Documentazione WeatherApp

# Indice

1 Indice 2

2 Introduzione 3

2.1 Informazioni sul progetto 3

2.2 Abstract 3

2.3 Scopo 3

3 Analisi 4

3.1 Analisi del dominio 4

3.2 Analisi e specifica dei requisiti 4

3.2.1 Spiegazione elementi tabella dei requisiti: 7

3.3 Use case 8

3.4 Pianificazione 8

3.5 Analisi dei mezzi 11

3.5.1 Software 11

3.5.2 Hardware 11

4 Progettazione 13

4.1 Design delle interfacce 13

4.2 Diagramma d’architettura 13

5 Implementazione 14

5.1 fetchWeatherData 14

5.2 fetchOpenWeather() 14

5.3 updateCurrentTemperature() 15

5.4 processForecastData() 16

5.5 getWeeklyForecast() 17

5.6 extractDataByCity() 19

5.7 createChart() 19

5.8 getTemperature() 21

6 Test 21

6.1 Protocollo di test 21

6.2 Risultati test 24

6.3 Mancanze/limitazioni conosciute 26

7 Consuntivo 26

8 Conclusioni 28

8.1 Sviluppi futuri 28

8.2 Considerazioni personali 28

9 Bibliografia 29

9.1 Sitografia 29

10 Indice delle figure 29

11 Allegati 29

# Introduzione

## Informazioni sul progetto

In questo capitolo raccogliere le informazioni relative al progetto, ad esempio:

* Allievo: Amos Haefliger
* Docente: Geo Petrini
* Scuola: Scuola Arti e Mestieri di Trevano, TI
* Sezione: Informatica
* Materia: M306
* Data di inizio: 04.09.2024
* Data di consegna: 18.12.2024

## Abstract

The weatherApp application aims to make it easy and straightforward to view certain weather forecasts for selected cities around the world chosen by the end user. The application shows the current temperature with its precipitation percentages, the forecast for the coming week with its precipitation percentages, the ability to save certain cities to a list of favorite cities so that forecasts for these cities can be viewed quickly, and the display of a graph that takes into account a history of the cities saved in the favorite list.

## Scopo

Lo scopo del progetto è quello di poter tenere traccia della temperatura corrente, delle previsioni future e di una visualizzazione anche grafica delle temperature per determinate città nel mondo. L’applicativo è pensato per tutti, con un facile utilizzo, e una chiara visualizzazione grafica delle informazioni.

All’interno del progetto sarà presente anche una semplice guida per guidare un possibile utente in difficoltà.

# Analisi

## Analisi del dominio

L’applicativo sarà utilizzabile su più piattaforme (Windows,Linux,MacOs) ma solo su firefox, può venir utilizzato da chiunque vista la sua estrema facilità, e non ha un target preciso. Chiunque abbia bisogno di visualizzare in modo chiaro e veloce le previsioni meteo può utilizzare l’applicativo. In mercato ci sono migliaia di prodotti simili ma la facilità d’uso e la chiara visualizzazione dei dati reputo che sia un punto a favore per la WeatherApp. Naturalmente non sono necessarie skill, conoscenze o competenze particolari per riuscire ad utilizzare il programma nel modo corretto.

## Analisi e specifica dei requisiti

In base all’accurata analisi eseguita con il committente e con la continua comunicazione con esso abbiamo appurato i seguenti requisiti:

|  |  |
| --- | --- |
| **Req-01** | |
| **Nome** | Visualizza temperatura, precipitazioni, umidità e vento correnti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente visualizza le informazioni meteo giornaliere |
| **Sotto requisiti** | |
| **001** | Città da visualizzare scritta correttamente |
| **002** | Città da visualizzare presente nella lista dell’API di openWeather |
| **003** | Aver confermato la ricerca della città desiderata |

|  |  |
| --- | --- |
| **Req-02** | |
| **Nome** | Visualizza previsioni settimanali |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’utente visualizza le previsioni per la settimana a venire |
| **Sotto requisiti** | |
| **001** | Città da visualizzare scritta correttamente |
| **002** | Città da visualizzare presente nella lista dell’API di openWeather |
| **003** | Aver confermato la ricerca della città desiderata |

|  |  |
| --- | --- |
| **Req-03** | |
| **Nome** | Aggiunta città ai preferiti |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Aggiunta della città corrente ai preferiti |
| **Sotto requisiti** | |
| **001** | Città da visualizzare scritta correttamente |
| **002** | Città da visualizzare presente nella lista dell’API di openWeather |
| **003** | Aver confermato la ricerca della città desiderata |
| **004** | Aver confermato l’aggiunta della città ai preferiti tramite il button |

