Progetto pgCOVAPI

Silvio Caprara

Maturità 2021

Contents

1 Introduzione

Il progetto **pgCOVAPI** è nato con l'intenzione di realizzare un'applicazione informatica legata a temi di attualità e con un utilizzo concreto. La pandemia COVID-19 ha portato alla creazione di numerosi sistemi informatici implementati per innumerevoli scopi come: la raccolta, l'immagazzinamento, l'elaborazione e la mostra di dati rappresentanti la situazione dell'epidemia.

Per questo motivo, si è realizzato un servizio WEB accessibile a tutti capace di fornire dati concreti provenienti dalla protezione civile riguardanti l'epidemia COVID-19 in Italia. Questo servizio, definito con l'acronimo **API** (Application Program Interface), è un sistema informatico capace di interagire con basi di dati (ad esempio leggere informazioni da un database) ricevendo semplici richieste effettuate utilizzando la barra di ricerca del browser. Come definito dall'acronimo, un'API é un software tramite il quale è possibile implementare ulteriori applicazioni. In supporto a questa affermazione si é inoltre realizzato un sito web che elabora e mostra in formato grafico i dati forniti dall'API da noi implementata.

Per la realizzazione del sistema sono state utilizzate tecnologie attuali, in uso anche nel mondo del lavoro in piattaforme come Facebook, Instagram e Discord. Ogni membro del gruppo ha lavorato su tutte le componenti costituenti l'applicazione, sperimentando il ruolo di *full-stack developer*[1].

2 Diagramma dei casi d'uso

Un utente può interfacciarsi al sistema informatico in due diverse modalità. Un normale utente interessato alla visualizzazione dei dati elaborati in formato grafico ha la possibilità di visitare il sito web.

Figure 1: Casi d'uso per utente generico

[1]Si definisce full-stack developer uno sviluppatore "a tutto tondo", in grado di lavorare su tutta la pila di tecnologie che costituiscono un'applicazione informatica.

Uno sviluppatore può invece interfacciarsi all'API del sistema, richiedendo i dati per elaborarli in una propria applicazione.

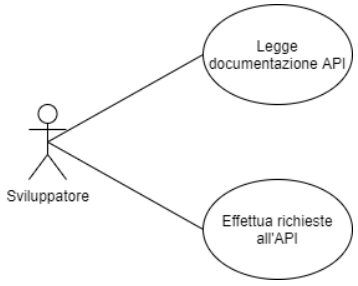


Figure 2: Casi d'uso per sviluppatore

3 Progettazione dell'interfaccia utente

Come è possibile leggere dai diagrammi dei casi d'uso, un utente che visita il sito può solamente visualizzare dati ed informazioni in diversi formati; non sono presenti funzioni di registrazione, modifica o cancellazione dati.
Durante la progettazione dell'interfaccia l'obiettivo principale è stato quello di organizzare gli elementi nel modo più semplice possibile.
Tutte le catture schermata riportate sono state effettuate durante lo sviluppo, per questo motivo non definitive

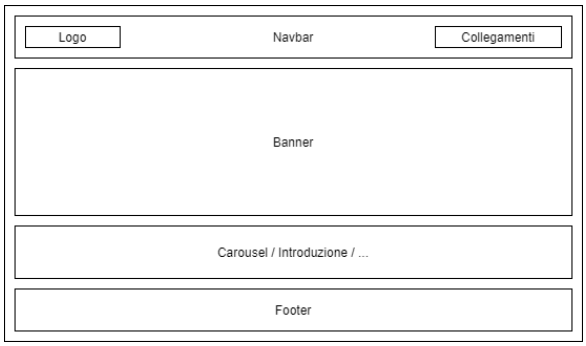


Figure 3: Progettazione Homepage

Figure 4: Homepage

Il visitatore viene accolto con una semplice barra di navigazione (presente in tutte le pagine) ed un banner di introduzione grande quanto lo schermo. Scorrendo verso il basso si è pianificata la presenza di diversi elementi come uno "slideshow" con i dati COVID-19 giornalieri, un blocco di testo con un introduzione al progetto etc...
Al fondo di ogni pagina è presente il footer che riporta semplici informazioni

come i nomi degli autori del progetto.

Nella schermata dedita alla rappresentazione dei dati in formato grafico si è pensato di dedicare la maggior parte dello spazio proprio al grafico stesso.

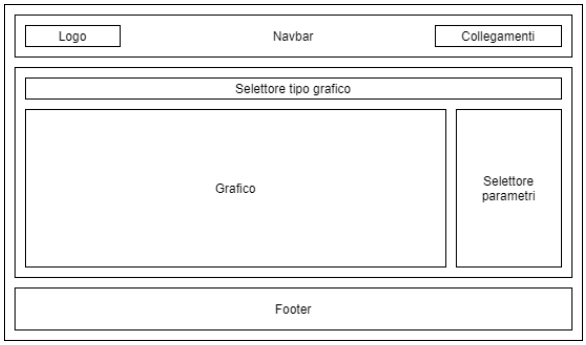


Figure 5: Progettazione Grafico

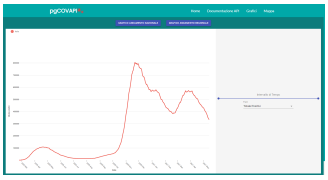


Figure 6: Grafico

Essendo l'interfaccia sviluppata per sfruttare al meglio la nostra API, si è dedicata parte della schermata ai diversi selettori che permettono di applicare filtri e selezionare quali dati visualizzare.

Per l'interfaccia con il compito di mostrare la mappa il pensiero è stato analogo, con la sola differenza che i selettori sono stati rimossi non essendoci alcuna necessità di selezionare la regione/provincia da visualizzare o un intervallo di tempo.

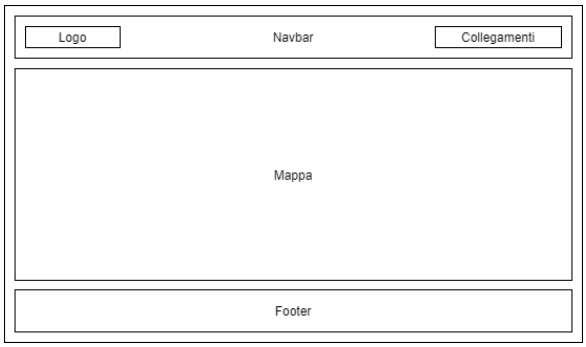


Figure 7: Progettazione Mappa

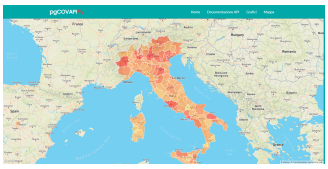


Figure 8: Mappa

4 Architettura del sistema

Il sistema informatico segue una struttura a tre livelli (*3-tier architecture*). Lo standard é il seguente:

- Client - front-end
- Server - back-end
- DBServer

Il **front-end** é la parte di sistema che si interfaccia e interagisce con l'utente. E' definito come Client tier poichè il codice risiede sulla macchina dell'utente. Nel nostro caso é implementato sotto forma di sito web in grado di rappresentare diverse informazioni in diversi formati secondo le necessità dell'utente.

Il **back-end** é invece l'insieme del codice in esecuzione su un Server che ha il compito di ricevere richieste, elaborare dati ed infine inviare una risposta al Client. Nel nostro caso il back-end è costituito dall'API, la quale dopo aver ricevuto una richiesta dal front-end da noi implementato oppure dall'applicazione di un'altro sviluppatore, interroga il database per restituire i dati richiesti.

L'ultimo tier è costituito dal **DBServer**, una macchina che esegue un servizio di gestione dati (DBMS) e ha una memoria in cui immagazzina i dati.
Sotto è rappresentata in forma grafica l'architettura del sistema. Tutte le tecnologie citate nel diagramma di deployment saranno spiegate in dettaglio nelle sezioni successive.

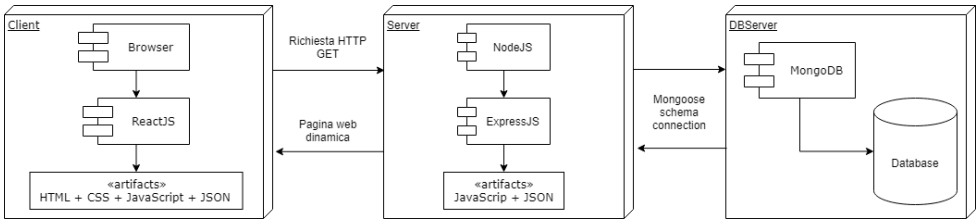


Figure 9: Diagramma di deployment

La comunicazione tra nodi verrà descritta nella sezione 4.5.

4.1 MERN Stack

La sigla **MERN** sta per MongoDB, ExpressJS, ReactJS, NodeJS e sono le quattro tecnologie principali che formano un'applicazione MERN:

- MongoDB - DBMS
- ExpressJS - Framework WEB per NodeJS

- ReactJS - Libreria JavaScript per front-end
- NodeJS - WebServer JavaScript

Il MERN stack è una delle varianti dell'originale **MEAN** stack (MongoDB, Express, Angular, Node). Esistono diversi altri stack, un esempio ne è il **MENV** (MongoDB, Express, Vue, Node).

JavaScript è un linguaggio che può essere eseguito solamente su un browser. NodeJS è un ambiente di esecuzione (*runtime enviroment*) che permette di eseguire codice scritto in linguaggio JavaScript su un Server. NodeJS è largamente utilizzato per la costruzione di applicazioni web dinamiche grazie anche alla facilità di sviluppo resa possibile dalla filosofia "JavaScript everywhere", ovvero ogni sistema è realizzato solamente con JavaScript e tramite l'utilizzo dei file di dati JSON. NodeJS utilizza un'architettura ad eventi asincrona che è in grado di supportare l'implementazione di applicazioni con comunicazioni real-time come giochi multi-giocatore su browser.

4.2 MongoDB

MongoDB è un DBMS (DataBase Management System) **non relazionale**, orientato ai **documenti**.
Partendo da questa definizione, è necessario approfondire la differenza tra i classici DBMS **SQL** e i sempre più popolari DBMS **NoSQL**.

4.2.1 Differenza DBMS SQL e NoSQL

I DBMS SQL utilizzano una struttura a tabelle: tutti i record presenti all'interno di una tabella devono condividere la stessa struttura. Le tabelle sono relazionate tra di loro tramite PK (chiavi primarie) e FK (chiavi esterne), ottenendo così una struttura fissa e rigida che richiede un'attenta analisi (seguendo ad esempio regole di *normalizzazione*, ovvero le regole per mantenere un database in una condizione consistente).

alunni		
id	nome	cognome
1	Silvio	Caprara
2	Pietro	Chiartano
3	Andrea	Zhou

argomento_elaborato	
id_persona	argomento
1	Backend
2	Database
3	Frontend

Figure 10: Database relazionale

All'interno di MongoDB i dati hanno una struttura più libera. I record non sono più delle righe all'interno di tabelle ma **documenti BSON** (JSON binari)

e sono salvati all'interno di **collections**. Si perde anche la rigida struttura dei dati nelle tabelle: ogni documento può avere una struttura diversa dagli altri presenti all'interno della stessa collection. La flessibilità e l'utilizzo della struttura JSON ha fatto sì che sistemi NoSQL come MongoDB assumessero sempre più popolarità nello sviluppo di applicazioni (sono infatti stati citati gli stack MENV MERN e MEAN, tutti i quali utilizzano MongoDB).

4.2.2 Diversi sistemi NoSQL

I database NoSQL possono essere categorizzati in:

- Database a documenti
- Database a grafo
- Database chiave-valore
- Database orientati a colonne

Database a documenti: Come già spiegato, i database a documenti come MongoDB immagazzinano i dati all'interno di collections e non tabelle. I record contenuti all'interno di quest'ultime non sono righe ma documenti.
Non si ha più una stretta relazione tra collections diverse e i documenti possono avere una struttura variabile.

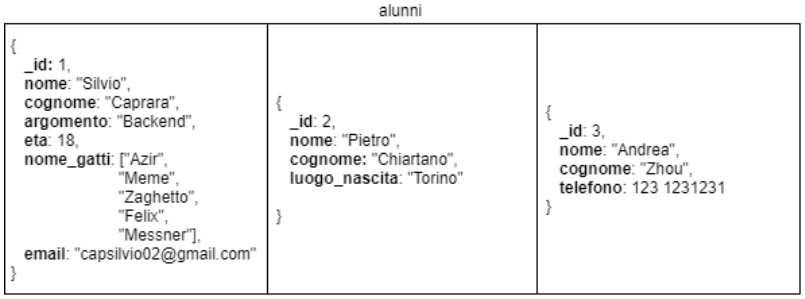


Figure 11: Database a documenti

Database a grafo: I database a grafo sono una tipologia di database NoSQL che utilizzando nodi e archi per la rappresentazione di dati. Le informazioni vengono salvate nei nodi e nei collegamenti tra nodi: se ad esempio due nodi rappresentano due diverse città, nel loro collegamento potrebbe essere memorizzata la distanza che le collega. I database a grafo presentano i seguenti vantaggi:

- Hanno uno schema entità-relazione meno rigido
- Sono più veloci dei database relazionali, soprattutto nella mappatura di applicazioni orientate agli oggetti

- Non richiedono pesanti operazioni come le **join** tra tabelle.
- Sono altamente scalabili a fronte di grandi quantità di dati.

Un comune utilizzo dei database orientati ai grafi si trova nelle applicazioni social network, nelle quali tramite gli archi che riportano le informazioni in comune tra due utenti è possibile trovare altri contatti consigliati costruendo una ”rete di conoscenti”.

Figure 12: Database a grafo

Database chiave-valore: I database chiave-valore si basano su una tabella che contiene solo due colonne: una contiene un identificatore univoco e l’altra un valore. Il valore può essere un numero intero, una stringa o persino un oggetto complesso.
Il modello chiave-valore è caratterizzato da un’elevata velocità e scalabilità: non è richiesto uno schema omogeneo e la semplice struttura permette di recuperare informazioni utilizzando una specifica chiave. Quest’ultimo fattore è anche uno svantaggio non potendo ricavare le informazioni in altri modi.
Questi database vengono utilizzati ad esempio per i carrelli di acquisto online.

