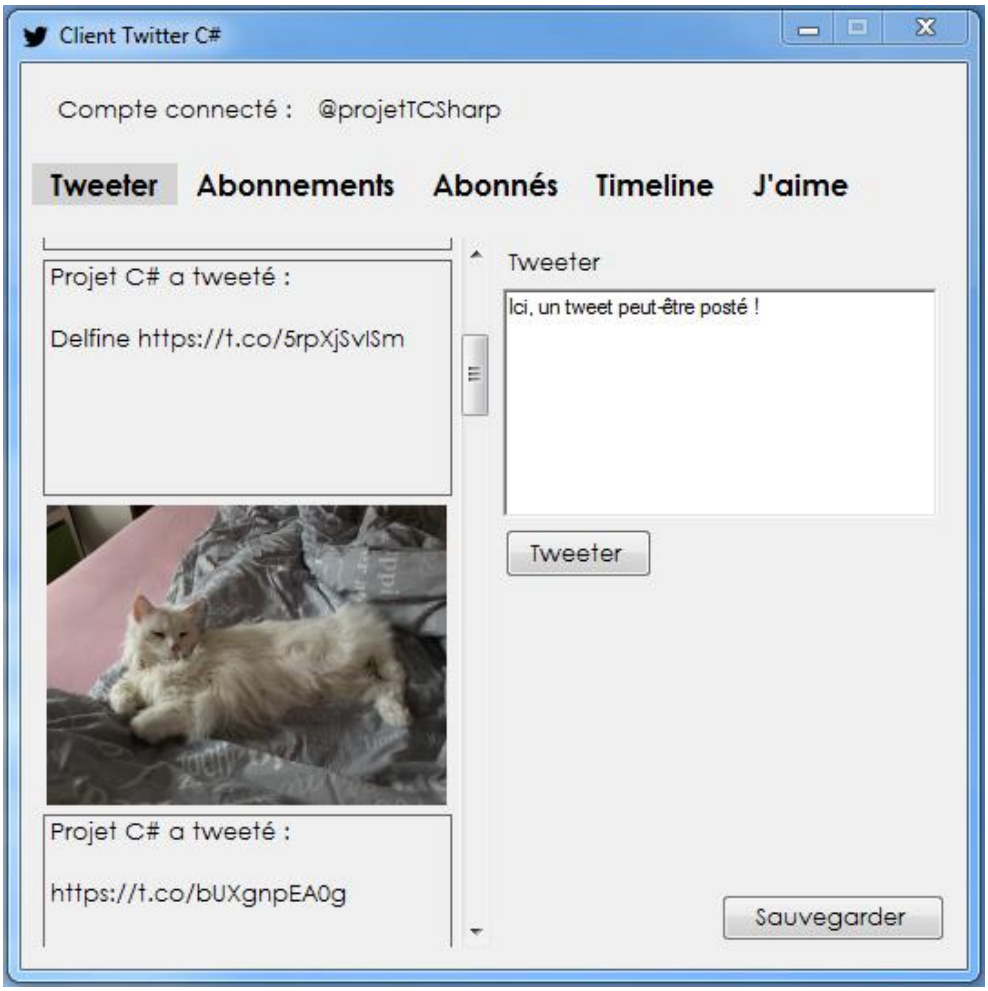
Client Twitter en C#



Alison Savary – FIN 2

ETML, Lausanne

4 semaine

Jonathan Melly

Table des matières

1 SPÉCIFICATIONS 3
1.1 TITRE 3
1.2 DESCRIPTION 3
1.3 MATÉRIEL ET LOGICIELS À DISPOSITION 3
1.4 PRÉREQUIS 3
2 PLANIFICATION INITIALE 4
3 ANALYSE 4
3.1 OPPORTUNITÉS 4
3.2 DOCUMENT D'ANALYSE ET CONCEPTION 4
3.3 CONCEPTION DES TESTS 6
4 RÉALISATION 6
4.1 DOSSIER DE RÉALISATION 6
4.1.1 Interface graphique 7
4.1.2 Authentification avec OAuth 1.0 8
4.1.3 Requêtes POST 14
4.1.4 Requêtes GET 16
4.1.5 Affichage des informations 18
4.2 MODIFICATIONS 19
4.2.1 Sauvegarder les tweets dans un fichier texte externe 19
4.2.2 Gérer l'affichage des images des tweets 20
4.2.3 Schéma 21
5 TESTS 21
5.1 DOSSIER DES TESTS 21
6 CONCLUSION 25
6.1 BILAN DES FONCTIONNALITÉS DEMANDÉES 25
6.2 BILAN DE LA PLANIFICATION 25
6.3 BILAN PERSONNEL 26
7 DIVERS 26
7.1 JOURNAL DE TRAVAIL 26
7.2 WEBOGRAPHIE 26

1 SPÉCIFICATIONS

1.1 Titre

Client Twitter en C# avec utilisation de l'API REST de Twitter

1.2 Description

En prévision du pré-TPI et TPI, monsieur Melly a donné ce projet pour rester dans les domaines du TPI qui sont le web et la programmation. Ce projet permet de découvrir l'architecture REST et les API, ainsi que la communication avec les requêtes http depuis un code en C#.

1.3 Matériel et logiciels à disposition

- 1 PC de labo avec Windows 7
- Visual Studio 2015
- Librairie JSON.NET (Version 10.0.1)
- Word
- Excel
- .NET Version 4.6

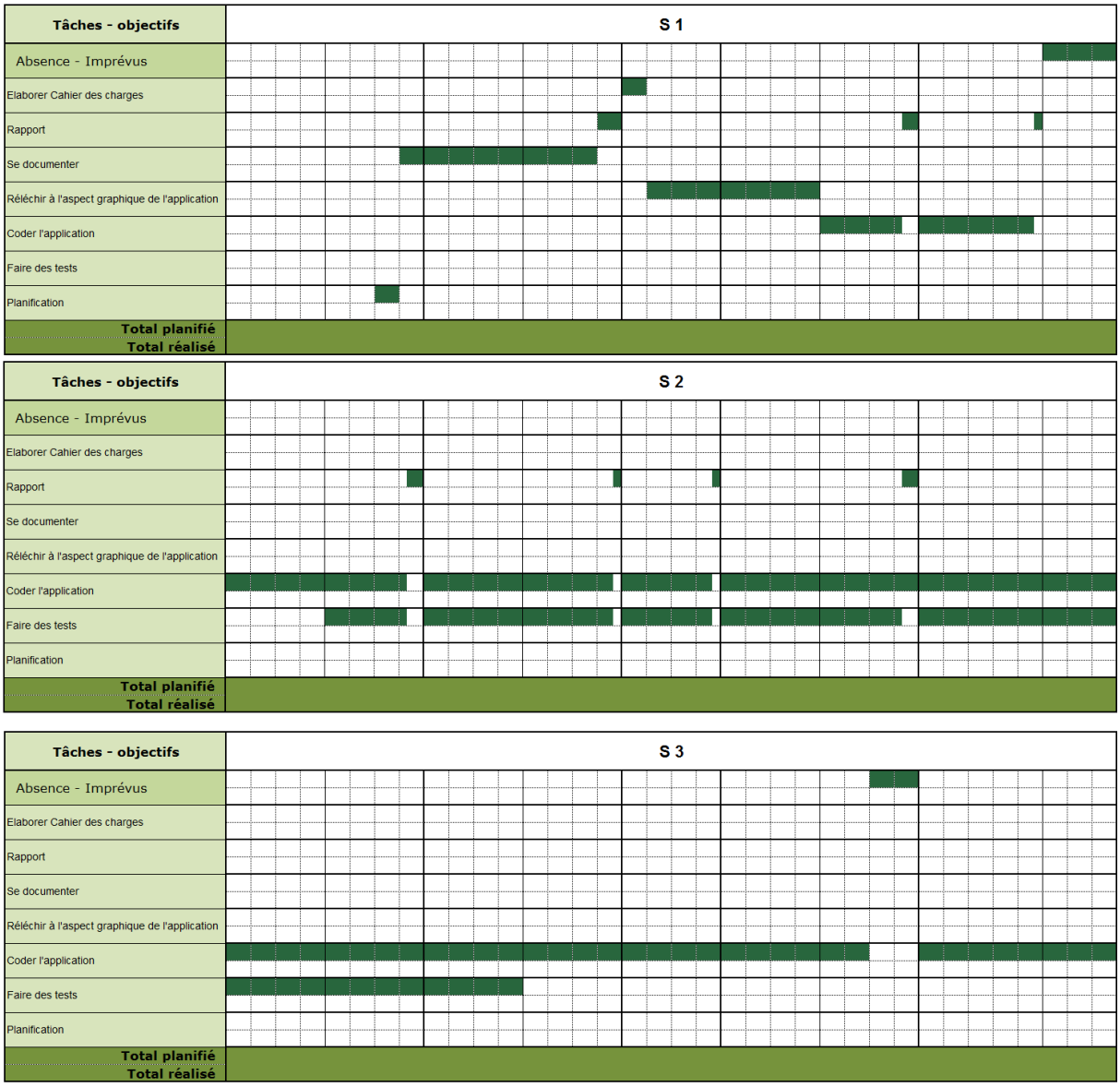
1.4 Prérequis

Avoir suivi les modules 103 et 303 de l'ETML.

