# Twitter Project

## Introduction

On the surface, the premise of Twitter is pretty simple. In some ways it is a slimmed down Facebook. In others, it’s a more sophisticated text messaging system than one would find on a phone. There are a lot of pieces that need to be considered to make a successful Twitter clone, however. This document will examine some of these.

## Security

Critical to any internet accessible software is security. When users read a Tweet, they expect it actually came from the user that sent it. When they create a username and password, they expect those to be securely stored. Personal information that they provide to Twitter is expected to be inaccessible to those that don’t have permission to see it.

Security should exist as its own stand-alone subsystem. The security databases should reside on completely different servers than the messaging data. The data should be encrypted in the databases, and should exist inside its own firewall, with only select APIs that allow access to the data to the rest of the system.

Twitter also acts as a middleman in internet shopping transactions. Users have the ability to store credit cards in Twitter and can then use those to place orders with merchants. This data would also need to be stored with the most secure data.

### What is kept here?

* Usernames
* Passwords
* Credit Card information
* Logging information
  + Times of logins
  + IPs connected from
* Security question answers
* Email address

## Messaging and User Customization

The next layer up is the messaging layer. This layer is outside the firewall from Security, but still behind a firewall protecting it from the internet directly. There are several sublayers here, which will be detailed separately.

### Database Optimization layer

Twitter handles billions of requests from millions of users every day. When a user logs into the webpage, first, Twitter assembles a list of everyone the user is following. It gathers the posts from those followed users. It then displays all of the data in a prescribed order. There may me algorithms that help limit the data displayed based off of user preferences or behaviors. If a user is following 500 other users, that is 500 profiles worth of data to parse and display in a meaningful fashion. Those messages also need to be checked against their users’ security settings to ensure they should even be displayed to followers.

The sheer volume of data that needs to be processed on each login requires very optimized database design and queries. All database calls would pass through this layer. The code here would be responsible for coming up with ideally optimized queries to pass into the database handler. The databases themselves would need to be constructed with proper indexing, paging, and server side procedures created.

### Message Direction layer

Twitter messages can go to multiple targets. Maybe a message is directed to a particular user, so @User gets called in the message. Or maybe a user posts a public message about a current topic, so they use a hashtag to discuss #PresidentialElection, for example. The Message Director would act as the postmaster ensuring that the message ends up in the proper location.

The Message Director is the code that calls the Database layer and puts together all of the messages that appear on a home page.

There are 3 ways to direct a message. The first is to just tweet a message. This message appears to all of my followers, so when they view their homepage, my message will appear there. The second is to handle a directed message. For example, if a user directs a message to @WhiteHouse, an admin for the @WhiteHouse account can see that message, and then retweet it to their own homepage if they find it worthwhile. The final way is to handle a public post. Every message posted with the tag #Politics would get added to a massive virtual bulletin board where users can browse every post made to it. When a user goes to view #Politics, the Message Director needs to assemble all of the tweets from potentially thousands of users and display them meaningfully.

#### What is the Message Director responsible for?

* Calling the Database layer for data
* Assembling data for home pages using user’s follower information
* Directing messages to a particular account if specified in the tweet
* Assembling posts to put under hashtags

### User Customization layer

This is the layer where users determine what their homepage contains. It helps the user determine which pages to follow. Beyond the basic settings and preferences that are settable by the end user, there is also a complex series of algorithms that help a user find new profiles to follow. For example, if a user follows several college basketball teams, the algorithm might suggest the NCAA basketball page, ESPN, and Fox Sports.

These intelligent algorithms are critical to users returning to Twitter frequently. They pull up news stories that might be of interest to a particular user. They display the trends of the day, so users know what other people are talking about. They keep the home pages fresh by constantly suggesting new pages to follow. While the previous layers are the nuts and bolts required for Twitter to exist and scale properly, this layer is the reason why Twitter exists.

## Web Hosting and APIs

This is the final major layer. This is the piece that sits outside the firewall, accessible to anyone on the internet. The code here is what takes all of the data and messages assembled by all of the previous layers and displays them in a way that is pleasing to the end user. No data is stored at this layer. Instead, the code here is more concerned with purely web and internet usability.

### What happens here?

* Code load balances web requests across server farms
* Web servers create the dynamic pages to display to web browsers
* APIs are exposed allowing connection to the data outside the web pages
  + Phone/Tablet Apps
  + Other websites
* Communicates with other layers to authenticate users and assemble their data