Chiave	Valore
31237	"Avogadro"
31238	42
31240	{ nome: "Silvio", cognome: "Caprara" }

Figure 13: Database chiave-valore

Database orientati a colonne: I database orientati a colonne, a differenza dei classici database relazionali (orientati a righe), sono specializzati nel recupero di intere colonne di dati e vengono tipicamente utilizzati in analisi dei dati. I database a colonne presentano un numero di colonne diverso per ogni riga, potendo essere aggiunte e rimosse in base alla necessità.

"Silvio"	Cognome	Nazionalita
	"Caprara"	"ITA"
	98712	98712
"Andrea"	Cognome	
	"Zhou"	
	98713	
"Pietro"	Cognome	Altezza
	"Chiartano"	1,85
	98714	98714

Figure 14: Database orientati a colonne

4.2.3 ACID, CAP e BASE

I DBMS relazionali transazionali seguono un insieme di regole definite con l'acronimo **ACID**. Un sistema si definisce transazionale se supporta il meccanismo delle transaction, ovvero insiemi di istruzioni elementari che vengono eseguite come se fossero una singola operazione.
Per ACID si intendono le seguenti proprietà:

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity: La transazione deve essere atomica, ovvero considerata come una singola operazione. Dal punto di vista pratico, ogni transazione può terminare in due soli modi: con un **commit** oppure con un **rollback**.
Con l'istruzione di commit vengono confermate tutte le modifiche effettuate. Dopo un rollback il database viene ripristinato allo stato iniziale prima dell'inizio della transaction.

Consistency: Nei database relazionali le tabelle sono legate tra loro da chiavi primarie e chiavi esterne che prendono il nome di vincoli. Seguendo la regola della consistency, il DBMS deve gestire le transazioni in modo da non infrangere i vincoli presenti.

Isolation: Per la regola dell'isolation ogni transazione deve essere isolata dalle

altre. Nel caso due transazioni vengano eseguite in contemporanea, il risultato finale effettivo equivarrà all'esecuzione sequenziale delle due transazioni.

Durability: Dopo l'istruzione di commit, le modifiche vengono consolidate e sono durevoli nel tempo.

L'acronimo **CAP** è invece riferito più in generale ai sistemi distribuiti ma è necessario per poi spiegare il concetto di BASE. Il teorema CAP (chiamato anche teorema di Brewer) si riferisce ai sistemi di calcolo distribuiti, ovvero sistemi informatici composti da più macchine fisiche, che possono essere anche distribuite geograficamente, le quali cooperano al fine di realizzare un obiettivo comune. Le diverse macchine di un sistema distribuito prendono il nome di nodi. Le caratteristiche che un sistema distribuito ideale dovrebbe avere sono quindi:

Consistency: Ogni lettura restituisce il dato più recente o un errore.

Availability: Ogni richiesta rivolta al sistema deve ricevere una risposta che non sia una un errore. Non è però garantito che l'informazione ricevuta sia aggiornata.

Partition tolerance: Garanzia che il sistema continui a funzionare anche nel caso di guasti tra i nodi (ad esempio perdita di pacchetti).

Sempre per definizione, un sistema distribuito non può essere in grado in alcun modo di fornire tutte e tre le funzioni contemporaneamente. Ad esempio, quando si ha un errore di rete tra le partizioni è necessario decidere se cancellare l'operazione (quindi diminuire la disponibilità ma aumentare la consistenza), oppure procedere con l'operazione (garantendo la disponibilità ma rischiando inconsistenza).

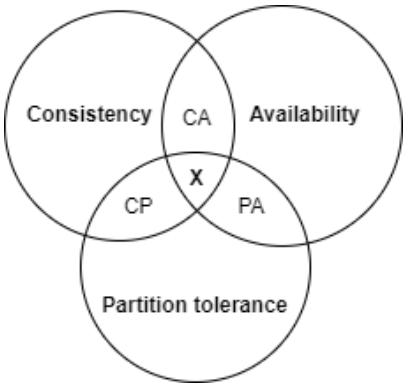


Figure 15: CAP

Il paradigma **BASE** venne pensato per sistemi distribuiti non in grado di gestire transazioni e possiede le seguenti caratteristiche:

Basically available: DBMS NoSQL come MongoDB lavorano tramite strutture **cluster**[2] e preferiscono favorire la disponibilità (availability) al posto della consistenza (consistency), per questo motivo diffondono e duplicano le informazioni tra i nodi del cluster in modo da poter sempre rispondere con una risposta che non sia un errore.

Soft state: La reale definizione di "soft state" prevede che le informazioni all'interno di un database scompaiano nel caso non vengano costantemente aggiornate. In contrapposizione con "hard state" si intende una condizione fissa, consolidata, dove i dati possono rimanere salvati *ipoteticamente* per sempre.
Da un punto di vista più concreto per "soft state" si intende che i dati possono cambiare anche senza l'intervento dell'utente. Questo accade a causa del principio di availability, dove i dati aggiornati vengono propagati tra i nodi. Sempre per questo motivo, l'utente può ricevere una risposta da un nodo che non ha i dati aggiornati ed è compito dello sviluppatore risolvere questi problemi.

Eventually consistent: La caratteristica di "eventuale consistenza" non è nient'altro che il risultato delle proprietà prima descritte. Gli aggiornamenti vengono propagati tra i nodi lentamente: questo non vuol dire però che il database non raggiungerà mai uno stato di consistenza tra i suoi nodi.

Dalla versione 4.0 rilasciata nel novembre 2018, MongoDB è anche in grado di supportare le transazioni ACID.

4.2.4 Configurazione MongoDB Atlas

MongoDB offre una comoda versione cloud del database sotto il nome di MongoDB Atlas. Oltre ad essere facilmente disponibile online, è uno dei pochi servizi che offre spazio un discreto spazio di archiviazione gratuitamente.
Dopo aver creato un profilo è possibile selezionare l'opzione "Create a new cluster" e selezionare una locazione fisica. Nel nostro caso abbiamo scelto Francoforte.

Tramite l'interfaccia web e la voce nel menù "Clusters" è possibile visualizzare ed accedere a tutti i cluster creati. Il cluster è di tipo **Replica Set - 3 nodes**, ovvero i dati vengono propagati su tre nodi diversi. Sempre dall'interfaccia grafica è possibile creare e visualizzare le collections. In questo modo abbiamo definito le collections iniziali report_regioni e report_nazioni.

[2] Un cluster è un sistema di calcolo informatico costituito da più macchine fisiche connesse tra loro che operano per la soluzione di un problema comune. La potenza di calcolo del sistema è data, teoricamente, dalla somma delle potenze di calcolo delle singole macchine.

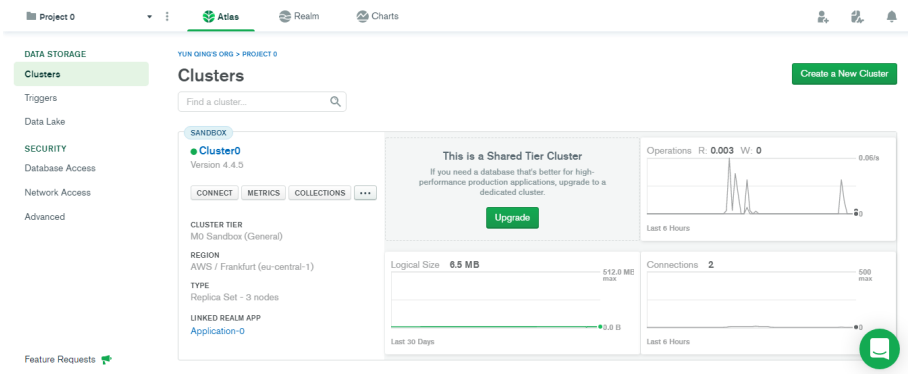


Figure 16: Interfaccia MongoDB Atlas

Vengono offerti diversi metodi per la connessione al cluster: è possibile utilizzare l'applicazione desktop **MongoDB Compass**, la CLI "**mongo shell"** e infine c'è la possibilità di ottenere i driver per connettere le proprie applicazioni (vedi Dotenv).

Una prima inizializzazione dei dati all'interno delle collection è stata fatta manualmente. Dopo aver visitato la repository Github del Dipartimento di Protezione Civile si sono scariati i file JSON contenenti l'andamento nazionale e l'andamento regionale dell'epidemia. Tramite l'applicazione Compass e la funzione drag-and-drop, trascinando i file scaricati all'interno dell'interfaccia dell'applicazione si sono inseriti i dati all'interno del database.
Dopo questa prima configurazione, attualmente i dati vengono continuamente aggiornati tramite uno script automatico (vedi Node-schedule).

I dati sono resi disponibili dalla Protezione Civile sotto i termini della licenza ***Creative Commons Attribution 4.0 International****, la quale permette la copia, redistribuzione, trasformazione e anche la produzione commerciale.*

4.3 ExpressJS

ExpressJS (o semplicemente Express) è un framework utilizzato per lo sviluppo del back-end di applicazioni web single-page o API pubbliche, rilasciato come software gratuito open-source. Il framework fornisce numerose funzionalità che facilitano l'implementazione di sistemi, fornendo allo sviluppatore metodi pronti che nascondono le complessità sottostanti. Per questo motivo, Express è diventato un *de facto standard*[3] per la realizzazione del back-end di applicazioni.

[3]Per *de facto standard* uno standard non ufficiale diventato tale grazie al grande uso del prodotto

4.3.1 Cos'è una REST API

Molto spesso gli sviluppatori definiscono le API da loro sviluppate utilizzando l'acronimo **REST** o **RESTful**.
Con REST (Representational State Transfer) si intendono una serie di proprietà da seguire durante l'implementazione di un servizio API web; non essendo un protocollo, è possibile non rispettare tutte le caratteristiche previste.

Il paradigma che si oppone all'architettura REST è il **protocollo SOAP** (Simple Object Access Protocol), un vero e proprio protocollo utilizzato per lo sviluppo di API più complesse, più sicure e dirette all'uso enterprise.
Le API SOAP non sfruttano le proprietà applicative del protocollo HTTP come i metodi GET, POST e DELETE ma lo utilizzano solo come mezzo di trasmissione. La differenza principale consiste anche nel loro scopo: le API REST forniscono risorse quando un Client invia richieste HTTP, le API SOAP sono costruite sul concetto di "chiamata di procedura remota", ovvero forniscono ad un utente un insieme di metodi richiamabili da remoto (RMI - Remote Method Invocation).

Tornando ad analizzare nello specifico il paradigma REST, un'API deve possedere le seguenti caratteristiche:

- **Client-Server**: Il servizio deve interporsi tra i due Layer separandone i ruoli
- **Stateless**: Nessun informazione deve essere salvata dal server
- **Cacheable**: Il Client può salvare le risposte del server nella cache
- **Layered system**: L'architettura a strati permette di collocare il servizio in un server e i dati in un altro server
- **Code on demand** (opzionale): I server possono permettere ai client di estendere i loro servizi ricevendo codice eseguibile
- **Uniform Interface**: Interfaccia di comunicazione omogenea tra client e server

Nell'anno 2000 Roy Thomas Fielding pubblicò le proprietà di un servizio REST in una dissertazione all'Università di Californa, Irvine, per il suo dottorato di filosofia in Information and Computer Science.
Partendo dall'articolo ufficiale, siti web che hanno pubblicato articoli su questo argomento hanno trattato il paradigma in modi differenti, a volte omettendo a volte aggiungendo caratteristiche.
Alcuni articoli definiscono come vincolo principale di un'API RESTful la proprietà **Uniform Interface**, che si sviluppa a sua volta in:

- **Manipolazione dei dati tramite una loro rappresentazione**: non si accede alle risorse direttamente ma tramite una loro rappresentazione

- **Messaggi autodescrittivi**: I messaggi che il Client riceve devono contenere le informazioni necessarie per istruire il Client stesso nell'elaborazione dell'informazione
- **Ipermedia come motore dello stato dell'applicazione**: Ogni risposta deve contenere al suo interno collegamenti per raggiungere altre risorse. Il Client in teoria dovrebbe essere a conoscenza solo dell'indirizzo root del servizio, tutte le altre risorse vengono richieste come uno spider che analizza la struttura di un sito web.
- **Identificazione delle risorse nelle richieste**: Definizione di un meccanismo per identificare quali risorse sono accessibili tramite il servizio

Altre fonti ancora citano la necessità di un servizio RESTful Web di implementare tutte le operazioni **CRUD** (Create, Read, Update, Delete) utilizzando i metodi HTTP **POST** (inviare dati al server tramite body HTTP), **GET** (richiedere dati dal server), **UPDATE** (richiedere l'aggiornamento di una risorsa) e **DELETE** (richiedere l'eliminazione di risorse).

In conclusione, la maggior parte delle API che vengono definite RESTful non lo sono, per questo motivo è corretto chiamarle semplicemente API Web.

4.3.2 Moduli utilizzati

Questa sezione descriverà i principali moduli NodeJS utilizzati per la realizzazione del back-end.

Axios: Axios è una libreria per NodeJS che permette di realizzare richieste HTTP direttamente dal codice JavaScript, sfruttando anche il meccanismo delle Promise API. Axios permette ad esempio il funzionamento dello scheduler di aggiornamento del database, potendo richiedere i file JSON della protezione civile tramite una richiesta HTTP.

```
1 Axios.get("https://pgcovapi.herokuapp.com/api/nazione")
2             .then(function (response) {
3                 //do something
4             })
5             .catch(function (error) {
6                 console.log(error);
7             })
```

ExpressJS: Express è uno dei quattro elementi fondamentali dello stack. Come già introdotto, il modulo è un "web application framework" utilizzato nello sviluppo di applicazioni single-page o di API pubbliche ed è in grado di fornire numerose funzionalità come:

- Routing efficiente
- Funzioni HTTP di redirect, caching etc...

