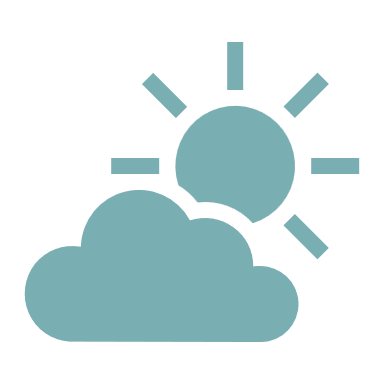
**WeatherWise** – Cloud Computing

Mattia **Piazzalunga** & Davide **Soldati**



**Table of contents**

[Introduction 3](#_Toc154822748)

[Introduction 3](#_Toc154822749)

[Functionality and implementation details 3](#_Toc154822750)

[Some WeatherWise screenshots and explanation 3](#_Toc154822751)

[Technologies, languages and tools 3](#_Toc154822752)

[Universally Unique Identifier (UUID) 3](#_Toc154822753)

[Microservices 3](#_Toc154822754)

[Programming Languages, Frameworks and API 3](#_Toc154822755)

[CRUD 3](#_Toc154822756)

[API and API Rest interface 4](#_Toc154822757)

[An external API: Open Meteo 5](#_Toc154822758)

[An external API: Geonames 5](#_Toc154822759)

[Organization of the application: the architecture of PlaDat 5](#_Toc154822760)

[A microservices pattern: qui nome 5](#_Toc154822761)

[A Cloud-Native Application 5](#_Toc154822762)

[Introduction 5](#_Toc154822763)

[Docker 5](#_Toc154822764)

[Kubernates 5](#_Toc154822765)

[Version control: Git e GitHub 5](#_Toc154822766)

[The DevOps Culture 6](#_Toc154822767)

[The Continuos Integration Pipepline 6](#_Toc154822768)

[How to run WeatherWise 6](#_Toc154822769)

[A short guide 6](#_Toc154822770)

[Other information 6](#_Toc154822771)

[README, License, Authors e .gitignore 6](#_Toc154822772)

[Released version 6](#_Toc154822773)

[Useful Links 7](#_Toc154822774)

[LICENSE 7](#_Toc154822775)

[Conclusion 8](#_Toc154822776)

[The team 8](#_Toc154822777)

Introduction

Introduction

//parlare di weaterwise cosa è

Functionality and implementation details

La prima realease di WeatherWise porta con se alcune future intuitive per la consultazione semplice ed intuitiva del meteo di una data città, con qualche interessante funzionalità aggiuntiva:

* Consultazione (get) del meteo della città in un range di 7 giorni, compresa temperatura e umidità;
* Possibilità di fornire (post) feedback per il meteo della città attuale (la previsione). Il feedback per una data città può essere modificato (put/delete) al massimo entro un’ora da quando è stato fornito. Il sistema permette di associare alle previsioni una probabilità di correttezza sul luogo (get) data dai feedbacks degli ultimi 7 giorni;
* Possibilità di fornire (post) un consiglio su un “luogo di interesse” da visitare in città, nonché ottenerne una panoramica di tutti i posti da visitare in quel luogo (getAll);
* Il sistema permette l’inserimento controllato delle città fornendo un elenco aggiornato di esse (get).

Some WeatherWise screenshots and explanation

//qui

Technologies, languages and tools

Universally Unique Identifier (UUID)

As can be seen in the previous points, entities in our reality are identified by an id of type **UUID**. The UUID, Universally Unique Identifier, is an **identifier** used in software projects and **standardized** by the Open Software Foundation, created for the purpose of enabling a distributed system to identify information in the absence of a centralized coordinating system: anyone can create a UUID and use it with **reasonable probability that it will not be used by anyone else**.

In its canonical form, the UUID is represented by **32 hexadecimal characters** divided into five groups and generally includes a reference to the network address of the host generating the UUID, a timestamp and an arbitrary component. Given the extensive use of UUIDs in **entreprise applications**, a use confirmed by the internship experiences of group members, despite PlaDat's small size, we preferred to use this type of primary key for our entities while trying to give a more realistic fit to the design.

