Competenze assimilate grazie al progetto	
Design di prototipi di grafiche con software figma.	Competenza trasversale
Gestione delle interfacce grafiche in C con gtk.	Competenza trasversale
Funzionamento e scrittura in C di file di immagini vettoriali (svg).	**Obiettivo**
Funzionamento, lettura e scrittura di un file csv.	**Obiettivo**
Funzionamento e manipolazioni di APIs di multinazionali del settore tech, anche mediante librerie di codice importate e analizzate in python.	Competenza trasversale
Funzionamento e manipolazione dati in formato json in python.	Competenza trasversale
Programmazione open source sicura grazie alla variabili di ambiente in python (.env variables).	Competenza trasversale
Manipolazione grafica di video.	Competenza trasversale
Stesura di documentazione efficace.	Competenza trasversale
Lavoro collaborativo con repository remota, tramite tools git e github.	Competenza trasversale

CREDITS

Un progetto di:
Matteo Bini <binim@fermimn.edu.it>, <mail@matteobini.me>
Simone Tardiani <tardianis@fermimn.edu.it>
Ryan Znaidi <rznaidi@fermimn.edu.it>

studenti della classe 3CIIN
all'Istituto Superiore Enrico Fermi di Mantova.

LICENZA

Relazione delle modalità di lavoro e di apprendimento, divise per “aspetto”: ovvero le avversità superate durante la progettazione e implementazione del software.

Creazione di un software con interfaccia grafica in grado di generare immagini vettoriali contenenti informazioni di un tweet.

Aspetto 0 - divisione e organizzazione del lavoro

Per una maggiore efficienza, il gruppo ha deciso, come prima cosa, di organizzare la divisione del lavoro. Ogni componente del gruppo si è occupato alla revisione del codice, ma successivamente:

+ Matteo Bini si è occupato della scrittura e pubblicazione della presente documentazione;
+ Simone Tardiani si è occupato della produzione del filmato;
+ Ryan Znaidi si è occupato della esposizione mediante slides.

Essendo il progetto ambizioso, il gruppo ha deciso che anche la scrittura del codice dovesse essere divisa tra i componenti. Infatti, i tre maggiori aspetti dello sviluppo del programma sono stati divisi equamente tra gli sviluppatori:

+ Matteo Bini si è occupato della realizzazione grafica del programma;
+ Simone Tardiani ha gestito le APIs e i contatti con Twitter, Inc.;
+ Ryan Znaidi si è occupato della scrittura dei file immagine.

La progettazione del layout dell’immagine è stata prodotta sinergicamente da tutti i componenti, dal momento che il contenuto prodotto secondo noi deve godere delle massime attenzioni.

Per lavorare tutti sullo stesso progetto, abbiamo deciso di creare una repository github. L’IDE scelto è CodeBlocks, programma open-source e cross-platform indicato per la creazione di programmi C con interfaccia grafica.

Per lavorare sulla stessa codebase, abbiamo creato una repository su github e tramite il software git abbiamo gestito le modifiche e i merge del codice. Il codice da noi prodotto è open-source e disponibile al link: https://github.com/matteobini/tw_to_svg.

Aspetto 1 - impostazione del layout grafico dell'immagine da produrre e prototipi vari

A cura di Ryan Znaidi

Cosa è un file SVG

Il linguaggio SVG è quasi totalmente XML, per questo può essere implementato in un file HTML o può essere creato come file standalone. A un codice SVG può essere linkato un file CSS esternamente, ma solo se è all'interno di un file HTML; se il file è standalone il codice CSS deve essere per forza incluso in un tag style al interno del file.

Basi del file SVG: Il file SVG si apre con un tag svg (dato che questo linguaggio è estremamente simile a HTML per quanto riguarda la sintassi).

```
<svg width="720" height="1280" version="1.1"
                        xmlns="http://www.w3.org/2000/svg">
<!--  CODICE  -->
</svg>
```

Dentro questo tag verrà inserita qualunque cosa (come per il tag `<html>` in HTML).

+ **width/height**: Definiscono l'altezza e la larghezza dell'immagine.
+ **version**: Definisce la versione di codice SVG che andremo ad usare.
+ **xmlns**: Definisce il namespace dell' XML.