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})
app.listen(3000)
```

Nel codice sopra riportato è possibile notare quanto sia semplice l'utilizzo di Express dopo l'installazione del modulo. Dopo aver incluso il framework e creato un oggetto **app**, partendo da quest'ultimo si possono definire le diverse route dell'API e la porta di ascolto. Importante tener presente che il codice riprodotto è solamente un esempio: un sistema costruito con ordine si presenterà in forma più complessa.

Cors: Cors è un modulo che svolge la funzione di middleware per Express e permette di abilitare il meccanismo **CORS** (cross-origin resource sharing), ovvero le richieste di risorse da parte di domini esterni. Questa dipendenza costituisce il cuore del concetto di API pubblica, permettendo anche ad altre applicazioni che risiedono su domini differenti dalla nostra applicazione di usufruire dei dati forniti dal servizio da noi implementato.

```
app.use(cors())
```

Specificando ad Express che intendiamo utilizzare cors come middleware, tutte le route saranno *CORS enabled*, ovvero raggiungibili da sorgenti esterne.

Dotenv: Dotenv è un modulo che permette di caricare le variabili poste dentro un file chiamato **.env** all'interno di process.env, una variabile globale che appartiene a NodeJS. Tramite dotenv abbiamo potuto creare una variabile che contenesse l'indirizzo per il collegamento al database MongoDB Atlas. Per motivi di sicurezza il file .env non è presente pubblicamente nella repository di Github ma si trova solamente sui computer per lo sviluppo in locale: una qualsiasi persona in possesso della variabile di connessione potrebbe accedere e modificare il database.

```
require('dotenv').config();
const uri = process.env.ATLAS_URI;
```

Per pubblicare il sito sulla piattaforma di hosting si è dovuto specificare all'interno della dashboard di controllo l'indirizzo di collegamento.

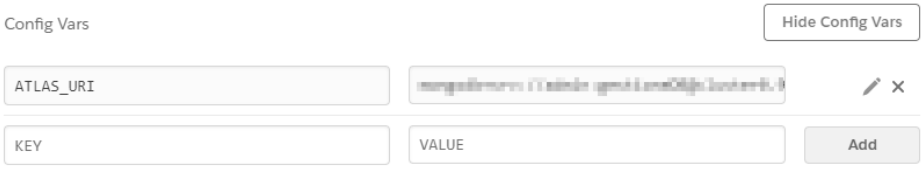


Figure 17: Heroku Dashboard

Mongoose: Mongoose è una libreria JavaScript che permette di creare modelli ("schemas") per semplificare l'interazione con il database MongoDB.
Tramite lo URI che abbiamo salvato all'interno del process.env scriviamo il codice che permetta di stabilire una connessione con il database.

```
const uri = process.env.ATLAS_URI;
mongoose.connect(uri, { useNewUrlParser: true,
                        useCreateIndex: true,
                        useUnifiedTopology: true });

const connection = mongoose.connection;
connection.once('open', () => {
    console.log(chalk.green(`Connected to MongoDB`));
});
```

In seguito si creano dei modelli che rispecchino i documenti presenti all'interno del database MongoDB. Sotto è riportato un esempio dello schema Regione.

```
const regioneSchema = new Schema({
    data: { type: String },
    stato: { type: String },
    codice_regione: {type: Number},
    denominazione_regione: {type: String},
    lat: {type: Number},
    long: {type: Number},
    codice_regione: {type: Number},
    ricoverati_con_sintomi: { type: Number },
    terapia_intensiva: { type: Number },
    totale_ospedalizzati: { type: Number },
    isolamento_domiciliare: { type: Number },
    totale_positivi: { type: Number },
    variazione_totale_positivi: { type: Number },
    nuovi_positivi: { type: Number },
    dimessi_guariti: { type: Number },
    deceduti: { type: Number },
    casi_da_sospetto_diagnostico: {},
    casi_da_screening: {},
    totale_casi: { type: Number },
    tamponi: { type: Number },
    casi_testati: { type: Number },
    note: { type: String },
    ingressi_terapia_intensiva: { type: Number },
```

```
25      note_test: { type: String },
26      note_casi: { type: String },
27      totale_positivi_test_molecolare: { type: Number },
28      totale_positivi_test_antigenico_rapido: { type: Number },
29      tamponi_test_molecolare: { type: Number },
30      tamponi_test_antigenico_rapido: { type: Number }
31  })
```

Dopo aver creato un modello, Mongoose permette un'estrema semplificazione nelle interazioni con il database.

```
1  Regione.find(query)
2          .sort({ "data": 1, "denominazione_regione" : 1})
3          .then(regione => res.json(regione))
4          .catch(err => res.status(400).json('Error: ' + err));
```

Il codice qui sopra ritorna in formato JSON tutti i dati delle regioni ordinati per data e per denominazione regione.

Node-schedule: node-schedule è un modulo che permette di programmare l'esecuzione di funzioni tramite l'utilizzo di un timer. E' possibile far eseguire una funzione ad un preciso orario oppure dopo intervalli definiti. Nel nostro caso il node-scheduler ci permette di controllare automaticamente eventuali aggiornamenti nei dati della protezione civile e riflettere le modifiche all'interno del nestro database.

```
1  const regioniJob =
2  scheduler.scheduleJob('00 00 * * * *',  scripts.updateRegioni);
```

Nodemon: Nodemon è un modulo utilizzato durante lo sviluppo di applicazioni NodeJS, il quale permette di aggiornare l'applicazione ogni volta che vengono rilevate delle modifiche all'interno dei file.

4.3.3 Implementazione del codice

Per avviare in modo ottimale lo sviliuppo di un'applicazione NodeJS è consigliato utilizzare il comando shell **npm init** all'interno della cartella del progetto vuota.

```
1  npm init
```

Il comando è un potente strumento che guida lo sviluppatore nella compilazione del file **package.json**. Il documento package.json contiene varie informazioni sul progetto come il nome, la licenza, gli autori...

```
1  {
```

```
2   "name": "pgcovapi",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "dependencies": {
7     "axios": "^0.21.1",
8     "cors": "^2.8.5",
9     "dotenv": "^8.2.0",
10    "express": "^4.17.1",
11    "mongoose": "^5.12.5",
12    "node": "^15.14.0",
13    "node-schedule": "^2.0.0",
14    "nodemon": "^2.0.7",
15  },
16  "scripts": {
17    "start": "node server.js",
18  },
19  "author": "Caprara Silvio, Chiartano Pietro, Zhou Yun Qing",
20  "license": "ISC"
21 }
```

Il package.json è anche il cuore della gestione delle dipendenze per NodeJS. Ogni volta che si esegue il comando **npm install** per installare una dipendenza, il nome e la versione di quest'ultima vengono aggiunte nell'oggetto **dependencies**. Quando è necessario lavorare al progetto da un'altra postazione, al posto di importare da un archivio cloud tutta la cartella **node_modules**, il comando npm install utilizzato senza nessun parametro installerà tutte le dipendenze presenti all'interno del package.json. Proprio grazie a questa funzione tutta la cartella contenente i moduli NodeJS non è pubblicata nella repository Github.

In seguito si sono installati tutti moduli necessari che sono già stati spiegati nella sezione precedente, esempio:

```
1 npm install express
```

Il file principale del back-end è **server.js**, nel quale si configura ExpressJS e si definisce il middleware CORS.

```
1 const app = express();
2 const port = process.env.PORT || 5000;
3 app.use(cors());
4 app.use(express.json());
```

Successivamente si instaura la connessione a MongoDB Atlas tramite Mongoose utilizzando il driver salvato nel file .env.

```
1 const uri = process.env.ATLAS_URI;
2 mongoose.connect(uri, { useNewUrlParser: true,
3                         useCreateIndex: true,
```

```
4                         useUnifiedTopology: true });
5
6 const connection = mongoose.connection;
7 connection.once('open', () => {
8     console.log(chalk.green(`Connected to MongoDB`));
9 });
```

L'obiettivo finale è inviare all'utente informazioni quando effettua una richiesta inserendo un URL (Unified Resource Locator) all'interno della barra di ricerca del browser.

Figure 18: URL Route nazione

/api/nazione è il nome del percorso che abbiamo deciso di definire. Prima di scrivere il codice è stato quindi necessario definire le convenzioni a cui attenerci per la nomenclatura delle diverse rotte.

Innanzitutto, tutte le rotte devono essere precedute da **/api/** per differenziare il servizio API che invia dati in formato JSON dal percorso **pgcovapi.herokuapp.com/** che invierà all'utente la pagina WEB.

Tutto questo discorso è più comprensibile se viene definito il concetto di **single-page application**. Utilizzando tecnologie più comuni come HTML e php quando si parla di diverse rotte si presuppone che esistano determinati file in determinate cartelle. Ad esempio, se consideriamo il percorso *sito.com/immagine/immagine.png*, vorrà dire che a partire dalla cartella principale del sito esisterà una cartella di nome "immagine" che conterrà un'immagine nominata "immagine.png".
In una single page application questo discorso non è valido. Se definiamo la route /api/nazione non è necessario che esista una cartella "api" contenente un file "nazione". Tutto lo URL diventa quindi una grande variabile che è possibile utilizzare per creare condizioni nel codice.

Continuando l'esempio iniziale del percorso **/api/nazione**, è necessario creare il file contenente il codice che gestirà richieste, filtrerà i dati e li resituirà in formato JSON. Come in ogni documento vanno importati i moduli necessari:

```
1 const express = require('express');
2 const router = express.Router();
3 let Nazione = require("../../models/nazione.model"); //MongooseModel
```

In seguito si descrivono i comportamenti dell'API in base al percorso scritto dall'utente. Solitamente l'implementazione inizia dalla route **root** (radice), ovvero '/'.

```
router.route('/').get((req, res) => {
    //something
});
```

Al fine di arricchire le funzioni dell'API, sono stati implementati diversi filtri come la possibilità di richiedere i dati di un preciso mese o definire una data di partenza e di fine. E' possibile anche ricvere una risposta "alleggerita" specificando quale campo dei documenti si vuole ricevere.

I parametri di un'API si possono definire **in-path** oppure **in-query**.
I parametri in-path vanno inseriti nel percorso:

```
//pgcovapi.herokuapp.com/api/regioni/{regione}
pgcovapi.herokuapp.com/api/regioni/Piemonte
```

I parametri in-query invece vengono inseriti al fine dello URL ed il loro utilizzo è consigliato nel caso i parametri siano opzionali:

```
pgcovapi.herokuapp.com/api/nazione/?campo=totale_positivi
```

La possibilità di avere parametri opzionali è particolarmente efficace quando si va a costruire dinamicamente una richiesta dall'API al database.
Prima di tutto vanno ricavati i diversi parametri presenti nella richiesta dell'utente tramite l'oggetto **req**:

```
let pMese = req.query.mese || null;
let param = req.query.campo || null;
let days = req.query.giorni || null;
let startDate = req.query.dataInizio || null
let endDate = req.query.dataFine || null;
```

I parametri in-query risiedono in **req.query** ed in seguito il nome del parametro che l'utente ha inserito. Nel caso il parametro sia in-path bisogna ricavare il valore da **req.params** ed in seguito sempre il nome del parametro.
Dopo aver ricavato i diversi valori (oppure **null** nel caso l'utente non abbia utilizzato il parametro), si costruisce dinamicamente la query.

```
let query = {};
if (pMese) {
    //spMese[0] = Anno
    //spMese[1] = Mese
    let spMese = pMese.split('-');
    query.data = { $gte: spMese[0] + "-" + spMese[1] + "-00",
                   $lte: spMese[0] + "-" + spMese[1] + "-31" }
}
```

Le query possono essere costruite dinamicamente sempre sfruttando le funzioni offerte da Mongoose. Si inizia quindi definendo un oggetto vuoto e generico: "query". Il parametro pMese andrà a filtrare i dati in base alla data presente nel campo "data" di ogni documento. Per questo motivo qualunque sarà il filtro che andremo a definire andrà assegnato a **query.data**.
Nel codice sopra riportato il valore del mese che è inserito in un formato mm-yyyy viene diviso e manipolato al fine di ottenere dal primo all'ultimo giorno del mese.

Nel caso invece si sia utilizzato il parametro startDate per definire una data di inizio nei documenti codice sarebbe il seguente:

```
query.data = { $gte: startDate }
```

Si assegna quindi sempre a query.data un filtro che ricerchi tutti i documenti con data uguale o maggiore alla data inserita (in questo caso in formato gg-mm-yyyy).

Dopo aver costruito tutta la query si utilizza il modello Mongoose importato all'inizio del documento per effettuare la richiesta, utilizzando come parametro del metodo .find l'oggetto query contenente i filtri appena creati.

```
Nazione.find(query)
        .sort({ "data": 1 })
        .select(param)
        .then(nazione => res.json(nazione))
        .catch(err => res.status(400).json('Error: ') + err);
```

Dopo aver implementato il codice di gestione della richiesta bisogna tornare al file principale server.js e specificare ad express come comportarsi quando riceve una richiesta ad un dato percorso.

Per prima cosa si include il file che abbiamo appena scritto.

```
const nazioneRouter = require('./routes/api/nazione');
```

ed in seguito si specifica ad express la condizione per utilizzare quella route, ovvero quando l'utente scrive **/api/nazione** dopo l'URL del sito.

