1.4	DialBackendBridge	36
6.1.5	DialFrontendBridge	37
6.1.6	DialMenuLoader	39
6.1.7	Widget	42
6.2	Applicazione UWP	43
6.2.1	Core	43
6.2.2	AulosCollection e AulosIcon	43
6.2.3	DialController	44
6.2.4	WebNotifier	44
6.2.5	DialMenuItem	44
6.2.6	MainPage	44
7	Posizionamento su schermo	45
8	Evoluzione	49
8.1	Sezione Esempio	49
8.2	Sezione Esempio	49
9	UtilizzoDelServizio	51
10	Testing	53
10.1	MSTest	53
11	Conclusioni	55
12	SviluppiFuturi	57

Elenco dei codici

6.1	Esempio creazione servizio	33
6.2	Recupero oggetto C#	34
6.3	Wrapping oggetto recuperato	35
6.4	Interfaccia DialBackendBridge	36
6.5	EventEmitter esposti da DialFrontendBridge	37
6.6	Inserimento classe DialFrontendbridge in window	37
6.7	Metodi esposti da DialFrontendBridge	38
6.8	Metodo loadLevelMenuUp	39
6.9	Metodo loadLevelMenuDown	39
6.10	Metodo loadLevelMenuUp	40
6.11	Metodo componentSubscribe	41
6.12	Metodo menuSubscribe	41
6.13	Sottoscrizione evento rotazione Radial Gauge	42
7.1	Metodo ScreenContactStartedAsync	46
7.2	Spostamento cursore	46
7.3	Spostamento cursore	47
9.1	Creazione nuova voce da widget	51
9.2	Ascolto Invoke della voce di menu'	51
9.3	Ascolto eventi associati alla voce di menu' selezionata	52
10.1	Test SetMenu	53
10.2	Test aggiunta voce di Menu'	53

Elenco delle figure

4.1	Aulos	21
4.2	Angular	21
4.3	Typescript	22
4.4	HTML	23
4.5	KendoUI	23
4.6	Microsoft UWP	23
4.7	C#	24
4.8	Microsoft Dial	25
4.9	Git	26
4.10	GitLab	26
4.11	Miro	26
4.12	Teams	26
4.13	Trello	27
4.14	Visual Studio	27
4.15	Visual Studio Code	27
7.1	Microsoft Dial posizionato sopra un surface pro 6	45
11.1	Potenziometro utilizzato attualmente	55
11.2	Potenziometro utilizzato attualmente	56

Elenco delle tabelle

1. Introduzione

Considerato che, negli ultimi anni, lo sviluppo e controllo di software aziendali si sta spostando in maniera decisa verso il Web, permettendo una maggior portabilità e scalabilità, con conseguente controllo da remoto tramite qualsiasi dispositivo connesso alla rete e che l'utilizzo di dispositivi di input che utilizzano il protocollo HID é sempre piú richiesto e talvolta necessario per una miglior esperienza utente, il gruppo Loccioni ci ha proposto di analizzare le possibili strade da percorrere per integrare un dispositivo Microsoft, in particolare il Surface Dial, che grazie al posizionamento su schermo touch-screen, permetta l'acquisizione di componenti e il controllo degli stessi. Lo scopo finale di questo progetto é quindi quello poter integrare il dispositivo "Microsoft Surface Dial" nei software Loccioni, sia lato Web che Windows Form, con l'obiettivo di rendere piú fluida e intuitiva l'esperienza utente, consentendo di svolgere operazioni in parallelo che, normalmente vengono eseguite tramite dispositivi di input classici come mouse e tastiera, attraverso l'utilizzo di entrambe le mani, mantenendo il focus di un dispositivo su un determinato compito.

Aggiungere foto banchi test loccioni e allungare il brodo.

1.1 Obiettivo

Descrivere brevemente la visione di Luca per l'utilizzo finale del dial nei banchi test Loccioni

1.2 Struttura della Tesi

L'esposizione di questo elaborato sará cosi suddivisa: - il capitolo n "nome" descrive bla bla bla - il capitolo n "nome" descrive bla bla bla - il capitolo n "nome" descrive bla bla bla - il capitolo n "nome" descrive bla bla bla

2. Analisi e Studio di Fattibilità

Nel nostro primo incontro con l'impresa Loccioni siamo stati accompagnati dal nostro tutor aziendale, Luca Mazzuferi, nelle varie aree dell'impresa per verificare dove il Dial possa effettivamente aiutare l'operatore nelle azioni quotidiane per la calibrazione o la modifica di dati in modo efficiente e sicuro, migliorandone la user experience.

L'idea iniziale era quella di utilizzare il Dial per il controllo di macchinari, che attualmente fanno uso di programmi Windows Forms per la manipolazione dei dati, mentre si appoggiano ad un loro Framework Web per la visualizzazione in tempo reale dei dati stessi.

Per prima cosa ci siamo dedicati alla lettura della documentazione scritta da Microsoft e siamo giunti alla conclusione che le funzionalità base del Dial sono quattro eventi principali:

- **Click:** Pressione rapida del bottone presente nel dispositivo.
- **Pressione prolungata:** Pressione prolungata del bottone presente nel dispositivo.
- **Rotazione:** Evento rotazione che avviene ruotando il dispositivo
- **Posizionamento sullo schermo:** Il dispositivo può essere appoggiato in alcuni schermi compatibili ed essere riconosciuto dal sistema.

Una volta individuate le potenzialità del Dial, abbiamo iniziato una fase di testing su applicazioni native per comprendere come e dove potevamo utilizzare le librerie messe a disposizione per il dispositivo, con la successiva creazione di un piccolo prototipo che sfruttasse e mostrasse le suddette funzionalità.

Successivamente ci siamo dedicati alla comunicazione tra un qualsiasi tipo di ambiente software e il Dial, scoprendo che il dispositivo comunica attraverso il protocollo HID con qualsiasi Hardware, ma senza l'utilizzo della Libreria RadialController, si vanno a perdere tutte quelle funzionalità che permettono l'utilizzo completo del Dial, rendendolo meno necessario in termini di miglioramento della User Experience.

2.1 Ricerca

Durante lo studio di fattibilità, ci siamo dedicati alla ricerca di progetti che utilizzassero il Dial indifferentemente dall'ambiente, da quello grafico a quello audio, trovando solamente un programma che permetteva l'utilizzo del Dial in un Sintetizzatore Software, attraverso il protocollo MIDI, limitando il controllo alla semplice rotazione e pressione, togliendo tutte quelle funzionalità più interessanti, come il posizionamento del dispositivo sullo schermo e la visualizzazione dinamica del Menu in relazione alla posizione in cui trova.

Per questo motivo, abbiamo deciso in concordato con i nostri tutor Aziendali di non gestire noi la comunicazione tra dispositivo e software, ma di lasciare questo compito alle librerie messe a disposizione da Microsoft.

Dato il crescente sviluppo dell'utilizzo di applicazioni web e del relativo controllo delle stesse attraverso dispositivi HID, abbiamo quindi proseguito con la ricerca per l'integrazione dei suddetti dispositivi nel mondo Web, accantonando per il momento, la possibilità di integrarlo negli applicativi Windows Forms Loccioni, anche a causa del necessario refactoring di tutte le applicazioni che intendessero utilizzarlo.

Considerando che il posizionamento su schermo è supportato solamente da determinati dispositivi Microsoft (Surface Studio, Surface Pro 4/5/6, ...) abbiamo pensato di creare un'applicazione UWP (Universal Windows Platform) che potesse girare nativamente su tali dispositivi. Per poter infine, acquisire il controllo di una pagina web, abbiamo deciso di utilizzare una WebView avviata a RunTime permettendo all'utente di poter utilizzare il Dial sui dispositivi che supportano l'interazione con lo schermo grazie alle Librerie Native, ma anche di poterla utilizzare senza l'utilizzo del Dial, attraverso un qualsiasi browser web.

2.2 Problematiche riscontrate

Una volta avvenuto il caricamento della pagina Web nell'applicazione UWP, ci rimaneva da risolvere il problema della comunicazione tra due linguaggi differenti, quello Web, nel nostro caso, basato sul TypeScript e quello UWP interamente in C e XAML.

