

[Show Submission Credentials](#)

P3. Cloud Storage - Heterogeneous Storage on the Cloud

P3. Cloud Storage - Heterogeneous Storage on the Cloud

 Introduction


 Scenario: Build Your Own Social Network Website

 Task 1: Implementing Basic Login with SQL

 Task 2: Storing Social Graph using Neo4j

 Task 3: Build Homepage using MongoDB

 Task 4: Put Everything Together

 Task 5: Caching

 Project Reflection Task (Mandatory, graded)

 Project Survey

 Project Discussion

Introduction

Heterogeneous Storage on the Cloud (Part 2)

Information

Learning Objectives

This project will encompass the following learning objectives:

1. Compare the advantages and disadvantages of SQL and NoSQL databases and their suitable application domains.
2. Configure, populate and deploy several heterogeneous, SQL and NoSQL, databases in a social network web service context to understand the necessity of heterogeneous storage for different data models.
3. Provision, configure and manage cloud resources to provide several NoSQL databases and a Database-as-a-Service (DBaaS) cloud offering.
4. Design effective database schema based on the requirements of an application.
5. Design and develop efficient database queries using the Java API for databases to fetch data from heterogeneous SQL and NoSQL databases.
6. Implement and evaluate MySQL indexing to improve the performance of MySQL queries.
7. Implement and evaluate MongoDB indexing to improve the performance of NoSQL queries.
8. Implement the caching mechanism to achieve faster responses.
9. Implement Infrastructure as Code (IaC) using Terraform and Azure Resource Manager (ARM) to orchestrate and manage virtual machines and databases

Information

General Details

The following table contains general information about this project module:

Prerequisites	Java, MySQL, and MySQL Connector
Primers	NoSQL Primer, MySQL Primer, MongoDB Primer, Neo4j Primer
Applicable languages	Java
Applicable cloud platform	Azure Only
Tags Required	Key: project Value: social-network

Danger

Special Deadline

You will only have **one week** to finish this section of Project 3, instead of the regular two-week period.

Resource Tagging

For this project, assign the tag with Key: **project** and Value: **social-network** for all resources.

Grading Penalties

The following table outlines the violations of project rules and their corresponding grade penalties for this week's project.

These rules apply for the week when the current project is active.

Violation	Penalty of the project grade
Incomplete submission of required files (including all the given Java files)	-10%
Submitting your Azure or Andrew credentials in your code for grading	-100%
Submitting database credentials in your code for grading	-100%
Submitting only executables (.jar , .pyc , etc.) without human-readable code (.py , .java , .sh , etc.)	-100%
Attempting to hack/tamper the autograder in any way	-100%
Cheating, plagiarism, or unauthorized assistance (please refer to the university policy on academic integrity and our syllabus)	-200% & potential dismissal

Tagging and Budget Notes on Azure

We suggest that you tag your Azure resources. You will not incur any penalty if you do not tag resources in this project.

We recommend you run the terraform commands from Azure Cloud Shell while setting up the Student-VM

Be sure to use the correct subscription when you are using the Azure portal or Azure CLI. If you have multiple subscriptions, you can use the command below to change the default subscription.

```
az account list --output table --refresh
az account set --subscription <subscription_id>
```

In this project, the recommended Azure budget is \$50. Your subscription will be disabled for the entire course if you overspend the total subscription budget. You will NOT be able to continue working on Azure in this course and we will not increase the budget or reactivate your subscription.

We suggest that you plan the budget carefully. Note that Azure Classroom subscription has its own cost monitoring portal which is different from the Cost Management + Billing (https://portal.azure.com/#blade/Microsoft_Azure_Billing/ModernBillingMenuBlade/BillingAccounts)

blade used by most subscription types. **To check the remaining budget of Azure Classroom subscriptions, please visit Azure Education Hub (https://portal.azure.com/#blade/Microsoft_Azure_Education/EducationMenuBlade/overview).**

Information

Grade Distribution

Social Network - Login Task	20 points
Social Network - Neo4j Task	20 points
Social Network - MongoDB Task	15 points
Social Network - Timeline Task	20 points
Social Network - Timeline Cache Task	10 points
Reflection and Discussion	5 points
Manual Grading	10 points

Introduction

In the Cloud Storage - SQL and NoSQL (Part 1) module, you provisioned instances and set up various database systems. Many of the tasks involved in designing, configuring and managing databases are non-trivial. In fact, in the traditional IT model where software and resources are owned by individual organizations, multiple full-time personnel are hired simply to configure, monitor and maintain database systems, known as database administrators or DBAs.

There are multiple tasks involved in keeping a database within operational parameters for an application. The database applications must be monitored periodically to ensure that query performance is within the expected requirement of the application. Data in the database must be periodically backed-up or snapshotted in order to be able to recover in case of any failure.

An important value-added service that many cloud services offer is the ability to outsource the responsibility for running and maintaining database systems from the user to the cloud provider. In the cloud computing industry, this type of database services has been typically referred to as Database-as-a-Service (DBaaS).

Social Networking Application Fundamentals

Social networking applications such as Facebook, Twitter, and Instagram provide people with a method to virtually connect with others, share their life events with others, and see what others are up to. Today, billions of people across the world use these applications. As you can imagine, these services require complex and well-designed back-end systems to handle the various types of user data, provide consistently fast performance, low latency, and use cutting-edge analytics to derive information that is valuable to these companies and

advertisers, in near-real-time. In this project, we will walk you through a process to help you build a social network that extends Reddit.com, a social network where people can view friends' recent activities (comments) on Reddit.com!

Before building our social networking website, we need to first be aware of the heterogeneity of data, functionality, and requirements that exist in a typical social networking website. Let's take Facebook as an example, each post you make may contain text, links, images, videos, and/or any combination of them. In addition, news feed and new connections are generated every second. In Facebook, we need to store user relation graphs, users' information, messages, timeline activities, photos, videos, etc. It also supports many different features: viewing timeline, messaging friends, uploading videos, and etc. Those activities are conducted on a very large scale. Consider a timeline view: a single HTTP request to retrieve and display a page in a social network may trigger a large number of subsequent requests and database actions at the back-end. In most cases, it does not make sense to store all the data in the same manner. Therefore, applications usually require developers to be able to deal with a number of different databases, making use of their respective advantages to achieve better efficiency and performance. The heterogeneity of the data we have, the functions we enable, and the requirements we need to satisfy cannot be accomplished with a single type of database and simple queries. See the following tables for the scale of data on Facebook as well as its recent growth:

Data Type	Total Size	Technology	Query Latency
Facebook Graph	single digit petabytes	MySQL and TAO	< few milliseconds
Facebook Messages and Time Series Data	tens of petabytes	HBase and HDFS	< 200 milliseconds
Facebook Photos	high tens of petabytes	Haystack	< 250 milliseconds
Data Warehouse	hundreds of petabytes	Hive, HDFS, and Hadoop	< 1 minute

	Facebook Users	Queries/Day	Incoming Data/Day	Nodes in WareHouse	Size (Total)
Growth in past 4 yr	14X	60X	250X	260X	2500X
		2010	2012	2015	2018
Growth in Instagram Users		25,000	30 million	400 million	1 billion

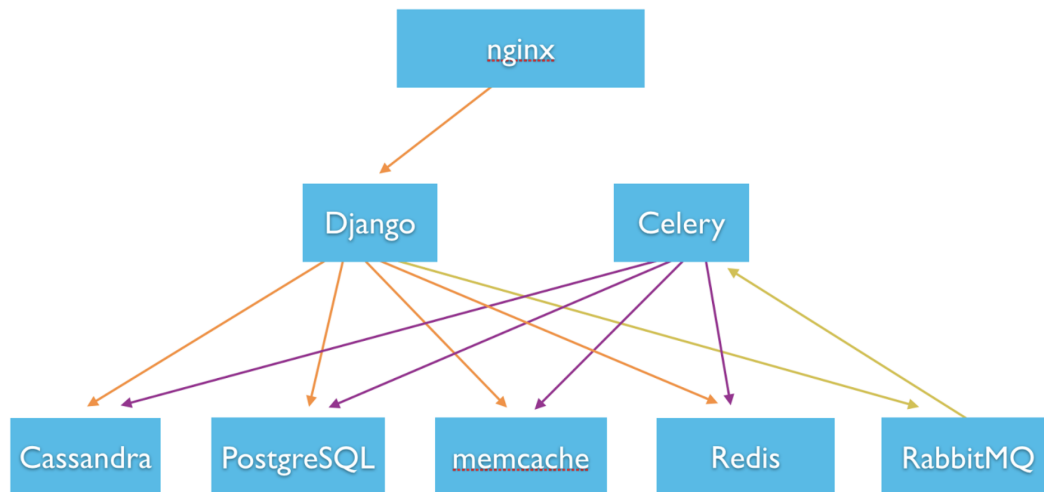


Figure 1.1: Instagram Stack Diagram

From the above tables and diagram, we can clearly see the large growth rate, the large scale of data, the various data types, and the different backend datastore systems needed within a complex social networking website like Facebook and Instagram. As a result, even a single request made by the front-end of a social network application will trigger requests to many internal back-end services to retrieve fresh information, serve valid advertisements, etc. Here is an example of a typical fan-out graph for a small user-generated request to Facebook. Every leaf node (lowest tier) in the graph represents a single computation, whereas non-leaf nodes indicate aggregation and additional computation. Often a single database cannot handle all of these queries, leading to distributed back-ends that employ replication and sharding to serve data effectively.

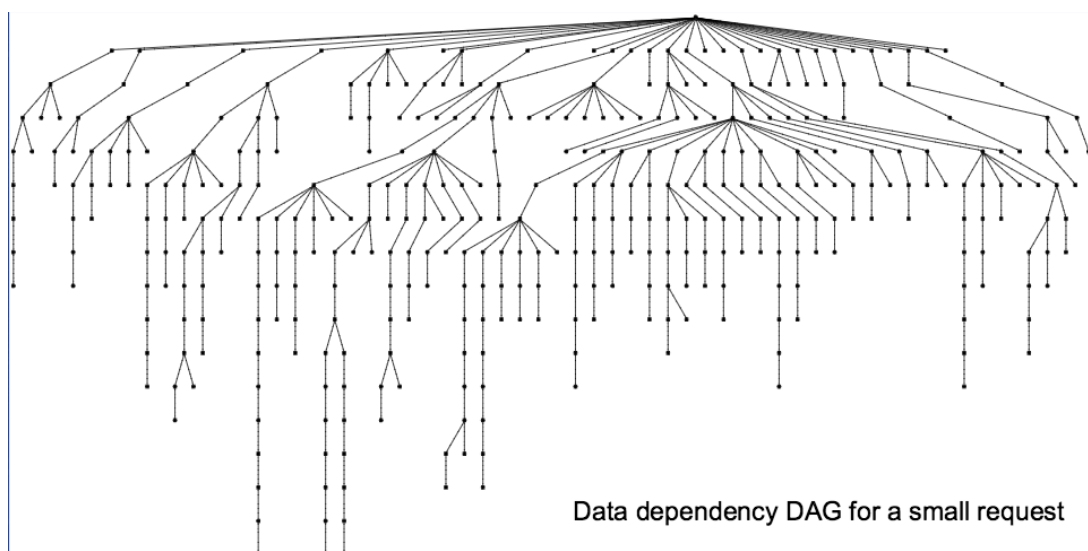


Figure 1.2: A sample fan-out tree at Facebook

Database-as-a-Service (DBaaS)

Database-as-a-Service is one or more managed database services offered by cloud providers. In DBaaS, cloud providers offer packaged database services that users can configure and deploy while the cloud provider performs the traditional database administration and

maintenance functions. The database can seamlessly scale based on load. They are maintained, upgraded, and backed-up by the cloud provider. DBaaS also provides better monitoring and failure resiliency than typical off-the-shelf database installations, and can hence improve application availability and manageability. DBaaS can typically handle many of the routine tasks that typically require full-time DBAs.

In this module, we will introduce you to Azure's Database-as-a-Service (DBaaS) offerings. DBaaS services enable users to rapidly configure and deploy a database system without having to manually launch instances and configure database software. You will learn about a fully-managed relational database service, Azure Database for MySQL server.

