

**Università degli Studi di Camerino**

---

**SCUOLA DI SCIENZE E TECNOLOGIE**

**Corso di Laurea in Informatica (Classe L-31)**



## **Project: Multiuser Configurable DataView Web Dashboard**

Candidati

**Tommaso Carletti**

Matricola 100889

**Luca Cervioni**

Matricola 100993

**Dmitry Mingazov**

Matricola 101884

**Lucia Passeri**

Matricola 100824

Relatore

**Prof.ssa Emanuela Merelli**

Corelatore

**Luca Mazzuferi**



# Indice

<b>1</b>	<b>Architettura legacy</b>	<b>9</b>
1.1	Struttura del widget . . . . .	9
1.1.1	Widget Component . . . . .	9
1.1.2	Widget . . . . .	10
1.1.3	Widget Service . . . . .	10
1.1.4	Widget Module . . . . .	11
1.2	Struttura backend . . . . .	12



# Elenco dei codici

1.1	orders.component.ts . . . . .	9
1.2	booking-view-widget.ts . . . . .	10
1.3	booking-view-widget.service.ts . . . . .	10
1.4	booking-view-widget.module.ts . . . . .	11



## Elenco delle figure





# 1. Introduzione

L'obiettivo di questo documento consiste nell'esposizione del lavoro svolto durante il periodo di Stage++ in Loccioni. Le competenze apprese spaziano dal know-how che caratterizza l'impresa e i suoi collaboratori, le capacità organizzative che derivano dal lavorare in un team e, naturalmente, nelle tecnologie utilizzate e l'architettura sviluppata. Stilare report settimanali ci ha consentito di tracciare con precisione i progressi del nostro progetto, oltre che a poterci confrontare con esperti per poter garantire uno sviluppo regolare e senza errori di sorta. Una caratteristica che spicca nel far parte di questo ambiente è quella della collaborazione orizzontale, dove noi studenti abbiamo percepito l'importanza attribuitaci: tutto ciò ha comportato un'esperienza di valore, dove le conoscenze apprese negli studi sono state sfruttate ed integrate completamente. Questo ci ha consentito peraltro di avere un rapporto diretto con sviluppatori altamente qualificati, altrimenti impossibile in altre realtà dove una divisione del lavoro verticale implicherebbe una minore integrazione dei tirocinanti. Poter prender parte allo stage anche nel periodo in cui l'emergenza sanitaria dovuta al COVID-19 costringeva al lavoro da remoto è stato estremamente valido a livello formativo. Sebbene alcune attività fossero impossibili da attuare, l'esperienza di apprendimento non si è mai fermata, garantendo in pieno la continuità del tirocinio. Quello che la Loccioni pone al centro della sua value proposition si può riassumere in una frase riportata nel sito dell'impresa stessa: "Trasformiamo i dati in valore" [Loccioni]. La sua principale attività è quella di progettare e realizzare sistemi di misura per testare e controllare componenti di auto, lavatrici ed altri strumenti elettronici. Queste mansioni sono svolte in "linee di test"; ogni "linea" è strutturata in più "banchi" che a loro volta sono composti da più "stazioni", all'interno delle quali avviene il test effettivo del pezzo, o del DuT (device under test).

I risultati dei test sono poi visualizzabili tramite widget all'interno di dashboard remote, prendendo i dati da un server centrale, o collocate in prossimità della macchina.

## 1.1 Obiettivo

Il proposito del progetto è stato quello di ideare e realizzare una sovrastruttura per il framework AULOS, che verrà approfondito nel capitolo ??, per agevolare e rendere più efficiente lo sviluppo dei widget. Punto focale per ottenere questo risultato è stato effettuare uno studio delle caratteristiche che accomunavano più widget tramite un processo di graduale generalizzazione. Quello che ne risulta è la creazione stessa dei widget in questione, con caratteristiche di flessibilità ed adattabilità: sarà infatti facilitato il ritrovamento dei dati dello stesso formato provenienti da fonti diverse.

Una lista di widget realizzati per lo studio sopracitato è presente nel capitolo ??.



## 2. Tecnologie e strumenti

Di seguito verranno dettagliate le tecnologie e gli strumenti impiegati nello sviluppo del progetto.

### 2.1 Aulos



Figura 2.1: Logo di AULOS

AULOS è il framework utilizzato all'interno dell'impresa Loccioni per standardizzare lo sviluppo e rendere riconoscibile ed uniforme lo stile grafico di applicativi per la visualizzazione dati. La parte frontend utilizza la tecnologia Angular, mentre la parte backend può essere scritta utilizzando linguaggi come C# e LabView.

Un componente fondamentale sul quale si è basato il lavoro svolto per il progetto è la dashboard, nata dall'esigenza di monitorare in tempo reale i dati provenienti dalle macchine Loccioni. Offre un ambiente di authoring che permette ai clienti di creare ad hoc in maniera semplice e veloce il proprio pannello personalizzabile [**AULOS**]. La dashboard è composta da:

- l'elenco di tutti i possibili widget selezionabili dall'utente.
- la griglia centrale in cui l'utente può selezionare, spostare, ridimensionare o eliminare un widget.
- il pannello di configurazione, che si attiva selezionando un widget e permette all'utente di andare a modificare le impostazioni legate ad esso.

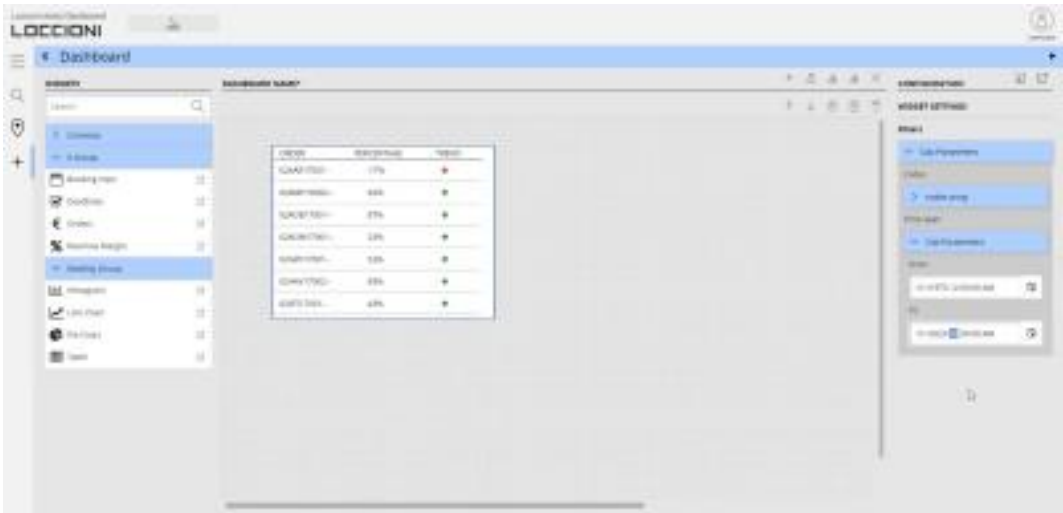


Figura 2.2: Dashboard del framework AULOS

Il widget, attore principale all'interno della dashboard, è una miniapplicazione che permette all'utente di visualizzare dati e svolgere semplici azioni. Nel progetto discusso ne sono stati sviluppati diversi relativi ai dati generati dalle macchine ed ai flussi gestionali interni all'impresa.

Order Code	Customer	Date	Amount	Margin Amount
03AOR13003--/V15	223913	28/06/2017	11600	58.62
02ARU15001--/V1	226789	29/06/2015	325	26.15
02HRC14001--	228574	13/06/2014	215000	39.53
02DSPHC14041	235212	05/06/2014	1320	38.64
02DSPHC14040	226789	27/05/2014	325	26.15
02DSPHC14037	233907	08/05/2014	264	40
08WUF14002--	234754	30/04/2014	1805	19.67
02DSPHC14035	233791	29/04/2014	440	38.64
02DSPHC14036	229758	29/04/2014	5970	51.59
02URD14001--	231133	28/04/2014	330	42.42
02DSPHC14034	228809	22/04/2014	3870.2	41.85
02DSPHC14032	228497	11/04/2014	596	32.21
02DSPHC14033	228883	11/04/2014	600	49.33
02SRV14006--	228883	10/04/2014	39500	81.01
02DSPHC14030	229758	07/04/2014	773.5	25.27
02DSPHC14031	232716	04/04/2014	3708	49.6
02DSPHC14029	231915	03/04/2014	5675	36.19
02SRV14003--	228499	28/03/2014	70000	87.86
02SRV14004--	228497	26/02/2014	88740	86.48

Figura 2.3: Widget realizzato per la visualizzazione degli ordini

**Storyboard** AULOS offre inoltre una pagina di storyboard, contenente molteplici storie, per aiutare gli sviluppatori nel suo utilizzo. Una storia si occupa di descrivere tutti gli stati di rendering interessanti di un componente UI e di fornirne una spiegazione implementativa [STORYBOOK].

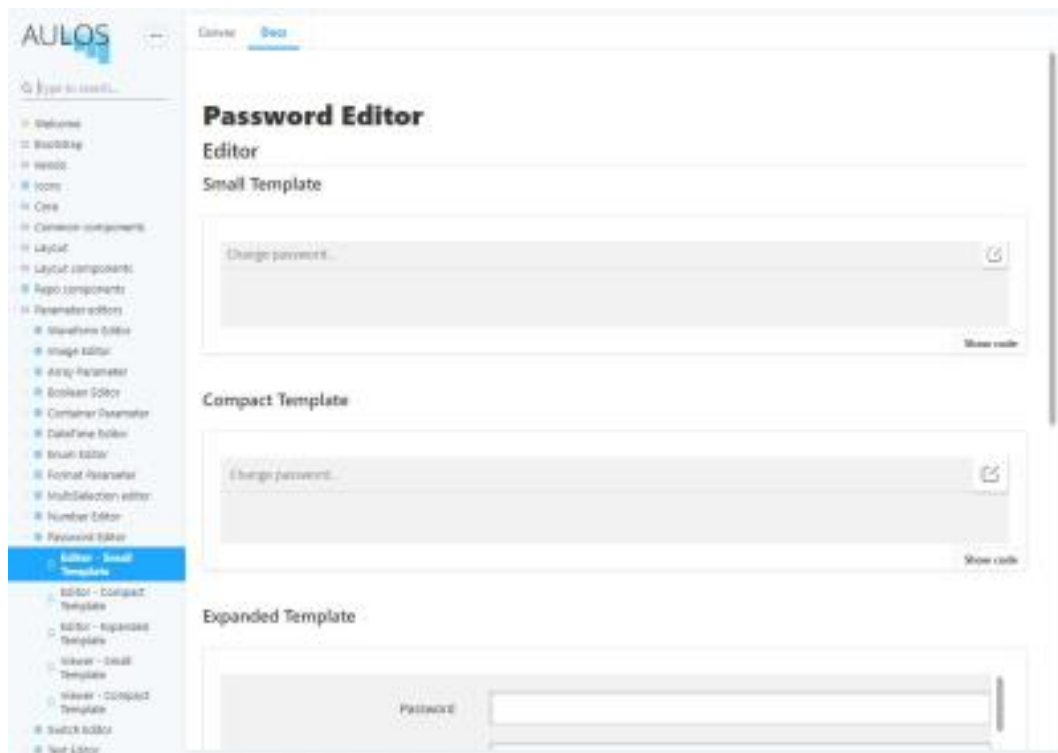


Figura 2.4: Vista sullo storybook del framework AULOS

## 2.2 Angular 10



Figura 2.5: Logo Angular

L'applicativo frontend del progetto discusso nella tesi è stato sviluppato con Angular. Angular è una piattaforma e un framework per costruire applicazioni client single-page usando i linguaggi HTML e TypeScript [Ang].

**HTML** HTML (Hypertext Markup Language) è il linguaggio markup standard per i documenti visualizzabili in un browser web [Htm].

**TypeScript** TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft. Estende la sintassi di JavaScript, essendo un suo superset, 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. [Typ] L'architettura di un'applicazione Angular si basa su alcuni concetti fondamentali. Gli elementi costitutivi di base sono gli NgModules, che forniscono un contesto di compilazione per i Component. Questi ultimi sono elementi fondamentali per la costruzione delle UI in Angular. Ciascun Component definisce una classe che contiene dati e logica dell'applicazione ed è associato ad un modello HTML che definisce come verrà visualizzato nell'ambiente di destinazione. Il decoratore `@Component()` identifica la classe immediatamente sotto di essa come un componente e fornisce il modello e i relativi metadati specifici del componente. I decoratori sono funzioni che modificano le classi JavaScript. Angular definisce un numero di decoratori che associano specifici tipi di metadati alle classi, in modo che il sistema sappia cosa significano quelle classi e come dovrebbero funzionare.[Ang]

```
1 import { Component, OnInit } from '@angular/core';
2 @Component({
3   selector: 'app-root',
4   templateUrl: './app.component.html'
5 })
6 export class AppComponent implements OnInit {
7
8   constructor() { }
9
10  ngOnInit() { }
11
12 }
```

Codice 2.1: Esempio di un Component in Angular

Per i dati o la logica che non sono associati a una vista specifica e che si desidera condividere tra i componenti, si crea una classe di servizio. Una definizione di classe di servizio è immediatamente preceduta dal decoratore `@Injectable()`. Il decoratore fornisce i metadati che consentono ad altri provider di essere inseriti come dipendenze nella tua classe. La dependency injection (DI) consente di mantenere le classi dei componenti snelle ed efficienti. Non recuperano dati dal server, convalidano l'input dell'utente o accedono direttamente alla console; delegano tali compiti ai servizi. [Ang]