Le comunicazioni dovevano avvenire in maniera bi-direzionale, questo significa che non é solamente l'applicazione UWP a comunicare con la pagina Web, ma anche viceversa. Affinché l'applicazione potesse comunicare con la pagina Web, grazie alla funzionalità `InvokeScriptAsync`, messa a disposizione dalla `WebView`, abbiamo definito delle chiamate asincrone che potessero inviare parametri o notificare l'avvenuta emissione di un evento alla pagina Web stessa. Il problema era come poter comunicare dalla pagina Web all'applicazione UWP che fosse avvenuto qualcosa.

Per risolvere questa problematica ci siamo trovati davanti a due possibilità:

1. Utilizzare la funzionalità della `WebView` chiamata `ScriptNotify` che permette la notifica all'applicazione UWP dell'avvenuta emissione di un determinato evento nella pagina Web.
2. Utilizzare il decoratore `AllowForWeb` su una classe, affinché un'istanza di essa, possa essere iniettata nella pagina Web.

Dopo aver letto la documentazione e effettuato vari test, la scelta è ricaduta sul decoratore `AllowForWeb`, in quanto lo `ScriptNotify` permetteva di mettersi in ascolto di solamente un determinato evento e permetteva il ritorno di una sola stringa come parametro, oltre al fatto che leggendo su varie Community come StackOverflow, venisse sconsigliata questa pratica poiché non sicura in quanto andava necessariamente abilitato lato UWP l'URI di ogni singola pagina da controllare e limitante in termini di prestazioni.

In contrapposizione, utilizzare `AllowForWeb` su di una classe e iniettare un'istanza nella pagina, ci permetteva un maggior controllo della comunicazione tra Web e UWP grazie all'utilizzo dei metodi messi a disposizione dall'oggetto, permettendo di passare maggiori informazioni e parametri tipati e non solamente stringhe.

A quel punto l'unico problema rimasto era l'utilizzo di questo oggetto all'interno del Framework Aulos Loccioni, che abbiamo però risolto attraverso lo sviluppo di un servizio Angular che aggiunge livelli di sicurezza e definisce delle API di utilizzo per gli sviluppatori futuri che vorranno integrare l'utilizzo del Dial nelle proprie pagine Web.

3. Metodologie Di Sviluppo

Durante lo sviluppo del progetto abbiamo utilizzato uno sviluppo Scrum unito al modello incrementale, utilizzando un Scrum per quanto riguarda la distribuzione del lavoro e il rapporto con l'azienda, mentre il rilascio incrementale è stato utilizzato una volta scoperte tutte le potenzialità del Dial e il corretto funzionamento su pagine web.

3.1 Scrum

Scrum è un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo, creato e sviluppato da Ken Schwaber e Jeff Sutherland.

Scrum enfatizza tutti gli aspetti di gestione di progetto legati a contesti in cui è difficile pianificare in anticipo. Vengono utilizzati meccanismi propri di un "processo di controllo empirico", in cui cicli di feedback che ne costituiscono le tecniche di management fondamentali risultano in opposizione alla gestione basata sul concetto tradizionale di command-and-control. Il suo approccio alla pianificazione e gestione di progetti è quello di portare l'autorità decisionale al livello di proprietà e certezze operative.

Sono 3 i ruoli che vengono individuati nella metodologia Scrum: *Product Owner*, *Scrum Master* e *Team di Sviluppo*:

- Il **Product Owner** definisce il lavoro da svolgere e l'ordine con cui viene completato. Raccoglie la voce degli stakeholder (clienti, management e chiunque abbia un interesse nel prodotto), le necessità dell'utente finale, i requisiti del mercato e sulla base di questi elementi stabilisce le priorità di sviluppo per il Team Scrum.
- Lo **Scrum Master** è il responsabile del processo, e un leader a servizio (servant-leader) dello Scrum Team. Conoscitore esperto della metodologia Scrum, sa come applicarla e si assicura che il Team comprenda e segua le regole che la caratterizzano, perché il progetto abbia successo. Inoltre favorisce il lavoro del Team di Sviluppo, rimuovendo ostacoli, organizzando meeting di confronto, e soprattutto proteggendolo da ogni possibile distrazione: ogni membro del gruppo deve poter lavorare al 100 per 100 sullo sviluppo, e lo Scrum Master si assicura che questo avvenga.
- Il **Team di Sviluppo** è la squadra di lavoro, composta da 3 a 9 persone. Anche lo Scrum Master può far parte del Team di Sviluppo. Chi concretamente porta a termine gli Sprint e fornisce le funzionalità da implementare è questo insieme coordinato di persone, autogestito e cross-funzionale.

Per quanto riguarda il nostro progetto, abbiamo applicato il framework Scrum con i ruoli definiti nel seguente schema:

- **Product Owner:** Luca Mazzuferi
- **Scrum Master:** Marco Allegrezza, Diego Bonura
- **Team di Sviluppo:** Daniele Moschini, Michele Benedetti

3.2 Rilascio Incrementale

Per **modello incrementale** si intende, nell'ambito dell'ingegneria del software, un modello di sviluppo di un progetto software basato sulla successione dei seguenti passi principali:

- Pianificazione
- Analisi dei requisiti
- Progetto
- Implementazione
- Prove
- Valutazione

Questo ciclo può essere ripetuto diverse volte, in cui ogni "incremento" riduce il rischio di fallimento e produce nuovo valore. Il ciclo viene ripetuto fino a che la valutazione del prodotto diviene soddisfacente rispetto ai requisiti previsti.

L'utilizzo del modello incrementale è consigliabile quando si ha, fin dall'inizio della progettazione, una visione abbastanza chiara dell'intero progetto, perché occorre fare in modo che la realizzazione della generica versione k risulti utile per la realizzazione della versione $k+1$.

Un approccio incrementale è particolarmente indicato in tutti quei casi in cui la specifica dei requisiti risulti particolarmente difficoltosa e di difficile stesura (semi)formale. L'uso di questo modello di sviluppo favorisce la creazione di prototipi, ovvero parti di applicazione funzionanti, che a loro volta favoriscono il dialogo con il cliente e la validazione dei requisiti.

4. Tecnologie e strumenti

Durante lo sviluppo del progetto, sono state utilizzate varie tecnologie, strumenti, linguaggi e relativi framework che verranno dettagliatamente descritti di seguito.

4.1 Microsoft Dial



Figura 4.1: Microsoft Dial

4.2 Aulos



Figura 4.2: Aulos

AULOS è un framework ad uso interno per la standardizzazione dello sviluppo delle commesse. Gli obiettivi principali di questo framework sono:

- rendere standard e subito disponibili quelle funzionalità che sono di frequente implementazione tra le commesse;
- fornire uno standard UI e UX che sia ben riconoscibile, ma soprattutto coerente tra commesse diverse;
- garantire l'interoperabilità tra tecnologie diverse.

AULOS fa uso di tecnologie web come Angular per il proprio frontend, mentre il backend – a cui esso si aggancia e che regola il funzionamento della commessa – può essere scritto in più linguaggi come C# e LabView. Da qui si nota come AULOS sia anche uno standard di comunicazione: l'utilizzo di API apre alla possibilità di sviluppare il backend nella tecnologia che meglio si sposa con il problema da risolvere.



Figura 4.3: Angular

4.3 Angular

Angular 2+ (o semplicemente Angular) è un framework open source per lo sviluppo di applicazioni web con licenza MIT, evoluzione di AngularJS. Sviluppato principalmente da Google, la sua prima release è avvenuta il 14 settembre 2016. Angular è stato completamente riscritto rispetto a AngularJS e le due versioni non sono compatibili. Il linguaggio di programmazione usato per AngularJS è JavaScript mentre quello di Angular è TypeScript. Le applicazioni sviluppate in Angular vengono eseguite interamente dal web browser dopo essere state scaricate dal web server (elaborazione lato client). Questo comporta il risparmio di dover spedire indietro la pagina web al web-server ogni volta che c'è una richiesta di azione da parte dell'utente. Il codice generato da Angular gira su tutti i principali web browser moderni quali ad esempio Chrome, Microsoft Edge, Opera, Firefox, Safari ed altri. Angular è stato progettato per fornire uno strumento facile e veloce per sviluppare applicazioni che girano su qualunque piattaforma inclusi smartphone e tablet. Infatti le applicazioni web in Angular in combinazione con il toolkit open source Bootstrap diventano responsive, ossia il design del sito web si adatta in funzione alle dimensioni del dispositivo utilizzato. È in corso di sviluppo un altro toolkit di design responsivo, Flex Layout, più semplice da usare rispetto a Bootstrap e concepito appositamente per Angular. Altro toolkit che facilita la progettazione in Angular è Angular Material, una serie di componenti che permette di creare una pagina web molto velocemente: con l'utilizzo combinato di Flex Layout ed Angular Material si possono creare siti e applicazioni web responsive molto avanzate basate su Angular.