Posséder un compte Twitter associé à un numéro de téléphone afin de pouvoir créer une application et obtenir les jetons pour la communication avec l'API grâce à OAuth. Installer Newtonsoft.Json à partir de NuGet sur Visual Studio.

2 PLANIFICATION INITIALE

Le projet se déroule du 13 mars au 7 avril 2017 sur un total de 144 périodes.



La semaine quatre n'est pas vraiment planifiée au départ et les semaines deux et trois ont subi plusieurs modifications lors de l'avancement du projet.

3 ANALYSE

3.1 Opportunités

Découvrir l'architecture REST. Apprendre à communiquer en C# avec une API REST et les requêtes http sans utiliser de librairie. Découvrir et traiter le langage JSON.

3.2 Document d'analyse et conception

L'application est un client Twitter. Twitter est un réseau social permettant d'écrire de petits textes de 140 caractères au maximum, il est possible de suivre des personnes et d'être suivi.

Voici la maquette de l'application, les fonctionnalités sont détaillées après les captures suivantes (Figures 1 à 3).

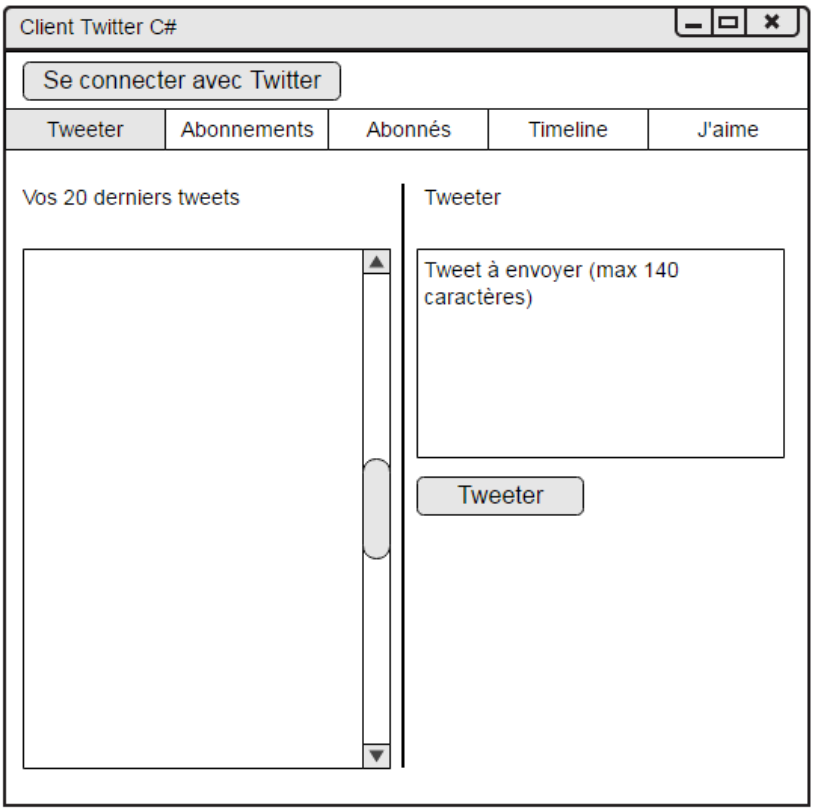


Figure 1 : Interface permettant de tweeter

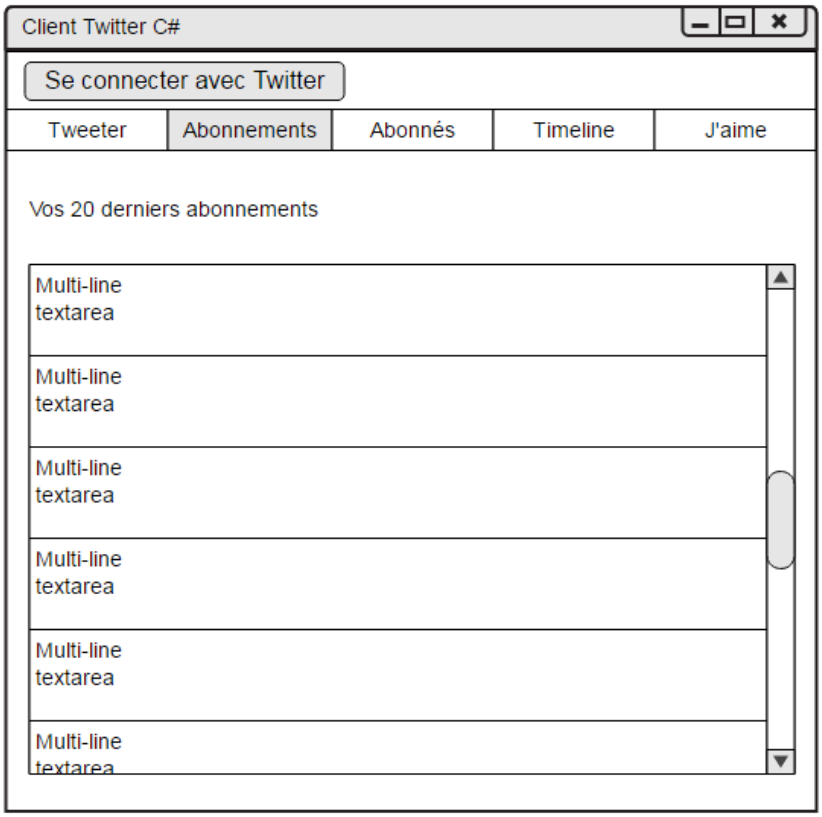


Figure 2 : Interface permettant de voir ses abonnements ou abonnés

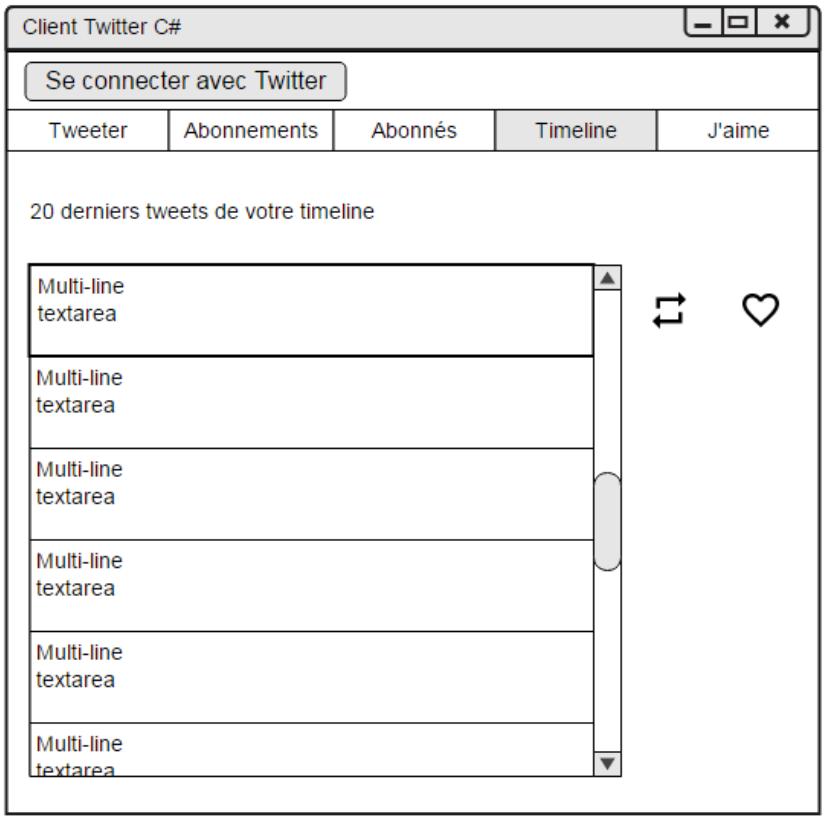


Figure 3 : Interface permettant de voir sa timeline ou les tweets aimés

L'application dispose de plusieurs fonctionnalités disponibles grâce à un menu de navigation.

Le premier « menu » : Tweeter, permet à l'utilisateur de voir ses 20 derniers tweets et d'envoyer des tweets. Le deuxième et troisième « menu » : Abonnements et Abonnés, permettent de voir respectivement ses 20 derniers abonnements et abonnés. Le quatrième « menu » : Timline, permet de voir les 20 derniers tweets composant sa timeline. Pour finir, le « menu » J'aime, permet de voir les derniers tweets aimés. Les tweets de la timeline et les tweets aimés offrent la possibilité de les aimer, ne plus aimer et de les retweeter, ne plus les retweeter.