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4 import { Data } from '...';
5
6 @Injectable({
7   providedIn: 'root',
8 })
9 export class AppService {
10
11   constructor( private http: HttpClient, private url: string ) { }
12
13   public getData(): Observable<Data[]> {
14     return this.http.get<Data[]>(this.url);
15   }
16
17 }
```

Codice 2.2: Esempio di un Service in Angular

Per la gestione interna dei pacchetti in Angular è stato utilizzato NPM (Node Package Manager). NPM è un gestore di pacchetti per il linguaggio di programmazione JavaScript. È il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js. Consiste in un client da linea di comando, chiamato anch'esso npm, e un

database online di pacchetti pubblici e privati, chiamato npm registry. Il registry è accessibile via client e i pacchetti disponibili sono consultabili sul sito web di npm. [Npm]

## 2.3 RxJs



Figura 2.6: Logo RxJs

RxJS (Reactive Extensions for JavaScript) è una libreria per la programmazione reattiva <sup>1</sup> che utilizza Observable che semplifica la composizione di codice asincrono o basato su callback. RxJS fornisce un'implementazione di Observable e funzioni di utilità al fine di lavorarci [RxJS]. Queste ultime possono essere utilizzate per:

- Conversione del codice esistente per operazioni asincrone in Observable
- Iterare sui valori di uno stream
- Mappare valori di diversi tipi
- Filtrare stream
- Effettuare la composizione di più stream

## 2.4 KendoUI



Figura 2.7: Logo Kendo UI

Kendo UI è un framework integrale di interfaccia utente HTML5 per costruire applicazioni e siti web interattivi e ad alte prestazioni. [Ken] Il framework è stato usato all'interno dell'applicativo frontend con l'obiettivo di conferire un aspetto grafico più accattivante ai widget. Kendo UI per Angular offre un vasto numero di componenti:

<sup>1</sup>La programmazione reattiva è un paradigma di programmazione asincrona che riguarda i flussi di dati e la propagazione del cambiamento.

CHARTS	BUTTONS	DATE INPUTS	DROPDOWNS	INPUTS	LAYOUT	
Area Charts	Button	Calendar	AutoComplete	CheckBox	Avatar	
Bar Charts	ButtonGroup	DateInput	ComboBox	ColorPicker	Card	
Box Plots	Chip	DatePicker	DropDownList	FormField	ContextMenu	
Bubble Charts	ChipList	DateRange	MultiSelect	MaskedTextBox	Drawer	
Bullet Charts	DropDownButton	MultiViewCalendar	GAUGES	NumericTextBox	Menu	
Donut Charts	SplitButton	TimePicker		Arc Gauge	RadioButton	PanelBar
Funnel Charts	COMMON FEATURES	DIALOGS		Slider	Popup	
Line Charts		Linear Gauge	RangeSlider	Ripple		
Pie Charts		Radial Gauge	Switch	ScrollView		
Polar Charts	Excel Export	Window	GRIDS	TextArea	Sortable	
Radar Charts	PDF Export	LABELS		TextBox	Splitter	
Range Bar Charts	File Saver	Label	UPLOAD	NOTIFICATIONS	Stepper	
Scatter Plots	CONVERSATIONAL UI	FloatingLabel			Upload	Notification
Waterfall		Chat	SCHEDULER	FileSelect	PAGER	Tooltip
Sparkline	Scheduler		ICONS			ToolBar
Stock Chart	TREELIST	INDICATORS	Icon	Pager	TreeView	
TreeList			SVGIcon	PROGRESS BARS	NAVIGATION	
EDITOR	LISTVIEW					
Editor	Listview					
		Badge		ChunkProgress	AppBar	
		Loader		ProgressBar	Breadcrumb	

Figura 2.8: Componenti di Kendo UI per Angular

Ogni componente include un set completo di funzionalità già pronte all'uso [Ken]. Nel caso della Kendo Grid, componente frequentemente utilizzato nel progetto per rappresentare dei dati all'interno di una tabella, vi è ad esempio la possibilità di filtrare, ordinare, modificare e raggruppare i dati agevolmente.

The screenshot shows a Kendo UI Grid with the following features:

- Search:** A search bar at the top left with the placeholder text "Search in all columns..."
- Export:** Two buttons at the top right: "Export to PDF" and "Export to Excel".
- Grouping:** A message "Drag a column header and drop it here to group by that column" above the grid headers.
- Grid Columns:**
  - Employee Group:** Contains "Contact Name", "Job Title", "Country", and "Status".
  - Performance Group:** Contains "Rating" and "Engagement".
- Data Rows:**
  - Sig Jeannel: Human Resources Assistant III, USA, Online, 4 stars, 100% engagement.
  - Shelden Greyes: Operator, UK, Online, 5 stars, 40% engagement.
  - Megen Cody: Operator, Brazil, Online, 3 stars, 66% engagement.
  - Clevey Thursfield: VP Quality Control, Brazil, Online, 4 stars, 58% engagement.
  - Ruthi Baldini: Data Coordinator, Brazil, Online, 4 stars, 37% engagement.
- Pagination:** A footer bar showing "1 - 20 of 100 items" and a set of navigation buttons.

Figura 2.9: Esempio di Kendo Grid per Angular



## 2.5 .NET



Figura 2.10: Logo di .NET

.NET è una piattaforma di sviluppo ideata da Microsoft per costruire differenti tipi di applicazioni, utilizzando il C#, F# o Visual Basic.[Micc]

- **C#** è un linguaggio di programmazione object-oriented semplice, moderno e fortemente tipizzato.
- **F#** è un linguaggio di programmazione funzionale cross-platform e open-source. Include anche tratti di programmazione imperativa e object-oriented.
- **Visual Basic** è un linguaggio di programmazione intuitivo con una sintassi semplice volto a scrivere app object-oriented e fortemente tipizzate.

Viene divisa in due versioni principali:

- **.NET Core** è la versione cross-platform, e viene utilizzata per lo sviluppo di siti, servizi e console apps. È questa la versione che è stata utilizzata nello sviluppo dell'applicativo backend.
- **.NET Framework** è la versione solo per Windows, e viene utilizzata per lo sviluppo di ogni tipo di app che può essere eseguita di Windows

### 2.5.1 ASP.NET



Figura 2.11: ASP.NET

ASP.NET è un framework di sviluppo open-source per web applications lato server. È stato sviluppato Microsoft per permettere ai programmatori di costruire pagine web dinamiche, applicazioni e servizi.[Micd]  
Analogamente a .NET, presenta due versioni:

- **ASP.NET Core** è la versione cross-platform che utilizza il runtime di .NET Core.
- **ASP.NET Core 4.x** è la versione solo per Windows, utilizza il runtime di .NET Framework

### 2.5.2 Entity Framework



Figura 2.12: Entity Framework

Entity Framework è un ORM framework (Object-relational mapping) per .NET supportato da Microsoft. Permette agli sviluppatori di lavorare con i dati usando oggetti specifici del dominio senza concentrarsi sulle tabelle e colonne dei database in cui sono memorizzati i dati.

### 2.5.3 Entity Framework Core

Entity Framework Core è una versione leggera, estensibile, open source e cross-platform di Entity Framework. È stato utilizzato per lavorare con i dati del progetto memorizzati in server SQL e MySQL.

## 2.6 Linq



Figura 2.13: Logo Nest

LINQ (Language-Integrated Query) è il nome di un set di tecnologie basate sull'integrazione delle funzionalità di query direttamente nel linguaggio C#. In genere, le query sui dati vengono espresse come stringhe semplici senza il controllo dei tipi in fase di compilazione o il supporto IntelliSense. È anche necessario imparare un linguaggio di query diverso per ogni tipo di origine dati: database SQL, documenti XML, svariati servizi Web e così via. Con LINQ, una query è un costrutto del linguaggio di prima classe, come le classi, i metodi e gli eventi. È possibile scrivere query su insiemi di oggetti fortemente tipizzati usando le parole chiave del linguaggio e gli operatori comuni. La famiglia di tecnologie LINQ offre coerenza per l'esecuzione di query per oggetti (LINQ to Objects), database relazionali (LINQ to SQL) e XML (LINQ to XML). Per uno sviluppatore che scrive query, la parte integrata nel linguaggio più visibile di LINQ è l'espressione di query. Le espressioni di query vengono scritte con una sintassi di query dichiarativa. Tramite la sintassi di query è possibile eseguire operazioni di filtro, ordinamento e raggruppamento sulle origini dati usando una quantità minima di codice [Lin].

## 2.7 ElasticSearch



Figura 2.14: Logo ElasticSearch

Elasticsearch è un motore di ricerca e analisi full-text open source altamente scalabile. Consente di archiviare, cercare e analizzare grandi volumi di dati rapidamente e quasi in tempo reale. Viene generalmente utilizzato come motore sottostante per applicazioni che hanno caratteristiche e requisiti di ricerca complessi. Elasticsearch dispone di diversi casi d'uso [Ela]:

- consente di avere una visione più ampia delle informazioni a disposizione senza risultare dispersivo, anche con agglomerati di dati estremamente consistenti grazie alle "aggregazioni".

- combina diversi tipi di ricerche: strutturate, non strutturate, geografiche, ricerca di applicazioni, analisi della sicurezza, metriche e registrazione.
- si adatta a qualsiasi situazione che va dal semplice computer con un singolo nodo fino ad arrivare ad un cluster con centinaia di server, rendendo la creazione di un prototipo più agevole.
- utilizza API RESTful standard e JSON. L'apporto costante della comunità ha portato alla creazione e al mantenimento di client in molti linguaggi come Java, Python, .NET, SQL, Perl, PHP.
- è possibile utilizzare le funzionalità di ricerca e analisi in tempo reale di Elasticsearch per lavorare sui big data utilizzando il connettore Elasticsearch-Hadoop (ES-Hadoop).

Data l'elevata efficienza di questo software diverse compagnie informatiche l'hanno scelto per i loro prodotti. Gli ambiti interessati sono quelli di ricerca per quanto concerne Tinder, dove Elasticsearch permette di rendere il sistema più reattivo e veloce, mentre nel caso di Netflix l'utilizzo principale è focalizzato al giusto indirizzamento di notifiche push, email e messaggistica [Ela]. Nel progetto discusso, Elasticsearch è stato utilizzato nell'implementazione del layer di accesso dei dati relativi alle macchine che testano le componenti degli elettrodomestici Whirlpool. In sinergia con questo strumento è stata impiegata la dashboard di visualizzazione dati Kibana.

## 2.8 Nest



Figura 2.15: Logo Nest

NEST è un client Elasticsearch .NET di alto livello che è ancora molto fedele all'API Elasticsearch originale. Tutte le richieste e le risposte sono esposte attraverso i tipi, rendendolo ideale per essere subito operativi. Tutti i metodi disponibili all'interno di NEST sono esposti sia come versioni sincrone che asincrone, con quest'ultima che utilizza il suffisso idiomatrico `Async` sul nome del metodo [Nes]. Di seguito la medesima richiesta effettuata in Postman tramite un json ?? ed in Nest ??.

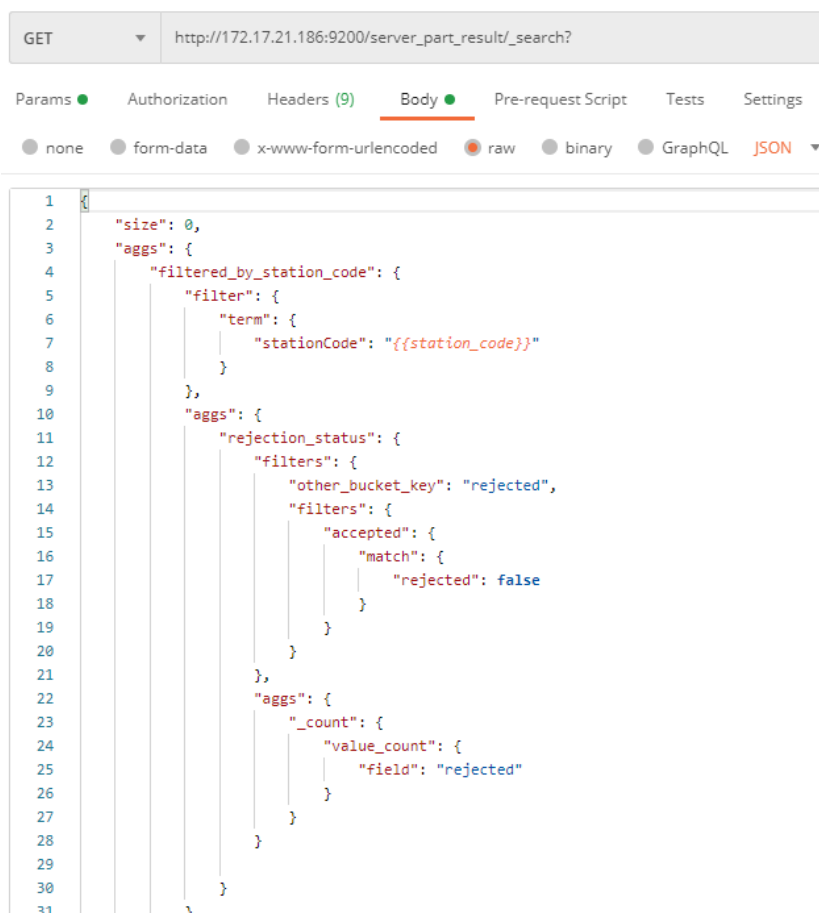


Figura 2.16: Logo Nest