Neo4j - A Graph Database

Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend. Neo4j is referred to as a native graph database because it efficiently implements the property graph model down to the storage level. The data is stored exactly as you draw a graph on a whiteboard, and the database uses pointers to navigate and traverse the graph. As a result, the execution time for each query is proportional only to the size of the portion of the graph traversed, rather than the size of the overall graph.

MongoDB - A Document Store

In addition, you will deploy and use another well-known cross-platform NoSQL database. MongoDB is a Document Oriented Database, also referred to as a Document Store. Document stores differ significantly from traditional relational databases in that they are weakly typed while RDBMS (<https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm>) are strongly typed. Document Stores are schema-free; most columns or fields are optional and can be added or removed at any time. A Document Store is a subtype of key-value stores, but while the value content may be opaque to a key-value store, fields in a document store can be indexed. In addition, queries to a typical Document Store such as MongoDB are designed to offer a richer experience with current programming techniques. For all these reasons, MongoDB has become a seamless back-end for the MEAN stack (<http://mean.io>) and is also used by many companies across the globe to develop web applications.

Architectural Overview

In this module, you will build a social network application that retrieves different types of media from different back-end systems. Generally speaking, a modern social networking site, at least, needs the following three components: user profiles, user activities, and a big dataset analytics system.

1. User Profiles
 - a. User Authentication System (such as a Single-Sign-On or SSO)
 - b. User Info / Profile
 - c. Action Log
 - d. Social Graph of the User: follower, followee, family, etc.
2. User Activity System - All user-generated media
3. Big Data Analytics System
 - a. Search System
 - b. User Behaviour Analysis

Scenario: Build Your Own Social Network Website

Social Network

Scenario

A few weeks after you started to work on the database systems at Carnegie SoShall, the Team Leader noticed your very particular set of skills in implementing scalable storage systems and decided to assign you a standalone task that would explore new opportunities for your enterprise. Your previous projects equipped you with a handy toolkit for data analytics, distributed file systems, and web servers.

You are assigned a project to build a social networking website that uses Reddit.com data to allow people to view each other's comments in a timeline fashion. This offers a view for users to notice what their friends are interested in and up to these days. Your Team leader asks you to build a prototype using a limited budget. You are also required to report to the Project Manager within 1 week's time.

You have chosen RedditBook as the name of your application. By reviewing how WhatsApp, Instagram, and similar teams built successful products by drilling through exactly one focused feature, you will confine your scope of development to the theme of a social network only for comments on Reddit.com. User authentication and social networking are compulsory elements and, beyond that, your core feature is to let users share comments and comment reviews. After brainstorming, you concluded that even building such a single application would end up with numerous types of data. As a matter of fact, there is no such thing as an omnipotent database that would function perfectly in the backend. Therefore, you explored multiple data storage options and realized that a combination of three data storage systems would satisfy your needs, as long as you come up with a proper schema design for each of them. Since you have no extra budget to recruit teammates to help you build up a scalable infrastructure, you decided to work on your own and build your first prototype with cloud resources.

Your friends at Sail() Inc. helped you do the heavy weight lifting by building a simple web front-end, so you are only responsible for implementing the backend.

Project Architecture

In the next set of tasks, you will use the simple web service architecture shown in the following figure.

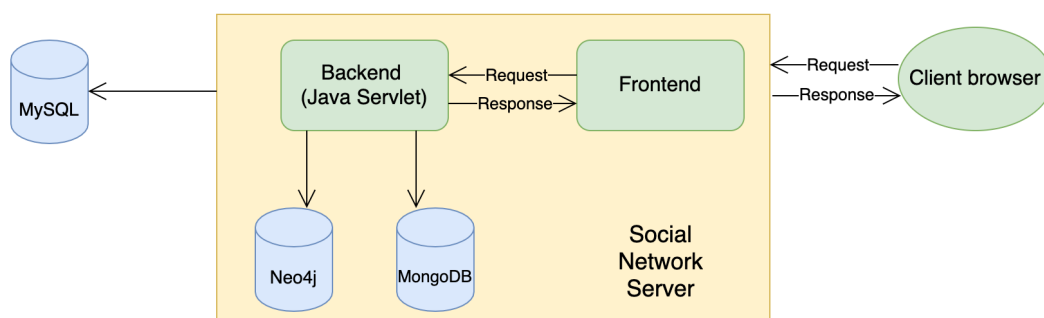


Figure 3: Project Architecture Overview

You will need to load three different datasets into three different kinds of databases (SQL, Neo4j, MongoDB) and query the heterogeneous database system for correct answers which will be returned in JSON format as a response to the HTTP requests.

You need to design your own schema for all the databases. Your backend implementations should be able to deal with four kinds of queries simultaneously. The frontend will show you what your social network looks like and will prove useful for your tasks.

Task Setup

Provision VM instances using Terraform

To complete this project, you will work on a `Standard_B2ms` VM instance using the VHD file provided by us.

1. **You should use Azure Cloud Shell to launch the student VM.** If you haven't tried it before, please refer to this page (<https://docs.microsoft.com/en-us/azure/cloud-shell/overview>) to learn about it. You will be asked to choose between Bash and Powershell when you launch cloud shell for the first time, and you should choose 'bash' for this project.
2. Then, make sure you're authenticated with Azure. Make use of the Azure Cloud Shell and log in with it.

```
az login
```

3. **Warning: As you may have multiple subscriptions, make sure you are using the Azure subscription for this course.**

```
az account set --subscription $SUBSCRIPTION_ID
```

4. Run the following terraform commands to set up the student instance.

```
wget https://clouddeveloper.blob.core.windows.net/assets/social-network/project/build-azure-vm.tgz
tar -xvzf build-azure-vm.tgz

# You will need to set the password needed to ssh into your VM
# The password must be at least 10 characters in length and must contain
# at least one digit, one non-alphanumeric character, and one upper or lower case letter.

export TF_VAR_password=<input_password>

terraform init

terraform apply
```

5. Obtain the instance public IP from the output instructions of Terraform.

Setting up the Frontend

1. Go to `http://INSTANCE_PUBLIC_IP` and choose **P3. Heterogeneous Storage on the Cloud (Part 2) (Frontend)** to register your frontend boilerplate code. This may take several minutes.

2. Once you see Frontend code for the Social Networks has been deployed , SSH in the VM as the `clouduser` as before.

3. Run the following two commands to start the frontend server:

```
cd ~/SN2
npm start
# once you see the following logs, the server is ready
#
# > SN2@0.0.0 start /home/clouduser/SN2
# > node ./bin/www
# The server will run as a foreground process.
# You can press CTRL+C to kill a foreground process to stop the server.
```

4. Go to `http://INSTANCE_PUBLIC_IP:3000` in your browser, if you can see the following page, your frontend is correctly set up.

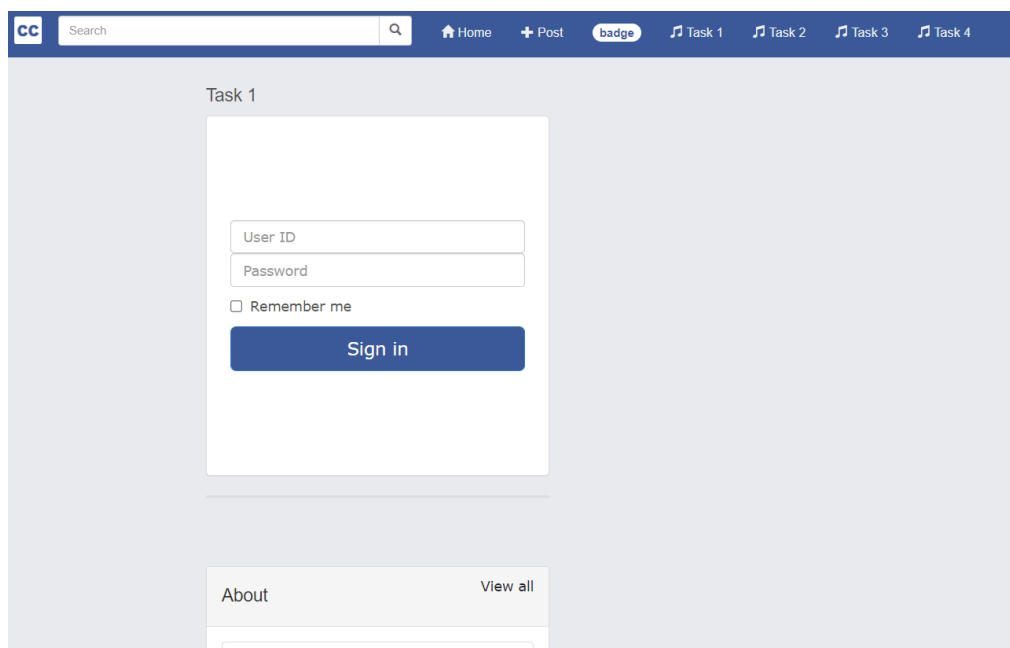


Figure 4: View of the basic frontend page.

Setting up the Backend

Each file comes with boilerplate code necessary for grading purposes. You will find a file called `MiniSite.java` which launches the server for this project. All the deployments of Servlets are written in this file and you should **NOT** change anything in `MiniSite.java`.

1. Go to `http://INSTANCE_PUBLIC_IP` and choose **P3. Heterogeneous Storage on the Cloud (Part 2) (Backend)** to register your backend boilerplate code. This may take several minutes.
2. Once you see Backend code for the Social Networks has been deployed , SSH in the VM as the `clouduser` as before.
3. There are several `.java` files in the `~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory. Your task is to complete:

```
* ProfileServlet.java (Task 1)
* FollowerServlet.java (Task 2)
* HomepageServlet.java (Task 3)
* TimelineServlet.java (Task 4)
* TimelineServletWithCache.java (Task 5)
```

4. You can also find the executable submitter and references files under the same directory.

5. Go to the `~/social_network_backend` directory.

```
cd ~/social_network_backend
```

6. Build the project and launch the server with the following command:

```
mvn clean package exec:java -Dmaven.test.skip=true

# The test cases need to be skipped for now as your implementation
# is tested by our test cases for correctness.

# once you see the following logs, the server is ready
<timestamp> INFO  xnio:104 - XNIO version 3.3.8.Final
<timestamp> INFO  nio:55 - XNIO NIO Implementation Version 3.3.8.Final
# The server will run as a foreground process.
# You can press CTRL+C to kill a foreground process to stop the server.
```

7. To connect the backend to your frontend, update the file at `~/SN2/routes/backend.js` in your frontend code with the instance's public IP. **Note: You should put the IP address only. Do not prepend "http://".**

8. Go to `http://INSTANCE_PUBLIC_IP:3000` in your browser. Don't input anything in the Login text boxes and click "Sign In", and then verify that you see the default response

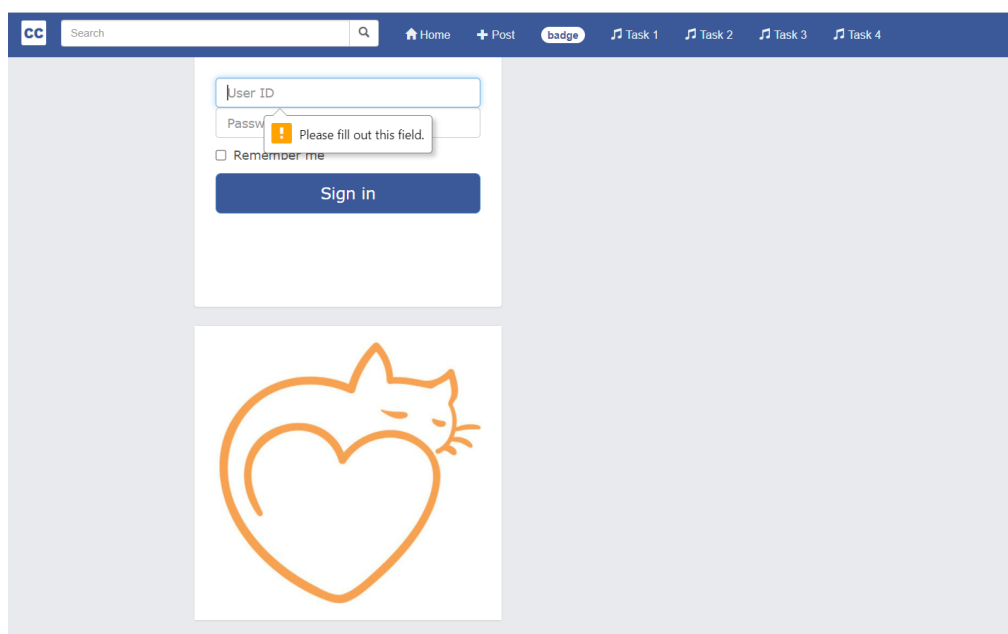


Figure 5: View of the default response.