Pour réaliser cette application, il est indispensable de pouvoir communiquer avec l'API REST de Twitter.

3.3 Conception des tests

Les tests sont effectués depuis le poste de travail utilisé pour concevoir l'application. Ils se font selon les scénarios décrit au point 5.1.

4 RÉALISATION

4.1 Dossier de Réalisation

Pour ce projet, un ordinateur permettant l'utilisation de Visual Studio 2015 est nécessaire. Visual Studio 2015 doit être installé, ainsi qu'une librairie complémentaire pour traiter le JSON (JavaScript Object Notation). La librairie utilisée est Newtonsoft.Json, Json.NET, elle est disponible dans les packages NuGet et facile à installer. La version actuelle de l'API Twitter est la version 1.1.

Avant la réalisation du code de l'application, une réflexion sur les fonctionnalités est nécessaire ainsi que la création d'un mockup. Cette analyse est donnée au point 3.2.

4.1.1 Interface graphique

Le client Twitter en C# a été pensé pour être réaliser avec Windows Forms. Premièrement il faut mettre en place l'interface graphique à l'aide des outils mis à disposition par Visual Studio. L'application est majoritairement composée de labels, boutons, conteneurs et d'un menu.

Une fois la mise en forme graphique terminée, le code permettant l'affichage de la bonne interface selon le menu sélectionné doit être réalisé.

Chaque partie du menu dispose d'une fonction rendant visible les éléments de l'interface associée (Par exemple, la Figure 4). Cette fonction en appelle deux autres, la première *hideAllInterfaces() (Voir Figure 5)* permet de cacher toutes les interfaces, la seconde *changeMenuColor()(Voir Figure 6)* permet d'accentuer la partie du menu actuellement sélectionné.

```
/// <summary>
/// Affiche l'interface qui permet de tweeter
/// </summary>
/// <param name="sender">Contrôle provoquant l'événement</param>
/// <param name="e">Données liées à l'événement</param>
private void tweeterToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Cacher toutes les interfaces
    hideAllInterfaces();
    // Afficher l'interface pour tweeter
    tweeterSplitContainer.Visible = true;

    // Création d'un objet ToolStripMenuItem
    ToolStripMenuItem selectedMenu = new ToolStripMenuItem();
    // Récupération du sender
    selectedMenu = sender as ToolStripMenuItem;
    // et envoi de l'objet en paramètre de la fonction
    changeMenuColor(selectedMenu);
}
```

Figure 4 : Exemple d'une des fonctions affichant l'interface voulue

```
/// <summary>
/// Cache les différentes interfaces graphiques
/// </summary>
private void hideAllInterfaces()
{
    tweeterSplitContainer.Visible = false;
    followingFollowersSplitContainer.Visible = false;
    timelineLikeSplitContainer.Visible = false;

    followingLabel.Visible = false;
    followersLabel.Visible = false;
    likeLabel.Visible = false;
    timelineLabel.Visible = false;
    LikePictureBox.Visible = false;
    retweetPictureBox.Visible = false;
}
```

Figure 5 : Fonction cachant toutes les interfaces

```
/// <summary>
/// Change la couleur de fond de l'élément du menu sélectionné
/// et s'assure que le reste du menu est de couleur neutre.
/// </summary>
/// <param name="selectedMenu">Elément du menu à mettre en couleur</param>
private void changeMenuColor(ToolStripMenuItem selectedMenu)
{
    // Remet la couleur par défaut à tous les éléments du menu
    tweeterToolStripMenuItem.BackColor = DefaultBackColor;
    followingToolStripMenuItem.BackColor = DefaultBackColor;
    followersToolStripMenuItem.BackColor = DefaultBackColor;
    timelineToolStripMenuItem.BackColor = DefaultBackColor;
    likeToolStripMenuItem.BackColor = DefaultBackColor;

    // Accentue le menu sélectionné
    selectedMenu.BackColor = Color.LightGray;
}
```

Figure 6 : Fonction accentuant le menu sélectionné

Quelques petits réglages sur les contrôles de la Form sont nécessaires pour terminer cette étape. Il faut empêcher l'utilisateur d'agrandir la fenêtre, ainsi que fixer les séparateurs des différents conteneurs. En plus de la vérification faite dans le code, il est conseillé de limiter la taille de la zone de texte à 140 caractères, étant la limite officielle pour un tweet.

4.1.2 Authentification avec OAuth 1.0

Une fois l'interface graphique terminée et fonctionnelle, la prochaine étape est de réussir à créer le header pour l'authentification nécessaire à chaque requête. Pour se faire, il existe plusieurs librairies compatibles avec le C# permettant une communication facilitée avec l'API, mais dans le cadre de ce projet, le but est de communiquer avec l'API de Twitter sans utiliser de librairie tierce. Un tutoriel sur le web explique et décrit cette étape à cette adresse : How To Post A Tweet Using C# For Single User (en anglais).

La première étape consiste en la création d'une application via le site https://apps.twitter.com/. Cette étape permet d'obtenir des droits et codes pour communiquer avec l'API de Twitter. Il faut faire attention à ce que l'application possède bien les droits en lecture et écriture, sinon il sera impossible d'envoyer des requêtes avec POST. L'accès et les autorisations pour communiquer avec l'API se font avec OAuth, il faut donc commencer par créer une fonction permettant de générer l'en-tête d'authentification. Celui-ci est composé de plusieurs paramètres :

- oauth_consumer_key : Clé disponible depuis la gestion de l'application sur https://apps.twitter.com/
- oauth _nonce : Une chaine de caractère aléatoire qui doit être différente pour chaque requête
- oauth_signature : Signature générée à partir des paramètres OAuth et de ceux de la requête, une fonction est dédiée à sa création
- oauth_signature_method : Méthode de hashage utilisée (HMAC-SHA1)
- oauth_timestamp : Timestamp correspondant à l'envoi de la requête
- oauth_token : Token disponible depuis la gestion de l'application sur https://apps.twitter.com/
- oauth_version : Version de OAuth utilisée (1.0)

Voici un schéma représentant comment sont obtenus la Consumer Key, le Consumer Secret et le Token et le Token Secret, voir Figure 7.