```
app.use('/api/nazione', nazioneRouter);
```

In questo modo si è creata la prima rotta dell'API. In seguito sono state implementate le rotte più complesse **/api/regioni** e **/api/province**.

Uno sviluppatore può leggere l'intero elenco di parametri disponibili utilizzando

la documentazione WEB.

4.3.4 Documentazione WEB

Le documentazioni WEB sono uno dei modi più semplici per presentare le diverse funzioni di un'API o di una libreria.

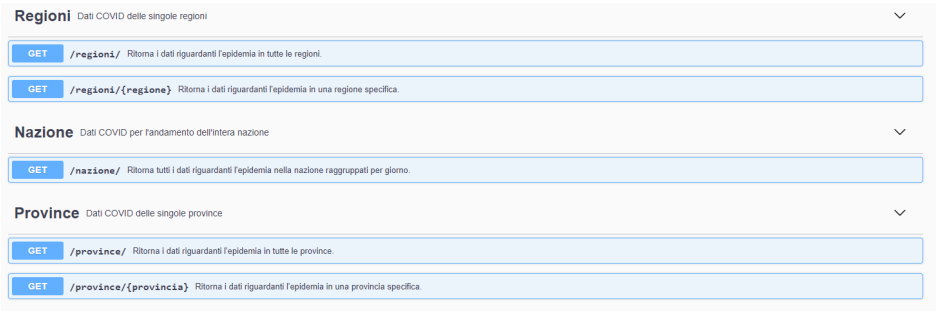


Figure 19: SwaggerUI Documentation

Anzichè essere semplici documenti PDF, sfruttano tutte le proprietà delle pagine web moderne: interattività e dinamicità.
Da questa pagina pubblicata online tramite le Github Pages (raggiungibile dalla repository Github del progetto) è possibile visualizzare tutte le diverse route, leggere la descrizione di tutti i parametri e la struttura di tutti i documenti presenti nel database.
Il framework utilizzato, **SwaggerUI**, permette di generare queste pagine dinamiche partendo da un file di configurazione .json nel quale è necessario seguire una precisa struttura comprensibile dal tool (a sua volta documentata nel sito ufficiale Swagger.io/docs). Sulla pagina è anche possibile compilare e provare le diverse rotte tramite il tasto **Try it out**.

4.4 ReactJS

ReactJS è una libreria open-source JavaScript creata da Facebook utilizzata per la realizzazione del lato front-end di applicazioni Web. React permette di realizzare single-page applications: come già descritto per le route dell'API, lo URL presente nella barra di ricerca del browser non corrisponde ai documenti del sito bensì diventa anche qui una variabile che ci permette di decidere quali **componenti** andare a **renderizzare**.

```
1 const App = () => {
2 return (
3 <>
4 <Router>
5     <Navbar />
```

```
6       <Switch>
7           <Route path='/' exact>
8               <Banner />
9               <DataCarousel/>
10          </Route>
11
12          <Route path='/grafici'>
13              <ChartContainer />
14          </Route>
15
16          <Route path='/mappa'>
17              <Map />
18          </Route>
19      </Switch>
20      <Footer />
21  </Router>
22  </>
23  )}
```

Anche React utilizza un **Router** per decidere quali componenti mostrare all'utente. Nel codice sopra riportato quando si richiede il percorso "radice" ('/') del sito, verranno renderizzati i componenti **Banner** e **DataCarousel**.

I componenti sono la base di React e sono costituiti da blocchi di codice JavaScript ed elementi HTML (più correttamente **JSX**, una fusione tra HTML e JavaScript). Uno dei vantaggi di React è la possibilità di riutilizzare i diversi componenti: è quindi importante implementare il codice nel modo più generico possibile.

La flessibilità e velocità sono proprietà di React garantite dall'utilizzo di un **DOM virtuale**. Tutto il codice scriveremo verrà iniettato all'interno di un singolo elemento HTML presente nell'unico file HTML nel progetto: index.html.

```
1  //index.html
2  <div id="root"></div>
```

```
1  //index.js
2  ReactDOM.render(
3    <App />,
4    document.getElementById('root')
5  );
```

Come è possibile vedere dal codice, l'intero componente **App** verrà inserito dentro il div HTML con identificativo "root".
La possibilità di decidere quali componenti renderizzare alleggerisce e velocizza notevolmente la navigazione nel sito (nei siti realizzati con HTML gli elementi vengono nascosti tramite proprietà CSS come *display: none* ma sono sempre presenti nel DOM e possono rallentare la pagina).

La dinamicità di React si ottiene tramite lo **state**. Ogni componente è dotato dalla sua creazione di un oggetto state. Questo oggetto può contenere diverse strutture di dati: stringhe, numeri interi, oggetti e array. Ogni volta che lo state di un componente viene aggiornato, l'intero componente viene ricaricato. Per questo motivo React viene definito come un sistema "data-driven", ovvero guidato dal cambiamento dei dati.

Nel nostro caso lo state viene spesso utilizzato per aggiornare i componenti dopo l'arrivo delle risposte dall'API. Quando si richiede la visualizzazione del componente che mostra il grafico, viene effettuata una richiesta all'API e allo stesso tempo mostrato il box del grafico. Essendo JavaScript asincrono, il grafico viene renderizzato indipendentemente dal fatto che la risposta contenente i dati sia arrivata o no. Per questo motivo è necessario che il grafico faccia riferimento ai dati presenti nello state e che la risposta al suo arrivo venga salvata nello state. In questo modo il grafico verrà inizialmente mostrato vuoto e verrà aggiornato non appena arriveranno i dati.

4.5 Comunicazione Client-Server

La comunicazione inizia quando un Client effettua una richiesta HTTP/HTTPS al WebServer di Heroku dove avviene l'hosting dell'applicazione Web. In base all'URL della richiesta il Server deciderà quale comportamento assumere.

```
1  //Back-end Routes
2  app.use('/api/regioni', regioniRouter);
3  app.use('/api/nazione', nazioneRouter);
4  app.use('/api/province', provinceRouter)
5  app.use('/api/', rootRouter);
6
7  //Front-end
8  app.get("*", (req, res) => {
9      res.sendFile(path.join(__dirname, "client", "build", "index.html"));
10 });
```

Nel caso la richiesta sia diretta verso l'API, il Server invierà una richiesta al DBServer (MongoDB Atlas) tramite il modulo Moongoose, ricevendo in risposta i dati in formato JSON che verranno a loro volta restituiti al Client.
Se la richiesta è invece rivolta verso la root del sito (pgcovapi.herokuapp.com/), il Server invierà i file statici che compongono l'interfaccia grafica dell'applicazione, ovvero il file index.html nel quale è iniettato da React il DOM Virtuale e tutti i suoi componenti React e file CSS. Per mostrare i dati in forma grafica il front-end dovrà in ogni caso effettuare le dovute richieste all'API.

5 Architettura di rete

In accordo con il tema del progetto, si è immaginato di dover realizzare una rete aziendale per la Protezione Civile.
La rete sarà costruita su un solo piano e prevederà una zona per gli impiegati, un ufficio amministrazione e un locale protetto nel quale saranno collocati i Server. Si considera inoltre la presenza di una sala riunioni nella quale ospiti e collaboratori potranno collegarsi con dispositivi mobili.

5.1 Schema logico e piano di indirizzi

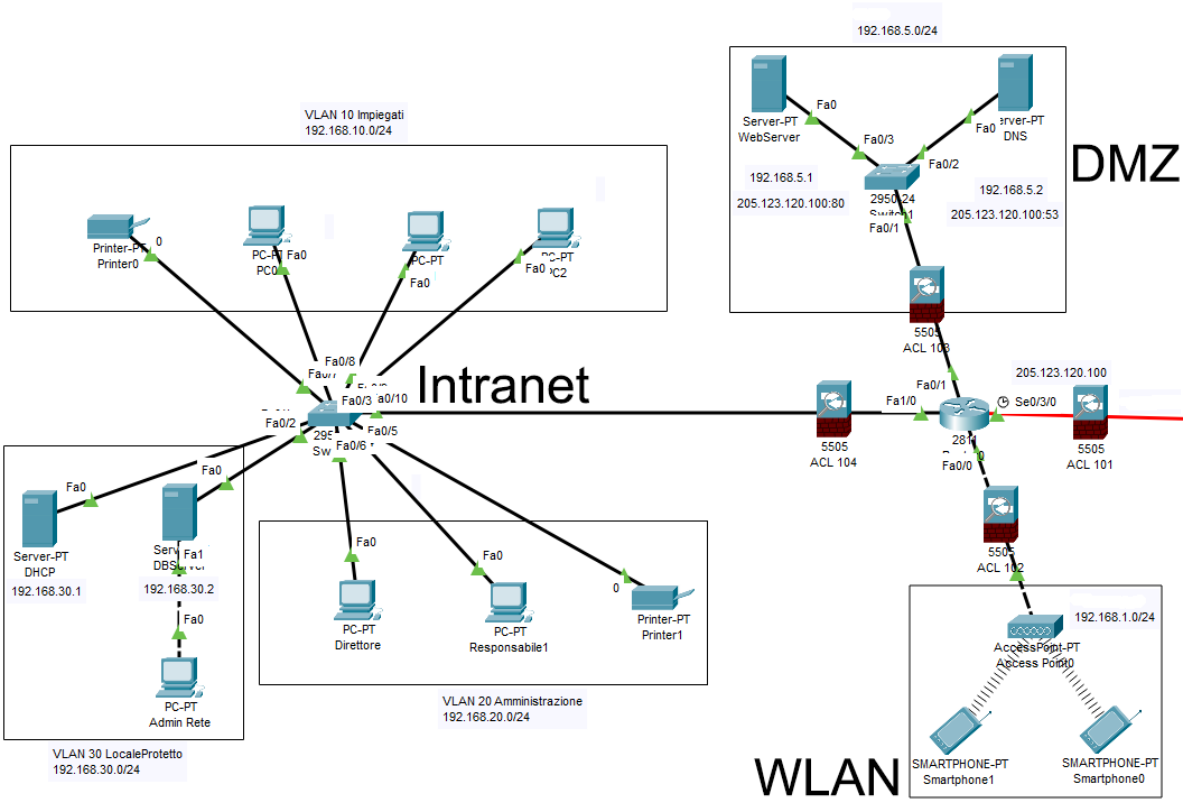


Figure 20: Schema logico

La topologia della rete è a stella. Il nodo centrale è il router che si interfaccia all'esterno (**Internet**) e divide la rete aziendale in tre diverse parti.
La prima è la **intranet**, ovvero la rete interna che comprende i dispositivi dei lavoratori e i Server che non necessitano di comunicare con l'esterno.

La seconda parte è costituita dalla **DMZ** (Demilitarized Zone), una rete "cuscinetto" che ha la una funzione di ricevere il traffico dall'esterno della rete, in modo tale che le richieste non arrivino direttamente alla intranet rischiando l'ingresso di pacchetti maliziosi. Inoltre, all'interno della DMZ sono posti i Server che devono offrire servizi non solo ai dipendenti dell'azienda ma anche a utenti pubblici, ad esempio il Server con il compito di fornire le pagine web.
Infine, il router è collegato tramite cavo ad un Access Point realizzando una WLAN (Wireless Local Area Network), ovvero la rete a cui possono collegarsi tramite Wi-Fi collaboratori e ospiti.
Il piano di indirizzi è il seguente:

Nome	Indirizzo IP/CIDR	Indirizzo IP Pubblico/CIDR	Gateway
PC0	DHCP (VLAN10)	205.123.120.100:(PAT)/24	DHCP (VLAN10)
PC1	DHCP (VLAN10)	205.123.120.100:(PAT)/24	DHCP (VLAN10)
PC2	DHCP (VLAN10)	205.123.120.100:(PAT)/24	DHCP (VLAN10)
Printer0	DHCP (VLAN10)	205.123.120.100:(PAT)/24	DHCP (VLAN10)
Direttore	DHCP (VLAN20)	205.123.120.100:(PAT)/24	DHCP (VLAN20)
Responsabile1	DHCP (VLAN20)	205.123.120.100:(PAT)/24	DHCP (VLAN20)
Printer1	DHCP (VLAN20)	205.123.120.100:(PAT)/24	DHCP (VLAN20)
DHCP	192.168.30.1/24	/	/
DBServer	192.168.30.2/24	/	/
WebServer	192.168.5.1/24	205.123.120.100:80/24	192.168.5.254/24
DNS Server	192.168.5.2/24	205.123.120.100:53/24	192.168.5.254/24
Smartphone0	DHCP (WLAN)	205.123.120.100:(PAT)/24	DHCP (WLAN)
Smartphone1	DHCP (WLAN)	205.123.120.100:(PAT)/24	DHCP (WLAN)
Router0 Fa1/0.10	192.168.10.254	/	/
Router0 Fa1/0.20	192.168.20.254	/	/
Router0 Fa1/0.30	192.168.30.254	/	/
Router0 Fa0/1	192.168.5.254/24	/	/
Router0 Fa0/0	192.168.40.254/24	/	/
Router0 Se0/3/0	205.123.120.100/24	/	/

I servizi citati in tabella verranno descritti nella sezione successiva.

5.2 Protocolli, servizi e tecniche di sicurezza

All'interno della rete sono stati configurati servizi e protocolli che ne garantiscono il funzionamento e la proteggono con meccanismi di sicurezza.
Facendo riferimento alla disposizione fisica indicata nell'introduzione, dopo aver piazziato i dispositivi all'interno della configurazione le prime tecnologie configurate sono state le **VLAN**.

5.2.1 VLAN

Le VLAN (Virtual Local Area Network) sono un inseme di tecnologie che permettono di dividere il dominio di broadcast[4] di una rete in più sotto-reti virtuali. Le reti fittizie create in questo modo condividono la stessa configurazione fisica

[4]Quando si invia un messaggio all'interno di una rete in modalità **broadcast**, il messaggio viene inviato a tutti i dispositivi all'interno della rete. Il dominio di broadcast è quindi l'insieme di dispositivi che riceverebbero il messaggio.

ma perdono la possibilità di comunicare tra di loro.

Il protocollo VLAN si configura sul nodo centrale della intranet, ovvero lo switch. Lo switch al suo interno ha un database in cui registra le diverse VLAN configurate. La tabella ha sole due colonne: il nome della VLAN ed il suo codice identificativo, un valore che va da 0 a 4096.
E' necessario scegliere in ogni interfaccia in utilizzo sul dispositivo una delle due modalità esistenti per il suo funzionamento: Access Port o Trunk Port. Definiamo **Access Port** una porta che collega un dispositivo finale al nodo che si occupa di gestire le VLAN. Il collegamento prende il nome di **Access Link**. Per configurare una porta access è necessario selezionare l'interfaccia, la modalità e la VLAN di apparteneza.

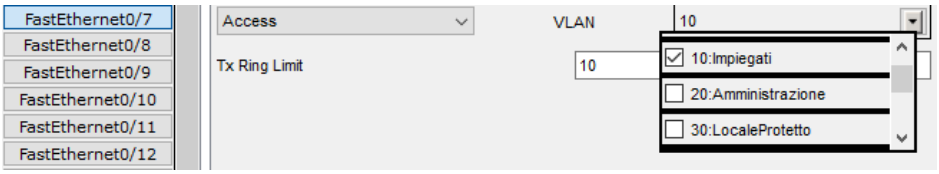


Figure 21: Access Link

Si definisce **Trunk Link** il collegamento che permette al suo interno il passaggio di pacchetti provenienti da VLAN diverse. Per permettere la creazione di collegamenti trunk viene utilizzato il protocollo **802.1Q** il quale fa sì che, al passaggio del pacchetto da un access link ad un trunk link, lo switch aggiunga al pacchetto un header[5] di protocollo 802.1Q (4 byte) che identifica la VLAN di provenienza dei pacchetti. La struttura dell'header è la seguente:

TPI - Tag Protocol Identifier (2 Byte)

- EtherType (2 Byte) - Valore impostato a 0x8100 il quale evidenzia che il pacchetto ha l'header 802.1Q

TCI - Tag Control Information (2 Byte)

- User Priority / Priority Code Point (PCP) (3 bit) - Valore che indica il livello di priorità del frame
- Drop Eligible Indicator (DEI) (1 bit) - I frame segnalati con questo flag possono essere ignorati in caso di congestione della rete
- VLAN ID (VID) (12 bit) - Campo che identifica l'ID delle VLAN il quale può assumere un valore che va da 0 a 4096.

[5]Si definisce header una parte del pacchetto che contiene informazioni di controllo necessarie al funzionamento della rete

All'interno della nostra rete tutti i collegamenti sono access tranne il cavo che collega il router allo switch. Per realizzare un trunk link è necessario selezionare l'interfaccia, la modalità trunk e tutte le VLAN che avranno il permesso di attraversare il canale.

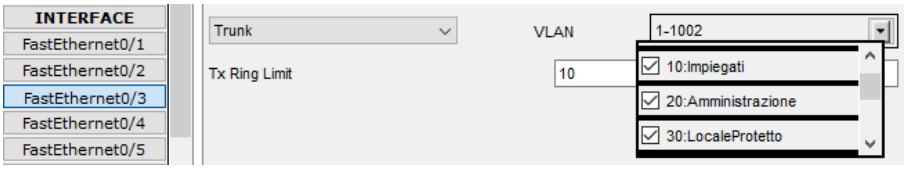


Figure 22: Trunk Link

I vantaggi delle VLAN sono quindi:

- Riduzione delle dimensioni dei domini di broadcast
- Semplificazione della gestione dei dispositivi (con anche la possibilità di riassegnare le porte)
- I dispositivi vengono collegati logicamente superando le limitazioni fisiche
- Possibilità di dividere logicamente i dipartimenti o i team di progetto

Nel caso si voglia permettere la comunicazione tra VLAN differenti esiste una tecnica che prende il nome di **Router on-a-stick**, in grado di modificare il VLAN ID presente all'interno dell'header di protocollo 802.1Q. Aprendo la CLI (Command Line Interface) del router si devono creare tante **sotto interfacce** quante sono le VLAN in grado di attraversare il collegamento trunk, nel nostro caso tre. Le sotto interfacce sono solamente logiche ma è possibile applicare ad ognuna di esse una diversa configurazione.
Sono riportati sotto tutti i comandi che sono stati utilizzati per configurare una delle sotto interfacce.

