LetsGITit - Design Document

Table of Contents

1. Project Title and Authors

1.1. Assigned Team Number

????

1.2. Team Name

LetsGITit

1.3. Team Members

Name	USC ID	USC Email
Edward Han	7421483867	hane@usc.edu
Joyce Wang	7039608014	joycew@usc.edu
Robert Diersing	3445624516	rdiersin@usc.edu
Summer Seo	8321974769	summerys@usc.edu
Woonghee Lee	5094791538	woonghel@usc.edu

2. Preface

2.1. Usage

This document will be able to be used as a guide to implement the KnowItAll system. The architectural design page will let the reader get an intuitive idea of what is going on. The package and class sections will allow the user to organize the system. The sequence diagrams allow the reader to implement the connections between classes and important event handling.

2.2. Interaction With Other Systems

This system will interact with usc email by sending an email to create a new user. The

system will send a link that links to another server inside of the system. KnowItAll will be a Web Application

3. Introduction

Introduction to KnowItAll

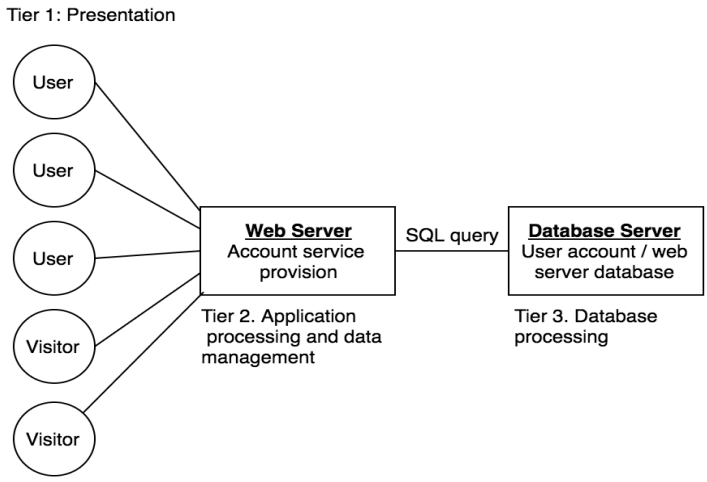
KnowItAll will be designed based on a Three-Tier Architectural Design system to highlight the major components that make up our web application and be organized according to a Client-Server Architecture Style. The Three-Tier Architectural Design and Client-Server Architecture Style closely correlate in terms of functionality, design, and organization. Our web application can be categorized by a three-tier system, where the 1st layer is the presentation layer, the 2nd layer is the application processing and data management layer, and the 3rd layer is the database processing layer. The Client-Server Architecture Style is made up of components and connectors, where the components for KnowItAll will be clients, server, database, and the connectors are a client to server and server to database bi-directional relationship, both of which translate seamlessly into our design pattern. Tier 1 will hold our client components, Tier 2 will hold our server component, and Tier 3 will hold our database component. And the relationship between each tier is maintained by the connectors, or the bidirectional component relationships. We ultimately feel that this design pattern and architectural style succinctly encapsulates all KnowItAll's functionality.

4. Architectural Design

4.1. We are using a Three-Tier Architectural Design

4.1.1. Diagram

https://en.wikipedia.org/wiki/Multitier_architecture



4.1.2. Tier1: Presentation Layer (User Interface)

This layer represents the front-end portion of our web application design. The presentation layer holds all our different clients and GUI components and is essentially the visual layer where users interact and start with, all on the KnowItAll homepage. Shares only a direct bi-directional relationship with Tier 2.

4.1.3. Tier2: Business Logic (Functionalities)

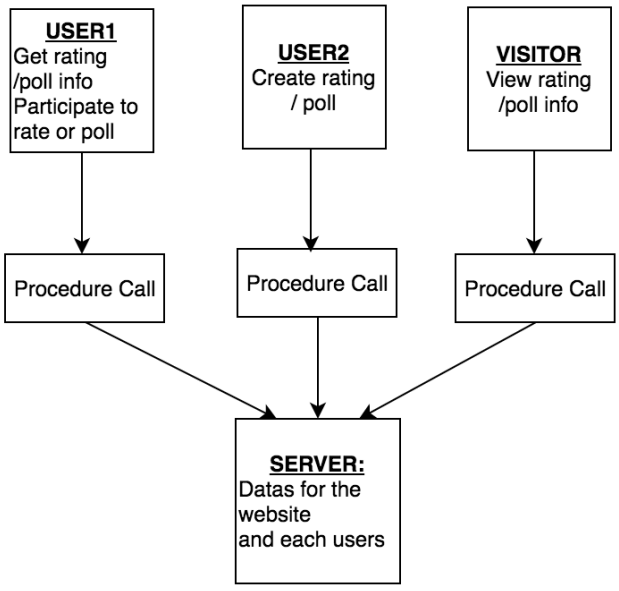
This layer is where the application processes and manages the data from the client. We will be using a web server for this to implement logic and how to best process and handle all incoming and outgoing data. Tier2 shares a link with almost all the other components/layers, and maintains a bidirectional relationship with Tier 1 and Tier 3, both for incoming and outgoing data.

4.1.4. Tier3: Data Layer (Database)

Tier 3 is the final layer which is essentially a data server that holds and secures all of our data. The data is called upon based by RPCs from Tier 2.

4.2. Architectural Style

We are using a Client-Server Architectural Style for our web application. Our rationale for choosing the client-server architectural style is that it is the most concise organization of our different components and data processing. In this approach, the server does not know and should not know the number or identities of clients, which exactly describes how our server should function. The server should only serve to match client information with the database, but not exploit it in any way for security/anonymity purposes. Our components will consist of a client-base, a web server, and a database. Our client-base includes users and visitors who send data to the server. Our server handles signals that occur when the clients interact with GUI components that will signal events to the server for data processing and transfer. The server also transfers data to the database for updates to client information and/or extracts wanted data from the database to update the user interface on the client side.



4.3. Components

4.3.1. Clients

In KnowItAll, clients are the initial components that will trigger different events by interacting with the various UI components on our web application. Our

clients are defined as either users or visitors. Users are clients who are registered in the system with a valid USC ID while visitors are merely onlookers who have limited access to KnowItAll. Users have the ability to modify and create polls and ratings, and search for and view existing polls and ratings by categories and tag names. Visitors will only be able to search for and view but will not be granted access to modify or create anything. Generally speaking, the function of our clients should mostly be to simply send and receive data nearly instantaneously by interacting with the different GUI components that will signal to the server what action to take. We will create a User Class to implement this.

4.3.2. Server

The server, specifically web server, is the main control center for KnowItAll. Any type of data processing, data transfer, service request, error handling will be taken care of in the server. The responsibility of the server is to interact with both the user and the database so that all the data is consistent with one another and displays the correct and updated information. The server backend will mainly be implemented using java servlets. From the client, incoming data or input includes but is not limited to login credentials, signals fired for a poll/rating creation button pressed, search text for categories etc. Output data will be refreshed dynamically so data appears appropriately for the client. The server will also execute SQL statements to the database to retrieve data, and the SQL database will respond in turn by fetching and returning the desired data to the server.

4.3.3. Database

The database will be implemented using MySQL. The database will contain information about the registered users and each of the user's survey. Once any of the information is updated inside the database, it will immediately make a call to change the user interface. Because every email is unique, we will be using the email address to avoid duplicate users. Also for security reason, we'll be hashing all the passwords when storing to our database. Each survey will contain a key so we can reference the owner of the survey in the future when needed.