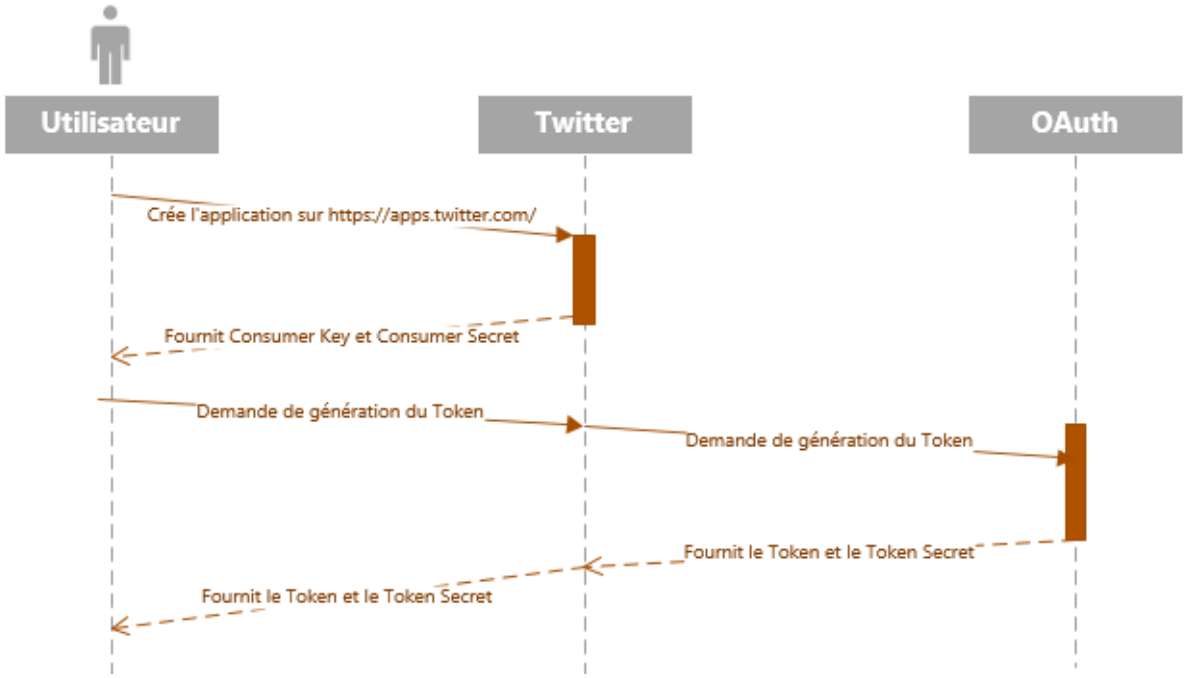


Figure 7 : Obtention des paramètres OAuth

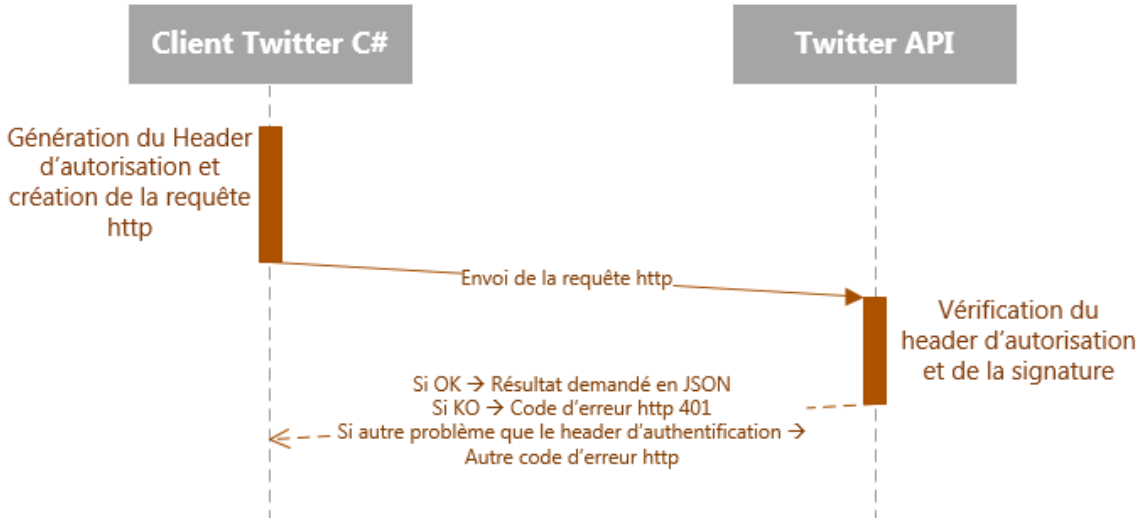


Figure 8 : Vérification de la requête http

Les différents paramètres donnés par la gestion de l'application sur le site prévu à cet effet, peuvent être stockés de manière indirecte dans l'application afin de pouvoir y accéder sans les laisser visibles dans le code principal. Pour cela, il faut que la référence System.Configuration soit présente, sinon il faut l'ajouter.

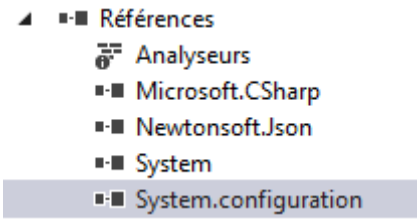


Figure 9 : Référence System.configuration

Ensuite il suffit d'ajouter les valeurs dans le fichier App.config comme sur la Figure 10.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
    </startup>
  <appSettings>
    <add key="consumerKey" value="[redacted]"/>
    <add key="consumerSecret" value="[redacted]"/>
    <add key="accessToken" value="[redacted]"/>
    <add key="accessTokenSecret" value="[redacted]"/>
  </appSettings>
</configuration>
```

Figure 10 : Fichier App.config

La prochaine étape est la réalisation d'une fonction pour générer le oauth_nonce, n'importe quelle fonction réalisant une chaine de caractères alphanumérique aléatoire fonctionne. Ensuite vient la fonction permettant de générer le timestamp, elle possède un paramètre de type date, c'est la date qui est transmise au moment de la création de la requête http. La fonction est reprise du tutoriel mentionné précédemment.

Il reste la partie la plus difficile, la génération totale du header et la création de la signature OAuth. La fonction générant le header est simplement une fonction ajoutant tous les paramètres OAuth dans une chaine de caractères. Pour le client Twitter, la fonction utilisée est la même que dans le tutoriel mais avec quelques adaptations pour prendre en charge les autres fonctionnalités du client. Elle possède deux paramètres qui sont ensuite transmis à la fonction générant la signature OAuth.

```
/// <summary>
/// Génère le header "Authorizartion" pour la requête http
/// </summary>
/// <param name="strParameter">Paramètre faisant parti de la requête http comme un nom d'utilisateur ou le tweet</param>
/// <returns>Un string contenant le header</returns>
private string GenerateAuthorizationHeader(string strParameter, string requestType)
{
    // Genère le oauth_nonce et le oauth_timestamp
    string strNonce = GenerateNonce();
    double timestamp = GenerateTimestamp(DateTime.Now);

    string dst = string.Empty;
    // Création du header
    dst += "OAuth ";
    dst += string.Format("oauth_consumer_key=\"{0}\", ", Uri.EscapeDataString(strOauthConsumerKey));
    dst += string.Format("oauth_nonce=\"{0}\", ", Uri.EscapeDataString(strNonce));
    dst += string.Format("oauth_signature=\"{0}\", ", Uri.EscapeDataString(GenerateOauthSignature(strParameter, strNonce, timestamp.ToString(), requestType)));
    dst += string.Format("oauth_signature_method=\"{0}\", ", Uri.EscapeDataString(STR_OAUTH_SIGNATURE_METHODE));
    dst += string.Format("oauth_timestamp=\"{0}\", ", timestamp);
    dst += string.Format("oauth_token=\"{0}\", ", Uri.EscapeDataString(strOauthAccessToken));
    dst += string.Format("oauth_version=\"{0}\"", Uri.EscapeDataString(STR_OAUTH_VERSION));
    return dst;
}
```

Figure 11 : Fonction de génération du Header

La génération de la signature se fait dans une autre fonction dédiée, elle aussi reprise et adaptée. Elle reprend les paramètres OAuth sans la signature et les ajoute dans une chaine de caractères.

```
/// <summary>
/// Génére la signature oauth
/// </summary>
/// <param name="strParameter">Paramètre faisant parti de la requête http comme un nom d'utilisateur ou le tweet</param>
/// <param name="strNonce">Chaine de caractère aléatoire générée pour chaque requête</param>
/// <param name="timestamp">Timestamp de la date de la requête</param>
/// <returns>La signature oauth sous forme de string</returns>
private string generateOauthSignature(string strParameter, string strNonce, string timestamp, string strRequestType)
{

    string result = string.Empty; // Elément à hasher et signer
    string dst = string.Empty; // Partie spécifications OAuth à signer en combinant avec l'URL

    dst += string.Format("oauth_consumer_key={0}&", Uri.EscapeDataString(strOauthConsumerKey));
    dst += string.Format("oauth_nonce={0}&", Uri.EscapeDataString(strNonce));
    dst += string.Format("oauth_signature_method={0}&", Uri.EscapeDataString(STR_OAUTH_SIGNATURE_METHODE));
    dst += string.Format("oauth_timestamp={0}&", timestamp);
    dst += string.Format("oauth_token={0}&", Uri.EscapeDataString(strOauthAccessToken));
    dst += string.Format("oauth_version={0}", Uri.EscapeDataString(STR_OAUTH_VERSION));
```

Figure 12 : Fonction de génération de la signature (Début)

La chaine de caractères se finit différemment selon le type de requête, ce choix est géré avec le paramètre *requestType*, le paramètre *strParameter* contient une valeur permettant de préciser la requête (un tweet à poster, un id d'utilisateur ou un id de tweet).

```
    // Selon le type de requête qui sera envoyée, ajoute la dernière ligne correspondante
    switch (strRequestType)
    {
        case "tweet":
            {
                dst += string.Format("&status={0}", Uri.EscapeDataString(strParameter));
                break;
            }
        case "findTweets":
            {
                dst += string.Format("&user_id={0}", Uri.EscapeDataString(strParameter));
                break;
            }
        case "findFollowings":
            {
                dst += string.Format("&user_id={0}", Uri.EscapeDataString(strParameter));
                break;
            }
        case "findFollowers":
            {
                dst += string.Format("&user_id={0}", Uri.EscapeDataString(strParameter));
                break;
            }
        case "findLikes":
            {
                dst += string.Format("&user_id={0}", Uri.EscapeDataString(strParameter));
                break;
            }
        case "likeTweet":
            {
                dst += string.Format("&id={0}", Uri.EscapeDataString(strParameter));
                break;
            }
        case "unlikeTweet":
            {
                dst += string.Format("&id={0}", Uri.EscapeDataString(strParameter));
                break;
            }
    }
```

Figure 13 : Fonction de génération de la signature (Suite)

La clé est constituée du ConsumerSecret et du AccessTokenSecret, qui sont visibles depuis le site de gestion des applications Twitter, les deux valeurs sont échappées et séparées par un &.