```

public async Task<PieChartData> GetStationRefusedByCode(long stationCode)
{
    var result = await this._context.Client.SearchAsync<PartResult>(x => x
        .Index(_context.Settings.PartResultIndex).Take(0)
        .Aggregations(agg => agg
            .Filter("filtered_by_code", f => f.Filter(f => f.Term(t => t.StationCode, stationCode))
                .Aggregations(agg => agg
                    .Filters("station_by_rejected", f => f
                        .OtherBucket().OtherBucketKey("passed")
                        .NamedFilters(f => f
                            .Filter("rejected", fil => fil.Term(b => b.Rejected, true))
                        ).Aggregations(agg => agg.ValueCount("_count", v => v.Field(f => f.Id))))
                    )
                )
            ));
    var resData = new PieChartData
    {
        Refused = (long)result.Aggregations.Filter("filtered_by_code")
            .Filters("station_by_rejected").Filter("rejected").ValueCount("_count").Value,
        Passed = (long)result.Aggregations.Filter("filtered_by_code")
            .Filters("station_by_rejected").Filter("passed").ValueCount("_count").Value
    };
    return resData;
}
  
```

Figura 2.17: Logo Nest

## 2.9 Newtonsoft JSON



Figura 2.18: Newtonsoft JSON.NET

JSON.NET è un framework open source di gestione JSON per framework .NET che mette a disposizione funzionalità di serializzazione e deserializzazione di oggetti .NET, manipolazione di JSON utilizzando sintassi LINQ (Sec. ??), query su JSON utilizzando sintassi pseudo-XPath [**XPATH**], elevata performance relativamente alle librerie di serializzazione incluse in .NET e conversione XML-JSON.<sup>2</sup>

## 2.10 Git



Figura 2.19: Logo Git

Git è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando. [Gita] Un software di controllo versione è un'applicazione che permette di tenere traccia di tutti gli aggiornamenti apportati ad uno specifico progetto software, o "repository". Oltre a mostrare una visione più generale e strutturata sul progetto, Git consente agli sviluppatori di implementare codice in team nella maniera più efficiente possibile cercando di ridurre i conflitti che possono nascere da uno sviluppo parallelo. Questo viene reso possibile grazie ad un sistema di branches, letteralmente rami, che permettono allo sviluppatore di lavorare in autonomia in uno spazio completamente isolato dal resto del progetto. Per riunificare poi il nuovo codice implementato con il branch principale, lo sviluppatore potrà effettuare un'operazione di merge.

---

<sup>2</sup>NEWTONSOFT.

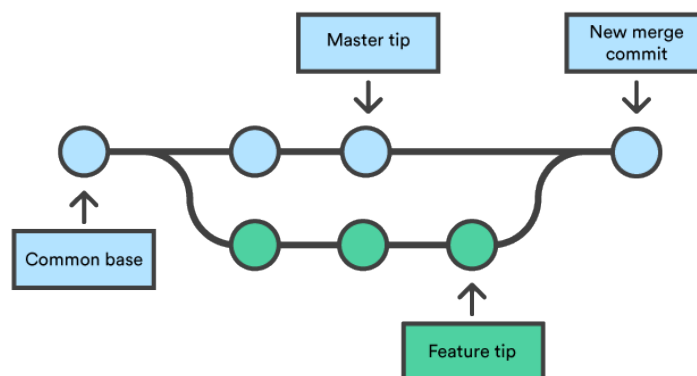


Figura 2.20: Esempio di merge di due branch in Git

## 2.11 Gitlab



Figura 2.21: Logo GitLab

GitLab è una piattaforma web open source che permette la gestione di repository Git e di funzioni trouble ticket [Gitb]. La piattaforma si basa sull'offrire uno spazio virtuale remoto, pubblico o privato, dove archiviare il proprio repository, permettendo agli sviluppatori di lavorare in team. Inoltre GitLab consente di avere un controllo più immediato mettendo a disposizione dello sviluppatore una vista sui branch creati e una history delle modifiche apportate al repository. Ulteriore funzionalità è quella dei trouble ticket, ovvero le "issues" all'interno di GitLab. Questo sistema consente di creare dei ticket per richiedere agli sviluppatori di aggiungere nuove funzionalità o di risolvere uno specifico bug. Una volta soddisfatta la richiesta il ticket potrà essere chiuso e disattivato dallo sviluppatore. Il progetto discusso è stato archiviato su GitLab in un dominio privato Loccioni.

IT > unicom-stage-plus-plus-2020 > AulosDashboard > Repository

develop aulosdashboard / + ▾ History Find file Web IDE Clone ▾

Merge branch 'develop' of...  
Dmitry Mingazov authored 3 days ago 619e39c3

Name	Last commit	Last update
.vscode	[UPDATE] - several libraries with newer versions	1 month ago
e2e	Angular update #1	2 months ago
src	Merge branch 'develop' of https://git.loccioni.com/IT/unicam-stag...	3 days ago
tasks	Angular update #1	2 months ago
.browserslistrc	Angular update #1	2 months ago
.editorconfig	First commit	3 months ago
.gitignore	First commit	3 months ago
.npmrc	First commit	3 months ago
README.md	[UPDATE] - Readme changes	1 week ago
angular.json	[ADD] - Revenue margin widget parametrization	1 month ago

Figura 2.22: Vista su una delle due repository caricate su GitLab

## 2.12 Visual Studio



Figura 2.23: Logo Visual Studio

Nello sviluppo dell'applicativo backend è stato fatto uso di Visual Studio: un IDE estendibile e con funzionalità complete per la creazione di applicazioni moderne per Android, iOS e Windows, nonché di applicazioni Web e servizi cloud [202]. Gli strumenti utilizzati sono:

- **Sviluppo ASP.NET e Web** Viene offerta la possibilità di creare applicazioni Web usando ASP.NET Core, ASP.NET (.NET Framework), HTML/JavaScript e i contenitori. Esso è volto a massimizzare la produttività sviluppando applicazioni Web .NET con ASP.NET Core e tecnologie basate su standard quali HTML e JavaScript. Questo componente trova applicazione in diversi campi:
  - Sito Web con pagine Razor in ASP.NET Core
  - API Web con ASP.NET Core MVC
  - App Web in tempo reale con ASP.NET Core SignalR



Alcuni dei componenti messi a disposizione sono IntelliSense (per il completamento del codice), esplorazione del codice e refactoring per C#, Visual Basic e F# [202].

- **Sviluppo per desktop .NET** Viene offerta la possibilità di creare applicazioni Web con .NET Framework, applicazioni client per computer o dispositivi da rendere disponibili tramite Microsoft Store, WPF, Windows Forms e console con C#, Visual Basic e F#. Questo componente trova applicazione in diversi campi quali: Windows Presentation Foundation (WPF) e Windows Forms. I componenti messi a disposizione sono gli strumenti per:


- Strumenti per sviluppo di applicazioni desktop .NET
- Strumenti di sviluppo per .NET Framework 4.x
- Strumenti di profilatura .NET
- Supporto per i linguaggi C# e Visual Basic
- Strumenti di Entity Framework 6
- IntelliTrace
- Debugger JIT
- Live Unit Testing
- Live Share



## 2.13 Visual Studio Code



Figura 2.24: Logo Visual Studio Code

Nello sviluppo dell'applicativo frontend è stato fatto uso di Visual Studio Code: è un editor di codice sorgente leggero ma potente che viene eseguito sul desktop ed è disponibile per Windows, macOS e Linux. Viene fornito con il supporto integrato per JavaScript, TypeScript e Node.js e ha un ricco ecosistema di estensioni per altri linguaggi (come C ++, C#, Java, Python, PHP, Go) e runtime (come .NET e Unity) [Micg]. Vi è la possibilità di installare delle estensioni e quelle impiegate nel progetto sono:

-  **Bracket Pair Colorizer 2:** Questa estensione consente di identificare le parentesi corrispondenti con i colori. L'utente può definire quali token abbinare e quali colori utilizzare.

-  **GitLens — Git supercharged:** Permette, per ciascuna parte di codice, di visualizzare da chi è stato scritto tramite le annotazioni Git blame e la lente del codice, è possibile navigare ed esplorare senza problemi i repository Git.
-  **Live Share:** Consente di modificare in modo collaborativo ed eseguire il debug con altri collaboratori in tempo reale. Gli sviluppatori che partecipano alle tue sessioni ricevono tutto il contesto dell'editor dal tuo ambiente, il che garantisce che possano iniziare immediatamente a collaborare in modo produttivo, senza la necessità di clonare alcun repository o installare SDK.
- **Markdown Preview Enhanced:** Estensione che fornisce molte funzionalità utili come la sincronizzazione automatica dello scorrimento, la composizione matematica, la sirena, PlantUML, pandoc, esportazione PDF, blocco di codice, scrittore di presentazioni, ecc. Molte delle sue idee sono ispirate da Markdown Preview Plus e RStudio Markdown .
- **Material Icon Theme:** Rende possibile il cambiamento del colore dell'icona della cartella predefinita e modificare il design delle icone delle cartelle, utilizzando la tavolozza dei comandi:

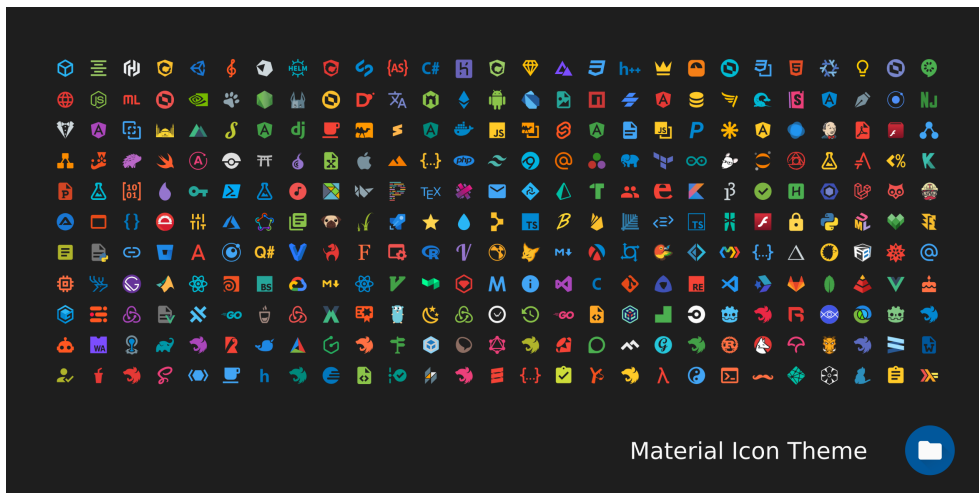


Figura 2.25: Material Icon Theme

- **Package Json Upgrade:** Mostra gli aggiornamenti disponibili in package.json. Offre azioni rapide per guidare nell'aggiornamento.

## 2.14 HeidiSQL



Figura 2.26: Logo HeidiSQL

HeidiSQL è un software gratuito ed ha l'obiettivo di essere di facile comprensione, consente di visualizzare e modificare dati e strutture da computer che eseguono uno dei sistemi di database MariaDB, MySQL, Microsoft SQL, PostgreSQL e SQLite. Appartiene agli strumenti più popolari per MariaDB e MySQL in tutto il mondo [Bec]. Tale strumento è stato utilizzato per la comunicazione con il server, al fine di manipolare le informazioni nei database che il server gestisce.

## 2.15 Microsoft SQL Server Management Studio



Figura 2.27: Logo MicrosoftSQL Server

SQL Server Management Studio (SSMS) è un ambiente gratuito ed integrato per la gestione di qualsiasi infrastruttura SQL, da SQL Server al database SQL di Azure. SSMS offre gli strumenti per configurare, monitorare e amministrare le istanze di SQL Server e i database. Usare SSMS per distribuire, monitorare e aggiornare i componenti del livello dati usati dalle applicazioni, nonché per creare query e script. È possibile usare SSMS per eseguire query, progettare e gestire database e data warehouse in qualsiasi posizione, nel computer locale o nel cloud [Mss].

## 2.16 Postman



Figura 2.28: Logo Postman

Postman è uno strumento di sviluppo di API (application programming interface) che aiuta a costruire, testare e modificare API. Permette di fare vari tipi di richieste HTTP (GET, POST, PUT, PATCH), salvare gli ambienti per usi futuri e convertire le API in codice per vari linguaggi (come JavaScript, Python) [Pos]. Nel corso dello sviluppo Postman è stato uno strumento essenziale per testare in modo veloce e immediato le richieste HTTP al backend.

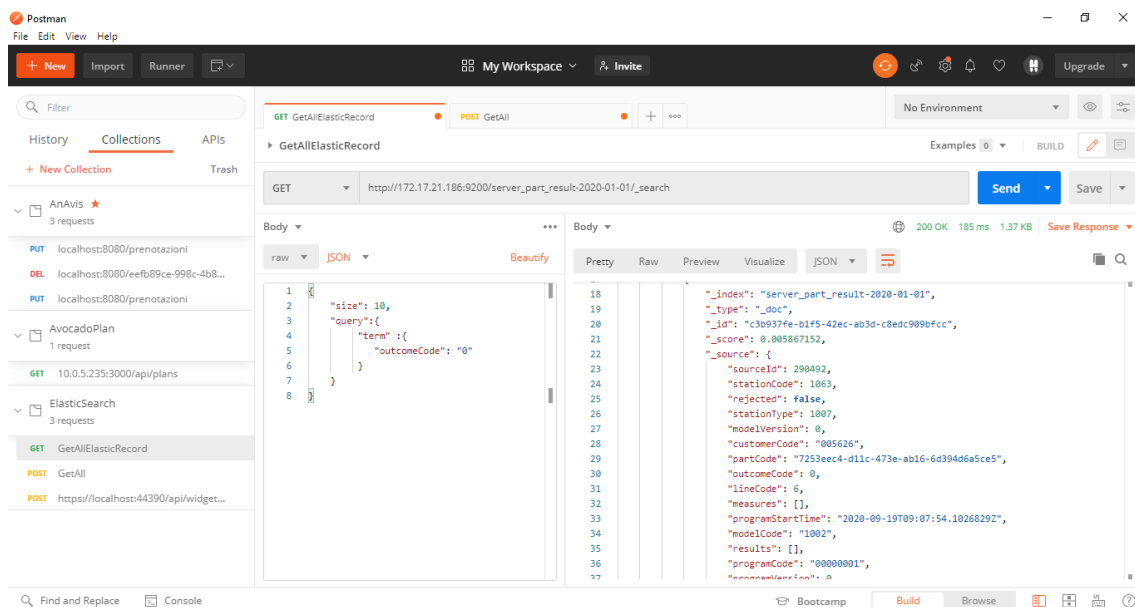


Figura 2.29: Schermata di Postman

## 2.17 JMeter



Figura 2.30: Logo JMeter

Apache JMeter<sup>TM</sup> è un software open source sviluppato interamente in Java e progettato per fare test di carico e misurare le prestazioni. Originariamente era stato pensato solamente per testare le applicazioni Web ma nel tempo sono state aggiunte altre funzionalità di testing. Può essere utilizzato per testare la resistenza di un server, una rete o un oggetto simulando diversi tipi di carichi pesanti per analizzarne le prestazioni complessive. Le principali tipologie di applicazioni/server/protocolli che possono essere testate sono:

- Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...)
- SOAP / REST Webservices
- FTP
- Database via JDBC
- LDAP
- Message-oriented middleware (MOM) via JMS
- Mail - SMTP(S), POP3(S) and IMAP(S)
- Native commands or shell scripts
- TCP
- Java Objects

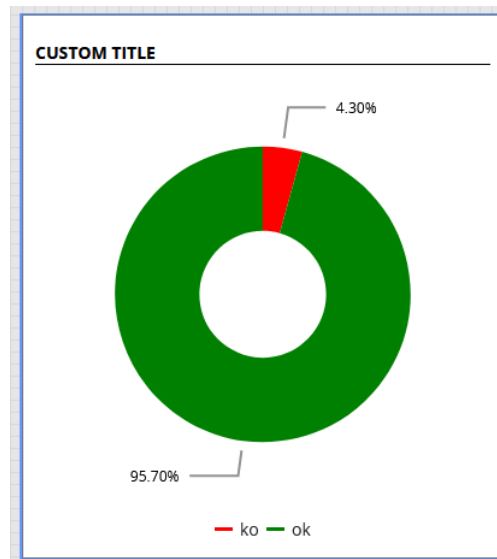
[Fou] Apache JMeter è stato utilizzato per testare l'applicativo backend del progetto.



## 3. Architettura

In questo capitolo verrà esposta la struttura dell'applicativo e le differenze apportate al sistema precedentemente utilizzato, nonché le procedure di scambio dati e configurazione del sistema.

### 3.1 Widget



#### 3.1.1 Composizione

Un widget è una piccola applicazione, semplice e immediata, che mostra all'utente dati e informazioni, di solito sotto forma di grafici o tabelle.

Prendendo ad esempio il pie chart widget si può vedere come un widget lato frontend sia formato da diversi file. La classe contenente i parametri volti alla visualizzazione grafica del widget si chiama `PieChartWidget`. Estende la classe `BaseWidget`, dove viene gestita tutta la logica per la presa dei dati dal backend, ed è decorata con il `WidgetDecorator`, il quale aggiunge ulteriore comportamento al widget.

```
1 @WidgetDecorator({
2     endpoint: 'tuple',
3     type: WidgetTypeEnum.MOBILITY,
4     sourceable: true,
5     adapter: PieChartAdapter,
6     refreshableInfo: {
7         refreshable: true,
8     }
9 })
```

```
9  })
10 export class PieChartWidget extends BaseWidget {
11
12     private titleParameter: StringParameter;
13
14     constructor(injector: Injector, widgetDescriptor: WidgetDescriptor) {
15         super(injector, widgetDescriptor);
16         this.setSize(420, 450);
17     }
18
19     public get title(): string {
20         return this.titleParameter.value;
21     }
22
23     protected createConfigParameters(): Observable<Parameter<unknown>[]> {
24         this.createTitleParameter();
25         return super.createConfigParameters();
26     }
27
28     private createTitleParameter() {
29         this.titleParameter = new StringParameter('title', 'Title', 'My title');
30         this.titleParameter.longText = 'Title Color';
31         this.configParameters.push(this.titleParameter);
32     }
33 }
```

Codice 3.1: Classe PieChartWidget

Tutta la logica riguardante la visualizzazione dei dati del widget si trova nella classe PieChartComponent. Al suo interno viene specificato come la label debba essere mostrata nel pie chart (con il metodo labelContent) e viene implementato il metodo getDataCallback, richiesto dalla classe BaseWidgetComponent che viene estesa da PieChartComponent, per salvarsi i dati ogni qualvolta il widget riceva dati dal backend.

```
1  @Component({
2      selector: 'app-pie-chart-widget',
3      templateUrl: './pie-chart.component.html',
4      encapsulation: ViewEncapsulation.None,
5      changeDetection: ChangeDetectionStrategy.OnPush
6  })
7  export class PieChartComponent extends BaseWidgetComponent<PieChartWidget> {
8
9      public data: TupleData[] = [];
10     public backgroundColor = '#ffffff';
11
12     constructor(widgetActions: WidgetActions, injector: Injector) {
13         super(widgetActions, injector);
14         this.labelContent = this.labelContent.bind(this);
15     }
16
17     public labelContent(args: LegendLabelsContentArgs): string {
18         return `${args.dataItem.value.toFixed(2)}%`;
19     }
20
21     protected getDataCallback(value: any): void {
22         this.data = value;
23     }
24 }
```

Codice 3.2: Classe PieChartComponent

Nel file HTML, di cui viene passato il path al @Component decorator, viene specificato l'aspetto grafico che avrà il widget all'interno della dashboard.

```
1  <aulos-widget-layout [widget]="widget" [backgroundColor]="backgroundColor">
2      <aulos-widget-title>
```



```

3      {{ widget.title }}
4    </aulos-widget-title>
5    <aulos-widget-content>
6      <kendo-chart #customChart [chartArea]="{opacity: 0}">
7        <kendo-chart-legend position="bottom"></kendo-chart-legend>
8        <kendo-chart-series>
9          <kendo-chart-series-item type="donut" [data]="data" field="value"
10             categoryField="label" colorField="color">
11            <kendo-chart-series-item-labels position="outsideEnd" color="#000"
12               background="#FFFFFF" [content]="labelContent">
13              </kendo-chart-series-item-labels>
14            </kendo-chart-series-item>
15          </kendo-chart-series>
16        </kendo-chart>
17      </aulos-widget-content>
18    </aulos-widget-layout>

```

Codice 3.3: File pie-chart.component.html

Infine ogni widget viene incluso in un modulo Angular. All'avvio dell'applicazione il widget viene registrato nel sistema con la chiamata del metodo `registerWidgets`, presente nel modulo. Ciò permette alla dashboard di riconoscere il widget come tale e di mostrarlo selezionabile al momento della configurazione della dashboard.

```

1 private static registerWidgets(widgetsService: WidgetComponentsService): () =>
2   Promise<any> {
3     const result = (): Promise<any> => {
4       return new Promise((resolve, reject) => {
5         const myWidgetDescriptor: WidgetDescriptor = {
6           code: 'pieChartWidgetCode',
7           shortText: 'Pie Chart',
8           longText: 'Descrizione',
9           icon: 'fa fa-pie-chart',
10          group: 'Mobility Group'
11        };
12        widgetsService.register(myWidgetDescriptor, PieChartWidget,
13          PieChartComponent);
14
15        resolve();
16      });
17    };
18    return result;
19  }

```

Codice 3.4: Metodo all'interno del modulo che registra il widget nel sistema

### 3.1.2 Le classi BaseWidget e BaseWidgetComponent

Le classi `BaseWidget` e `BaseWidgetComponent` estendono rispettivamente le classi, già presenti nel framework Aulos, `Widget` e `WidgetComponent`. `BaseWidget` e `BaseWidgetComponent` sono classi nate dall'esigenza del progetto di trovare una soluzione per generalizzare il funzionamento dei widget e di rendere la vita più facile ai futuri sviluppatori. Infatti nelle due classi risiede tutto il comportamento per la raccolta dei dati dal backend, comune a tutti i widget. All'interno del `BaseWidget` viene iniettato il `WidgetService`, servizio in cui avvengono le chiamate REST, e vengono creati i parametri `source` e `filters`.

```

1 export class BaseWidget extends Widget {
2
3   ...
4
5   private createSourceParameter() {
6     this.sourceParameter = new StringParameter('source', 'Source', '');
7     this.sourceParameter.longText = 'Source';

```

```

8      this.sourceParameter.descriptor.editor = new EditorDescriptor(EDITOR_BASE_PATH,
9          'SourceEditor');
10    }
11
12    private createFiltersParameter() {
13        this.filtersParameter = new ContainerParameter('filters', 'Filters', null);
14        this.configParameters.push(this.filtersParameter);
15    }
16
17 }

```

Codice 3.5: Creazione dei parametri all'interno della classe BaseWidget

Il parametro `source` serve per definire la source a cui chiedere dati, mentre il parametro `filter` si occupa di filtrare la richiesta in base ai filtri presi dal backend e gestiti lato frontend. Il tutto viene inizializzato e caricato tramite il metodo `initSourceCalls`.

