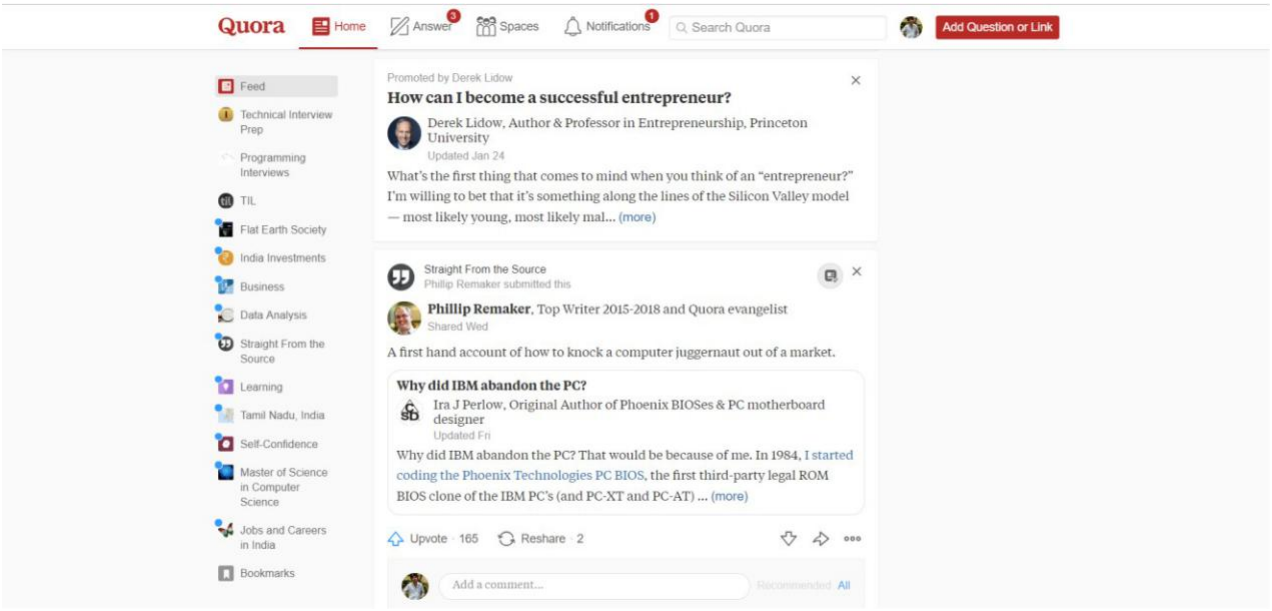
# Enterprise Distributed Systems

**This is a group project with 4 people in the team.**

## Quora

Quora is a question-and-answer website where questions are asked, answered, edited, and organized by its community of users in the form of opinions. Its publisher, Quora Inc., is based in Mountain View, California. The company was founded in June 2009, and the website was made available to the public for the first time on June 21, 2010. Users can collaborate by editing questions and suggesting edits to answers that have been submitted by other users.



## Project Overview

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

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

- Users
- Questions
- Answers

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 Questions, Answers 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 Quora. You must manage the following information for this entity:

⇒ First Name
⇒ Last Name
⇒ City
⇒ State
⇨ Zip code
⇨ Profile Image
⇒ Education
⇨ Career Information
⇨ Description about yourself
⇒ Profile Credential (Profile credential will be displayed throughout the application along with profile name)

• **Question**

⇒ Question ID
⇒ Question
⇒ Question Owner
⇒ Topic(s) (Collection of topics which the question is related to)

⇒ Followers

⇒ Answers (Collection of Answer objects)

⇒ Posted Date/Time

- **Answer**

  ⇨ AnswerID ⇨

  Answer

  ⇨ AnswerOwner ⇨

  Images (Optional)

  ⇨ isAnonymous (User can answer the question as an anonymous user – not revealing his identity)

  ⇨ Upvotes (Tracks the upvotes for the answer) ⇨

  Downvotes

  ⇨ Comments (Collection of Comments from users)

  ⇨ Posted Date/Time

- **User Feed**

  ⇒ Collection of Questions and Answers

  ⇒ User can view individual answer by clicking on a particular question.

  ⇨ User can answer any question from the feed

  ⇨ User can follow a question from his feed. When a new answer for the followed question is added a notification is sent to user.