```
// Création de la clé pour la signature
string signingKey = string.Empty;
signingKey = string.Format("{0}&{1}", Uri.EscapeDataString(strOauthConsumerSecret), Uri.EscapeDataString(strOauthAccessTokenSecret));
```

Figure 14 : Fonction de génération de la signature (Clé)

La signature est ensuite générée à partir du verbe http, de l'url de la requête et de la chaine de caractères générée précédemment. Le résultat est ensuite haché, signé et retourné pour le header.

```
string oauth_signature;
// Selon le type de requête, création de la signature correspondante
switch (strRequestType)
{
    case "tweet":
        {
            result += "POST&";
            result += Uri.EscapeDataString(strPostTweetUrl);
            result += "&";
            result += Uri.EscapeDataString(dst);
            using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
            {
                oauth_signature = Convert.ToBase64String(
                    hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
            }
            return oauth_signature;
        }
    case "findTweets":
        {
            result += "GET&";
            result += Uri.EscapeDataString(strGetTweetUrl);
            result += "&";
            result += Uri.EscapeDataString(dst);
            using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
            {
                oauth_signature = Convert.ToBase64String(
                    hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
            }
            return oauth_signature;
        }
    case "findFollowings":
        {
            result += "GET&";
            result += Uri.EscapeDataString(strGetFollowingsUrl);
            result += "&";
            result += Uri.EscapeDataString(dst);
            using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
            {
                oauth_signature = Convert.ToBase64String(
                    hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
            }
            return oauth_signature;
        }
```

Figure 15 : Fonction de génération de la signature (Suite)

```
case "likeTweet":
    {
        result += "POST&";
        result += Uri.EscapeDataString(strPostLikeTweetUrl);
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
case "unlikeTweet":
    {
        result += "POST&";
        result += Uri.EscapeDataString(strPostUnlikeTweetUrl);
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
case "retweet":
    {
        result += "POST&";
        result += Uri.EscapeDataString(string.Format(strPostRetweetUrl, strLastClickedTweetID));
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
case "unretweet":
    {
        result += "POST&";
        result += Uri.EscapeDataString(string.Format(strPostUnretweetUrl, strLastClickedTweetID));
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
```

Figure 16 : Fonction de génération de la signature (Suite)

```
case "findFollowers":
    {
        result += "GET&";
        result += Uri.EscapeDataString(strGetFollowersUrl);
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
case "findTimeline":
    {
        result += "GET&";
        result += Uri.EscapeDataString(strGetTimelineUrl);
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
case "findLikes":
    {
        result += "GET&";
        result += Uri.EscapeDataString(strGetLikesUrl);
        result += "&";
        result += Uri.EscapeDataString(dst);
        using (HMACSHA1 hasher = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(signingKey)))
        {
            oauth_signature = Convert.ToBase64String(
                hasher.ComputeHash(ASCIIEncoding.ASCII.GetBytes(result)));
        }
        return oauth_signature;
    }
```

Figure 17 : Fonction de génération de la signature (Fin)

4.1.3 Requêtes POST

Les requêtes POST sont utilisées lorsque le but est de faire une action sur le serveur concerné. Dans le cas de Twitter, une requête POST est utilisée pour poster un nouveau tweet ou faire une interaction sur un tweet comme le retweeter. La première requête POST du client Twitter est la requête permettant de poster un tweet. L'url utilisée pour celle-ci est la suivante : https://api.twitter.com/1.1/statuses/update.json. La fonction permettant de créer et envoyer cette requête a besoin d'un paramètre, le texte du tweet, qui est récupéré de la zone de texte prévue à cet effet. Pour éviter au maximum les erreurs, le texte est vérifié avant l'appel de la fonction envoyant la requête http.

```
/// <summary>
/// Traite le texte à tweeter avant de l'envoyer à la fonction créant la requête http
/// </summary>
/// <param name="sender">Contrôle provoquant l'événement</param>
/// <param name="e">Données liées à l'événement</param>
private void tweeterButton_Click(object sender, EventArgs e)
{
    // Tweet à poster
    string strStatus = tweeterRichTextBox.Text;

    // N'y a-t-il que des espaces ?
    bool boolOnlySpaces = true;

    // Vérifie que le texte n'est pas constitué uniquement d'espace
    for (int i = 0; i < strStatus.Length; i++)
    {
        if (strStatus[i] != Convert.ToChar(" "))
        {
            boolOnlySpaces = false;
        }
    }

    // Si le tweet n'est pas vide et ne dépasse pas les 140 caractères propre à Twitter, il est envoyé.
    if (strStatus.Length <= 140 && strStatus.Length > 0 && !boolOnlySpaces)
    {
        sendTweet(strStatus);
        tweeterRichTextBox.Text = "";
        // Provoque la réactualisation des derniers tweets
        findLastTweets();
    }
    else
    {
        MessageBox.Show("Votre tweet doit faire entre 1 et 140 caractères !", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

Figure 18 : Fonction vérifiant le tweet

Si le tweet passe les tests effectués, il est ensuite envoyé à la fonction créant la requête http. La création de la requête POST en elle-même se fait dans une fonction dédiée afin d'éviter la répétition du même code pour toutes les requêtes POST différentes. Elle prend en paramètre le header d'authentification, l'url et le body de la requête.

```
/// <summary>
/// Crée une requête http POST avec les paramètres donnés
/// </summary>
/// <param name="strAuthHeader">Header d'authentification</param>
/// <param name="strPostUrl">URL Resource</param>
/// <param name="strPostBody">Body de la requête</param>
/// <returns>Requête http POST prête à être envoyée </returns>
private WebRequest createPostRequest(string strAuthHeader, string strPostUrl, string strPostBody)
{
    // Création de la requête avec l'url
    HttpWebRequest postRequest = (HttpWebRequest)WebRequest.Create(strPostUrl);
    // Ajout du header
    postRequest.Headers.Add("Authorization", strAuthHeader);
    // Ajout de la méthode, verbe http
    postRequest.Method = "POST";
    postRequest.UserAgent = "OAuth gem v0.4.4";
    // Ajout de l'hôte
    postRequest.Host = "api.twitter.com";
    // Spécification du type de contenu
    postRequest.ContentType = "application/x-www-form-urlencoded";
    postRequest.ServicePoint.Expect100Continue = false;
    postRequest.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

    // Ajout du contenu du body de la requête s'il y en a un.
    if (strPostBody.Length > 0)
    {
        using (Stream stream = postRequest.GetRequestStream())
        {
            byte[] content = Encoding.UTF8.GetBytes(strPostBody);
            stream.Write(content, 0, content.Length);
        }
    }

    return postRequest;
}
```

Figure 19 : Fonction créant une requête http POST

La fonction s'occupant de l'envoi de la requête génère le header spécifique ainsi que le body de la requête, il les envoie ensuite à la fonction créant la requête avec les bons paramètres, qui à son tour retransmet la requête prête à être envoyée.

```
/// <summary>
/// Envoie la requête http qui permet de poster un tweet et gère les erreurs
/// </summary>
/// <param name="strTweet">Tweet à poster</param>
private void sendTweet(string strTweet)
{
    // Génération du header avec les paramètres d'authentification
    string strAuthHeader = generateAuthorizationHeader(strTweet, "tweet");
    // Création du body de la requête
    string strPostBody = "status=" + Uri.EscapeDataString(strTweet);

    WebRequest postRequest = createPostRequest(strAuthHeader, strPostTweetUrl, strPostBody);

    try // Permet la gestion des exceptions si le serveur retourne une erreur.
    {
        // Envoi de la requête
        WebResponse postResponse = postRequest.GetResponse();
        postResponse.Close();
    }
    catch (WebException e) // Si une erreur se produit
    {
        // Récupère le message d'erreur et l'affiche dans une MessageBox
        string strErreur = Convert.ToString(e.Message);
        MessageBox.Show(strErreur, "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Figure 20 : Fonction provoquant l'envoi d'un nouveau tweet

Les autres requêtes utilisant POST (retweeter, ne plus retweeté, aimer, ne plus aimer un tweet) utilisent toutes la fonction de la Figure 19. Chaque requête à sa propre fonction afin de pouvoir générer le bon header selon les paramètres voulus.

4.1.4 Requêtes GET

Dès lors qu'il s'agit d'obtenir une information de l'API, la requête utilise la méthode GET. La première requête utilisée par le client est celle qui permet d'obtenir ses 20 derniers tweets. Cette requête est faite à partir de l'url https://api.twitter.com/1.1/statuses/user_timeline.json et possède un paramètre, l'*user_id* (identifiant numérique de l'utilisateur Twitter) duquel les tweets sont récupérés. Le header de la requête est d'abord généré grâce à la fonction prévue à cet effet. La requête est ensuite créée sous forme de WebRequest avec l'url et son paramètre. Le header, ainsi que d'autre propriétés de la requête, sont ajoutés. Comme pour la requête POST, une fonction est dédiée à sa création, et est utilisée pour toutes les requêtes GET à venir.

