HumorMe! – **A Social Media Platform**

Oct 6th, 2023

SHRISTI SHRESTHA

Hosted at: http://3.143.17.104:3000

Demo:

Contents

Introduction

HumorMe! is a fun community space for people to share their sense of humor through jokes. Like the name suggests (HumorMe!), the idea is to spread humor and offer a virtual place of belonging for people. With rise of memes and funny reels in apps like Facebook and Instagram, there is a huge spike of users who consume exclusively funny content every day. The existing applications like Facebook, Reddit, Instagram, and Twitter are becoming extensively heterogenous in their content for example, politics, health, messaging, and others. This web app, HumorMe! exclusively caters to the humor content and can be thought as a starting point for aspiring comedians. Based on this idea, the webapp showcases a proof-of-concept designed for anyone. The app is hosted at at http://3.143.17.104:3000 and offers an easy-to-use public space to post, comment and rate jokes. Users can browse through jokes anonymously and are required to sign up to engage in activities such as create jokes, comment, and rate jokes from other users. There are profile pages set up for each user who signed up and users can follow each other for customized content in future.

HumorMe! is a three-tier web application served by three individual servers: Apache serving client app, tomcat serving backend server (serving RESTful APIs), and a Postgres data server. The Apache server serves the presentation layer to this app and is composed of both static and dynamic content delivered to the browsers. The client content is developed and build using NextJS framework which supports/compiles JSX into HTML content. The client app is hosted at http://3.143.17.104:3000. We used SpringBoot framework to build backend server which exposes RESTful APIs and is hosted at a separate EC2 instance (check http://3.141.47.20:8081/swagger-ui.html). SpringBoot is a widely popular framework to build microservices that also builds an embedded Apache Tomcat server with the Java Servlets. We used this embedded Tomcat server to deploy our backend server. Postgres server is hosted at the same EC2 instance as backend server and are separated logically. User interactions such as page load, click buttons, submit forms trigger API calls (HTTP) to the backend server. The backend server processes API requests, run data query in Postgres and returns the data in JSON format to the users' browsers.

Related Works

System Architectural Design

The system architecture follows the general three-tier web application design as described earlier in the Introduction section. There are two Amazon EC2 instances launched each serving a backend server (including a Postgres server), and the client server respectively. The first instance runs the Apache Web Server hosting the static web content at its root and can be accessed at http://3.143.17.104:3000. The second instance runs embedded Apache Tomcat server containing the Java Servlets. A Postgres server is installed in the same instance and configured to allow access to the SpringBoot application through a Java Database Connector (JDBC) library. The backend server is hosted at http://3.141.47.20:8081/ and its RESTful APIs can be accessed here "/swagger-ui.html". The client app makes HTTP calls to the RESTful APIs directly by using the public URL of the EC2 instance. Deployment guidelines are provided in detail with the source codes.

The clientside application is developed using NextJS framework which supports React (JavaScript XML, JSX) and provides configuration for building static content from JSX. For design, we used Ant Design (Antd) Library which provides high level components such as button, form, menu, dropdown, layout, and so on (https://ant.design). We also used "axios" to make API calls to the backend server (https://www.npmjs.com/package/axios). The libraries are installed using Node Package Manager (NPM) which is supported by NextJS. Any dynamic content required to populate the web pages undergo the process of making API calls to the server and inserting the responded data into the appropriate JSX elements. Since the backend and the client servers are hosted at separate domains, we also enabled Cross Origin Resource Sharing (CORS) functionality in the backend to allow the requests from the clientside. In other words, before making API calls using methods like POST, GET, PATCH, PUT AND DELETE, the browser makes an additional call "OPTIONS" to the server to check whether the server resource can be served in different origins (server and client hosted at two different host origins). The server responds with a header indicating that the requested resource can be served in different origin. Once this communication is established between the server and the client, the browser makes the APIs calls to access the resources and server responds accordingly. In the clientside, we have used three types of storage: local storage, redux and state variables to store data from the server. Local storage holds data in the browser and lives even after the browser is closed. Redux stores the data throughout the session and is removed/reloaded after page reload. State variables store values within each component level and are removed once the DOM removes the component from the browser.

Before we dive into the details of our RESTful APIs, let's investigate the data models used in our Postgres server. Postgres Server is a relational database management system for persistent storage and ease of integration with Java Servlets using JDBC library. Figure 1 shows the Entity Relationship model used in our HumorMe! server.