Ora si potrebbe già iniziare a scrivere del codice per creare delle forme, tuttavia non avrebbero colore o attributi particolari, per questo serve il tag style per inserire dei CSS.

Il tag *style* funziona allo stesso modo che in HTML.

Spiegazione del codice

Per prima cosa, creiamo un tag <style> dove saranno contenuti i CSS dell'immagine.

```
<defs>
<style type="text/css">
      rect{
            fill: black;
      }
      .centered_text{
            font-family: 'Inter', sans-serif;
            dominant-baseline:middle;
            text-anchor:middle;
            inline-size: 500px;
            overflow-wrap: break-word;
      }
      .like{
            fill: white;
      }
</style>
</defs>
```

Iniziamo a scrivere il corpo del programma che contiene tutti i testi e le immagini.
Il name, username, il numero di likes è variabile in base al tweet. Questi dati verranno dettati dal programma e, per il momento, li coloriamo in blu. Applichiamo a queste informazioni dei CSS diversi per il posizionamento, lo stile, etc; inoltre creiamo anche lo sfondo nero del tweet con un tag <rect>.

```
<rect width="1000" height="500" x="0" y="0"/>
      <text x="50%" y="29%" font-size="48"
      font-weight="bold" fill="#E7E9EA" class="centered_text">
            { name }
      </text>
      <text x="50%" y="35%" fill="#72777C" font-size="24"
            class="centered_text">
            { screen_name }
      </text>
      <text x="10%" y="10%" fill="white" font-size="24"
            font-weight="bold" class="centered_text">
            { likes }
            <tspan fill="#72777C" font-weight="normal">
                  Likes
            </tspan>
      </text>
      <text x="90%" y="10%" fill="white" font-size="24"
            font-weight="bold" class="centered_text">
            { retweets }
```

```
            <tspan fill="#72777C" font-weight="normal">
                Retweets
            </tspan>
        </text>
```

Il testo del tweet ha bisogno di un'attenzione particolare poiché non è possibile mandare a capo automaticamente un testo in SVG, quindi bisogna usare un tag <tspan> che crea una sezione di testo che poi spostiamo una riga sotto attraverso le coordinate. Il programma C decide ogni quante parole mandare a capo il testo.

```
<text x="50%" y="50%" font-size="36"  fill="#E7E9EA"
    class="centered_text">
    <tspan x="50%" dy="0em">
        { tweet (riga 1) }
    </tspan>
    <tspan x="50%" dy="1.2em">
        { tweet (riga N) }
    </tspan>
</text>
```

Creiamo l'immagine del cuore per il like attraverso un tag <path> che crea una forma seguendo delle istruzioni di movimento; inoltre inseriamo la forma all'interno di un contenitore col tag <g>.
A questo punto spostiamo il contenitore al centro della pagina e lo scaliamo a una dimensione adatta con l'attributo: `transform="translate(500 460) scale(0.50 0.50)"`; applichiamo anche l'attributo `transform="translate(-50 -50)"` direttamente alla forma per portarla al centro del contenitore.

```
<g transform="translate(500 460) scale(0.50 0.50)">
    <path transform="translate(-50 -50)" class="like"
        d="M92.[..].27z">
   </path>
</g>
```

Aggiungiamo l'animazione del colore del cuore col tag <animate> e lo applichiamo al tag <path>.

```
<g transform="translate(500 460) scale(0.50 0.50)">
    <path transform="translate(-50 -50)" class="like"
d="M92.[..].27z">
    <animate
        attributeName="fill"
        values="white;red;red"
        dur="0.5s"
        repeatCount="1"
        begin="click"
        fill="freeze"
    />
    </path>
</g>
```

. . . mentre per l'animazione della dimensione utilizziamo il tag `</animateTransform>` nel tag `<g>`.

```
<g transform="translate(500 460) scale(0.50 0.50)">
      <path transform="translate(-50 -50)" class="like"
            d="M92.[..].27z">
            <animate
                  attributeName="fill"
                  values="white;red;red"
                  dur="0.5s"
                  repeatCount="1"
                  begin="click"
                  fill="freeze"
            />
      </path>
      <animateTransform
            attributeName="transform"
            type="scale"
            values="1;1.25;1;1;1"
            dur="0.5s"
            repeatCount="1"
            additive="sum"
            begin="click"
      />
</g>
```