9. Now that you are done setting up, it is time to prepare the data sources for your own social networking site!

Information

Notes on Java Servlet

1. Each of the `xxxServlet.java` files is dedicated to one specific task.
2. Within each Servlet file, there is a method called `doGet` which you need to implement to complete this project.
3. Inside each `doGet` method, you will need to parse the request, query your backend database(s), parse the result, and send a response back to the client.
4. A basic example of how to process a request and a response are shown in the Servlet.
5. You may want to initialize your database connection in the constructor of each Servlet.

Information

Notes on the JSON data format

1. The starter code uses the Google GSON package to deal with JSON. Feel free to use any other JSON libraries you like such as Jackson, but be aware that the dataset supplied may contain non-ASCII characters and you should handle them gracefully.
2. When you write database queries, do not assume that the data is clean. Always consider possible corner cases. **For example, it is possible that when you search for a follower of a user, the user actually does not have any followers.**

Task 1: Implementing Basic Login with SQL

Social Network Login Task

Database-as-a-Service (DBaaS)

The term “Database-as-a-Service” (DBaaS) refers to software that enables users to set up, operate and scale databases using a common set of abstractions (primitives), without having to either know or care about the exact implementations of those abstractions for the specific database. The administrative capabilities offered by the service include scaling, securing, monitoring, tuning, and upgrade of the database and the underlying technologies, which are managed by the cloud vendor. These administrative tasks are automated, allowing users to focus on optimizing applications that use database resources.

Azure Database for MySQL

As per Azure documentation, Azure Database for MySQL is a relational database service in the Microsoft cloud based on the MySQL Community Edition (available under the GPLv2 license) database engine, versions 5.6, 5.7, and 8.0.

Authentication with RDBMS

Until your social networking frontend is connected to the backend that fetches data from the database, the frontend page simply displays a default image of a cat.

To bring the social networking app to life, the first thing to do is to set up a login web service (in our case, our basic login Profile Servlet) that accepts a Username/Password pair and returns the user's name and profile picture.

As authentication involves highly structured data and is normally done with a relational database, **you will use MySQL in this scenario**. When a user logs into the website with `<username>` and `<password>`, the backend server looks up the pair to see if there is a match in the SQL database table. If `<username>` and `<password>` are valid, the corresponding user's `<name>` and `<profile_picture>` are displayed on the page. Otherwise, the word "Unauthorized" will be displayed in place of a username.

A Quick Introduction to Mockito for TDD

Information

For your scalable social networking application with heterogeneous backends, development can be constrained by the fact that debugging and testing the application may require frequent access to database resources and user data.

However, the direct use of user-generated data for testing and debugging your application is a risky process involving privacy and security concerns. Furthermore, making real database connections to production databases is a costly and slow process for test-driven development.

There are primarily two approaches to implementing a testing framework in such a scenario:

1. Provision a database with dummy data resembling actual user data, as part of the test suite.
2. Mock database objects and effectively spoof their methods, replacing the actual responses with the expected static responses.

It is apparent that while the first approach may be better at identifying specific database issues, for a well-known database like MySQL you may opt for the second approach to further speed up your development.

Mockito (<https://site.mockito.org/>) is an open-source testing framework that allows the creation of test double objects (mock objects). It is used to mock interfaces so that the specific functionality of an application can be tested without using real resources such as databases, expensive API calls, etc.

We have provided you with two test cases written in Mockito that test your Profile Service for Task 1. A correct implementation will pass both the test cases, however, the test cases only test part of your implementation!

Now that you have a good understanding of MySQL and testing MySQL backends, let's move forward to implement your ProfileServlet for Task 1! To begin, you need to provision an Azure-managed MySQL database and store the user information in it.

Setting up Azure Database for Managed SQL

1. Provision a managed MySQL instance by downloading and deploying the ARM template for the Azure database for MySQL

```
mkdir azure-sql && cd azure-sql
wget https://clouddeveloper.blob.core.windows.net/assets/social-network/project/azure-sql.tgz
tar -xvzf azure-sql.tgz

# in parameters.json, set the value of the following parameter:
# `serverName.value`
# This will be used to identify your Azure Server

# note that the value should be the andrew_id username
(for e.g. xyz@andrew.cmu.edu; username is xyz)
az deployment group create --name "sql-instance" --resource-group "social-network-rg" --template-file "template.json" --parameters "parameters.json"

# You will be prompted with the following where you should type your password.
# The password must be at least 10 characters in length and must contain
# at least one digit, one non-alphanumeric character, and one upper or lower case letter.
Please provide securestring value for 'administratorLoginPassword' (? for help):
```

2. Before you use MySQL client to connect to the Azure database for MySQL instance, you must whitelist the IP address of the client machine.

- Go to Azure Portal -> Azure Database for MySQL server -> Select the database resource
- Under Settings -> Connection Security
- Set Allow access to Azure services to Yes
- Add the Public IP of your student VM instance as the Start IP and the End IP
- Click Save

3. SSH into the workspace VM as `clouduser` with the password you set when you provision the VM.

```
ssh clouduser@"$INSTANCE_PUBLIC_IP"
```

4. Validate the connection by logging on to the MySQL instance. You need to specify the host when connecting to a remote server. You can visit Azure Portal -> Azure Database for MySQL server -> Select the database you created to view the Server Name and Server Admin Login Name fields. For your convenience, you might want to set the server name, admin name, and the password you set above as environment variables.

```
export MYSQL_HOST=<db_server_name>
export MYSQL_NAME=<admin_username>
export MYSQL_PWD=<password>
```

5. Then you can log in MySQL using the following commands:

```
mysql -h ${MYSQL_HOST} -u ${MYSQL_NAME} --password=${MYSQL_PWD}
# Welcome to the MySQL monitor.  Commands end with ; or \g.
# Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
#
# Oracle is a registered trademark of Oracle Corporation and/or its
# affiliates. Other names may be trademarks of their respective
# owners.
#
# Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

# exit the shell
mysql> exit
```

6. Create a SQL script titled `load_users.sql` **in the `social_network_backend/` folder** and paste the following code into the file

```
## Step 1 create the database
drop database if exists reddit_db;
create database reddit_db;
use reddit_db;

## Step 2 create the users table
drop table if exists `users`;
create table `users` (
  `username` varchar(140) default null,
  `pwd` varchar(140) default null,
  `profile_photo_url` varchar(140) default null
);

## Step 3 create an index
create index user_index ON users(username);

## Step 4 load data to the users table
load data local infile 'users.csv' into table users columns terminated by
',' LINES TERMINATED BY '\n';
```

7. Download `users.csv` to your VM and load the data to the database. To load data (<http://dev.mysql.com/doc/refman/5.7/en/load-data.html>) from the backend, the `--local-infile` flag should be present in the login bash command. You may use `less users.csv` to explore the CSV file and inspect the data. Note that the load operation takes 10-15 minutes to complete.

```
wget https://clouddeveloper.blob.core.windows.net/datasets/social-network/project/users.csv

# load the data file into the users table
mysql -h ${MYSQL_HOST} -u ${MYSQL_NAME} --password=${MYSQL_PWD} --local-in
file=1 < load_users.sql
```

8. Access your data from a MySQL shell to validate the data.

```
mysql -h ${MYSQL_HOST} -u ${MYSQL_NAME} --password=${MYSQL_PWD}
# (in mysql shell)
USE reddit_db;
SHOW TABLES;
SELECT * FROM users LIMIT 10;
SELECT profile_photo_url FROM users WHERE username = 'zzzzzzzlaf' AND pwd
= '9685b4fd5fd93487c92167af8ac837dd';
```

Task Implementation

Information

Each of your databases will be accessed in more than one task. Therefore, it is recommended that you write modularized and reusable programs. We recommend that you finish reading the whole writeup before you start writing code. Aspire to achieve a modularized and reusable design.

1. After confirming that your database does contain your data, implement `ProfileServlet.java` to solve this task. You may not make changes to the structure of the existing methods in this class, however, you may create new ones as long as they do not replace existing methods. This is to ensure that the test cases run for your implementation.
2. The MySQL Connector/J dependency has already been provided in the `pom.xml` file of the Maven project on the backend deployment. You do not need to deal with library dependency.
3. To figure out how to interact with the MySQL database, read more about JDBC here (<http://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm>) or refer to your solution in Task 1.
4. The query in this task simulates the login process of a user and tests whether your backend system is functioning as expected. Your web service will receive a `<Username,Password>` pair, and you need to query your backend database to see if the pair is valid. You should construct your response accordingly:

1. If YES, send back the user's Name and Profile Image URL.
2. If NOT, set Name as "Unauthorized" and Profile Image URL as "#".

5. Be sure to set the environment variables before launching the backend:


```
export MYSQL_HOST=
export MYSQL_NAME=
export MYSQL_PWD=
```

6. In order to save you some time, here is a sample code to create the JSON response:

```
JsonObject result = new JsonObject();
result.addProperty("name", name);
result.addProperty("profile", profile_image_url);
response.setContentType("text/html; charset=UTF-8");
response.setCharacterEncoding("UTF-8");
PrintWriter writer = response.getWriter();
writer.print(result.toString());
writer.close();
```

Sample Backend Request for Task 1

The format that your backend will receive is as follows:

Request Format:

```
GET /task1?id=[UserName]&pwd=[Password]
```

Response Format (JSON Format):

```
{"name":"my_name", "profile":"profile_image_url"}
```

Sample Request and Response

A sample request to your web service will look like this:

```
http://INSTANCE_PUBLIC_IP:8080/MiniSite/task1?id=zzzzzzlaf&pwd=9685b4fd5fd93487c92167af8ac837dd
```

The expected response that your web service should construct and return a JSON object looks like this (the order of the pairs does not matter):

```
{"profile":"https://farm1.staticflickr.com/376/31987883501_dab802bef4_m.jpg", "name":"zzzzzzlaf"}
```

A JSON object is an **unordered** set of name/value pairs.

Test Task 1 with the Frontend

After implementing the servlet and setting the environment variables, start the Java web server first by running:

```
cd /home/clouduser/social_network_backend
mvn clean package exec:java -Dmaven.test.skip=true
```

The test cases need to be skipped for now as your implementation is tested by our test cases for correctness.

With your front-end server setup, navigate to `http://INSTANCE_PUBLIC_IP:3000`

Click the Task1 button on the navigation bar. If you have connected backend to the frontend, you will be able to retrieve a username and a profile image when you enter a correct login username and password.