Assumptions and constraints besides the cardinalities specified in the diagram are:

- All columns in tables are nullable except the primary keys
- All primary keys hold unique constraints and throw error when violated
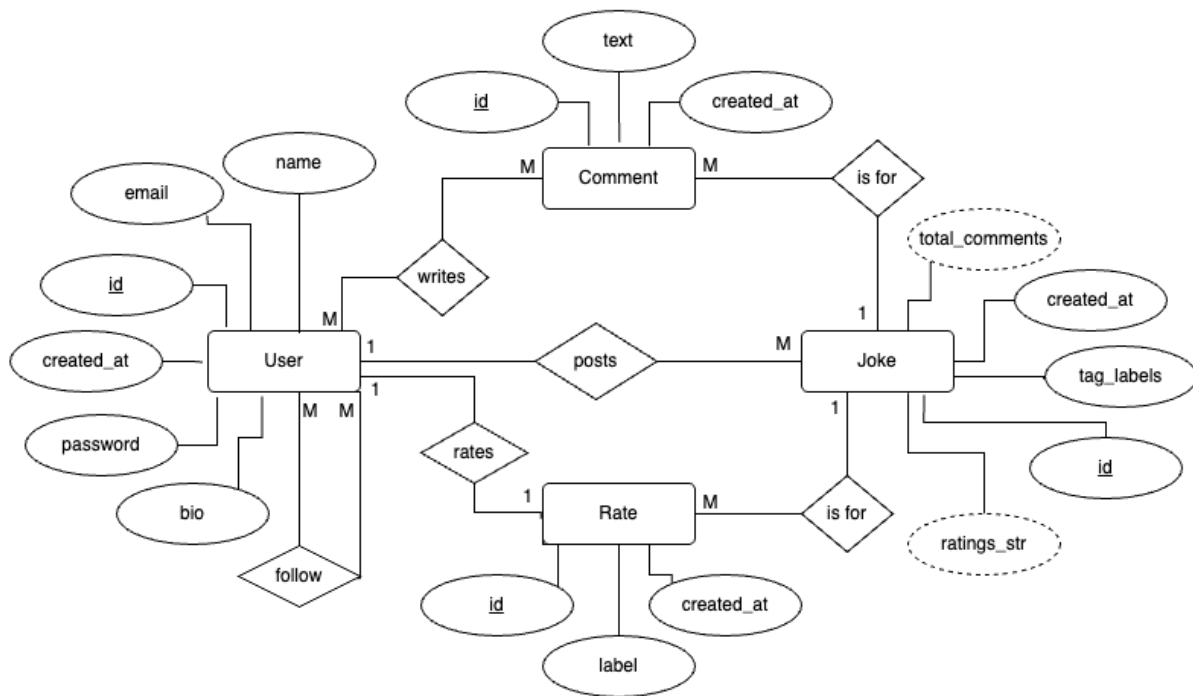
Figure 1 Entity Relationship diagram

The entities presented in the Figure 1 above are represented by entities in SpringBoot. We used @Entity annotation in each model class that implicitly transforms the Java Classes into relational tables without requiring us to write the SQL for table creation (unless we are dealing with version control at schema level, which is not used in this implementation). We also used Java Persistence API (JPA) library, a widely used Object Relational Mapping (ORM) specification, to make queries such as insert, update, delete and select into the database. In what follows, we describe the relational tables generated using our data model classes in Java.

Table: users

The table stores the information of users once they sign up in the platform. A unique identified of type Integer (Long in Java) is created automatically when an insert query in this table is issued. A unique constraint is added to the column "email" so that users with same email address are prevented from registering as users in the platform. The password column stores the hash value (using BCryptPasswordEncoder provided by SpringBoot Security library) of the raw password users used in their registration process.

Table: jokes

The table stores information details of joke related posts from users. Here, text and labels are directly supplied by the users, and each joke is associated with a single user (one to one mapping). The table stores the summary of each label ratings (lame, dark, funny, and hilarious)

inside column "ratings" as a string value and are transformed into key values when returning the data to the client. The column "totalComments" stores the total number of text reviews commented for the joke and is dependent on the number comments for the joke.

Table: comments

The table stores detail information related to a text review commented on a particular joke. It has a one-to-one mapping with a user and a joke. In other words, a comment is only made by a single user for a single joke.

Table: ratings

The table stores label rating information by a user to a joke. Therefore, it also has a one-to-one mapping with a user and a joke. A label rating replaces the existing "like or thumbs up or star-based" ratings we have seen in other existing platforms like Facebook and Reddit. Here, a user can rate a joke using one of the four rating labels "lame, funny, hilarious and dark". This label is not the same thing as labels when posting a joke. In creating a joke, we add custom labels which is equivalent to saying "search tags" for the joke.