```
/// <summary>
/// Crée une requête http GET avec les paramètres donnés
/// </summary>
/// <param name="strAuthHeader">Header d'authentification</param>
/// <param name="strGetUrl">URL Resource</param>
/// <returns>Requête http GET prête à être envoyée </returns>
private WebRequest createGetRequest(string strAuthHeader, string strGetUrl)
{
    // Création de la requête avec l'url suivi des paramètres
    HttpWebRequest getRequest = (HttpWebRequest)WebRequest.Create(strGetUrl);
    // Ajout du header
    getRequest.Headers.Add("Authorization", strAuthHeader);
    // Ajout de la méthode, verbe http
    getRequest.Method = "GET";
    getRequest.UserAgent = "OAuth gem v0.4.4";
    // Ajout de l'hôte
    getRequest.Host = "api.twitter.com";
    // Spécification du type de contenu
    getRequest.ContentType = "application/x-www-form-urlencoded";
    getRequest.ServicePoint.Expect100Continue = false;
    getRequest.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;

    return getRequest;
}
```

Figure 21 : Fonction créant une requête http GET

Chaque requête dispose de sa propre fonction pour l'envoi de la requête ainsi que le traitement des informations reçues en langage JSON. Le début de la fonction est similaire aux fonctions utilisant des requêtes POST, la principale différence consiste à récupérer dans un string les informations reçues avant de fermer la requête.

```
/// <summary>
/// Crée la requête http permettant d'obtenir les 20 derniers tweets postés par l'utilisateur
/// et l'envoie. Le résultat est obtenu en JSON. Traitement et affichage des informations reçues.
/// </summary>
private void findLastTweets()
{
    // Génération du header avec les paramètres d'authentification
    string strAuthHeader = generateAuthorizationHeader(strUserId, "findTweets");
    // Gestion des paramètres de la requête
    string postbody = "user_id=" + Uri.EscapeDataString(strUserId);

    WebRequest getRequest = createGetRequest(strAuthHeader, strGetTweetUrl + "?" + postbody);

    // Vide la liste de PictureBox qui contiennent les images des tweets
    for(int i = 0; i < list_tweetImagesPictureBox.Count; i = 0)
    {
        tweeterSplitContainer.Panel1.Controls.Remove(list_tweetImagesPictureBox[i]);
        list_tweetImagesPictureBox[i].Dispose();
        list_tweetImagesPictureBox.Remove(list_tweetImagesPictureBox[i]);
    }
    // Index pour la liste de PictureBox
    int intIndex = 0;
    int intCpt = 0; // Compteur du nombre d'exécution de la boucle foreach
    int intY = 30; // Position Y de départ des Label

    try // Essaie d'envoyer la requête, s'il n'y a pas d'erreur, continue.
    {
        // Envoi de la requête
        WebResponse getResponse = getRequest.GetResponse();
        // Récupération des données reçues et stockage dans un string
        StreamReader reader = new StreamReader(getResponse.GetResponseStream());
        string strJson = reader.ReadToEnd();
        getResponse.Close();
```

Figure 22 : Fonction permettant de récupérer les 20 derniers tweets (Début)

La suite de la fonction est différente selon la fonction, car il s'agit de traiter le résultat reçu en JSON afin de l'afficher dans l'application. Par exemple, voici la suite de la fonction obtenant les 20 derniers tweets postés par l'utilisateur. Plus tard dans le projet, le traitement des images des tweets a été demandé, cette partie est visible ici, mais elle est expliquée en détail au point 4.2.2.

```
        // Création d'un tableau JSON avec le string des données reçues
        jArrayTweets = JArray.Parse(strJson);

        // Parcourt le tableau JSON et pour chaque tweet,
        foreach (JObject joTweet in jArrayTweets.Children())
        {
            // ajoute le nom d'utilisateur et le texte du tweet dans un label
            tab_tweetsLabel[intCpt].Text = Convert.ToString(joTweet["user"]["name"]) + " a tweeté : \n\n" + Convert.ToString(joTweet["text"]);
            // Définit la position du label
            tab_tweetsLabel[intCpt].Location = new System.Drawing.Point(0, intY);
            // Ajoute le label dans le conteneur voulu
            tweeterSplitContainer.Panel1.Controls.Add(tab_tweetsLabel[intCpt]);

            try // Si le tweet possède des médias
            {
                JArray jaMedia = (JArray)joTweet["extended_entities"]["media"];
                int intCptMedia = 0;
                foreach (JObject joMedia in jaMedia)
                {
                    // Si le media est bien une photo
                    if (Convert.ToString(jaMedia[intCptMedia]["type"]) == "photo")
                    {
                        // Ajoute une PictureBox avec l'image
                        list_tweetImagesPictureBox.Add(new PictureBox());
                        string strMediaUrl = Convert.ToString(jaMedia[intCptMedia]["media_url"]); // Donne l'url .jpg
                        intY += 135;
                        list_tweetImagesPictureBox[intIndex].Load(strMediaUrl);
                        list_tweetImagesPictureBox[intIndex].Name = "tweetImagePictureBox";
                        list_tweetImagesPictureBox[intIndex].Size = new System.Drawing.Size(225, 165);
                        list_tweetImagesPictureBox[intIndex].Location = new System.Drawing.Point(0, intY);
                        list_tweetImagesPictureBox[intIndex].SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
                        tweeterSplitContainer.Panel1.Controls.Add(list_tweetImagesPictureBox[intIndex]);
                        intIndex++;
                        intY += 35;
                    }
                    intCptMedia++;
                }
            }
            catch (NullReferenceException e)
            {
            }
            // Auguemente la position Y et le compteur
            intY += 135;
            intCpt++;
        }
    }
    catch (WebException e)
    {
        string strErreur = Convert.ToString(e.Message);
        MessageBox.Show(strErreur, "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Figure 23 : Fonction permettant de récupérer les 20 derniers tweets (Fin)

Les fonctions pour obtenir ses abonnements ou abonnés, sa timeline ou ses tweets aimés suivent le même principe.

4.1.5 Affichage des informations

Tous les tweets récupérés sont affichés dans l'application grâce à des labels créés dans une fonction au chargement de l'application. Ils sont regroupés en trois tableau différents selon leurs dimensions.

```
// Tableaux de label pour l'affichage des données récupérées
private Label[] tab_tweetsLabel = new Label[20]; // Derniers tweets postés
private Label[] tab_FollowingsFollowersLabel = new Label[20]; // Abonnements ou abonnés
private Label[] tab_TimelineLikesLabel = new Label[20]; // Timeline ou les tweets aimés
```

Figure 24 : Déclarations des tableaux de labels

```
/// <summary>
/// Crée les tableaux de labels permettant d'afficher différentes données
/// </summary>
private void createLabelsTables()
{
    for (int i = 0; i < 20; i++)
    {
        // Tableau de label pour les 20 derniers tweets
        this.tab_tweetsLabel[i] = new Label();
        this.tab_tweetsLabel[i].AutoSize = false;
        this.tab_tweetsLabel[i].Font = new System.Drawing.Font("Century Gothic", 9.75F, System.Drawing.FontStyle.Regular,
        this.tab_tweetsLabel[i].Name = "tweetLabel" + i;
        this.tab_tweetsLabel[i].Size = new System.Drawing.Size(225, 130);
        this.tab_tweetsLabel[i].BorderStyle = BorderStyle.FixedSingle;
        this.tab_tweetsLabel[i].Text = "";

        // Tableau de label pour les 20 derniers abonnements et abonnés
        this.tab_FollowingsFollowersLabel[i] = new Label();
        this.tab_FollowingsFollowersLabel[i].AutoSize = false;
        this.tab_FollowingsFollowersLabel[i].Font = new System.Drawing.Font("Century Gothic", 9.75F, System.Drawing.FontS
        this.tab_FollowingsFollowersLabel[i].Name = "followingLabel" + i;
        this.tab_FollowingsFollowersLabel[i].Size = new System.Drawing.Size(480, 20);
        this.tab_FollowingsFollowersLabel[i].BorderStyle = BorderStyle.FixedSingle;
        this.tab_FollowingsFollowersLabel[i].Text = "";

        // Tableau de label pour les 20 derniers tweets de la timeline et tweets aimés
        this.tab_TimelineLikesLabel[i] = new Label();
        this.tab_TimelineLikesLabel[i].AutoSize = false;
        this.tab_TimelineLikesLabel[i].Font = new System.Drawing.Font("Century Gothic", 9.75F, System.Drawing.FontStyle.R
        this.tab_TimelineLikesLabel[i].Name = "tweetLabel" + i;
        this.tab_TimelineLikesLabel[i].Size = new System.Drawing.Size(340, 110);
        this.tab_TimelineLikesLabel[i].BorderStyle = BorderStyle.FixedSingle;
        this.tab_TimelineLikesLabel[i].Text = "";
        // Ajout de la méthode Label_Click pour l'événement Click
        this.tab_TimelineLikesLabel[i].Click += new System.EventHandler(this.label_Click);
    }
}
```