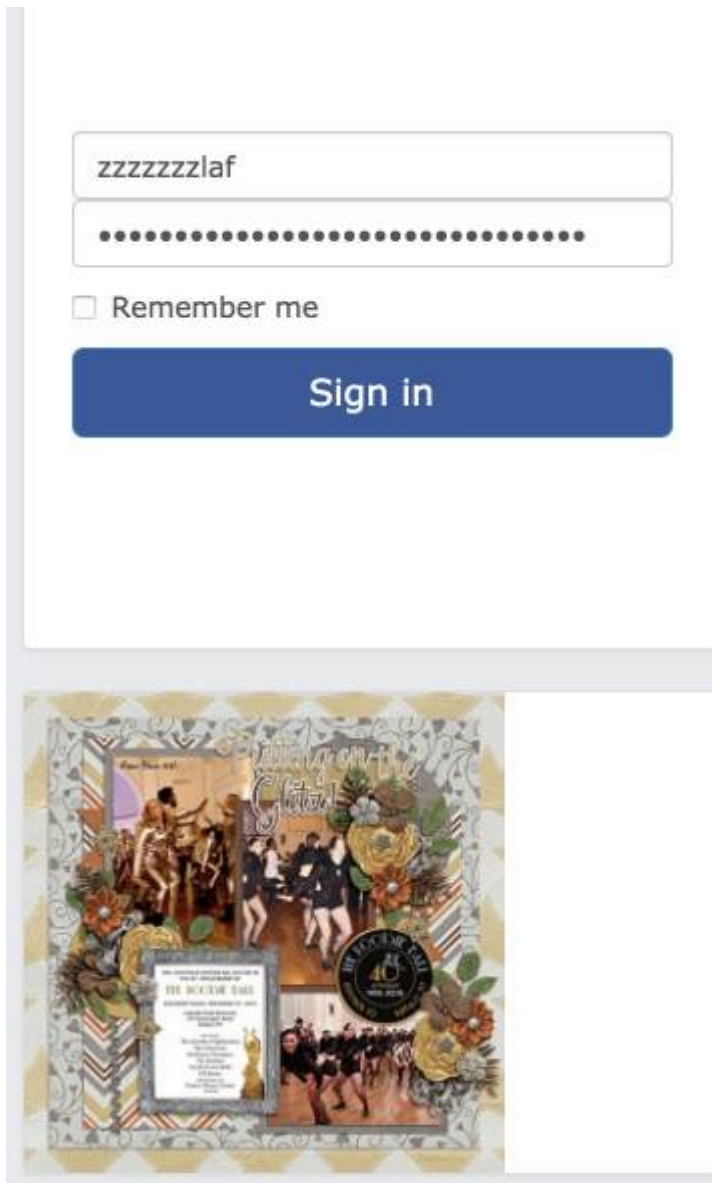


Figure 5: View of the login page.

You can also directly send requests using the **curl** command, this might be more efficient for you to test your code. An example for Task 1 can be:

```
BACKEND_ADDRESS="<INSTANCE_PUBLIC_IP>"
curl "http://${BACKEND_ADDRESS}:8080/MiniSite/task1?id=zzzzzzzlaf&pwd=9685b4fd5fd93487c92167af8ac837dd"
```

If you get an unexpected token d error from the frontend, it is very likely that you did not connect to your database successfully. Please double-check whether your code can get your username, password, and URL successfully.

What to Submit

1. Make sure the following files are under the

~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory:

- ProfileServlet.java
- FollowerServlet.java
- HomepageServlet.java

- TimelineServlet.java
 - Cache.java
 - TimelineWithCacheServlet.java
 - MiniSite.java and any other helper source code files
 - references file
 - submitter
2. It is good practice to always include comments to **explain complicated logic and functions** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.

How to Submit

1. Copy your reference file and Java code to
`~/social_network_backend/src/main/java/edu/cmu/cc/minisite/`.
2. Keep your backend running in one SSH session.
3. To run the submitter, launch a second terminal and create a second SSH connection. Run the submitter under the
`~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory. You should make sure that your backend is running and can respond to queries.
4. You do not need to finish all the tasks before receiving a score. You will most likely submit after each task.
5. Execute the submitter under the
`~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory.

```
export HISTIGNORE="export*" # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME="your_submission_username"
export SUBMISSION_PASSWORD="your_submission_pwd"
./submitter -t task1
```

Information

This project consists of multiple environment variables to be exported on the terminal. You may want to use tools like `tmux` or `screen` to prevent session timeout.

Information

If you see the following warning when submitting a task: `*WARNING : deprecated key derivation used`. Using `-iter` or `-pbkdf2` would be better.

Please ignore this warning for now, as we are currently fixing it.

Troubleshooting

If you get a `"-"` as a score for your submission, it is because your web service took too long to respond to our autograder. Try to test your backend with the `"curl"` command or by using the frontend application and make sure that your backend responds in a reasonable timeframe.

The submitter will send a grading request to the Auto Grading Service and the amount of time taken to finish the grading depends on the performance of your backend web server. We don't grade on performance but there is a timeout on each request. So if you receive feedback such as "Grading timeout", try to make your query more efficient.

If you face an error such as npm not installed, please re-try launching the project again, so that the required libraries are installed.

If the grader reports a 5xx Internal Server Message, check whether the backend server has crashed. There could be multiple reasons why there was an error, but the callstack should provide enough information to pinpoint to the issue. We encourage you to add as many try-catch blocks as required for easier identification of errors.

Task 2: Storing Social Graph using Neo4j

Social Network Neo4j Task

Before you move on, we require you to complete the Neo4j primer.

Scenario

In this task, you will need to build a database system to store the social graph.

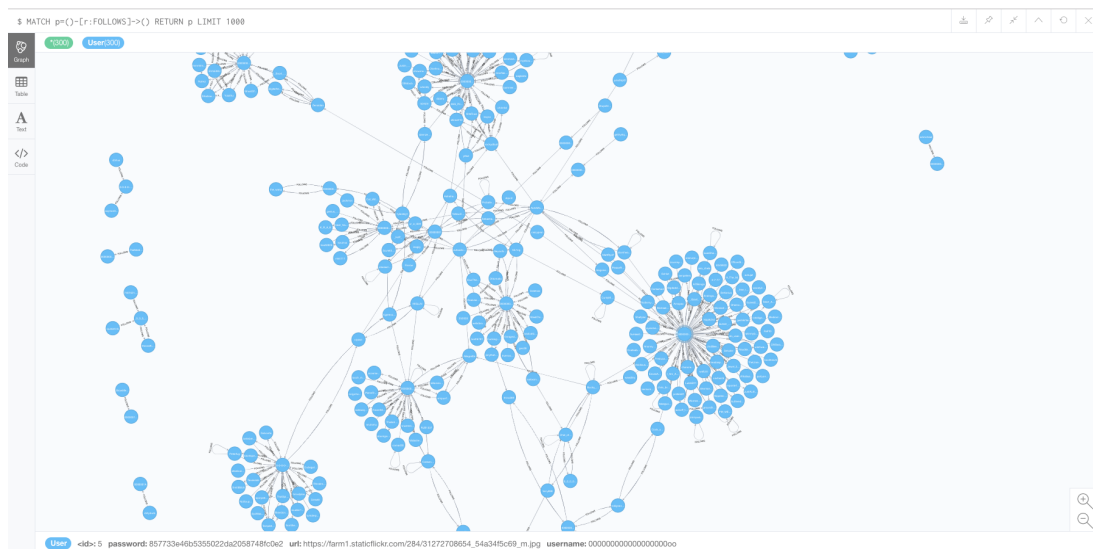


Figure 6: A sample of the social graph

The raw data is in `<Followee, Follower>` pairs. Therefore, you need to choose a data model to store the social graph so that when we write a query for a user, your system can provide the list of people that follow this user.

Data model examples to store graphs

1. Adjacency Matrix

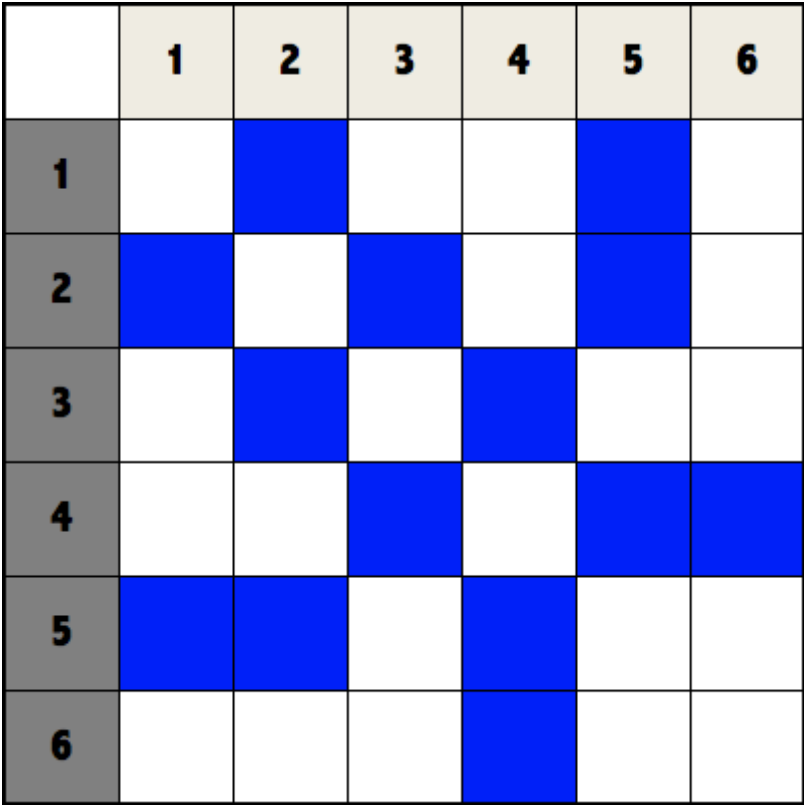


Figure 7: An example of adjacency matrix

2. Adjacency List

1	2
1	5
2	1
2	3
2	5
3	2
3	4
4	3
4	5
4	6
5	1
5	2
5	4
6	4

Figure 8: An example of an adjacency list

3. Native Nodes and Edges

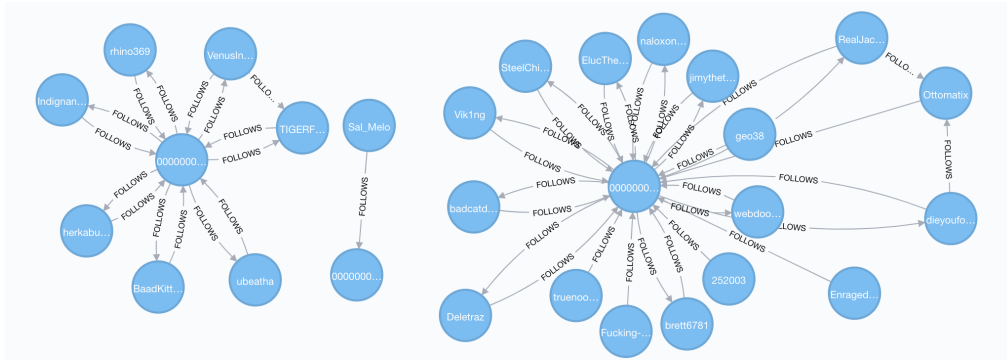


Figure 9: An example of native nodes and edges

Bulk Data Import CSV into Neo4j

Now you should have learned the basics of the Neo4j paradigm and Cypher as well as the comparison between SQL and Cypher. You are now prepared to import and query the real-world dataset in the project using Neo4j. Neo4j database has already been set up on the student VM.

To import CSV data, one approach is to use the Cypher command `LOAD CSV`. However, as with any query that updates the graph, it will run in a transaction in Neo4j to achieve data integrity. Hence, `LOAD CSV` is suitable for small-sized data.

To do batch imports of large amounts of data into a new Neo4j database from CSV files, `neo4j-admin import` is the best approach that bypasses the overhead of the transactional layer and utilizes as much memory, disk I/O, and computing power as possible.

Note that this tool can only be used to load data into a previously unused database.