|  |  |
| --- | --- |
| **Req-04** | |
| **Nome** | Lista preferiti |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | L’utente visualizza la lista delle città preferite |
| **Sotto requisiti** | |
| **001** | Avere almeno una città salvata nelle preferite |

|  |  |
| --- | --- |
| **Req-04** | |
| **Nome** | TextToSpeech |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | L’utente si fa restituire una versione vocale della situazione corrente |
| **Sotto requisiti** | |
| **001** | Città per la quale far restituire il vocale scritta correttamente |
| **002** | Città per la quale sentire il vocale presente nella lista di openWeather |
| **003** | Aver confermato la ricerca della città desiderata |
| **004** | Aver confermato la richiesta del vocale tramite l’apposito button del microfono |

|  |  |
| --- | --- |
| **Req-05** | |
| **Nome** | Visualizzazione grafico |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | L’utente visualizza il grafico della città interessata con uno storico delle temperature di 10 anni |
| **Sotto requisiti** | |
| **001** | Città per la quale visualizzare il grafico scritta correttamente |
| **002** | Città per la quale visualizzare il grafico presente nella lista di openWeather |
| **003** | Aver confermato la ricerca della città desiderata |

|  |  |
| --- | --- |
| **Req-06** | |
| **Nome** | Visualizzazione grafico |
| **Priorità** | 2 |
| **Versione** | 2.0 |
| **Note** | L’utente visualizza il grafico di una delle città preferite con uno storico da quando la città preferita è stata salvata nei cookie |
| **Sotto requisiti** | |
| **001** | Disporre di almeno una città nella lista dei preferiti |
| **002** | Selezionare una città di quelle nei preferiti (se si vuole utilizzare il button apposito nella lista delle preferite) |
| **003** | Aver confermato la ricerca della città preferita (se passata manualmente) |

|  |  |
| --- | --- |
| **Req-06** | |
| **Nome** | TextToSpeech |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | L’utente sceglie una città e sente la descrizione della situazione |
| **Sotto requisiti** | |
| **001** | Selezionare una città |
| **002** | Avere delle casse, delle cuffie o comunque un dispositivo audio collegato al pc |

### Spiegazione elementi tabella dei requisiti:

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto.

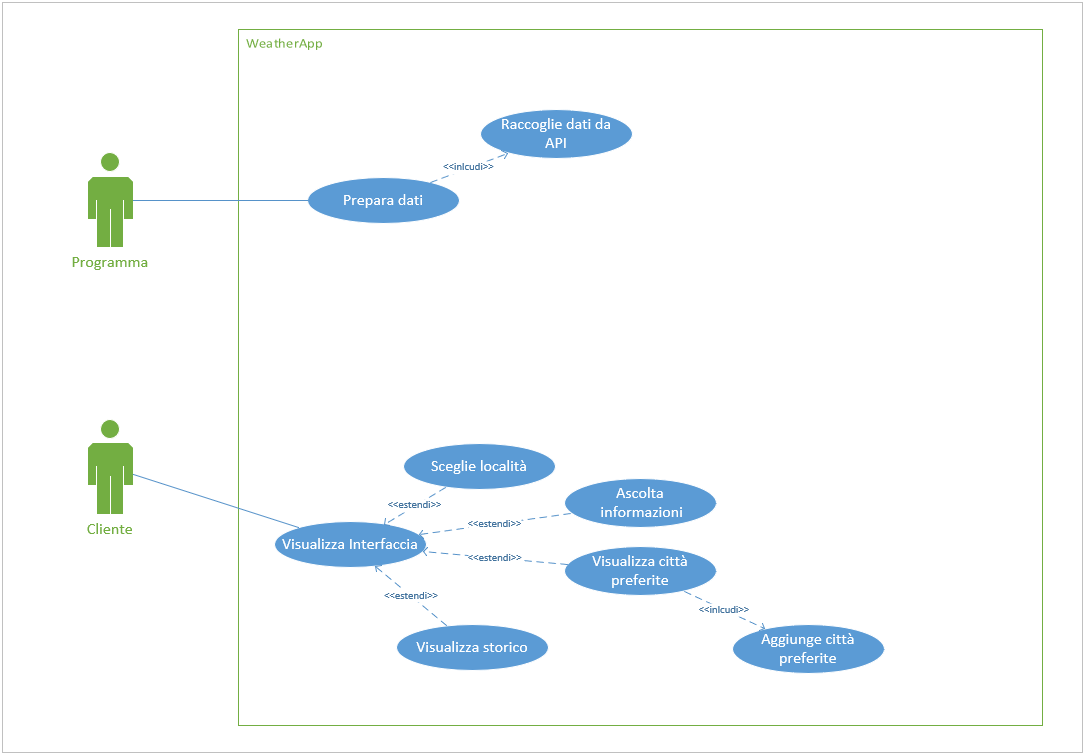


Figura 1 UseCase

## Pianificazione

Prima di stabilire una pianificazione bisogna avere almeno una vaga idea del modello di sviluppo che si intende adottare. In questa sezione bisognerà inserire il modello concettuale di sviluppo che si seguirà durante il progetto. Gli elementi di riferimento per una buona pianificazione derivano da una scomposizione top-down della problematica del progetto.

