Massmine for the Masses

Project Team: Morgan McIntyre, Logan Hornbuckle, Patricia Tanzer, Joshua Moore

Table of Contents

[**1. Project Definition**](#_f6pns9jm1fk3) **4**

[1.1 Why](#_wckxeknchnp2) 4

[1.2 What](#_8pqoao6zguhn) 4

[1.3 How](#_ac67agwzegxq) 4

[**2. Project Requirements**](#_s8tvbcpndfdn) **4**

[2.1 Functional](#_nut9bfa4x5r4) 4

[2.2 Usability](#_p6102w55pc12) 4

[2.2.1 User interface](#_cm0uglhdzq4d) 4

[2.2.2 Performance](#_obcm4tqhdre9) 5

[2.3 System](#_oc4t3wniflsg) 5

[2.3.1 Hardware](#_683g7pjlevav) 5

[2.3.2 Software](#_nc8gzajo58wn) 5

[2.3.3 Database](#_pi0zjke0ru85) 5

[2.4 Security](#_xzvt94xc9fhb) 5

[**3. Project Specification**](#_eksquogfbmju) **6**

[3.1 Focus](#_rg433nvb9yv) 6

[3.2 Domain](#_hmz5zpc4eyke) 6

[3.3 Area](#_y5trxio2ijz9) 6

[3.4 Libraries](#_jked61g3emcr) 6

[3.5 Development Environment](#_13rpc1v4zvqk) 6

[3.6 Framework](#_8uisdeskcnpl) 6

[3.7 Platform](#_mzvwrs9cvj6c) 6

[3.8 Genre](#_5y6p0zahmzml) 7

[**4. System – Design Perspective**](#_a5j4ixnmnpey) **7**

[4.1 Identify subsystems – design point of view](#_3umvy5vsdveo) 7

[4.2 Sub-System Communication](#_23df5mtsj783) 12

[4.3 Entity Relationship Model](#_qvg43d2qlux8) 15

[4.4 Overall operation - System Model](#_3y7zpqrcr1im)  17

[**5. System – Analysis Perspective**](#_lnwvi8acowhv) **17**

[5.1 Identify subsystems – analysis point of view](#_bxo700v8xz0t) 18

[5.2 System (Tables and Description)](#_bj18u0w7fth6) 19

[5.2.1 Data analysis](#_it8ed9kqjnr)  19

[5.2.2 Process models](#_at9axl9vgcdb)  31

[5.3 Algorithm Analysis](#_933hvher64ls) 31

[5.3.1 Big - O analysis of overall System and Sub-Systems](#_wt06g8nan9n8) 31

[**6. Project Scrum Report**](#_vlc1r9u9hvb9) **32**

[6.1 Product Backlog (Table / Diagram)](#_mcoqhh8a85vm) 32

[6.2 Sprint Backlog (Table / Diagram)](#_jj6jdkr9qw9u) 33

[6.3 Burndown Chart](#_vx7yl22u7qwn) 34

[**7. Subsystems**](#_dbbrecckf7l2) **36**

[7.1 Subsystem 1 – User Management -](#_lwdx0mmrvoty) Patti 36

[7.2 Subsystem 2 – Analysis Management -](#_qwg9ggqud3v0) Morgan 36

[7.3 Subsystem 3 – Query Management -](#_hqjuk6py8zx) Logan 37

[7.4 Subsystem 4 – Massmine Integration -](#_3ixyol1ashjz) Josh 37

[**8. Complete System**](#_hrqicvy4qa2o) **38**

# **1. Project Definition**

## 1.1 Why

Massmine ([www.massmine.org](https://www.massmine.org/)) is a command-line tool useful in the data analysis field in that it can pull massive amounts of data from sources like Twitter and give them to the user for further analysis. However, because it is a command-line tool in the Linux operating system, not every user understands how to work with it. Additionally, it produces that data in a form which is difficult to parse for the non-technically oriented. This makes massmine very difficult to work with, despite its usefulness.

## **1.2 What**

The goal of this project it to simplify this process for the average user by creating a web application that allows a user to create queries for Twitter data in a simple, easily understandable user interface. The data will also be presented in a simple user interface to allow for basic analysis without the user ever having to directly interact with it.

## 1.3 How

The web application will take query information from the user such as search terms and time periods, and send this information to a massmine instance running on the host server. The data retrieved from this massmine instance will be parsed and stored in a database which can then be accessed at a later date through the web application.

# 2. Project Requirements

## 2.1 Functional

2.1.1 Dockerized web application and associated database to:

* + 1. Store user information (username, password, Twitter authorization key, short bio)
    2. Store collected Twitter data (tweet id, author, content, etc.)
    3. Create and update new users (change oauth key, bio)
    4. Allow users to gather Twitter data. The mass of data collected by a query is a ‘study’.
    5. Allow users to perform basic analysis on this Twitter data.

## 2.2 Usability

### 2.2.1 User interface

* + 1. Login interface: should allow the creation of accounts, as well as signing in.
    2. User account interface: Allow user to enter or change their Twitter authentication code, as well as bio.
    3. Study interface: Lists the studies that a user has access to and allows them to open Analysis interfaces on those studies. Another link allows them to create a new study or remove an old one. Optional but useful: Allow renaming of studies and inviting/removing additional users from viewing a study.
    4. Collection interface: Allow the user to create queries or ‘studies’.
    5. Analysis interface: Allow the user to run basic analyses, such as aggregate by hour, on a study. (Optional: allow user to export analysis for presentation, etc).

### 2.2.2 Performance

* + 1. The user should not notice any significant time delay when running analyses on data already gathered. Depending on the query, data likely will not be available immediately after a query is given, but this should not impact the user from signing out, back in, or running additional queries and analyses in the meanwhile. A collection database will keep past queries to maintain efficient analysis.
    2. Multiple uses should be able to use the web-app at once. Up to 50 accounts expected.

## 2.3 System

### 2.3.1 Hardware

* + 1. Any computer or server capable of running a Docker container.

### 2.3.2 Software

* + 1. Web application from Django framework, programmed in Python and HTML.
    2. Additional modules in Python to communicate with Massmine..

### 2.3.3 Database

* + 1. SQLite, provided by Django framework.

## 2.4 Security

* 1. At-rest encryption of user authentication keys. Passwords used to sign in to the account will be salted and hashed. At this time, looking at Django REST encryption for this feature.
  2. The framework used, Django, has built-in protections and security features. Django guards against cross site scripting, cross site request forgery, SQL injection, and clickjacking.
  3. The user queries will be scrubbed for invalid input

# 3. Project Specification

## 3.1 Focus

* + Develop an open source, web-based application to collect and analyze social network data using Massmine.

## 3.2 Domain

* + The intended user for this application would be academic researchers and universities that may host their own version of this project. This application should appeal to those interested in social media research that lack the necessary command-line skills to fully utilize massmine.

## 3.3 Area

* + Big data/data analysis - This project will involve large sets of metadata that must be parsed through and organized efficiently.
  + Database management - There will be several databases used and maintained within the scope of this project.
  + Web application/web application security

## 3.4 Libraries

* + The following are examples of Python libraries that will be used: shell\_plus, clean\_pyc, json encoding and decoding, create\_template tags, describe\_form, notes

## 3.5 Development Environment

* + Pip virtualenv
  + Django 2.1.7
  + Ubuntu 18.04 host(virtual machine)

## 3.6 Framework

* + Django 2.1.7, a high-level Python web framework. This framework is scalable and secure.
  + SQLite is our framework for the databases. It is the default database for Django and fills our needs.

## 3.7 Platform

* + Web application meant for desktop access. This platform allows for user usage on any operating system, although the server should be running a Linux distribution to run the Massmine tool.