Table: user_followers

The table stores information about who is following whom. In other words, it stores two columns: "user_id" and "follower_id". Here, "follower_id" indicates the users who are following the given "user_id". This way, we can keep a track of the number of followers and the number of followings for each user. Followers indicate other users who are following a given user. Following indicates other users to whom a given user is following. Since this is a many-to-many relationship between the users, the same "user_id" is allowed to be repetitive in the table.

Detailed Description of Components

In what follows, we describe each component of the client app the users can interact with in the platform. Each shown component requires at least one direct API call to the server.

Account Registration and Login

There is a "Account" button at the top of the page. Users can sign up by submitting a username, their email, and a password. Once signed up, they can sign in by going to the "Sign in" tab. Once, signed in, users can actively engage in creating, rating, and commenting on jokes. There is a "Sign out" button that logs the users out of the session. Once signed in, users need not worry about re-sign in even after they close the browser.
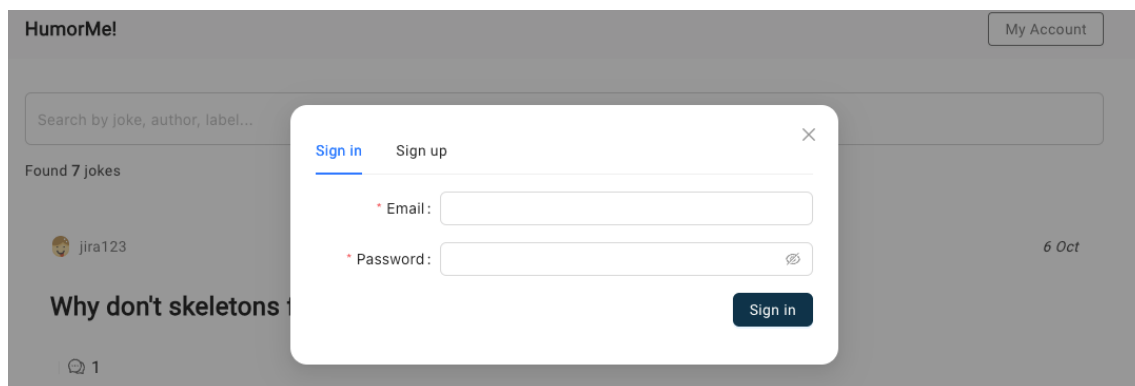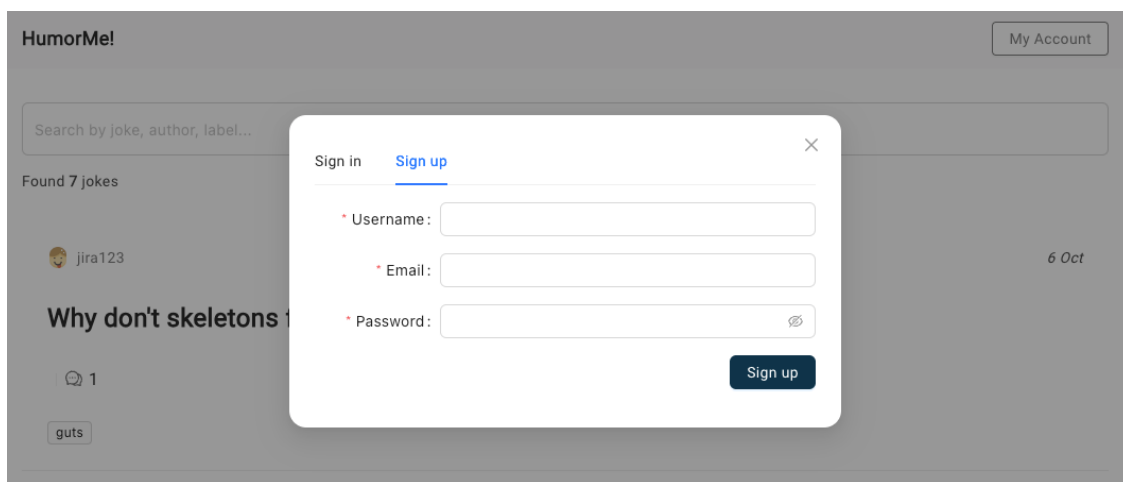
View and Search Joke Posts