Run the following commands to download and import the CSV data into Neo4j. **Warning:** **Make sure that you run the commands in order. Skipping any steps or running the commands in the wrong order can lead to subtle errors.**

```

# connect to the student VM as the user `clouduser`
ssh clouduser@"$INSTANCE_PUBLIC_IP"

# download the csv files
wget https://clouddeveloper.blob.core.windows.net/datasets/social-network/project/users.csv
wget https://clouddeveloper.blob.core.windows.net/datasets/social-network/project/links.csv

# stop the neo4j server first
sudo service neo4j stop

# purge the existing default database named as graph.db
# as the data import tool can only import to a new database
sudo rm -rf /var/lib/neo4j/data/databases/graph.db

# define the column header for users.csv and links.csv

# username is a unique identifier
echo 'username:ID,password,url' > users_header.csv

# be careful of the direction here
# the schema of users.csv is "[Followee, Follower]"
# and "Follower" follows "Followee"
echo ':END_ID,:START_ID' > links_header.csv

# import the dataset as the root user; the graph.db will be owned by the root user
# and cannot be accessed by neo4j until you run the next `chown` command
sudo neo4j-admin import --database=graph.db --nodes:User "users_header.csv,users.csv" --relationships:FOLLOWS "links_header.csv,links.csv"
# once the import is finished, you can see a report as follows:
# ...
# IMPORT DONE in 4m 1s 303ms.
# Imported:
#   2611448 nodes
#   16024501 relationships
#   7834340 properties
# Peak memory usage: 1.06 GB

# change the ownership of the database to neo4j
sudo chown -R neo4j:adm /var/lib/neo4j/data/databases/graph.db

# do not forget to start the neo4j service, note that this can take 1-2 minutes
sudo service neo4j start

# run the following command, and make sure you can see the log "Started"
sudo service neo4j status
# the sample log that indicates the service has started
Jul 31 06:03:05 CloudComputingProjectVM neo4j[68909]: 2021-07-31 06:03:05.411+00
00 INFO  ===== Neo4j 3.5.14 =====
Jul 31 06:03:05 CloudComputingProjectVM neo4j[68909]: 2021-07-31 06:03:05.420+00
00 INFO  Starting...
Jul 31 06:03:07 CloudComputingProjectVM neo4j[68909]: 2021-07-31 06:03:07.553+00
00 INFO  Bolt enabled on 0.0.0.0:7687.

```

```
Jul 31 06:03:09 CloudComputingProjectVM neo4j[68909]: 2021-07-31 06:03:09.038+00
00 INFO Started.
```

```
Jul 31 06:03:10 CloudComputingProjectVM neo4j[68909]: 2021-07-31 06:03:10.054+00
00 INFO Remote interface available at http://localhost:7474/
```

If you cannot see "Started" in `sudo service neo4j status` and the server gets stuck at Starting... ; it is very likely that you did not run the commands in order and the `graph.db` is not owned by `neo4j:adm`. **If you run into this error, we expect you to debug this on your own by revisiting and rerunning the commands instead of posting on Piazza.**

Neo4j Browser User Interface

Navigate to `http://<INSTANCE_PUBLIC_IP>:7474`.

Log in with the username `neo4j` and the default password `neo4j`. You will be prompted to change the password immediately. Choose your own Neo4j password and you will need it later.

Figure 15: Get prompted to change the default password

In the Neo4j Browser User Interface, run the following query to validate the data:

```
MATCH (u) RETURN COUNT(u);
```

Figure 16: The sample query and its result to count the nodes

With the relationships loaded, you can query the followers given a user.

```
MATCH (follower:User)-[r:FOLLOWS]->(followee:User)
WHERE followee.username = "zylo_"
RETURN followee, follower;
```

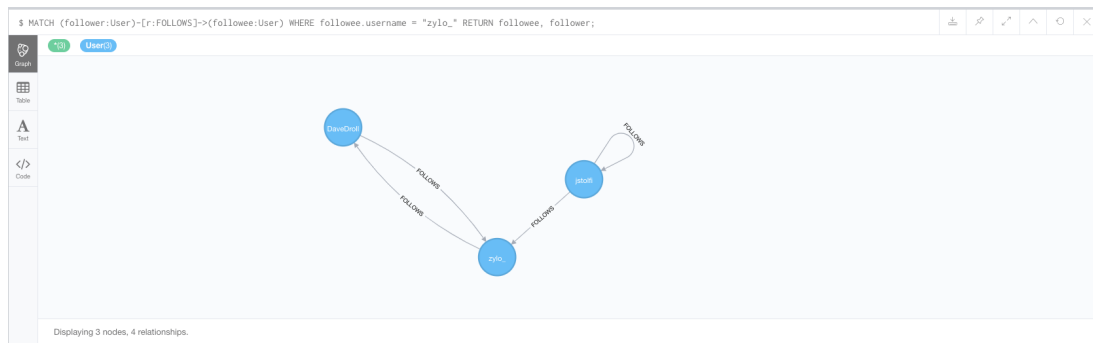



Figure 17: The sample query and its result to query the followers

You can also sample the social network.

```
MATCH p=()-[r:FOLLOWS]->() RETURN p LIMIT 1000
```

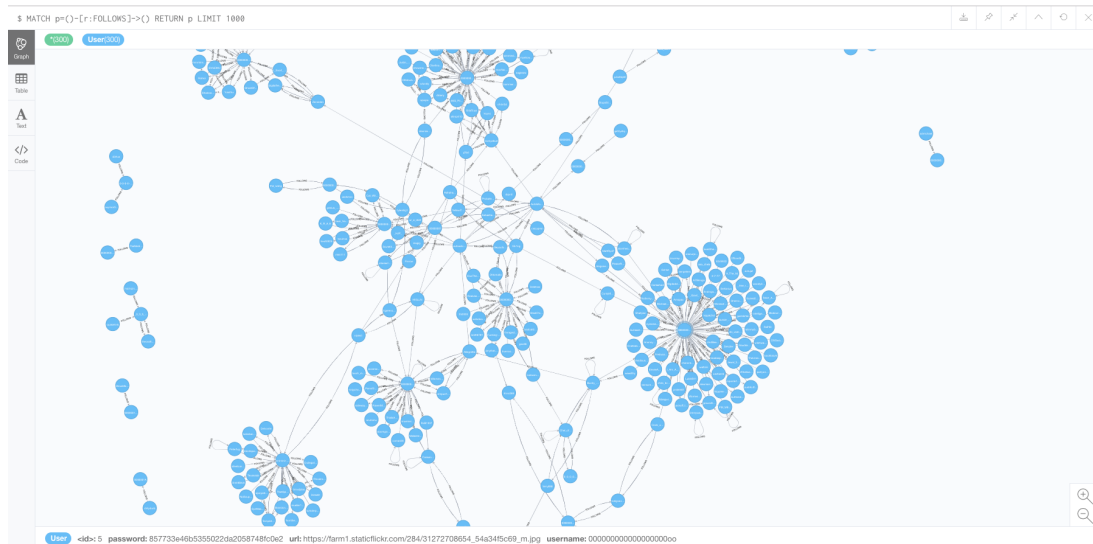


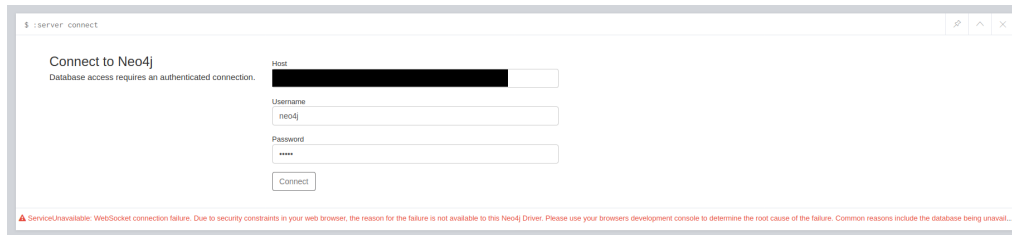
Figure 18: The sample query and its result to query a portion of the social network

Troubleshooting

1. If you encounter this error when you query Neo4j in the chrome browser, you might need to upgrade your chrome to the latest version ($\geq 80.0.3987.106$).



2. If you encounter this error when you log in to Neo4j with firefox, please first go to https://<INSTANCE_PUBLIC_IP>:7687 to accept the certificate and then go back to http://<INSTANCE_PUBLIC_IP>:7474 to log in.



3. If you are still not able to resolve the issue, please consider using other browsers like safari. You can also try to query the database using the `cypher-shell` CLI tool in your Neo4j virtual machine. Cypher-shell is packaged as `/usr/bin/cypher-shell` in your virtual machine.

Task Implementation

Information

Each of your databases will be accessed in more than one task. Therefore, it is recommended that you write modularized and reusable programs. We recommend that you finish reading the whole writeup before you start writing code. Aspire to achieve a modularized and reusable design.

1. Read the example (<https://github.com/neo4j/neo4j-java-driver>) that shows how you can connect to the Neo4j database.
2. You will implement your solution in `FollowerServlet.java`. The query in this task asks for the followers of a given user and sorts the followers **lexicographically** in **ascending** order by username. Hint: You can use the `ORDER BY` clause in Cypher to sort results, similar to SQL.
3. Your web application will receive a single username as a request parameter, and you need to send back the username and Profile Image URL of all of his/her followers.
4. If you do not find any followers of a given user, return an empty list like this: `{"followers": []}`. Note this also applies to all the tasks in this module. You should always follow the defensive programming techniques when you are building web services, and handle corner cases such as empty results.
5. Be sure to set the environment variables before launching the backend:

```
export NEO4J_HOST=
export NEO4J_NAME=
export NEO4J_PWD=
```

6. To connect using the Neo4j API, you may want to import, but are not restricted to, the following packages:

```
import org.neo4j.driver.v1.AuthTokens;
import org.neo4j.driver.v1.Driver;
import org.neo4j.driver.v1.GraphDatabase;
import org.neo4j.driver.v1.Record;
import org.neo4j.driver.v1.Session;
import org.neo4j.driver.v1.StatementResult;
```

7. Here is a Java example to sample 5 User nodes and print the usernames.

```
private final Driver driver = GraphDatabase.driver("bolt://<endpoint>:7687", AuthTokens.basic(username, password));
try (Session session = driver.session()) {
    StatementResult rs = session.run("MATCH (u:User) RETURN u.username LIMIT 5");
    while (rs.hasNext()) {
        Record record = rs.next();
        System.out.println(record.get(0).asString());
    }
}
```

8. Read more about the Neo4j Java APIs and examples here (<https://github.com/neo4j/neo4j-java-driver>).

Request Format:

```
GET /task2?id=[UserName]
```

Response Format (in JSON):

```
{"followers":[{"name":"follower_name_1", "profile":"profile_image_url_1"}, {"name":"follower_name_2", "profile":"profile_image_url_2"}, ...]}
```

Below is sample code to add a JsonObject to JsonArray (Google GSON) in order to construct the response object.

```
JsonArray followers = new JsonArray();
JsonObject follower = new JsonObject();

follower.addProperty("name", "follower_name_1");
follower.addProperty("profile", "profile_image_url");
followers.add(follower);
```

Sample Request and Response

Sample Request:

```
http://INSTANCE_PUBLIC_IP:8080/MiniSite/task2?id=zylo_
```

Sample Response:

```
{"followers":[{"profile":"https://farm9.staticflickr.com/8007/29550375982_ce265dbf12_m.jpg", "name":"DaveDroll"}, {"profile":"https://farm8.staticflickr.com/7548/15784200337_fe06762b92_m.jpg", "name":"jstolfi"}]}
```

Test Task 2 with the Frontend

After implementing the servlet and setting the environment variables, start the Java web server first by running:

```
cd /home/clouduser/social_network_backend
mvn clean package exec:java -Dmaven.test.skip=true
```

Go to the Task 2 tab, enter a username (choose one from the dataset), and hit Search and you should be able to see the following screen if your backend is implemented correctly:

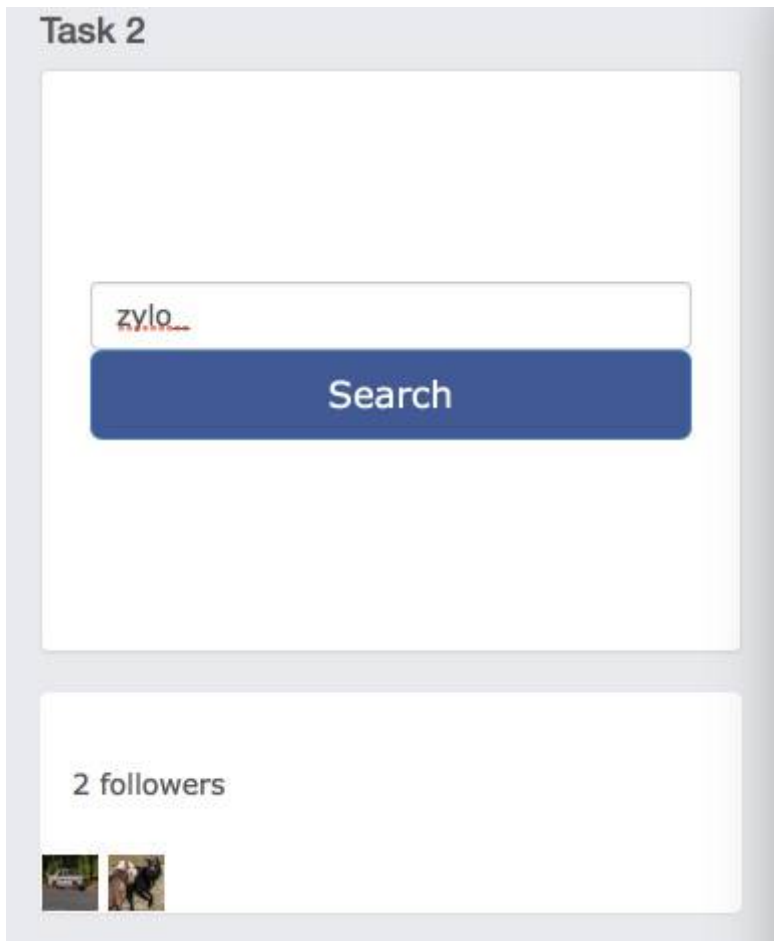


Figure 19: View of the followers

What to Submit

1. Make sure the following files are under the

~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory:

- ProfileServlet.java
- FollowerServlet.java
- HomepageServlet.java
- TimelineServlet.java
- Cache.java
- TimelineWithCacheServlet.java
- MiniSite.java and any other helper source code files
- references file
- submitter

2. It is good practice to always include comments to **explain complicated logic and functions** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.