4.4. Connectors

4.4.1. Client to Server (Bi-directional)

There will be a bi-directional connection between the client and server. On the client side, the client will make many and various requests to the server. For the first time user, the client will request to validate the email address. After the client registers, an email with a link will be sent to the client from the server side for verification. If the client is already a registered user, the LoginHandler()

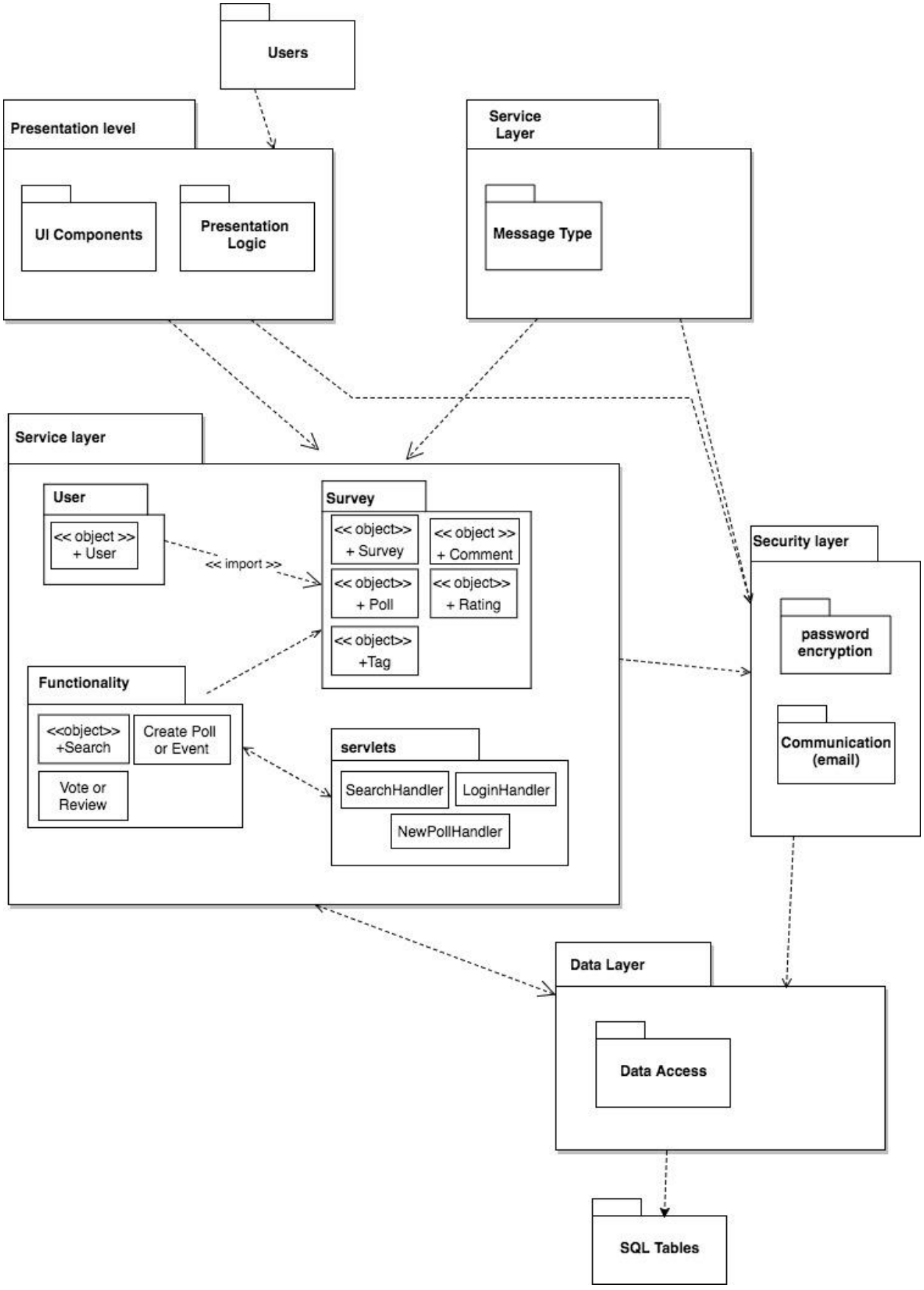
method will handle the client's request to see whether he/she exists in our database. Once the client has been verified, the client may make a search request which should prompt the server to handle this request and execute this query to our SQL database, all in a searchHandler() method that returns a set of search results. While visitors can also make a search requests without verification, only a valid user may request the server to add his/her own polls and ratings. Server will then use NewPollHandler() method to add the poll or the rating in our database. After it's successfully created, server will make a request to update the user interface so that the client's new survey will be displayed or appear in the future once refreshed. Similar method will apply when the user decides to delete his/her survey.

4.4.2. Server to Database (Bi-directional)

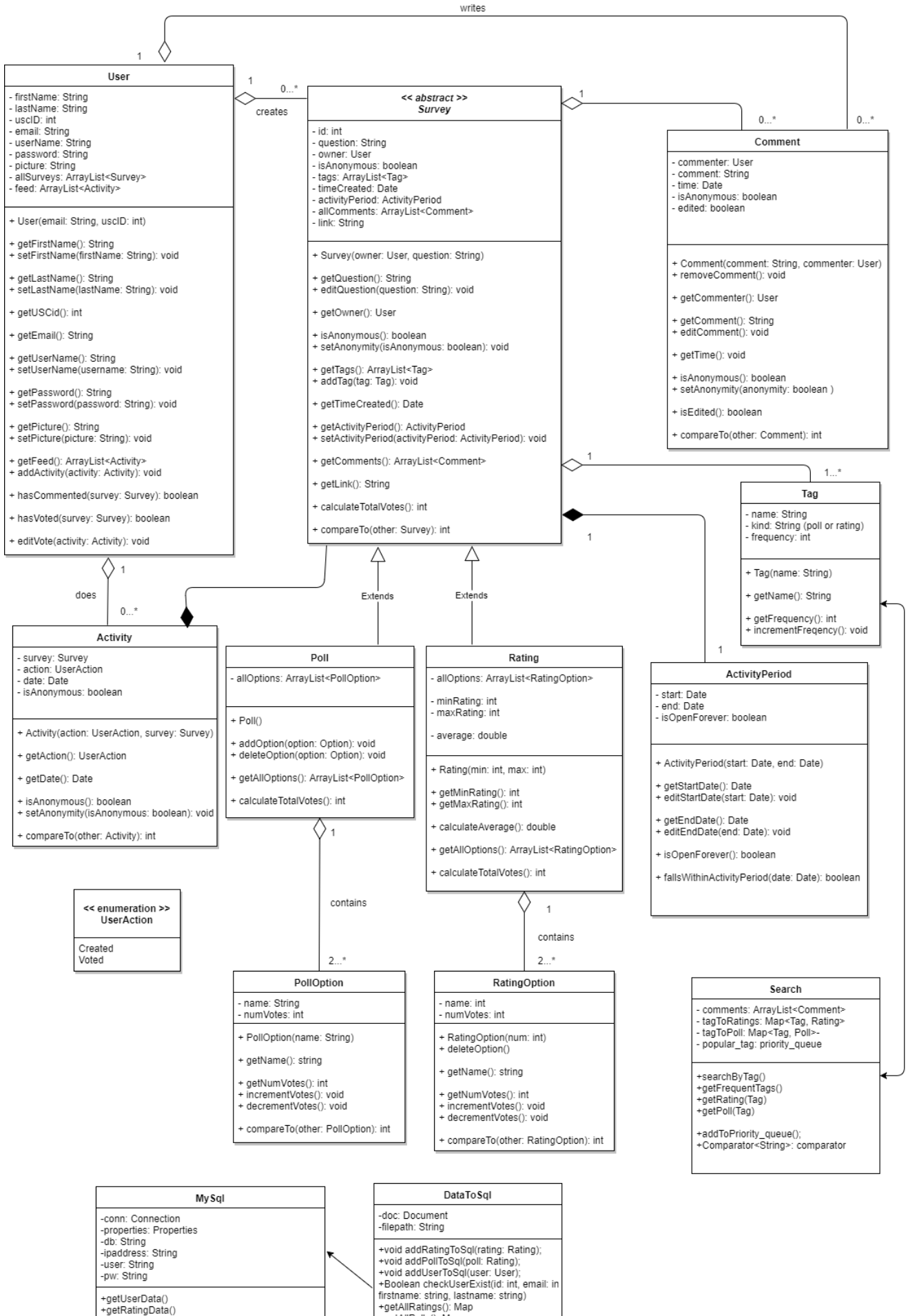
To connect the server and database, we will be using JDBC. The JDBC API will effectively handle all incoming client requests from the server, primarily through RPCs. Our RPCs will be SQL statements that can be executed nicely as a dual function of java and sql, thanks to the JDBC API. Once the server queries the database for data that the client wants by executing SQL statements, the database should return this, in the form of whether the data exists, all matching or relevant elements of the data, etc. We also handle this data return in the server and effectively establish the bi-directional connection for the server and database. While the server should not take the job of the database and store any information, it should utilize some variables to keep track of specific requests that the client has made for the database. These requests can be boolean/binary or unary in nature and fluctuate depending on what the database returns.

