Studente: Dawid Węglarz
Matricola: 277268

Reti di Calcolatori

Progetto per la sessione autunnale 2019/2020

Docente: Prof. Antonio Della Selva

1. Introduzione

Questo progetto ha come obbiettivo la realizzazione di un tool che permette di scansionare pagine web e scoprire quali tecnologie web utilizzano.

In particolare il tool si basa sul utilizzo di espressioni regolari con le quali cerca degli indicatori nel codice HTML della pagina per capire se è presente o meno una certa tecnologia web.

Per esempio attraverso la regular expression:

<meta[^>]+generator[^>]+content\s=\s*.Joomla*

possiamo vedere quali siti utilizzano il CMS Joomla.

2. Progettazione

Il progetto sarà composto da due componenti principali:

- Un **backend** che si occupa di fornire i dati a chi li richiede e gestire le query verso il database.
- Uno o più **client** che si occupano di visualizzare i dati e trasmettere l'input dal utente al server.

Il **backend** esporrà un API che permette sia di aggiungere che vedere le tecnologie e siti web che sono presenti nel database. Le tecnologie saranno composte da un identificativo e da una espressione regolare. I siti web invece avranno un nome simbolico e un URL.

Durante l'inserimento di un nuovo sito web verrà attivato un trigger che scaricherà la pagina web tramite una richiesta GET e confronterà il contenuto con tutte le espressioni regolari presenti al momento. Alla fine di questo processo verranno create tutte le associazioni tra i match e rese visibili all'utente.

I dati possono essere recuperati successivamente attraverso una richiesta GET ai seguenti indirizzi:

http://51.158.173.57:9000/api/v1/tech/ (JSON delle tecnologie web)

http://51.158.173.57:9000/api/v1/webpage/ (JSON delle pagine web)

I dati possono essere inserti facendo una richiesta POST agli stessi indirizzi inserendo nel body della richiesta i dati richiesti.

Il **client** farà uso dell'API per richiedere e inserire nuovi dati. Una volta ottenuti i dati si possono fare molte cose: creare statistiche, grafici o interazioni più complesse ma si possono anche semplicemente visualizzare avendo già delle informazioni interessanti.

3. Implementazione

Il backend è stato realizzato in **Python**, in particolare utilizzando la libreria **Django**. Questa libreria permette di lavorare e gestire le query al database senza dover conoscere la sua implementazione. Tutti i soggetti vengono rappresentati come classi (chiamati anche modelli) e gestiti da Django.

```
class Technology(models.Model):
    name = models.CharField("Technology name", max_length=20)
    regex = models.CharField("Regular expression", max_length=100)


class Webpage(models.Model):
    name = models.CharField("Page name", max_length=20)
    url = models.CharField("Page URL", max_length=200, blank=False)
    technologies = models.ManyToManyField(Technology, blank=True)
```

I due modelli che rappresentano le tecnologie e le pagine web.

Un altra componente fondamentale è un plugin di Django, chiamato **Tastypie**, che aggiunge la possibilità di definire un ulteriore astrazione dai proprio modelli e con questa creare una interfaccia API completa e facilmente estendibile.