4.4 TypeScript



Figura 4.4: Typescript

TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft. Si tratta di un Super-set di JavaScript che basa le sue caratteristiche su ECMAScript 6; capo del progetto è Anders Hejlsberg. Il linguaggio estende la sintassi di JavaScript in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica. È stato progettato per lo sviluppo di grandi applicazioni ed è destinato a essere compilato in JavaScript per poter essere interpretato da qualunque web browser o app.

4.5 HTML



Figura 4.5: HTML

L'HTML è un linguaggio di pubblico dominio, la cui sintassi è stabilita dal World Wide Web Consortium (W3C). È derivato dall'SGML, un metalinguaggio finalizzato alla definizione di linguaggi utilizzabili per la stesura di documenti destinati alla trasmissione in formato elettronico. La versione attuale, la quinta, è stata rilasciata dal W3C nell'ottobre 2014. Il motivo principale che ha spinto il W3C e i suoi membri a sviluppare HTML5 è stata la necessità di fornire direttamente le funzionalità che in precedenza erano fruibili tramite estensioni proprietarie all'esterno dei browser, come Adobe Flash e simili. Un secondo obiettivo che gli sviluppatori si erano prefissati era quello di garantire una maggiore compatibilità tra i diversi browser, indipendentemente dalla piattaforma software utilizzata, e principalmente mirata all'espansione dei dispositivi mobili.

4.6 KendoUI



Figura 4.6: KendoUI

Kendo UI è un framework integrale di interfaccia utente HTML5 per costruire applicazioni e siti web interattivi e ad alte prestazioni.

4.7 Microsoft UWP



Figura 4.7: Microsoft UWP

UWP è uno dei numerosi modi per creare applicazioni client per Windows. Le app UWP usano le API WinRT per offrire potenti funzionalità avanzate di interfaccia utente e asincrone, ideali per i dispositivi connessi a Internet. Per scaricare gli strumenti necessari per iniziare a creare le app UWP, vedere Effettuare la configurazione e quindi Scrivere la prima app ed è la soluzione ideale per creare app che vengono eseguite nei dispositivi Windows 10 e possono essere combinate con altre piattaforme. Le app

UWP possono usare le API Win32 e le classi .NET (vedere Set di API per le app UWP, DLL per le app UWP e .NET per le app UWP). Il progetto di sviluppo Microsoft continua a evolversi e, insieme a iniziative come WinUI, MSIX e Project Reunion, UWP rappresenta uno strumento potente per la creazione di app client. Un'app UWP è:

- Sicura: le app UWP dichiarano le risorse del dispositivo e i dati a cui accedono. L'utente deve autorizzare tale accesso.
- In grado di usare un'API comune in tutti i dispositivi che eseguono Windows 10.
- In grado di usare funzionalità specifiche del dispositivo e di adattare l'interfaccia utente alle dimensioni, alle risoluzioni e ai DPI dello schermo di dispositivi diversi.
- Disponibile nel Microsoft Store in tutti i dispositivi (o solo a quelli specificati) che eseguono Windows 10. Il Microsoft Store offre diversi modi per realizzare profitti con un'app.
- In grado di essere installata e disinstallata senza rischio o danni per il computer.
- Coinvolgente: è possibile usare riquadri animati, notifiche push e attività utente che interagiscono con Sequenza temporale di Windows e la funzionalità di ripristino della ricerca dal punto in cui è stata interrotta di Cortana per coinvolgere gli utenti.
- Programmabile in C#, C++, Visual Basic e Javascript. Per l'interfaccia utente usare WinUI, XAML, HTML o DirectX

4.8 C#



Figura 4.8: C#

Il C# è un linguaggio di programmazione orientato agli oggetti sviluppato da Microsoft all'interno dell'iniziativa .NET, e successivamente approvato come standard dalla ECMA (ECMA-334) e ISO (norma ISO/IEC 23270). La sintassi e struttura del C# prendono spunto da vari linguaggi nati precedentemente, in particolare Delphi, C++, Java e Visual Basic.

4.9 Libreria RadialController

La libreria RadialController e le API correlate consentono di personalizzare sia il menù dei comandi integrato che l'esperienza di interazione supportata dall'app. Metodi utilizzati:

- `RadialController.CreateForCurrentView()`
Metodo statico che consente di ricevere un oggetto della classe `RadialController` creato appositamente per il contesto in cui gira l'applicazione (Dial utilizzato, versione del sistema operativo, tipo di hardware, ...)
- `.CreateFromIcon("Sample", icon)`
Metodo che consente di creare un `RadialControllerMenuItem` ovvero un oggetto che inserito all'interno del menu verrà visualizzato e consentirà di svolgere un determinato comportamento quando selezionato.
- `.Menu.Items.`
 - `Add(RadialControllerMenuItem item)`
 - Inserisce il `RadialControllerMenuItem` all'interno del menu.
 - `CreateFromKnownIcon(string displayText, RadialControllerMenuKnownIcon value)`
 - Crea un oggetto di tipo `RadialControllerMenuItem` da un'icona già presente nella libreria.
 - `CreateFromIcon(string displayText, RandomAccessStreamReference icon)`
 - Crea un oggetto di tipo `RadialControllerMenuItem` da un'icona presente negli assets
 - `CreateFromFontGlyph(string displayText, string glyph, string fontFamily)`
 - Crea un oggetto di tipo `RadialControllerMenuItem` da un font installato nel sistema / inserito nell'app, e da un codice esadecimale ad esso associato

Eventi utilizzati:

- `radialController.ButtonClicked`
Evento che informa della pressione breve del dispositivo Dial.
- `radialController.ButtonPressed`
Evento che informa della pressione prolungata del dispositivo Dial
- `radialController.RotationChanged`
Evento che informa della rotazione del dispositivo Dial
- `.ScreenContactStarted`
Evento che informa del posizionamento del Dial su schermo
- `.ScreenContactContinued`
Evento che informa sull'utilizzo del Dial su schermo
- `.ScreenContactEnded`
Evento che informa sulla fine del posizionamento su schermo

API correlate

`RadialControllerConfiguration` è una classe correlata alla classe `RadialController` e permette di configurare l'oggetto della medesima classe modificando le sue proprietà. `SetDefaultMenuItems` consente di scegliere tra le voci di menù già predisposte dalla libreria quale inserire nel menù.



Figura 4.9: Git

4.10 Git

4.11 GitLab



Figura 4.10: GitLab

4.12 Miro



Figura 4.11: Miro

4.13 Teams



Figura 4.12: Teams



Figura 4.13: Trello

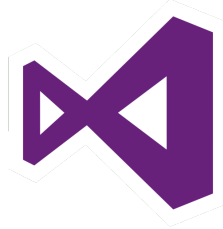


Figura 4.14: Visual Studio

4.14 Trello

4.15 Visual Studio

4.16 Remote Debugger

4.17 Visual Studio Code



Figura 4.15: Visual Studio Code

5. Fasi Di Sviluppo

In questo capitolo verranno dettagliate le fasi atte allo sviluppo del progetto che é stato possibile suddividere in 4 *macro fasi*:

- **Prima Fase:** Analisi e Studio di fattibilità.
- **Seconda Fase:** Implementazione comunicazione bi-direzionale tra UWP e Web.
- **Terza Fase:** Ottimizzazione della complessità nella comunicazione.
- **Quarta Fase:** Creazione Servizio Angular e implementazione posizionamento sullo schermo.

5.1 Prima Fase

Nella prima fase, durata circa un mese e mezzo, ci siamo dedicati alla scoperta delle tecnologie utilizzate e alla fattibilità delle richieste presentate dall'Impresa. Per prima cosa ci siamo dedicati allo studio del Microsoft Dial e alle sue potenzialità, scoprendo le librerie dedicate ad esso e le applicazioni che le supportavano.

La prima importante decisione è stata quale piattaforma utilizzare per l'avvio del progetto, è la decisione è poi ricaduta sulle applicazioni UWP per 3 motivi principali:

1. Il continuo rilascio da parte di Microsoft di miglioramenti su questo tipo di piattaforma, dato che diventerà la base di tutti i programmi dell'Impresa che utilizzano il Dial.
2. La documentazione scritta ed i progetti d'esempio che utilizzano il Dial sono tutti fatti attraverso UWP.
3. 'applicazione UWP consente l'utilizzo al suo interno di una WebView che ci permette di comunicare con la pagina Web.