5. Detailed Design

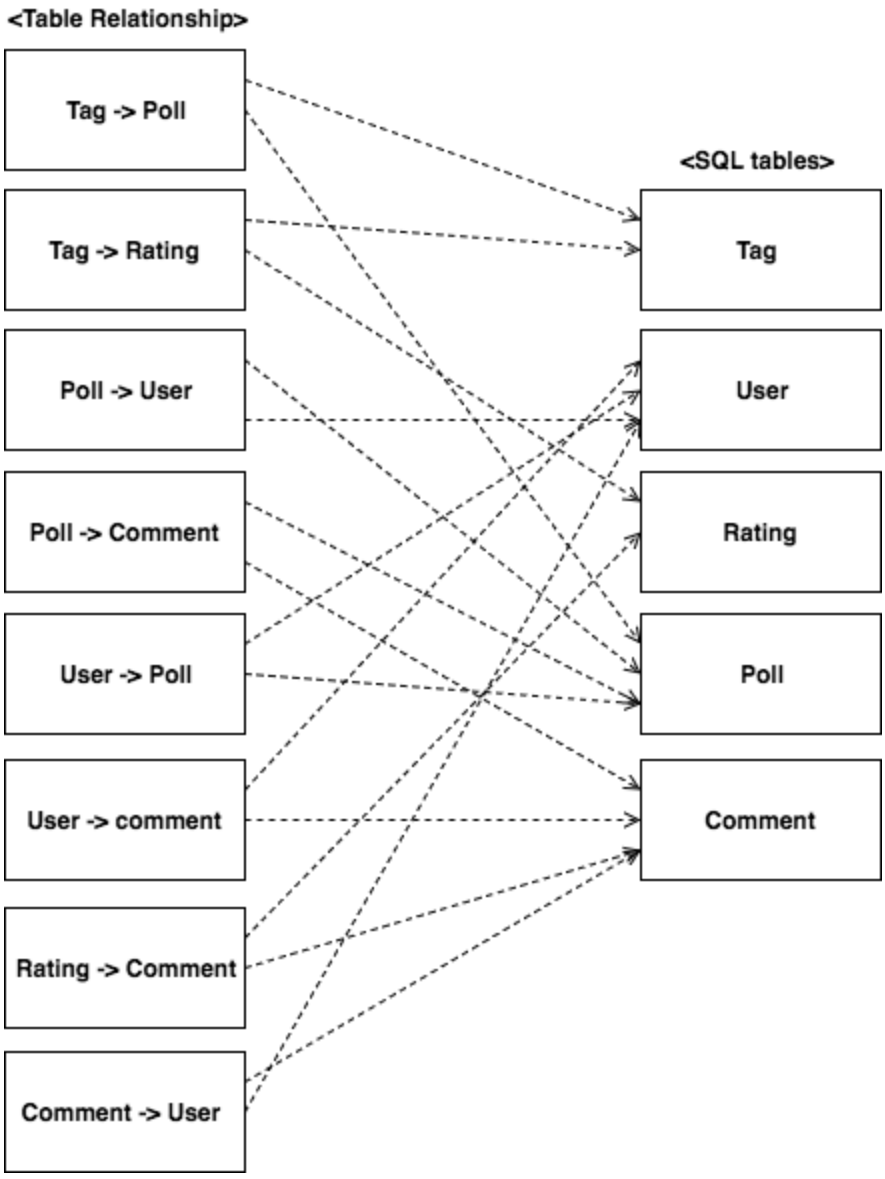
5.1. Package Diagram


Users
Presentation level
UI Components
Presentation Logic
Service Layer
Message Type
Service layer
User
<< object >> + User
<< import >>
Survey
<< object>> + Survey
<< object >> + Comment
<< object>> + Poll
<< object>> + Rating
<< object>> +Tag
Functionality
<<object>> +Search
Create Poll or Event
Vote or Review
servlets
SearchHandler
LoginHandler
NewPollHandler
Security layer
password encryption
Communication (email)
Data Layer
Data Access
SQL Tables