```

1  export class BaseWidget extends Widget {
2
3      ...
4
5      private async initSourceCalls() {
6          this.filtersParameter.stateChanges.subscribe(_ => {
7              if (this.filtersSet && !this.widgetDTO) {
8                  this.getData();
9              }
10         });
11
12         await this.sourceParameter.valueChanges.subscribe(sourceCode => {
13             this.sourceSet = true;
14             this.widgetService.getSourceProperties(sourceCode).subscribe
15                 (sourceProperties => {
16                 this.metadata = sourceProperties.metadata;
17                 this.setFilters(sourceProperties.filters);
18                 this.notifyStateChanged();
19             });
20         });
21         this.initSource();
22     }
23
24     private getData() {
25         if (!this.sourceParameter.value) { return; }
26         if (!this.isFiltersReady()) { return; }
27         this.widgetService.getData(this.endpoint, this.createRequest()).subscribe
28             (value => {
29             const data = MetadataService.map(value, this.metadata);
30             this.widgetDataSubject.next(data);
31         });
32     }
33
34     private initSource() {
35         this.widgetService.getSource(this.endpoint).subscribe(sourceCode => {
36             this.sourceParameter.value = sourceCode;
37             this.sourceSet = true;
38         });
39     }
40
41 }

```

Codice 3.6: Inizializzazione dei parametri all'interno della classe BaseWidget

Nel `BaseWidgetComponent` vi è la sottoscrizione al `dataObservable`, attribuito `Observable` fornito da `BaseWidget`, la quale si occupa di chiamare il metodo `getDataCallback` ogni qualvolta vengono presi dati dal backend.

```

1  @Directive()
2  export abstract class BaseWidgetComponent<T extends BaseWidget> extends
3      WidgetComponent<T> implements OnInit, OnDestroy{

```

```

4
5
6
7 private initData() {
8     this.widget.dataObservable.subscribe(value => {
9         if (!value)
10             {return;}
11         this.getDataCallback(value);
12         this.loading = false;
13         this.changeDetectorRef.markForCheck();
14     });
15 }
16
17 protected abstract getDataCallback(value: any): void;
18 }

```

Codice 3.7: Metodi per l'inizializzazione dei dati all'interno della classe `BaseWidgetComponent`

### 3.1.3 Il servizio `WidgetService`

Nel `WidgetService` avviene tutta la comunicazione tra frontend e backend. La sua responsabilità è quella di fare chiamate REST per ottenere i dati da far visualizzare all'interno dei widget. Il principale metodo che si occupa di fare ciò è il metodo `getData` in cui, passatagli una `GenericRequest` composta da un `sourceCode` e un array di filtri, manda una richiesta POST al backend.

```

1 @Injectable({
2     providedIn: 'root'
3 })
4 export class WidgetService {
5
6     ...
7
8     public getData(widgetUrl: string, genericRequest: GenericRequest):
9         Observable<unknown[]> {
10         const adapt = Adapters.getAdapter(widgetUrl);
11         const restCall = this.http.post<unknown[]>(`${this.widgetsUrl}/${widgetUrl}`,
12             genericRequest);
13         return adapt ? restCall.pipe(map((data) => adapt(data))) : restCall;
14     }
15
16 }

```

Codice 3.8: Metodo `getData` all'interno della classe `WidgetService`

Se esiste un'implementazione dell'Adapter, interfaccia funzionale con il singolo metodo `adapt`, per quel determinato widget, i dati che verranno presi saranno adattati prima di essere restituiti. In caso contrario verranno passati al widget dati grezzi.

La lista di sources disponibili e la possibile lista di filtri e metadati si ricavano rispettivamente dal metodo `getSources` e dal metodo `getSourceProperties`.

```

1 @Injectable({
2     providedIn: 'root'
3 })
4 export class WidgetService {
5
6     ...
7
8     public getSources(widgetUrl: string): Observable<SourceInfo[]> {
9         return this.http.get<SourceInfo[]>(`${this.apiUrl}/${widgetUrl}/discovery`);
10     }
11
12     public getSourceProperties(sourceCode: string): Observable<SourceProperties> {

```

```
13     return this.http.get<SourceProperties>(`${this.apiUrl}/filters/${sourceCode}`);
14   }
15 }
16 }
```

Codice 3.9: Metodi `getSources` e `getSourceProperties` all'interno della classe `WidgetService`

Se un widget è di tipo `MOBILITY`, ovvero legato a dei dati provenienti dalle macchine Loccioni, avrà bisogno della possibilità di filtrare i dati in base alla linea, banco e stazione. Per questo all'interno del `WidgetService` vi sono 3 metodi predisposti per il recupero di questi ultimi.

```
1 @Injectable({
2   providedIn: 'root'
3 })
4 export class WidgetService {
5   ...
6
7   public getLines(sourceCode: string): Observable<number[]>
8   {
9     return this.http.get<number[]>(`${this.apiUrl}/topology/${sourceCode}
10      /lines`);
11   }
12
13   public getBenches(sourceCode: string, lineCode: number): Observable<number[]>
14   {
15     return this.http.get<number[]>(`${this.apiUrl}/topology/${sourceCode}
16      /benches/${lineCode}`);
17   }
18
19   public getStations(sourceCode: string, stationCode: number): Observable<number[]>
20   {
21     return this.http.get<number[]>(`${this.apiUrl}/topology/${sourceCode}
22      /stations/${stationCode}`);
23   }
24 }
25
26 }
```

Codice 3.10: Metodi per recuperare linee, banchi e stazioni all'interno della classe `WidgetService`

### 3.1.4 Il decorator factory `WidgetDecorator`

Il `WidgetDecorator` ha la responsabilità di aggiungere comportamento a un widget in base alle opzioni passate con l'interfaccia `WidgetOptions`.

`WidgetOptions` ha la seguente struttura:

```
1 export interface WidgetOptions {
2   // indicates which endpoint the widget has to use to take data from
3   endpoint: string;
4   // indicates the widget type (currently IT/MOBILITY)
5   type: WidgetTypeEnum;
6   // indicates whether or not the widget has a selectable source parameter
7   sourceable?: boolean;
8   /* indicates whether or not the widget has a refresh parameter and can be
9    used to set the default refresh rate*/
10  refreshableInfo?: RefreshableInfo;
11  /* indicates the adapter class used to transform received data into a
12   more useful form*/
13  adapter?: any;
14 }
15
16 export interface RefreshableInfo {
```

```

17   refreshable: boolean;
18   defaultRefreshRate?: RefreshRateEnum;
19 }

```

Codice 3.11: Struttura delle WidgetOptions e delle RefreshableInfo

Il comportamento viene aggiunto al widget tramite altri decorators verificando le opzioni passate al momento della decorazione.

```

1  export const WidgetDecorator: (options: WidgetOptions) => (target: Function) =>
2    void = (options: WidgetOptions) => {
3
4    return (target: Function) => {
5      target.prototype.baseWidgetDecorator = true;
6      target.prototype.endpoint = options.endpoint;
7
8      switch (options.type) {
9        case WidgetTypeEnum.IT:
10         break;
11        case WidgetTypeEnum.MOBILITY:
12         Mobility(target);
13         break;
14      }
15      if (options.sourceable) {
16         Sourceable(target);
17      }
18      if (options.refreshableInfo?.refreshable) {
19         Refreshable({defaultRefreshRate: options.refreshableInfo
20           .defaultRefreshRate})(target);
21      }
22
23      if (options.adapter) {
24         const adaptFunc = options.adapter.prototype.adapt;
25         if (adaptFunc) {
26            Adapters.setAdapter(target.prototype.endpoint, adaptFunc);
27         }
28      }
29    };
30 };

```

Codice 3.12: Decorator factory WidgetDecorator

### 3.1.5 Mobility Decorator

Il Mobility decorator si occupa di aggiungere i parametri atti al filtraggio dei dati di un widget in base alla linea, banco e stazione.

```

1  export const Mobility: (target: Function) => void = (target: Function) => {
2
3    ...
4
5    target.prototype.createTopologyParameters = function(...args) {
6      target.prototype.lineCodeParameter = new NumberParameter('lineCode',
7        'Line Code', 0);
8      target.prototype.benchCodeParameter = new NumberParameter('benchCode',
9        'Bench Code', 0);
10     target.prototype.stationCodeParameter = new NumberParameter('stationCode',
11       'Station Code', 0);
12
13     this.lineCodeParameter.descriptor.editor = new EditorDescriptor(
14       EDITOR_BASE_PATH, 'DropdownEditor', []);
15     this.benchCodeParameter.descriptor.editor = new EditorDescriptor(
16       EDITOR_BASE_PATH, 'DropdownEditor', []);
17     this.stationCodeParameter.descriptor.editor = new EditorDescriptor(
18       EDITOR_BASE_PATH, 'DropdownEditor', []);
19
20     this.configParameters.push(this.lineCodeParameter);

```

```
21     this.configParameters.push(this.benchCodeParameter);
22     this.configParameters.push(this.stationCodeParameter);
23   };
24
25   };
```

Codice 3.13: Metodo di creazione dei parametri all'interno del decorator Mobility

Il metodo che inizializza e definisce il comportamento che il widget avrà al cambio di uno dei parametri è il metodo `initTopology`.

```
1  export const Mobility: (target: Function) => void = (target: Function) => {
2
3    ...
4
5    target.prototype.initTopology = function(...args) {
6      if (this.sourceSet) {
7        this.getLines(this.sourceParameter.value, this);
8      }
9      this.sourceParameter.valueChanges.subscribe(sourceCode => {
10         this.getLines(sourceCode, this);
11       });
12
13      this.lineCodeParameter.valueChanges.subscribe(lineCode => {
14         this.resetBench(this);
15         this.getData(this);
16         if (this.sourceParameter.value) {
17           this.widgetService.getBenchs(this.sourceParameter.value, lineCode)
18             .subscribe(benchCodes => {
19               this.benchCodes = benchCodes;
20               this.benchCodeParameter.descriptor.editor.options = benchCodes;
21               this.benchCodeParameter.notifyStateChanged();
22             });
23         }
24       });
25
26      this.benchCodeParameter.valueChanges.subscribe(benchCode => {
27         this.resetStation(this);
28         this.getData(this);
29         if (this.sourceParameter.value) {
30           this.widgetService.getStations(this.sourceParameter.value, benchCode)
31             .subscribe(stationCodes => {
32               this.stationCodes = stationCodes;
33               this.stationCodeParameter.descriptor.editor.options = stationCodes;
34               this.stationCodeParameter.notifyStateChanged();
35             });
36         }
37       });
38
39      this.stationCodeParameter.valueChanges.subscribe(_ => {
40         this.getData(this);
41       });
42     };
43
44   };
```

Codice 3.14: Metodo `initTopology` all'interno del decorator Mobility

### 3.1.6 Sourceable decorator

Il Sourceable decorator dà al widget la possibilità di far selezione all'utente la source dove prendere i dati dal backend.

Ogni widget che estende `BaseWidget` ha al suo interno il parametro `SourceParameter`. Nel `baseWidget` `initSource` si occupa di impostare l'unica source da cui il

widget prenderà i dati, mentre con l'override all'interno del Sourceable decorator il metodo ha lo scopo di impostare la source in base a ciò che selezionerà l'utente.

```

1 export class BaseWidget extends Widget {
2
3   ...
4
5   private initSource() {
6     this.widgetService.getSource(this.endpoint).subscribe(sourceCode => {
7       this.sourceParameter.value = sourceCode;
8       this.sourceSet = true;
9     });
10  }
11
12 }
13
14
15 export const Sourceable = (target: Function) => {
16
17   ...
18
19   target.prototype.initSource = function() {
20     this.sourceParameter.valueChanges.subscribe(sourceCode => {
21       this.sourceParameter.value = sourceCode;
22     });
23   };
24
25 };

```

Codice 3.15: Metodo initSource e override

La configurazione delle possibili source disponibili avviene nell'override del metodo initSourceCalls. Le options per l'editor del sourceParameter vengono impostate con il metodo setSources.

```

1 export const Sourceable = (target: Function) => {
2
3   ...
4
5   const oldInitSourceCalls = target.prototype.initSourceCalls;
6   target.prototype.initSourceCalls = function() {
7     this.widgetService.getSources(target.prototype.endpoint).subscribe(sources => {
8       this.setSources(sources);
9     });
10    if (oldInitSourceCalls){
11      oldInitSourceCalls.apply(this);
12    }
13  };
14
15   target.prototype.setSources = function(sources: SourceInfo[]) {
16     this.sourceParameter.descriptor.editor.options = sources;
17   };
18
19 };

```

Codice 3.16: Metodo setSources e override del metodo initSourceCalls all'interno del Sourceable Decorator

### 3.1.7 Refreshable decorator

Il Refreshable decorator permette al widget di aggiornarsi e ricaricare i dati ad uno specifico intervallo di tempo. Il valore di default dell'intervallo può essere cambiato passando all'interno delle RefreshableInfo anche un defaultRefreshRate. L'intervallo poi potrà essere selezionato dall'utente al momento di configurazione della dashboard grazie al refreshRateParameter che viene aggiunto al widget.

```

1 export const Refreshable: (options?: RefreshableOptions) => (Function) =>
2   void = (options?: RefreshableOptions) => {
3     return (target: Function) => {
4
5       ...
6
7       target.prototype.createRefreshParameter = function() {
8         target.prototype.refreshRateParameter = new NumberParameter('refreshRate',
9           'Refresh Rate', this.defaultRefreshRate);
10        this.refreshRateParameter.descriptor.editor = new EditorDescriptor(
11          EDITOR_BASE_PATH,
12          'EnumEditor',
13          this.generateDefaultEnumOptions());
14        this.configParameters.push(this.refreshRateParameter);
15      };
16
17    };
18  };

```

Codice 3.17: Creazione del refreshRateParameter all'interno del Refreshable decorator

Ogni qualvolta un utente cambi intervallo di refresh viene effettuata una nuova registrazione all'Observable presente nel RefreshService, legato a quel determinato intervallo, e viene annullata la precedente registrazione.

```

1 export const Refreshable: (options?: RefreshableOptions) => (Function) =>
2   void = (options?: RefreshableOptions) => {
3     return (target: Function) => {
4
5       ...
6
7       target.prototype.updateSubscription = function(refreshRate: number) {
8         if (this.subscription) { this.subscription.unsubscribe(); }
9         this.subscription = this.refreshService.getObservableOfInterval(refreshRate)
10          .pipe(
11            exhaustMap(() => { this.getData(); return EMPTY; })
12          )
13          .subscribe();
14      };
15
16    };
17  };

```

Codice 3.18: Metodo updateSubscription all'interno del Refreshable decorator

Il RefreshService ha la responsabilità di tenere salvati gli Observable legati a tutti gli intervalli richiesti.

```

1 @Injectable({
2   providedIn: 'root'
3 })
4 export class RefreshService {
5
6   private intervals: Map<number, Observable<any>> = new Map();
7   constructor() { }
8
9   public getObservableOfInterval(refreshRate: number): Observable<any> {
10     if (!this.intervals.has(refreshRate)) {
11       this.intervals.set(refreshRate, interval(refreshRate).pipe(
12         multicast(new Subject()),
13         refCount()
14       ));
15     }
16     return this.intervals.get(refreshRate);
17   }
18 }

```

Codice 3.19: Classe RefreshService



Nello specifico il metodo `getObservableOfInterval` ritorna l'Observable dell'intervallo di tempo fornito. Se l'intervallo non è presente all'interno di `intervals` allora lo inserisce creando l'Observable legato ad esso.

L'Observable viene creato con il metodo `interval`, il quale crea un Observable che emette numeri sequenziali all'intervallo di tempo specificato. Poi viene trasformato con il metodo `multicast` in un `ConnectableObservable`, particolare tipo di Observable che non emetterà valori finché non sarà chiamato il metodo `connect`. Questa chiamata viene resa automatizzata grazie al metodo `refCount`, il quale gestisce la connessione in modo tale da mantenerla attiva fintanto ci siano observer in ascolto.