## 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 Quora account (Reference Quora 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 questions
  - Search for topics
  - Search for people, view people profiles.
  - Read answers, bookmark them.
  - Upvote/Downvote an answer
  - Comment on an answer
  - Follow a question/topic
  - Answer a question
  - 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 Answers added by user
- View questions added by user
- View bookmarked answers
- A 'Your content' page which shows a timeline of activities performed by user. (https://www.quora.com/content)
- Perform filter on Your content page on type of activity (Questions Asked, Questions Followed, Answers), time of activity (year based) and sort order (Newest first, Oldest first).
- Pagination feature should be added to suitable pages.

**Answer a question feature:**

- A user can answer any question.
- An answer may include text, images, hyperlinks etc.
- User may opt to answer a question anonymously.
- User should be able to edit his/her previous answers.
- Provide option for other user to comment on the answer
- Provide option for other users to upvote or downvote your answer

- 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 Answers with its views (Bar, Pie or any kind of graph)
2) Top 10 answers with more upvotes (Bar, Pie or any kind of graph)
3) Top 5 answers with more downvotes.
4) Graph for number of Bookmarked answers

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.

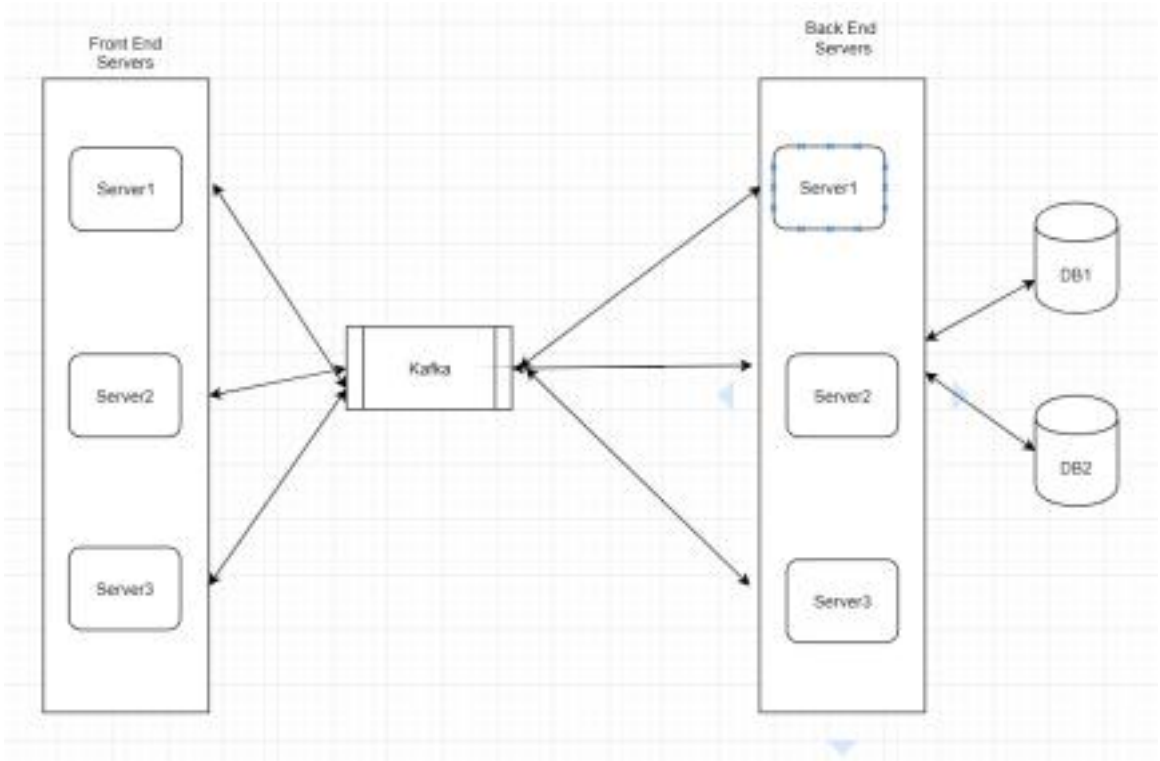
## Tier 3 — Database Schema and Database Creation

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, 10,000 Questions and 10,000 Answers. 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)

## 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