

Enterprise Distributed Systems

Class Project – Twitter Simulation (Fall 2019)

Project Due Date: 4th December 2:30 PM before class

Presentation: 20th November (Upload Presentation to canvas before the class.)

Demo: 4th December in class

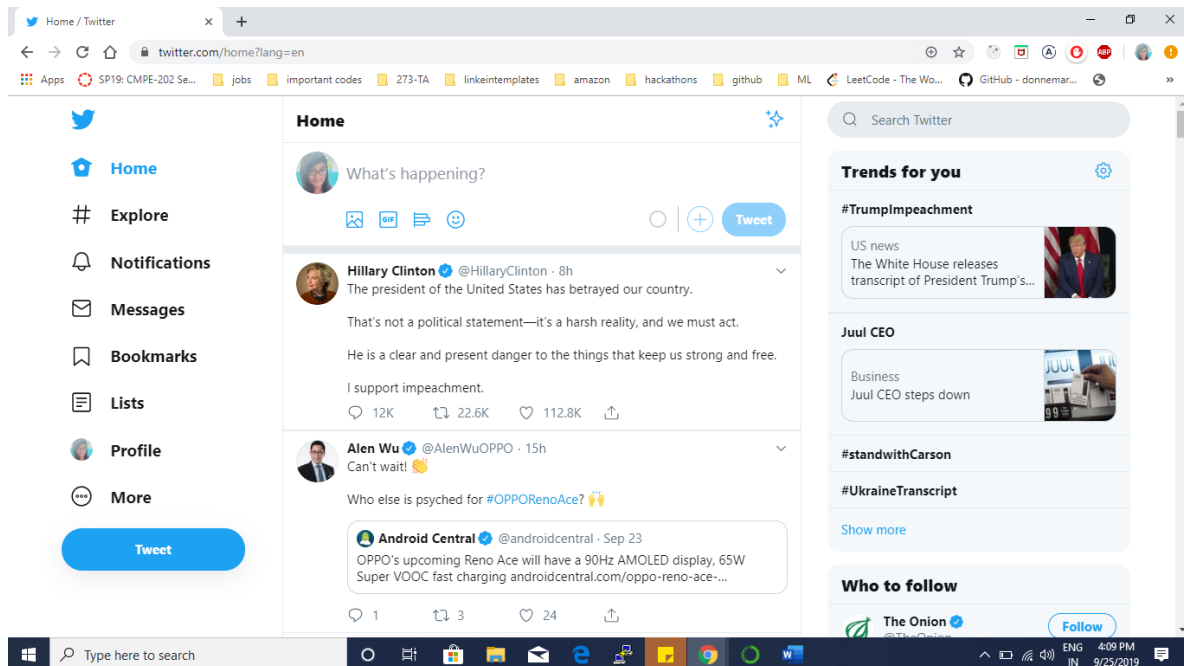
This is a peer reviewed group project with up to 6 people in the team.

**Turn in the following API document on or before 23rd October 2:30 PM on canvas.
No late submissions will be accepted!**

- An API design document of services provided by the application.

Twitter

Twitter (/ˈtwɪtər/) is a microblogging and social networking service on which users post and interact with messages known as "tweets". Registered users can post, like, and retweet tweets, but unregistered users can only read them. Twitter, Inc. is based in San Francisco, California.



Project Overview

In this project, you will design a 3-tier distributed application that will implement the functions of **Twitter** for different cloud services. You will build on the techniques used in your lab assignments to create the system.

In Twitter prototype, you will manage and implement different types of objects:

- Users
- Tweets
- Lists

For each type of object, you will also need to implement an associated **database schema** that will be responsible for representing how a given object should be stored in a database.