How to Submit

1. Copy your reference file and Java code to
~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ .
2. Keep your backend running in one SSH session.
3. To run the submitter, launch a second terminal and create a second SSH connection. Run the submitter under the
~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory. You should make sure that your backend is running and can respond to queries.
4. You do not need to finish all the tasks before receiving a score. You will most likely submit after each task.
5. Execute the submitter under the
~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory.

```
export HISTIGNORE="export*" # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME="your_submission_username"
export SUBMISSION_PASSWORD="your_submission_pwd"
./submitter -t task2
```

Troubleshooting

1. If you get a "-" as a score for your submission, it is because your web service took too long to respond to our autograder. Try to test your backend with the "curl" command or by using the frontend application and make sure that your backend responds in a reasonable timeframe. The submitter will send a grading request to the Auto Grading Service and the amount of time taken to finish the grading depends on the performance of your backend web server. We don't grade on performance but there is a timeout on each request. So if you receive feedback such as "Grading timeout", try to make your query more efficient.
2. In case of the Caused by:
org.neo4j.driver.v1.exceptions.ServiceUnavailableException: Connection to the database terminated. This can happen due to network instabilities, or due to restarts of the database exception, make sure that the Neo4J service is running and you are using the correct username and password when connecting to Neo4J
3. If you are not able to reach port 3000 of the VM, please check whether port 3000 has been exposed in the Azure UI. If it is not present, you will have to expose port 3000 for the inbound network.

Task 3: Build Homepage using MongoDB

Social Network MongoDB Task

After creating a reliable user and relationship management backend, it is time to host the content served on this social network.

Key-value Store

A key-value store provides the simplest possible data model and is exactly what the name suggests: it's a storage system that stores values indexed by a key. You are limited to query by key and the values are opaque, i.e., the store doesn't know anything about them. This allows very fast read and write operations (a simple disk access) and you can think of this model as a non-volatile cache (i.e., it suits the cases well when you need fast access to long-lived data by key).

Document Store

A document-oriented database extends the previous key-value store model. Values are stored in a structured format (i.e. a document) that the database can understand. This is different from a traditional RDBMS since the format can be extended or truncated at will. For example, a document could be a blog post with the comments and the tags stored in a denormalized way. Since the data is structured, the store can do more work (like indexing fields of the document) and you're not limited to query-by-key. Document databases allow fetching the data for an entire page with a single query and are well suited for content-oriented applications (which is why big sites like Facebook or Amazon adopt them).

MongoDB

MongoDB is mainly a document-oriented database that has a lot of relational database features. It stores sparse and unstructured data in BSON format (which is the binary version of JSON) and thus supports the storage of complex data types. The document-oriented model makes MongoDB very easy to scale out in distributed servers. The query language of MongoDB has object-oriented features and also supports a sequential programming structure. It can realize almost all query functions of a relational database. Indexing is also supported by MongoDB.

BSON

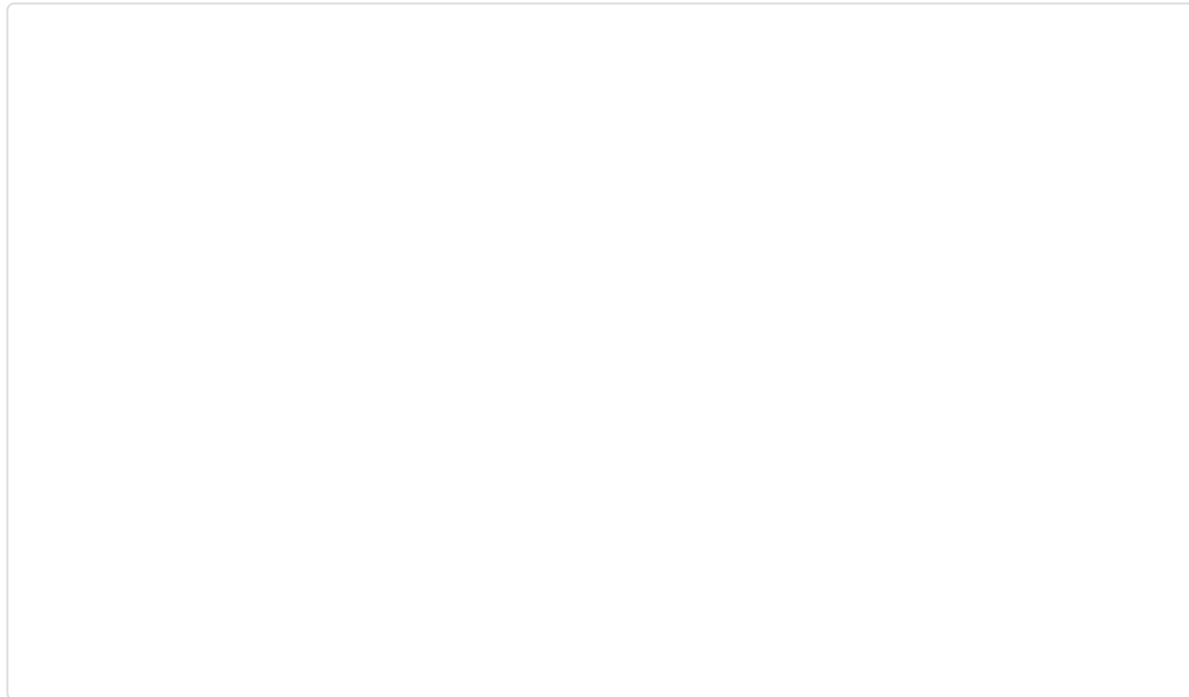
A Document is a data abstraction that is used in the interaction between the server and the client. For MongoDB, documents are encoded into BSON before being transferred from client to server or from server to client. BSON is used for the following three purposes:

1. **Space efficiency** – BSON occupies much less space than does plain JSON, even in the worst case.
2. **Mobility** – BSON sometimes introduces a small amount of overhead in the transferred data to ease transmission. For example, a size header is used in place of a terminating character to ease data modification.
3. **Performance** – BSON encoding and decoding are fast in the context of many programming languages.

Setting up

In Task 3, you will need to build a database system to store all the comments on the social network using MongoDB. The raw data is provided in JSON format where each comment is represented by a complex nested JSON Object. You need to add appropriate indexes to speed up the queries.

In this project, you are only required to use a standalone MongoDB, but the following video gives you more insight into a distributed MongoDB architecture that enhances error resiliency with replication and sharding:



Video: Introduction to MongoDB

1. Go to `http://INSTANCE_PUBLIC_IP` to register your student VM with MongoDB and choose **P3. Heterogeneous Storage on the Cloud (Part 2) (MongoDB)**. This may take several minutes. Note that it may take 1-2 minutes before you can access it.
2. Once you see MongoDB has been installed for Task 3, SSH in the VM as the `clouduser` as before.
3. Start the MongoDB service with:

```
mongod
```

4. When the Mongo daemon is running, you may use another console to log into the Mongo shell with:

```
mongo
```

Interacting with the database

Like MySQL, the MongoDB installation comes with a shell in which you may interact with the database content using JavaScript. Read [Mongo Shell guide](https://docs.mongodb.com/manual/reference/mongo-shell/) (<https://docs.mongodb.com/manual/reference/mongo-shell/>) for more details. Below is a set of basic Mongo Shell commands.

Creating and using a database. This command creates a new database if it does not exist and switches the context to that database:

```
use db_name
```

Unlike RDBMS, MongoDB is flexible with fields in documents, and even allows documents to be nested.

Examples to insert a document:

```
db.collection_name.insert ({
    "name":"sample document"
})
db.collection_name.insert({
    "name":"sample document", "user": {"uid": "Tom"}
})
```

Documents can be selected from the database with the `find()` method, which takes the first parameter as a set of filters:

```
db.collection_name.find({"name":"sample document"})
```

And you can limit the number of rows returned in a `find()` operation by chaining a `limit()` at the back:

```
db.collection_name.find().limit(30)
```

The Java API for MongoDB is similar to the commands in the Mongo Shell. For a thorough introduction to MongoDB's query language and how to develop a Java client, please refer to the Mongo Java guide (<https://docs.mongodb.com/drivers/java/sync/current/>).

Load data

After the MongoDB task is set up, connect to the instance. You can download `posts.json` and use the `mongoimport` utility (<https://docs.mongodb.com/manual/reference/program/mongoimport/>) that comes with the MongoDB installation to load data into the database.

```
# Download the data file
wget https://clouddeveloper.blob.core.windows.net/datasets/social-network/project/posts.json

# Load the data file into MongoDB
# 1. Database name    -- reddit_db
# 2. Collection name  -- posts
mongoimport --db reddit_db --collection posts --drop --file posts.json

# Once the import is completed, you should see the following message
37564141 document(s) imported successfully. 0 document(s) failed to import.
```

Importing ~10GBs of data may take around 20 - 30 minutes. You can continue reading the writeup in the meantime. Finishing the writeup first may help you get the big picture and come up with an optimal project-solving strategy.

In each document, the `_id` field serves as a primary key and is automatically added to the document. After you have imported the posts data in JSON format into your MongoDB instance, you will notice the additional field `_id` in the document.

Task Implementation

Information

Each of your databases will be accessed in more than one task. Therefore, it is recommended that you write modularized and reusable programs. We recommend that you finish reading the whole writeup before you start writing code. Aspire to achieve a modularized and reusable design.