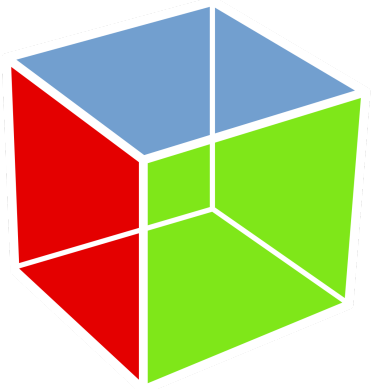
Infine aggiungiamo l'immagine del profilo dell'utente, tuttavia per questo progetto ci serve un immagine circolare anche se quella fornita da twitter è quadrata, per questo usiamo il tag `<clipPath>` che crea una forma che andrà a tagliare in modo corretto l'immagine del profilo.

```
<circle cx="500" cy="60" r="50" fill="gray" />
<clipPath id="clipCircle">
      <circle r="50" cx="500" cy="60"/>
</clipPath>
<image
      href="{ pfp_link }"
      height="100" width="100" x="450" y="10"
      clip-path="url(#clipCircle)"
/>
```

L'acronimo pfp abbrevia "profile picture".

Aspetto 2 - realizzazione dell'interfaccia grafica

A cura di Matteo Bini



Per l'interfaccia grafica, ho deciso di utilizzare gtk, una libreria C/C++ che facilita lo sviluppo di questi applicativi con GUI. Per compilare il progetto, è infatti essenziale installare gtk dal sito ufficiale per il proprio sistema operativo. Io personalmente sviluppo su sistemi linux, ma il mio codice è compilabile nativamente anche per Windows e MacOS (anche con processori di ultima generazione Apple Silicon).

L'interfaccia costruita è estremamente semplice, composta da due widget adibiti all'input del id/link del tweet e un pulsante per confermare la scelta. Nel presente documento descriverò il codice dell'interfaccia e del collegamento al codice scritto per affrontare l'aspetto 2 dello sviluppo del programma.

All'interno della funzione principale, inizio dichiarando le variabili dei widget che utilizzerò:

```
GtkWidget *win = NULL;        // finestra
GtkWidget *entry = NULL;      // widget per text input field
GtkWidget *hbox = NULL;       // widget container (tipo div)
GtkWidget *button1;           // bottone per confermare input
```

Successivamente, configuro e inizializzo gtk per il mio progetto:

```
g_log_set_handler("Gtk",
                  G_LOG_LEVEL_WARNING,
                  (GLogFunc) gtk_false, NULL);
gtk_init(&argc, &argv);
g_log_set_handler("Gtk", G_LOG_LEVEL_WARNING,
                  g_log_default_handler, NULL);
```

A questo punto, sono pronto a inizializzare la mia finestra:

```
win = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_default_size(GTK_WINDOW(win), 300, 200);
```

Il primo parametro della funzione `gtk_window_set_default_size` indica la finestra alla quale mi riferisco, il secondo e il terzo indicano la dimensione della finestra (prima in X, poi in Y).

All'interno della finestra, devo mostrare l'input field - il campo dell'input - nel quale l'utente incollerà l'id/link del tweet:

```
entry = gtk_entry_new();
gtk_entry_set_text(entry,"Insert tweet id or link");
gtk_widget_set_size_request(entry, 80, 80);
```

Il secondo parametro di "gtk_entry_set_text" (testo tra virgolette), è la stringa che verrà mostrata all'utente nel widget di input, così da aiutarlo a capire cosa fare e rendere il programma più intuitivo.