Figure 25 : Fonction remplissant les tableaux de labels

4.2 Modifications

4.2.1 Sauvegarder les tweets dans un fichier texte externe

Durant le projet, Monsieur Melly a trouvé intéressant le fait de pouvoir sauvegarder les tweets. Il a donc demandé à ce que cette fonctionnalité soit implémentée. Pour se faire, il faut ajouter un bouton « Sauvegarder » qui déclenche la sauvegarde des tweets actuellement visibles. La fonction commence par définir où se situe l'utilisateur grâce à la couleur différente de l'élément du menu sélectionné puis parcourt le tableau de labels correspondant afin d'en récupérer les textes et les stocker dans un autre tableau prévu pour la sauvegarde.

```
/// <summary>
/// Sauvegarde les tweets dans un fichier texte
/// </summary>
/// <param name="sender">Contrôle provoquant l'événement</param>
/// <param name="e">Données liées à l'événement
/// </param>
private void saveTweetsButton_Click(object sender, EventArgs e)
{
    // Si l'on se trouve sur l'interface pour tweeter
    if(tweeterToolStripMenuItem.BackColor == Color.LightGray)
    {
        // Ajoute un titre au début du tableau
        tab_strSavedTweets[0] = "Vos derniers tweets : ";
        int intIndex = 1;
        for(int i = 0; i < tab_tweetsLabel.Length; i++)
        {
            // Récupère le contenu des labels
            tab_strSavedTweets[intIndex] = tab_tweetsLabel[i].Text;
            intIndex++;
        }
        // Crée est écrit le fichier
        System.IO.File.WriteAllLines(@"C:\Users\savaryal\Desktop\LastTweets.txt", tab_strSavedTweets);

    }
```

Figure 26 : Fonction sauvegardant les tweets

Les deux autres conditions de la fonction sont identiques, à l'exception du nom du fichier et de la première ligne du tableau des tweets sauvegardés. Elle termine par une MessageBox avertissant l'utilisateur que la sauvegarde a bien été effectuée. Pour tester cette fonctionnalité, il faut changer le chemin des fichiers et l'adapter à la session ouverte sur le poste de test.

4.2.2 Gérer l'affichage des images des tweets

Lors d'une démonstration des fonctionnalités réalisées et d'une demande d'aide pour une erreur sur les fonctionnalités aimer / ne plus aimer un tweet, la décision d'afficher les images des tweets a été prise. Pour la réalisation de cette fonctionnalité, il est nécessaire de pouvoir utiliser un tableau dynamique contenant des PictureBox. Or lors de la déclaration d'un tableau, il est nécessaire de lui donner une taille fixe, et il est impossible de prévoir combien d'images doit contenir le tableau. Après quelques recherches, il s'est avéré que l'utilisation d'une liste de PictureBox permet de surmonter ce problème.

```
// Liste dynamique de PictureBox pour l'affichage des images
private List<PictureBox> list_tweetImagesPictureBox = new List<PictureBox>();
```

Figure 27 : Déclaration d'une liste de PictureBox

Dans chaque fonction affichant des tweets, il faut ajouter, au début, le code permettant de vider la liste et de libérer la mémoire, afin que le programme ne prenne pas de plus en plus de mémoire à chaque changement d'affichage de tweets.

```
// Vide la liste de PictureBox qui contiennent les images des tweets
for(int i = 0; i < list_tweetImagesPictureBox.Count; i = 0)
{
    tweeterSplitContainer.Panel1.Controls.Remove(list_tweetImagesPictureBox[i]);
    list_tweetImagesPictureBox[i].Dispose();
    list_tweetImagesPictureBox.Remove(list_tweetImagesPictureBox[i]);
}
// Index pour la liste de PictureBox
int intIndex = 0;
```

Figure 28 : Boucle permettant de vider la liste de PictureBox

Puis, lors du parcourt de chaque tweet afin d'afficher leur texte dans un label, il faut vérifier si le tweet possède des médias de type « photo » et récupérer leur adresse. Ensuite, une PictureBox contenant l'image est créée et ajoutée à la liste, elle est ensuite ajoutée sous son tweet dans le panel s'occupant de leur affichage.

```
try // Si le tweet possède des médias
{
    JArray jaMedia = (JArray)joTweet["extended_entities"]["media"];
    int intCptMedia = 0;
    foreach (JObject joMedia in jaMedia)
    {
        // Si le media est bien une photo
        if (Convert.ToString(jaMedia[intCptMedia]["type"]) == "photo")
        {
            // Ajoute une PictureBox avec l'image
            list_tweetImagesPictureBox.Add(new PictureBox());
            string strMediaUrl = Convert.ToString(jaMedia[intCptMedia]["media_url"]); // Donne l'url .jpg
            intY += 135;
            list_tweetImagesPictureBox[intIndex].Load(strMediaUrl);
            list_tweetImagesPictureBox[intIndex].Name = "tweetImagePictureBox";
            list_tweetImagesPictureBox[intIndex].Size = new System.Drawing.Size(225, 165);
            list_tweetImagesPictureBox[intIndex].Location = new System.Drawing.Point(0, intY);
            list_tweetImagesPictureBox[intIndex].SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;
            tweeterSplitContainer.Panel1.Controls.Add(list_tweetImagesPictureBox[intIndex]);
            intIndex++;
            intY += 35;
        }
        intCptMedia++;
    }
}
catch (NullReferenceException e)
{
}
```

Figure 29 : Gestion des médias du tweet

4.2.3 Schéma

Un schéma représentant les interactions dans l'application est disponible sous : schemaEvenementiel.pdf.

5 TESTS

5.1 Dossier des tests

Les tests sont effectués selon les scénarios suivant. Le canevas des scénarios est pris du fichier créé par M. Melly se trouvant dans K:\INF\Maitres-Eleves\Modeles\TestsFonctionnels. Les tests sont directement effectués sur le poste où l'application a été créée. Pour les tests, la version exécutable « release » est utilisée.

Version de l'application testée	1.0.0
Date des tests	06.04.2017
Tests effectués par	Alison Savary

Scénario 1 : L'application se lance sans erreur

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Lancer l'exécutable	

Assert / Then	L'application affiche les 20 derniers tweets ou moins du compte Projet C# et l'interface permettant de tweeter. Les images sont bien disposées en dessous de leur tweet.	

Résultat : OK

Remarque :

Le séparateur du panel est focus à l'ouverture de l'application

Scénario 2 : Poster un tweet

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Ecrire dans la zone de test n'importe quel tweet et le poster en cliquant sur le bouton « Tweeter ».	Tester différent cas, tweet normal, vide, uniquement des espaces…
Assert / Then	Le tweet est posté et apparait à gauche au début de la liste de tweet. Si les tweets en dessous avaient des images, elles sont toujours disposées en dessous de leur tweet.	

Résultat : OK

Remarque :

Lors d'un tweet non conforme, un message avertit l'utilisateur et le tweet n'est pas posté.

Scénario 3 : Sauvegarder les tweets

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	Lancer l'application sur la session savaryal
Act / When	Sur n'importe quelle page affichant des tweets, appuyer sur le bouton « Sauvegarder »	Un message confirmant la sauvegarde apparait.
Assert / Then	Contrôler le contenu du fichier sauvegardé, contient les tweets avec le format : **nom du compte a tweeté : le texte du tweet.**	Les fichiers sont sauvegardé sur le bureau.

Résultat : OK

Remarque :

Ne fonctionne que si l'application est lancée sur la session savaryal.

Scénario 4 : Afficher les abonnements

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Cliquer sur le menu « Abonnements »	
Assert / Then	Contrôler que les 20 derniers abonnements ou moins sont affichés	

Résultat : OK

Scénario 5 : Afficher les abonnés

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Cliquer sur le menu « Abonnés »	
Assert / Then	Contrôler que les 20 derniers abonnés ou moins sont affichés	

Résultat : OK

Scénario 6 : Afficher la timeline

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Cliquer sur le menu « Timeline »	
Assert / Then	Les 20 derniers tweets de la timeline sont affichés et les images des tweets se trouvent sous le tweet correspondant.	

Résultat : OK

Scénario 7 : Afficher les tweets aimés

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Cliquer sur le menu « J'aime »	
Assert / Then	Les 20 derniers tweets aimés sont affichés et les images des tweets se trouvent sous le tweet correspondant. Si l'on clique sur les tweets, deux images apparaissent, le cœur est toujours plein.	

Résultat : OK

Scénario 8 : Retweeter / Ne plus retweeter

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Dans la partie « Timeline » ou « J'aime » cliquer sur un tweet puis cliquer sur l'image des deux flèches.	Si les flèches sont vides, le tweet sera retweeté, si les flèches sont vides, le retweet sera enlevé.
Assert / Then	Lors du clic sur les flèches, un message avertit l'utilisateur de l'action. La liste de tweet est réactualisée, si l'on clique sur le tweet précédemment retweeté / plus retweeté, l'image des flèches a changé.	