```
1 Router0(config)#interface FastEthernet1/0.10
2 Router0(config-subinf)#ip address 192.168.10.254 255.255.255.0
3 Router0(config-subinf)#encapsulation dot1q 10
```

Per ogni sotto interfaccia è necessario assegnare un indirizzo IP, la maschera di rete e specificare che si intende utilizzare il protocollo 802.1Q. I passaggi vanno ripetuti per le sotto-interfacce .20 e .30 (i numeri assegnati sono stati scelti per comodità, in modo che rispecchino l'identificativo delle VLAN).
Da questo momento, un pacchetto che parte dal PC del direttore appartenente alla VLAN 20 potrà comunicare con i PC della VLAN 10 passando per il router, dove quest'ultimo cambierà il VLAN ID contenuto nel pacchetto.
Per iniziare a testare il funzionamento della rete è necessario configurare i dispositivi con i loro indirizzi IP. Per non doverlo fare manualmente, è stato utilizzato il protocollo **DHCP**.

5.2.2 DHCP

Il protocollo DHCP (Dynamic Host Configuration Protocol) è stato utilizzato per configurare dinamicamente i dispositivi, assegnando loro un indirizzo di rete, una maschera di rete e altri parametri come il default-gatway e l'indirizzo del Server DNS di riferimento (il protocollo DNS verrà spiegato in seguito). Un dispositivo può quindi utilizzare una configurazione statica (assegnata da un amministratore di rete), oppure può chiedere la configurazione ad un Server DHCP.
Il servizio è stato configurato tramite l'interfaccia grafica di PacketTracer.

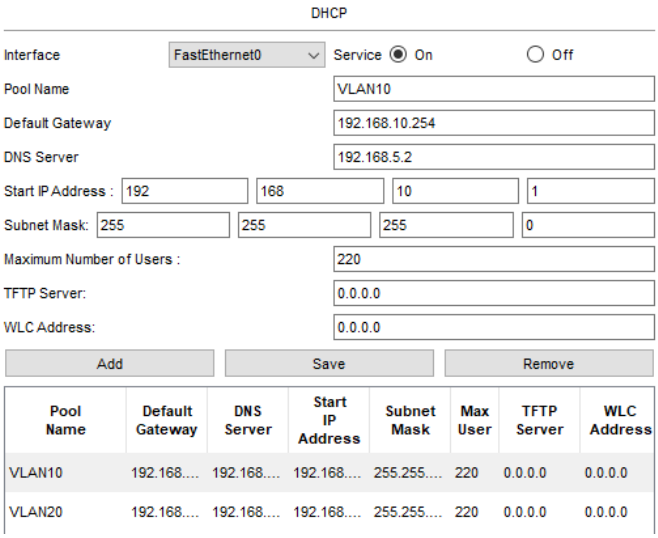


Figure 23: DHCP

Un singolo Server DHCP può fornire configurazioni diverse salvate in **pool** diverse. Ogni VLAN ha bisogno di una pool di indirizzi diversa: nel nostro caso sono state create le pool "VLAN10" e "VLAN20". Non è stata creata una pool per la VLAN30 essendo utilizzata solo per separare i Server, i quali necessitano di indirizzi statici in modo da essere sempre identificabili.
Facendo riferimento all'immagine, nella pool VLAN10 (impiegati) gli indirizzi partono dal 192.168.10.1 e hanno una maschera di rete 255.255.255.0. Il loro indirizzo di default-gateway corrisponde con la sotto interfaccia del router prima configurata e il loro Server DNS di riferimento è quello posto nella DMZ di indirizzo 192.168.5.2.

Per richiedere una configurazione ad un Server DHCP avvengono i seguenti passaggi:
1) Il Client (prendiamo per esempio PC0 all'interno della VLAN 10) invia in broadcast un messaggio di **DHCP Discover**, utilizzato per ricercare un Server

DHCP all'interno della rete. Il messaggio di discover contiene anche tutti i parametri di cui il dispositivo necessita. Qui sorge il primo problema: il pacchetto non può raggiungere il Server DHCP se si trova all'interno di un'altra rete logica. Una volta raggiunto il router, quest'ultimo scarterà il pacchetto mandano il messaggio di errore: "Nessun servizio DHCP configurato su questo dispositivo". E' quindi necessario istruire il Server ad inoltrare il pacchetto verso il Server DHCP al posto di scartarlo. Il comando IP helper-address svolge propria questa funzione:

```
1 Router0(config)#interface FastEthernet1/0.10
2 Router0(config-subinf)#ip helper-address 192.168.30.1
```

Come è possibile vedere dai comandi l'helper-address va applicato alle sotto-interfacce, quindi andrà assegnato anche alla sotto interfaccia .20.

2) Il Server, dopo aver ricevuto il messaggio di DHCP discover, risponde con una **DHCP Offer**. Il messaggio di offerta contiene tutti i parametri che il Client aveva richiesto.

3) Il Client riceve e analizza l'offerta del Server DHCP. E' possibile che nella rete siano presenti diversi Server DHCP, per questo motivo il Client deve scegliere una delle configurazioni e inviare una **DHCP Request** in broadcast. Tutti i messaggi DHCP contengono un ID di transazione: i Server che riceveranno la DHCP request capiranno di essere stati scelti o no tramite questo identificativo. Nel caso l'ID sia diverso da quello inviato nell'offerta, il Server chiude la comunicazione.

4) Il Server DHCP invia una **DHCP ACK** (Acknowledge) di conferma con i parametri richiesti.

Dopo questo scambio in quattro fasi, il dispositivo inizia ad utilizzare la configurazione ricevuta. Da questo momento è possibile selezionare DHCP nella schermata di configurazione di ogni dispositivo per avere una rete con dispositivi funzionanti.

Figure 24: DHCP PC 0

Anche i dispositivi mobili degli ospiti e collaboratori necessitano di una configurazione DHCP per poter navigare in Internet. Permettere l'ingresso dei pacchetti DHCP all'interno della intranet non è una buona scelta dal punto di

vista della sicurezza. Per il motivo appena citato e sfruttando la possibilità di poter configurare il protocollo anche su un router, è stata creata una DHCP pool anche nel Router0.

