

San Jose State University

Software Engineering Department



CMPE 273 - Enterprise Distributed systems

Project: Reddit Prototype

Instructor: Dr Simon Shim

Submitted By

Group 6

Name	SJSU ID
Divya Mittal	015351349
Kiran Bala Devineni	015218866
Naitik Vajani	015270710
Nikhil Raj Karlapudi	014668121
Saketh Reddy Banda	014843881
Vinay Guda	015255123

Work Partition:

- **Team Member 1 (Divya Mittal) :**

Created login page with validations • signup page with validations • logout page for Reddit with validations • Designed database for SQL and MongoDB • MongoDB model creation • S3 Bucket setup for image upload • Create Community • Edit Community • Delete Community • My Community page to show all the communities created by the owner • Different sorting implementation for myCommunity page • Upload multiple images using s3 • Setting up Kafka for the application • Kafka Migration for multiple api's • Dark Mode implementation for the pages implemented • Community Analytics page to show different graphs related to a community • Messaging app • Redis client caching • Styling for the application using bootstrap and material ui • JWT token implementation for the APIs • Pagination for different components • Worked on load balancer • AWS configuration of application • Complete end to end testing of the application with multiple bug fixes • Mocha test cases

- **Team Member 2 (Kiran Bala Devineni) :**

Created APIs for • community search • user profile details • Developed the Community Search Page which required: • showing the list of all the communities with most recently created first • Implemented both regular and load more on scroll pagination i.e., infinite scroll for community search results • upvote or downvote a community • sort communities by created at, most number of users, most number of posts, most upvoted posts in ascending and descending order • Navigation bar search field functionality in community search page • search for a community by name • Worked on the User Profile Page which involved: • Navigating to a particular user's profile page on clicking other users' profiles displayed in search page • showing the user's profile details like their picture, name, location preferences and the communities they are part of • Developed the User Interface for both community search page and user profile details page • Migration of community search and user profile details APIs into Kafka • System Architecture design

- **Team Member 3 (Naitik Vajani):**

Created APIs for • creating a post • removing a user from a community • sending the invites • showing the status of invites for moderator • accepting an invite • rejecting an invite • getting all the posts for dashboard • Community Analytics • Add/Edit/Delete a topic • get communities for owner • add a comment • get communities created by a user • get communities details • check the user is the moderator for giving invitation page access • get Listed users details • worked on complex aggregate APIs for different pages • Developed Invitation page for community owner to send the invites • Notification page to show the invitation • accept/reject the invites • dashboard page to show the posts from all the communities in which the user is a member • Established the Redis structure • Worked on load balancer • AWS configuration of application • Complete end to end testing of the application for multiple bug fixes • Mocha test cases

- **Team Member 4 (Nikhil Raj Karlapudi) :**

Connection to Amazon RDS • setting up of sequelize ORM for MySQL queries • Schema design for MySQL and MongoDB • Created APIs for the entire community moderation page, Adding multiple selected users to a particular community which the user is the moderator of and rejecting multiple for the same • Removing multiple users from multiple selected communities and then deleted all the subsequent Posts and comments by those users in those communities • Pagination in community moderation page • Setting up Kafka skeleton and migrating API • Kafka support for JWT authentication • Multiple APIs for community analytics page and displaying graphs for the same • Complete UI/UX interactions for login, signup and navbar components • Dark mode support for pages implemented • Worked on end to end testing and multiple bug fixes • Mocha test cases. Improved UI and placement for multiple components in the application.