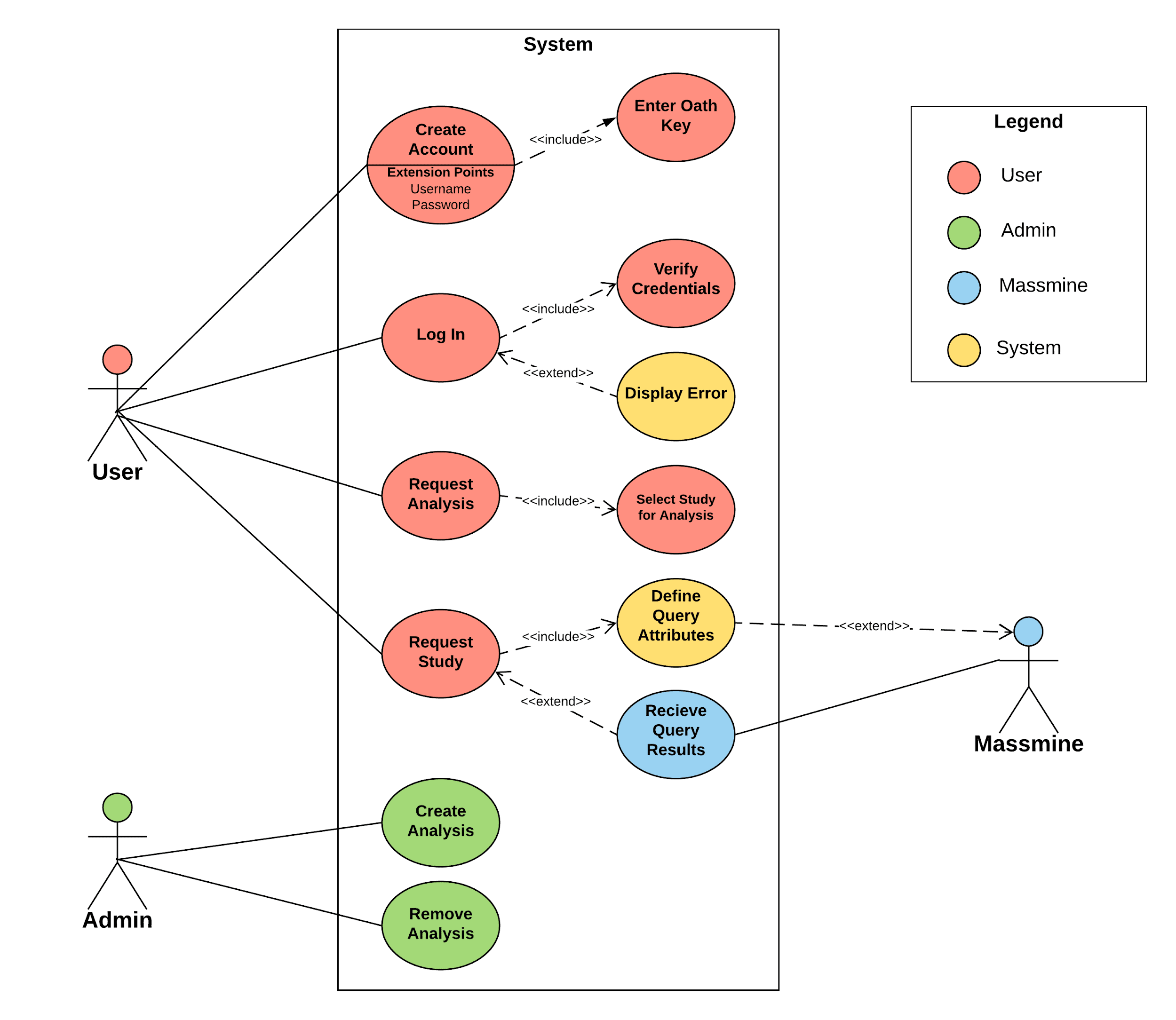
## 3.8 Genre

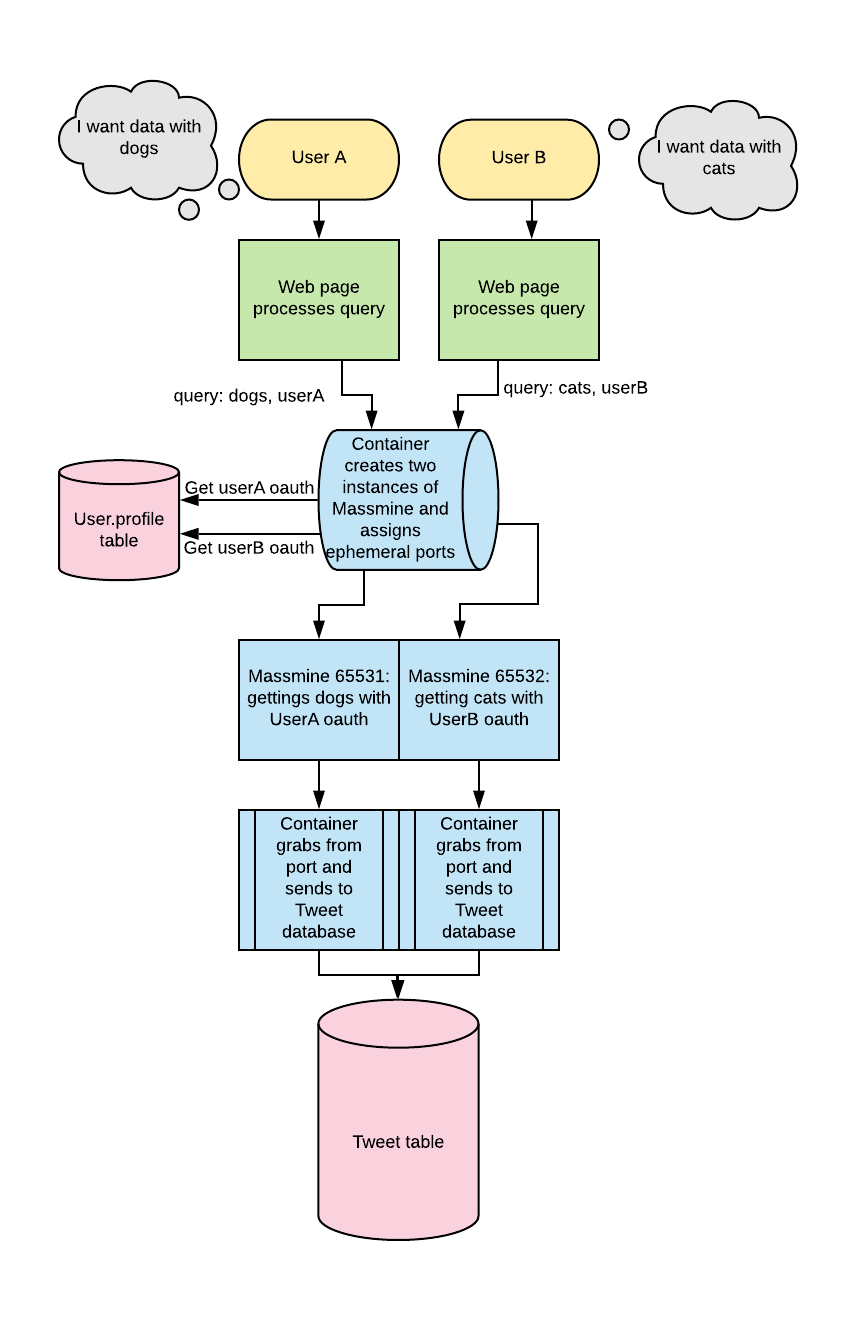
* + Web application layered over the command-line tool Massmine.

# **4. System – Design Perspective**

## 4.1 Identify subsystems – design point of view

* + Subsystem 1: User Management - Patti
    - Home page
      * Gives options to register, log in, or go to the admin page (requires authentication)
    - User Profile:
      * Username
      * Email
      * Password
      * Oauth
      * Bio (not required)
      * First name (not required)
      * Last name (not required)
    - Registration:
      * Takes some user profile information at time of registration. Ensures password meets security standards
    - Edit Profile:
      * Allow user to change everything in their profile except their username.
    - Login/Logout
      * The user should be able to log out from any page
      * Login should only be available from the home page
    - Administration
      * Can edit, delete, create users
      * Can see all user information
  + Subsystem 2: Analysis Management: - Morgan
    - Analysis types:
      * tweet\_type - This sorts a study by tweet type (tweet, reply, retweet)
      * freq\_words - Sorts by the most frequently used words in a study
      * freq\_hashtags - Sorts by the most frequently used hashtags in a study
      * act\_authors - Sorts by the most active users (by how many posts they have)
      * pop\_authors - Sorts by authors with the most followers
      * men\_accounts - Sorts by most mentioned accounts
      * lang - Lets user filter/sort languages
      * device - Sorts by source device
      * date - Sorts by date
      * times\_retweeted - Sorts by how many times each tweet was retweeted (most popular tweets)
      * times\_favorited - Sorts by how many times the tweet is favorited
      * location - Sorts by country, all tweets with no geo information will be excluded
    - Stored as a model in Django
    - User may choose one of these analyses on the Analysis page and feed it the study ID of the study to be sorted, so that the results are graphically displayed
  + Subsystem: Query Management - Logan
  + Subsystem: Massmine Integration - Josh
* System Use-Case Diagram: - Logan



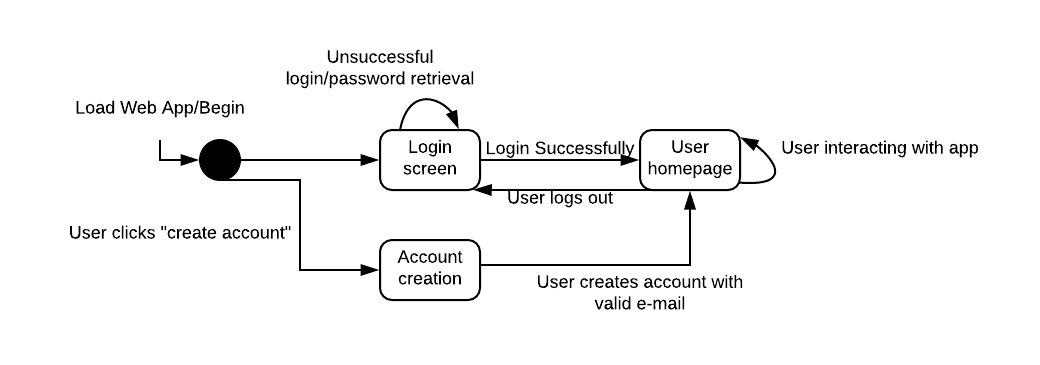
* Massmine integration: - Patti
  + - 
* Web interface - Home page mockup

## 4.2 Sub-System Communication

* Controls by subsystem - Morgan + Pati

## Subsystem 1: User Management and Templates

* + - Django has built-in controls to prevent cross site scripting and cross site request forgery. - Morgan
    - The user may navigate from page to page with buttons on the top of the screen. - Morgan
    - Data entry, including username, bio, and oauth key, will be checked for sql injection. Password is already scrubbed for invalid characters. - Patti
    - Account creation will require a unique username and a password that fits length and complexity standards. - Morgan
    - Login will require the correct username and password combination and verified in database. - Patti
    - User passwords will be hashed and salted in the user database. - Patti
    - User credentials must be verified to access the edit profile page. - Patti
    - Login diagram - Morgan



## Subsystem 2: Analysis Management - Morgan

* + - User credentials must be verified to access the analysis page.
    - This subsystem will communicate with the Massmine integration section.
    - The user may choose the type of analysis to be displayed from a drop-down menu.
    - This subsystem is the only one to call the analysis model.
    - Only uses tweets from user’s own studies.
    - References the study database based on user input (study id), but will only accepts integers to avoid injection.

## Subsystem 3: Query - Patti

* + - User credentials must be verified to access this page.
    - User can select various qualifications on their search such as date and a word or phrase to search for.
    - Input words or phrases should be scrubbed for command injection.
  + Subsystem 4: Massmine Integration - Morgan
    - Users need an Oauth keys to be able to gather twitter data. In the Massmine communication if they don’t have a valid Oauth key they can’t request a study.
    - Massmine itself is a command line tool, and will be running concurrently on the local machine.
* I/O by subsystem - Morgan + Patti
  + Inputs
    - Subsystem 1: User Management - Patti
      * Username from user
      * Password from user
      * Oauth key from user
      * Bio from user (not required)
      * First name from user (not required)
      * Last name from user (not required)
      * Randomly generated study ids
      * Twitter data from server

## Subsystem 2: Analysis - Morgan

* + - * Analysis requirements from user (choose from a drop-down menu)
      * Study ID to distinguish which study will be used

## Subsystem 3: Query - Morgan

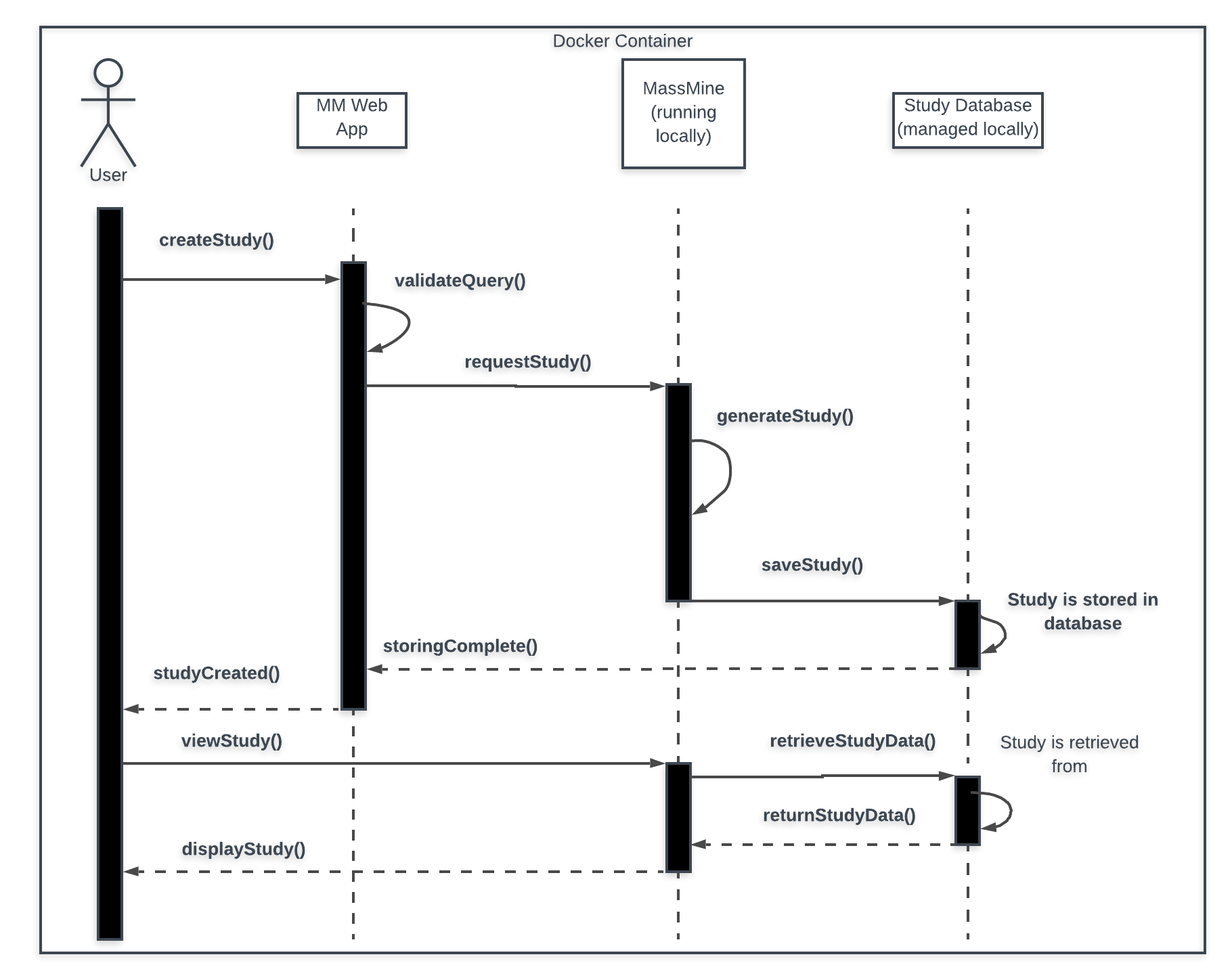
* + - * Study requirements from user (date, search terms, etc.)
    - Subsystem 4: Massmine communication - Morgan
      * User information (oauth key)
      * User query
  + Outputs - Morgan
    - Subsystem 1: User Management
      * Valid/Invalid response for authentication checking