```

1 <aulos-editor-layout [editorConfiguration]="editorConfiguration">
2   <ng-template aulosEditorSmallTemplate>
3     <div class="input-group" [class.hidden]="disabled">
4       <input type="text"
5         class="form-control form-control-sm is-invalid"
6         [placeholder]=mySelection[0] ? mySelection[0] :
7         'Choose data source...' | aulosTranslate
8         disabled readonly>
9       <button class="btn btn-sm btn-icon" [disabled]="disabled" (click)="
10         showExpandedModal()">
11         <i class="fontaulos fontaulos-edit"></i>
12       </button>
13     </div>
14   </ng-template>
15   <ng-template aulosEditorCompactTemplate>
16     <div class="input-group" [class.hidden]="disabled">
17       <input
18         type="text"
19         class="form-control is-invalid"
20         [placeholder]="this.model.value ?
21         this.model.value :
22         'Choose data source...' | aulosTranslate"
23         disabled
24         readonly>
25       <button class="btn btn-icon" [disabled]="disabled" (click)="showExpandedModal()">
26         <i class="fontaulos fontaulos-edit"></i>
27       </button>
28     </div>
29   </ng-template>
30   <ng-template aulosEditorExpandedTemplate>
31     <kendo-grid
32       [data]="gridData"
33       [style.maxlength]="501"
34       [filterable]="true"
35       [filter]="state.filter"
36       [selectable]="selectableSettings"
37       [sortable]="true"
38       style="padding-top: 2%; padding-bottom: 5%;"
39       (dataStateChange)="dataStateChange($event)"
40       kendoGridSelectBy="sourceCode"
41       [selectedKeys]="tmpSelection"
42     >
43     <kendo-grid-checkbox-column width="60%"></kendo-grid-checkbox-column>
44     <kendo-grid-column field="sourceCode" filter="text"></kendo-grid-column>
45     <kendo-grid-column field="shortText" filter="text"></kendo-grid-column>
46     <kendo-grid-column field="longText" filter="text"></kendo-grid-column>
47     <ng-template kendoGridNoRecordsTemplate>
48       No sources available.
49     </ng-template>
50   </kendo-grid>
51 </ng-template>
</aulos-editor-layout>

```

Codice 3.20: Classe RefreshService

## 3.2 Security

**Module** L'autenticazione degli utenti all'interno della dashboard è gestita dall' `ItSecurityModule`. Nell'`ItSecurityModule` vengono registrati nell'array `providers`, che gestisce la dependency injection, i service necessari specificando per ognuno di essi la classe astratta a cui fanno riferimento e la classe d'implementazione.

```
1 @NgModule({
2   imports: [HttpClientModule],
3 })
4 export class ItSecurityModule {
5   /**
6    * Main configuration to call on root application module
7    */
8   public static forRoot(): ModuleWithProviders<ItSecurityModule> {
9     return {
10      ngModule: ItSecurityModule,
11      providers: [
12        {
13          provide: AbstractUserFactory,
14          useClass: ItUserFactory
15        },
16        {
17          provide: AbstractTokenProviderService,
18          useClass: ItAccessTokenProviderService,
19        },
20        {
21          provide: ItRefreshTokenProviderService,
22          useClass: ItRefreshTokenProviderService,
23        },
24        {
25          provide: AbstractSecurityService,
26          useClass: ItSecurityService
27        },
28        {
29          provide: HTTP_INTERCEPTORS,
30          useClass: ITHttpSecurityInterceptor,
31          multi: true
32        },
33        {
34          provide: APP_INITIALIZER,
35          useFactory: ItSecurityModule.appInitializer,
36          deps: [AbstractTokenProviderService, ItRefreshTokenProviderService,
37               AbstractSecurityService],
38          multi: true
39        }
40      ]
41    };
42  }
43  ...
44 }
```

Codice 3.21: Injection dei service nell'`ItSecurityModule`

In `providers` vengono fornite le implementazioni delle classi astratte `AbstractSecurityService`, `AbstractUserFactory` e `AbstractTokenProviderService`, parte del framework Aulos. Tutta la logica principale dell'autenticazione risiede all'interno dell'`ItSecurityService`. La sua responsabilità è quella di autenticare e disconnettere l'utente nella dashboard.

```
1 @Injectable({
2   providedIn: 'root'
3 })
4 export class ItSecurityService extends AbstractSecurityService {
5
6   ...
7 }
```

```

7
8 login(username: string, password: string): Observable<ItUser> {
9
10 ...
11
12 return this.httpClient.post<JWTToken>(`${this.serviceConfiguration.baseUrl}/
13 issue/oauth2/token`, { username, password, grant_type: 'password',
14   scope: 'urn:gl-services-infrastructure' }, { headers })
15   .pipe(
16     map(response => {
17       const jwtUser = this.decodeToken(response.access_token);
18       /* We have to set autologin to false to be sure to enable login phase
19        (when default user logout)*/
20       this.serviceConfiguration.autoLogin = false;
21       // Store current token so next call will have correct token
22       this.accessTokenProviderService.store(btoa(JSON.stringify(jwtUser)));
23       this.refreshTokenProviderService.store(response.refresh_token);
24       this.currentJWTUser.next(jwtUser);
25       return jwtUser;
26     }),
27     concatMap(jwtUser => {
28       return this.getUser(jwtUser.loginName)
29         .pipe(map(user => {
30           this.currentUser.next(user);
31           return user;
32         }));
33     }));
34   }
35
36 logout(): Observable<unknown> {
37   this.accessTokenProviderService.remove();
38   this.refreshTokenProviderService.remove();
39   this.currentUser.next(null);
40   return EMPTY;
41 }
42
43 }

```

Codice 3.22: Principali metodi della classe ItSecurityService

**Grant** Si è utilizzato il Resource Owner Password Credentials Grant [IETe] in quanto nel caso d'uso in questione si ha un elevato grado di fiducia tra frontend e backend. Le credenziali vengono mandate all'Authorization Server che restituisce, se le credenziali sono corrette, un access token ed un refresh token.

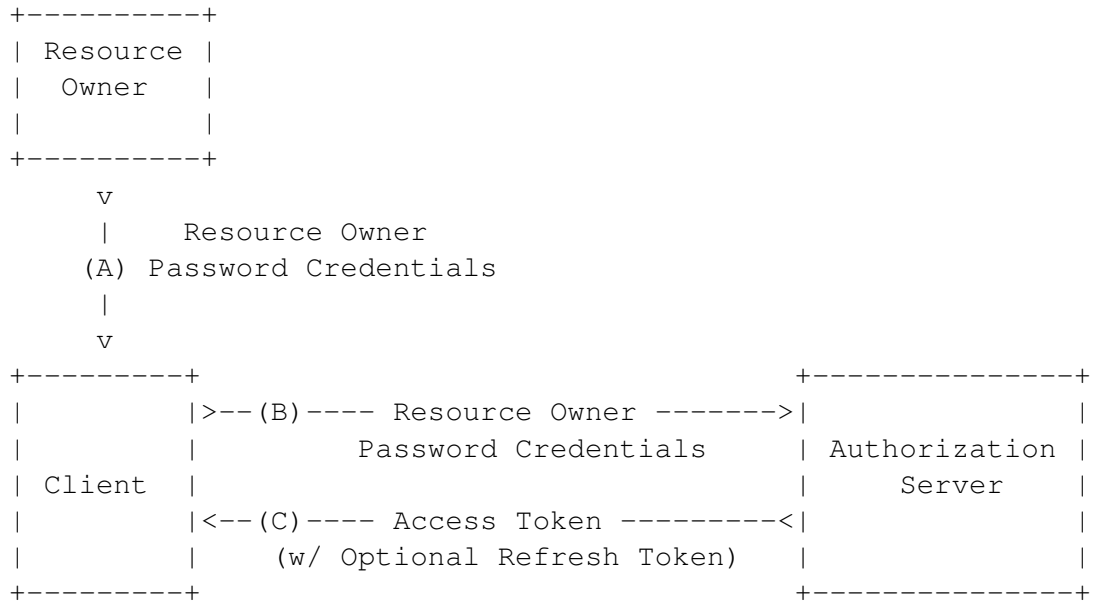


Figura 3.1: Resource Owner Password Credentials Flow [IETe]

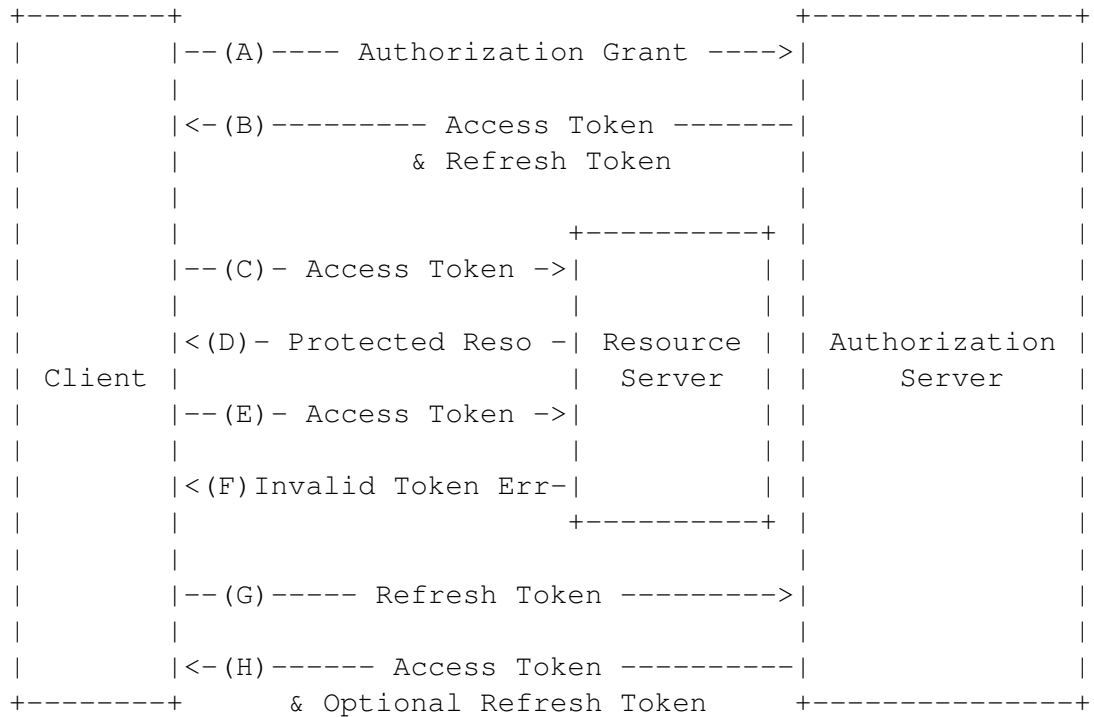


Figura 3.2: Token Refresh Flow [IETd]

Il refresh token viene utilizzato dal Client per richiedere all'Authorization Server un nuovo access token quando quest'ultimo è scaduto. La memorizzazione dell'access e del refresh token viene svolta rispettivamente dall'ItAccessTokenProviderService e dall'ItRefreshTokenProviderService. Entrambe le classi estendono la classe astratta AbstractTokenProviderService.

```

1 export declare abstract class AbstractTokenProviderService {
2   /**
3    * Get token from storage
4    */
5   abstract get(): Observable<string>;
6   /**
7    * Store token on storage
8    */
9   abstract store(token: string): Observable<void>;
10  /**
11   * Remove token on storage
12   */
13  abstract remove(): Observable<void>;
14 }

```

Codice 3.23: Classe astratta AbstractTokenProviderService

All'avvio della dashboard l'ItSecurityModule autentica automaticamente l'utente se l'access token è già presente nel local storage.

```

1 @NgModule({
2   imports: [HttpClientModule],
3 })
4 export class ItSecurityModule {
5
6   ...
7
8   public static appInitializer(
9     accessTokenProviderService: AbstractTokenProviderService,
10    refreshTokenProviderService: ItRefreshTokenProviderService,
11    securityService: AbstractSecurityService
12  ): () => Promise<void> {
13    const result = (): Promise<void> => {
14      return new Promise((resolve, reject) => {
15        // Check if token is already present on browser storage
16        const sources = [accessTokenProviderService.get(),
17          refreshTokenProviderService.get()];
18        forkJoin(sources).subscribe((val) => {
19          const accTok: string = val[0];
20          const refrTok: string = val[1];
21          if (accTok && refrTok) {
22            securityService.loginWithToken((JSON.parse(atob(accTok)) as ItJWTUser)
23              .accessToken).subscribe(() => resolve());
24          } else {
25            resolve();
26          }
27        }, (error) => console.error(error));
28      });
29    };
30    return result;
31  }
32 }

```

Codice 3.24: Login con token nell'ItSecurityModule

**Interceptor** Chi intercetta e gestisce le richieste e le risposte http è l'`ItHttpSecurityInterceptor`. Nello specifico l'`ItHttpSecurityInterceptor` si occupa di:

- decorare le richieste http in uscita con token di autorizzazione, se l'header di autenticazione non è già presente nella richiesta,
- richiedere un access token tramite la procedura di refresh token in caso di risposta 401 a una richiesta al backend,
- ripetere la richiesta fallita con codice 401 dopo aver ottenuto il nuovo access token,
- effettuare il logout in caso di fallimento della procedura di refresh.

### 3.3 Parametri

Un parametro, rappresentato dalla classe `Parameter`, rappresenta un valore con determinate caratteristiche che viene utilizzato in svariati contesti all'interno di AULOS.

```
1 public class Parameter
2 {
3     public Parameter();
4     public string Code { get; set; }
5     public ParameterDescriptor Descriptor { get; set; }
6     public string Value { get; set; }
7     public List<Parameter> SubParameters { get; set; }
8     public string DescriptorCode { get; set; }
9     public StandardParameterType Type { get; set; }
10    public Parameter GetSubParameter(string code);
11 }
```

Codice 3.25: `Parameter.cs`

- Code è l'identificativo univoco del parametro
- Descriptor è il descrittore del parametro, utilizzato per creare il parametro lato frontend
- Value è il valore del parametro
- SubParameters sono i sottoparametri del parametro (un parametro può rappresentare un'aggregazione di altri parametri)
- Type rappresenta il tipo del valore rappresentato dal parametro

La classe `ParameterDescriptor` presenta

```
1 public class ParameterDescriptor
2 {
3     public ParameterDescriptor();
4     public string Code { get; set; }
5     public string ShortText { get; set; }
6     public string LongText { get; set; }
7     public StandardParameterType Type { get; set; }
8     public string ContentType { get; set; }
9     public ParameterEditor Editor { get; set; }
10    public string MeasureUnit { get; set; }
11    public string DefaultValue { get; set; }
12    public string Format { get; set; }
13    public ParameterValidator Validator { get; set; }
14    public List<ParameterDescriptor> SubParameters { get; set; }
```

```

15 public ParameterDescriptor ItemDescriptor { get; set; }
16 public List<string> ItemDescriptorCodes { get; set; }
17 public string WriteEnabledConditionCode { get; set; }
18 public string VariantType { get; set; }
19 }

```

Codice 3.26: ParameterDescriptor.cs

L'attributo Editor, istanza della classe ParameterEditor, descrive l'eventuale Editor utilizzato lato frontend per permettere la modifica del parametro da parte dell'utente

### 3.3.1 Parameter Editor

La classe ParameterEditor è così composta:

```

1 public class ParameterEditor
2 {
3     public ParameterEditor();
4     public string Type { get; set; }
5     public string Path { get; set; }
6     public dynamic Options { get; set; }
7 }

```

Codice 3.27: ParameterEditor.cs

- Type è il codice univoco dell'Editor memorizzato lato frontend
- Path è il path dove AULOS dovrà cercare l'editor
- Options sono le eventuali opzioni che l'editor può avere

### Custom Editors

AULOS fornisce diversi Editor per permettere all'utente di modificare i parametri, ma è possibile crearne di nuovi per casi d'uso più specifici. Un Editor è un Component Angular che va registrato all'interno dell'EditorFactory attraverso il metodo register .

```

1 export class DashboardParameterEditorsModule {
2
3     ...
4
5     public static appInitializer(): () => Promise<void> {
6         const result = (): Promise<void> => {
7             return new Promise((resolve, reject) => {
8
9                 ...
10
11                 EditorFactory.register(
12                     EDITOR_BASE_PATH,
13                     'DropdownEditor',
14                     DropdownEditorComponent);
15                 resolve();
16             });
17         };
18         return result;
19     }
20 }

```

Codice 3.28: dashboard-parameter-editors.module.ts

Il template dell'Editor Component contiene tre direttive strutturali (aulosEditorSmallTemplate, aulosEditorCompactTemplate e aulosEditorExpandedTemplate) le quali contengono il codice html che deve essere visualizzato a seconda dello spazio in cui è contenuto l'Editor.