Le principali modifiche che ho apportato all'API sono un validatore che durante le richieste POST controlla che il link inserito dal utente sia valido e raggiungibile. Nel caso delle tecnologie web viene controllata la correttezza dell'espressione regolare.
Oltre a questo durante una richiesta POST per inserire una nuova pagina web viene fatta partire una funzione che scaricare il contenuto della pagina web e lo confronta con tutte le espressioni regolari già presenti nel database inserendo una relazione tra i due modelli in caso di match.

```
{
    "id": 7,
    "name": "CSS",
    "regex": "<link rel=.stylesheet[^>]",
    "resource_uri": "/api/v1/tech/7/"
},
{
    "id": 8,
    "name": "PHP",
    "regex": "\\.php(?:$|\\?)",
    "resource_uri": "/api/v1/tech/8/"
},
{
    "id": 9,
    "name": "React",
    "regex": "<.*id=\"react-root\".*[^>]",
    "resource_uri": "/api/v1/tech/9/"
},
{
    "id": 10,
    "name": "Captcha",
    "regex": "<meta.*name=.captcha-bypass.*[^>]",
    "resource_uri": "/api/v1/tech/10/"
},
{
    "id": 11,
    "name": "Microsoft ASP.NET",
    "regex": "<.*\\.asp.*[^>]",
    "resource_uri": "/api/v1/tech/11/"
},
{
    "id": 12,
    "name": "Google Analytics",
    "regex": "<meta.*name=.google-analytics.*[^>]",
    "resource_uri": "/api/v1/tech/12/"
},
```

```
{
    "id": 13,
    "name": "Github",
    "resource_uri": "/api/v1/webpage/13/",
    "technologies": [
        "/api/v1/tech/3/",
        "/api/v1/tech/12/"
    ],
    "url": "https://github.com"
},
{
    "id": 14,
    "name": "Reddit",
    "resource_uri": "/api/v1/webpage/14/",
    "technologies": [],
    "url": "https://www.reddit.com/"
},
{
    "id": 15,
    "name": "Youtube",
    "resource_uri": "/api/v1/webpage/15/",
    "technologies": [
        "/api/v1/tech/7/"
    ],
    "url": "https://www.youtube.com"
},
{
    "id": 18,
    "name": "Microsoft",
    "resource_uri": "/api/v1/webpage/18/",
    "technologies": [
        "/api/v1/tech/2/",
        "/api/v1/tech/7/",
        "/api/v1/tech/11/"
    ],
    "url": "https://www.microsoft.com"
},
```


Esempi di risposta del API (tecnologie/pagine web)

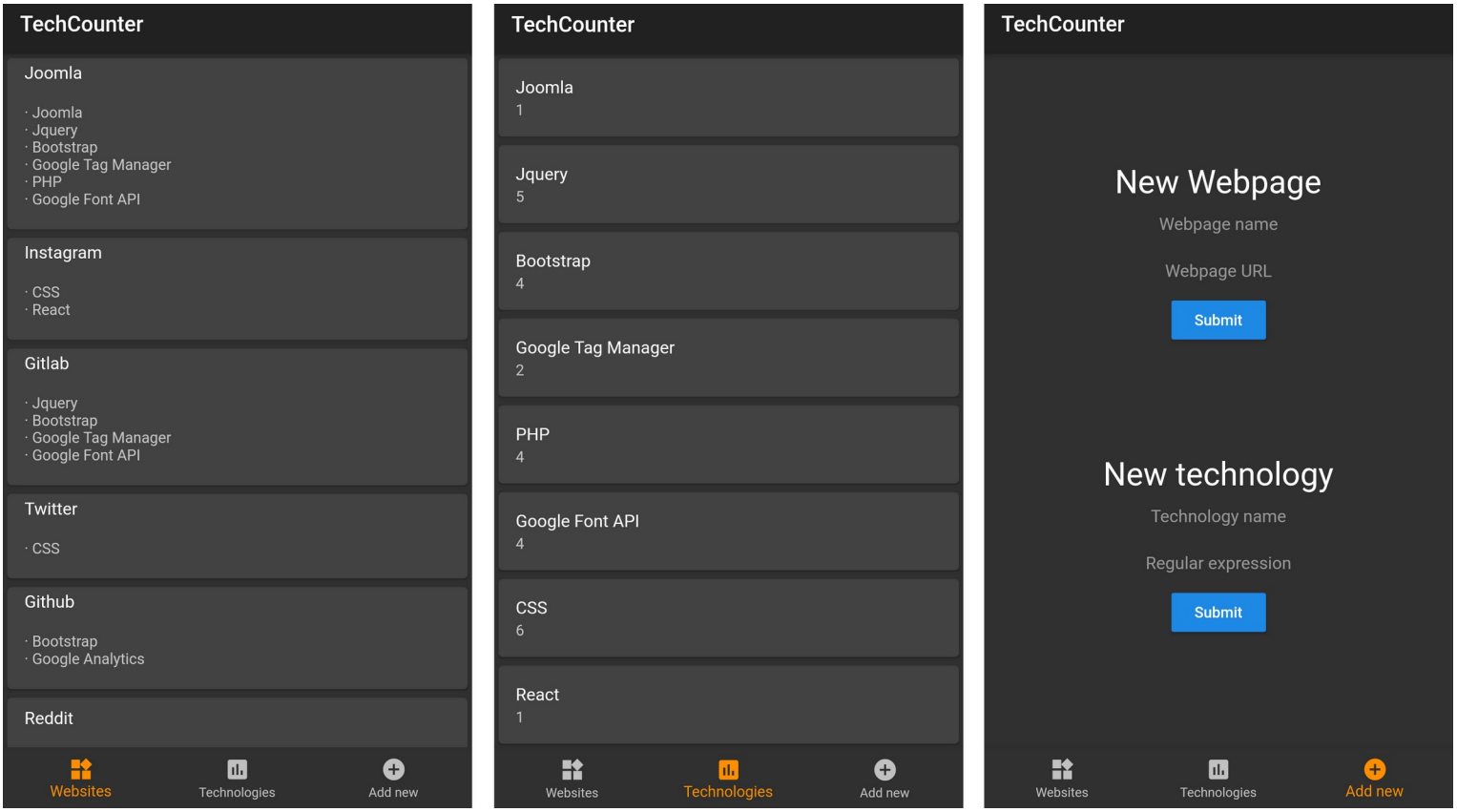
Per quanto riguarda il lato **client** ho deciso di sviluppare un app mobile piuttosto che una web app. Questa poteva essere facilmente implementata grazie a Django ma rendeva tutto un po troppo compatto e non sfruttava per niente l'API.