Your system should be built upon some type of distributed architecture. You have to use message queues as the middleware technology. You will be given a great deal of “artistic liberty” in how you choose to implement the system. **These are the basic fields that needed. You can modify this schema according to you need and assign primary keys and foreign keys. Example: Create relation between User and Tweets, Lists etc. Above mentioned are just examples.**

Your system must support the following types of entities:

• **User** – It represents information about an individual user registered on Twitter. You must manage the following information for this entity:

- ⇒ First Name
- ⇒ Last Name
- ⇒ City
- ⇒ State
- ⇒ Zip code
- ⇒ Profile Image
- ⇒ Description about yourself
- ⇒ Username (Username will be displayed throughout the application along with profile name)

• **Tweet**

- ⇒ Tweet(**280 characters limit**)
- ⇒ Tweet can have Images
- ⇒ Tweet Owner

- ⇒ Retweets
- ⇒ Replies
- ⇒ Likes
- ⇒ Posted Date/Time

- **List**

- ⇒ curated group of Twitter accounts.
- ⇒ create your own lists
- ⇒ subscribe to lists created by others.
- ⇒ list timeline will show you a stream of Tweets from only the accounts on that list.
- ⇒ Posted Date/Time

- **User Feed**

- ⇒ Collection of Tweets.
- ⇒ User can view individual tweet by clicking on it.
- ⇒ User can unfollow a person from his feed.
- ⇒ User retweet from his feed.
- ⇒ User like or reply a tweet from his feed.

Project Requirements

Your project will consist of three tiers:

- The client tier, where the user will interact with your system
- The middle tier/middleware/messaging system, where the majority of the processing takes place
- The third tier, comprising a database to store the state of your entity objects

Tier 1 — Client Requirements

The client will be an application that allows a user to do the following:

- **User Module/Service:**

- Create a Twitter account (Reference Twitter website for fields information)
- Delete/Deactivate an existing Account.
- Change user's profile information (name, address, profile image etc) – *This function must support the ability to change ALL attributes*
- Display information about the user.
- Search for topics using hashtag.
- Search for people
- Retweet a tweet
- Reply on a tweet
- Like a tweet
- Write a Tweet
- Send/Receive messages
- Follow user and provide option for other user users to follow you

User Dashboard:

- View list of followers and people followed by the user
- Inbox to manage messages
- View tweets added by user
- View retweets added by user
- View bookmarked tweets.
- View tweets liked by user
- Pagination feature should be added to suitable pages.

Tweet feature:

- A tweet may include text, images, hyperlinks etc.
 - A tweet can be bookmarked.
 - A tweet can be deleted by the owner.
 - Provide option for other like/reply/retweet your tweet.
-
- Sample User Analysis Report



A User will have his dashboard which will have different types of graphs and charts for the statistics that needs to be displayed.

Graphs Criteria

- 1) Retrieve data from database and show top 10 tweets with its views (Bar, Pie or any kind of graph)
- 2) Top 10 tweets with more likes (Bar, Pie or any kind of graph)
- 3) Top 5 tweets with more retweets.
- 4) Graph for number of tweets(hourly/daily/monthly).

User's Graph Criteria

- 1) Graph for his profile views/day (Just a count. Graph should have 30 days limit.)

The client should have a pleasing look and be simple to understand. Error conditions and feedback to the user should be displayed in a status area on the user interface. Ideally, you will use an IDE tool to create your GUI.

Tier 2 — Middleware

You will need to provide/design/enhance a middleware that can accomplish the above requirements. You have to implement it using REST based Web Services as Middleware technology. Next, decide on the Interface your service would be exposing. You should ensure that you appropriately handle error conditions/exceptions/failure states and return meaningful information to your caller. Make sure you handle the specific failure cases described below.

ON OR BEFORE the date listed below, you must turn in a document describing the interface your server will use which precisely means that you have to give API request-response descriptions.

Your project should also include a model class that uses standard modules to access your database. Your entity objects will use the data object to select/insert/update the data in the relational database.

User Kafka as a messaging platform for communication between front-end channels with backend systems.

Extra Credit: (10 points)

Use of GRPC microservices (all components of a domain should be classified as a microservice) as discussed in the class.

*There is no partial credit for extra credit, you will either get full marks or none based on your implementation.