5.2. Class Diagram

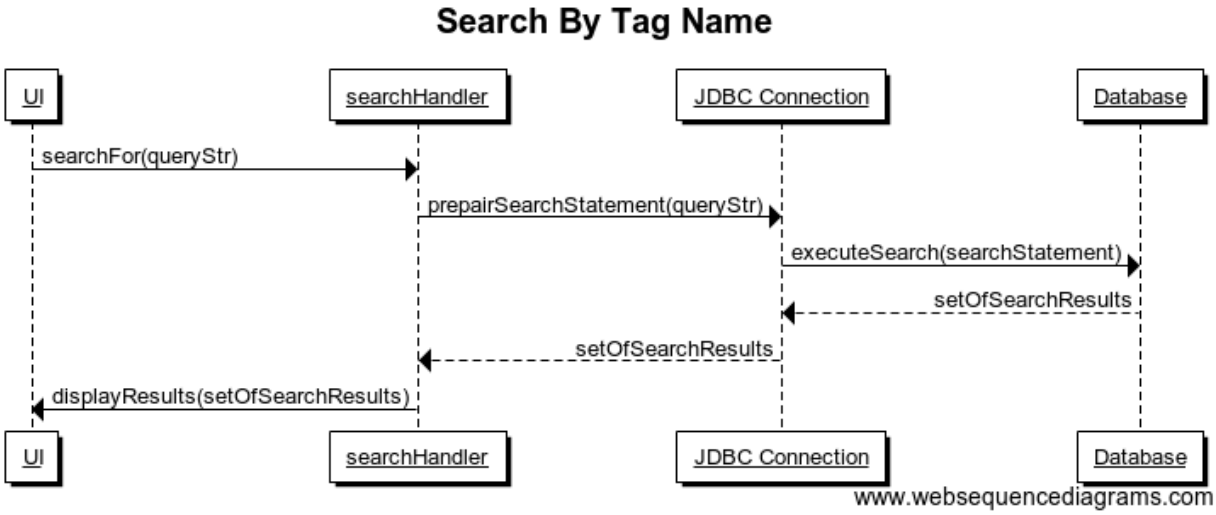

writes
1
User
- firstName: String
- lastName: String
- uscID: int
- email: String
- userName: String
- password: String
- picture: String
- allSurveys: ArrayList<Survey>
- feed: ArrayList<Activity>
+ User(email: String, uscID: int)
+ getFirstName(): String
+ setFirstName(firstName: String): void
+ getLastName(): String
+ setLastName(lastName: String): void
+ getUSCid(): int
+ getEmail(): String
+ getUserName(): String
+ setUserName(username: String): void
+ getPassword(): String
+ setPassword(password: String): void
+ getPicture(): String
+ setPicture(picture: String): void
+ getFeed(): ArrayList<Activity>
+ addActivity(activity: Activity): void
+ hasCommented(survey: Survey): boolean
+ hasVoted(survey: Survey): boolean
+ editVote(activity: Activity): void
1
0...*
creates
<< abstract >>
Survey
- id: int
- question: String
- owner: User
- isAnonymous: boolean
- tags: ArrayList<Tag>
- timeCreated: Date
- activityPeriod: ActivityPeriod
- allComments: ArrayList<Comment>
- link: String
+ Survey(owner: User, question: String)
+ getQuestion(): String
+ editQuestion(question: String): void
+ getOwner(): User
+ isAnonymous(): boolean
+ setAnonymity(isAnonymous: boolean): void
+ getTags(): ArrayList<Tag>
+ addTag(tag: Tag): void
+ getTimeCreated(): Date
+ getActivityPeriod(): ActivityPeriod
+ setActivityPeriod(activityPeriod: ActivityPeriod): void
+ getComments(): ArrayList<Comment>
+ getLink(): String
+ calculateTotalVotes(): int
+ compareTo(other: Survey): int
1
0...*
Comment
- commenter: User
- comment: String
- time: Date
- isAnonymous: boolean
- edited: boolean
+ Comment(comment: String, commenter: User)
+ removeComment(): void
+ getCommenter(): User
+ getComment(): String
+ editComment(): void
+ getTime(): void
+ isAnonymous(): boolean
+ setAnonymity(anonymity: boolean)
+ isEdited(): boolean
+ compareTo(other: Comment): int
0...*
1
does
1
0...*
Activity
- survey: Survey
- action: UserAction
- date: Date
- isAnonymous: boolean
+ Activity(action: UserAction, survey: Survey)
+ getAction(): UserAction
+ getDate(): Date
+ isAnonymous(): boolean
+ setAnonymity(isAnonymous: boolean): void
+ compareTo(other: Activity): int
Extends
Extends
Poll
- allOptions: ArrayList<PollOption>
+ Poll()
+ addOption(option: Option): void
+ deleteOption(option: Option): void
+ getAllOptions(): ArrayList<PollOption>
+ calculateTotalVotes(): int
Rating
- allOptions: ArrayList<RatingOption>
- minRating: int
- maxRating: int
- average: double
+ Rating(min: int, max: int)
+ getMinRating(): int
+ getMaxRating(): int
+ calculateAverage(): double
+ getAllOptions(): ArrayList<RatingOption>
+ calculateTotalVotes(): int
1
1...*
Tag
- name: String
- kind: String (poll or rating)
- frequency: int
+ Tag(name: String)
+ getName(): String
+ getFrequency(): int
+ incrementFreqency(): void
1
1
ActivityPeriod
- start: Date
- end: Date
- isOpenForever: boolean
+ ActivityPeriod(start: Date, end: Date)
+ getStartDate(): Date
+ editStartDate(start: Date): void
+ getEndDate(): Date
+ editEndDate(end: Date): void
+ isOpenForever(): boolean
+ fallsWithinActivityPeriod(date: Date): boolean
<< enumeration >>
UserAction
Created
Voted
1
contains
2...*
1
contains
2...*
PollOption
- name: String
- numVotes: int
+ PollOption(name: String)
+ getName(): string
+ getNumVotes(): int
+ incrementVotes(): void
+ decrementVotes(): void
+ compareTo(other: PollOption): int
RatingOption
- name: int
- numVotes: int
+ RatingOption(num: int)
+ deleteOption()
+ getName(): string
+ getNumVotes(): int
+ incrementVotes(): void
+ decrementVotes(): void
+ compareTo(other: RatingOption): int
Search
- comments: ArrayList<Comment>
- tagToRatings: Map<Tag, Rating>
- tagToPoll: Map<Tag, Poll>-
- popular_tag: priority_queue
+searchByTag()
+getFrequentTags()
+getRating(Tag)
+getPoll(Tag)
+addToPriority_queue();
+Comparator<String>: comparator
MySql
-conn: Connection
-properties: Properties
-db: String
-ipaddress: String
-user: String
-pw: String
+getUserData()
+getRatingData()
DataToSql
-doc: Document
-filepath: String
+void addRatingToSql(rating: Rating);
+void addPollToSql(poll: Rating);
+void addUserToSql(user: User);
+Boolean checkUserExist(id: int, email: in
firstname: string, lastname: string)
+getAllRatings(): Map