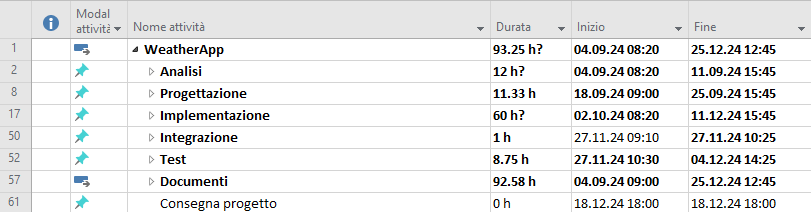


Figura 2 Diagramma Gantt preventivo



Figura 3 Diagramma Gantt preventivo - Analisi

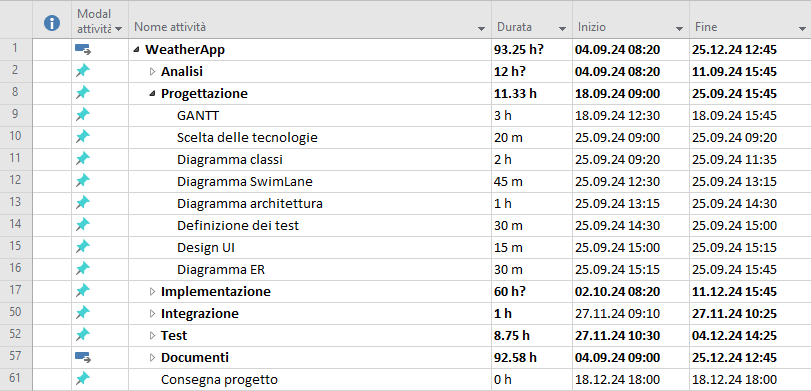


Figura 4 Diagramma Gantt preventivo – Progettazione

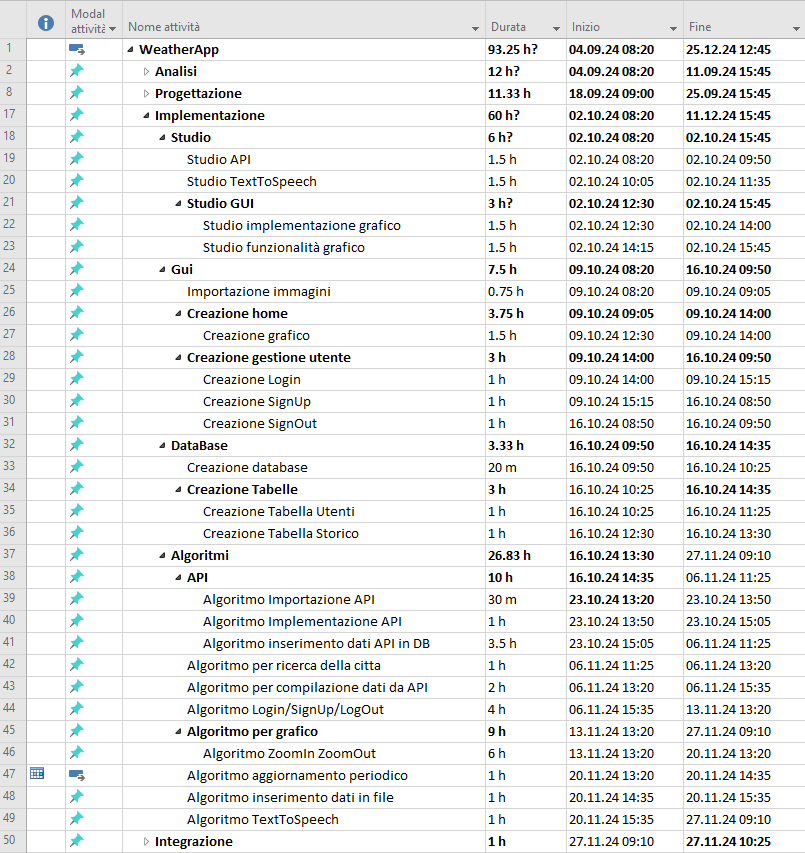


Figura 5 Diagramma Gantt preventivo – Implementazione - Algoritmi

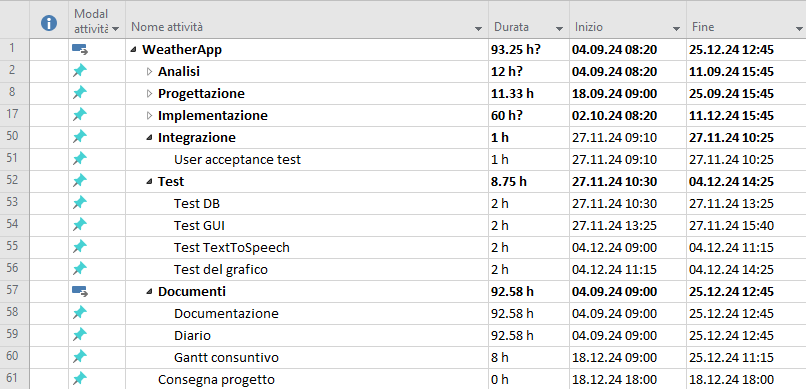


Figura 6 Diagramma Gantt preventivo – Integrazione – Test - Documenti

## Analisi dei mezzi

Per realizzare il progetto ho utilizzato i seguenti mezzi:  
Pc scolastico per la gran parte del lavoro, sviluppo, documentazione, progettazione

Pc privato per una piccola parte di sviluppo e refactoring del codice

Firefox per aiutarmi in fase di sviluppo per visualizzare il lavoro, nonché piattaforma di utilizzo del progetto.

Google chrome per documentarmi, trovare soluzioni e visualizzare esempi di mockup

Terminale per caricare il progetto sul repository di github

Visio per la parte di progettazione

Project per la parte di pianificazione

### Software

Di software ho utilizzato visual studio code per lo sviluppo dell’applicativo, firefox per testare e visualizzare il progetto, google chrome per documentarmi, trovare soluzioni e visualizzare esempi di mockup, terminale per caricare il progetto sul repository di github, visio per la parte di progettazione, project per pianificare il progetto

Come librerie ho utilizzato BootStrap per la grafica e chart.js per lo sviluppo del grafico.