Both technologies are used in the development of WatherWise.

Microservices

//qui

Per ogni microservizio abbiamo un’architettura logica così divisa:

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente

Nel dettaglio:

* Il controller riceve le richieste http dall’esterno;
* La logica di business viene eseguita nel service layer;
* Per salvare le informazioni nel repository facciamo uso di un layer repository.

Spring Cloud

Spring Cloud è un progetto sotto l’ecosistema di progetti spring che si concentra sulla creazione di applicativi a microservizi affidabili e robusti. Esso è un framework open-source sviluppato in ambiente Java ed è ampiamente utilizzato nella programmazione di applicazioni Java enterprise. L'utilizzo di Spring Cloud semplifica lo sviluppo di applicazioni basate su microservizi, fornendo un insieme di strumenti e librerie che affrontano le sfide tipiche dell'architettura microservizi, come la gestione della configurazione, la discovery dei servizi e la comunicazione tra i componenti.

Programming Languages, Frameworks and API

//qui

CRUD

**CRUD** is an acronym that refers to the **four main operations for manipulating entities in a relational database**:

* **Create**: creating tables, data, and relationships;
* **Read**: reading what was created;
* **Update**: modification, by updating, of what was created;
* **Delete**: deleting what was created.

These operations were implemented in our project specifically for **DB management**.

API and API Rest interface

**APIs** (Application Programming Interface) are a set of **definitions and protocols** by which application software is built and integrated. A **REST API**, on the other hand, is a **programming interface that uses HTTP to handle remote data**. In other words, for most developers, **designing a RESTful API consists of:**

* **defining web-accessible resources and identifying them with URLs**;
* **mapping CRUD** (Create, Retrieve, Update, Delete) **operations on those resources to HTTP protocol verbs** (Post, Get, Put, Delete).

Due to the fact that almost all devices support HTTP, they can easily work with the REST interface without further implementation. The result is web services that are particularly valued for a high degree of platform **independence, scalability, performance, interoperability**, **and flexibility**. For the reasons just mentioned, we decided to use the **REST architecture to implement our platform**, also in anticipation of a possible, future expansion of functionality.

For this reason, we also designed, in this sense, **APIs that were built ad-Hoc** to differentiate HTTP calls.

**Microservice 1 – Weather Consulting**

Percorso base: /meteo

|  |  |  |
| --- | --- | --- |
| **Path** | **Method** | **Description** |
| **/{cityName}/{number}** | GET | Ritorna il meteo della città a {number} giorni dall’odierno, fornendo, inoltre, indicazioni relative ad umidità e temperatura. |

**Microservice 2 – Feebacks**

Percorso base: /feedbacks

|  |  |  |
| --- | --- | --- |
| **Path** | **Method** | **Description** |
| **/{cityName}** | GET | Ritorna una probabilità riguardo alla correttezza del meteo sulla base dei feedbacks forniti negli ultimi 7 giorni. |
| **/** | POST | Aggiunge un feedback sull’effettività della previsione meteo della città fornendo la città ed il valore binario. |
| **/{cityName}/{progressiveID}** | PUT | Modifica uno specifico feedback, invertendone il {binaryValue}. Questa funzionalità è applicabile entro un’ora dalla POST. |
| **/{cityName}** | DELETE | Elmina uno specifico feedback. Questa funzionalità è applicabile entro un’ora dalla POST. |

**Microservice 3 – Luoghi di interesse**

Percorso base: /places

|  |  |  |
| --- | --- | --- |
| **Path** | **Method** | **Description** |
| **/getAll/{cityName}** | GET | Ritorna tutti i luoghi di interesse di una data città |
| **/** | POST | Aggiunge un “luogo di interesse” per una città fornendo il nome della città, il nome del luogo, il nome dell’utente che lo consiglia, una valutazione ed una breve descrizione. |