Questa prima fase è stata molto impegnativa, poichè la documentazione presente è basilare e non spiega interamente come svolgere le azioni più complesse, ma siamo riusciti a portarla a termine anche grazie al nostro collega Sebastiano Verdolini, che ha svolto il suo stage universitario con noi.

5.2 Seconda Fase

Nella seconda fase, durata circa 5 settimane, abbiamo sviluppato prototipi di applicazioni UWP atti a testare i vari comportamenti del Dial e le sue funzionalità native,

integrando verso la fine una pagina HTML statica all'interno della WebView, includendo delle funzioni JavaScript per la comunicazione UWP → Web.

Oltre a questo abbiamo seguito un **Webinar** su Angular e sul framework Aulos, creato ed utilizzato, dall'impresa stessa. Nella parte conclusiva della fase siamo riusciti ad utilizzare un decoratore di classe chiamato **AllowForWeb** presente in UWP che ci ha permesso di iniettare a RunTime in una nostra pagina Web, un oggetto C# che rappresenta il ponte di comunicazione tra le due piattaforme in maniera *bi-direzionale*.

5.3 Terza Fase

Nella terza fase abbiamo iniziato a lavorare sul GetStarted fornitoci dal gruppo Loccioni con i servizi Aulos da loro sviluppati. Una volta compreso il funzionamento del Framework Aulos ci siamo spostati sull'integrazione della pagina Web nella nostra WebView al fine di poterla controllare grazie agli sviluppi conseguiti durante la precedente fase.

Dopo vari tentativi e difficoltà, siamo riusciti ad integrarla all'interno della nostra UWP, ma rimaneva il problema principale riscontrato in questa fase, ovvero il posizionamento all'interno della parte Web delle funzioni richiamabili dalla UWP. Grazie all'aiuto dei nostri tutor siamo arrivati alla soluzione che ci consente di inserire una classe con i relativi metodi messi a disposizione all'interno della variabile globale "window" contenuta all'interno delle pagine web, la quale permette uno scope globale degli oggetti inseriti al suo interno e dei loro relativi metodi e variabili.

Discutendo con i nostri Tutor sui vantaggi e gli svantaggi che questo approccio presenta, anche se non pienamente consigliato in termini di sicurezza, ci è stato comunque consentito l'utilizzo, in quanto viene già utilizzato in altre applicazioni Loccioni su rete privata e quindi meno soggetta a possibili usi indesiderati, inoltre al momento dello sviluppo, non vi erano altre soluzioni possibili e funzionanti per la comunicazione.

Una volta integrata la comunicazione all'interno di Angular abbiamo iniziato la definizione delle API attraverso le quali è possibile lo scambio di informazioni tra le due applicazioni.

5.4 Quarta Fase

Nella quarta ed ultima fase ci siamo dedicati allo sviluppo del prodotto "finale" ovvero la creazione di un servizio Angular utilizzabile all'interno dei componenti o sotto-servizi, per l'integrazione su Dashboard Loccioni del dispositivo Surface Dial.

La fase è iniziata modificando radicalmente le API create in precedenza, attraverso lo spostamento di responsabilità che prima erano presenti all'interno dell'UWP, dentro alla parte Web, consentendo così dei messaggi più leggeri nella comunicazione e allo stesso tempo lasciando più libertà ai futuri utilizzatori del servizio, in quanto sino a questo momento, la modifica di un determinato componente avveniva attraverso una dipendenza diretta con il DOM della pagina ricercando l'identificatore dell'elemento da

controllare.

La modifica ci ha quindi permesso di rimuovere la dipendenza diretta dal DOM e dagli identificatori, permettendo un comportamento dinamico a seconda dell'evento intercettato dal servizio.

Alla fine di questa fase abbiamo ricevuto il materiale sul quale effettuare i test per l'utilizzo del Dial su schermo, che ci ha obbligato ad aggiungere funzionalità al servizio, non previste in precedenza, ma consentendoci l'acquisizione di elementi Web della dashboard, attraverso il posizionamento fisico del dispositivo sullo schermo.

6. Architettura

In questo capitolo verrà esposta l'architettura finale dell'applicativo sviluppato che permette la comunicazione tra il Surface Dial e la pagina web.

Il progetto si presenta suddiviso in due parti ben distinte:

- **Servizio Angular.**
- **Applicazione UWP** (Universal Windows Platform).

6.1 Servizio Angular

In Angular, i Servizi rappresentano un tassello fondamentale per la realizzazione di un'applicazione. Un servizio è solitamente rappresentato da una classe indipendente dalla View che viene definita per svolgere un compito ben preciso ed effettuare delle operazioni strettamente correlate tenendo in mente il principio di singola responsabilità.

È possibile definire più servizi all'interno di un'applicazione, ognuno dei quali si occupa di portare a termine un determinato incarico incrementando la modularità del progetto, inoltre un servizio può a sua volta dipendere da altri servizi, che vengono definiti nel costruttore del servizio stesso.

Avvalendosi di Angular CLI, possiamo creare un Servizio tramite il comando:

```
1 ng generate service <nome-del-servizio> <flag-opzionali>
```

Codice 6.1: Esempio creazione servizio

Una volta creati, i servizi possono essere iniettati all'interno di uno o più componenti grazie al meccanismo di *Dependency Injection*.

Per poter utilizzare un servizio da noi definito, dobbiamo registrarlo con uno degli Injector presenti in Angular. Esiste infatti una gerarchia di Injector, che Angular provvede a creare, alla base dei quali c'è il cosiddetto Root Injector, che viene creato in fase di bootstrap dell'applicazione e i servizi registrati con quest'ultimo sono disponibili per tutta l'applicazione.

Per ogni Injector esiste sempre una sola istanza di un determinato servizio, come specificato nella documentazione ufficiale:

"Services are singletons within the scope of an injector. That is, there is at most one instance of a service in a given injector."

L' Injector si occupa di creare le dipendenze e si serve di un Provider, un oggetto il quale indica all'Injector come ottenere o creare un'istanza di una dipendenza. Per ogni dipendenza che si vuole usare nell'applicazione si deve registrare un provider con un Injector, in modo che quest'ultimo possa poi utilizzare il provider per creare nuova istanza di quella dipendenza.

6.1.1 Singleton services

Per restare in linea con le direttive utilizzare dal gruppo Loccioni nello sviluppo dei loro servizi, abbiamo utilizzato il `forRoot()` pattern per l'injection del servizio nell'applicazione.

Questo pattern permette di specificare un modulo personalizzato per il servizio, il quale contiene un metodo statico chiamato appunto `forRoot()` che restituisce un `ModuleWithProvider` parametrizzato al modulo del Servizio.

Questo pattern permette di effettuare l'import del modulo nell'`AppModule` richiamando il metodo `forRoot()`, così da non dover appesantire il Provider del modulo iniziale e in linea di principio è bene utilizzarlo quando un provider prende in ingresso dei parametri, così da non dover specificarli ogni qualvolta il modulo venga utilizzato.

6.1.2 DialService

Lo scopo finale del servizio che abbiamo implementato è quello di poterlo utilizzare come ponte o intermediario tra il Surface Dial, che si interfaccia nativamente con l'applicazione UWP, la quale contiene una `WebView` con all'interno caricata una pagina custom Loccioni, e i componenti della View all'interno della pagina che intendono utilizzare il servizio.

Il componente che dipenderà da questo determinato servizio, chiamato appunto `DialService`, potrà facilmente utilizzare le funzionalità messe a disposizione dal Surface Dial e interpretare gli eventi intercettati a `RunTime`.

Il core del servizio implementato è rappresentato dalla Classe `DialService`, essa infatti si occupa di inizializzare il servizio e rendere disponibile la comunicazione bi-direzionale tra il lato Web e quello UWP. In particolare si occupa di recuperare l'oggetto `C#` iniettato a sua volta dentro l'oggetto `window` della pagina web, con visibilità globale.

```
1 this.dial = (window as any).dial;
```

Codice 6.2: Recupero oggetto `C#`

L'oggetto dial viene poi inserito per questioni di sicurezza in una classe DialProxy, la quale implementa a sua volta l'interfaccia da noi sviluppata DialBackendBridge, nella quale sono presenti tutte le funzionalità dell'oggetto "dial", richiamabili quindi da Web.