### Hardware

L’hardware utilizzato per sviluppare il progetto è il seguente:

**PC di scuola:**

* OS: Windows 10, 64 bit
* CPU: Intel i7-13700, 2.1GHz
* RAM: 32 GB, 4400MHz
* GPU: NVIDIA T400, 4GB

**PC portatile personale:**

* OS: Windows 11, 64 bit
* CPU: Intel i7-12650H, 2.3GHz
* RAM: 16 GB, 4800MHz
* GPU: NVIDIA RTX 3050 Ti Laptop Edition, 4GB

**SSD (Samsung T7):**

* MEMORIA: 1000 GB
* USB: 3.1 Type-C

# Progettazione

Questo capitolo descrive esaustivamente come deve essere realizzato il prodotto fin nei suoi dettagli. Una buona progettazione permette all’esecutore di evitare fraintendimenti e imprecisioni nell’implementazione del prodotto.

## Design delle interfacce

Descrizione delle interfacce interne ed esterne del sistema e dell’interfaccia utente. La progettazione delle interfacce è basata sulle informazioni ricavate durante la fase di analisi e realizzata tramite mockups.

## Diagramma d’architettura

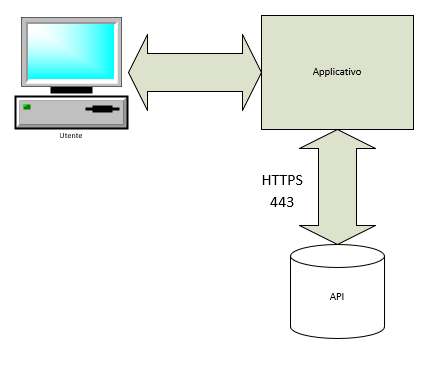
Diagramma che rappresenta come lavora il programma

Figura 7 Diagramma d’architettura

# Implementazione

## fetchWeatherData

async function fetchWeatherData(city, apiKey) {

    var response = await fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric&lang=it`);

    var data = await response.json();

    return data;

}

Con questa funzione facciamo la chiamata all’API basandoci sulla città e sull’API key. Restituiamo il data set con il quale lavoreremo dopo.