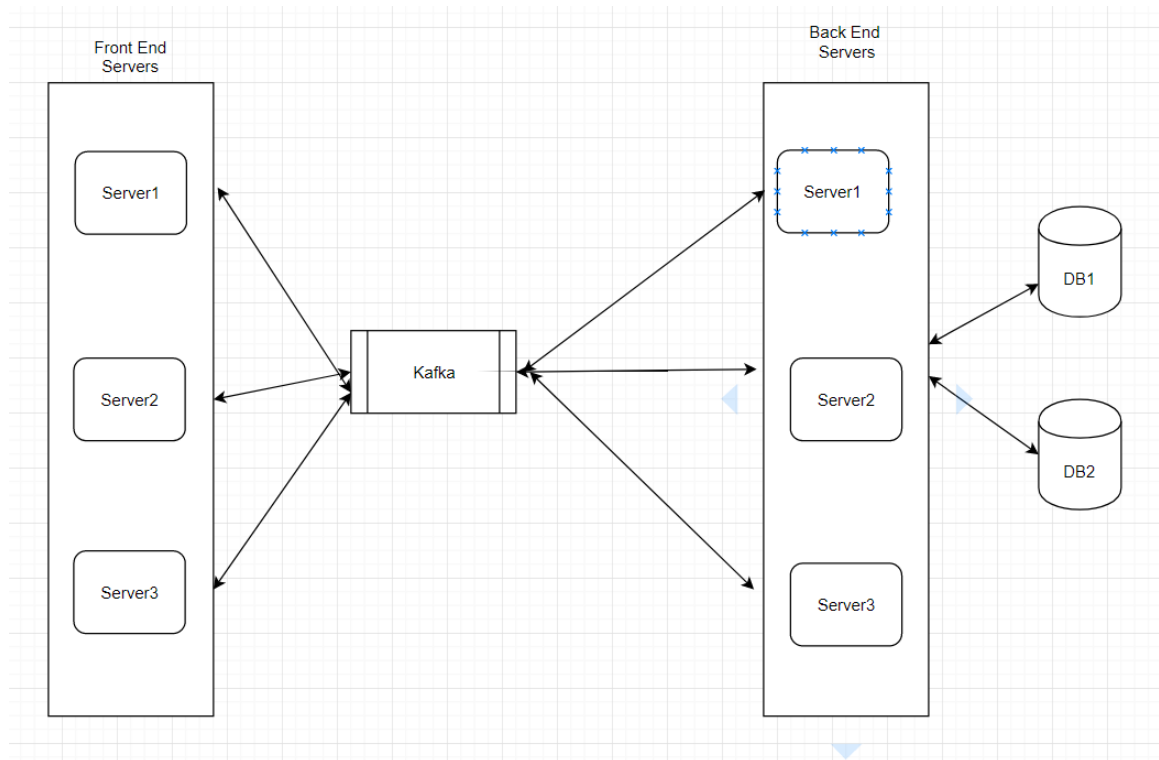
Tier 3 — Database Schema and Database Creation

You will need to design a database table that will store your relational data. Choose column names and types that are appropriate for the type of data you are using. You will also need a simple program that creates your database and its table. The MySQL Workbench is a very efficient and easy tool for it.

You will need to decide which data are stored in MongoDB and MySQL. (Hint: think about pros and cons of MongoDB vs MySQL)

Distributed System Deployment

Follow the distributed services architecture shown below.



Scalability, Performance and Reliability

The hallmark of any good object container is scalability. A highly scalable system will not experience degraded performance or excessive resource consumption when managing large numbers of objects, nor when processing large numbers of simultaneous requests. You need to make sure that your system can handle many listings, users and incoming requests.

Pay careful attention to how you manage “expensive” resources like database connections.

Your system should easily be able to manage 10,000 Users, 10000 Tweets. Consider these numbers as **minimum** requirements.

Further, for all operations defined above, you need to ensure that if a particular operation fails, your system is left in a consistent state. Consider this requirement carefully as you design your project as some operations may involve multiple database operations. You may need to make use of transactions to model these operations and roll back the database in case of failure.

Performance Charts that shows difference in performance by adding different distributed features at a time

Chart 1: Single Chart of No Performance Optimization, Connection Pooling, Kafka, SQL Caching, Other optimization technique discussed in the class.

Chart 2: Performance Comparison of services with below deployment configurations

1. 1 UI Web Server 1 DB instance 1 Services server
2. 1 UI Web Server 1 DB instance 3 Services Servers
3. 2 UI Web Server 2 DB instance 4 Services Servers

PowerPoint Presentation (upload to Canvas)

1. Names and group number
2. Database Schema
3. Performance graph to show different features combination such as B (Base), S (SQL Caching), D (DB connection pooling), K (Kafka)
a) B b) B+D c) B+D+S d) B+D+S+K e) B+D+S+K+ other techniques
Populate DB with at least 10,000 users, 10,000 questions and 10,000 answers before measure performance.
4. Performance graph for distributed systems

Testing

To test the robustness of your system, you need to design a test harness that will exercise all the functions that a regular client would use. This test harness is typically a command-line program. You can use your test harness to evaluate scalability (described above) by having the test harness create thousands of listings and users. Use your test harness to debug problems in the server implementation before writing your GUI.