```
1 Router0(config)#ip dhcp pool WLAN
2 Router0(dhcp-config)#network 192.168.40.1 255.255.255.0
3 Router0(dhcp-config)#default-router 192.168.40.254
4 Router0(dhcp-config)#dns-server 192.168.5.2
```

Dopo aver concluso la configurazione degli elementi presenti della intranet, si sono attivati i servizi dei Server nella DMZ.

5.2.3 Web Server

Il ServerWeb ha il compito di fornire pagine web in formato HTML (Hyper Text Markup Language) quando riceve richieste HTTP (Hyper Text Transfer Protocol) alla porta tcp 80. Nel contesto del nostro progetto, il WebServer sarà il dispositivo che eseguirà il back-end dell'architettura e fornirà all'utente la pagina web dinamica o i dati in formato JSON.
Un pacchetto HTTP ha la seguente struttura:

Start-line:

- Metodo: Il metodo determina il tipo di richiesta che può essere: GET (unico metodo utilizzano nel nostro caso per richiedere informazioni), HEAD, POST, PUT, DELETE, CONNECT, TRACE.
- Path-to-file: Il path to file determina, partendo dal dominio del sito, il percorso da seguire per raggiungere la risorsa.
- Versione HTTP: La versione HTTP utilizzata attualmente è la 1.1. A differenza della precedente versione 1.0, la 1.1 permette di migliorare le prestazioni di una comunicazione lasciando aperto il canale TCP per richiedere tutte le risorse necessarie, al posto di instaurare una nuova comunicazione per richiedere ogni risorsa.

Header:

- Tipo di documento: HTML, CSS etc...
- Tipo di browser utilizzato
- Data della richiesta
- etc..

Body

- Il body è utilizzato nelle richieste POST per inviare dati al Server.

Per configurare il servizio HTTP è stato sufficiente assegnare un indirizzo statico al Server e abilitare dall'interfaccia grafica il protocollo HTTP. Dalla stessa schermata è possibile anche attivare il protocollo HTTPS, ovvero la versione sicura di HTTP che utilizza i protocolli SSL/TLS.

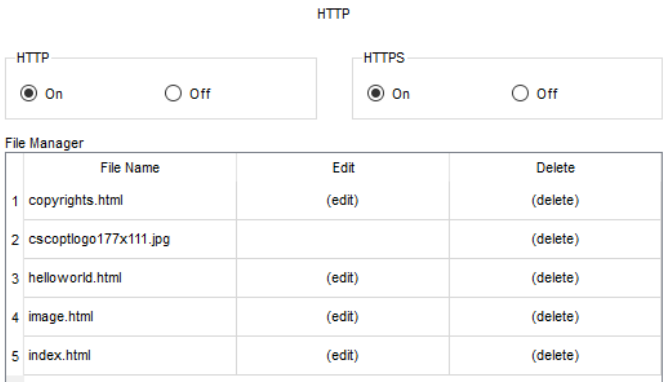


Figure 25: HTTP

5.2.4 HTTPS (SSL/TLS)

Il protocollo HTTPS (Secure Hyper Text Transfer Protocol) è la versione "sicura" del protocollo HTTP. HTTPS utilizza i protocolli SSL (Secure Sockets Layer) e TLS (Transport Layer Security), dove l'ultimo è l'evoluzione del primo, per costruire un canale di comunicazione criptato dal nostro browser al Server. TLS è una suite di protocolli che operano ai livelli 4 (trasporto) e 5 (sessione) della pila ISO/OSI.
La connessione viene instaurata utilizzando il protocollo **TLS Handshake** tramite il quale il Client autentica il Server e i due nodi si accordano sui protocolli che andranno ad utilizzare per criptare e autenticare i messaggi.

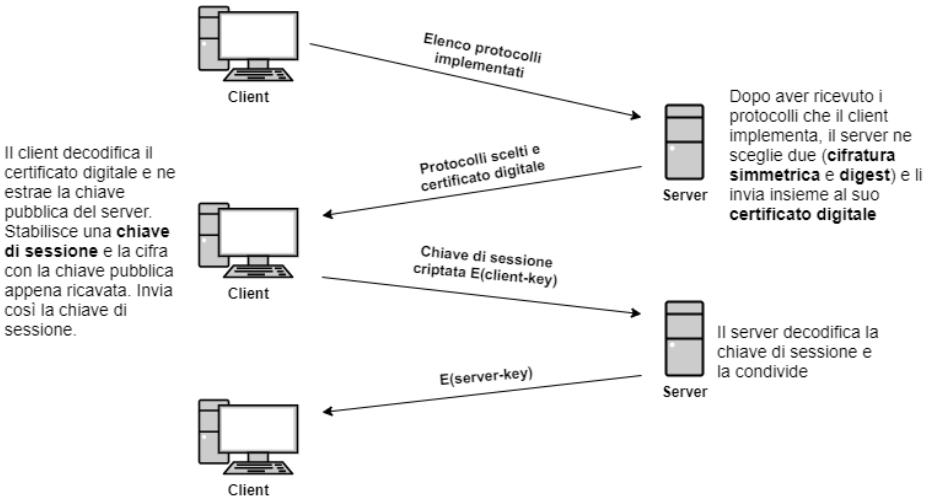


Figure 26: TLS Handshake

Dopo aver effettuato l'handshake, i protocolli stabiliti vengono passati al protocollo **TLS Record** che si occupa di cifrare e autenticare i pacchetti. I pacchetti saranno incapsulati tramite protocollo TCP.

5.2.5 DNS Server

Il protocollo DNS (Domain Name System) ha il compito di **risolvere** (tradurre) i nomi di dominio che utilizziamo comunemente con i loro rispettivi indirizzi di rete (es. nome: google.com, indirizzo: 8.8.8.8).
La configurazione del protocollo DNS permette al "pubblico" di raggiungere il WebServer tramite il nome "pgcovapi.herokuapp.com" anzichè utilizzare l'indirizzo pubblico definito dal PAT statico 205.123.120.100 (*sezione successiva*).
Il protocollo ha una struttura gerarchica realizzata su un database distribuito.

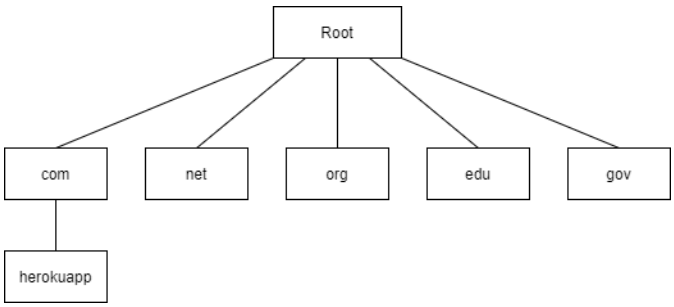


Figure 27: Gerarchia DNS

Alla cima della piramide risiedono i Server root. Nel mondo ne esistono 13, nominati dalla lettera A alla lettera M, clonati centinaia di volte per garantire prestazioni elevate e ridondanza in caso di guasti o congestione. Questi Server contengono all'interno dei loro database gli indirizzi dei Server DNS del livello inferiore, denominati TLD (Top Level Domains).
I Server TLD si occupano dei nomi di dominio che terminano con .com, .net e così via e contengono a loro volta gli indirizzi dei server autoritativi.
I Server autoritativi contengono quindi gli indirizzi veri e propri dei Server web che offrono le pagine HTML.
Nel nostro caso, l' applicazione web da noi sviluppata è pubblicata all'indirizzo pgcovapi.herokuapp.com. Un utente che vuole accedere al sito utilizzando il nome di dominio dovrà rivolgersi ad un Server DNS root per ottenere l'indirizzo di un Server TLD che si occupa della denominazione **.com**. Successivamente, invierà una richiesta al Server TLD per ricevere l'indirizzo del Server autoritativo che si occupa del nome di dominio "herokuapp". Infine, l'utente invierà la richiesta al Server autoritativo per ricevere l'indirizzo vero e proprio del Server web in grado di fornire la pagina web.

Questo esempio è un'estrema semplificazione del funzionamento vero e proprio del protocollo DNS: ci sono diversi modi per effettuare le richieste ai vari Server DNS (ricorsivo / iterativo) e non è il Client vero e proprio ad effettuare la richiesta ma un nodo che prende il nome di "local DNS".

5.2.6 Protocollo NAT

Fino ad ora per la configurazione dei dispositivi sono sempre stati utilizzati indirizzi che iniziano con 192.168.X.X: questi indirizzi prendono il nome di **indirizzi privati** IPv4. Per la natura stessa del protocollo IPv4, possono esistere solamente 2^{32} (circa 4 miliardi) di indirizzi IPv4 diversi e questa quantità non è più sufficiente per identificare distintamente tutti i dispositivi esistenti.
Per questo motivo nel 1996 (rfc1918) alcuni gruppi di indirizzi vennero definiti come "indirizzi privati":

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

Tutti questi indirizzi non possono essere utilizzati per uscire dalle reti private ma hanno il vantaggio di poter essere riutilizzati illimitatamente nelle reti private.

Il protocollo **NAT**(Network Address Translation) ha il compito di tradurre gli indirizzi privati dei pacchetti in indirizzi pubblici per permettere la loro trasmissione in Internet. Il dispositivo sul quale sarà configurato il protocollo salverà nella propria memoria una **NAT table** che associa indirizzi privati e rispettivi indirizzi pubblici. Il protocollo può essere configurato in quattro diverse

modalità

NAT statico: Il NAT statico associa un indirizzo privato ad un indirizzo pubblico. L'amministratore di rete ha il compito di registrare manualmente nella NAT table la coppia indirizzo privato-indirizzo pubblico. Questa modalià non aiuta quindi a preservare il numero di indirizzi pubblici disponibili ma fornisce solo un indirizzo per navigare in rete.

Indirizzo privato	**Indirizzo pubblico**
192.168.5.1	205.123.120.100

Table 1: NAT statico

NAT dinamico: Il NAT dinamico ha a disposizione una pool di indirizzi pubblici. Quando un pacchetto con indirizzo privato deve uscire dalla rete, il dispositivo su cui è configurato il NAT dinamico prende un indirizzo dalla pool e lo usa per tradurre l'indirizzo privato, inserendo la coppia nella NAT table. Quando il pacchetto ritorna all'interno della rete, rimuove l'indirizzo dalla tabella e lo restituisce alla pool. Questo sistema permette di tradurre contemporaneamente un numero limitato di dispositivi che corrisponde al numero di indirizzi pubblici disponibili nel pool.

PAT (NAT overload): Il PAT è la modalità più conveniente permettendo di tradurre dinamicamente più indirizzi privati con un singolo indirizzo pubblico. Il ciò viene effettuato sfruttando i numeri di porta utilizzati al livello 4 (trasporto) della pila ISO/OSI.

Indirizzo privato	**Indirizzo pubblico**
192.168.10.1 : 9836	205.123.120.100 : 9836
192.168.10.2 : 2832	205.123.120.100 : 2832
192.168.10.3 : 4321	205.123.120.100 : 4321

Table 2: PAT

PAT statico: Il PAT statico (definito anche Port Forwarding) ha un funzionamento analogo al NAT statico, con la possibilità di poter definire staticamente sia gli indirizzi di rete sia le porte. Anche in questo caso l'amministratore di rete deve compilare la NAT table manualmente.

All'interno della nostra rete il NAT è stato configurato nelle modalità PAT e PAT statico.
Il PAT statico ci ha permesso di utilizzare il singolo indirizzo pubblico a nostra disposizione (205.123.120.100) per identificare distintamente i due Server presenti nella DMZ (DNS e Web).

Server	Indirizzo privato	Indirizzo pubblico
Web	192.168.5.1:80	205.123.120.100:80
DNS	192.168.5.2:53	205.123.120.100:53

Table 3: Port Forwarding

Nella configurazione del protocollo NAT bisogna definire quali sono le interfacce interne (indirizzo privato) e quali sono le interfacce rivolte verso l'esterno (indirizzo pubblico).

```
1 Router0(config)#interface Serial0/3/0
2 Router0(config-if)#ip nat outisde
3 Router0(config-if)#exit
4 Router0(config)#interface FastEthernet0/1
5 Router0(config-if)#ip nat inside
6 Router0(config-if)#exit
```

In seguito si inseriscono all'interno della NAT table le coppie indirizzo-privato, indirizzo-pubblico

```
1 #ip nat inside source static tcp 192.168.5.1 80 205.123.120.100 80
2 #ip nat inside source static udp 192.168.5.2 53 205.123.120.100 53
```

Dopo aver configurato il PAT statico si è prevista la possibilità per gli utenti all'interno della intranet di voler effettuare richieste in Internet. Tenendo sempre a mente che si ha a disposizione un singolo indirizzo IP, l'unica scelta possibile era configurare anche il PAT.
In questo caso il "lato interno" sarà l'interfaccia FastEthernet1/0 (quella rivolta alla intranet) e l'interfaccia esterna sarà sempre la Serial0/3/0. Per far funzionare il PAT in questo caso sarà però necessario definire tutte le sotto interfacce realizzate precedentemente durante la configurazione delle VLAN come interfacce inside.
Dal punto di vista pratico, il PAT necessita una regola (ACL) che evidenzi quali indirizzi debbano essere tradotti con l'indirizzo pubblico.

```
1 Router(config)#access-list 10 permit 192.168.0.0 0.0.255.255
```

In questo modo stiamo comunicando al router di tradurre tutti gli indirizzi che inizino con 192.168. In seguito si attiva il PAT specificando la modalità **overload**.