## fetchOpenWeather()

/\*\*

 \* Funzione che esegue la richiesta all'API

 \* @param {città sulla quale fare la richiesta} city

 \* @param {API key di openWeather} apiKey

 \* @returns

 \*/

async function fetchOpenWeather(city, apiKey) {

    var data = await fetchWeatherData(city, apiKey);

    var temp = Math.round(data.main.temp);

    var iconCode = data.weather[0].icon;

    var iconUrl = `https://openweathermap.org/img/wn/${iconCode}@2x.png`;

    var humidity = data.main.humidity;

    var windSpeed = data.wind.speed;

    var description = data.weather[0].description;

    var precipitation = data.rain ? data.rain["1h"] : 0;

    return { temp, iconUrl, humidity, windSpeed, description, precipitation };

}

In questa funzione chiamiamo l’API grazie alla funzione fetchWeatherData(). Andiamo ad estrapolare dal dataSet che ci restituisce la temperatura, l’url dell’icona che andremo poi ad utilizare e visualizzare in un secondo momento, la percentuale d’umidità, la velocità del vento, una breve descrizione della situazione e la percentuale di precipitazioni

## updateCurrentTemperature()

/\*\*

 \* Funzione che aggiorna la situazione giornaliera

 \*/

async function updateCurrentTemperature() {

    try {

        var { temp, iconUrl, humidity, windSpeed, description, precipitation } = await fetchOpenWeather(city, OPENWEATHER\_API\_KEY);

        document.getElementById('city').innerHTML = `${city}`;

        document.getElementById('today-temp').innerHTML = `${temp}°C`;

        document.getElementById("weather-icon").src = iconUrl;

        document.getElementById('today-humidity').innerHTML = `Umidità: ${humidity}%`;

        document.getElementById('today-wind').innerHTML = `Vento: ${windSpeed} km/h`;

        document.getElementById('today-precip').innerHTML = `Precipitazioni: ${precipitation}%`;

        document.getElementById('weather-description').innerHTML = `Oggi c'è ${description}, la temperatura è di ${temp} gradi Celsius e la probabilità di precipitazioni è pari a ${precipitation}%`;

    } catch (error) {

        var modal = new bootstrap.Modal(document.getElementById('errorModal'));

        document.getElementById("modal-body").innerHTML = "Nome città errato";

        modal.show();

        cleanBackdrop(); // Funzione per rimuovere le sovrapposizioni dei modali

    }

}

In questa funzione andiamo sempre ad utilizzare i dati che ci restituisce la funzione fetchOpenWeather() grazie alla funzione fetchWeatherData per la chiamata all’API e per il dataSet. Successivamente andiamo a riempire i campi pre-definiti in HTML con i dati che riceviamo. In caso di errore facciamo visualizzare un modal di bootStrap per spiegare l’errore.

## processForecastData()

function processForecastData(forecastList) {

    var groupedData = forecastList.reduce((acc, forecast) => {

        var date = forecast.dt\_txt.split(' ')[0]; // Estrai solo la data (es. "2024-11-21")

        if (!acc[date]) acc[date] = [];

        acc[date].push(forecast);

        return acc;

    }, {});

    // Calcola i valori medi per ogni giorno

    return Object.keys(groupedData).map(date => {

        var dayData = groupedData[date];

        var temps = dayData.map(f => f.main.temp);

        var avgTemp = temps.reduce((sum, temp) => sum + temp, 0) / temps.length;

        // Estrai altri valori interessanti

        var humidity = dayData[0].main.humidity; // Umidità

        var windSpeed = dayData[0].wind.speed; // Velocità del vento

        var precipitation = dayData[0].pop \* 100; // Probabilità di pioggia in percentuale

        return {

            date,

            avgTemp,

            description: dayData[0].weather[0].description, // Usa la descrizione del primo intervallo

            icon: dayData[0].weather[0].icon, // Usa l'icona del primo intervallo

            humidity,

            windSpeed,

            precipitation

        };

    });

}

Questa funzione ci serve principalmente come supporto ma essendo una funzione comunque core che va a preparare i dati mi sembrava giusto mostrarla.

## getWeeklyForecast()

/\*\*

 \* Funzione per ricevere le previsioni settimanali

 \* @param {città per la quale prendere le previsioni} city

 \* @param {apiKey di openWeather} apiKey

 \*/

async function getWeeklyForecast(city, apiKey) {

    try {

        var { lat, lon } = await getCityCoordinates(city, apiKey);

        var response = await fetch(`https://api.openweathermap.org/data/2.5/forecast?lat=${lat}&lon=${lon}&appid=${apiKey}&units=metric&lang=it`);

        var data = await response.json();

        if (!data.list || data.list.length === 0) {

            throw new Error('Nessuna previsione settimanale disponibile');