5.3. Database Diagram

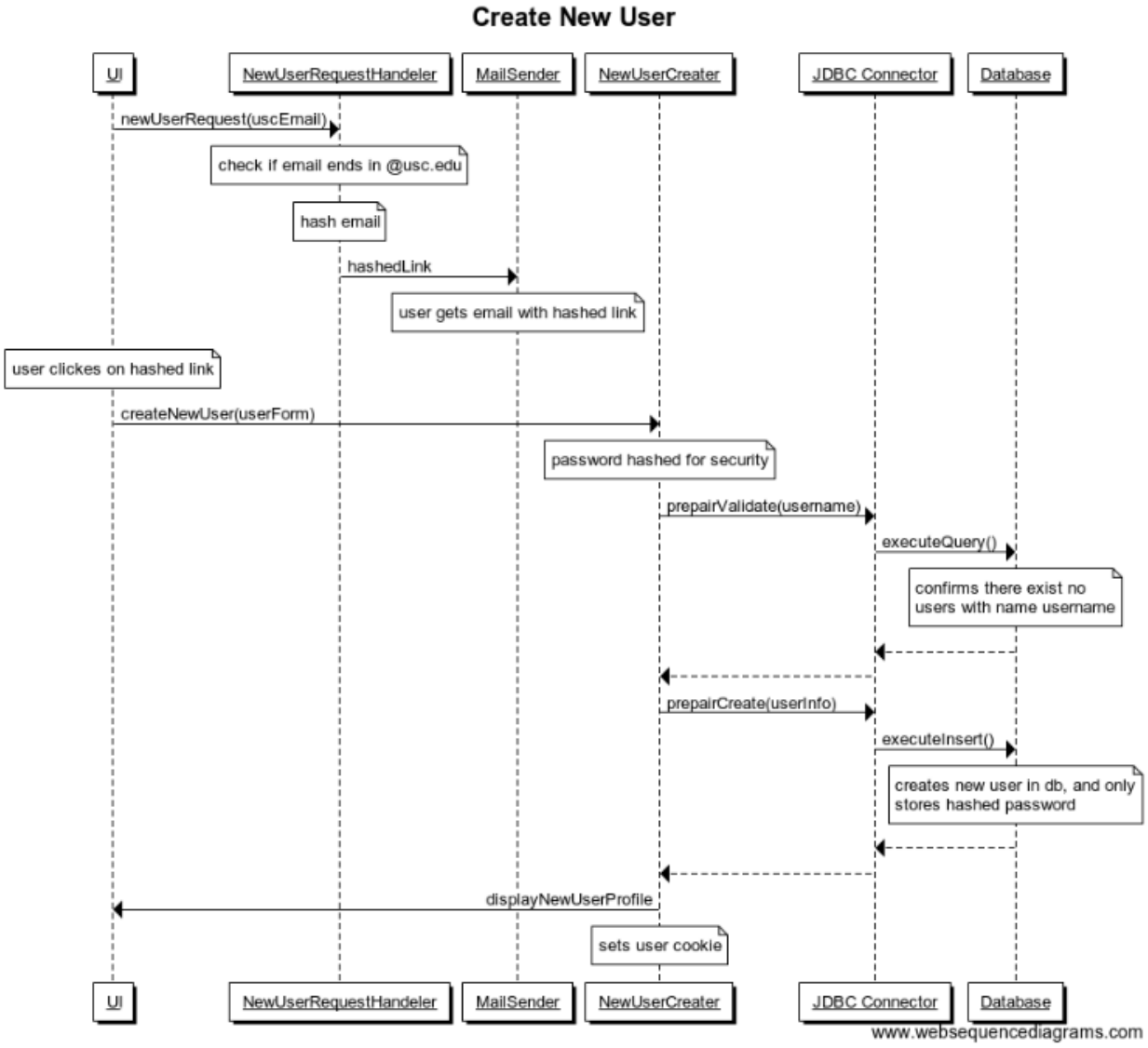


5.4. Sequence Diagrams

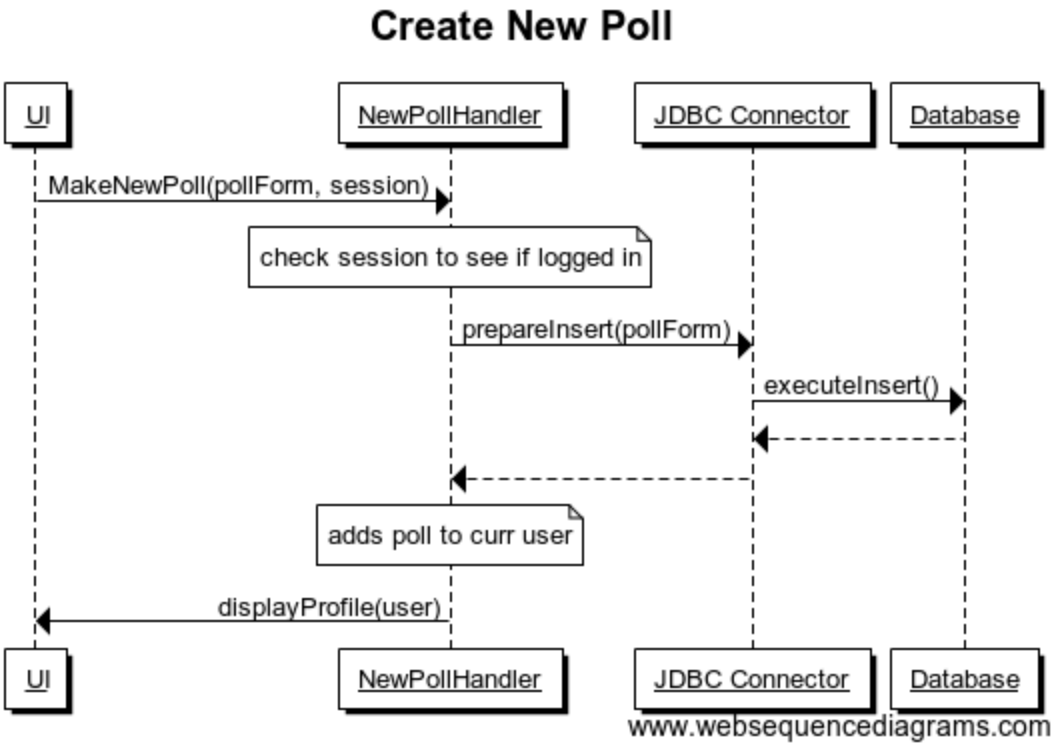
5.4.1. Search By Tag Name



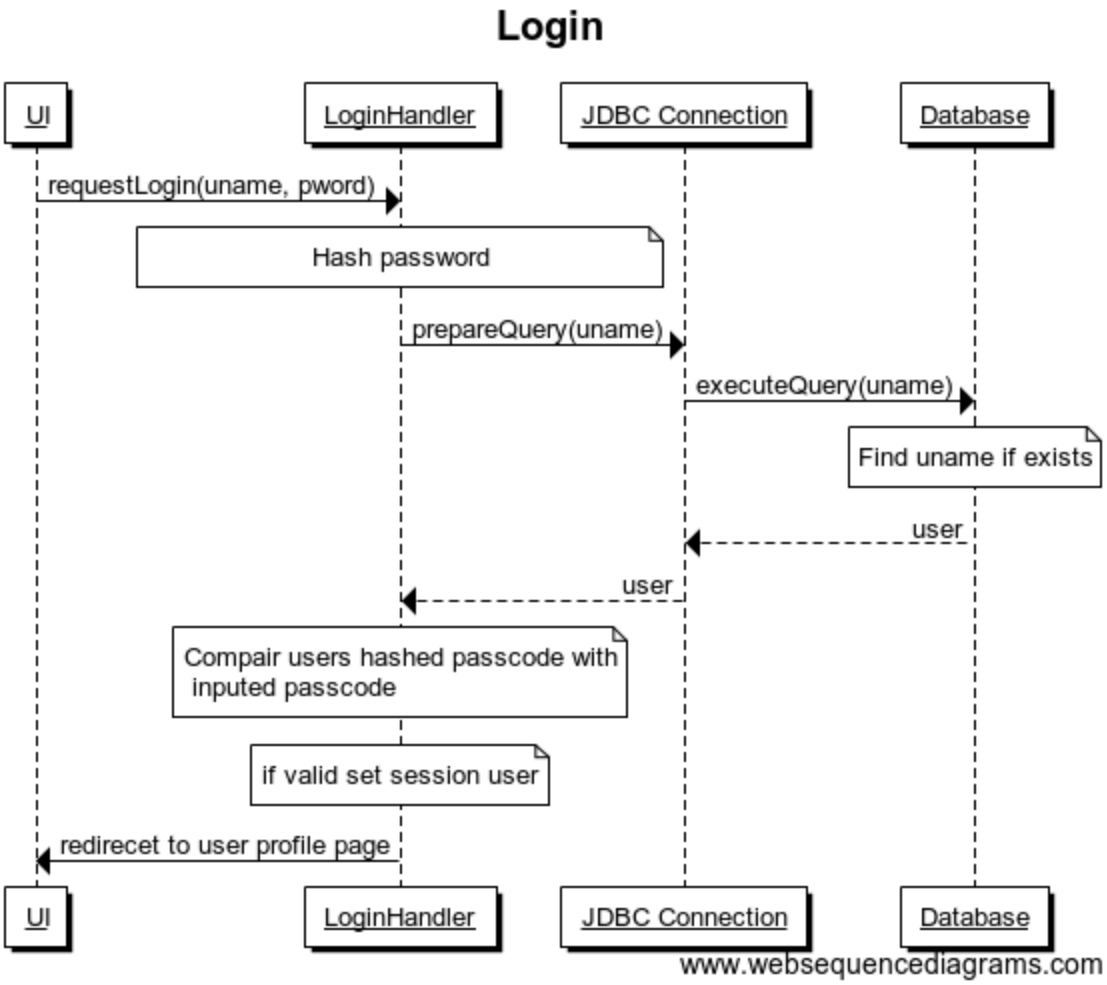
5.4.2. Create New User



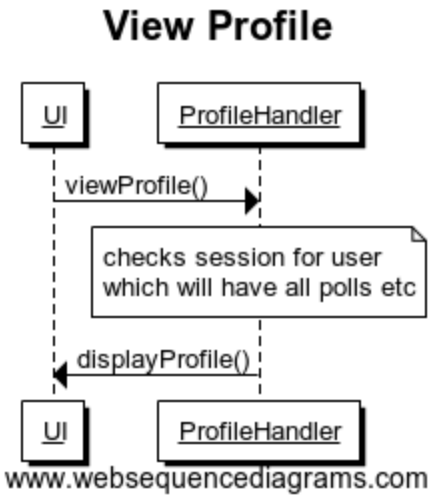
5.4.3. Create New Poll



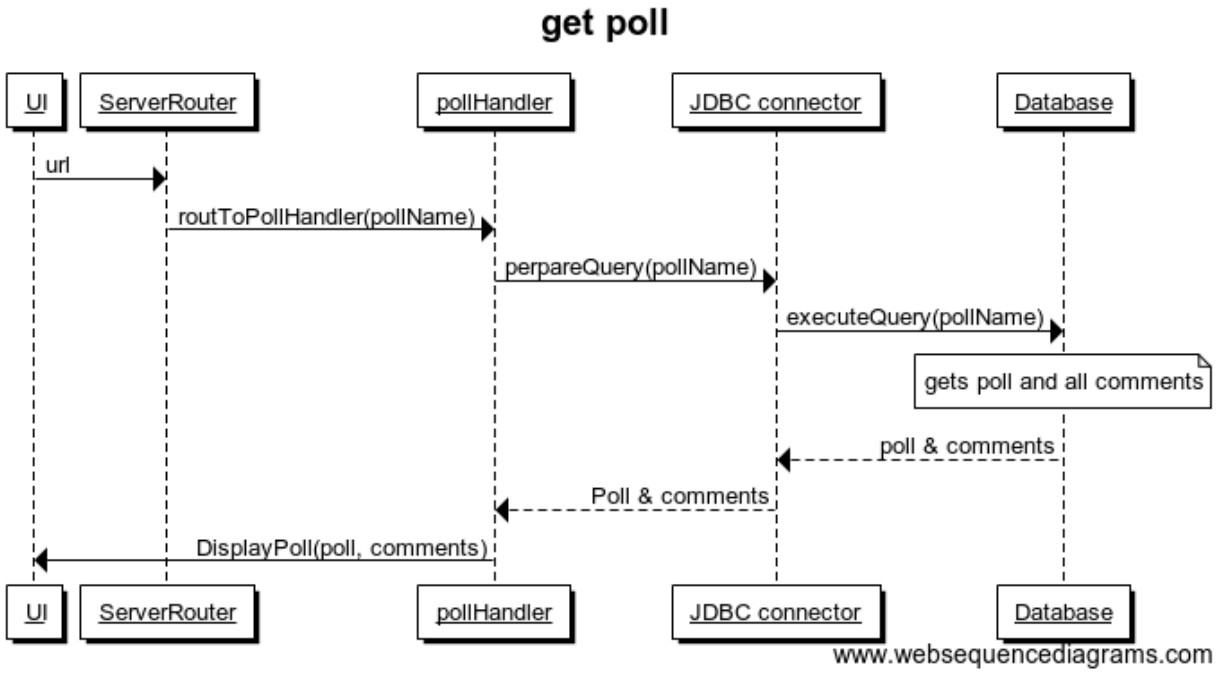
5.4.4. Login


Login
UI
LoginHandler
JDBC Connection
Database
requestLogin(uname, pword)
Hash password
prepareQuery(uname)
executeQuery(uname)
Find uname if exists
user
user
Compair users hashed passcode with inputed passcode
if valid set session user
redirecet to user profile page
UI
LoginHandler
JDBC Connection
Database
www.websequencediagrams.com