```
1 Router(config)#ip nat inside source list 10 interface Se0/3/0 overload
```

5.2.7 WLAN

Le WLAN (Wireless Local Area Network) sono reti che permettono la connessione dei dispositivi tramite Wi-Fi (Wireless Fidelty).

Un dispositivo finale (es. celluare) che si collega ad una rete tramite Wi-Fi prende solitamente il nome di STA (Stazione) e utilizza un modulo di trasmissione WT (Wireless Terminal).
Se tutte le STA sono paritarie la rete si definisce ad-hoc. Nella nostra rete è presente un Access Point collegato ad una LAN che coordina il traffico: in questo caso il modello si definisce **infrastructured**.

Il protocollo **WPA2** è utilizzato per autenticare una STA da parte di un Access Point e può essere configurato in due diverse modalità: Personal ed Enterprise. Nella modalità **enterprise** è neccessario avere nella rete un terzo dispositivo detto KDC (Key Distribution Center) che contenga delle chiavi segrete per ogni utente. Gli utenti devono connettersi alla rete utilizzando un username ed una password per essere riconosciuti dal Server.
Nella modalità **personal** è presente una singola chiave condivisa tra AP e STA detta **PSK**. In questo caso è necessario stabilire un secgreto condiviso (costituito da due chiavi di sessione dette PTK e GTK) a partire dalla singola PSK tramite un **4-way-handshake** per ottenere una comunicazione criptata.

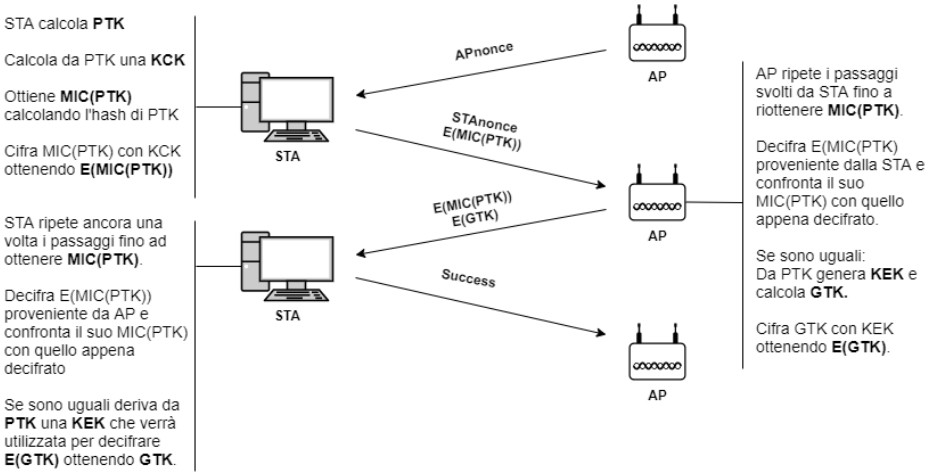


Figure 28: 4 Way Handshake - WPA2 Personal

In riferimento all'immagine sopra riportata, la **PTK** è ottenuta utilizzando la PSK, APnonce, STAnonce, AP Mac address, STA Mac address.

Essendo la WLAN realizzata per ospiti e collaboratori e avendo creato diverse regole (ACL, spiegate nella sezione successiva) che limitano notevolmente la

comunicazione dei dispositivi wireless, si è scelto di configurare il protocollo in modalità personal.

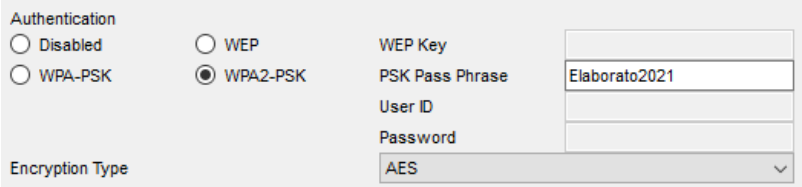


Figure 29: 4 Way Handshake - WPA2 Personal

5.2.8 Access Control Lists

Le ACL sono una lista di regole che permettono di definire delle condizioni di accesso o uscita dei pacchetti in una rete.
Ogni ACL è costituita da una o più regole **ordinate**. Ogni regola può controllare ad esempio l'indirizzo di provenienza, l'indirizzo di destinazione, il protocollo utilizzato e decidere se bloccare (deny) o lasciar passare (permit).
Le ACL si dividono in **Standard** ACL ed **Extended** ACL.
Le Standard permettono solamente di controllare l'indirizzo IP sorgente e il loro numero identificativo va da 1 a 99 e da 1300 a 1399.
Le ACL estese permettono invece di controllare protocollo del pacchetto, indirizzo sorgente e destinazione, le porte sorgente e destinazione e possono anche controllarne lo stato (se si utilizza ad esempio nella creazione della regola la parola **established**, verrà controllato se il pacchetto appartiene ad una comunicazione già precedentemente instaurata).
Le ACL sono state quindi utilizzate per proteggere la rete da eventuali attacchi; le vulnerabilità principali di cui abbiamo tenuto conto sono:

- Attacchi provenienti dall'esterno verso la intranet.
- Attacchi provenienti dall'esterno verso la DMZ.
- Attacchi provenienti dalla DMZ verso la intranet (nel caso uno dei Server venga corrotto).

La prima lista di controllo accessi che è stata definita è l'ACL estesa **101**, in modalità **in** sull'interfaccia **Serial0/3/0**.

Tipo	**Protocollo**	**Sorgente**	**Destinazione**	**State**
permit	tcp	any	host 205.123.120.100 eq 80	
permit	tcp	any	host 205.1230.120.100 eq 443	
permit	udp	any	host 205.123.120.100 eq 53	
permit	tcp	any eq 80	any	established
permit	tcp	any eq 443	any	established

Le prime tre regole permettono l'ingresso dei pacchetti provenienti da qualsiasi dispositivo esterno a patto che siano rivolti ai Server DNS e Web presenti nella DMZ. Le ultime due regole lasciano invece passare i pacchetti che appartengono a comunicazioni HTTP / HTTPS già instaurate da i dispositivi presenti all'interno della rete.
Tramite queste cinque regole si è bloccato tutto il traffico dall'esterno a meno che non siano richieste rivolte ai servizi offerti dalla nostra rete o risposte a richieste effettuate dalla nostra rete aziendale.

Successivamente si è definita l'ACL estesa **102**, in modalità **in** sull'interfaccia del router **FastEthernet0/0** (quella rivolta verso la WLAN). I dispositivi wireless devono essere "trattati con cura" potendosi facilmente connettere alla rete con il semplice inserimento di una password.

Tipo	**Protocollo**	**Sorgente**	**Destinazione**	**State**
deny	ip	any	192.168.10.0 0.0.0.255	
deny	ip	any	192.168.20.0 0.0.0.255	
deny	ip	any	192.168.30.0 0.0.0.255	
permit	tcp	any	any	
permit	udp	any	any	

Le prime tre regole negano il passaggio di tutto il traffico (protocollo ip), al fine di evitare attacchi come **ping flooding**[6]. La quarta regola permette invece il passaggio dei pacchetti rivolti a qualsiasi dispositivo a patto che il protocollo utilizzato sia il TCP (nel caso ad esempio si vogliano effettuare richieste a Server Web, ovunque siano). L'ultima regola lascia il passaggio dei pacchetti UDP anche questi rivolti a qualsiasi dispositivo (ricordando che il protocollo DNS è alla porta udp 53).

Come già scritto, la terza vulnerabilità che abbiamo individuato è stata la possibilità dei Server all'interno della DMZ di subire un attacco e diventare elementi malevoli per la rete. La soluzione adottata è stata quella di creare diverse regole che lasciassero passare solamente i pacchetti appartenenti ai servizi offerti dai Server in modalità **established**. L'ACL **103** è stata quindi applicata in modalità **in** sull'interfaccia **Fa0/1** del router.

Tipo	**Protocollo**	**Sorgente**	**Destinazione**	**State**
permit	udp	host 102.169.5.2 eq 53	any	
permit	tcp	host 192.168.5.1 eq 80	any	established
permit	tcp	host 192.168.5.1 eq 443	any	established

[6]Il ping flooding è tipo di attacco informatico che consiste nell'inviare numerosi messaggi di ping (echo request) ad un dispositivo. Il bersaglio smette di offrire i suoi servizi dovendo rispondere alle richieste (echo reply).

5.3 Ipotesi per protezione da guasti

Per proteggere i dati, si prevede di effettuare frequentemente un backup e di utilizzare gruppi di continuità per continuare a garantire il funzionamento dei dispositivi in caso di blackout.
In caso di guasto dei server, si propone l'aggiunta di server di backup che possano entrare in funzione quando necessario (ad esempio l'aggiunta di un server DHCP secondario che possa estendere le configurazioni già in uso fornite dal server DHCP principale).
Per garantire un costante funzionamento della rete, si prevede un secondo router che possa filtrare il traffico in caso di guasto di quello principale.

Queste sono sole ipotesi per una rete fittizia. Nella realtà si è preferito esternalizzare i servizi affidandosi a servizi di hosting (Heroku e MongoDB Atlas).
Il database MongoDB Atlas replica i dati all'interno di tre diversi nodi fisici per garantire la disponibilità delle informazioni anche in caso di guasti.

6 Gestione progetto

Lo svolgimento del progetto è stato accompagnato dalle pratiche di project management e dalla documentazione proposte dal **PMBOK**, il Project Management Body of Knowledge.

6.1 Project Charter

Per prima cosa si è realizzato il Project Charter, un documento che a grandi linee delinea le caratteristiche del progetto, il suo obiettivo e le risorse necessarie.

1. **OBIETTIVI**
Realizzazione di un'applicazione web dedicata alla visualizzazione dei dati riguardanti l'epidemia COVID in Italia. Sarà basata su un'interfaccia di programmazione (API) in grado di fornire dati in formato JSON. Verrà adottato il **MERN** stack: • MongoDB (DBMS) • ExpressJS (Backend Framework) • ReactJS (Single Page Application Framework) • NodeJS (Runtime Enviroment) Le diverse route dell'API, la quale sarà resa pubblica, permetteranno una facile implementazione da parte di applicazioni anche esterne. Verrà sviluppato un frontend tramite ReactJS che si interfaccerà al backend tramite il client http Axios. Sarà facile comprendere le funzioni fornite dall'API grazie alla documentazione WEB realizzata tramite Swagger UI e pubblicata tramite le Github Pages.
2. **PRINCIPALI DELIVERABLE**
Realizzazione: Link API *Realizzazione: API web documentation (Github Pages)* *Realizzazione: Frontend* *Realizzazione: Documentazione progetto*
3. **MILESTONE**
Codice sorgente backend *Web documentation* *Codice sorgente frontend* *Deployment*

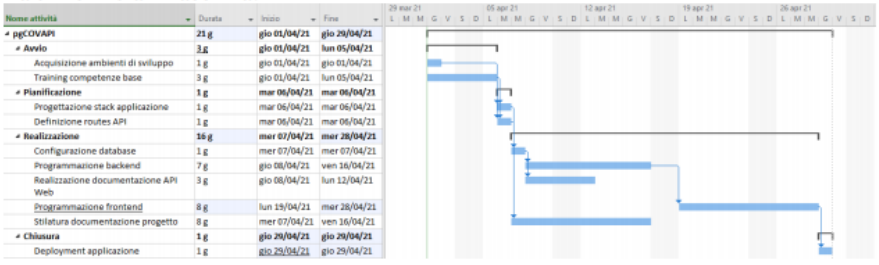
4. VINCOLI E DIPENDENZE

Le risorse necessarie sono:

- Training tecnico sul sistema e linguaggi di sviluppo
- Spazio hosting Heroku
- Spazio hosting MongoDB Atlas

5. TEMPISTICA PRELIMINARE

Il progetto inizia l'1 aprile 2021 e deve cocludersi entro il 29 aprile 2021
La timeline di massima:



Nome attività	Durata	Inizio	Fine
pgCOVAPI	**21 g**	**gio 01/04/21**	**gio 29/04/21**
Avvio	**3 g**	**gio 01/04/21**	**lun 05/04/21**
Acquisizione ambienti di sviluppo	1 g	gio 01/04/21	gio 01/04/21
Training competenze base	3 g	gio 01/04/21	lun 05/04/21
Pianificazione	**1 g**	**mar 06/04/21**	**mar 06/04/21**
Progettazione stack applicazione	1 g	mar 06/04/21	mar 06/04/21
Definizione routes API	1 g	mar 06/04/21	mar 06/04/21
Realizzazione	**16 g**	**mer 07/04/21**	**mer 28/04/21**
Configurazione database	1 g	mer 07/04/21	mer 07/04/21
Programmazione backend	7 g	gio 08/04/21	ven 16/04/21
Realizzazione documentazione API Web	3 g	gio 08/04/21	lun 12/04/21
Programmazione frontend	8 g	lun 19/04/21	mer 28/04/21
Stilatura documentazione progetto	8 g	mer 07/04/21	ven 16/04/21
Chiusura	**1 g**	**gio 29/04/21**	**gio 29/04/21**
Deployment applicazione	1 g	gio 29/04/21	gio 29/04/21

6. PRINCIPALI RISORSE E LIMITI DI COSTO

Le risorse materiali impegnate riguardano le postazioni di lavoro dei programmatori, dal costo medio di 1300 euro e la disponibilità di una connessione di rete.
Il servizio di hosting per il database (MongoDB Atlas) e il servizio di hosting per l'applicazione web (Heroku) non comportano costi aggiuntivi grazie ai loro piani gratuiti.
I programmatori hanno un costo ben definito di 10 euro/h.

7. DOCUMENTI DI RIFERIMENTO E ALLEGATI

MongoDB Manual
Swagger UI Docs
ExpressJS guide
ReactJS Basics

8. STRUTTURA ORGANIZZATIVA

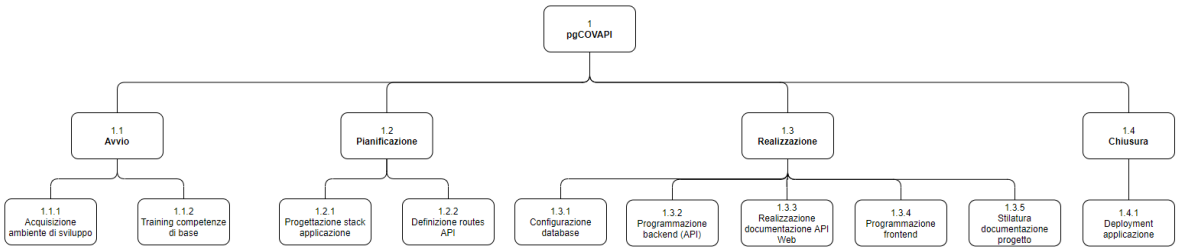
Project Mangager del progetto è l'alunno Silvio Caprara
Il Team di progetto è costituito da 3 programmatori: Silvio Caprara, Pietro Chiartano, Yun Qing Zhou

9. AUTORIZZAZIONE

Approvato da: (Responsabili del Progetto) Data: 01/04/2021

6.2 WBS

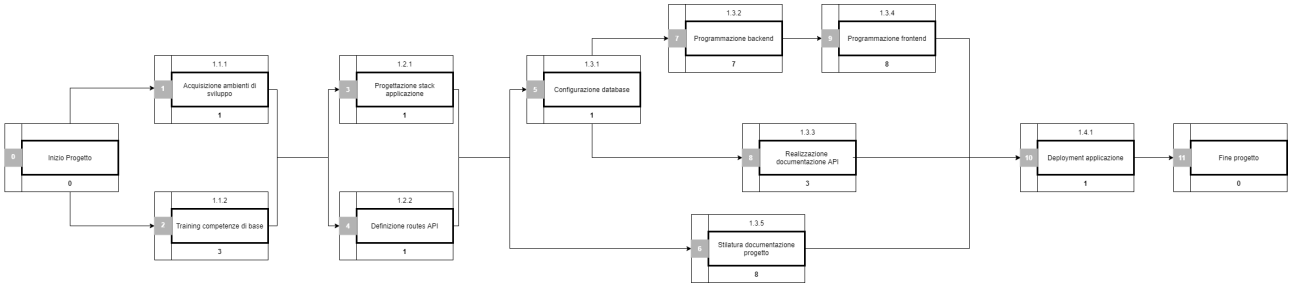
La **WBS** (Work Breakdown Structure) è uno strumento che permette di rappresentare il 100% delle attività di un progetto seguendo una struttura gerarchica.



WBS	Descrizione	Durata (gg)
1.1 AVVIO		
1.1.1	Acquisizione ambiente di sviluppo	1
1.1.2	Training competenze di base	3
1.2 PIANIFICAZIONE		
1.2.1	Progettazione stack applicazione	1
1.2.2	Definizione routes API	1
1.3 REALIZZAZIONE		
1.3.1	Configurazione database	1
1.3.2	Programmazione backend (API)	7
1.3.3	Realizzazione documentazione API Web	3
1.3.4	Programmazione frontend	8
1.3.5	Stilatura documentazione progetto	8
1.4 CHIUSURA		
1.4.1	Deployment applicazione	1

6.3 PDM

Il PDM (Diagramma reticolare delle precedenze) permette di rappresentare graficamente le dipendenze tra le diverse attività del progetto, identificando predecessori e successori.



6.4 Calcolo dei tempi

Dopo aver determinato successori e predecessori, il calcolo dei tempi permette di identificare la durata del progetto, lo **scorrimento** delle singole attività e determinare se sono **critiche** o no. Il calcolo dei tempi si divide nel calcolo delle date al più presto e al più tardi.
Lo scorrimento determina di quanto un'attività può ritardare senza variare la data di fine del progetto e si ottiene confrontando la data al più tardi dell'attività con quella al più presto. Nel caso lo scorrimento corrisponda a 0, l'attività si definisce critica.

Il calcolo dei tempi è stato realizzato tramite i fogli di calcolo Google Sheet ed è stato automatizzato grazie ad un codice da noi scritto sfruttando l'integrazione del tool con GAS (Google App Script), un linguaggio di programmazione per le applicazioni Google basato su JavaScript.

Data inizio progetto:	01/04/2021									
				Al più presto		Al più tardi				
N	Nome attività	Durata	Predecessori	Inizio	Fine	Inizio	Fine	Successore	Scorrimento	Attività Critica
	pgCOVAPI									
0	Inizio Progetto			**gio 01/04/21**	gio 01/04/21	**gio 01/04/21**	gio 01/04/21	1;2		
1	Acquisizione ambienti di sviluppo	1		01/04/2021	01/04/2021	05/04/2021	05/04/2021	3	4	No
2	Training competenze base	3		01/04/2021	05/04/2021	01/04/2021	05/04/2021	3	0	Sì
3	Progettazione stack applicazione	1	1;2	06/04/2021	06/04/2021	06/04/2021	06/04/2021	5;6	0	Sì
4	Definizione routes API	1	1;2	06/04/2021	06/04/2021	06/04/2021	06/04/2021	5;6	0	Sì