        }

        var dailyForecasts = processForecastData(data.list);

        let forecastHtml = `

            <h2 class="text-center my-3">Previsioni per la settimana</h2>

            <h3 class="card-title text-center">${city}</h3>

            <div class="row gx-3 gy-3 justify-content-center align-items-stretch">`;

        // Cicla sui dati giornalieri per creare un elenco di previsioni

        dailyForecasts.forEach((day) => {

            var date = new Date(day.date);

            var dayName = date.toLocaleDateString('it-IT', { weekday: 'long' });

            var temperature = Math.round(day.avgTemp.toFixed(1)); // Temperatura media arrotondata

            var description = day.description; // Descrizione meteo

            var icon = day.icon; // Icona meteo

            var humidity = day.humidity; // Umidità

            var windSpeed = day.windSpeed; // Velocità del vento

            var precipitation = day.precipitation; // Probabilità di pioggia

            var iconUrl = `https://openweathermap.org/img/wn/${icon}@2x.png`;

            forecastHtml += `

                <div class="col-12 col-md-6 col-lg-4">

                <div class="card weather-card text-center shadow-sm h-100" style="position: relative; overflow: hidden;">

                <div class="card-background" style="background-image: url('${iconUrl}');"></div>

                        <div class="card-body d-flex flex-column justify-content-between" style="position: relative; z-index: 1;">

                            <h5 class="card-title">${dayName}</h5>

                            <p class="card-text">Temperatura media: ${temperature}°C</p>

                            <p class="card-text">${description}</p>

                            <p class="card-text">Umidità: ${humidity}%</p>

                            <p class="card-text">Vento: ${windSpeed} km/h</p>

                            <p class="card-text">Precipitazioni: ${precipitation}%</p>

                        </div>

                    </div>

                </div>

            `;

        });

        forecastHtml += '</div>';

        document.getElementById('weekly-forecast').innerHTML = forecastHtml;

    } catch (error) {

        var modal = new bootstrap.Modal(document.getElementById('errorModal'));

        document.getElementById("modal-body").innerHTML = "Previsioni settimanali non trovate";

        modal.show();

        cleanBackdrop();

    }

}

In questa funzione andiamo come prima cosa a richiamare la funzione getCityCoordinates() che ci restituisce latitudine e longitudine della città sulla quale vogliamo trovare le previsioni settimanali. Per questo motivo quindi non possiamo riutilizzare la funzione fetchWeatherData(). Sucessivamente processiamo i dati che riceviamo dalla richiesta all’API tramite la funzione processForecastData(). Una volta processati i dati andiamo a prenderne i valori interessanti quali temperatura, descrizione, l’icona ecc… Successivamente andiamo a stampare a livello HTML all’interno del div weekly-forecast una lista di giorni con le informazioni che ci interessano, giorni che sono ciclati dal foreach che si basa sul set di dati che ci ha restituito il processForecastData(). In caso di errore stampiamo un modal per spiegare l’errore.

## extractDataByCity()

function extractDataByCity(city) {

    // Recupera i dati storici dallo storage

    var history = getHistory();

    var dates = []; // Array per memorizzare le date

    var temperatures = []; // Array per memorizzare le temperature

    // Filtra i dati dello storico per città specifica

    history.forEach(record => {

        if (record.citta === city) {

            dates.push(record.ora);   // Aggiungi l'ora dei dati

            temperatures.push(record.temperatura);  // Aggiungi la temperatura

        }

    });

    // Ritorna un oggetto contenente le etichette (date) e le temperature per la città

    return { dates, temperatures };

}

Questa funzione prende prima di tutto i dati dal localStorage grazie alla funzione getHistory(), in caso non esistano ancora restituisce un array vuoto. Successivamente prepara due array: uno per le temperature (asse y) e uno per le date (asse x). Successivamente per ogni record dello storico inserisce negli array appositi i record dell’ora dei dati e della temperatura. Questi array li andremo ad utilizzare nella funzione per la creazione del grafico.

## createChart()

function createChart(city) {

    // Estrae i dati relativi alla città specifica

    var { dates, temperatures } = extractDataByCity(city);

    // Se non ci sono dati per la città, mostra un errore

    if (dates.length === 0 || temperatures.length === 0) {

        var modal = new bootstrap.Modal(document.getElementById('errorModal'));

        document.getElementById("modal-body").innerHTML = `Grafico non disponibile per: ${city}`;modal.show();  cleanBackdrop(); return;}

    // Se esiste già un grafico, distruggilo per crearne uno nuovo

    if (myChart) {myChart.destroy();}

    // Ottieni il contesto del canvas per il grafico

    var ctx = document.getElementById('temperatureChart').getContext('2d');