## Subsystem 2: Analysis - Morgan

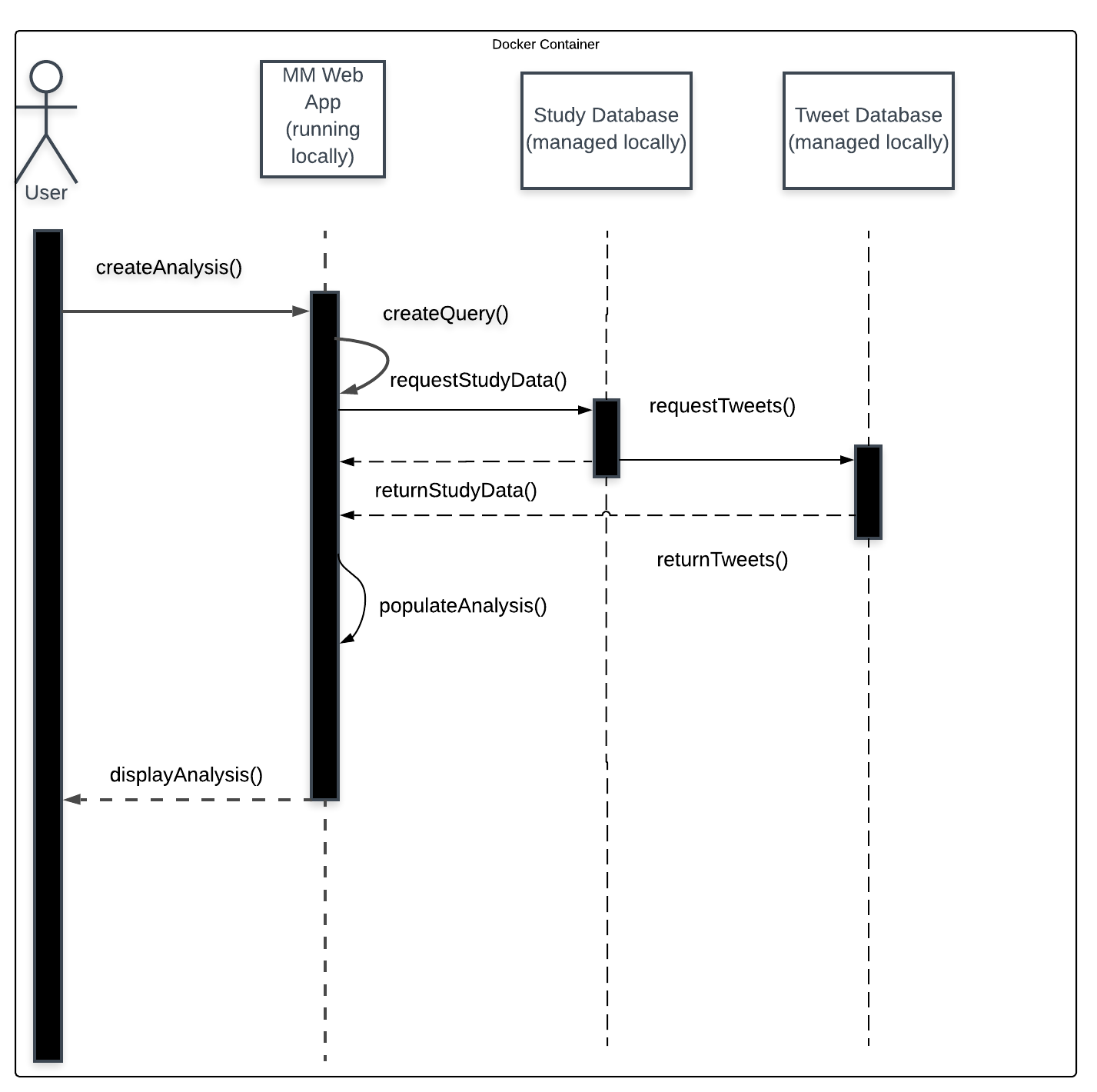
* + - * Gives options for analysis (tweets to select, conditions to set)
      * Analysis results to user as graphical display
      * Gives error message if the user is not logged in on analysis page.

## Subsystem 3: Query -

* + - * Gives options for query (date, time, phrases)
      * Sends command line query to massmine
    - Subsystem 4: Massmine communication
      * Results from Massmine to tweet storage
      * Displays user’s studies
* Dataflow - Morgan
  + Data-flow for creating a study