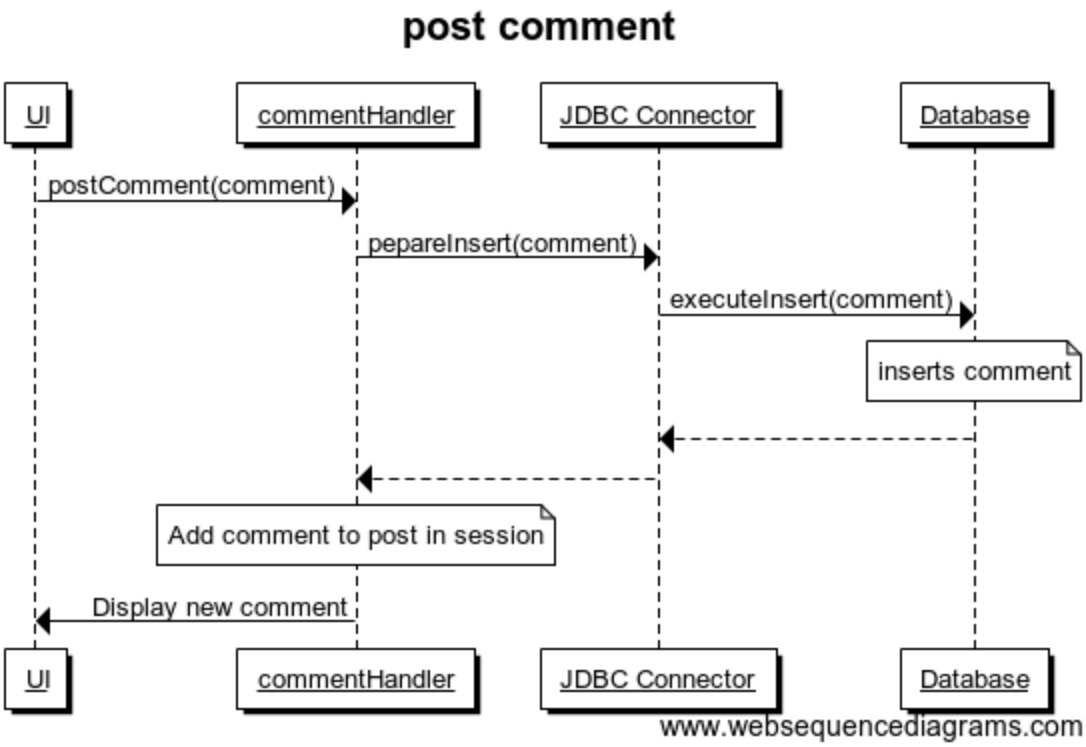
5.4.5. View Profile



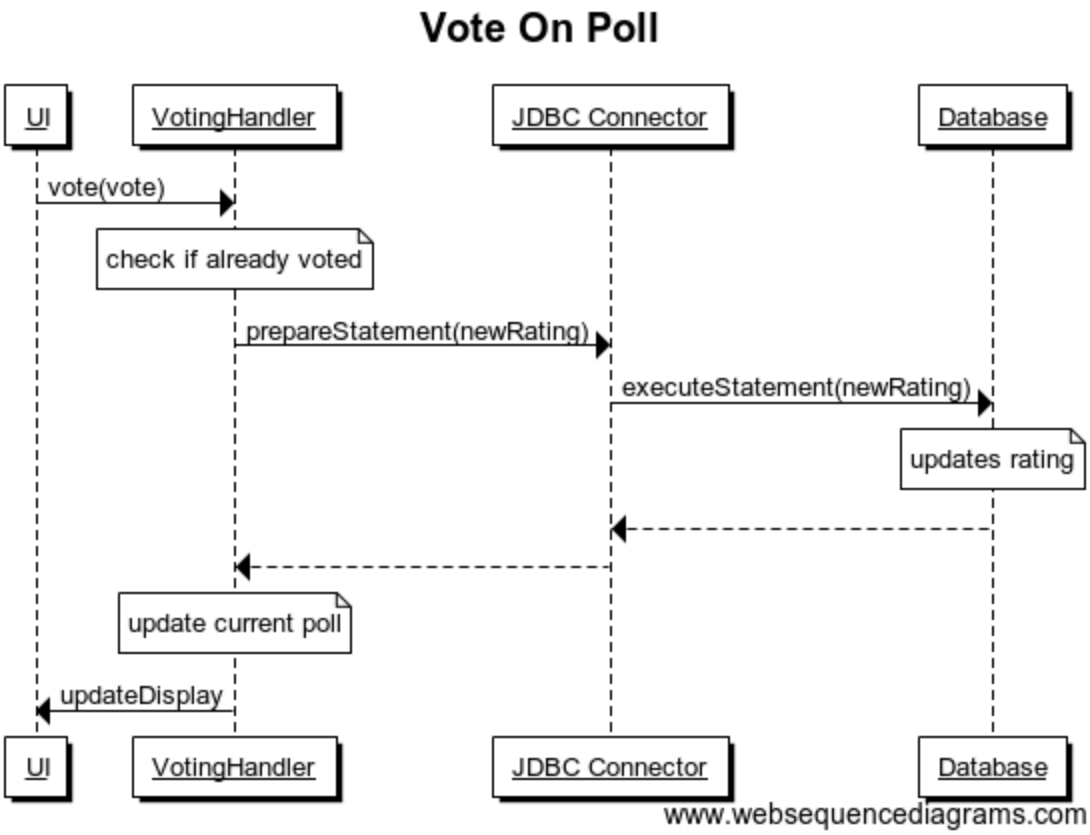
5.4.6. Get Poll



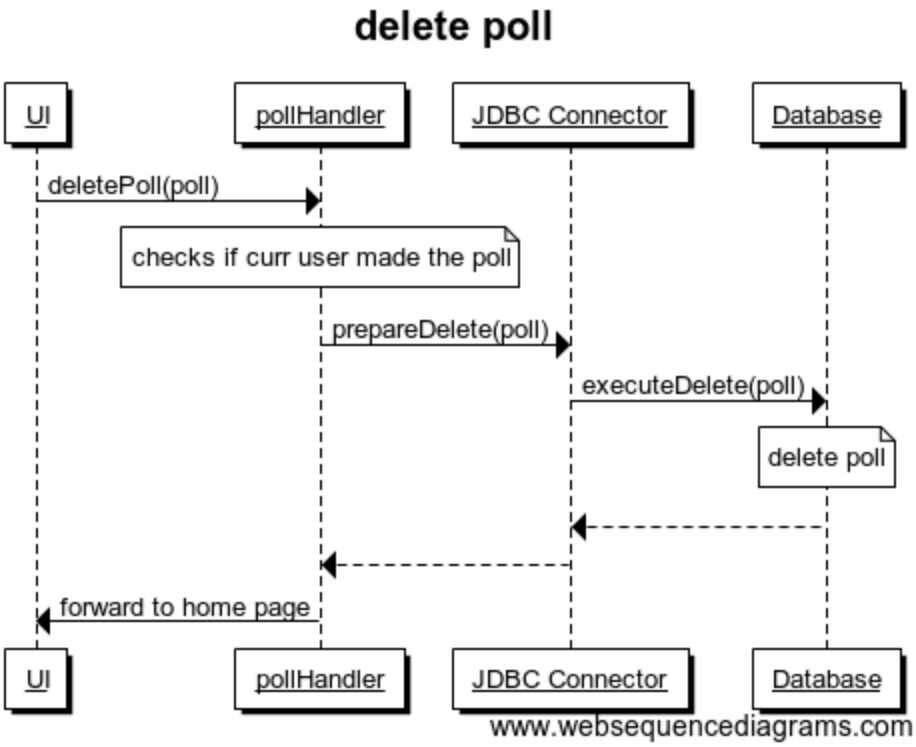
5.4.7. Post Comment



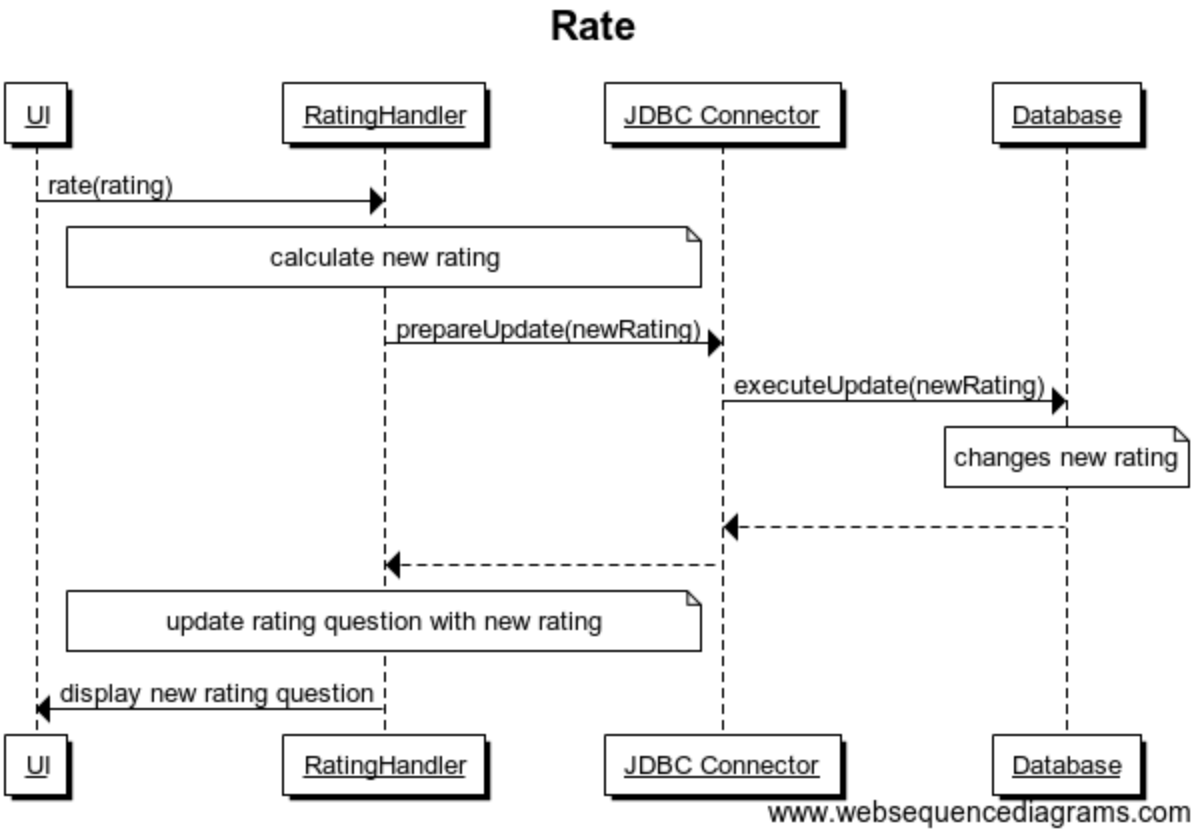
5.4.8. Vote On Poll



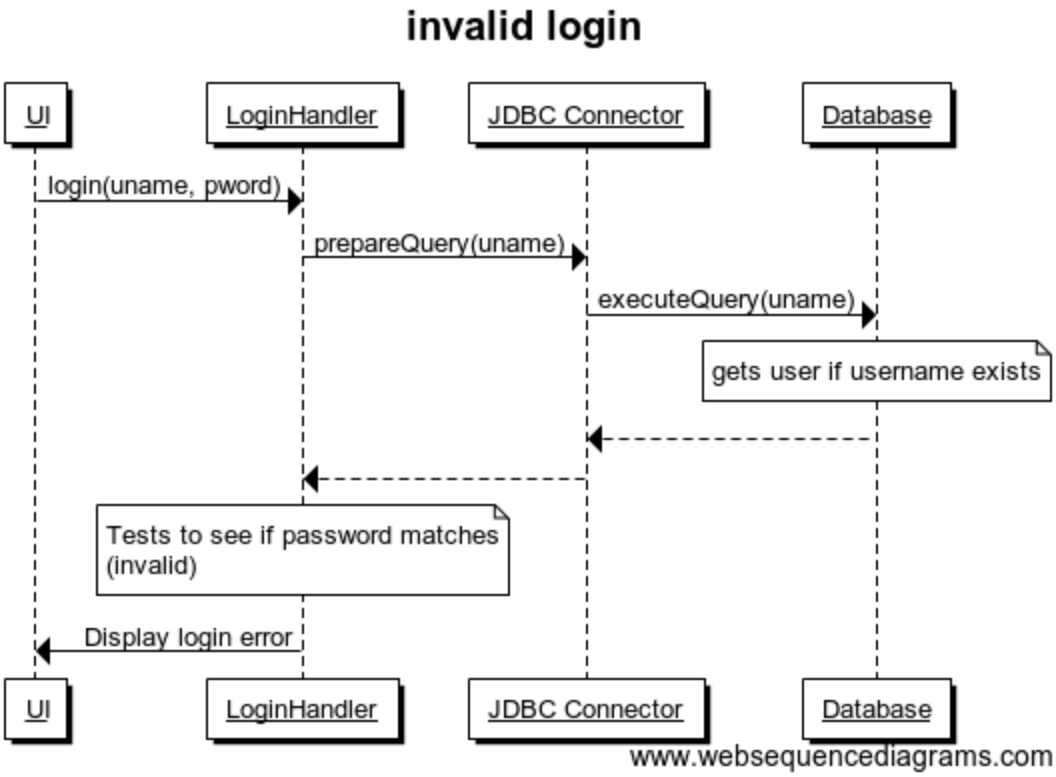
5.4.9. Delete Poll



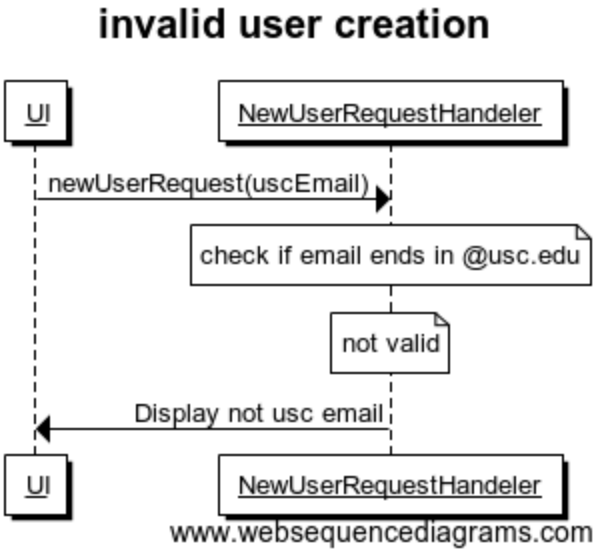
5.4.10. Rate


Rate
UI
RatingHandler
JDBC Connector
Database
rate(rating)