- **Team Member 5 (Saketh Reddy Banda):**

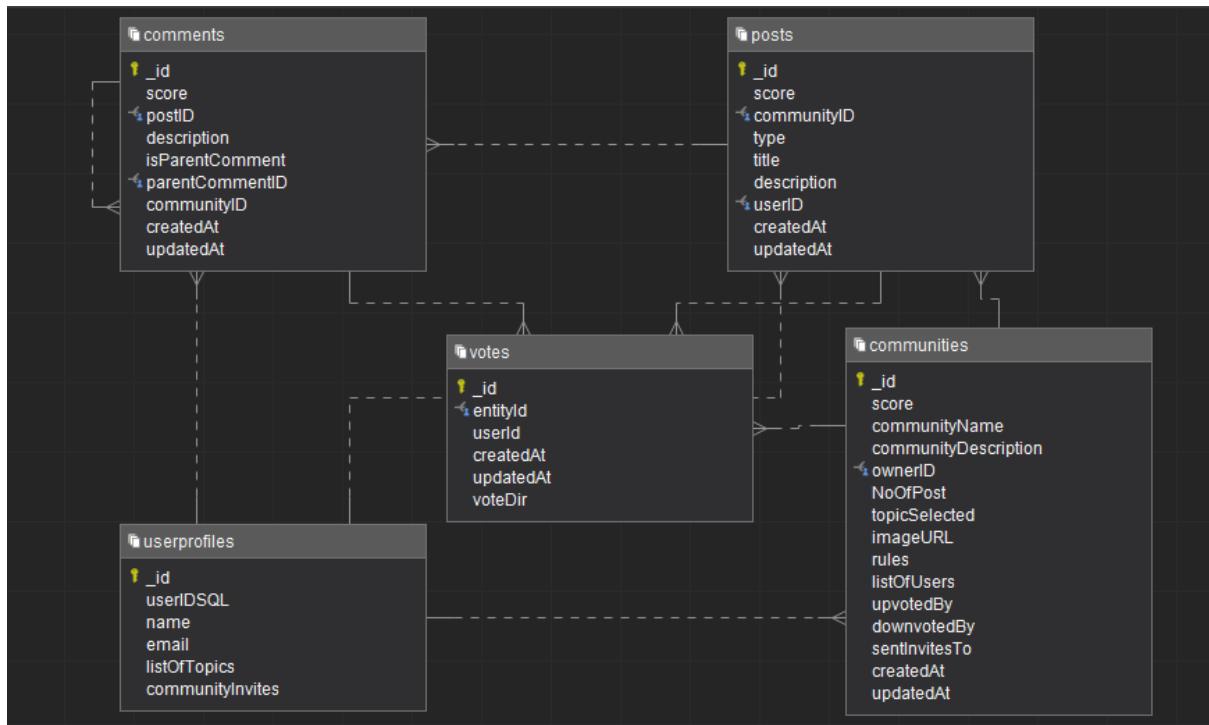
React express setup. Community page and create post UI. Navigation in React. Widgets in community page. Rules and community details. Database design and API documentation. Voting schema design for community, posts and comments. voting and commenting UI. Kafka backend for multiple apis. Post card and comments count implementation. Fast rendering of votes count which doesn't make the UI slow. Added Css using react bootstrap for multiple UI components. Wrote optimized Backend apis for getting posts , comments(sorting based on upvotes and tiebreaker as createdAt) and communities. Tried implementing redis for storing top posts to show to users for fast rendering of the UI. Dark mode styling for community page. Getting relative time for the date posted. Added scripts in package.json. Fixed some of the minor bugs and tested the entire application.

Team Member 6 (Vinay Guda) :

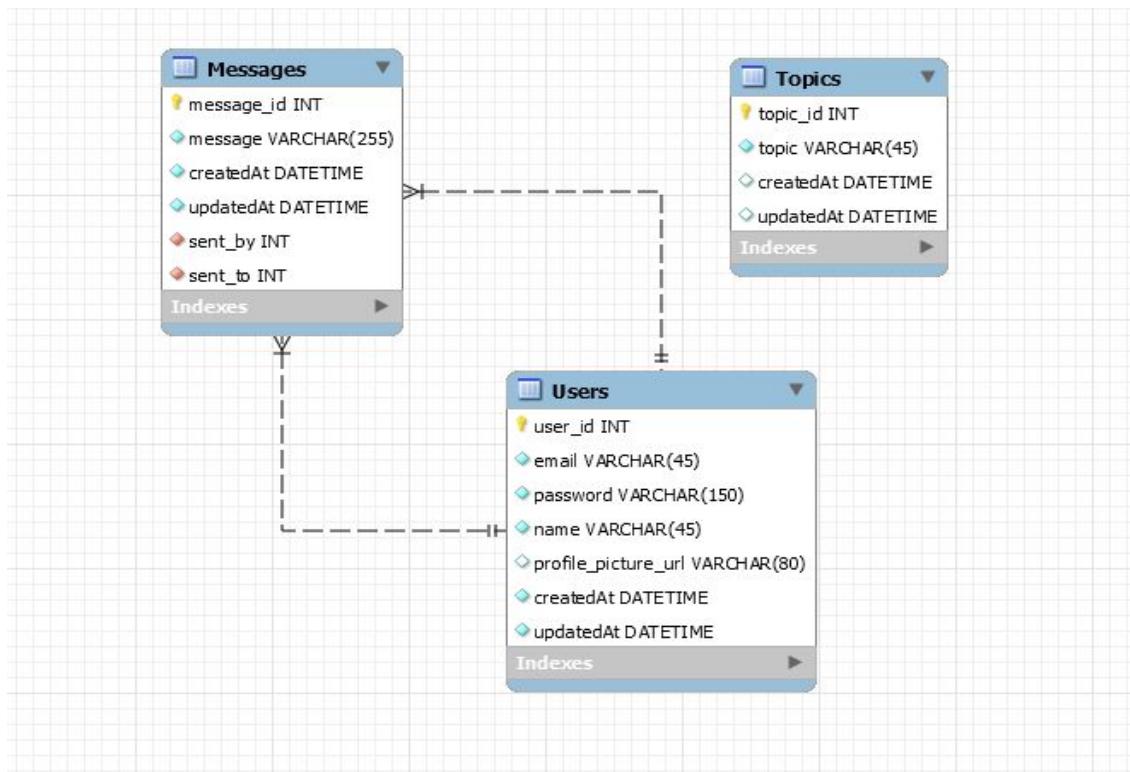
Established all pages base routing and navigation in React Frontend • Home page widgets • User profile screen • Add/Delete/Edit topic in the user profile • Community Home Page, Community Home Page Widgets and their interactions, Post, Detailed Post View, Comments and Comment thread UI Design and implementation. • Community Home Page header to show multiple variations based on user status(Moderator/Join/Leave/Accept, Reject/Request denied/Waiting for Approval/Invite Rejected) • Create Post Screen • Navigating to a particular user's profile page on clicking other users' profiles displayed in post, comment, community widgets • Dark Mode support with material theming. • Implemented application level loader to highlight the ongoing background API call's status with set and unset methods available to all pages • Mongo collection and connection • Error 404 page • No data available Reddit style UI screen in multiple screens • Reddit style randomizing user/community profile pictures with avatars. • Utils functions to calculate relative time. • Complete end-to-end testing of the application and bug fixes all around the application. • Implemented controlled voting behaviour on vote buttons to avoid uncontrolled clicks. • Kafka migration for 15+ APIs • AWS deployment and load balancing • Github branch maintenance - PRs from dev to main.

