Projektrapport - Grupp 11 - Twitetr

Beskrivning av tjänst

Vår tjänst är en webbapplikation där en användare kan skicka tweets till Twitter och få texten stavningskontrolleras innan sändning. Om rättningsförslag hittas så lyses orden upp som röda och användaren får själv välja om hen vill rätta ordet. Därefter är tweetet redo att publiceras.

Teknisk lösning

För hemsidans struktur, styling och upplägg så har HTML och CSS använts. Utöver dessa så används JavaScript för en del beteenden såsom att ändra färg på bokstäver som är felstavade, samt göra dessa ord klick-bara för att kunna rätta stavningen för de ord som trycks på.

API:ets backend är utvecklat i Java, tillsammans med Java-baserade mikroramverket Sparkjava, som används för att bl.a. mappa ändpunkter.

Apache Velocity Engine används för att på ett enkelt sätt kunna skicka information mellan server och HTML-sida.

com.googlecode.json-simple samt com.google.code.gson används för att hantera och leverera JSON-objekt.

okhttp används för att skicka HTTP-requests.

Datakällor och API:er som har använts
Vi har använt två API:er, Twitter och Libris.
Libris API har använts för att rättstava användares meningar. Meningen att rättas skickas till Libris som returnerar en XML-fil med rättstavad mening. Denna data går vi igenom och lägger till i ett eget JSON-objekt, där varje rättat ord beskrivs i mer detalj än vad Libris API erbjuder. Vi lägger nämligen till vilket index i meningen som det rättade ordet har och vad det ursprungliga samt föreslagna ordet är.

Twitters API används för att ge användaren möjligheten att logga in och tweeta genom vår hemsida. Inloggningen och tweetandet sker med hjälp av tekniken OAuth 1.0.

Användarmanual

Installationsinstruktioner

Installationsinstruktioner avses för programmet Eclipse (https://www.eclipse.org/downloads/), men liknande utvecklingsverktyg gå säkert lika bra.

1. Starta Eclipse IDE

2. Gå till “File” -> “Import” -> “Existing Maven Project” (skriv i sökrutan om det skulle behövas)
3. Tryck på “Browse” i rutan som kommer upp. Hitta och välj mapp “twitetr”.
4. Tryck på Finish och vänta tills Eclipse byggt upp projektet. Färdigt!

Körinstruktioner

För att köra projektet, se till att ovan installationsinstruktioner först utförts.

1. Hitta projektet “twitetr” i Eclipse.
2. Gå in i “Java Resources” -> “src/main/java” -> “twitetr”
3. Högerklicka Java-filen WebAppRunner, och gå ner till “Run As” -> “Java Application”
4. När klassen körs så startas API:t och webapplikationen. Men API:t kan köras separat genom att gå in i paketet “api” -> kör “APIRunner”.
5. Servern är nu startad och hemsidan kan kommas åt i valfri browser genom “localhost:4567/”. API-anrop görs också mot localhost:4567/

API-design

Vårt API är väldigt simpelt och har ett viktigt syfte: att kunna skicka stavningskontrollerade tweets genom vårt API. För att detta ska kunna ske så har vi använt oss av Libris API för stavningskontrollen, och Twitters API för att publicera användares tweet på Twitter.

Integration med Libris är väldigt simpel. En nyckel erhålls manuellt från Libris hemsida som låter oss använda deras API. Ett anrop kan sedan göras mot “http://api.libris.kb.se/bibspell/spell?query={text}&key=key” där nyckeln läggs till efter “key=”, och meningen som önskas rättas sätts in istället för “text”. Returdatan från Libris är i form av XML. Vi bearbetar denna data med Java, och lägger till informationen i ett JSON-objekt. Strukturen på denna finns beskrivet i API-dokumentationen.

Tyvärr var Libris API dock ganska begränsat. En del meningar kunde den inte rätta, trots att de var relativt simpla. Exempelvis så ger queryn “Ska vi gå till teattern imorgon” inga rättningsförslag från Libris.

Integrationen av Twitter’s API är lite mer komplicerad, speciellt för att detta kräver autentisering med OAuth. Detta innebär att de flesta anrop mot Twitter kräver att man följer både Twitters och OAuth:s väldigt strikta instruktioner. T.ex. måste OAuth:s header-värden vara sorterade i bokstavsordning, samt innehålla ett unikt värde för varje anrop, kallat nonce-värde. Sedan måste specifika header-värden i anropet mot Twitter signeras med HMAC-SHA1 signatursmetod. Det finns många bibliotek som hjälper med denna integration, som Twitter4j, men vi valde att inte använda några sådana verktyg för att bilda en bättre förståelse av denna process - vilket var kul och lärande, men frustrerande att få rätt på.