The home page shows the list of jokes sorted in recent order. Users can anonymously read the jokes and view the comments of the jokes. There is a search input at the top of the list where users can search joke by typing their query. A query is a text that is used to match with jokes' texts, users' names, and jokes' labels (tags). The search input also allows clear which resets the jokes list. There is a specific page for each joke, and underneath the joke, there is a list of

comments posted by users (including the joke creator) for the given joke. If the user is logged in, they can rate and comment on the joke as well.
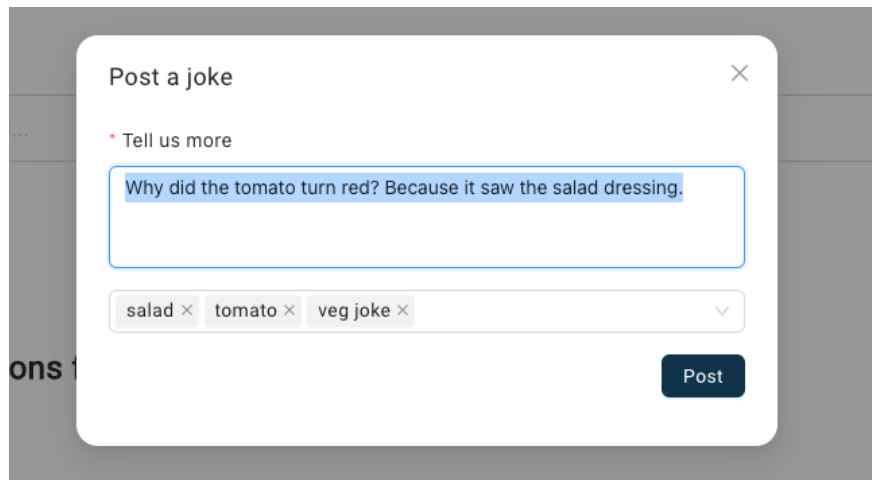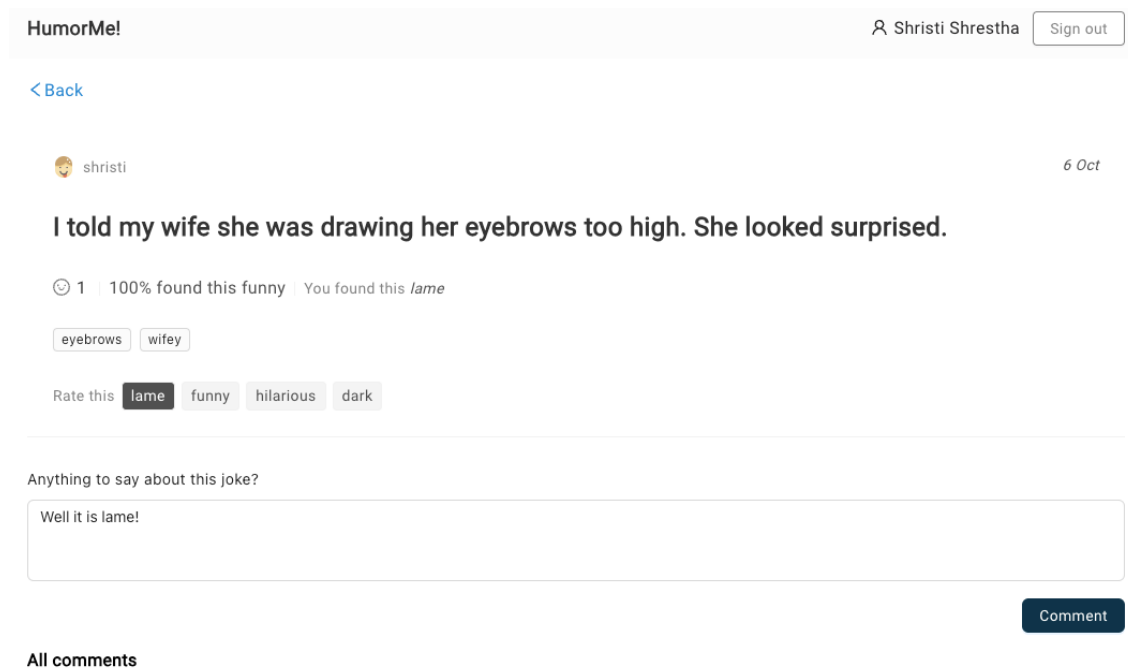

Post a Joke

Users can create a post telling their joke by submitting a form as shown below. The label tags are optional and are used as a search keywords only. A logged user is only allowed to create a post and can write as many jokes as he/she wants. All the jokes a user has created are available in their profile page as well.

Comment on a Joke

Rate a Joke

Rating a joke is an equivalent of sending likes or thumbs up in other social media platform. In here, since this is a humor space, we have defined four custom labels that can be used to rate a joke. The rating labels are lame, funny, hilarious, and dark. In the Figure below, the left and the right side show the before and after effect of rating a joke. Only the users who are logged in can rate a joke. Also, the creator is not allowed to rate their own jokes.