6.1.3 DialProxy

Al fine di aggiungere un livello di sicurezza sull'oggetto iniettato e sul suo successivo utilizzo, abbiamo deciso di implementare una classe chiamata DialProxy con la responsabilità di wrappare l'oggetto e quindi di intercettare le chiamate fatte su di esso.

La classe DialProxy prende quindi in ingresso nel suo costruttore un oggetto di tipo DialBackendBridge, che rappresenta l'oggetto iniettato, e come secondo parametro il servizio DialService per poter accedere rapidamente alle proprietà del servizio e poter effettuare determinati controlli prima di richiamare un metodo.

```
1  this.dialProxy = new DialProxy(this.dial, this);
```

Codice 6.3: Wrapping oggetto recuperato

6.1.4 DialBackendBridge

DialBackendBridge rappresenta l'interfaccia contenente la dichiarazione dei metodi implementati nella classe DialController dell'applicazione UWP.

In questo modo, tramite l'oggetto dial iniettato, sarà possibile richiamare a RunTime i metodi dichiarati nel C# attraverso il Web.

```
1 export interface DialBackendBridge{
2
3     getProductId(): string;
4
5     setMenu(arrayOfItem: any[]): void;
6
7     clearMenu(): void;
8
9     deleteItem(tag: string): void;
10
11     addItem(item: any): void;
12
13     createDialMenuItem(tag: string, displayText: string, icon: string): any;
14
15     setDefaultItem(defaultItems: string[]): void;
16
17     manualInvoke(tag: string): void;
18
19 }
```

Codice 6.4: Interfaccia DialBackendBridge

6.1.5 DialFrontendBridge

Per rendere questa comunicazione bi-direzionale, abbiamo creato una classe DialFrontendBridge con la responsabilità di rendere visibili dei metodi all'interno dell'oggetto window e quindi richiamabili tramite script dall'applicazione UWP.

L'invocazione di questi metodi comporta la successiva emissione di un determinato evento a coloro che vi si sottoscrivono, con la possibilità del passaggio di parametri di tipo stringa.

Tramite l'utilizzo degli EventEmitter messi a disposizione dal core di Angular e definiti come mostrato nel codice sottostante, la classe DialFrontendBridge permette all'utilizzatore del servizio di sottoscrivere a determinati eventi pubblici, che quando richiamati emettono un determinato valore. RIFERIMENTO A CODICE

```

1 public onRotationEvent = new EventEmitter<{tag: string, degree: string}>();
2 public onPressedRotationEvent = new EventEmitter<{tag: string, degree: string}>();
3 public onClickEvent = new EventEmitter<string>();
4 public onInvokeEvent = new EventEmitter<string>();
5 public onScreenContactStartedEvent = new EventEmitter<{x: string, y: string}>();
6 public onScreenContactContinuedEvent = new EventEmitter<{x: string, y: string}>();
7 public onScreenContactEndedEvent = new EventEmitter();

```

Codice 6.5: EventEmitter esposti da DialFrontendBridge

Affinché questi metodi possano essere richiamati tramite script dall'applicazione UWP è stato necessario inserire l'istanza della classe all'interno dell'oggetto window, così da renderlo visibile e utilizzabile dall'esterno.

```

1 (window as any).DialFrontendBridge = this.dialFrontendbridge;

```

Codice 6.6: Inserimento classe DialFrontendbridge in window

```

1  /**
2   * Metodo richiamato da UWP che notifica la rotazione del Microsoft Dial.
3   * @param tag Identificatore dell'elemento che ha eseguito la rotazione
4   * @param degree Gradi della rotazione
5   */
6  public RotationEvent(tag: string, degree: string): string{
7      this.onRotationEvent.emit({tag, degree});
8      return degree;
9  }
10
11  /**
12   * Metodo richiamato da UWP che notifica la rotazione de
13   * l Microsoft Dial con il Dial premuto.
14   * @param tag Identificatore dell'elemento che ha eseguito la rotazione
15   * @param degree Gradi della rotazione
16   */
17  public PressedRotationEvent(tag: string, degree: string): string{
18      this.onPressedRotationEvent.emit({tag, degree});
19      return degree;
20  }
21
22  /**
23   * Metodo richiamato dall'UWP che notifica la pressione del del Microsoft Dial.
24   * @param tag Identificatore dell'elemento che ha eseguito il click sul Dial.
25   */
26  public ClickEvent(tag: string){
27      this.onClickEvent.emit(tag);
28  }
29
30  /**
31   * Metodo richiamato dall'UWP che notifica l'invocazione di un elemento di Menu.
32   * @param tag Identificatore del MenuItem che ha eseguito l'invoke.
33   */
34  public InvokeEvent(tag: string){
35      this.onInvokeEvent.emit(tag);
36  }
37
38  /**
39   * Metodo richiamato dall'UWP che notifica il contatto del
40   * dispositivo Dial con lo schermo nel punto con coordinate x e y.
41   * @param x Coordinata x della posizione
42   * @param y Coordinata y della posizione
43   */
44  public ScreenContactStartedEvent(x: string, y: string){
45      this.onScreenContactStartedEvent.emit({x, y});
46  }
47
48  /**
49   * Metodo richiamato dall'UWP che notifica lo spostamento del
50   * dispositivo Dial sullo schermo nel punto con coordinate x e y.
51   * @param x Coordinata x della posizione
52   * @param y Coordinata y della posizione
53   */
54  public ScreenContactContinuedEvent(x: string, y: string){
55      this.onScreenContactContinuedEvent.emit({x, y});
56  }
57
58  /**
59   * Metodo richiamato dall'UWP che norifica
60   * quando il Dial viene rimosso dallo schermo
61   */
62  public ScreenContactEndedEvent(){
63      this.onScreenContactEndedEvent.emit();
64  }

```

Codice 6.7: Metodi esposti da DialFrontendBridge

6.1.6 DialMenuLoader

Il secondo servizio implementato, chiamato DialMenuLoader é un perfetto esempio di come usufruire del servizio DialService all'interno di Componenti o Servizi.

DialMenuLoader ha lo scopo di creare dinamicamente le voci di Menu da inserire nel dial e di definire per ciascuna voce il comportamento, che in questo caso rappresenta la navigazione tra le voci di menu di un ServerLayout Lozioni.

Grazie al servizio DialService e a quello Router inseriti nel costruttore, questo sottoservizio permette di sottoscrivere automaticamente agli eventi di navigazione della pagina e a quelli del dial, così da caricare le giuste voci del menu in base al livello in cui si trova, caricando la voce di menu Home solo se si "atterra" in un componente altrimenti quella Back per navigare il menu nel senso opposto.

```

1  /**
2   * Carica un menu nel Dial e lo rende navigabile.
3   * @param menu Menu da caricare nel Dial.
4   */
5  public loadMenu(menu: Menu){
6      this.menu = menu;
7      this.menuSubscribe();
8      this.currentParent = this.menu.items[0];
9      this.loadLevelMenuUp(this.currentParent);
10 }

```

Codice 6.8: Metodo loadLevelMenuUp

Il servizio espone un metodo pubblico chiamato **loadMenu()**, il quale prende come parametro in ingresso un oggetto di tipo *Menu* passatogli dal layout dell'applicazione contenente tutte le voci di Menu disponibili all'interno della pagina web.

```

1  /**
2   * Carica i menu contenuti nell'item.
3   * @param item Item su cui caricare i sotto-menu
4   */
5  private loadLevelMenuDown(item: MenuItem){
6      this.currentMenuList = [];
7      item.items.forEach(child => {
8          this.currentMenuList.push(
9              this.dialService.dialProxy.createDialMenuItem(
10                 child.containerId,
11                 child.label,
12                 child.icon
13             ),
14          );
15      },
16      );
17      this.currentMenuList.push(
18          this.dialService.dialProxy.createDialMenuItem(
19              'Back',
20              'Back',
21              'back'
22          ),
23      );
24      this.dialService.dialProxy.setMenu(
25          this.currentMenuList
26      )