Database Schema:

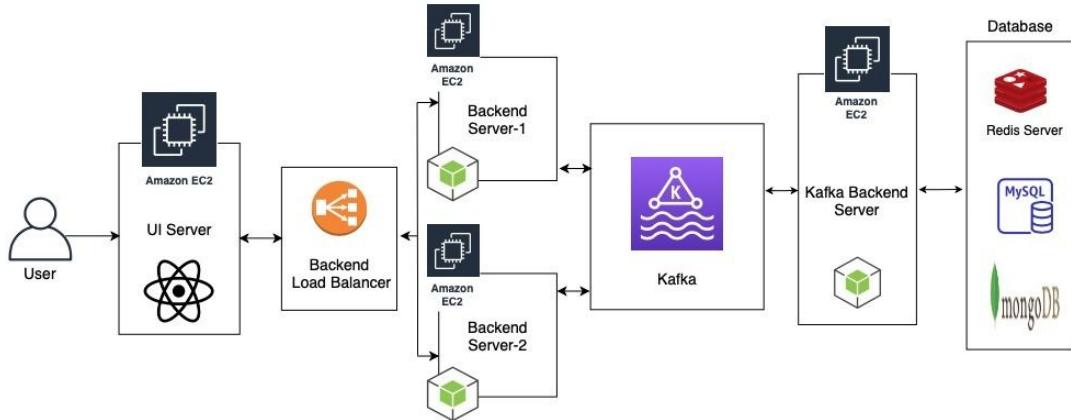
MongoDB:



MySQL:



System Design and Architecture:



- Object management policy:

- Community, Post, Comment objects were implemented in MongoDB so as to achieve vertical scalability on the fly. As the data inside the documents is ever increasing its suggestible to implement it in MongoDB. Before accessing any object in the document, the request is first authenticated and authorized through JWT and then the objects are returned. Images related to profile, community and posts are stored in the AWS S3 bucket. We are also making use of the life cycle policy to move the images which are not accessed frequently to S3 IA and if the images are not accessed for over a year, we move them to AWS Glacier.