calculate new rating
prepareUpdate(newRating)
executeUpdate(newRating)
changes new rating
update rating question with new rating
display new rating question
UI
RatingHandler
JDBC Connector
Database
www.websequencediagrams.com

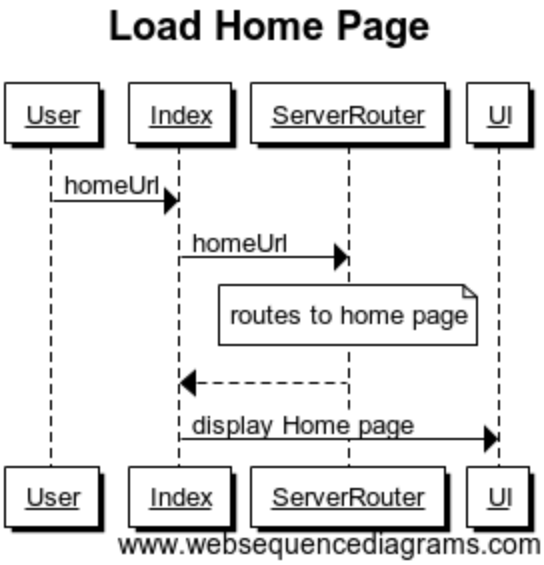
5.4.11. Invalid Login



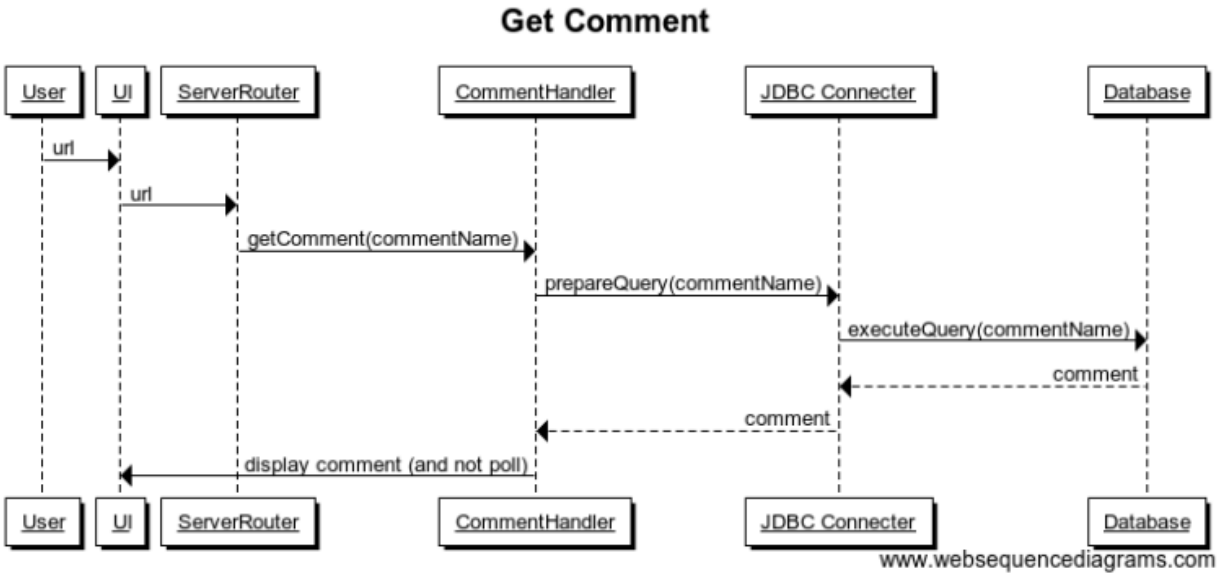
5.4.12. Invalid User Creation



5.4.13. Load Home Page



5.4.14. Get Comment



5.4.15. Delete Comment