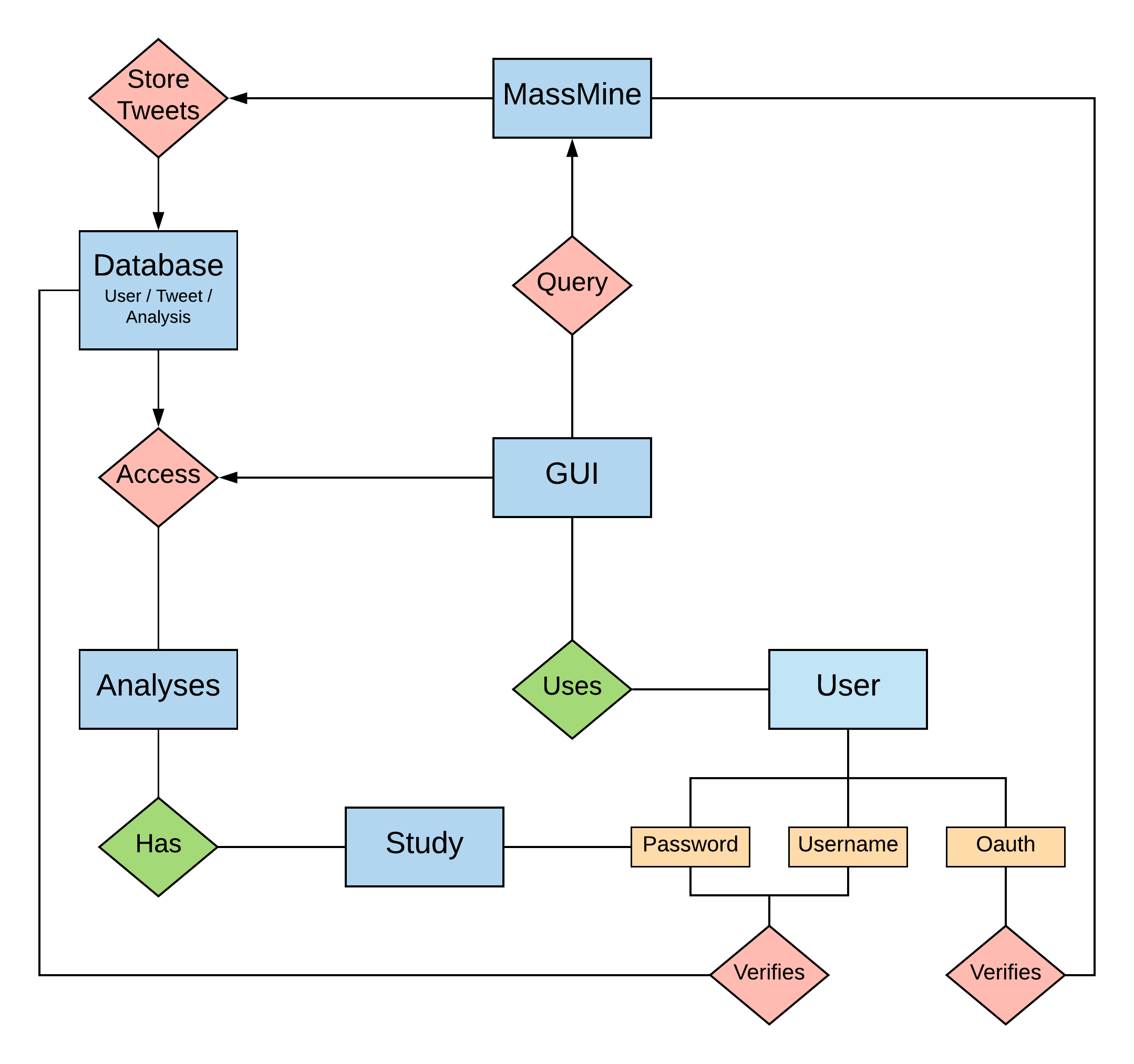


* Data-flow for creating an analysis - Morgan

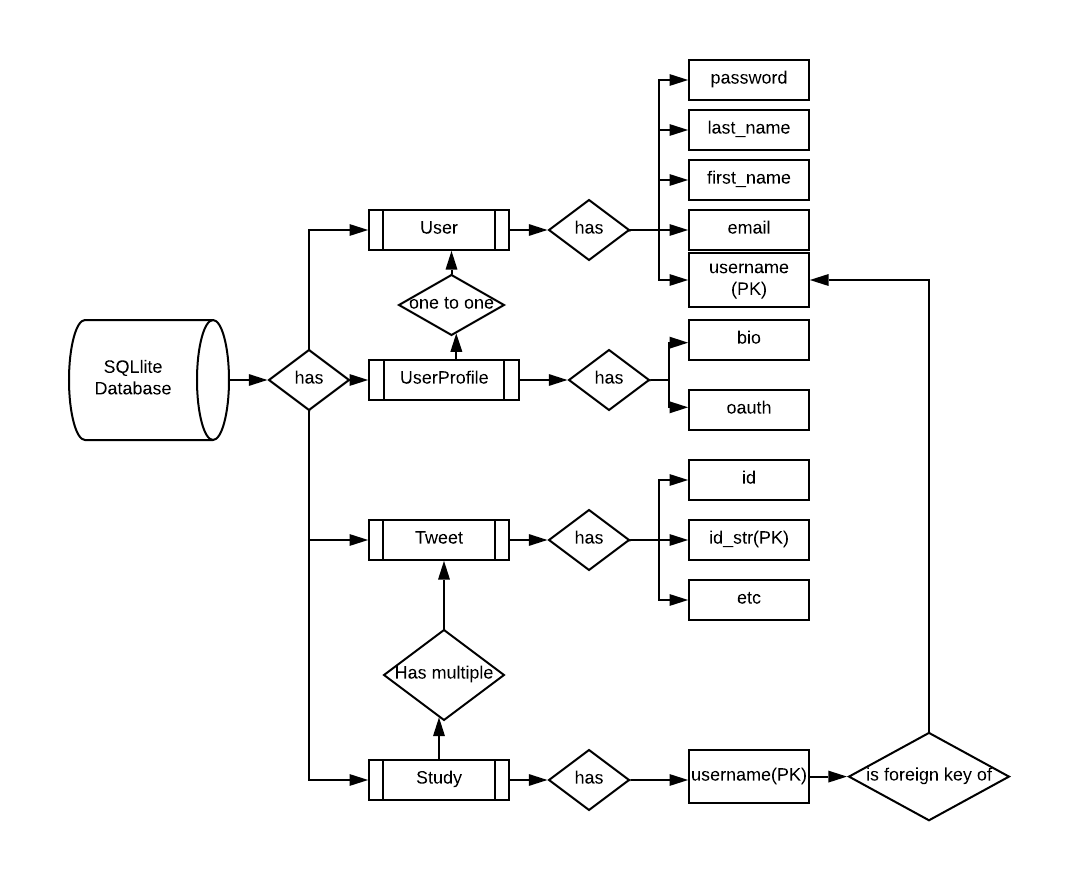


## 4.3 Entity Relationship Model

* System Wide ERM: - Logan

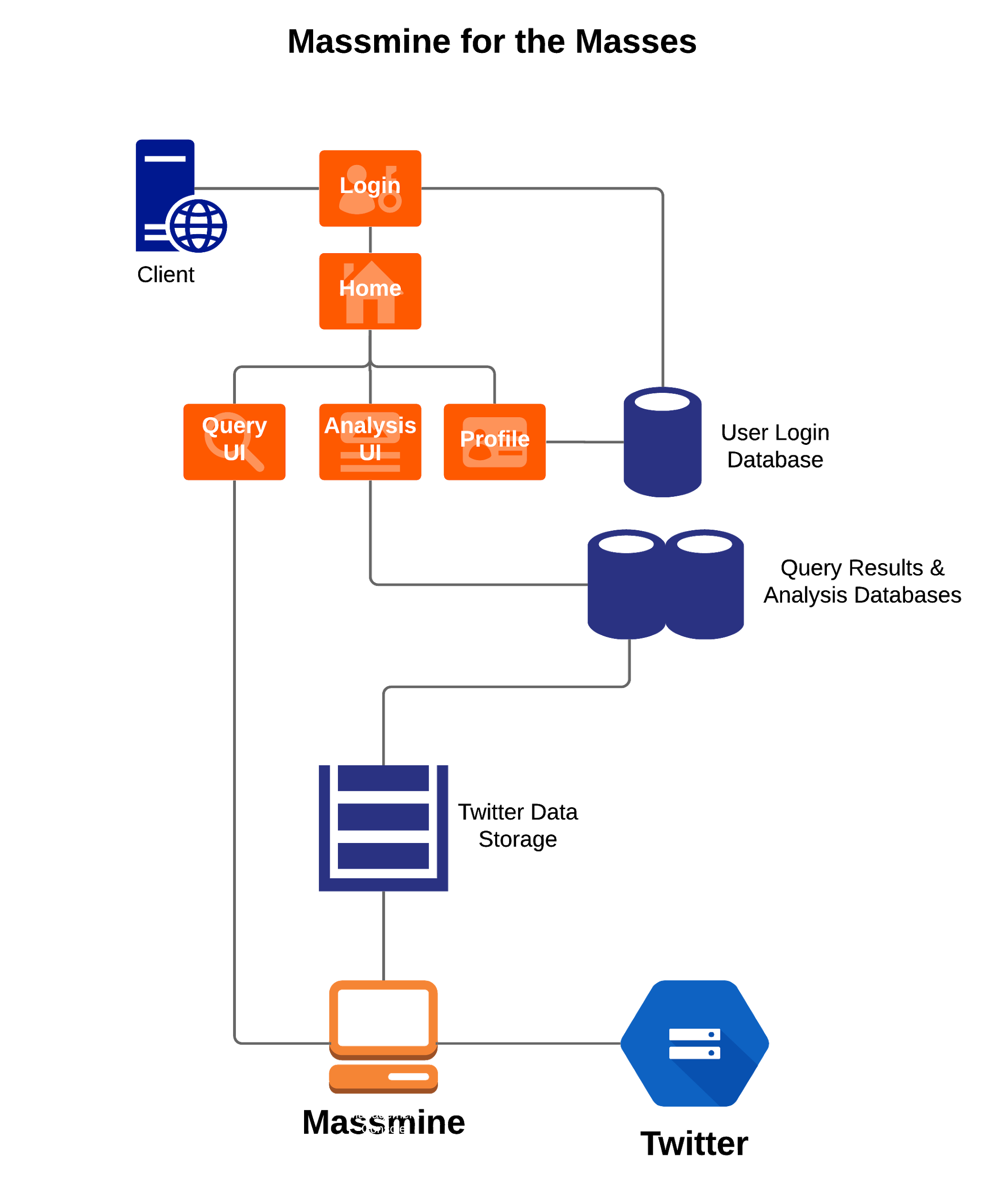


* Database ERM: - Patti



## 

## 4.4 Overall operation - System Model - Logan



# **5. System – Analysis Perspective**

## 5.1 Identify subsystems – analysis point of view

## Subsystem 1: User Management - Patti

* + Web GUI
    - Registration
    - Login
    - Logout
    - Edit Profile
    - Change Password
  + Data Storage:
    - User and User Profile model stored in local database.
  + Data Structure:
    - User model is provided by django.contrib.auth.
    - User Profile has a one-to-one relationship with the User model to add extended attributes.
  + Performance:
    - Anticipate up to 10 users per Docker container.

## Subsystem 2: Analysis???

* + GUI Structure:
    - Django-defined Models consisting of appropriate Field classes, specified by class attributes.
    - The only required part of a Django Model is the list of database fields it defines.
    - Web Interface should allow any open connection to interact with the login page, while internal pages require authentication.

## Performance:

## The server should support at least 25 simultaneous user sessions.

## Subsystem 3: Query???

* + Analysis Structure:
    - Each analysis type is a filter that can be applied to a particular set of data
    - Multiple filters can be layered for more complex analyses

## Performance

* + - The analysis graphics should update as the analyses is working, so that users are not waiting long periods for data to display
* Subsystem 4: Massmine communication - Patti
  + Structure:
    - Host runs an instance of Massmine for each user query.
    - JSON file(s) returned by each query are parsed and inserted into the tweet table.
  + Performance:
    - The host should support at up to 10 instances of Massmine (one per expected user).

## 5.2 System (Tables and Description)

### 5.2.1 Data analysis - Patti

* + - Data dictionary:
    - User model: <https://docs.djangoproject.com/en/2.1/ref/contrib/auth/>

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Type** |
| username | Name identifying a user | CharField, max length 150. May contain alphanumeric, @, +, . and - characters. |
| password | Hashed and salted password to confirm user login | CharField. |
| first\_name | Optional detail if users wish to personalize their accounts. | CharField. 30 characters or fewer. |
| last\_name | Optional detail if users wish to personalize their accounts. | CharField. 150 characters or fewer. |
| email | Optional detail if users wish to personalize their accounts. In future releases could be used to provide notifications and recover password support. | CharField, formatted as email address. |
| is\_staff | Designates whether this user can access the admin site. | Boolean. |
| is\_active | Designates whether user is active. May use instead of deleting user. | Boolean. |
| is\_superuser | Designates that this user has all permissions. | Boolean. |
| last\_login | Datetime of the user’s last login | DateTime |
| date\_join | Datetime of when account was created. | DateTime |

* + - User Profile

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Type** |
| bio | Optional detail if user wishes to personalize account. | TextField, max\_length 500. |
| oauth | Twitter authorization key | CharField, max\_length 50 |

* + - Tweet:
      * + (all attributes except for study\_id are the same as those in the tweet object found at <https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html> )