Vårt API:s syfte som beskrevs ovan exponerar vi med två metoder: **/tweetcheck**, som returnerar ett JSON-objekt med stavningskontrollerad sträng, och **/tweet**, som skickar upp en sträng till Twitter. Nedan beskrivs metoderna mer i detalj.

/tweet är den mest väsentliga metoden för vårt API. Den tar emot ett JSON-objekt med text som önskas tweetas, samt två Oauth värden: access_token och secret_key. Resterande oauth värden, d.v.s. oauth_nonce, oauth_signature, oauth_signature_method, oauth_timestamp, genererar vårt API. Detta för att göra det så enkelt som möjligt för framtida utvecklare att använda vårt API.

Metoden tar emot värdena i anropets header då Oauth-värden är känsliga och bör gömmas från URL:n. Dessutom så erbjuder det större struktur på indatan.

Vår andra metod däremot, **/tweetcheck**, bör användas innan /tweet anropas, för att få ens mening stavningskontrollerad. Metoden tar endast emot ett värde: en sträng. Eftersom det endast är ett värde, och detta inte är känslig data, valde vi att hålla denna metod simpel genom att ta emot datan i URL-parameter, t.ex. genom /tweetcheck?userTweet=abc.

API:ts båda metoder använder POST som metodtyp. Detta är särskilt viktigt för /tweet som tar emot känslig data, i vårt fall oauth-uppgifter.

Vi valde att leverera vår API:ts data i JSON-format då detta erbjuder ett strukturerat och enkelt sätt att returnera data på.

Som tidigare nämnt, kräver metod-anropet /tweet Oauth-uppgifter. Tanken var från början att vårt API skulle erbjuda en metod som eventuella framtida utvecklare hade kunnat skicka sina egna användare till för att få deras oauth_consumer_key samt oauth_token från Twitter. Problemet med detta är att för att en inloggning med Twitter ska ske, vill Twitter enligt deras dokumentation att man ska omdirigera användaren till https://api.twitter.com/oauth/authenticate, där användaren får logga in på sitt konto. Sedan skickas användaren tillbaka till en callback_url med oauth-uppgifterna i parametern. Denna callback_url måste först whitelistas på Twitters webbsida, och det är här vi mötte vårt problem. Det innebär att vi inte hade kunnat låta framtida utvecklare specificera sin egen callback_url där autentiseringsuppgifterna ankommer. Om ett annat API använder vårt API, och skickar deras användare till vår "/login"-metod, hade de tappat kontrollen över användaren då användaren omdirigeras till vårt callback_url, vilket är /success. Skulle någon vilja använda vårt API, måste de då därför först erhålla användarens oauth-uppgifter på egen hand. För att testa API-metoderna genom t.ex. Postman, måste man först göra ett anrop till /login (i webbapplikation) och sedan kopiera de oauth-uppgifter som skrivs ut i konsolen.

Vi byggde vårt API med Sparkjava då detta är ett väldigt simpelt mikroramverk som erbjuder viktiga funktioner som att mappa ändpunkter, ange metodtyp, definiera parametrar samt leverera returdata.

API-dokumentation

POST /tweetcheck?usertweet={text}

Returnerar ett förslag på rättstavningar för inskickad sträng

Indata

Parameter {**tweet}** : sträng som önskas rättas

Returdata

Responstyp: JSON

Exempel:

Anrop: http://localhost:4567/tweetcheck?usertweet=hejj+detta+är+ett+exempell

Respons:


```
{
    "query": "hejj detta är ett exempell",
    "found_suggestion": true,
    "suggested_sentence": "hej detta är ett exempel ",
    "flagged_words": [
        {
            "index": 0,
            "word": "hejj",
            "suggestion": "hej"
        },
        {
            "index": 4,
            "word": "exempell",
            "suggestion": "exempel"
        }
    ]
}
```


Returdata vid inga hittade förslag:

Anrop: http://localhost:4567/tweetcheck?usertweet=hejj+detta+är+ett+exempell

Respons:


```
{
    "query": "hej detta är ett exempel",
    "found_suggestion": false,
    "suggested_sentence": "",
    "flagged_words": []
}
```