- a. Following tasks to be tested using Mocha: Implement ten randomly selected REST web service API calls using Mocha. Display the output in the report.

Other Project Details

Turn in the following on or before the due date. No late submissions will be accepted!

- A title page listing the members of your group
- A page listing how each member of the group contributed to the project (keep this short, one paragraph per group member is sufficient)
- A short (5 pages max) write-up describing:
 - a) Your object management policy
 - b) How you handle “heavyweight” resources
 - c) The policy you used to decide when to write data into the database
- A screen capture of your client application, showing some actual data
- A code listing of your client application
- A code listing of your server implementations for the entity objects
- A code listing of your server implementation for the session object
- A code listing of your main server code
- A code listing of your database access class
- A code listing of your test class
- Any other code listing (utility classes, etc)

- Output from your test class (if applicable)
- A code listing of your database creation class (or script)
- A screen capture showing your database schema
- Observations and lessons learned (1 page max)

The possible project points are broken down as follows:

(One of grading criteria is subjective and relative strength and weakness of your project compared to other projects)

Total Score – 105 points

- **40pts for basic operation** – Your server implementation will be tested for proper operation of some aspect of your project. Each passed test is worth some point(s) toward the total score, depending on the test. Each failed test receives 0 points.
- **15 pts for scalability and robustness** – Your project will be judged on its scalability and robustness. I will test your program with thousands of objects; it should not exhibit sluggish performance, and it definitely should not crash. In addition to performance improvement techniques cover in the class, you are required to implement SQL caching using Redis and show performance analysis.
- **15 pts for distributed services** – Divide client requirements into distributed services and deploy them to AWS/Heroku etc. and Deploy the database to AWS/MLab etc. Each service will be running on backend connected by Kafka and divide your data into MongoDB and MySQL and provide performance data with justification on results.
- **15 pts for Analysis report, user tracking** – Devise your own web pages/user/job tracking using logs and explain why it is effective to analyze a web site and user behavior.
- **10 pts for the client** – You will get more points if your GUI is similar to Quora.com.
- **10 pts for your test class and project write-up**

Hints:

- Maintain a pool of DB connections – Remember that opening a database connection is a resource-intensive task. It would be advisable to pre-connect several database connections and make use of them throughout your database access class so you don't continually connect/disconnect when accessing your database.
- Cache entity lookups – To prevent a costly trip to the database, you can cache the state of entity objects in a local in-memory cache. If a request is made to retrieve the state of an object whose state is in the cache, you can reconstitute the object without checking the database. Be careful that you invalidate the contents of the cache when an object's state is changed. In particular, you are required to implement **SQL caching using Redis**.
- Don't write data to the database that didn't change since it was last read.
- Focus **FIRST** on implementing a complete project – remember that a complete implementation that passes all of the tests is worth as much as the performance and scalability part of the project.
- Do not over-optimize. Project groups in the past that have tried to implement complex Optimizations usually failed to complete their implementations.

Exceptions/Failure Modes

Your project MUST properly handle the following failure conditions

- Creating Duplicate User
- Addresses (for Users) that are not formed properly (see below)

For more failure conditions see Other Notes below

Other Notes

State abbreviation parameters

State abbreviations should be limited to valid US state abbreviations or full state names. A Google search will yield these valid values. Methods accepting state abbreviations as parameters are required to raise the 'malformed_state' exception (or equivalent) if the parameter does not match one of the accepted parameters. You do not need to handle US territories such as the Virgin Islands, Puerto Rico, Guam, etc.

Zip code parameters

Zip codes should adhere to the following pattern:

[0-9][0-9][0-9][0-9][0-9]

or

[0-9][0-9][0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]

Examples of valid Zip codes:

95123

95192

10293

90086-1929

Examples of invalid Zip codes:

1247

1829A

37849-392

2374-2384

Peer Review

Each team member should write review about other team members except himself/herself. It is confidential which means that you should not share your comments with other members. Peer Review is submitted separately on Canvas.