    // Crea un nuovo grafico di tipo 'line' (grafico a linee)

    myChart = new Chart(ctx, {

        type: 'line',  // Tipo di grafico (lineare)

        data: {

            labels: dates,  // Le etichette sono le date

            datasets: [{

                label: `Temperatura a ${city} (°C)`,

                data: temperatures,  // I dati sono le temperature

                borderColor: 'rgba(206, 212, 218, 1)',  // Colore della linea

                backgroundColor: 'rgba(75, 192, 192, 0.2)',  // Colore di sfondo

                fill: false,  // Non riempire sotto la linea

                tension: 0.1  // Impostazione per la curva della linea

            }]

        },

        options: {

            responsive: true,

            plugins: {

                legend: {

                    position: 'top',

                },

                tooltip: {

                    mode: 'index',

                    intersect: false,

                },

            },

            scales: {

                x: {

                    title: {

                        display: true,

                        text: 'Data'

                    },

                },

                y: {

                    title: {

                        display: true,

                        text: 'Temperatura (°C)'

                    },

                    min: Math.min(...temperatures) - 5,  // Imposta un limite inferiore per l'asse Y

                    max: Math.max(...temperatures) + 5   // Imposta un limite superiore per l'asse Y

                }

            }}});}

In questa funzione andiamo a creare il grafico basandosi sui dati che gli passiamo dalla funzione extractDataByCity(). In caso non disponessimo di dati per la città, il che significa che o non è nei preferiti o non lo è mai stata allora stampiamo un modal nel quale spieghiamo l’errore. Se c’è gia un grafico andiamo a distruggerlo in modo da non avere conflitti. Successivamente andiamo avanti con la classica creazione del grafico.

## getTemperature()

/\*\*

 \* Funzione che prende data, ora, città e temperatura

 \*

 \* @param {città interessata} city

 \*/

async function getTemperature(city) {

    try {

        var data = await fetchWeatherData(city, OPENWEATHER\_API\_KEY);

        var temp = data.main.temp;

        var { data: currentDate, ora: currentHour } = getDateTime();

        saveForecasts(city, currentDate, currentHour, temp);

    } catch (error) {

        console.error('Errore nel recupero dei dati meteo:', error);  // Logga eventuali errori

    }

}

Questa funzione prende temperatura, città, data e ora in modo da poi passarli alla funzione saveForecast() che ha lo scopo di salvare il dataSet nel localStorage.

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-001  REQ-01 | **Nome** | ShowCurrentDay |
| **Descrizione** | Visualizzare temperatura, precipitazioni, umidità e vento corrente | | |
| **Prerequisiti** | Vedi UseCase -> Req-01 | | |
| **Procedura** | 1. Avviare l’applicativo 2. Cercare una città | | |
| **Risultati attesi** | Visualizzare il tutto | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-002  REQ-02 | **Nome** | ShowForecast |
| **Descrizione** | Visualizzare temperatura, precipitazioni, umidità e vento dei giorni a venire | | |
| **Prerequisiti** | Vedi UseCase -> Req-02 | | |
| **Procedura** | 1. Avviare applicativo 2. Cercare una città | | |
| **Risultati attesi** | Visualizzare il tutto | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-003  REQ-03 | **Nome** | AddToFavorite |
| **Descrizione** | Visualizzare temperatura, precipitazioni, umidità e vento dei giorni a venire e della giornata corrente e che l’invio del button vada a buon fine | | |
| **Prerequisiti** | Vedi UseCase -> Req-03 | | |
| **Procedura** | 1. Avviare applicativo 2. Cercare una città 3. Confermare la città 4. Aggiungerla ai preferiti tramite l’apposito button | | |
| **Risultati attesi** | Visualizzare il tutto | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-004  REQ-04 | **Nome** | ShowNewFavoriteList |
| **Descrizione** | Visualizzare temperatura, precipitazioni, umidità e vento dei giorni a venire e della giornata corrente e vederla nella lista dei preferiti | | |
| **Prerequisiti** | Vedi UseCase -> Req-04 | | |
| **Procedura** | 1. Avviare applicativo 2. Cercare una città 3. Confermare la città 4. Aggiungerla ai preferiti tramite l’apposito button | | |
| **Risultati attesi** | Visualizzare il tutto | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-006  REQ-06 | **Nome** | ShowChart |
| **Descrizione** | Visualizzare temperatura, precipitazioni, umidità e vento dei giorni a venire e della giornata corrente vederla nella lista dei preferiti e vederne il grafico | | |
| **Prerequisiti** | Vedi UseCase -> Req-06 | | |
| **Procedura** | 1. Avviare applicativo 2. Selezionare una città dalla lista dei preferiti | | |
| **Risultati attesi** | Visualizzare il tutto | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-007  REQ-07 | **Nome** | testTextToSpeech |
| **Descrizione** | Sentire una voce robotica che ci descrive la situazione giornaliera secondo la descrizione che troviamo sotto il tasto blu del microfono | | |
| **Prerequisiti** | Avere o le casse abilitate o delle cuffie | | |
| **Procedura** | 1. Avviare applicativo 2. Selezionare una città 3. Premere il button blu per sentire il textToSpeech | | |
| **Risultati attesi** | Sentire | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Risultato ottenuto** | **Stato** |
| TC-001 |  | Passato |
| TC-002 |  | Passato |
| TC-003 |  | Passato |
| TC-004 |  | Passato |
| TC-006 |  | Passato |