```

27

}

Codice 6.9: Metodo loadLevelMenuDown

Il metodo svolgerà le opportune analisi per ogni voce di menu contenuta e creerà dinamicamente le voci di menu da inserire nel menu del dial con i relativi comportamenti al richiamo di uno di essi, attraverso i metodi privati **loadLevelMenuDown()** e **loadLevelMenuDown()**.

```
1  /**
2   * Carica il livello di voci di menu sopra parent.
3   * @param parent MenuItem contenente il livello corrente.
4   */
5  private loadLevelMenuUp(parent: MenuItem) {
6      this.currentMenuList = [];
7      if (parent.level === 1) {
8          this.menu.items.forEach(item => {
9              this.currentMenuList.push(
10                 this.dialService.dialProxy.createDialMenuItem(
11                     item.containerId,
12                     item.label,
13                     item.icon
14                 )
15             );
16         }
17     );
18     } else {
19         this.menu.planItems.filter((value) => {
20             return value.level === parent.level && value.parent === parent.parent;
21         }).forEach((item) => {
22             this.currentMenuList.push(
23                 this.dialService.dialProxy.createDialMenuItem(
24                     item.containerId,
25                     item.label,
26                     item.icon
27                 )
28             );
29         }
30     );
31     this.currentMenuList.push(
32         this.dialService.dialProxy.createDialMenuItem(
33             'Back',
34             'Back',
35             'back'
36         )
37     );
38 }
39 this.dialService.dialProxy.setMenu(this.currentMenuList);
40 }
```

Codice 6.10: Metodo loadLevelMenuUp

Inoltre, attraverso i metodi **componentSubscribe()** e **menuSubscribe()** il servizio effettua dinamicamente le sottoscrizioni e disiscrizioni agli eventi interessati per la navigazione tra le voci di Menu.


```

1  /**
2  * Sottoscrizione avviata quando si atterra in un componente
3  * da una voce di menu attraverso la navigazione con il dial
4  */
5  private componentSubscribe() {
6      this.dialService.flagForHome = true;
7      this.dialService.dialProxy.setMenu([]);
8      this.componentListener =
9      this.dialService.dialFrontendBridge.onInvokeEvent.subscribe(
10     (tag: string) => {
11         if (tag === 'home'){
12             this.dialService.flagForHome = false;
13             this.currentParent = new MenuItem();
14             this.currentParent.level = 1;
15             this.loadLevelMenuUp(this.currentParent);
16             this.menuSubscribe();
17             this.componentListener.unsubscribe();
18             this.router.navigateByUrl('/server');
19         }
20     }
21 );
22 }

```

Codice 6.11: Metodo componentSubscribe

```

1  /**
2  * Sottoscrizione avviata durante la navigazione tra le voci di menu non leaf
3  */
4  private menuSubscribe(){
5      this.dialService.flagForHome = false;
6      this.menuListener =
7      this.dialService.dialFrontendBridge.onInvokeEvent.subscribe(
8     (tag: string) => {
9         if (tag === 'Back'){
10             this.loadLevelMenuUp(this.currentParent);
11             this.currentParent = this.currentParent.parent;
12         }else{
13             this.menu.planItems.forEach(item => {
14                 if (tag === item.containerId){
15                     if (item.isLeaf){
16                         this.navigateToLeaf(item);
17                         this.menuListener.unsubscribe();
18                         this.componentSubscribe();
19                         this.currentParent = item;
20                     }
21                     else{
22                         this.loadLevelMenuDown(item);
23                         this.currentParent = item;
24                     }
25                 }
26             });
27         }
28     }
29 );
30 }

```

Codice 6.12: Metodo menuSubscribe

6.1.7 Widget

Un widget Aulos è un componente in esecuzione all'interno di una DashBoard con lo scopo di rappresentare o modificare i dati ottenuti dal backend collegato fisicamente ad una macchina.

Nella nostra applicazione abbiamo modificato degli Widgets già esistenti, utilizzando all'interno di essi il nostro servizio *DialService*, allo scopo di renderli controllabili attraverso il Dial. Siamo partiti tentando di agganciarsi ad un semplice Widget che visualizza un range di possibili valori rappresentato da uno *slider grafico*, con l'intento di variare, anche solo graficamente, il valore corrente attraverso l'evento rotazione del Dial.

```
1 if (this.widget.runMode) {  
2   if (this.dialService.isDialConnected) {  
3     this.createMenuVoice();  
4     this.dialService.dialFrontendBridge.onRotationEvent.subscribe(  
5       ({tag, degree}) => {  
6         if (tag === this.id) {  
7           this.value += (parseInt(degree, 10) * 5);  
8           this.changeDetectorRef.detectChanges();  
9         }  
10      });  
11    }  
12  }
```

Codice 6.13: Sottoscrizione evento rotazione Radial Gauge

La stessa cosa è stata fatta per un widget rappresentato graficamente da un *Radial Gauge*, fino ad arrivare a modificare valori lato Backend attraverso il *DialChannelWidget*, un widget creato da noi che contiene al suo interno, al contrario degli widget già presenti, il collegamento a *N* diversi canali modificabili. Questo ci ha permesso di sviluppare delle funzionalità custom per quel determinato Widget al fine di poterne acquisire il controllo, e scegliere canale per canale, il valore da settare, permettendo la variazione non solo grafica ma anche effettiva dei dati mostrati.

Per rendere sicura queste funzionalità abbiamo previsto la variazione dei valori con la singola rotazione e la successiva modifica lato backend qualora il dial venga premuto, altrimenti attraverso la funzionalità combinata di pressione e rotazione, così da evitare una involontaria modifica nel momento della presa possesso.

Analogamente all'evento sopra mostrato, chiamato *onRotationEvent*, gli eventi al quale il widget *ChannelDialWidget* si sottoscrive al fine di variare i valori di un canale selezionato, sia graficamente che in modo permanente lato backend, sono i seguenti:

- **onRotationEvent**
- **onClickEvent**
- **onPressedRotationEvent**

Durante la sua inizializzazione, il widget si sottoscrive all'evento di *Invoke*, con lo scopo di intercettare il tag sul quale viene richiamato il medesimo metodo dal Dial. Successivamente, se l'invoke richiamato restituisce il tag corrispondente all'ID del Widget in ascolto, viene pulito il menu del dial e vengono caricate le voci di menu inerenti ai

canali collegati a quel determinato Widget così da poter essere richiamati e intercettati a loro volta dalla sottoscrizione all'evento `onInvokeEvent`.

6.2 Applicazione UWP

La seconda parte, invece, è costituita da un' applicazione UWP ovvero Universal Windows Platform, presa in considerazione poiché la più recente tra i framework Microsoft e tra le sue caratteristiche troviamo la sicurezza e la portabilità tra le diverse piattaforme windows, ciò consente un maggiore supporto nel tempo e intercambiabilità tra i dispositivi dove utilizzarla.

L'applicazione `DialBridge` ha come responsabilità quella di comunicare con il Dispositivo HID attraverso le librerie fornite da Microsoft e allo stesso tempo comunicare con la pagina Web contenuta al suo interno.

Inizialmente abbiamo creato un'unica soluzione contenente sia la Web View che il Core dell'applicazione, ma abbiamo riscontrato un problema nell'injection dell'oggetto utilizzato per la comunicazione Web-UWP, poiché esso deve necessariamente essere iniettato a runtime.

Di conseguenza abbiamo creato due diverse soluzioni:

- **Core:** Contiene il nucleo dell'applicazione.
- **EmbeddedBrowserView:** Contiene la parte runtime del progetto e effettua l'injection dell'oggetto nella pagina Web.

6.2.1 Core

Nel Core è contenuta la parte del progetto UWP che non richiede la partenza a runtime e comprende le seguenti classi:

- `AulosCollection`
- `AulosIcon`
- `DialController`
- `WebNotifier`
- `DialMenuItem`

6.2.2 AulosCollection e AulosIcon

Queste due classi si occupano, insieme al main, della parte riguardante le icone che verranno poi visualizzate nel menu del Dial. All'avvio dell'applicazione nella classe main viene effettuata una chiamata GET ad un eventuale server Loccioni che consente di ricevere un file JSON contenente una mappa "nome"-`FontGlyph` che verrà poi utilizzato per creare un `DialMenuItem` insieme al Font Aulos creato da Loccioni.

L'utilizzo di questo approccio consente una maggiore facilità nella sostituzione delle icone, che non dipende dalle risorse dell'applicazione stessa ma da un server Loccioni.

6.2.3 DialController

La classe `DialController` è una tra le più importanti poiché una sua istanza verrà poi inserita all'interno della `WebView` e servirà al servizio `DialService` per la comunicazione Web-UWP. I metodi pubblici al suo interno sono gli stessi dell'interfaccia `DialBackendBridge` che vengono utilizzati appunto tramite l'oggetto iniettato inizialmente.

Al suo interno troviamo un'istanza della classe `RadialController`, ovvero la libreria Microsoft che permette la manipolazione del Dial, un'istanza di `RadialControllerConfiguration` e infine un `notifier`.