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| study\_id | Identifies the user study to which the tweet belongs |
| created\_at | UTC time when this Tweet was created. Example:  "created\_at":"Wed Aug 27 13:08:45 +0000 2008" |
| id | The integer representation of the unique identifier for this Tweet. This number is greater than 53 bits and some programming languages may have difficulty/silent defects in interpreting it. Using a signed 64 bit integer for storing this identifier is safe. Use id\_str for fetching the identifier to stay on the safe side. See [Twitter IDs, JSON and Snowflake](https://dev.twitter.com/overview/api/twitter-ids-json-and-snowflake) . Example:  "id":114749583439036416 |
| id\_str | The string representation of the unique identifier for this Tweet. Implementations should use this rather than the large integer in id. Example:  "id\_str":"114749583439036416" |
| text | The actual UTF-8 text of the status update. See [twitter-text](https://github.com/twitter/twitter-text/blob/master/rb/lib/twitter-text/regex.rb) for details on what characters are currently considered valid. Example:  "text":"Tweet Button, Follow Button, and Web Intents" |
| source | Utility used to post the Tweet, as an HTML-formatted string. Tweets from the Twitter website have a source value of web.  Example:  "source":"Twitter for Mac" |
| truncated | Indicates whether the value of the text parameter was truncated, for example, as a result of a retweet exceeding the original Tweet text length limit of 140 characters. Truncated text will end in ellipsis, like this ... Since Twitter now rejects long Tweets vs truncating them, the large majority of Tweets will have this set to false . Note that while native retweets may have their toplevel text property shortened, the original text will be available under the retweeted\_status object and the truncated parameter will be set to the value of the original status (in most cases, false ). Example:  "truncated":true |
| in\_reply\_to\_status\_id | *Nullable.* If the represented Tweet is a reply, this field will contain the integer representation of the original Tweet’s ID. Example:  "in\_reply\_to\_status\_id":114749583439036416 |
| in\_reply\_to\_status\_id\_str | *Nullable.* If the represented Tweet is a reply, this field will contain the string representation of the original Tweet’s ID. Example:  "in\_reply\_to\_status\_id\_str":"114749583439036416" |
| in\_reply\_to\_user\_id | *Nullable.* If the represented Tweet is a reply, this field will contain the integer representation of the original Tweet’s author ID. This will not necessarily always be the user directly mentioned in the Tweet. Example:  "in\_reply\_to\_user\_id":819797 |
| in\_reply\_to\_user\_id\_str | *Nullable.* If the represented Tweet is a reply, this field will contain the string representation of the original Tweet’s author ID. This will not necessarily always be the user directly mentioned in the Tweet. Example:  "in\_reply\_to\_user\_id\_str":"819797" |
| in\_reply\_to\_screen\_name | *Nullable.* If the represented Tweet is a reply, this field will contain the screen name of the original Tweet’s author. Example:  "in\_reply\_to\_screen\_name":"twitterapi" |
| user | The user who posted this Tweet. See User data dictionary for complete list of attributes.  Example highlighting select attributes:  {  "user": {  "id": 2244994945,  "id\_str": "2244994945",  "name": "TwitterDev",  "screen\_name": "TwitterDev",  "location": "Internet",  "url": "https://dev.twitter.com/",  "description": "Your source for Twitter news",  "verified": true,  "followers\_count": 477684,  "friends\_count": 1524,  "listed\_count": 1184,  "favourites\_count": 2151,  "statuses\_count": 3121,  "created\_at": "Sat Dec 14 04:35:55 +0000 2013",  "utc\_offset": null,  "time\_zone": null,  "geo\_enabled": true,  "lang": "en",  "profile\_image\_url\_https": "https://pbs.twimg.com/"  } } |
| coordinates | *Nullable.* Represents the geographic location of this Tweet as reported by the user or client application. The inner coordinates array is formatted as [geoJSON](http://www.geojson.org/) (longitude first, then latitude). Example:  "coordinates": {  "coordinates":  [  -75.14310264,  40.05701649  ],  "type":"Point" } |
| place | *Nullable* When present, indicates that the tweet is associated (but not necessarily originating from) a [Place](https://developer.twitter.com/overview/api/places) . Example:  "place": {  "attributes":{},  "bounding\_box":  {  "coordinates":  [[  [-77.119759,38.791645],  [-76.909393,38.791645],  [-76.909393,38.995548],  [-77.119759,38.995548]  ]],  "type":"Polygon"  },  "country":"United States",  "country\_code":"US",  "full\_name":"Washington, DC",  "id":"01fbe706f872cb32",  "name":"Washington",  "place\_type":"city",  "url":"http://api.twitter.com/1/geo/id/0172cb32.json" } |
| quoted\_status\_id | This field only surfaces when the Tweet is a quote Tweet. This field contains the integer value Tweet ID of the quoted Tweet. Example:  "quoted\_status\_id":114749583439036416 |
| quoted\_status\_id\_str | This field only surfaces when the Tweet is a quote Tweet. This is the string representation Tweet ID of the quoted Tweet. Example:  "quoted\_status\_id\_str":"114749583439036416" |
| is\_quote\_status | Indicates whether this is a Quoted Tweet. Example:  "is\_quote\_status":false |
| quoted\_status | This field only surfaces when the Tweet is a quote Tweet. This attribute contains the Tweet object of the original Tweet that was quoted. |
| retweeted\_status | Users can amplify the broadcast of Tweets authored by other users by [retweeting](https://developer.twitter.com/rest/reference/post/statuses/retweet/%3Aid) . Retweets can be distinguished from typical Tweets by the existence of a retweeted\_status attribute. This attribute contains a representation of the *original* Tweet that was retweeted. Note that retweets of retweets do not show representations of the intermediary retweet, but only the original Tweet. (Users can also [unretweet](https://developer.twitter.com/rest/reference/post/statuses/destroy/%3Aid) a retweet they created by deleting their retweet.) |
| quote\_count | *Nullable.* Indicates approximately how many times this Tweet has been quoted by Twitter users. Example:  "quote\_count":1138  Note: This object is only available with the Premium and Enterprise tier products. |
| reply\_count | Number of times this Tweet has been replied to. Example:  "reply\_count":1585  Note: This object is only available with the Premium and Enterprise tier products. |
| retweet\_count | Number of times this Tweet has been retweeted. Example:  "retweet\_count":1585 |
| favorite\_count | *Nullable.* Indicates approximately how many times this Tweet has been [liked](https://developer.twitter.com/rest/reference/post/favorites/create) by Twitter users. Example:  "favorite\_count":1138 |
| entities | Entities which have been parsed out of the text of the Tweet. Additionally see [Entities in Twitter Objects](https://developer.twitter.com/overview/api/entities-in-twitter-objects) . Example:  "entities": {  "hashtags":[],  "urls":[],  "user\_mentions":[],  "media":[],  "symbols":[]  "polls":[] } |
| extended\_entities | When between one and four native photos or one video or one animated GIF are in Tweet, contains an array 'media' metadata. Additionally see [Entities in Twitter Objects](https://developer.twitter.com/overview/api/entities-in-twitter-objects) . Example:  "entities": {  "media":[] } |
| favorited | *Nullable.* Indicates whether this Tweet has been liked by the authenticating user. Example:  "favorited":true |
| retweeted | Indicates whether this Tweet has been Retweeted by the authenticating user. Example:  "retweeted":false |
| possibly\_sensitive | *Nullable.* This field only surfaces when a Tweet contains a link. The meaning of the field doesn’t pertain to the Tweet content itself, but instead it is an indicator that the URL contained in the Tweet may contain content or media identified as sensitive content. Example:  "possibly\_sensitive":true |
| filter\_level | Indicates the maximum value of the [filter\_level](https://developer.twitter.com/streaming/overview/request-parameters#filter_level) parameter which may be used and still stream this Tweet. So a value of medium will be streamed on none, low, and medium streams.  Example:  "filter\_level": "medium" |
| lang | *Nullable.* When present, indicates a [BCP 47](http://tools.ietf.org/html/bcp47) language identifier corresponding to the machine-detected language of the Tweet text, or und if no language could be detected. See more documentation [HERE](http://support.gnip.com/apis/powertrack2.0/rules.html#Operators). Example:  "lang": "en" |
| matching\_rules | Present in *filtered* products such as Twitter Search and PowerTrack. Provides the *id* and *tag* associated with the rule that matched the Tweet. With PowerTrack, more than one rule can match a Tweet. See more documentation [HERE](http://support.gnip.com/enrichments/matching_rules.html). Example:  "matching\_rules": " [{  "tag": "rain Tweets",  "id": 831566737246023680,  "id\_str": "831566737246023680"  }, {  "tag": "snow Tweet",  "id": 831567402366218240,  "id\_str": "831567402366218240"   }]" |

|  |  |
| --- | --- |
| current\_user\_retweet | *Perspectival* Only surfaces on methods supporting the include\_my\_retweet parameter, when set to true. Details the Tweet ID of the user’s own retweet (if existent) of this Tweet. Example:  "current\_user\_retweet": {  "id": 26815871309,  "id\_str": "26815871309" } |
| scopes | A set of key-value pairs indicating the intended contextual delivery of the containing Tweet. Currently used by Twitter’s Promoted Products. Example:  "scopes":{"followers":false} |
| withheld\_copyright | When present and set to “true”, it indicates that this piece of content has been withheld due to a [DMCA complaint](http://en.wikipedia.org/wiki/Digital_Millennium_Copyright_Act) . Example:  "withheld\_copyright": true |
| withheld\_in\_countries | When present, indicates a list of uppercase [two-letter country codes](http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) this content is withheld from. Twitter supports the following non-country values for this field:  “XX” - Content is withheld in all countries “XY” - Content is withheld due to a DMCA request.  Example:  "withheld\_in\_countries": ["GR", "HK", "MY"] |
| withheld\_scope | When present, indicates whether the content being withheld is the “status” or a “user.”  Example:  "withheld\_scope": "status" |
| geo | **Deprecated.** *Nullable.* Use the coordinates field instead. This deprecated attribute has its coordinates formatted as *[lat, long]*, while all other Tweet geo is formatted as *[long, lat]* |

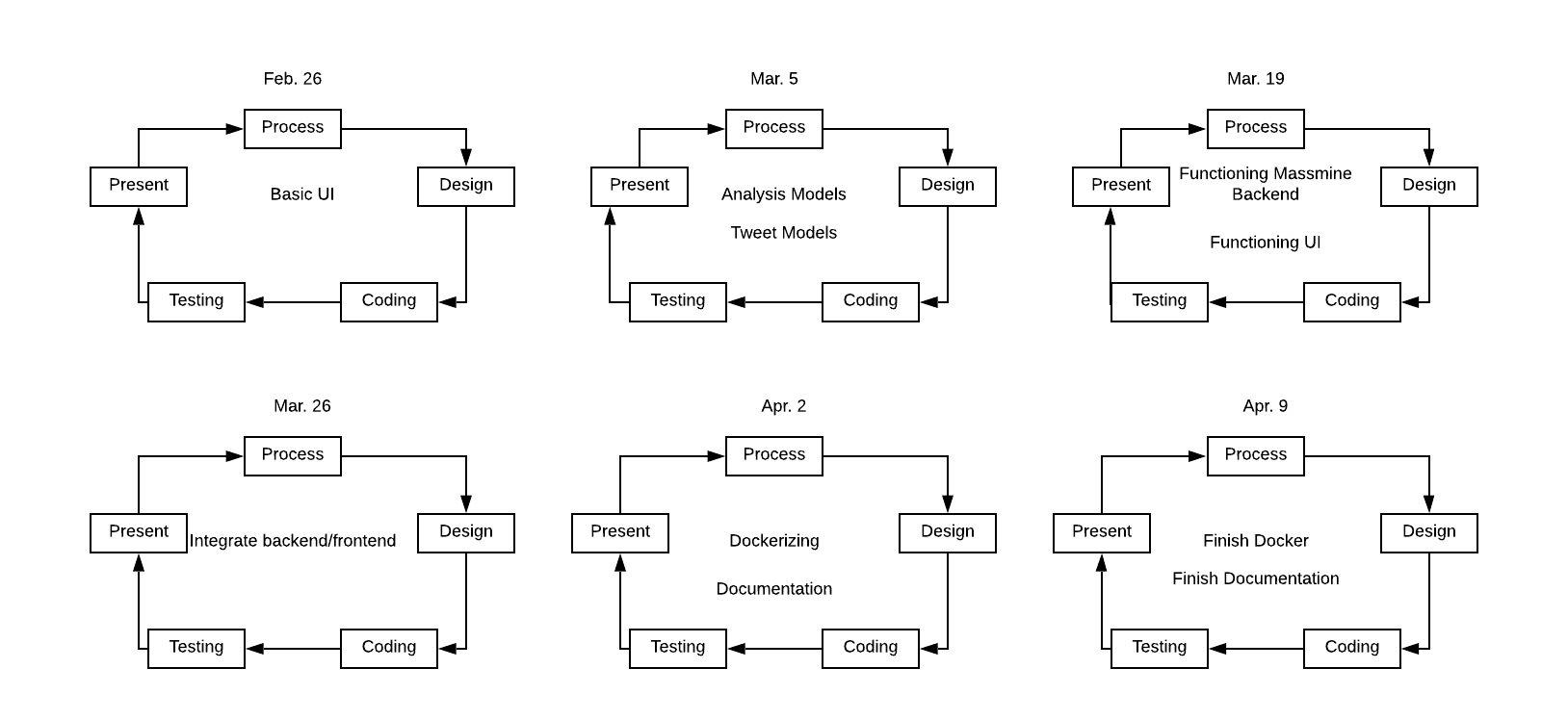
* + - * + Accessed by: Query page, analysis page
    - Analysis Model???
      * Name: analysis\_database
      * Collections:
        + analysis\_collection

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| query\_id | Identifies the particular query being used for analysis |
| query\_str | The actual query being run on a data set |

* + - * + Accessed by: Analysis page

### 5.2.2 Process models - Patti

* System Process Model (Agile):



## 5.3 Algorithm Analysis

### 5.3.1 Big - O analysis of overall System and Sub-Systems

* User Management - Patti
  + Updating/adding user: O(log n) - Looks first for username primary key. If username does not exist, adds entry
  + Checking authentication: O(log n) - Looks for username primary key and checks hashed password
* Analysis Management - Morgan

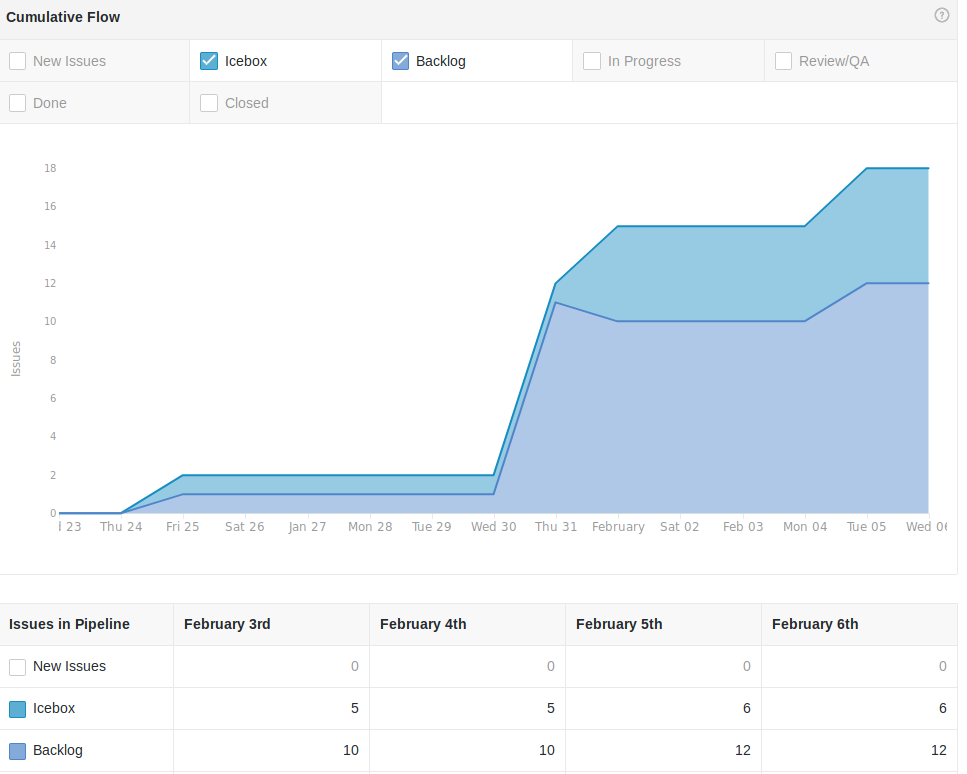
## O(n) - Displaying the analysis will require parsing through a study and displaying said study.