Delete a Joke

Users can delete their own joke by clicking the delete button and confirming their delete action. Once a joke is deleted, all the comments and ratings associated with the joke are also removed. The delete action in a joke post is only available when the user visit their own profile.
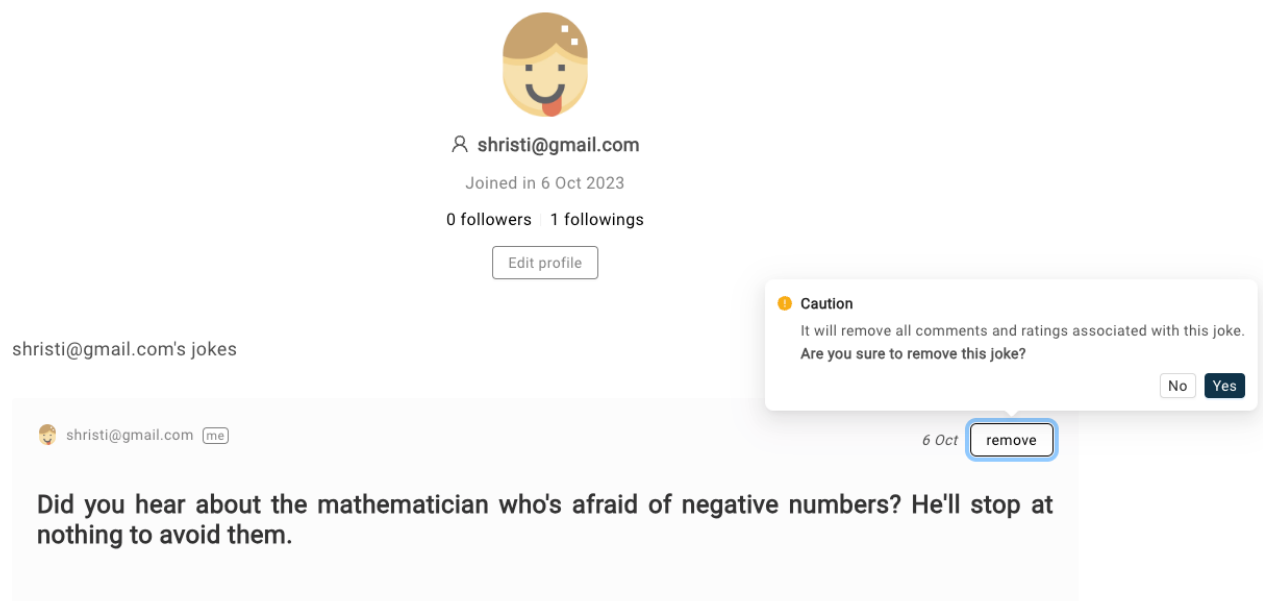
User Profile

Users can visit other users' profile by clicking the username shown at the top of the joke. In other user's profile, a user can see the jokes created by that user, as well as follow/unfollow them. A user can also visit their own profile by clicking their username shown at the top right side of the page. In their own profile, they can edit their username and bio information, as well as see the total number of their followers and followings. The following three figures show the profile visit anonymously, visit other user's profile, and visit own profile respectively.

HumorMe!   My Account

< Back to home

A test

Joined in 4 Oct 2023

1 followers   0 followings

test's jokes

test   4 Oct

fasfsf

☺ 55   65.45% found this funny

fasfsaf   fsdfsdf

HumorMe!   A shristi@gmail.com   Sign out

< Back to home

A test

Joined in 4 Oct 2023

1 followers   0 followings

You don't follow this person   follow

test's jokes

test   4 Oct

fasfsf

☺ 55   65.45% found this funny

fasfsaf   fsdfsdf

Rate this   lame   funny   hilarious   dark

HumorMe!   ⚲ shristi@gmail.com   [Sign out]

< Back to home

⚲ **shristi@gmail.com**

Joined in 6 Oct 2023

0 followers   1 followings

[Edit profile]

shristi@gmail.com's jokes

shristi@gmail.com [me]   *6 Oct*

Why did the bicycle fall over? Because it was two-tired.

[#tired]

Conclusion

The current implementation of HumorMe! app delivers a fully functional three-tier web application hosted at http://3.143.17.104:3000 that creates a fun community space for people to share their sense of humor. The service is relatively free of bugs with no observable errors under normal usage and is responsive with mobile browsers as well. The main area for future improvement is the content moderation to prevent people from posting offensive content to communities, gender, or specific race. There are other features that could also boost this platform use cases, such as creating space of aspiring comics and allowing them to inform others about their upcoming events. For now, the current deployed version is just enough to present the core idea (minimal viable product or MVP) of HumorMe! vision and what the future holds for this platform.