POST /tweet

Skickar upp användares tweet på Twitter

Indata

```
{
    "tweet": "Detta är ett Tweet!",
    "oauth": {
        "access_token": "1211322354434891776-PmrWMzcEwBiJ7J9txSkCVHEJWY503a",
        "secret_key": "fcaWHdelMBenrBg5aKBhoOYKTwxUKG6A39OFLks5fjn0T"
    }
}
```

Returdata vid lyckat anrop

Reponstyp: JSON

```
{
    "tweet": "Detta är ett Tweet!",
    "screen_name": "LukasRosb",
    "user_id": "1211322354434891776",
    "tweet_id": "1252034641323798530",
    "created at": "Mon Apr 20 00:41:34 +0000 2020",
    "status": "OK"
}
```

/tweet felmeddelanden

Saknade parametrar. Vanligt vid parameter med null-värden:

```
{
    "code": 1,
    "error_message": "Missing parameters"
}
```

Felmeddelande för felaktiga header-värden:

```
{
    "code": 2,
    "error_message": "Invalid parameter data"
}
```

Felmeddelande vid misslyckad autentisering mot Twitter

```
{
    "code": 32,
    "error_message": "Could not authenticate you"
}
```

Felmeddelande för felaktiga oauth värden:

```
{
    "code": 215,
    "error_message": "Bad authentication data"
}
```

Felmeddelande utgått oauth värden:

```
{
    "code": 89,
    "error_message": "Invalid or expired token"
}
```

Felmeddelande vid duplicerat tweet:

```
{
    "code": 187,
    "error_message": "Status is a duplicate"
}
```

Felmeddelande vid för långt tweet:

```
{
    "code": 186,
    "error_message": "Tweet needs to be a bit shorter"
}
```

Felmeddelande vid bad request:

```
{
    "code": 200,
    "error_message": "Bad request"
}
```

Kodexempel

Vid API-anrop till /tweet.json tas ett JSON-objekt emot med följande parametrar: oauth_secret, oauth_access_token och tweet. Detta görs med nedan kod:

```
String tweetJSON = request.body();
Object object = new JSONParser().parse(tweetJSON);
JSONObject jsonobject = (JSONObject) object;
JSONObject jsonobject2 = (JSONObject) jsonobject.get("oauth");
String access_token = (String) jsonobject2.get("access_token");
String secret_key = (String) jsonobject2.get("secret_key");
```

// Värdena skickas vidare till metoden tweet i klassen TweetHandler

```
String responseString = tweeter.Tweet(secret_key, access_token, tweet);
```

//POST-anrop till Twitter skapas och tweet läggs till i variabeln “status”

```
HttpPost post = new HttpPost("https://api.twitter.com/1.1/statuses/update.json?status="+status);
```

//Vi lägger till våra headers, parametrar för Oauth genereras i annan modul. Sedan skickar vi requestet mot Twitter.

```
post.addHeader("Content-Type", "application/x-www-form-urlencoded");
post.addHeader("Host", "api.twitter.com");
post.addHeader("Accept", "*/*");
post.addHeader("Connection", "keep-alive");
post.addHeader("Authorization",
                "OAuth oauth_consumer_key=\"" + consumerKey + "\", "
                + "oauth_nonce=\"" + nonce + "\", "
                + "oauth_signature=\"" + signature + "\", "
                + "oauth_signature_method=\"HMAC-SHA1\", "
                + "oauth_timestamp=\"" + time + "\", "
                + "oauth_token=\"" + token + "\", "
                + "oauth_version=\"1.0\"");
HttpResponse response = httpClient.execute(post);
HttpEntity entity = response.getEntity();
String responseString = EntityUtils.toString(entity, "UTF-8");
```

//Svaret lagras i responseString. Om tweetet lyckas innehåller första meningen “created_at”. Vi utnyttjar detta för att kontrollera om tweetet skickades eller inte. Hade också kunnat lösa detta med samma typ av JSON-hantering som sker nedan.

//Kod nedan körs om tweetet lyckas. Ansvarar för att lägga till respektive data i ett JSON-objekt.

```
responseCode = 200;
Object obj = new JSONParser().parse(responseString);
JSONObject jsonobj = (JSONObject) obj;
String created_at = (String) jsonobj.get("created_at");
String id = (String) jsonobj.get("id_str");
JSONObject jsonobj2 = (JSONObject) jsonobj.get("user");
String username = (String) jsonobj2.get("screen_name");
String userID = (String) jsonobj2.get("id_str");
return new TweetJSON(tweet, username, userID, created_at);
```