* + O(log(n)) - Retrieving tweets from the integrated django database
* Query Management - Patti
  + O(1) - formats user command and sends it to the massmine subsystem
* Massmine Integration - Morgan
  + Entering Tweet: O(1) - Assumes tweet-study pair does not exist already and enters a single entry to tweet\_info and another to tweet\_study\_index
  + Entering Tweet Group: O(n) - If there are n tweets in the Massmine output, there are n calls to the database

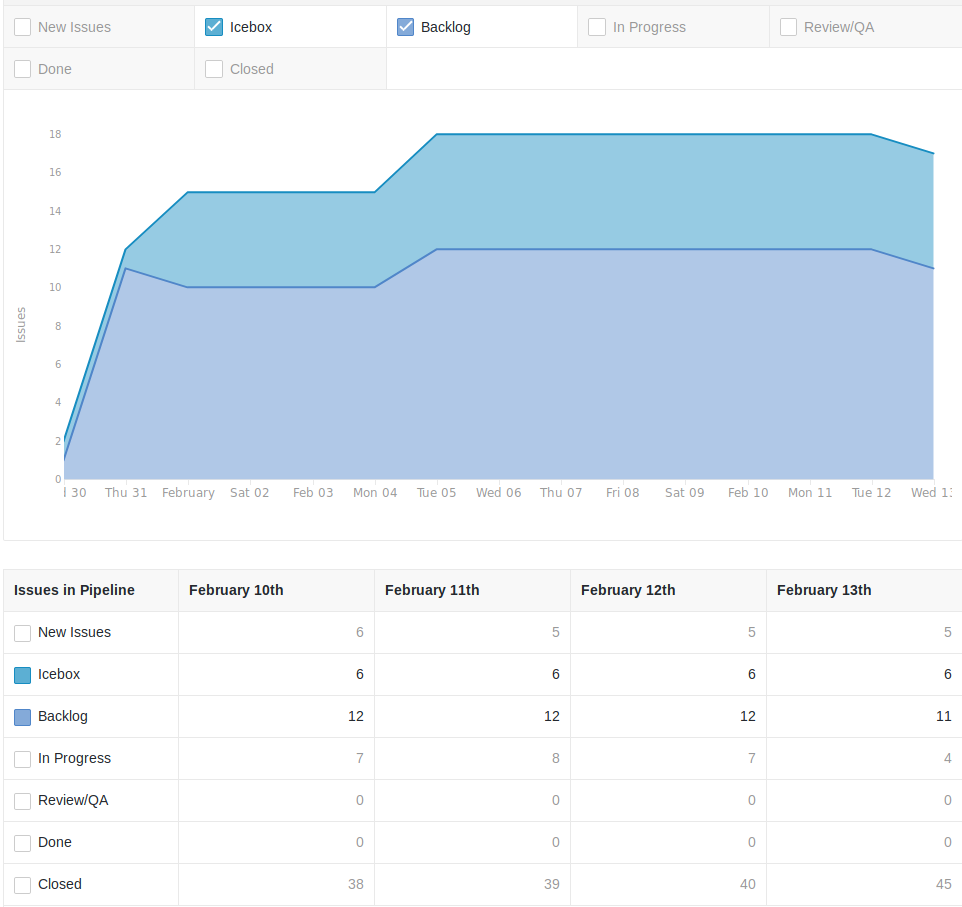
# **6. Project Scrum Report - Patti**

## 6.1 Product Backlog (Table / Diagram)

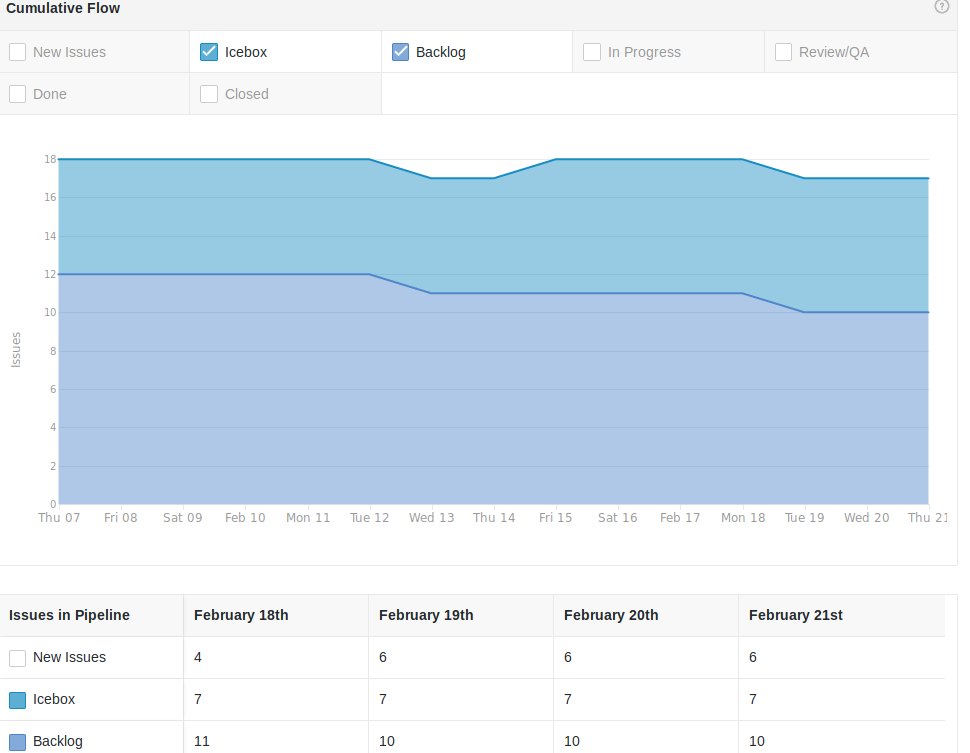
* Jan. 24 - Feb. 06



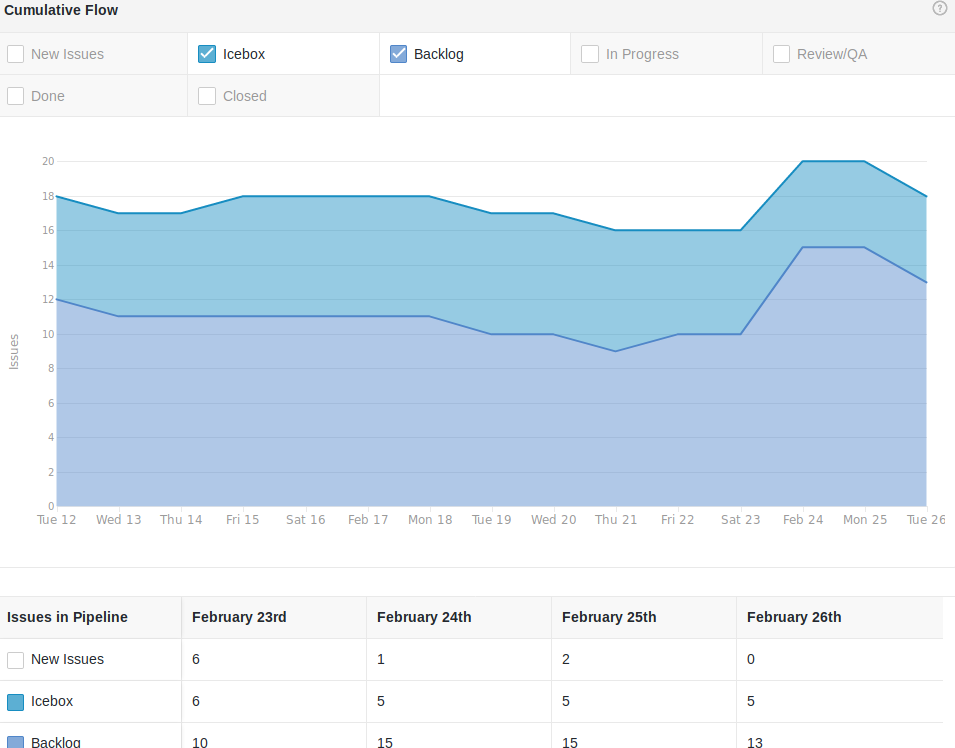
* Jan. 30-Feb. 12



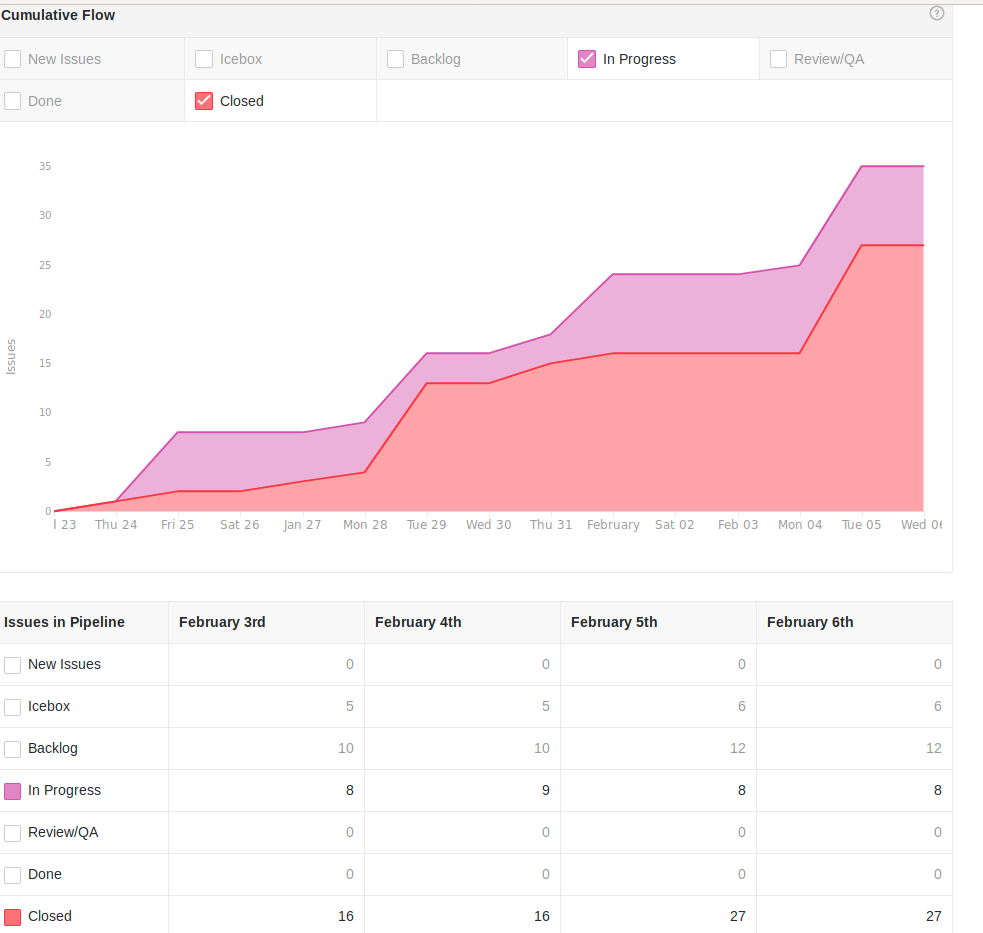
* Feb. 7-Feb. 21

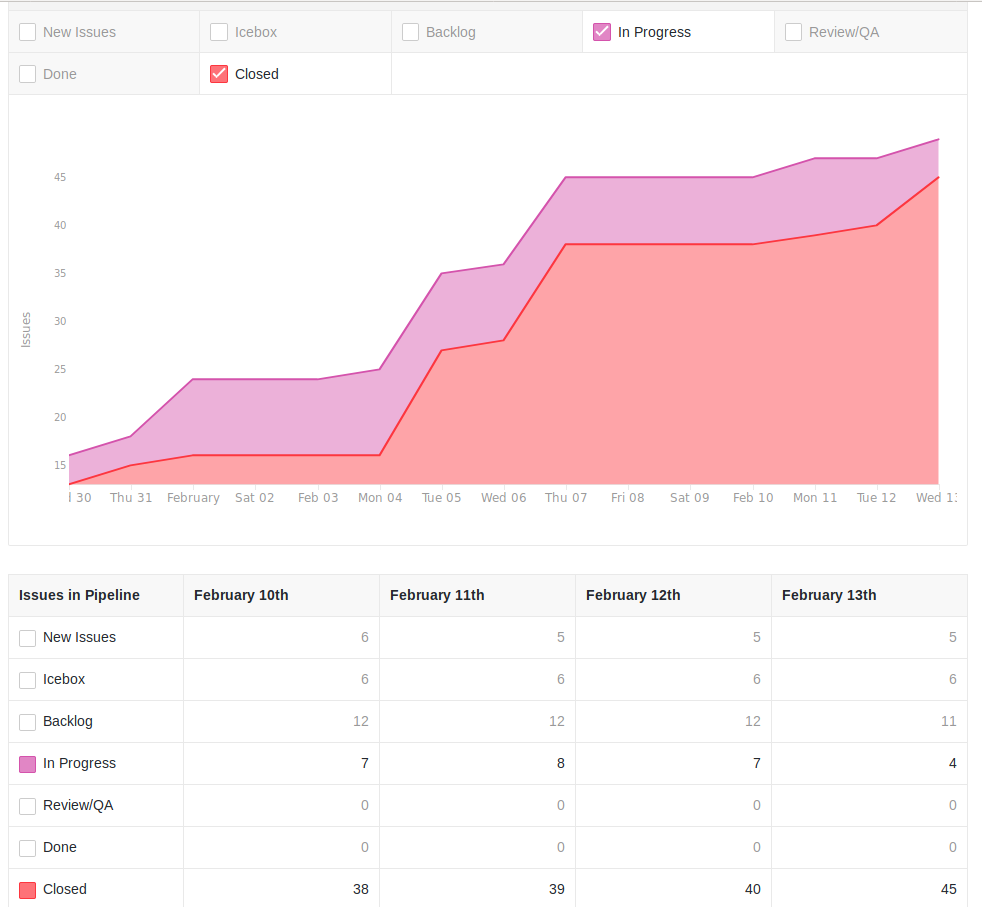


* Feb. 12 - Feb. 26

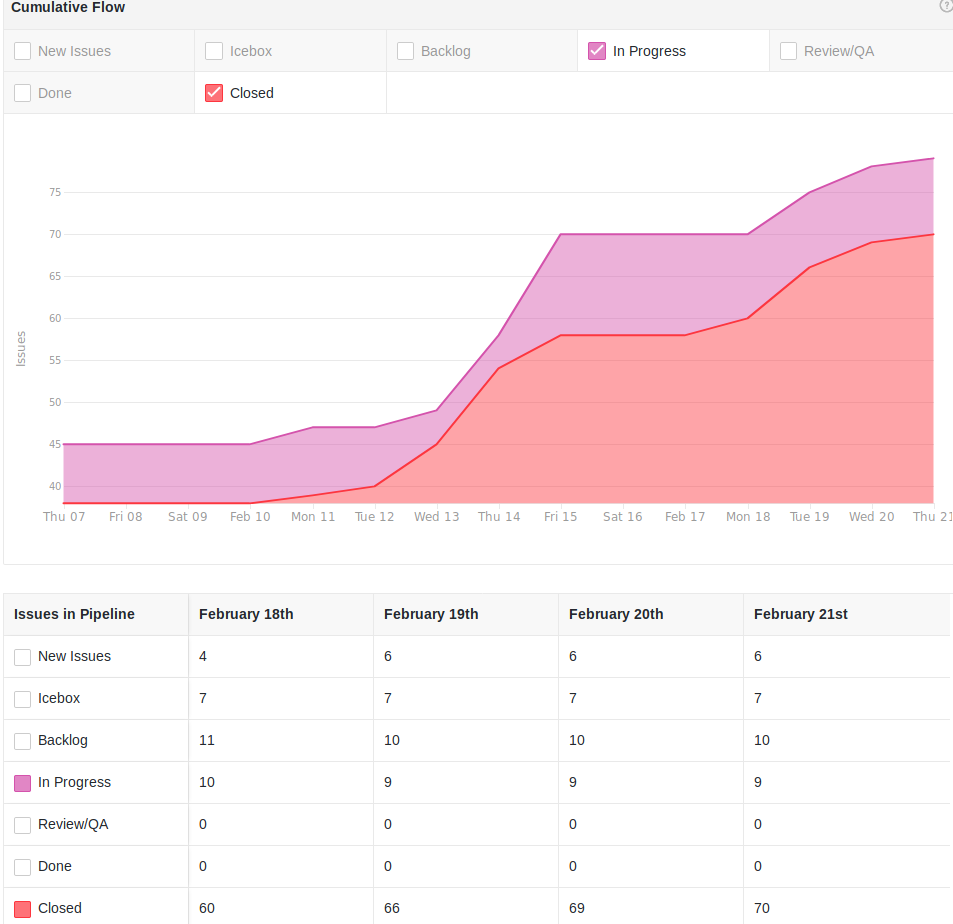