**Microservice 4 – Consultazione città.**

Percorso base: /cities

|  |  |  |
| --- | --- | --- |
| **Path** | **Method** | **Description** |
| **/{prefixCityName}** | GET | Ritorna 5 città che iniziano con un dato prefisso. |

API-Gateway

Per accedere ai microservizi non possiamo fare riferimento, semplicemente, al chimare porte diverse, poiché l’environment dei microservizi può avere istanze multiple di un microservizio che potrebbe runnare su porte differenti, uindi non possiamo fare affidamento sull’hard coding dei valori delle porte quando vogliamo chiamare i servizi. Aggiungiamo, dunque, un componente all’inizio del nostro architectural landscape: l’API gateway che è responsabile di route le richieste dagli utenti ai servizi corrispondenti. L’API Gateway, dunque, sarà un entry point nel nostro system landscape e se l’utente volesse chiamare un servizio, chiamerà normalmente API gateway con un URL normale.

API Gateway può, inoltre aggiungere alcune funzionalità come, nel nostro caso:

* Load balancer

Mentre dall’esterno la comunicazione con l’API gateway funziona con lo schema TLS e l’HTTPS protocol, verso l’interno, siccome API gateway fa parte della rete dei microservizi, non necessitiamo dell’HTTPS perché è una comunicazione interna e, dunque, possiamo utilizzare una comunicazione http. Per questo possiamo dire che l’API-Gatway svolge anche la funzione di SSL termination.

Nel nostro caso utilizziamo l’implementazione di spring cloud per l’API Gateway: Spring Cloud Gateway.

Service Registry: Eureka Server

Considerando di avere migliaia di microservizi, per mantenerli ci serve un Server Registry: quando un microservizio viene creato viene connesso al Service Registry che contiene tutte le informazioni sui microservizi, come URL, port, ..

Un pattern, il discrovery server, che salva tutte le informazioni dei microseervizi come nome, ip, ..

Circuit breaker: Hystrix

Quando il server è down reindirizziamo a questo particolare REST services

Ogni servizio ha il suo metodo di fallback /user

Se non riceviamo risposta dopo 4000 milliseconds quindi 4 second, ci rindirizziamo su quello che sono i fallback method

PRI

Molte properties sono disponibili in tutti i microservizi che abbiamo; quindi, se dobbiamo cambiarle a mano dobbiamo cambiarle in tutti i posti. Creaimao, quindi, un server di configurazione che ci permette di ottenere le configurazioni da usare nei nostri microservizi. Creeremo, dunque, un server di configurazione

An external API: Open Meteo

//qui

An external API: Geonames

//

Organization of the application: the architecture of PlaDat

//qui

A microservices pattern: qui nome

//qui

A Cloud-Native Application

Introduction

Docker

//

Kubernates

//

Version control: Git e GitHub

**Version control** helps developers **track and manage changes regarding code** in a software project. **Git** (<https://git-scm.com/>)is open-source distributed version control software, precisely the most widely used in the world, so much so that it is **considered a standard** (*according to a Stack Overflow developer survey, more than 87 percent of developers use Git*). It is critical for collaboration in project development: developers issue their work locally and then synchronize the repository copy with the copy in the server. Making it easier and more efficient for the team to use Git comes into play **GitHub** (<https://github.com/>) a **hosting service for software projects**, a true cloud-based implementation of Git.

The DevOps Culture

//

The Continuos Integration Pipepline

//

How to run WeatherWise

A short guide

//qui

Other information

README, License, Authors e .gitignore

To conclude with the analysis present in this report, we would like to highlight the task of **four files present in the project folder** and which have, despite often being underestimated, great importance: README, License, Authors, and .gitignore.

* **README** is a file that contains information about the files contained in an archive or directory and is commonly included in software packages;
* **License** is a file that contains the full text of the license chosen for the project without any modifications;
* **Authors** is a file that identifies who worked on a particular project which critical for copyright management;
* **.gitignore** is a file in the git system that contains a set of items to be ignored by the version control system.