Résultat : OK

Remarque :

Lors du cas où l'on retweet un tweet avec une image, revient sur la page des 20 derniers tweets, enlève le retweet et revient sur la page des 20 derniers tweets, la mise en forme des tweets tout en bas possède un espace vide et le dernier tweet apparait à nouveau.

Scénario 9 : Aimer / Ne plus aimer

Étape	**Description**	**Remarque**
Arrange / Given	Exécutable de l'application	
Act / When	Dans la partie « Timeline » ou « J'aime » cliquer sur un tweet puis cliquer sur l'image du cœur.	Si le cœur est vide, le tweet sera aimé, si le cœur est plein, le tweet ne sera plus aimé.
Assert / Then	Lors du clic sur le cœur, un message avertit l'utilisateur de l'action. La liste de tweet est réactualisée, si l'on clique sur le tweet précédemment aimé / plus aimé, l'image du cœur a changé. Si l'on aime plus un tweet se trouvant dans « J'aime » il n'y apparait plus. Si l'on aime un tweet, il se retrouve aussi dans « J'aime »	Un tweet se trouvant dans « J'aime » ne peut être que plus aimé.

Résultat : KO

Remarque :

La fonctionnalité ne marche pas. L'application retourne un message d'erreur spécifiant une erreur 401 Non autorisé. L'application continue tout de même de fonctionner.

6 CONCLUSION

6.1 Bilan des fonctionnalités demandées

A la fin de ce projet, presque toutes les fonctionnalités du cahier des charges sont atteintes. Les fonctionnalités aimer / ne plus aimer un tweet sont écrites dans le code en suivant le même type de procédure que pour les autres fonctionnalités, mais pour une cause non définie, lorsque les requêtes http de ces fonctionnalités sont envoyées, le serveur retourne une erreur 401 Non autorisé. Cette erreur est néanmoins gérée afin de ne pas provoquer l'arrêt de l'application. L'erreur doit certainement se trouver au niveau de la génération du header et de la signature, mais après plusieurs tests, elle reste pour l'instant introuvable.

Il existe également un problème de gestion des images lors d'un cas très précis (voir la remarque du scénario 8 au point 5.1).

Une des possibles améliorations du projet seraient de corriger l'erreur engendrée par l'action d'aimer / ne plus aimer un tweet, ainsi que la correction du problème d'image dans le cas mentionné ci-dessus. Pour aller plus loin, il serait intéressant d'ajouter une recherche de tweets mais aussi de personne. De pouvoir s'abonner et se désabonner des personnes visibles lors de la recherche. De pouvoir également visualiser les vidéos depuis l'application, mais aussi de pouvoir tweeter des photos et vidéos depuis l'application. Une autre fonctionnalité intéressante serait de pouvoir se connecter à d'autre compte grâce à l'application, cela demanderait une autre méthode de connexion avec OAuth et une gestion de la sécurité rigoureuse.

6.2 Bilan de la planification

La tâche correspondant aux tests a été largement surestimée. Réflcéchir à l'aspect graphique de l'application m'a également pris moins de temps que prévu. Au contraire, le rapport a pris plus de temps, ainsi que la recherche de documentation. Le temps de code de l'application correspond assez bien à ce qui était prévu, mais sa planification a plusieurs fois changer en cours de projet.

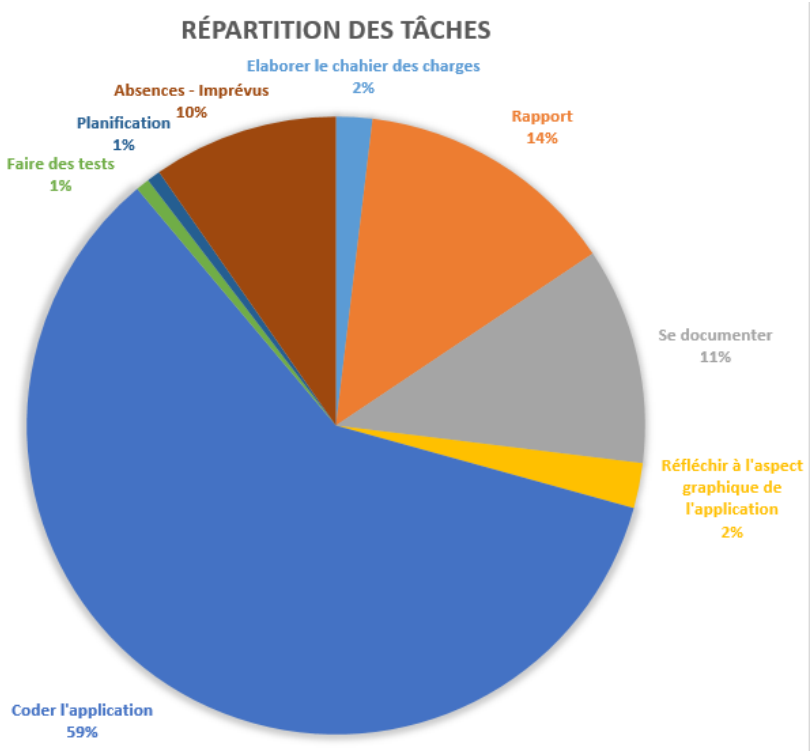


Figure 30 : Répartition des tâches réalisées

6.3 Bilan personnel

Avec ce projet, j'ai découvert ce qu'était qu'une API et l'architecture REST. C'est aussi la première fois que j'ai découvert les requêtes http ainsi que le langage JSON. C'était un projet très instructif et intéressant malgré toutes les fois où je me suis heurtée aux erreurs 401. C'était également une bonne préparation générale à tous ce qu'il faudra faire au TPI. Je me suis rendue compte que réaliser une planification d'un projet sur quatre semaine n'était pas facile, surtout avec les nouvelles choses qu'il fallait apprendre.

S'il fallait refaire ce projet, je réfléchirais beaucoup plus sur toutes les tâches à réaliser et à planifier. Je prendrais aussi plus en note les changements de la planification, ainsi que les difficultés rencontrées. Je reprendrais les mêmes documentations sur le web car elles m'ont beaucoup aidée à comprendre ce que je ne connaissais pas. Le fait de faire une maquette de l'application sur un site avant de la réaliser dans Windows Forms permet de bien se rendre compte de ce que je voulais réaliser et de ne pas perdre de temps avec les différents contrôles.

Je remercie Monsieur Melly pour m'avoir fait découvrir des notions que je ne connaissais pas et pour avoir été disponible afin de répondre à mes questions.

7 DIVERS

7.1 Journal de travail

Voir le fichier Excel *T-P_Appro2-savaryal-PlanificationEtJT.xlsm*

7.2 Webographie

- Vidyasagar Machupalli, API : Talk to Twitter API via C# - Part 1
  https://www.codeproject.com/articles/729577/api-talk-to-twitter-api-via-csharp-part
- Twitter Inc., Twitter Developer Documentation
  https://dev.twitter.com/docs
- Steven Moseley, Making a Twitter OAuth API Call Using C#
  http://www.i-avington.com/Posts/Post/making-a-twitter-oauth-api-call-using-c
- Walter G., How To Post A Tweet Using C# For Single User
  http://www.thatsoftwaredude.com/content/6289/how-to-post-a-tweet-using-c-for-single-user
- Emily Reese, Utilisez des API REST dans vos projets web
  https://openclassrooms.com/courses/utilisez-des-api-rest-dans-vos-projets-web
- REST API Tutorial, Learn REST : A RESTful Tutorial
  http://www.restapitutorial.com/
- Microsoft, Comment : envoyer des données à l'aide de la classe WebRequest
  https://msdn.microsoft.com/fr-fr/library/debx8sh9(v=vs.110).aspx
- Stack Overflow, AppSettings get value from .config file
  http://stackoverflow.com/questions/10766654/appsettings-get-value-from-config-file
- Umer Pasha, C# dot net code to get latest tweets using twitter
  https://umerpasha.wordpress.com/2013/06/13/c-code-to-get-latest-tweets-using-twitter-api-1-1/

- Newtonsoft, Json.NET Documentation
  http://www.newtonsoft.com/json/help/html/Introduction.htm

- Microsoft, Comment : écrire dans un fichier texte (Guide de programmation C#)
  https://msdn.microsoft.com/fr-fr/library/8bh11f1k.aspx

- Microsoft, try-catch (C# Reference)
  https://msdn.microsoft.com/en-us/library/0yd65esw.aspx

- Microsoft, PictureBox.Load Method (String)
  https://msdn.microsoft.com/en-us/library/f6ak7was.aspx

- OAuth Core Workgroup, OAuth Core 1.0
  https://oauth.net/core/1.0/