A questo punto, devo configurare, nella finestra, anche il bottone per confermare l'input:

```
button1 = gtk_button_new_with_mnemonic("Submit!");
```

Fatto questo, indico a gtk cosa fare quando il pulsante viene premuto, anche in relazione al testo presente nel buffer del campo (field) di input (variabile "entry").

```
g_signal_connect(button1,
                      "clicked",
                      G_CALLBACK(button_clicked), entry);
g_signal_connect(entry, "activate",
                      _CALLBACK(button_clicked),
                      entry);
```

A questo punto, grazie al widget container "hbox", dispongo nella finestra tutto quello che ho creato fino ad adesso.

```
/* decido la posizione del container nella finestra */
hbox = gtk_hbox_new(0,0);
/* posiziono il field a sinistra */
gtk_box_pack_start(GTK_BOX(hbox), entry, 1, 1, 0);
/* posiziono il button a destra */
gtk_box_pack_end(GTK_BOX(hbox), button1, 0, 0, 1);
/* finalmente aggiungo il container alla finestra */
gtk_container_add(GTK_CONTAINER(win), hbox);
```

Alla fine della funzione principale *main*, apro la finestra e preciso che deve rimanere aperta in loop.

```
g_signal_connect(win, "destroy", gtk_main_quit, NULL);
gtk_widget_show_all(win);
gtk_main(); /* per tenere la finestra aperta in loop */
return 0;
```

Ora che si è conclusa la programmazione dell'aspetto prettamente grafico, è il momento di analizzare il meccanismo che lega e connette l'interfaccia utente al servizio che permette di scaricare i dati forniti dalle APIs di twitter.

Ho scritto una funzione appositamente per il data entry che, sostanzialmente, permette il supporto all'inserimento del codice identificativo di un tweet sia mediante link sia id (l'id è anche l'ultima parte del link, senza slash).

Questa funzione si chiama “button_clicked” e l’ho già menzionata sopra, esattamente quando configuravo la reazione del pulsante al click, come parametro del callback.

Una volta terminata la “pulizia” dell’id, la funzione invoca la funzione “get_tweet”, che permette di eseguire, tramite la bash di sistema, il codice python adibito all’iterazione con le APIs di Twitter, Inc., ecco il codice della funzione, per l’esecuzione del programma:

```
char command[L] = "python3 ./GetTweetInfo.py ";
strcat(command, tweet_id); // effettuo append dell’id
system(command); // eseguo il comando
```

Il comando che la bash riceverà è il seguente:

```
python3 ./GetTweetInfo.py 1516620752040116231
```

Dopo l’esecuzione del codice in python, il programma C, proseguirà con la sua esecuzione prelevando da un file .csv salvato localmente. La funzione “system()” fa parte della C standard library (stdlib.h).

Aspetto 3 - relazione con Twitter, Inc. e modalità di connessione

A cura di Simone Tardiani

Purtroppo, le librerie C che facilitano questo tipo di servizi e interazione sono state tutte deprecate e inserite nella blacklist delle twitter developer guidelines, di conseguenza, per non cambiare progetto, abbiamo deciso di implementare questa funzione in python e comunicare con il resto del codice C tramite un file csv.

Per accedere ai server twitter, è necessario interagire con le APIs che l'azienda ha deciso di rendere disponibili pubblicamente. Per effettuare tutte le comunicazioni è stata necessaria una procedura di registrazione e autorizzazione al twitter developers portal che non mi dilungherò a spiegare ma che, alla fine di un processo di autenticazione e registrazione, fornisce quattro stringhe (token) che invieremo al server durante le richieste di dati (HTTP GET).

API è acronimo di Application Programming Interface, si tratta di software intermediario che permette a due applicazioni di comunicare tra di loro. Ogni volta che tramite una app come Facebook, invii un messaggio istantaneo, stai usando una API.

Il codice python adibito alla comunicazione con i server Twitter, Inc. è basato sull'utilizzo della libreria tweepy, che automatizza parte delle comunicazioni con le APIs.

È stato possibile rendere il codice open source tramite l'utilizzo di variabili d'ambiente (.env), che sono contenute in file che non vengono caricati su github e che permettono il salvataggio sicuro delle credenziali e dei dati sensibili di noi programmatori (i quattro token forniti dal portale sviluppatori twitter). Per accedere a questi file in modo sicuro e salvare il contenuto delle variabili, è necessaria la libreria python binaria getenv.

Le informazioni ottenute con le APIs, vengono salvate localmente in un file csv. Csv è acronimo di Comma Separated Values; si usa per salvare dati in un file di testo che possiamo immaginare come una tabella in cui le celle sono divise da una virgola. Proprio per questo, i file csv, di default, vengono aperti con programmi di spreadsheet (fogli di calcolo).