Released version

In software development, **version** corresponds to a certain **state in the development of a software**. Conventions for numbering a software version normally involve a triplet of numbers in the form **X.Y.Z**, where X, Y, and Z:

* **X is the major version**: which should increase only as a result of radical changes in the product, such as those that make it in some way incompatible with its earlier versions;
* **Y is the minor version**: which increases with the introduction of small features to complement existing ones, but maintaining substantial compatibility;
* **Z is the patch version**: which augments usually only by correcting errors with the same functionality.

We release the **first version of WeatherWise** (v1.0.0).

Useful Links

We report **useful links** to WeatherWise here:

* Link a **GitHub** repository

<https://github.com/mattiapiazzalunga/WeatherWise>;

LICENSE

WeatherWise is provided under **GPL-3.0 license.**

**But what does it mean?**

The GNU General Public License version 3.0 (**GPL-3.0**) is a software license developed by the **Free Software Foundation** (FSF). It is designed to ensure the **freedom of users to run, modify and share software.**

We **summarize the key features** of GPL licenses.

* **Each program with its own source**

Should a developer wish to distribute the result of his labors under a GPL license, he is required to always publish the source code alongside the usual pre-compiled version.

* **Each program must be accompanied by a copy of the licensed**

GPL requires that all works protected by it also include a copy of the full text from the license of use itself, in which the guaranteed freedoms and obligations to be respected are explained.

To preserve maximum clarity over time, it is not permitted to replace the license with a simple link: the actual full text must be conveyed.

* **No constraints until you distribute**

Users are free to take any listing released under the GPL and modify it as they wish. As long as the work derived from it remains within a circumscribed scope, the user is not required to make the source code of the derived application available.

This right also extends to professional circles: it is therefore perfectly permissible to take, for example, management software released under the GPL, adapt it to the specific needs of one's own business, and then use the resulting version without being forced to publish the source code for it.

* **If you make it publicly available, you also need the source**

The constraints of GPL become more stringent should one wish to make publicly available (and, therefore, also "sell") a program protected by that license or a derivative of it. In such a circumstance, the party is required to use the same GPL license, and is not allowed to impose more restrictive constraints.

It does not matter whether the changes are substantial such as a profound rewriting of certain functions, as minimal as the variation of a few strings of text or, again, whether it is the same program "rebranded": what matters is that, should the decision be made to distribute it to the public, the source of any derived version must also be made available in the same manner as provided by GPL (**copyleft** concept).

* **As long as the source is there, it can also be sold**

Nothing prohibits selling GPL-protected software: the important thing is that the presence of the source code is also guaranteed. It is clear, however, that the buyer can then choose to make that work available to third parties in a completely legal way.

This is not a contradiction: by selling GPL-protected software, the merchant is not paid for the product itself, but rather for the service provided, i.e., the "making available" of the product.

* **No linking from closed applications**

Another limitation that must be kept well in mind by those making derivative programs concerns the use of GPL-licensed libraries. So-called static linking (i.e., the copying, accomplished at compile time, of code contained in an external library within the main executable) is allowed only within programs that are later released under the same license.

In other words, it is not possible to make a closed source program using GPL-protected libraries.

Thanks to: <http://tinyurl.com/28f2bbzu>.



**Why** did we choose it?

* **To preserve freedom**
* We want our software to remain open and free; GPL-3.0 is an excellent choice. It protects users' freedom and helps preserve the sharing and collaboration approach.
* **Open development community**

GPL-3.0 is often associated with open source development projects and communities that promote collaboration. By using this license, you can take advantage of the support and contributions of a large community of developers.

Conclusion

The team

This report is maintained by the WeatherWise team, whose members are:

* Mattia **Piazzalunga** - 851931;
* Davide **Soldati** - 861178.

*We really care about the success of this project, so for any problems/understandings, we are available.*

*Main technologies used in WeatherWise (visual):*

*qui*