```

1 <aulos-editor-layout [editorConfiguration]="editorConfiguration">
2   <ng-template aulosEditorSmallTemplate>
3     <div class="input-group" [class.hidden]="disabled">
4       <input type="text"
5         class="form-control form-control-sm is-invalid"
6         [placeholder]="mySelection[0] ?
7         mySelection[0] :
8         'Choose data source...' | aulosTranslate"
9         disabled readonly>
10      <button class="btn btn-sm btn-icon" [disabled]="disabled" (click)="
11        showExpandedModal()">
12        <i class="fontaulos fontaulos-edit"></i>
13      </button>
14    </div>
15  </ng-template>
16  <ng-template aulosEditorCompactTemplate>
17    <div class="input-group" [class.hidden]="disabled">
18      <input
19        type="text"
20        class="form-control is-invalid"
21        [placeholder]="this.model.value ?
22        this.model.value :
23        'Choose data source...' | aulosTranslate"
24        disabled
25        readonly>
26      <button class="btn btn-icon" [disabled]="disabled" (click)="showExpandedModal()">
27        <i class="fontaulos fontaulos-edit"></i>
28      </button>
29    </div>
30  </ng-template>
31  <ng-template aulosEditorExpandedTemplate>
32    <kendo-grid
33      [data]="gridData"
34      [style.maxLength]="501"
35      [filterable]="true"
36      [filter]="state.filter"
37      [selectable]="selectableSettings"
38      [sortable]="true"
39      style="padding-top: 2%; padding-bottom: 5%;"
40      (dataStateChange)="dataStateChange($event)"
41      kendoGridSelectBy="sourceCode"
42      [selectedKeys]="tmpSelection"
43    >
44      <kendo-grid-checkbox-column width="60%"></kendo-grid-checkbox-column>
45      <kendo-grid-column field="sourceCode" filter="text"></kendo-grid-column>
46      <kendo-grid-column field="shortText" filter="text"></kendo-grid-column>
47      <kendo-grid-column field="longText" filter="text"></kendo-grid-column>
48      <ng-template kendoGridNoRecordsTemplate>
49        No sources available.
50      </ng-template>
51    </kendo-grid>
52  </ng-template>
53 </aulos-editor-layout>

```

Codice 3.29: source-editor.component.html

È necessario che l'Editor Component estenda BaseEditorComponent, classe che offre delle funzionalità di interfacciamento con AULOS, e che presenti la logica necessaria alla modifica del parametro a esso associato.



### 3.3.2 Utilizzo dei parametri nell'applicativo

Nell'applicativo, il parametro svolge una duplice funzione:

- permette di gestire la configurazione di un widget lato frontend
- viene utilizzato per specificare dei filtri da applicare in fase di ottenimento dei dati

La classe Widget e le sue estensioni nell'applicazione frontend hanno la responsabilità di gestire i parametri legati ad un widget. Questi parametri possono rappresentare un attributo grafico del widget, un valore della sua configurazione oppure un filtro da applicare ai dati da visualizzare.

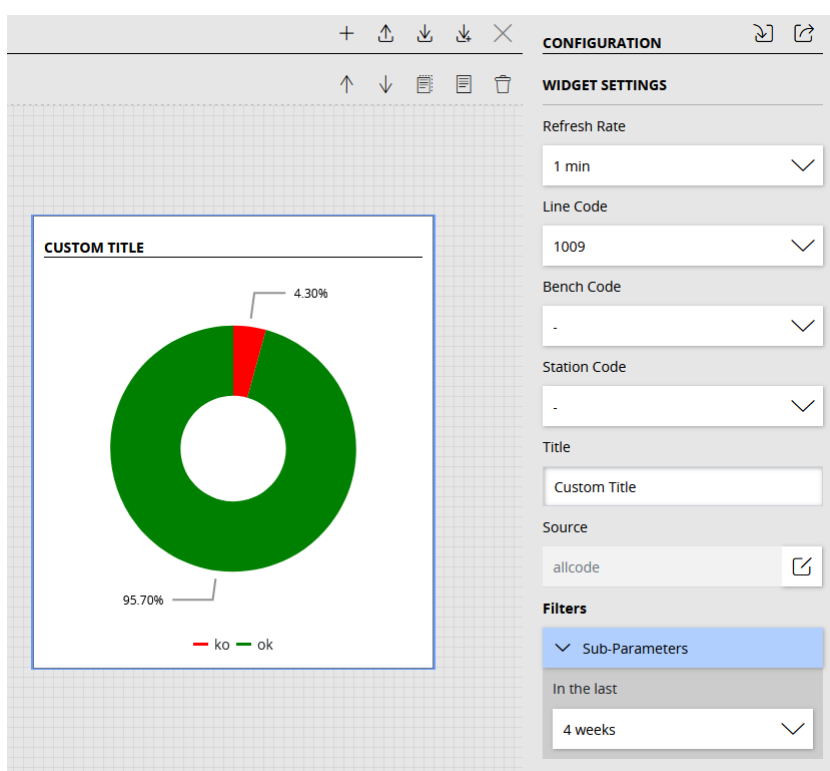


Figura 3.3: Possibile configurazione del widget PieChart

#### Filter Parameters

Data la capacità dei parametri di poter rappresentare dati eterogenei, sono stati utilizzati per rappresentare i possibili filtri di una query.

I Filter Parameters vengono generati attraverso dei ParameterDescriptor, presi con una chiamata REST (( ??)) dopo che è stata definita una source.

```

1 private async initSourceCalls() {
2
3     ...
4
5     await this.sourceParameter.valueChanges.subscribe(sourceCode => {
6         this.sourceSet = true;
    })
  }

```

```

7      this.widgetService.getSourceProperties(sourceCode).subscribe(sourceProperties
      => {
8          this.metadata = sourceProperties.metadata;
9          this.setFilters(sourceProperties.filters);
10         this.notifyStateChanged();
11     });
12 });
13 this.initSource();
14 }

```

Codice 3.30: initSourceCalls, base-widget.ts

## 3.4 Source Management

Una source è un'astrazione che rappresenta una sorgente di dati visualizzabili da un widget. Ogni source viene identificata da una DataSource, una classe che nel pattern Model View Controller si colloca tra il Controller e il Model. La gestione delle source nel backend permette il riutilizzo di uno stesso widget per la visualizzazione di dati provenienti da contesti diversi.

### 3.4.1 Creazione Source

Viene creata una classe DataSource che implementa l'interfaccia relativa al tipo di dato che deve ritornare. L'interfaccia presenterà un metodo volto ad ottenere i dati richiesti (e.g. GetData), che verrà implementato dalla DataSource.

```

1 interface ITupleDataSource
2 {
3     public TupleDTO[] GetData(List<Parameter> parameters);
4 }

```

Codice 3.31: ITupleDataSource.cs

```

1 public class AllRefusedDataSource : ITupleDataSource, ITopologyDataSource
2 {
3     ...
4     public TupleDTO[] GetData(List<Parameter> parameters)
5     {
6         var sam = new SeriousParameterHandler(parameters);
7         var hours = sam.GetValue<long>("from");
8         var now = DateTime.Now;
9         DateTime from = now.AddHours(-hours);
10        if (hours == -1)
11        {
12            from = new DateTime(0);
13        }
14        return sam.Contains("stationCode") ?
15            _repo.GetStationRefusedByCode(sam.GetValue<long>("stationCode"), from)
16            .GetAwaiter().GetResult() :
17            sam.Contains("benchCode") ?
18            _repo.GetBenchRefusedByCode(sam.GetValue<long>("benchCode"), from)
19            .GetAwaiter().GetResult() :
20            sam.Contains("lineCode") ?
21            _repo.GetLineRefusedByCode(sam.GetValue<long>("lineCode"), from)
22            .GetAwaiter().GetResult() :
23            _repo.GetAllRefused(from)
24            .GetAwaiter().GetResult();
25    }
26    ...
27 }

```

Codice 3.32: TupleDataSource.cs

### 3.4.2 Registrazione Source

Le informazioni di una source sono descritte da `SourceInfo` e sono le seguenti:

- `SourceCode` è l'identificativo della source
- `TDataSource` è il tipo della `DataSource` legata alla source
- `Parameters` sono i parametri messi a disposizione dalla source per modificare la query
- `Metadata` sono i metadati volti a modificare come vengono visualizzati i dati lato frontend

`Parameters` e `Metadata` vengono scritti in un file json all'interno della cartella `properties` con il nome `{SourceCode}.json`, come nel seguente esempio:

```

1 {
2   "metadata": {
3     "meta": {
4       "fieldName": "label"
5     },
6     "properties": {
7       "color": [
8         {
9           "field": "Passed",
10          "value": "green"
11        },
12        {
13          "field": "Refused",
14          "value": "red"
15        }
16      ],
17      ...
18    ]
19  }
20 },
21 "filters": [
22   {
23     "Code": "from",
24     "ShortText": "In the last",
25     "LongText": "From",
26     "Type": "number",
27     "DefaultValue": -1,
28     "Editor": {
29       "Type": "EnumEditor",
30       "Path": "@aulos/parameter-editors",
31       "Options": {
32         "comboValues": [
33           {
34             "label": "12 hours",
35             "value": 12
36           },
37           ...
38           {
39             "label": "Ever",
40             "value": -1
41           }
42         ]
43       }
44     }
45   }
46 ]
47 }
```

Codice 3.33: `allcode.json`

Per registrare una source, si crea una classe di configurazione che estende la classe astratta `DiscoveryConfiguration`, e si richiama all'interno del costruttore il metodo `AddSource`, passandogli le informazioni della source che si sta registrando.

```
1 public class TupleDiscoveryConfiguration : DiscoveryConfiguration
2 {
3     public TupleDiscoveryConfiguration()
4     {
5         AddSource(new SourceInfo
6         {
7             TDataSource = typeof(AllRefusedDataSource),
8             SourceCode = "allcode",
9             LongText = "All_refused",
10            ShortText = "All_refused"
11        });
12
13        AddSource(new SourceInfo
14        {
15            TDataSource = typeof(RefusedTypeDataSource),
16            SourceCode = "refusedtype",
17            LongText = "Refused_type",
18            ShortText = "Refused_type"
19        });
20    }
21 }
```

Codice 3.34: `TupleDiscoveryConfiguration.cs`

Il metodo `AddSource`, definito in `DiscoveryConfiguration`, ottiene attraverso `PropertiesConfiguration` le eventuali informazioni aggiuntive contenute nell'apposito file `{sourceCode}.json` per poi salvare la `SourceInfo` in una lista che verrà restituita attraverso il metodo pubblico `Configure`

```
1 public abstract class DiscoveryConfiguration
2 {
3     private List<SourceInfo> _sources = new List<SourceInfo>();
4     public List<SourceInfo> Configure()
5     {
6         return _sources;
7     }
8
9     protected void AddSource(SourceInfo source)
10    {
11        var sourceProperties = PropertiesConfiguration.GetFilters(source.SourceCode);
12        source.Parameters = sourceProperties.Filters.ToList();
13        source.Metadata = sourceProperties.Metadata;
14        _sources.Add(source);
15    }
16
17 }
```

Codice 3.35: `DiscoveryConfiguration.cs`

`DataDiscoveryConfiguration` ottiene tutte le classi che estendono `DiscoveryConfiguration`, per poi registrare in `SourceManagementService` tutte le source riscontrate

```
1 public static class DataDiscoveryConfiguration
2 {
3     public static void Configure(SourceManagementService sourceManagementService)
4     {
5         var configurations = AppDomain.CurrentDomain.GetAssemblies()
6             .SelectMany(x => x.GetTypes())
7             .Where(x => typeof(DiscoveryConfiguration).IsAssignableFrom(x) &&
8                 !x.IsInterface && !x.IsAbstract)
9             .ToList();
10    }
```

```

11     configurations.ForEach(configuration =>
12     {
13         sourceManagementService
14             .RegisterSources(
15                 ((DiscoveryConfiguration)Activator.CreateInstance(configuration))
16                 .Configure()
17             );
18     });
19 }
20 }
21 }

```

Codice 3.36: DataDiscoveryConfiguraton.cs

### 3.4.3 Interazioni Frontend

DataTypeCoupling è un enum che associa ad ogni risorsa il tipo di DataSource che utilizza.

```

1 ...
2
3 public enum DataTypeCoupling
4 {
5     [CouplingData(Label = "tuple", DataType = typeof(ITupleDataSource))]
6     tupleData,
7     [CouplingData(Label = "orders", DataType = typeof(OrdersDataSource))]
8     ordersData,
9
10    ...
11
12 }

```

Codice 3.37: DataTypeCoupling.cs

DiscoveryController presenta due endpoint:

- {resource}/discovery - restituisce lo SourceCode e le descrizioni delle source associate ad una DataSource del tipo associato ad resource, utilizzando il metodo GetDataSourcesByType di SourceManagementService
- filters/{sourceCode} - restituisce i parameters e i metadata di una specifica Source

Il sourceCode restituito durante questa fase verrà utilizzato dal frontend per richiedere i dati.

### 3.4.4 Utilizzo

Un Controller che restituisce dati in una determinata forma deve chiamarsi {resource}Controller, dove resource è una stringa registrata in DataTypeCoupling.

```

1 [Route("api/widgets/{controller}")]
2 [ApiController]
3 public class TupleController: ControllerBase
4 {
5
6     ...
7
8 }

```

Codice 3.38: TupleController.cs

Deve presentare un endpoint Post nella forma `api/widgets/{resource}`

```
1 [HttpPost]
2 public IActionResult Search([FromBody] GenericRequestDTO request)
3 {
4     var dataSource = _sourceManagementService
5         .GetDataSource<ITupleDataSource>(request.SourceCode);
6     return new ObjectResult(dataSource.GetData(request.Filters));
7 }
```

Codice 3.39: TupleController.cs

Prende come Body una `GenericRequest`, che presenta i seguenti campi:

- `SourceCode` indica da quale fonte di dati devono essere presi i dati.
- `Filters` sono gli eventuali parametri necessari alla esecuzione della query.

```
1 public class GenericRequestDTO
2 {
3     public string SourceCode { get; set; }
4     public List<Parameter> Filters { get; set; }
5 }
```

Codice 3.40: GenericRequestDTO.cs

Restituisce la risorsa gestita dal Controller utilizzando il metodo `GetData` della `DataSource` che ottiene attraverso il metodo `GetDataSource<T>` di `SourceManagementService`, passandogli il `SourceCode` contenuto nella richiesta

## 3.5 Topology

I widget lato mobility sono impiegati per mostrare i dati provenienti dalle macchine la cui struttura si articola in: linee, banchi e stazioni; queste ultime sono la parte del macchinario volta ad effettuare i test sui vari componenti che attraversano la linea.

### 3.5.1 ITopologyDataSource

L'interfaccia `ITopologyDataSource` contiene la definizione di tre metodi:

```
1 interface ITopologyDataSource
2 {
3     public List<int> GetLines();
4     public List<int> GetBenches(int lineCode);
5     public List<int> GetStations(int benchCode);
6 }
```

Codice 3.41: TopologyController.cs

Le classi che attualmente la implementano sono `AllRefusedDataSource` e `RefusedTypeDataSource`, le quali andranno a richiamare i metodi necessari per la richiesta della lista dei codici di linee, banchi o stazioni, presente nella repository collegata ad un database MySQL.

### 3.5.2 Controller Topology

Quando si procede con la creazione di un widget mobility è possibile scegliere la source, in modo tale da definire la fonte in cui reperire i dati da visualizzare. In seguito,

facendo riferimento al formato standard dei macchinari, si ha la possibilità di scegliere il codice della linea, del banco e della stazione di cui si vogliono osservare i risultati. Per poter scegliere questi codici vengono effettuate tre diverse chiamate al backend, in cui il controller definisce tre richieste `HttpGet`:

```

1  [HttpGet("{sourceCode}/lines")]
2  public IActionResult GetLines([FromRoute] string sourceCode)
3  {
4      var dataSource = _sourceManagementService
5      .GetDataSource<ITopologyDataSource>(sourceCode);
6      return new ObjectResult(dataSource.GetLines());
7  }
8
9  [HttpGet("{sourceCode}/benches/{lineCode}")]
10 public IActionResult GetBenches([FromRoute] string sourceCode, int lineCode)
11 {
12     var dataSource = _sourceManagementService
13     .GetDataSource<ITopologyDataSource>(sourceCode);
14     return new ObjectResult(dataSource.GetBenches(lineCode));
15 }
16
17 [HttpGet("{sourceCode}/stations/{benchCode}")]
18 public IActionResult GetStations([FromRoute] string sourceCode, int benchCode)
19 {
20     var dataSource = _sourceManagementService
21     .GetDataSource<ITopologyDataSource>(sourceCode);
22     return new ObjectResult(dataSource.GetStations(benchCode));
23 }

```

Codice 3.42: TopologyController.cs

Tramite il metodo `GetDataSource<DataSource>` definito in `SourceManagementService`, viene creata l'istanza della `DataSource` che implementa `ITopologyDataSource`, in base al code passato come parametro.

Fornendo questa interfaccia come tipo, si ha la certezza che siano stati implementati i metodi volti al reperimento delle liste di codici di linee, banchi e stazioni.

## 3.6 Persistence

Il Persistence Layer, in letteratura Data Access Layer, è utilizzato per la manipolazione dei dati archiviati in una memoria persistente.

La corrispondenza tra modello object oriented (domain model) e schema del database è data in gestione ad un framework ORM (object-relational mapping) [Mice]: Entity Framework, il quale ha la funzione di semplificare le operazioni di interrogazione e manipolazione dei dati.

### 3.6.1 Struttura Persistence

Nella conformazione che si denota in figura ??, ciascuna cartella definisce un contesto applicativo di ed è articolata in:

- **Entity configurations:** vi sono le configurazioni delle entità presenti in **Models**, effettuando il mapping di tabelle e colonne nel database,
- **Models:** ciascun modello è un'entità che rappresenta l'analogo del record, o riga, di una tabella,

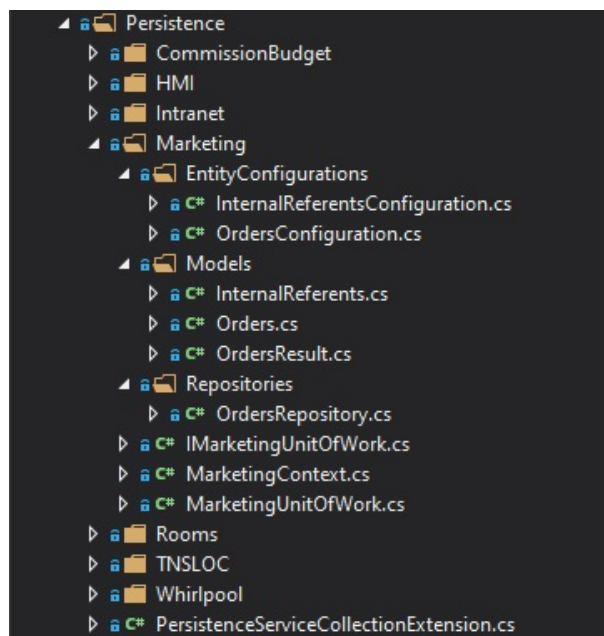


Figura 3.4: Formato cartella persistence

- **Repositories:** volte all'interrogazione e manipolazione dati tramite l'uso di Data Manipulation Language (quali Linq e Nest, rispettivamente per database SQL ed ElasticSearch),
- **IUnitOfWork:** interfaccia in cui vengono definite repository e metodo SaveChanges,
- **DbContext:** in questa classe viene fornita la connection string del database (definita in appsettings), si dichiarano i DbSet relativi ai model, che sono entity set utilizzati per le CRUD operations [Ent] ed in OnModelCreating si applicano le configurazioni dei model,
- **UnitOfWork:** implementa l'interfaccia creando la repository ed applicando il metodo SaveChanges al context; quest'ultimo ha la funzione di salvare un insieme di modifiche in una transazione, oppure di eseguire un **rollback** (ossia un ripristino allo stato che precede l'aggiornamento)

PersistenceServiceCollectionExtension si occupa di configurare le dependency injection per la creazione del context ElasticSearch.

## 3.7 Caching

Il caso d'uso del progetto in questione prevede una moltitudine di dashboard collegate simultaneamente a un solo backend; al fine di migliorare la performance è stato applicato un alleggerimento del carico tramite un middleware ASP.NET di cache. Ciò ha permesso la riduzione delle richieste effettivamente soddisfatte con dati generati appositamente da database SQL ed ElasticSearch sottostanti.



### 3.7.1 Implementazione

Il sistema di cache è stato implementato basandosi su di un'implementazione già esistente di `IMemoryCache` in ASP.NET Core. [Micb]

```
1 services.AddMemoryCache(options => Configuration.Bind("Cache", options));
```

Codice 3.43: Startup.cs, Memory Cache Injection

La chiave di cache viene generata dal frontend nell'interceptor http come hash MD5 [IETc] della concatenazione tra metodo [IETb], url con parametri [IETf] e corpo [IETa] della richiesta http in uscita.

```
1 mergeMap(token => {
2     this.md5.appendAsciiStr(request.method).appendAsciiStr(request.urlWithParams)
3     .appendStr(JSON.stringify(request.body));
4     request = request.clone({
5         setHeaders: {
6             Authorization: `Bearer ${token}`,
7             'AULOS-hash': this.md5.end(false) as string
8         }
9     });
10    return next.handle(request).pipe(catchError(error => {
11        if (error instanceof HttpErrorResponse && error.status === 401) {
12            return this.handle401Error(request, next);
13        } else {
14            return throwError(error);
15        }
16    }));
17 }
```

Codice 3.44: Http Interceptor, line 28

Il middleware di cache controlla se la richiesta in arrivo ha header `AULOS-hash` e, se esistente, controlla se ha già una entry in cache associata. In caso positivo, il middleware non chiama il middleware successivo nella catena e restituisce la entry salvata.

```
1 context.Request.Headers.TryGetValue("AULOS-hash", out StringValues key);
2     if (_cache.TryGetValue(key, out CustomCacheEntry entry))
3     {
4         await WriteResponse(context, entry);
5         return;
6     }
```

Codice 3.45: TotallyOriginalCachingMiddleware.cs, Cache hit scenario

Per permettere allo sviluppatore backend di decidere a quali richieste limitare l'inserimento in cache, è stato implementato un attributo di cache estensione di `ActionFilterAttribute` [Mica], che se anteposto a un controller o un metodo http di un controller permette di decorare il contesto della relativa richiesta http [Micf] in arrivo con flag `ServerDuration` necessario al distinguere richieste abilitate all'inserimento in cache. Il flag contiene la durata delle entry di cache relative all'azione a cui è applicato l'attributo di cache.

```
1
2 public class CacheAttribute : ActionFilterAttribute
3 {
4     public int ServerDuration { get; set; }
5     public override void OnActionExecuting(ActionExecutingContext context)
6     {
7         if (ServerDuration > 0)
8         {
9             context.HttpContext.Items["ServerDuration"] = $"{ServerDuration}";
10        }
11        else context.HttpContext.Items.Add("ServerDuration", 120);
12        base.OnActionExecuting(context);
13    }
14 }
```

14

}

Codice 3.46: CacheAttribute.cs

In caso negativo, il middleware lascia procedere la pipeline e, se presente nel contesto http [Micf] il flag `ServerDuration` generato da attributi di cache [Mica] (Fig. ??), salva la risposta generata dalla pipeline come entry in cache di durata `ServerDuration` e lascia procedere la pipeline. Se non presenti flag, il middleware lascia procedere la pipeline senza effettuare azioni.

```

1 entry = await CaptureResponse(context);
2
3     if (IsNotServerCachable(context))
4         return;
5     if (entry != null)
6     {
7         int serverCacheDuration = int.Parse((string)context.Items["ServerDuration"]);
8
9         _cache.Set(key, entry, TimeSpan.FromSeconds(serverCacheDuration));
10    }
11 }
```

Codice 3.47: TotallyOriginalCachingMiddleware.cs, Cache miss scenario

## 3.8 Load Testing

Si è testata la capacità del sistema di offrire funzionalità senza interruzioni all'aumentare del carico. Si è optato per effettuare load test su backend tramite una suite di test JMeter. [Fou]

### 3.8.1 Implementazione

I test sono stati impostati per effettuare automaticamente la procedura di login per poi usare l'access token così ottenuto per autenticare le richieste inviate verso il backend.

Sono stati simulati 250 utenti concorrenti emettenti richieste a endpoint che prevedono l'uso di cache, ripetendo test con lifetime variabili delle entry in cache. La dimensione della tabella di cache è stata mantenuta costante (60000 entry) generando casualmente valori di header `AULOS-hash` (Ch. ??). Il corpo delle richieste è stato fornito tramite un file .csv contenente tutte le possibili richieste generabili da un utente tramite la dashboard.

Testando la performance del backend quando sottoposto a un carico molto superiore a quello previsto dal caso d'uso dell'applicativo, è possibile avere un certo grado di sicurezza che il sistema sarà stabile una volta in utilizzo.

### 3.8.2 Risultati

Sono stati eseguiti test senza cache e con lifetime di entry in cache pari a 5, 15, 30 e 120 secondi. Una volta eseguiti tutti i test, si sono raccolte le seguenti statistiche, divise per durata delle entry di cache. Tutti i test hanno avuto la durata di un'ora. L'hit ratio di entry in cache è stato calcolato secondo la seguente formula, dove  $x$  è il prodotto tra lifetime delle entry in cache e throughput finale (richieste/s) restituito da JMeter.

$$f(x) = \left\{ \begin{array}{ll} \frac{f(x-1) + \frac{cache\_size - f(x-1)}{cache\_size}}{cache\_size}, & \text{for } x \geq 1 \\ 1 & \text{for } x = 0 \end{array} \right\} = \text{Cache hit ratio}$$

Figura 3.5: Cache hit ratio formula

La figura ?? rappresenta i dati relativi alle richieste al secondo mentre la figura ?? rappresenta la percentuale stimata di hit in cache durante il test, calcolata tramite la formula in fig. ??.

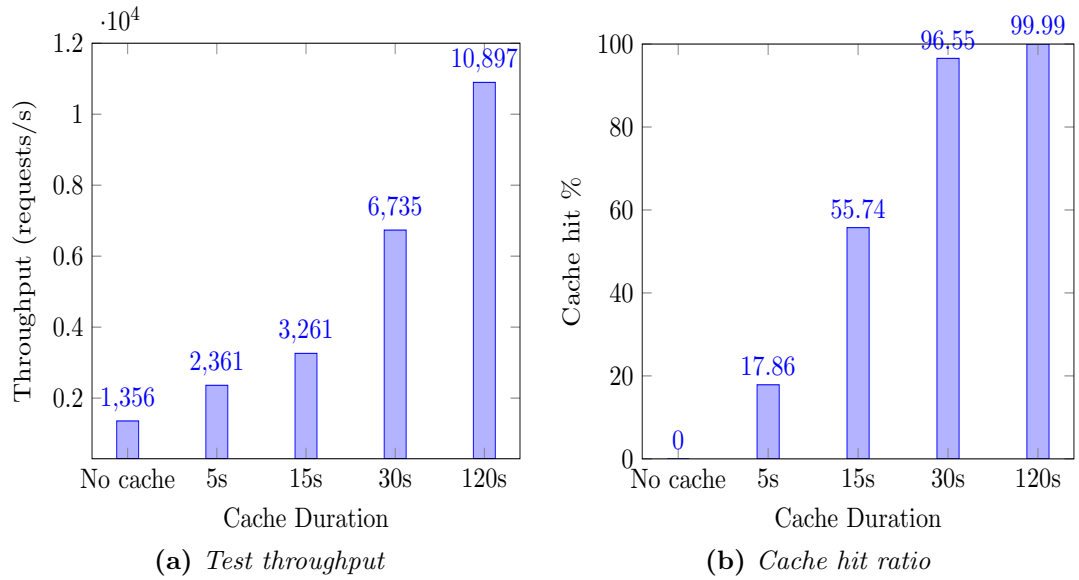


Figura 3.6: Dati estrapolati dai load test

## 3.9 Configurazione

Precedentemente, lo sviluppo – prettamente verticale – di un widget e la procedura di accesso dati per popolarlo di dati si articolava in:

Prima:

- Creazione component widget
  - Parte grafica HTML
  - Gestione interfacciamento con il widget
  - Interfacciamento con il service
  - Gestire interamente la logica di acquisizione dati
- Creazione widget
  - Creazione e registrazione di parametri
  - Messa a disposizione di valori dei parametri e relativi observable
- Creazione module
  - Registrazione widget
- Creazione service
  - Implementazione logica di acquisizione dati (Richieste http et cetera)
- Registrazione widget su module

Dopo:

- Creazione component widget
  - Parte grafica HTML
  - Definizione callback di aggiornamento dati
- Creazione widget
- Creazione module
- Creazione service
- Registrazione widget su module

### 3.9.1 Widget Component

Lo sviluppo di un Widget Component ha subito le seguenti modifiche architetturali: **Logica acquisizione dati** precedentemente la logica con la quale un widget otteneva i dati veniva definita all'interno del Component, come in codice **??**. Attualmente invece viene

```

1 export class OrdersComponent extends WidgetComponent<OrdersWidget> implements OnInit {
2
3   ...
4
5   ngOnInit() {
6     this.loading = true;
7     this.updateRequest();
8     this.widget.interestsParameterChanges.subscribe(() => {
9       this.updateRequest();
10      this.refreshData();
11    });
12    this.refreshData();
13  }
14
15  ...
16

```