Quindi per rendere il progetto più interessante (e didattico per me) ho deciso di sviluppare un applicazione in **Flutter**, un framework che permette di sviluppare applicazioni cross-platform con un unica codebase.

Il risultato è un applicazione, dall'aspetto minimale ma piacevole, che permette di vedere tutti i dati inseriti dagli utenti in tempo reale e inserirne di nuovi.

L'app è composta da tre viste dove vengo visualizzate:

1. La lista di tutte le pagine e delle tecnologie presenti in ciasuna di esse.
2. Una lista delle tecnologie con la frequenza di quante volte appaiono tra i siti presenti.
3. Due semplici form che permettono di inserire nuovi dati.



Le tre schermate dell'applicazione (Siti, tecnologie e aggiunta di nuovi dati)

2.1 Problematiche e considerazioni

Durante la realizzazione del progetto mi sono imbattuto in alcuni punti che mi hanno fatto riflettere e magari ho sottovalutato nella progettazione iniziale:

- Alcune tecnologie web sono difficili da rilevare guardando semplicemente il codice HTML di un dato URL. Per questo alcuni servizi online che si occupano proprio di questo controllano anche i vari file (JS, CSS,...) presenti nella pagina.

- Un altro problema è quello che una nuova versione di una particolare tecnologia potrebbe portare ad un falso negativo. Questo si potrebbe evitare costruendo una tabella di espressioni regolari cosi da coprire tutti questi casi e minimizzare questo evento.

3. Conclusioni

Da questo progetto possiamo vedere come attraverso delle semplici richieste GET e POST possiamo scoprire moltissimo su un sito web se sappiamo dove guardare.
In questo caso il tool facilita l’operazione di ricerca nell’HTML grazie all’algoritmo ma sopratutto al fatto che anche gli (ipotetici) utenti possono inserire nuovi “filtri” e contribuire nel rendere lo strumento più accurato.