1. You will implement your solution in `HomepageServlet.java`. The query in this task requests all the comments made by a user. Your web application will receive a single username as a request parameter, and you need to send back all of the user's comments.
2. Each line in `posts.json` is a JSON object for a single comment. The detailed structure is listed here:

```
{
  "cid": "xxx",                // Comment ID
  "parent_id": "xxx",         // The parent comment of this co
mment
  "uid": "xxxxx",             // User name of the comment
  "timestamp": "xxxxxxxxxx",   // When post is posted
  "content": "xxxxxx",        // Comment content
  "subreddit": "xxxx",        // Subreddit category of this co
mment
  "ups": xx                    // Number of ups received
  "downs": xx                  // Number of downs received
}
```

3. You should sort the comments by `ups` in **descending** order (from the largest to the smallest one). If there is a tie in the ups, sort the comments in **descending** order by their `timestamp`.
4. You **should exclude** `_id` field from each comment in the web response. One potential solution is to use Projections (<https://docs.mongodb.com/drivers/java/sync/current/fundamentals/crud/read-operations/project/>).
5. Be sure to set the environment variables before launching the backend:

```
export MONGO_HOST=127.0.0.1
```

Convert BSON to JSON in Java

MongoDB stores data in BSON format (which is the binary version of JSON). Hence, you need to convert BSON to JSON in `HomepageServlet.java`. One possible approach to convert BSON to JSON is by using Google GSON.

```
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.mongodb.client.MongoCursor;
import org.bson.Document;

MongoCursor<Document> cursor = collection.find(filter)
    .sort(sort).projection(projection).iterator();
try {
    while (cursor.hasNext()) {
        JsonObject jsonObject = new JsonParser().parse(cursor.next().toJson()).getAsJsonObject();
    }
} finally {
    cursor.close();
}
```

Request and Response Format

Request Format

```
GET /task3?id=[UserName]
```

Response Format:

Each JSON object (e.g. comment1_json) in the "comments" JSON array should follow the format described above.

```
{"comments":[{"comment1_json}, {comment2_json}, ...]}
```

Indexes

To speed up the web service in task 3 and task 4, you **MUST** select the proper fields and create indexes on them. You may index the `uid` field of the `posts` collection using the following command.

```
db.posts.createIndex({"uid": -1})
```

Refer to this

(<https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/>) document for more details on the MongoDB collection indexing.

Sample Backend Request for Task 3

After implementing the servlet and setting the environment variables, start the Java web server first by running:

```
cd /home/clouduser/social_network_backend
mvn clean package exec:java -Dmaven.test.skip=true
```

Sample Request:

```
http://<INSTANCE_PUBLIC_IP>:8080/MiniSite/task3?id=zylo_
```

Sample Response (Some comments are omitted):

```
{
  "comments": [
    {
      "uid": "zylo_",
      "downs": 0,
      "parent_id": "t1_crpy37f",
      "ups": 9,
      "subreddit": "Buttcoin",
      "content": "EDIT: What Derpy_Hooves11 said.\n\nIf I understand bitcoin correctly, what will happen is that BTC blocks will be valid blocks for both XT and BTC software, so XT will occasionally fork for say one block, but the fork will go nowhere because the majority will still be on BTC. Once the majority goes to XT, XT will start winning, and BTC will be left in the cold because it won't recognize the new blocks.",
      "cid": "t1_crpyt24",
      "timestamp": "1433001884"
    },
    (more comments here)
  ]
}
```

Test Task 3 with the Frontend

Go to the Task 3 tab, enter a reasonable user id, and hit search and you should be able to see the following screen if your backend is implemented:

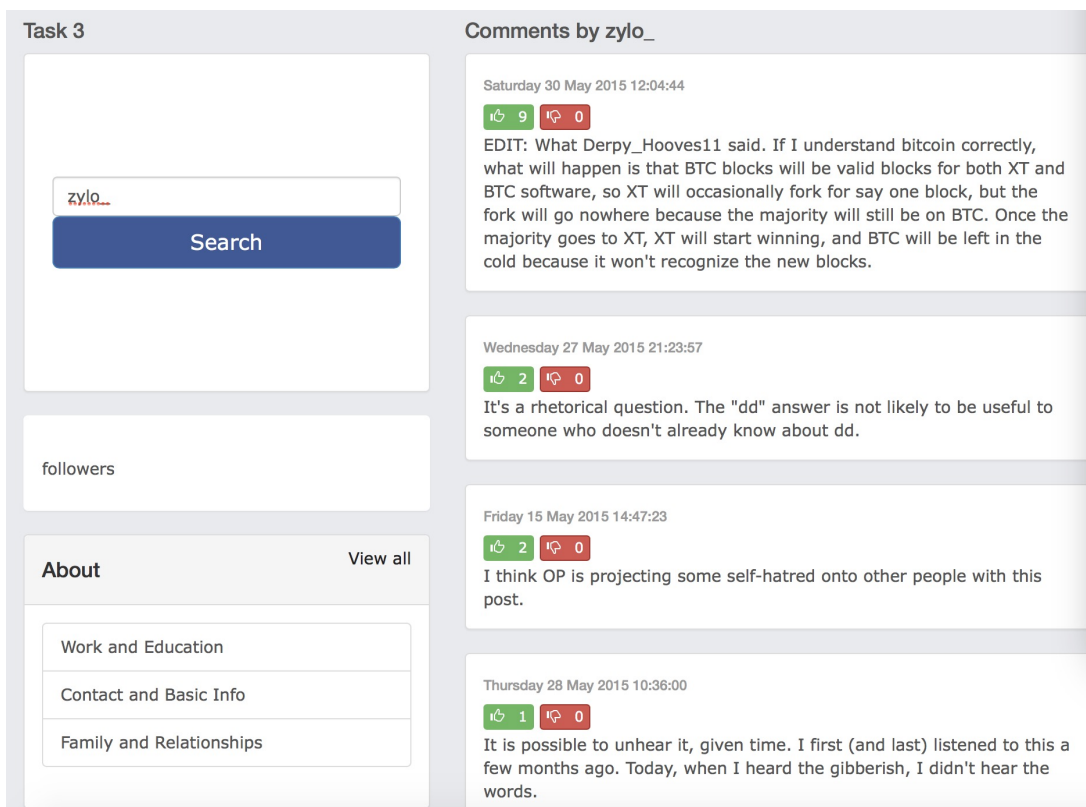


Figure 20: Task 3, view of the sorted comments.

What to Submit

1. Make sure the following files are under the

~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory:

- ProfileServlet.java
- FollowerServlet.java
- HomepageServlet.java
- TimelineServlet.java
- Cache.java
- TimelineWithCacheServlet.java
- MiniSite.java and any other helper source code files
- references file
- submitter

2. It is good practice to always include comments to **explain complicated logic and functions** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.

How to Submit

1. Copy your reference file and Java code to
`~/social_network_backend/src/main/java/edu/cmu/cc/minisite/`.
2. Keep your backend running in one SSH session.
3. To run the submitter, launch a second terminal and create a second SSH connection. Run the submitter under the
`~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory. Make sure that your backend is running and can respond to queries.
4. You do not need to finish all the tasks before receiving a score. You will most likely submit after each task.
5. Execute the submitter under the
`~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory.

```
export HISTIGNORE="export*" # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME="your_submission_username"
export SUBMISSION_PASSWORD="your_submission_pwd"
./submitter -t task3
```

Troubleshooting

If you get a "-" as a score for your submission, it is because your web service took too long to respond to our autograder. Make sure you have created the indexes correctly.

Try to test your backend with the "curl" command or by using the frontend application and make sure that your backend responds in a reasonable timeframe. The submitter will send a grading request to the Auto Grading Service and the amount of time taken to finish the grading depends on the performance of your backend web server. We don't grade on performance but there is a timeout on each request. So if you receive feedback such as "Grading timeout", try to make your query more efficient.

Task 4: Put Everything Together

Social Network Timeline Task

In Task 4, you will need to make use of all the databases you have worked on from task 1 to task 3 to implement a "Timeline page" for your simple social networking site. Hence, you will write a complex fan-out query. Fan-out in this context means splitting a large request payload into smaller batches and sending out the request multiple times (lower-level modules) to the service. This is because sometimes larger requests may take longer to complete than smaller requests. So it's worth testing whether the request handling can speed up if we turn one large request into a number of smaller requests instead.

You will implement your solution in `TimelineServlet.java`. The query in this task will display a user's timeline, which is similar to the timeline on your favorite social media platforms. For example, your Twitter timeline displays posts (texts, images, videos) from your followees (the people you follow). Similarly to Twitter, a user's timeline in this task will display comments from their followees.

Task Implementation

Information

In addition to learning about cloud storage, we expect you to follow the best coding practices in the industry. As we have recommended previously, your implementation from task 1 to task 3 should have been modular and reusable. Thus for this task, we ask you to avoid WET (Write Everything Twice). That is, you should be able to reuse some of your classes and methods from task 1 to task 3 because you will be performing some similar functionalities such as establishing connections to the databases. Please feel free to redesign and/or refactor servlets for previous tasks in order to support your query.

1. In this task, your web application will receive a single `username` (the user's timeline we are interested in) as a request parameter. Then, you will need to build queries that ask for the most `popular` 30 comments from the followees and the personal information of that given user. Note that `popular` refers to the comments with the most `ups`. More specifically, your response should consist of the user's **UserName, Profile Image URL, list of Followers**, and from the user's followees, you need to find and return the **30 most popular comments** and the **parent and grandparent comments** associated with each of them, if any. See the definition of **parent comment** and **grandparent comment** in point 4. A quick note on the difference between followers and followees:
 - Followers are the people that follow you.
 - Followees are the people you follow.
2. You should sort the followers **lexicographically** in **ascending** order by `username`.
3. You should sort the **most popular 30** comments from the **followees** based on `ups` in **descending order**. If there is a tie in the `ups`, sort the comments in **descending** order by their `timestamp`. If there are less than **30** comments, you should return all the comments.
4. For every comment, if its parent is in the database, you should put the parent into the comment JSON as a key-value pair `"parent" : COMMENT_JSON`. Similarly, if the parent comment of the parent comment is in the database, you should put it into **the comment JSON** as a key-value pair `"grand_parent" : COMMENT_JSON`. Note that, if `parent_id` of comment A is the same as `'cid'` of comment B, comment B is the parent of comment A.
5. Getting the same result as the sample response in writeup (see below) does not guarantee a full score. This is just an example to show you the response format. Try to test with more corner cases.

Sample Backend Request for Task 4

Sample Request:

```
GET http://<INSTANCE_PUBLIC_IP>:8080/MiniSite/task4?id=Branibor
```

Sample Response:

Note: `_id` field is generated by MongoDB automatically and you **MUST exclude** this field similar to Task 3. Besides, since the provided `posts.json` is only a subset of posts on Reddit, there might be users with field `parent_id` that exists but can not be found in our dataset.

```

{
  "followers": [
    {
      "profile": "https://farm4.staticflickr.com/3733/10498671606_caba28c5a6_m.jp
pg",
      "name": "Fortanono"
    },
    {
      "profile": "https://farm6.staticflickr.com/5779/20996394095_e3a527d780_m.jp
pg",
      "name": "That_Zeffia_guy"
    },
    (More followers...)
  ],
  "comments": [
    {
      "uid": "MacerV",
      "downs": 0,
      "parent_id": "t3_34u8s5",
      "ups": 466,
      "subreddit": "KerbalSpaceProgram",
      "content": "Just started chuckling uncontrollably at the gif. Good choic
e.",
      "cid": "t1_cqy3fxn",
      "timestamp": "1430757711"
    },
    {
      "uid": "MacerV",
      "downs": 0,
      "parent_id": "t3_37afc6",
      "ups": 197,
      "subreddit": "hockey",
      "content": "Well thats a rather redundant statement.",
      "cid": "t1_crl0nvs",
      "timestamp": "1432613962"
    },
    {
      "grand_parent": {
        "uid": "IAteOkayZebraVulva",
        "downs": 0,
        "parent_id": "t1_cr4lbp4",
        "ups": 421,
        "subreddit": "creepyPMs",
        "content": "Yup. She also threatened to kill herself and blamed my frien
d. But you know, she was the best big she could be.",
        "cid": "t1_cr4lxec",
        "timestamp": "1431276739"
      },
      "uid": "onthefence928",
      "parent": {
        "uid": "Das_Perderdernerter",
        "downs": 0,
        "parent_id": "t1_cr4lxec",

```

```

    "ups": 102,
    "subreddit": "creepyPMs",
    "content": "As someone not in the American schooling system, what's a
\\\"Big\\\"?",
    "cid": "t1_cr4oi20",
    "timestamp": "1431281880"
  },
  "downs": 0,
  "parent_id": "t1_cr4oi20",
  "ups": 173,
  "subreddit": "creepyPMs",
  "content": "In the greek fraternity system they gave a tradition where mor
e senior members play big sibling to younger members, it's supposed to work like
mentorship. ",
  "cid": "t1_cr4omt3",
  "timestamp": "1431282139"
},
{
  "grand_parent": {
    "uid": "Ragexz",
    "downs": 0,
    "parent_id": "t3_35e6or",
    "ups": 250,
    "subreddit": "KerbalSpaceProgram",
    "content": "Most linear conversation ever.",
    "cid": "t1_cr3kudr",
    "timestamp": "1431181222"
  },
  "uid": "MacerV",
  "parent": {
    "uid": "drillgorg",
    "downs": 0,
    "parent_id": "t1_cr3kudr",
    "ups": 185,
    "subreddit": "KerbalSpaceProgram",
    "content": "It's how us engineering students communicate.",
    "cid": "t1_cr3l0ay",
    "timestamp": "1431181614"
  },
  "downs": 0,
  "parent_id": "t1_cr3l0ay",
  "ups": 143,
  "subreddit": "KerbalSpaceProgram",
  "content": "I can confirm. You learn quickly to speak clearly and effectiv
ely because communication errors cause mistakes; sometimes lethal ones. ",
  "cid": "t1_cr3lsxq",
  "timestamp": "1431183463"
},

  (More comments...)

  "profile": "https://farm2.staticflickr.com/1519/26254768136_09c360b4c1_m.jpg",
  "name": "Branibor"
}

```


Information

Hint:

the JSON object returned in this task can be very long with a complicated structure. You may find it handy to compare two JSON objects using semantic JSON compare tools such as jsdiff (<http://www.jsdiff.com/>).