Nel costruttore abbiamo assegnato agli eventi presenti nella classe `RadialController` il relativo handler nella classe `WebNotifier`, in questo modo ogni volta che verrà emesso un determinato evento la pagina web riceverà una notifica e verrà svolta la funzione associata a quella determinata voce di menù.

6.2.4 WebNotifier

La classe `WebNotifier` contiene al suo interno una istanza della `WebView` ed ha come responsabilità quella di fornire i metodi che verranno poi utilizzati come `EventHandler` per gli eventi forniti dalla classe `RadialController`, in modo tale da riuscire a comunicare, attraverso la `WebView` al suo interno, alla pagina web l'emissione dell'evento stesso.

Al suo interno è presente anche un oggetto della classe `CoreWindow` utilizzato per spostare la posizione del mouse sullo schermo al primo contatto del Dial sulla superficie.

6.2.5 DialMenuItem

La classe `DialMenuItem` è una classe “modello” utilizzata per rappresentare un elemento contenuto nel Menu del Dial.

6.2.6 MainPage

Nella Main Page avviene l'inizializzazione dei componenti `RadialController` e la navigazione alla pagina web desiderata.

All'interno del metodo `NavigationStarting` troviamo il metodo `AddWebAllowedObject` che ci permette, una volta caricata la pagina, di iniettare l'oggetto della classe `DialController` per la comunicazione.

7. Posizionamento su schermo

Uno dei maggiori vantaggi dell'utilizzo di un surface Dial è la possibilità di posizionarlo sopra uno schermo compatibile e di cambiare il suo funzionamento in base alla posizione utilizzata. Questa funzionalità è gestita dal sistema operativo e comunica solamente con elementi nativi UWP, considerando ciò, abbiamo cercato un servizio Angular che ci permettesse di svolgere la stessa funzione all'interno di una pagina web ma senza risultati. Inizialmente abbiamo preso in considerazione la possibilità di suddividere il



Figura 7.1: Microsoft Dial posizionato sopra un surface pro 6

layout dell'applicazione nativa in N riquadri, all'interno dei quali avremo posizionato il relativo Widget da controllare. Questa soluzione presenta però delle limitazioni, in quanto in base alla risoluzione del dispositivo utilizzato, i widget caricabili in ogni riquadro del layout erano limitati al numero di riquadri definiti dall'applicazione UWP.

Abbiamo quindi cercato una soluzione, che spostasse la responsabilità di acquisire il posizionamento lato Web anziché lato UWP. Dopo vari tentativi, abbiamo escogitato una soluzione che ci permettesse di avere un numero variabile di Widget configurabili nella Dashboard e la successiva acquisizione di essi, per fare ciò abbiamo sfruttato Analizzando gli eventi richiamabili dalla classe RadialController, in particolare quelli inerenti al contatto con lo schermo, abbiamo notato prendevano in ingresso un oggetto di tipo RadialControllerScreenContact fornitogli dalla Libreria. Questo oggetto possiede due attributi relativi al Posizionamento (Position) e al Bound (Bounds) del rettangolo generato dal Dial sullo schermo.

Grazie all'attributo Position di tipo Point, é possibile ottenere, attraverso le coordinate X e Y, il punto centrale nel quale il Dial é posizionato.

```
1 internal async Task ScreenContactStartedAsync(  
2     RadialControllerScreenContactStartedEventArgs args  
3 )  
4 {  
5     this.simpleHaptics = args.SimpleHapticsController;  
6     string x = args.Contact.Position.X.ToString();  
7     string y = args.Contact.Position.Y.ToString();  
8     window.PointerPosition = new Point(  
9         args.Contact.Position.X,  
10        args.Contact.Position.Y  
11    );  
12     string function =  
13         "window.DialFrontendBridge.ScreenContactStartedEvent('" + x + "', '" + y + "')";  
14     await webView.InvokeScriptAsync("eval", new string[] { function });  
15     onScreen = true;  
16     inputInjector.InjectMouseInput(new[] { inputInfo });  
17     inputInfo.MouseOptions = InjectedInputMouseOptions.Move;  
18     inputInfo.DeltaY = 1;  
19 }
```

Codice 7.1: Metodo ScreenContactStartedAsync

Attraverso la libreria Window.UI.Core é possibile spostare il cursore in un punto definito della UI corrente. Per questo motivo, la classe WebNotifier presente un attributo private e readonly chiamato window, un oggetto di tipo WindowCore che permette di intercettare eventi relativi ai dispositivi di Input utilizzati ma anche di posizionare il cursore del dispositivo in una posizione specifica in base alle coordinate passategli. In questo modo, attraverso l'attributo PointerPosition siamo stato in grado di creare un nuovo oggetto di tipo Point con le coordinate X e Y acquisite dal Dial consentendoci di posizionare il cursore in quel determinato punto.

```
1 window.PointerPosition = new Point(args.Contact.Position.X, args.Contact.Position.Y);
```

Codice 7.2: Spostamento cursore

Questa soluzione ci ha permesso di spostare dinamicamente il cursore nella posizione centrale al disotto del Dial quando viene posizionato sopra lo schermo così da poter intercettare lato Web la posizione del cursore e permettere agli utilizzatori del servizio di riprodurre il comportamento appropriato per il Widget sopra il quale ci si posiziona. Abbiamo quindi previsto nel Template HTML del Widget un metodo OnMouseOver che venisse richiamato solamente qualora il Dial fosse effettivamente a contatto con lo schermo, ma il semplice spostamento del cursore nella nuova posizione non richiamava correttamente il metodo in quanto il cursore non effettuava un movimento, ma veniva ricreato in quella determinata posizione. Affinché la soluzione ideata funzionasse correttamente abbiamo importato una libreria chiamata Windows.UI.Input.Preview.Injection che mette a disposizione funzionalità per la simulazione di eventi di input.

Una volta posizionato il cursore nella nuova posizione, siamo stati in grado di simulare un impercettibile movimento del cursore di un singolo pixel, grazie al quale ci é stato possibile intercettare correttamente l'evento `OnMouseMove` dichiarato nel Template del Widget eseguendo il metodo associato.

```
1 inputInjector.InjectMouseInput(new[] { inputInfo });  
2 inputInfo.MouseOptions = InjectedInputMouseOptions.Move;  
3 inputInfo.DeltaY = 1;
```

Codice 7.3: Spostamento cursore

8. Evoluzione

Lorem ipsum dolor sit amet, consectetur adipisci elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullamco laboriosam, nisi ut aliquid ex ea commodi consequatur. Duis aute irure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

8.1 Sezione Esempio

8.2 Sezione Esempio

9. Utilizzo Del Servizio

Per utilizzare il servizio DialService, occorre importarlo nel costruttore del componente che si sta sviluppando, in questo modo avremo accesso agli eventi che ci consentono di avere notifiche dal Dial. Inizialmente è necessario creare una voce di Menù che rappresenta il widget all'interno del menù del Dial, così da avere la possibilità di acquisirne il controllo ed avere accesso alle sue funzionalità. Sotto è riportato un esempio di creazione di una voce di menù per il dial con la relativa aggiunta alla lista di voci di Menu già presenti.

```
1 private createMenuVoice() {  
2     const dialMenuVoice = this.dialService.dialProxy.createDialMenuItem(  
3         this.widget.id, this.widget.descriptor.shortText, this.widget.descriptor.icon);  
4     this.dialService.dialProxy.addItem(dialMenuVoice);  
5 }
```

Codice 9.1: Creazione nuova voce da widget

Una volta aggiunta la voce di Menu al Dial, essa potrà essere selezionata attraverso il metodo Invoke messo a disposizione dalla classe DialController. Il metodo Invoke per acquisire una voce di menu, potrà essere richiamato sia attraverso il menu contestuale del Dial che compare nello schermo, che tramite un nostro metodo chiamato appunto ManualInvoke che esegue questa azione manualmente, richiamando tramite Web il medesimo metodo, solamente se quella voce di menu é presente tra quelle disponibile nel Dial. In questo modo, il Widget non dovrà fare altro che mettersi in ascolto del metodo onInvokeEvent, il quale quando richiamato, restituisce una stringa definita “tag”, rappresentante la voce Menu acquisita. Se quel tag, corrisponde all'id del Widget in ascolto, significa che l'utente ha richiesto il controllo di quel determinato Widget.

```
1 this.dialService.dialFrontendBridge.onInvokeEvent.subscribe(tag => {  
2     if (tag === this.widget.id) {  
3         this.isActive = true;  
4         this.dialService.widget = this.widget.id;  
5         this.widgetDialStart();  
6         this.dialService.dialProxy.manualInvoke(this.channels[0].parameter.value);  
7     }
```