## 6.2 Sprint Backlog (Table / Diagram)

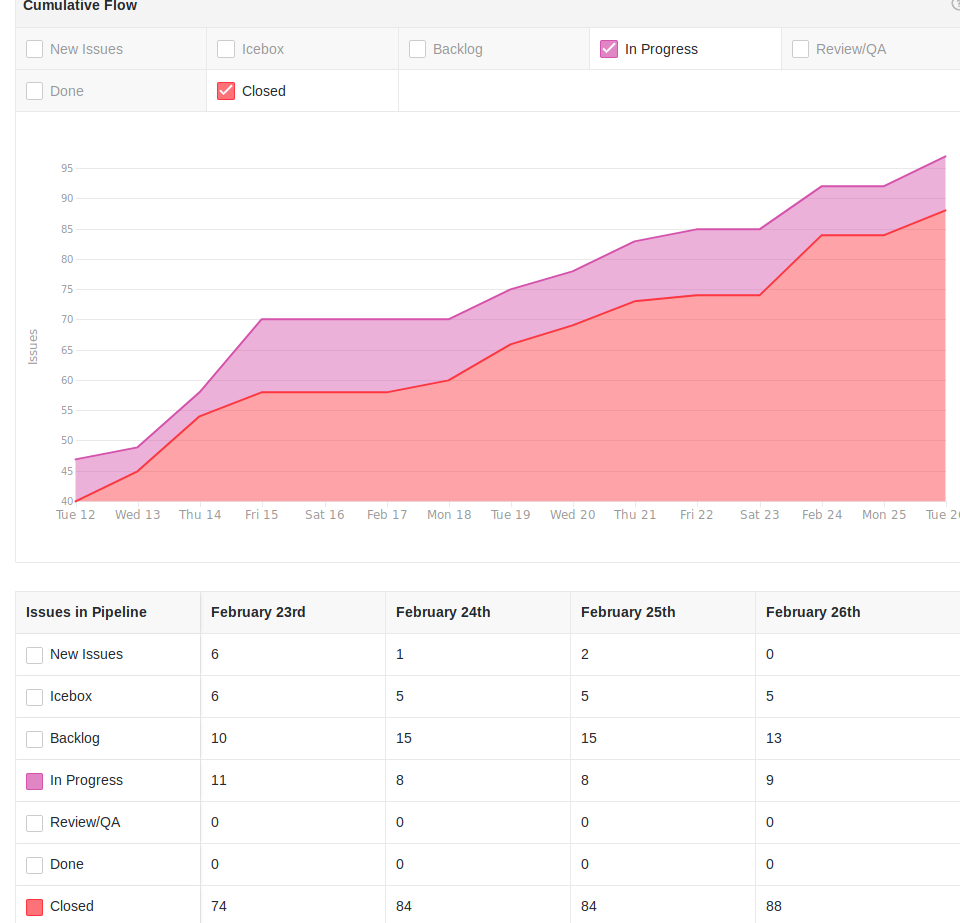
* Jan. 24-Feb.06 
* Jan. 30-Feb.12



* Feb 7 - Feb 21

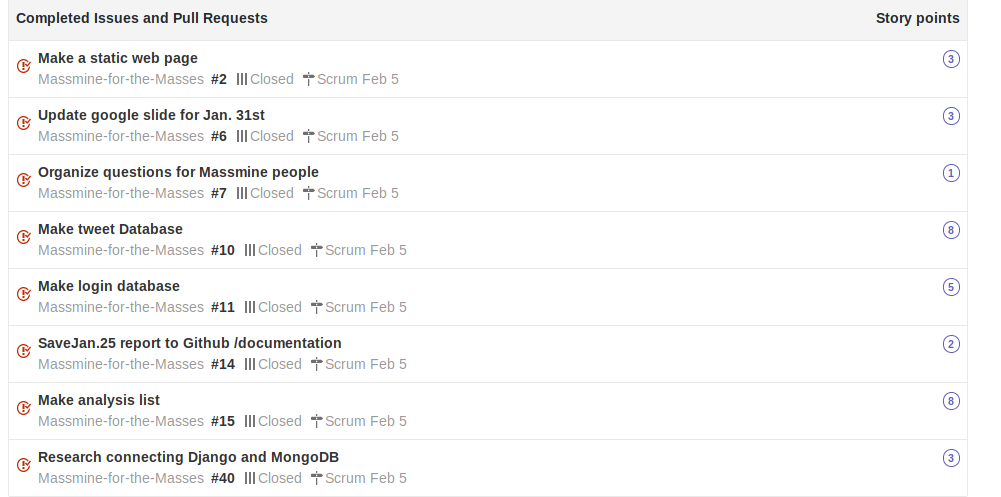
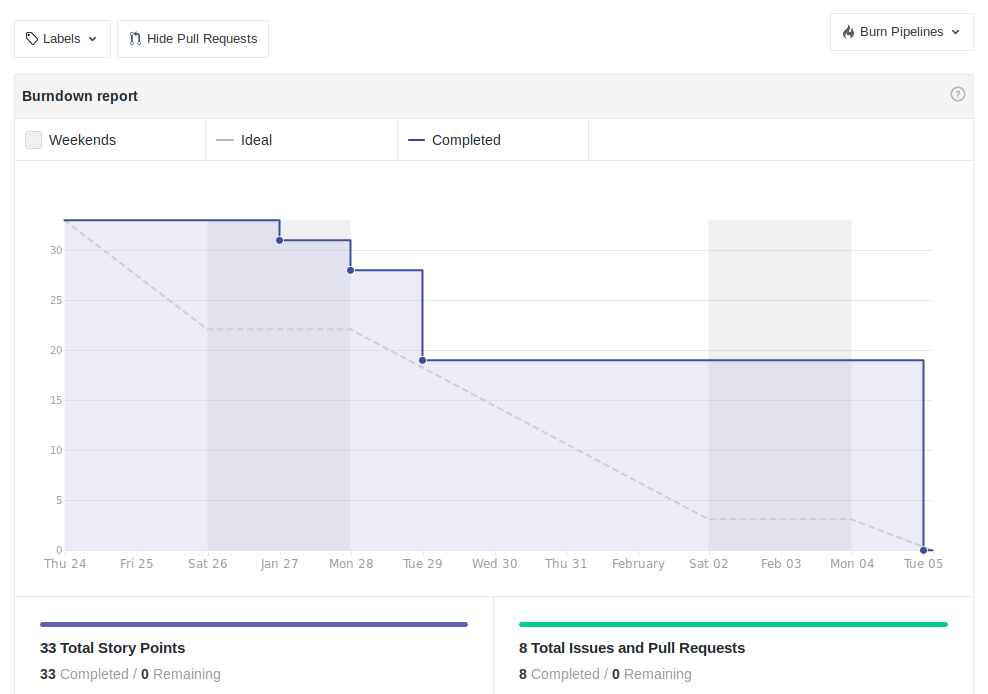


* Feb. 12 - Feb. 26

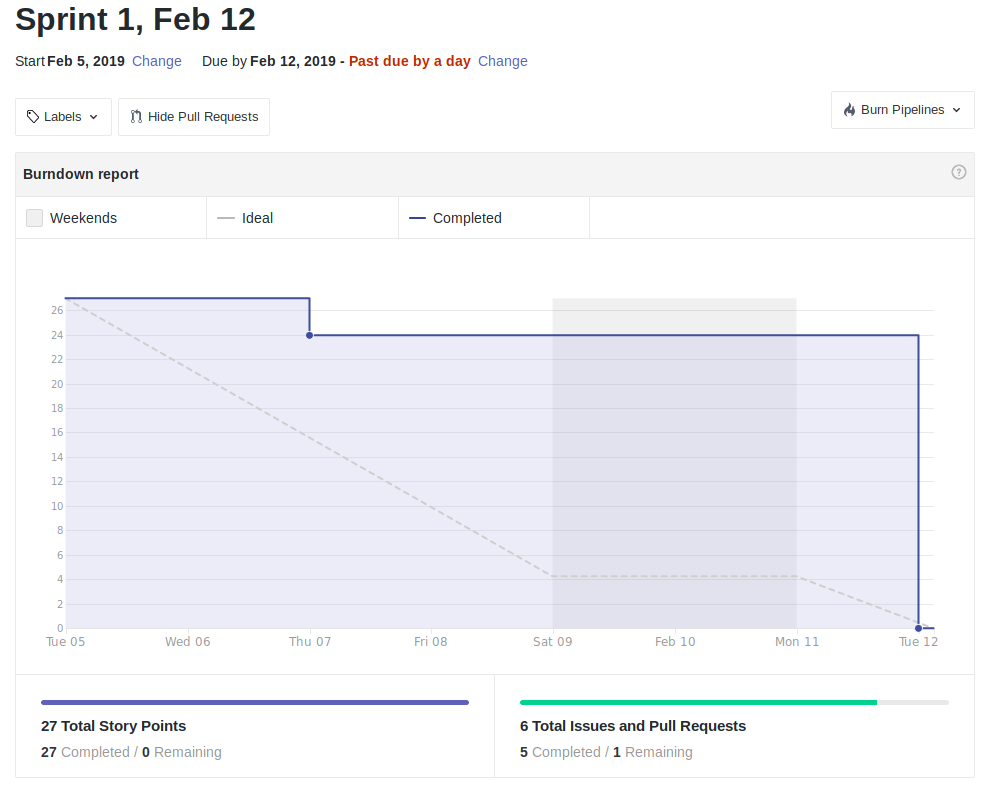


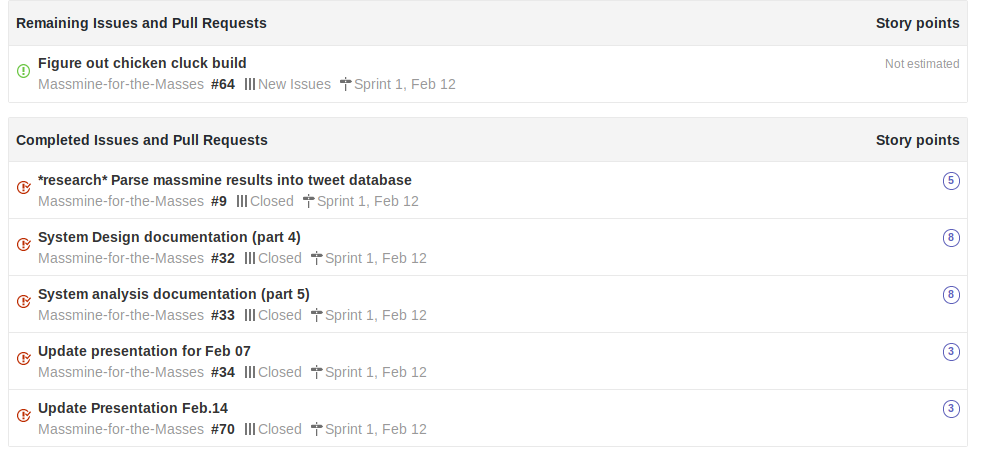
## 6.3 Burndown Chart

* For Scrum Feb 5:

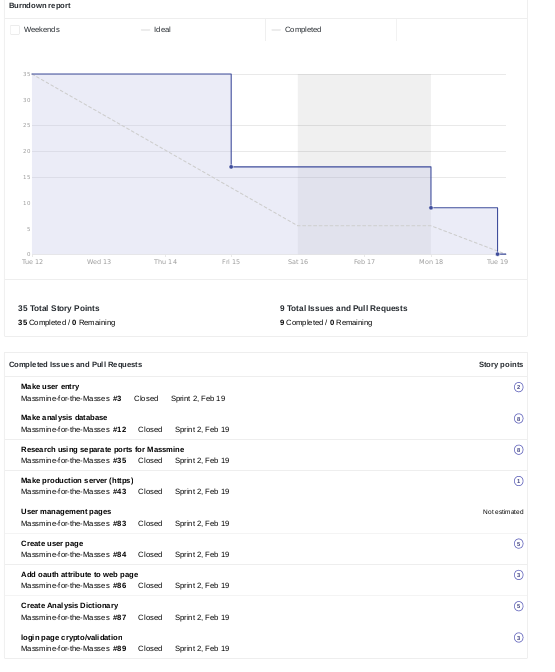


* For Scrum Feb 12:

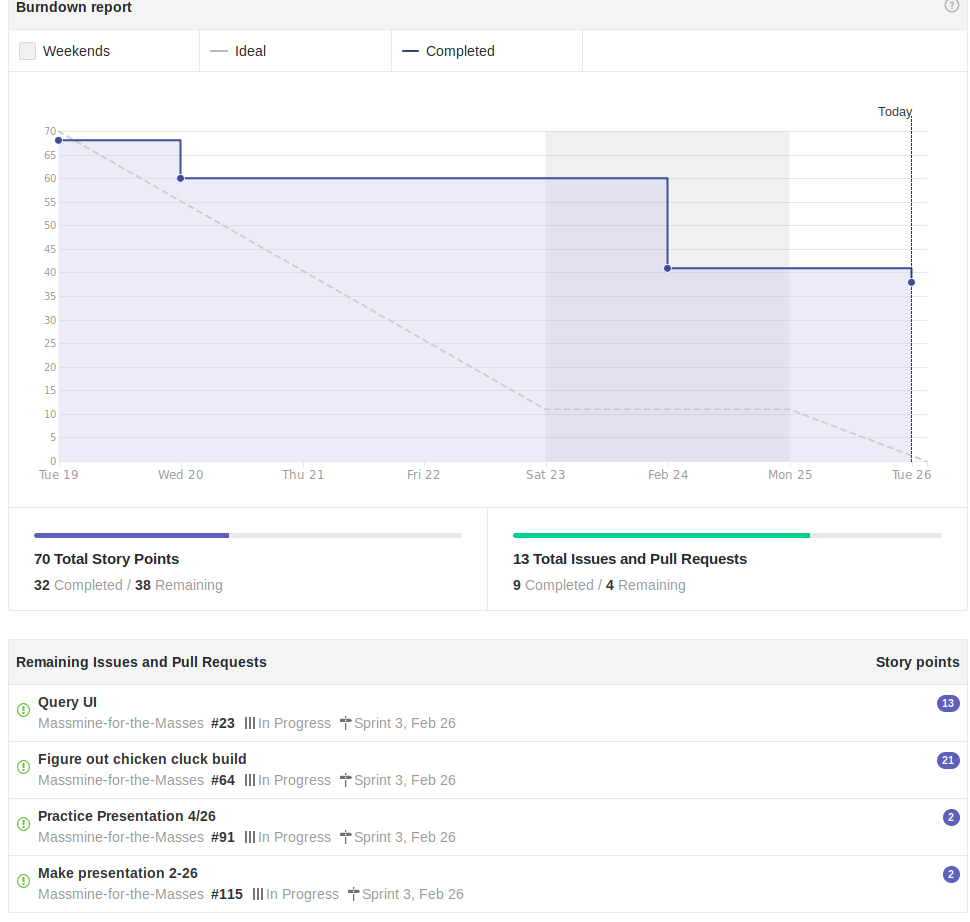




* For scrum Feb. 19:



* For scrum Feb. 26:



# **7. Subsystems**

## 

## **7.1 Subsystem 1** – User Management - *Patti*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams (eg if a UI, do a use-case)
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

## **7.2 Subsystem 2** – Analysis - *Morgan*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

## **7.3 Subsystem 3** – Query - *Logan*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

## **7.4 Subsystem 4** – Massmine Integration - *Josh*

* Initial design and model
  + Illustrate with class, use-case, UML, sequence ..... diagrams
  + Design choices
* Data dictionary
* If refined (changed over the course of project)
  + Reason for refinement (Pro versus Con)
  + Changes from initial model
  + Refined model analysis
  + Refined design (Diagram and Description)
* Scrum Backlog (Product and Sprint - Link to Section 6)
* Coding
  + Approach (Functional, OOP)
  + Language
* User training
  + Training / User manual (needed for final report)
* Testing

# **8. Complete System**

* Final software/hardware product
* Source code and user manual – screenshots as needed - Technical report
  + Github Link
* Evaluation by client and instructor
* Team Member Descriptions

## 