Test Task 4 with the Frontend

Go to the Task 4 tab and you should be able to see the following screen if your backend is implemented correctly:

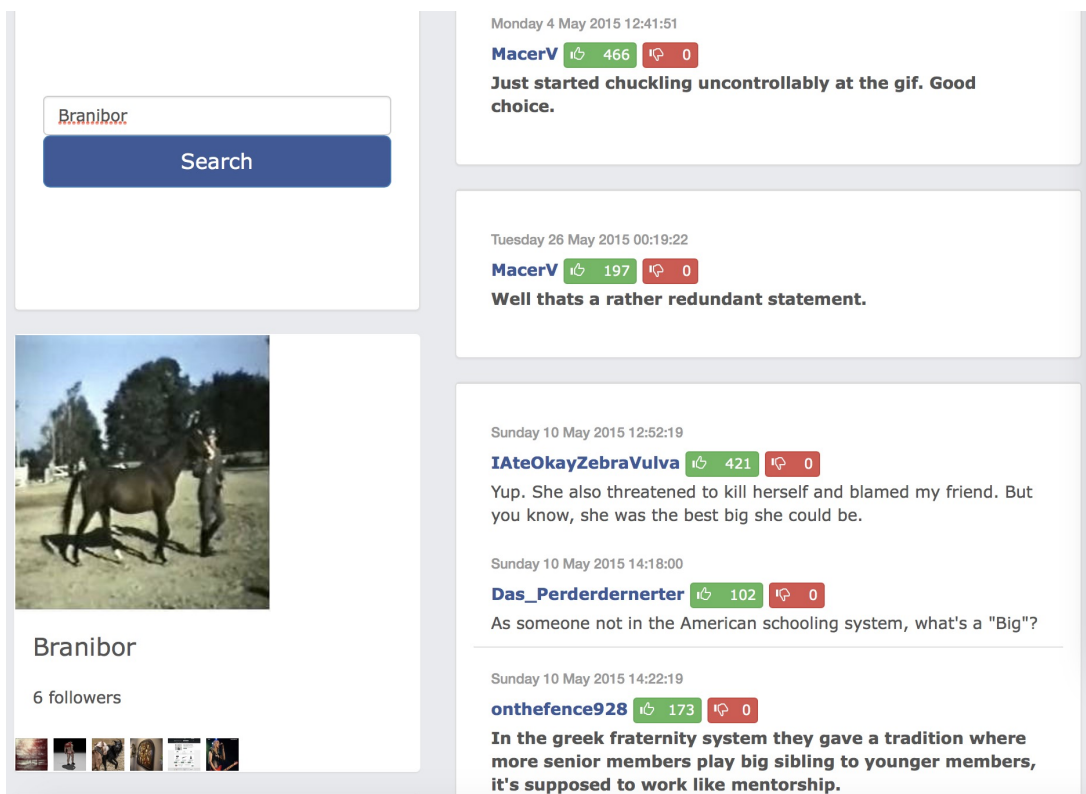


Figure 22: Task 4, view of the caption.

What to Submit

1. Make sure the following files are under the `~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory:
 - ProfileServlet.java
 - FollowerServlet.java
 - HomepageServlet.java
 - TimelineServlet.java
 - Cache.java
 - TimelineWithCacheServlet.java
 - MiniSite.java and any other helper source code files
 - references file
 - submitter

2. It is good practice to always include comments to **explain complicated logic and functions** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.

How to Submit

1. Copy your reference file and Java code to
~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ .
2. Keep your backend running in one SSH session.
3. To run the submitter, launch a second terminal and create a second SSH connection. Run the submitter under the
~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory. Make sure that your backend is running and can respond to queries.
4. You do not need to finish all the tasks before receiving a score. You will most likely submit after each task.
5. Execute the submitter under the
~/social_network_backend/src/main/java/edu/cmu/cc/minisite/ directory.

```
export HISTIGNORE="export*" # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME="your_submission_username"
export SUBMISSION_PASSWORD="your_submission_pwd"
./submitter -t task4
```

Troubleshooting

If you get a "-" as a score for your submission, it is because your web service took too long to respond to our autograder. Think of how you can make your query more efficient using indexes.

Try to test your backend with the "curl" command or by using the frontend application and make sure that your backend responds in a reasonable timeframe. The submitter will send a grading request to the Auto Grading Service and the amount of time taken to finish the grading depends on the performance of your backend web server. We don't grade on performance but there is a timeout on each request. So if you receive feedback such as "Grading timeout", try to make your query more efficient.

Task 5: Caching

Social Network Timeline Cache Task

After finishing Task 4, you are able to fetch the timeline for a given user. However, during the test, you can notice that the response time is long when you fetch a user with a large number of followers or followees. These kinds of users are the so-called "top users" or influencers on social media. In social media, queries related to the "top users" produce a high load on the server, which means that optimizing this type of query will benefit a majority of the users on social media.

To optimize this kind of search, most social media applications use a caching mechanism to provide a faster loading experience for users. In this task, you are required to implement the caching mechanism for the social networking timeline you created in Task 4.

The caching mechanism can be implemented in many different ways. Some databases have their own built-in caching layers while some companies use external databases to store requests and their corresponding responses.

MySQL caches depending on the storage engine being used, which was illustrated in a previous project.

Neo4j supports query plan caching. You can configure the query parameters to optimize the query speed as suggested here (<https://neo4j.com/developer/kb/understanding-neo4j-query-plan-caching/>). You can also warm up the cache to improve performance from a cold start. You are encouraged to read this tutorial (<https://neo4j.com/developer/kb/warm-the-cache-to-improve-performance-from-cold-start/>) although we will not cover it in this project.

MongoDB has its own query cache as well. MongoDB tries to cache everything in memory, including queries, fields, even the entire database, if it has built proper indices. MongoDB also provides commands (<https://docs.mongodb.com/manual/reference/method/js-plan-cache/>) to view what its inside caches look like.

Some applications have their own caching layer. Redis is a popular in-memory key-value database that is used as a caching layer for the storage and retrieval of key-value items.

In this task, you are going to implement a simple in-memory caching mechanism using the key-value pair store class named `Cache` provided to you in the starter Java code. It is important to note that, for internet-scale companies like Facebook or Twitter it is not financially viable to cache all user data. Thus, these companies sometimes cache "top users" or influencers. In your implementation, we also require you to only cache "top users".

In the real world, a caching mechanism is much more complex than what we have implemented here. This task is focused on enabling you to implement a basic caching mechanism.

Task Implementation

1. You will implement your solution in `TimelineWithCacheServlet.java`. The requirements for generating the timeline for a user are the same as those in Task 4. Please refer to Task 4 for detailed requirements.
2. In addition to task 4, you need to cache the requests related to the "top users". A user is considered a "top user" if this user has more than 300 followers. Since the requests related to the top users are being cached, your backend needs to return the response to these requests within 300ms. We do not have a strict time limit for new requests or first-time requests and requests related to normal users but you still need to handle these requests within 60s.
3. You should **NOT** cache the requests related to the normal users i.e., users who have less than or equal to 300 followers since these users may take up 95% of all users and caching all of them is impossible. Our grader will also test to see if your solution is caching all the requests.
4. You are required to use the `Cache` class that is provided as a part of the starter code to implement the caching mechanism. We have defined a class member variable called `Cache` and you should use this variable to implement your caching mechanism. You

can refer to the documentation provided in the starter code to understand how to use this class. You should NOT modify the `Cache` class and you should NOT use other key-value store classes.

Danger

1. Don't modify the `doGet` function inside `TimelineWithCacheServlet.java`, otherwise, you will get an incorrect score for this task.
2. To receive credit for this task, you must use the variable `cache` given in the starter code instead of defining your own.

What to Submit

1. Make sure the following files are under the `~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory:
 - `ProfileServlet.java`
 - `FollowerServlet.java`
 - `HomepageServlet.java`
 - `TimelineServlet.java`
 - `Cache.java`
 - `TimelineWithCacheServlet.java`
 - `MiniSite.java` and any other helper source code files
 - references file
 - `submitter`
2. It is good practice to always include comments to **explain complicated logic and functions** and **adopt a readable style** as you are writing the code. This is reflected in your code style and comments. We will manually grade your last submission of each task.

How to Submit

1. Copy your reference file and Java code to `~/social_network_backend/src/main/java/edu/cmu/cc/minisite/`.
2. Keep your backend running in one SSH session.
3. To run the submitter, launch a second terminal and create a second SSH connection. Run the submitter under the `~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory. Make sure that your backend server is running and can respond to queries.
4. You do not need to finish all the tasks before receiving a score. You will most likely submit after each task.
5. Execute the submitter under the `~/social_network_backend/src/main/java/edu/cmu/cc/minisite/` directory.

```
export HISTIGNORE="export*" # so that the following export commands will not be tracked into bash history
export SUBMISSION_USERNAME="your_submission_username"
export SUBMISSION_PASSWORD="your_submission_pwd"
./submitter -t task5
```

Delete the Azure Resource Group

Azure Resource Group enables you to organize your resources and arrange resources with the same life cycle in the same resource group. Ensure all resources are deleted by cross-checking on the Azure Portal as well.

By using Azure resource group, you can terminate all the resources you used in this project with one single command.

```
terraform destroy
```

Be sure to check that the Azure MySQL Server that you created is deleted as well!

Project Reflection Task (Mandatory, graded)

Project Reflection Task (Mandatory, graded)

Upon completing this project you will make one (1) post in the [Forum] P3. Cloud Storage - Heterogeneous Storage on the Cloud (<https://projects.sailplatform.org/cloud-forum/topic/publish/1173/>) to reflect on your experience before the project deadline. **Before you publish the post, please double-check that the category is the project category "the title of the module, e.g., Heterogeneous Storage on the Cloud", NOT the course category "the title of the course, e.g., Cloud Computing" or any other project category.**

Consider the following topics when creating your post, however, you should never share any code snippets in your reflection:

- Describe your approach to solving each task in this project. Explain alternative approaches that you decided not to take and why.
- Describe any interesting problems that you had overcome while completing this project.
- If you were going to do the project over again, how would you do it differently, and why?

After completing this task, confirm that your **Reflection Score** has been automatically updated on the scoreboard before the project deadline.

Project Survey

Project Survey

Please leave us feedback for this project here. This will help us strengthen this project for future offerings.

Project Discussion

Project Discussion Task (Mandatory, graded)

After this project has been completed (after the project deadline), all reflection posts by all students will become visible for review.

Your task is to reply and provide feedback to **3** posts in the [Forum] P3. Cloud Storage - Heterogeneous Storage on the Cloud (<https://projects.sailplatform.org/cloud-forum/category/1173/>), within **7** days after the project deadline.

After completing this task, confirm that your ***Discussion Score*** has been automatically updated on the scoreboard within 7 days after the project deadline.