Codice 9.2: Ascolto Invoke della voce di menu'

Una volta acquisito il controllo del Widget, bisognerà mettersi in ascolto delle funzioni richiamabili dal Dial, attraverso gli eventEmitter messi a disposizione dalla classe DialFrontendBridge, e implementare il comportamento di quel Widget all'emissione di determinati eventi come la Rotazione o il Click.

```
1 const rotation = this.dialService.dialFrontendBridge.onRotationEvent.subscribe(  
2   ({ tag, degree }) => {  
3     // Azione associata all'evento rotazione del Dial.  
4     if (tag === this.channels[this.currentChannel].parameter.value) {  
5       this.setSetterAtIndex(this.currentChannel,  
6         parseNumber(degree) * this.multiplicator  
7       );  
8     }  
9   });  
10  
11 const click = this.dialService.dialFrontendBridge.onClickEvent.subscribe(  
12   (tag) => {  
13     // Azione associata all'evento click del Dial.  
14     if (tag === this.channels[this.currentChannel].parameter.value) {  
15       this.setChannelValue(this.channels[this.currentChannel].parameter.code,  
16         this.getNumberSetterComponent(this.currentChannel).value);  
17     }  
18   });  
19  
20 const pressRot = this.dialService.dialFrontendBridge.onPressedRotationEvent.subscribe(  
21   ({ tag, degree }) => {  
22     // Azione associata all'evento rotazione con pressione del Dial.  
23     if (tag === this.channels[this.currentChannel].parameter.value) {  
24       this.setSetterAtIndex(  
25         this.currentChannel,  
26         parseNumber(degree) * this.multiplicator  
27       );  
28       this.setChannelValue(this.channels[this.currentChannel].parameter.code,  
29         this.getNumberSetterComponent(this.currentChannel).value);  
30     }  
31   });  
32   this.channelDialSubscriptions?.unsubscribe();  
33   this.channelDialSubscriptions = new Subscription();  
34   this.channelDialSubscriptions.add(rotation).add(click).add(pressRot);
```

Codice 9.3: Ascolto eventi associati alla voce di menu' selezionata

I metodo sopra implementati, permettono di eseguire una determinata azione quando viene richiamato un metodo presente in DialFrontendBridge come la rotazione o la pressione combinata alla rotazione, che permette quindi nel caso dell'acquisizione tramite il contatto del Dial sullo schermo, mantenendo il controllo e il relativo comportamento, anche appoggiandolo sulla scrivania, evitano il pericolo che nel tratto in cui é il dispositivo si trova sospeso, possano avvenire chiamate pericolose.

10. Testing

Per quanto riguarda i test svolti in ambito UWP abbiamo utilizzato MSTest fornito da Microsoft all'interno di Visual Studio per testare le funzionalità messe a disposizione dalla classe RadialController.

10.1 MSTest

Visual Studio Unit Testing Framework descrive la suite di strumenti di unit test di Microsoft integrata in alcune versioni di Visual Studio 2005 e successive. Qui sotto sono riportati due tra i vari test implementati nel progetto:

```
1 [UITestMethod]
2 public void TestSetMenu()
3 {
4     Assert.IsFalse(radialController.Menu.Items.Count > 0);
5     DialMenuItem item1 = dialController.CreateDialMenuItem("tag1", "d", "icon");
6     DialMenuItem item2 = dialController.CreateDialMenuItem("tag2", "d", "icon");
7     DialMenuItem item3 = dialController.CreateDialMenuItem("tag3", "d", "icon");
8     DialMenuItem item4 = dialController.CreateDialMenuItem("tag4", "d", "icon");
9     List<DialMenuItem> items = new List<DialMenuItem>();
10    items.Add(item1);
11    items.Add(item2);
12    items.Add(item3);
13    items.Add(item4);
14    dialController.SetMenu(items);
15    Assert.IsTrue(radialController.Menu.Items.Count == 4);
16 }
```

Codice 10.1: Test SetMenu

```
1 [UITestMethod]
2 public void TestAddItem() {
3     Assert.IsFalse(radialController.Menu.Items.Count > 0);
4     dialController.AddItem(dialController.CreateDialMenuItem("tag", "d", "icon"));
5     Assert.IsTrue(radialController.Menu.Items.Count > 0);
6 }
```

Codice 10.2: Test aggiunta voce di Menu'

11. Conclusioni

Durante lo studio di fattibilità e il successivo sviluppo, abbiamo constatato che le potenzialità del Dial in ambito industriale sono molteplici, dal miglioramento della User Experience alla precisione adottata nell'utilizzo. Grazie al DialService è stato possibile trasferirle anche in ambito web, in quanto all'interno dell'impresa Loccioni sono presenti moltissimi settori in cui il Dial migliorerebbe molto l'esperienza utente e porterebbe una diminuzione di tempi e costi.

Al momento dello sviluppo, la necessità di dover passare per un applicazione UWP rappresenta sicuramente una limitazione che andrà tolta qualora il progetto verrà rilasciato in maniera industriale. A favore di questa critica, nel mese passato, la Google stessa ha rilasciato in fase di Preview una libreria Web chiamata Trial for WebHID, la quale permetterà di far comunicare direttamente con la pagina Web i dispositivi che utilizzano il protocollo HID, come il Dial, permettendo agli sviluppatori futuri di non dover necessariamente passare attraverso una WebView o una applicazione nativa per quel dispositivo.



Figura 11.1: Potenzziometro utilizzato attualmente

Nel nostro caso, l'utilizzo finale del Dial nei banchi prova del gruppo Loccioni, comporterebbe una maggior sicurezza in termini di precisione nella variazione di dati, in quanto al momento vengono utilizzati dei semplici potenziometri con le evidenti limitazioni che essi comportano, come ad esempio la presenza di un inizio e fine corsa del potenziometro stesso, l'assenza di feedback durante l'utilizzo, la limitata corsa e di conseguenza la necessaria approssimazione dei singoli step eseguiti.

Queste problematiche verrebbero risolte attraverso il Dial, in quanto presenta una "corsa" illimitata, un feedback atpico personalizzabile in base alle operazioni che si svolgono, una precisione laser a 3600 punti e la possibilità di interagire direttamente con



Figura 11.2: Potenzimetro utilizzato attualmente

lo schermo, appoggiando il Dial su di esso, permettendo quindi una serie di combinazioni di utilizzo estremamente vasta.

12. Sviluppi Futuri

I possibili sviluppi sono molteplici e un'idea iniziale è quella di inserire un nuovo widget all'interno della dashboard Loccioni in grado di raffigurare attraverso un'immagine una parte del motore o elemento da controllare, in modo tale da avere un'interfaccia più intuitiva per la selezione, e poi, una volta selezionato il widget stesso, avere la possibilità di visualizzare i dettagli del componente attraverso il menù del Dial.

Una seconda modifica da apportare agli widget esistenti è sicuramente quella del controllo del moltiplicatore da applicare ad ogni rotazione avvenuta con il Dial, portando così maggiore precisione nei lavori più complessi e anche rapidità di utilizzo nelle parti che necessitano più rotazioni di Dial.

Uno sviluppo futuro sicuramente necessario è l'aggiunta di un controllo sui dati in input lato web, che momentaneamente non è presente nei nostri widget, ma è fatta solamente lato backend.

Per aumentare l'utilizzabilità dell'interfaccia utente è sicuramente necessaria una modifica degli widget da noi creati, per renderli più facilmente selezionabili dal Dial e per migliorare anche l'aspetto visivo. In caso poi il progetto arrivi al punto di essere utilizzato, per migliorare la manipolazione di dati in uso nelle dashboard attuali è necessario sicuramente apportare alcune modifiche alla dashboard per consentire un utilizzo attraverso il touchscreen di un qualsiasi dispositivo, in modo tale da rendere veramente fluido l'utilizzo di tutto l'ambiente e poter così eliminare il mouse dalle operazioni svolte più frequentemente.

Per concludere, un possibile miglioramento da apportare al codice è quello di una selezione più veloce ed accurata del widget attraverso il posizionamento sullo schermo, magari attraverso un movimento anticipato del mouse o un movimento continuo.

Ringraziamenti