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

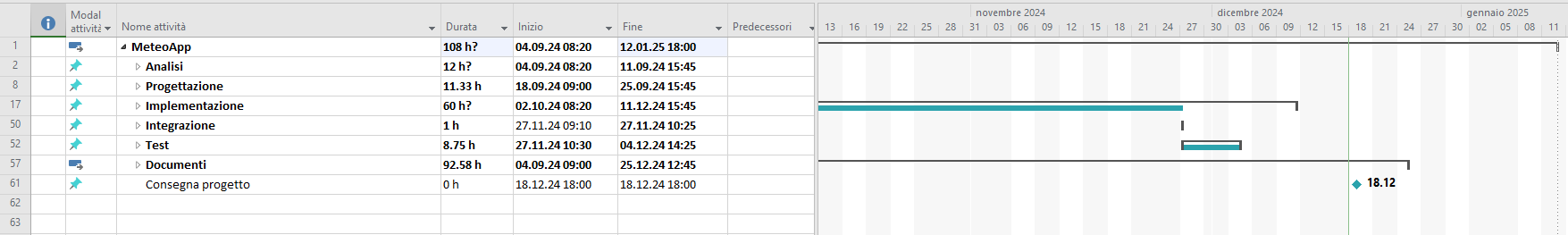
Non sono riuscito ad integrare un grafico che andasse nel passato di anni essendo che praticamente in tutte le API serviva un piano a pagamento, non sono riuscito ad integrare uno zoom a grafico, e non sono riuscito a rendere responsive il sito web

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

CONSUNTIVO

Figura 5 Esempio di diagramma di Gantt consuntivo



# Conclusioni

Trovo che sia un applicativo molto popolare e in generale un app non del tutto innovativa ma reputo che la facilità d’uso sia interessante e da considerare. Il progetto tutto sommato non è molto complesso, una volta che si capisce come lavorare con le API è abbastanza facile andare avanti, nel momento in cui però, vengono commissionate richieste partiolari come grafici zoomabili con uno storico annuale li si entra gia nel complesso.

## Sviluppi futuri

In futuro sarebbe interessante integrare lo zoom a grafico e rendere responsive il sito in modo da essere utilizzabile letteralmente da chiunque e ovunque.

In più riguarderei lo stile sia esteticamente che in funzionalità rendendolo tutto bootstrap, lo stesso varrebbe per l’implementazione di codice da fare o tutta solo con innerHTML o tutto con l’inserimento di codice HTML.

## Considerazioni personali

Ho imparato come lavorare un Json, come integrare le API in un progetto e come creare un grafico con la libreria di chart.js e in generale ho migliorato le mie skills con js.

# Bibliografia

## Sitografia

1. <https://chatgpt.com/>
2. <https://getbootstrap.com/>
3. <https://www.chartjs.org/>
4. <https://openweathermap.org/¨>
5. <https://stackoverflow.com/>
6. <https://dev.to/devsmitra/convert-text-to-speech-in-javascript-using-speech-synthesis-api-223g>

# Indice delle figure

[Figura 1 UseCase 7](#_Toc124775984)

[Figura 2 Diagramma Gantt preventivo 8](#_Toc124775985)

[Figura 3 Diagramma Gantt preventivo - Analisi 10](#_Toc124775986)

[Figura 4 Diagramma Gantt preventivo - Progettazione 11](#_Toc124775987)

[Figura 5 Diagramma Gantt preventivo – Implementazione - Algoritmi 16](#_Toc124775988)

[Figura 6 Diagramma Gantt preventivo – Integrazione – Test - Documenti 16](#_Toc124775988)

[Figura 5 Diagramma Gantt preventivo – Diagramma d’architettura 16](#_Toc124775988)

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente
* QdC
* FileUtilizzati
* Diagrammi