```

17 protected refreshData(): void {
18     this.ordersWidgetService.getOrders(this.request).subscribe(orders => {
19         this.setOrdersView(this.checkUnseenOrders(orders));
20         this.loading = false;
21         this.changeDetectorRef.markForCheck();
22     });
23 }

```

Codice 3.48: Utilizzo metodo refreshData, orders.component.ts

### 3.9.2 Backend

## 3.10 Widget Realizzati

Per pensare ed ideare una soluzione che permettesse di generalizzare la struttura dei widget e il loro trasferimento dati, durante il corso del progetto ne sono stati sviluppati diversi. All'interno della dashboard i widget sono stati suddivisi in due categorie: widget IT e widget Mobility. I primi mostrano dati relativi ai flussi interni gestionali della Loccioni mentre i secondi riguardano dati che vengono generati dai test effettuati nelle stazioni di prova dei macchinari.

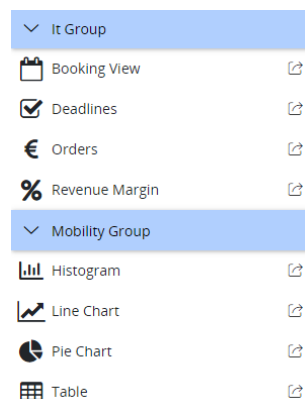
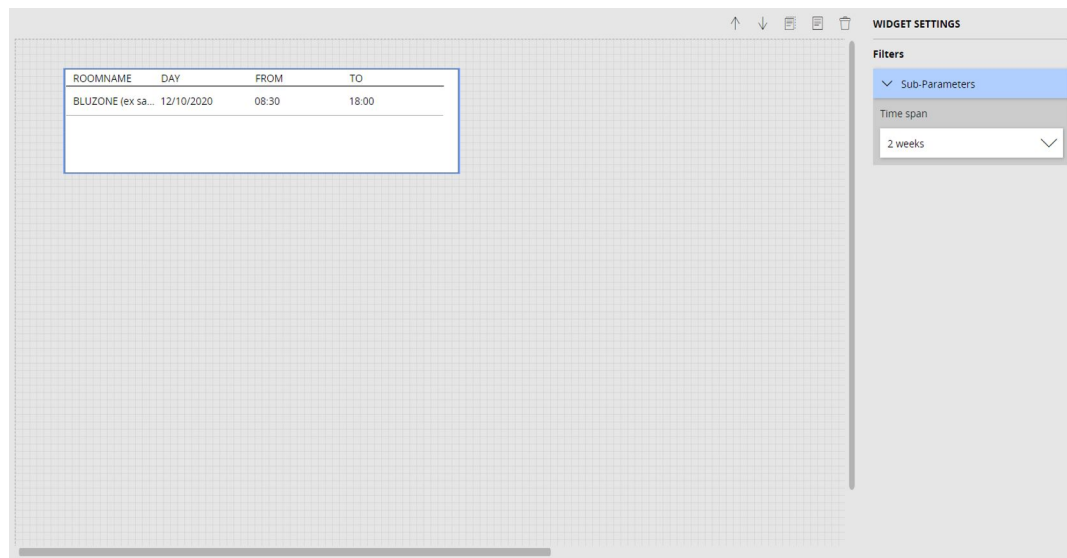


Figura 3.7: Lista dei widget realizzati

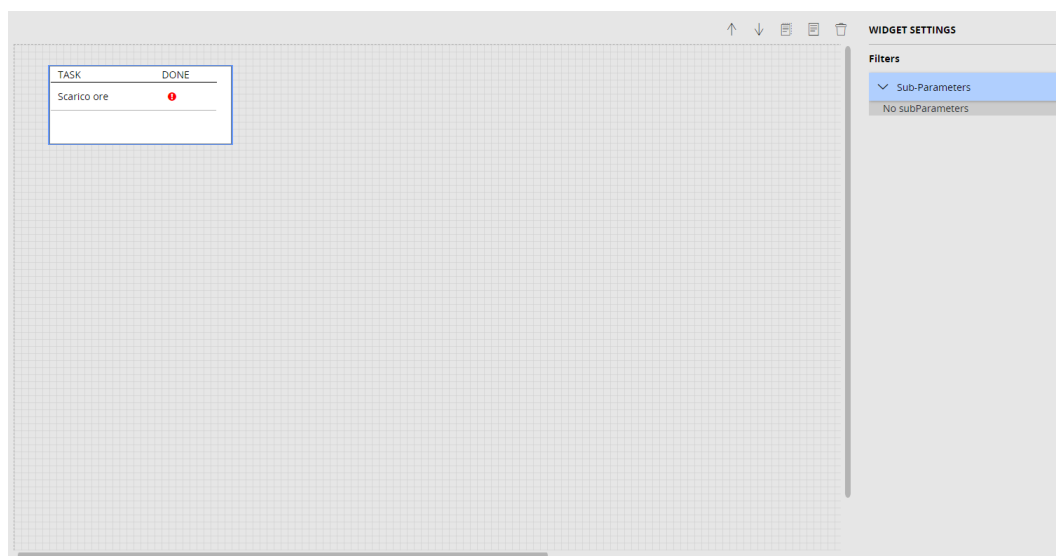
### Gruppo IT

- Booking view widget:

permette di visualizzare le prenotazioni dei salottini (stanze predisposte per effettuare riunioni). I parametri sulla destra filtrano i dati secondo l'arco di tempo selezionato, partendo dalla data corrente.



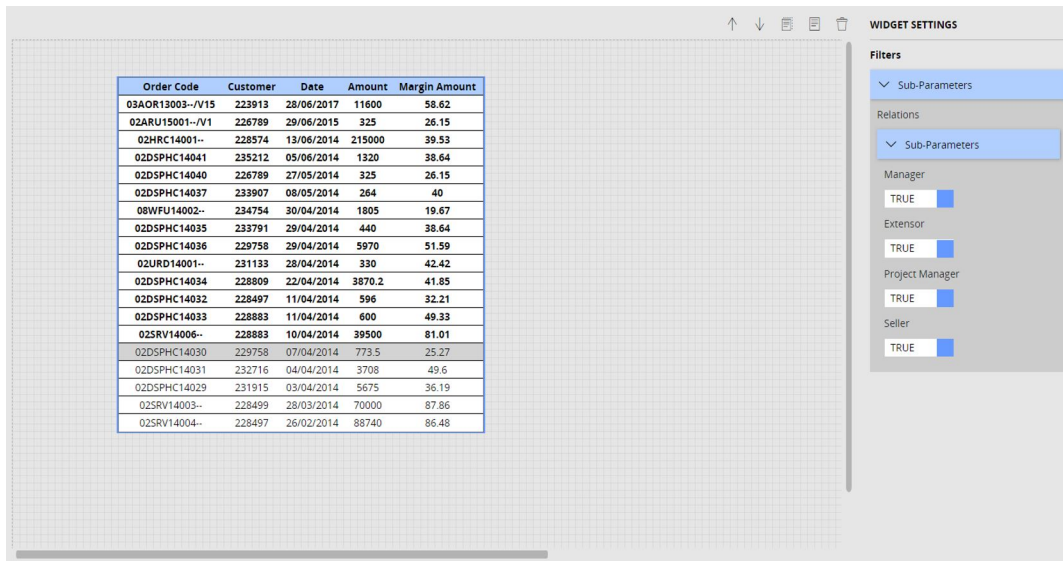
- Deadlines widget:



permette di visualizzare un allarme in base alle scadenze dell'utente autenticato nel sistema.

- Orders widget:

permette di visualizzare informazioni relative agli ordini. I parametri sulla destra filtrano i dati in base al ruolo che l'utente ricopre per quell'ordine.



Order Code	Customer	Date	Amount	Margin Amount
03AOR13003--/V15	223913	28/06/2017	11600	58.62
02ARU15001--/V1	226789	29/06/2015	325	26.15
02HRC14001--	228574	13/06/2014	215000	39.53
02DSPHC14041	235212	05/06/2014	1320	38.64
02DSPHC14040	226789	27/05/2014	325	26.15
02DSPHC14037	233907	08/05/2014	264	40
08WU14002--	234754	30/04/2014	1805	19.67
02DSPHC14035	233791	29/04/2014	440	38.64
02DSPHC14036	229758	29/04/2014	5970	51.59
02URD14001--	231133	28/04/2014	330	42.42
02DSPHC14034	228809	22/04/2014	3870.2	41.85
02DSPHC14032	228497	11/04/2014	596	32.21
02DSPHC14033	228883	11/04/2014	600	49.33
02SRV14006--	228883	10/04/2014	39500	81.01
02DSPHC14030	229758	07/04/2014	773.5	25.27
02DSPHC14031	232716	04/04/2014	3708	49.6
02DSPHC14029	231915	03/04/2014	5675	36.19
02SRV14003--	228499	28/03/2014	70000	87.86
02SRV14004--	228497	26/02/2014	88740	86.48

**WIDGET SETTINGS**

**Filters**

Sub-Parameters

Relations

Sub-Parameters

Manager

TRUE

Extensor

TRUE

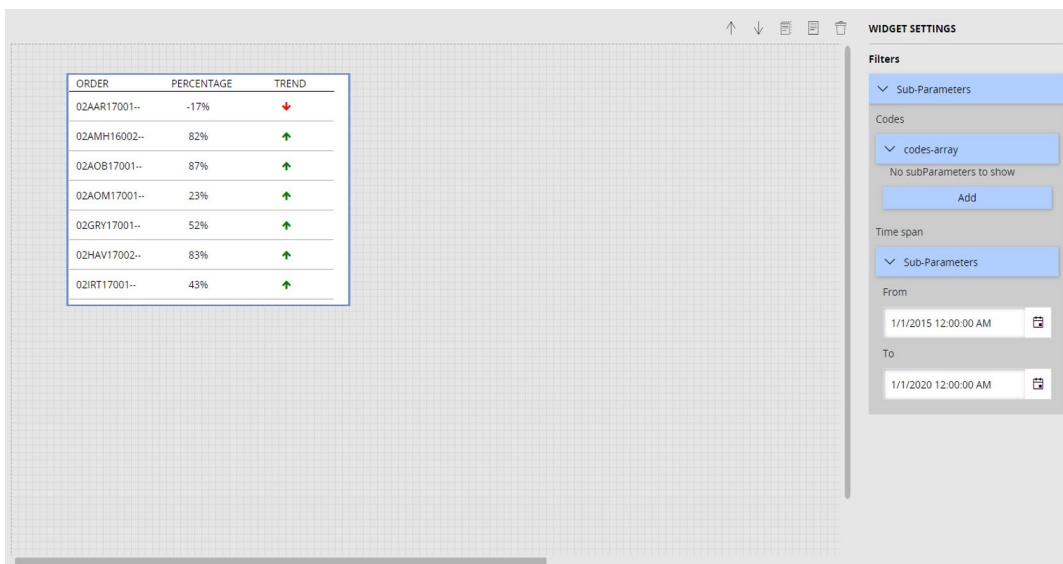
Project Manager

TRUE

Seller

TRUE

- Revenue margin widget:



ORDER	PERCENTAGE	TREND
02AAR17001--	-17%	↓
02AMH16002--	82%	↑
02AOB17001--	87%	↑
02AOM17001--	23%	↑
02GRV17001--	52%	↑
02HAV17002--	83%	↑
02IRT17001--	43%	↑

**WIDGET SETTINGS**

**Filters**

Sub-Parameters

Codes

codes-array

No subParameters to show

Add

Time span

Sub-Parameters

From

1/1/2015 12:00:00 AM

To

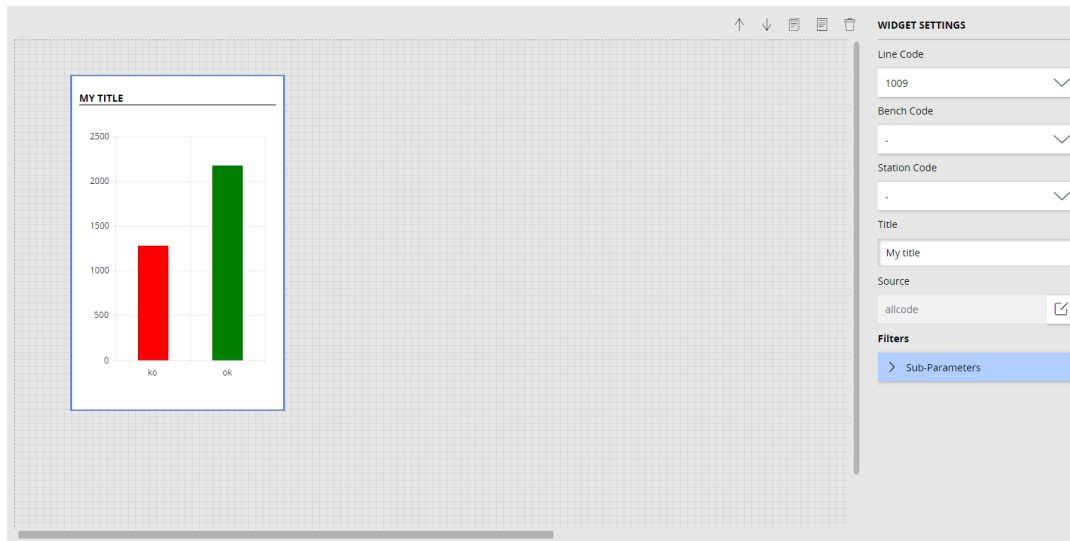
1/1/2020 12:00:00 AM

permette di visualizzare il trend delle commesse e il valore in percentuale. I parametri sulla destra filtrano i dati in base all'arco di tempo selezionato e permettono di aggiungere ulteriori commesse al widget.

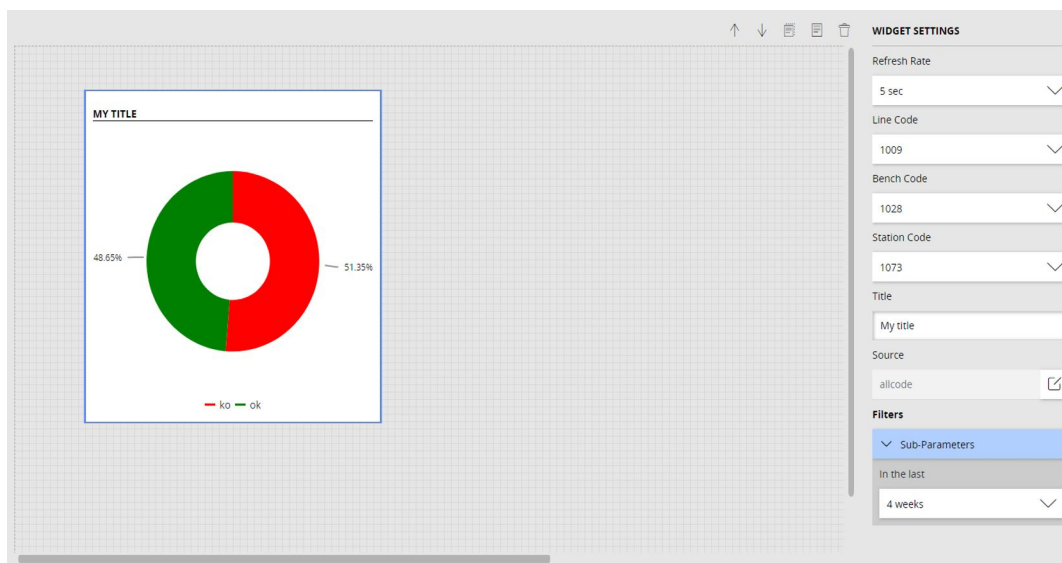
## Gruppo Mobility

- Histogram widget:

permette di visualizzare dati provenienti dalle macchine Loccioni in un grafico ad istogramma. I parametri sulla destra filtrano i dati e permettono all'utente di selezionare la source da cui prendere i dati.



- Pie chart widget:



permette di visualizzare dati provenienti dalle macchine Loccioni in un grafico a torta. I parametri sulla destra filtrano i dati e permettono all'utente di selezionare la source e di cambiare l'intervallo di tempo in cui il widget ricarica i dati dal backend.

- Line chart widget:

permette di visualizzare dati provenienti dalle macchine Loccioni in un grafico a linee.





- Table widget:

ST	OEE	A %	A Min	A Max	P	Q %	R
A05	95	80	4	43	22	52	74
A01	33	23	8	60	86	83	30
A07	82	4	15	54	8	7	2
A08	78	1	16	53	78	99	15
A09	95	80	23	43	22	52	74
A10	33	23	42	58	86	83	30
A11	82	4	6	66	8	7	2
A12	78	1	56	90	78	99	15
A13	95	80	17	43	22	52	74
A14	33	23	54	92	86	83	30
A15	82	4	6	73	8	7	2
A16	78	1	56	84	78	99	15

permette di visualizzare dati provenienti dalle macchine Loccioni in un grafico tabellare. I parametri sulla destra consentono all'utente di specificare le colonne e le righe che si vogliono caricare e consentono di impostare dei range secondo i quali un valore sarà visualizzato come positivo o negativo.

Il "line chart widget" ed il "table widget" non sono stati collegati ad alcun database ed utilizzano quindi dati fittizi.



# Bibliografia

- [202] Microsoft 2020. *Visual Studio*. URL: <https://visualstudio.microsoft.com/it/vs/community/>.
- [Ang] *Introduction to Angular concepts*. URL: <https://angular.io/>.
- [Bec] Ansgar Becker. *HeidiSQL*. URL: <https://www.heidisql.com/>.
- [Ela] *An Overview on Elasticsearch and its usage*. URL: <https://towardsdatascience.com/an-overview-on-elasticsearch-and-its-usage-e26df1d1d24a%7D>.
- [Ent] EntityFrameworkTutorial.net. *DbSet in EF*. URL: <https://www.entityframeworktutorial.net/entityframework6/dbset.aspx>.
- [Fou] Apache Software Foundation. *JMeter*. URL: <https://jmeter.apache.org/>.
- [Gita] *Git (software)*. URL: [https://it.wikipedia.org/wiki/Git\\_\(software\)](https://it.wikipedia.org/wiki/Git_(software)).
- [Gitb] *GitLab*. URL: <https://it.wikipedia.org/wiki/GitLab>.
- [Htm] *HTML*. URL: <https://en.wikipedia.org/wiki/HTML>.
- [IETa] IETF. *Http Body*. URL: <https://tools.ietf.org/html/rfc2616#section-4.3>.
- [IETb] IETF. *Http Methods*. URL: <https://tools.ietf.org/html/rfc2616#section-9>.
- [IETc] IETF. *Message Digest Algorithm MD5*. URL: <https://www.ietf.org/rfc/rfc1321.txt>.
- [IETd] IETF. *Refresh Token*. URL: <https://tools.ietf.org/html/rfc6749#section-1.5>.
- [IETe] IETF. *Resource Owner Password Credentials Grant*. URL: <https://tools.ietf.org/html/rfc6749#section-4.3>.
- [IETf] IETF. *URI Query Parameters*. URL: <https://tools.ietf.org/html/rfc3986#section-3.4>.
- [Ken] *Modern UI made easy*. URL: <https://www.telerik.com/>.
- [Lin] *Language Integrated Query*. URL: <https://docs.microsoft.com/it-it/dotnet/csharp/programming-guide/concepts/linq/>.
- [Mica] Microsoft. *Action Filters*. URL: <https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-3.1>.
- [Micb] Microsoft. *AddMemoryCache*. URL: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.memorycacheservicecollectionextensions.addmemorycache?view=dotnet-plat-ext-3.1>.
- [Micc] Microsoft. *ASP.NET*. URL: <https://dotnet.microsoft.com/>.
- [Micd] Microsoft. *ASP.NET*. URL: <https://dotnet.microsoft.com/apps/aspnet>.
- [Mice] Microsoft. *Entity Framework*. URL: <https://docs.microsoft.com/it-it/ef/>.

- [Micf] Microsoft. *HttpContext*. URL: <https://docs.microsoft.com/en-us/dotnet/api/system.web.httpcontext?view=netframework-4.8>.
- [Micg] Microsoft. *Visual Studio Code*. URL: <https://code.visualstudio.com/docs>.
- [Mss] *MicrosoftSQLServerManagementStudio*. URL: <https://docs.microsoft.com/it-it/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>.
- [Nes] *Nest*. URL: <https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/nest-getting-started.html>.
- [Npm] *Npm (Software)*. URL: [https://it.wikipedia.org/wiki/Npm\\_\(software\)](https://it.wikipedia.org/wiki/Npm_(software)).
- [Pos] *Postman*. URL: <https://www.geeksforgeeks.org/introduction-postman-api-development/>.
- [Typ] *TypeScript*. URL: <https://en.wikipedia.org/wiki/TypeScript>.