5	Configurazione database	1	3;4	07/04/2021	07/04/2021	07/04/2021	07/04/2021	7;8	0	Sì
6	Stilatura documentazione progetto	8	3;4	07/04/2021	16/04/2021	19/04/2021	28/04/2021	10	12	No
7	Programmazione backend	7	5	08/04/2021	16/04/2021	08/04/2021	16/04/2021	9	0	Sì
8	Realizzazione documentazione API Web	3	5	08/04/2021	12/04/2021	26/04/2021	28/04/2021	10	16	No
9	Programmazione frontend	8	7	19/04/2021	28/04/2021	19/04/2021	28/04/2021	10	0	Sì
10	Deployment applicazione	1	6;8;9	29/04/2021	29/04/2021	29/04/2021	29/04/2021	11	0	Sì
11	Fine Progetto		10	29/04/2021	**29/04/2021**	29/04/2021	**29/04/2021**			

6.5 RACI

La RACI è un documento che permette di rappresentare in modo tabellare i ruoli e le responsabilità di ogni lavoratore per ogni attività del progetto. RACI è un acronimo che identifica i quattro diversi ruoli:
Resposinble: Si definiscono con il ruolo responsible tutte le persone che svolgono concretamente il lavoro. E' necessario che venga definito almeno un responsible per attività
Accountable: Per accountable si intende la singola persona per attività che ne detiene l'ownership, ovvero ha l'ultima parola al momento del completamento del lavoro.
Consulted: Il ruolo consulted si assegna alle persone che hanno conoscenze sulle materie trattate nell'attività e possono quindi fornire informazioni per il progetto.
Informed: Tutte le persone che devono essere continuamente informate sulle condizioni e lo stato di avanzamento del progetto vengono assumono il ruolo di informed.

Attività	**Team di progetto**		
	Silvio Caprara (*PM/programmatore*)	Pietro Chiartano (*programmatore*)	Yun Qing Zhou (*programmatore*)
Acquisizione ambiente di sviluppo	**A**/R/C	R/C	R/C
Training competenze di base	R/C	R/C	**A**/R
Progettazione stack applicazione	**A**/R/C	R/C	R/C
Definizione routes API	R/C	R/C	**A**/R/C
Configurazione database	R/C	**A**/R	R/C
Programmazione backend	**A**/R	R/C	R/C
Realizzazione documentazione API Web	R/C	**A**/R/C	R/C
Programmazione frontend	R/C	R/C	**A**/R/C
Stilatura documentazione progetto	**A**/R/C	R/C	R/C
Deployment applicazione	**A**/R/C	R/C	R/C

6.6 RBS

L'RBS (Resource Breakdown Structure) è un documento che permette di calcolare i costi stimati di un progetto mettendone insieme i tre capisaldi: tempi, costi e risorse.

RBS e costi		
1.1.1 Acquisizione ambienti di sviluppo	Tot.	**3900**
Risorse umane:		
Pietro Chiartano	q = 3h, pu = 0	0
Silvio Caprara	q = 3h, pu = 0	0
Yun Qing Zhou	q = 2h, pu = 0	0
Risorse materiali:		
Postazioni di lavoro con PC	q = 3, pu = 1300	3900
Ambiente di sviluppo software	q = 3, pu = 0	0
Test Server	q = 1, pu = 0	0
1.1.2 Training competenze di base	Tot.	**0**
Risorse umane:		
Pietro Chiartano	q = 8h, pu = 0	0
Silvio Caprara	q = 8h, pu = 0	0
Yun Qing Zhou	q = 8h, pu = 0	0
1.2.1 Progettazione stack applicazione	Tot.	**0**
Risorse umane:		
Pietro Chiartano	q = 2h, pu = 0	0
Silvio Caprara	q = 3h, pu = 0	0
Yun Qing Zhou	q = 3h, pu = 0	0

1.2.2 Definizione routes API	Tot.	**0**
Risorse umane:		
Pietro Chiartano	q = 3h, pu = 0	0
Silvio Caprara	q = 2h, pu = 0	0
Yun Qing Zhou	q = 3h, pu = 0	0
1.3.1 Configurazione database	Tot.	**0**
Risorse umane:		
Pietro Chiartano	q = 3h, pu = 0	0
Silvio Caprara	q = 3h, pu = 0	0
Yun Qing Zhou	q = 2h, pu = 0	0
Risorse materiali:		
Sottoscrizione Database MongoDB Atlas	q = 1, pu = 0	0
1.3.2 Programmazione backend (API)	Tot.	**0**
Risorse umane:		
Pietro Chiartano	q = 16h, pu = 0	0
Silvio Caprara	q = 24h, pu = 0	0
Yun Qing Zhou	q = 16h, pu = 0	0
1.3.3 Realizzazione documentazione API Web	Tot.	**0**
Risorse umane:		
Pietro Chiartano	q = 8h, pu = 0	0
Silvio Caprara	q = 8h, pu = 0	0
Yun Qing Zhou	q = 8h, pu = 0	0

1.3.4 Programmazione frontend		Tot.	**0**
Risorse umane:			
	Pietro Chiartano	q = 18h, pu = 0	0
	Silvio Caprara	q = 18h, pu = 0	0
	Yun Qing Zhou	q = 28h, pu = 0	0
1.3.5 Stilatura documentazione progetto		Tot.	**0**
Risorse umane:			
	Pietro Chiartano	q = 22h, pu = 0	0
	Silvio Caprara	q = 22h, pu = 0	0
	Yun Qing Zhou	q = 20h, pu = 0	0
1.4.1 Deployment applicazione		Tot.	**0**
Risorse umane:			
	Pietro Chiartano	q = 2h, pu = 0	0
	Silvio Caprara	q = 3h, pu = 0	0
	Yun Qing Zhou	q = 3h, pu = 0	0
Risorse materiali:			
	Server di Produzione	q = 1, pu = 0	0
		Tot.	**3900**

7 Conclusioni

Dopo aver pubblicato il sito, ogni sera dopo le ore 7:00 è possibile visualizzare tramite il sito l'attuale situazione epidemiologica con i dati forniti dalla Protezione Civile. La piattaforma realizzata non è innovativa: come già detto nell'introduzione sono state sviluppate innumerevoli applicazioni inerenti a questo tema.

L'obiettivo principale è stato quindi quello di apprendere ed utilizzare tecnologie innovative che stanno prendendo sempre più piede nel mondo del lavoro (soprattutto all'estero, recentemente anche in Italia).

Per quanto ogni membro del gruppo si sia concentrato sull'implementazione di una singola tecnologia, tutti abbiamo dovuto utilizzare o almeno comprendere il funzionamento di ogni parte dell'applicazione. Il sito deve comunicare con l'API e quest'ultima deve interfacciarsi al database. Nel mio caso, avendo lavorato sull'elemento "collante" del progetto, ho dovuto chiedere all'amministratore del database quale fosse la struttura dei dati e quale fosse il metodo di connessione alla base di dati. Allo stesso tempo, ho dovuto comunicare allo sviluppatore del sito la struttura del servizio ed ascoltare i suggerimenti per semplificarne l'utilizzo.
E' stata quindi fondamentale la fase di progettazione nella quale abbiamo dovuto definire gli standard di comunicazione tra nodi, i nostri obiettivi ed i tempi di consegna da rispettare.

Grazie all'utilizzo della piattaforma Github abbiamo potuto tenuto traccia delle versioni del codice, registrare gli aggiornamenti e proporre modifiche.

Speriamo che lo studio di queste nuove tecnologie ci fornisca un aiuto durante i nostri studi universitari e ci prepari ai continui cambiamenti nel mondo informatico.

8 Sitografia

Documentazione MongoDB
(https://docs.mongodb.com/manual/)

Documentazione ReactJS
(https://it.reactjs.org/docs/getting-started.html)

Documentazione ExpressJS
(https://expressjs.com/it/api.html)

Documentazione NodeJS
(https://nodejs.org/it/docs/)

Documentazione Cors
(https://developer.mozilla.org/it/docs/Web/HTTP/CORS)

Documentazione Moongose
(https://mongoosejs.com/docs/)

Documentazione SchedulerJS
(https://www.npmjs.com/package/scheduler)

Documentazione Swagger V3
(https://swagger.io/docs/specification/about/)

Documentazione LaTeX
(https://www.overleaf.com/learn)

Documentazione Carousel
(https://github.com/leandrowd/react-responsive-carousel)

Documentazione MapboxGL
(https://docs.mapbox.com/mapbox-gl-js/api/map/)

REST: cos'è e le differenze con le Web API
(https://www.html.it/articoli/cos-e-rest-caratteristiche/)

Dissertazione Roy Thomas Fielding
(https://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm)

Wikipedia - Reprentational State Transfer
(https://it.wikipedia.org/wiki/Representational_State_Transfer)

Paradigma REST e differenze con il RESTful
(https://vitolavecchia.altervista.org/paradigma-rest-representational-state-transfer-e-differenze-con-il-restful/)

SOAP vs REST
(https://www.html.it/pag/19612/differenze-tra-web-service-rest-e-soap/)

Da SOAP a REST
(https://medium.com/uvdesk-engineering/api-paradigm-shift-soap-to-rest-11214310588d)

Red Hat - cos'è un'API
(https://www.redhat.com/it/topics/api/what-are-application-programming-interfaces)

Glossary

ACID Atomicity, Consistency, Isolation, Durability. Paradigma dei DMBS relazionali transazionali. 12

ACL Access Control List, regole che controllano l'ingresso e l'uscita di pacchetti attraverso un nodo. 39–42

AP Access Point, nodo di controllo delle WLAN. 40

API Tecnologia che permette il collegamento e la comunicazione tra due sistemi come database e interfaccia grafica. 5–8, 15, 17, 18, 22–25, 27

back-end Il back-end è il lato server di un'applicazione, ovvero tutto il meccanismo che un utente non vede ma permette che il tutto funzioni. 8, 15, 17, 21

BASE Basically Available, Soft state, Eventually consistent. Paradigma per sistemi distribuiti non transazionali. 13

BSON Binary JSON. 9

CAP Consistency, Availability, Partition Tolerance. Paradigma dei sistemi distribuiti. 13

CLI Command Line Interface, interfaccia testuale per l'esecuzione di comandi. 15, 31

Client In una comunicazione informatica è solitamente il nodo che effettua le richieste ad un server. 8, 16, 27, 32, 33, 35, 37

commit Istruzione che consolida le modifiche di una transazione. 12, 13

CORS cross-origin resource sharing. Tecnologia che permette le richieste da domini esterni. 18, 21

CRUD Create, Read, Update e Delete. Operazioni fondamentali per l'interazione con una base di dati. 17

CSS Cascade Style Sheet, linguaggio utilizzato per definire la formattazione di documenti HTML. 26, 27, 34

DBMS DataBase Management System, software con il compito di gestire il database, lavorando sulla struttura e sui dati. 8, 9, 12, 14

DHCP Dynamic Host Configuration Protocol, protocollo utilizzato per configurare dinamicamente dispositivi finali. 31–34

DMZ Demilitarized Zone, sottorete che contiene ed espone dei servizi ad una rete esterna non ritenuta sicura. 29, 32, 34, 38, 41, 42

DNS Domain Name System / Domain Name Server, protocollo che si occupa di tradurre i nomi di dominio in indirizzi IP. 32, 36–38, 42

DOM Document Object Model, tecnica per accedere e aggiornare dinamicamente il contenuto, la struttura e lo stile dei documenti tramite un linguaggio di scripting come JavaScript. 26, 27

FK Foreign Key, colonna contenente valori che fanno riferimento ad una Primary Key presente in un altra tabella. La foreign key costituisce quindi un vincolo tra due tabelle. 9

footer Nel contesto delle pagine web, è un blocco di testo al fondo di una pagina contenente solitamente informazioni come: autori, informazioni sul copyright, links, contatti etc... 6

framework Software utilizzato da sviluppatori per costruire applicazioni. Spesso presenta funzioni per semplificare lo sviluppo. 15, 17, 18, 25

front-end Il front-end di un'applicazione è costituito da tutti gli elementi grafici che l'utente vede e con i quali può interagire. 8, 9, 25, 27

full-stack developer Sviluppatore che lavora sia nel lato back-end che front-end di un'applicazione. 5

HTTP Hyper Text Transfer Protocol, protocollo utilizzato in Internet che definisce il metodo di trasmissione e la struttura dei messaggi in rete. 17, 27, 34, 35, 42

HTTPS Secure Hyper Text Transfer Protocol, versione sicura che implementa SSL/TLS. 27, 35, 42

intranet Rete aziendale privata complemante isolata dalla rete esterna. 28–30, 33, 34, 39, 41

JavaScript Linguaggio di programmazione open-source utilizzato principalmente per rendere pagine web interattive. E' stato rilasciato nel 1995. 9, 17, 19, 25–27, 47

JSON JavaScript Object Notation, struttura utilizzata per l'immagazzinamento e lo scambio di dati in un formato da noi leggibile. 9, 10, 15, 17, 20, 22, 27

JSX JavaScript XML. Permette la scrittura del linguaggio HTML in ReactJS. 26

KDC Key Distribution Center. 40

MEAN Stack di tecnologie per la realizzazione di applicazioni web formato da MongoDB, ExpressJS, AngularJS, NodeJS. 9, 10

MENV Stack di tecnologie per la realizzazione di applicazioni web formato da MongoDB, ExpressJS, VueJS, NodeJS. 9, 10

MERN Stack di tecnologie per la realizzazione di applicazioni web formato da MongoDB, ExpressJS, ReactJS, NodeJS. 8–10

NAT Network Address Translation, protocollo di rete che traduce gli indirizzi privati in indirizzi pubblici. 37–39

PAT Port Address Translation, protocollo di rete che traduce gli indirizzi privati in indirizzi pubblici utilizzando il numero di porta.. 38, 39

PK Primary Key, colonna (o insieme di colonne) che contiene valori unici permettendo l'identificazione delle singole righe all'interno della stessa tabella. 9

PMBOK Project Management Body of Knowledge. 44

query In generale, per query si intende una richiesta fatta da un utente, un dispositivo o un software.. 23, 24

REST Representational State Transfer, paradigma API basate su risorse. 16, 17

rollback Istruzione che annulla le modifiche di una transazione ripristinando lo stato iniziale. 12

Server Macchina ad alte prestazioni che fornisce un servizio web su richiesta. 8, 9, 27–29, 32–35, 37, 38, 40–42

SOAP Simple Object Access Protocol, protocollo per lo sviluppo di API basate su metodi a chiamata remota. 16

SSL Secure Socket Layer. 35

STA STA si intende un dispositivo finale che si collega ad una rete tramite Wi-Fi. 40

TLS Transport Layer Security. 35, 36

VLAN Virtual Local Area Network, tecnologia che permette di dividere una rete fisica in più sottoreti logiche. 29–32, 39

WLAN Wireless Local Area Network. 29, 40, 42

WPA2 Wi-Fi Protected Access versione 2. 40

WT Wireless Terminal, modulo per il collegamento wireless. 40