- **Handling heavyweight resources:**

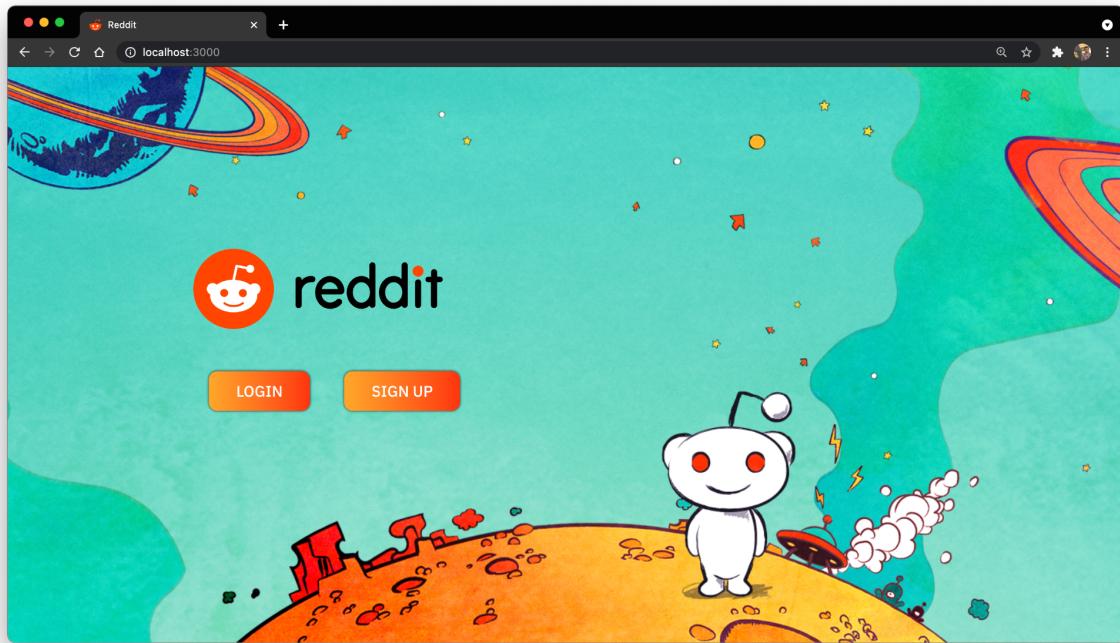
Read and Write operations on heavyweight resources take a lot of time and decrease the latency. So, we used Redis cache to store the heavy resources which are not changed frequently. For example: user profile is getting fetched at multiple locations to show the username and profile picture. We have added the user profile in Redis to minimize the load. However, data should not stay forever on Redis, therefore added a timeout for the data in Redis. Adding to that, heavy resources like images can be hosted on CDNs for low latency.

- **Policy for writing into Database:**

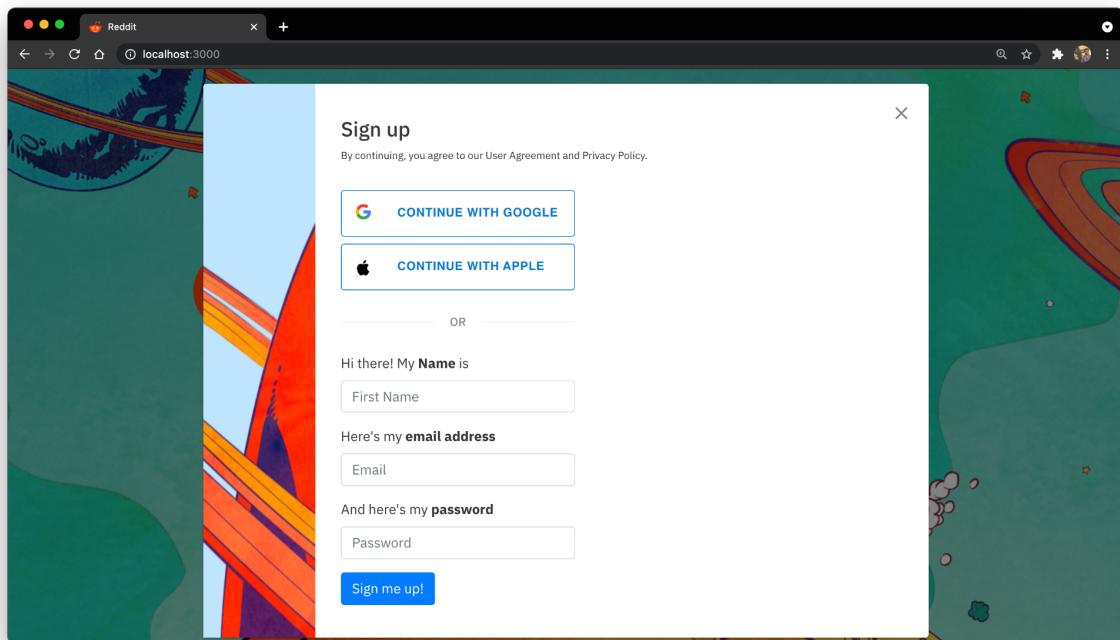
We used Redis Cache for faster access to the data. So, to avoid data inconsistency, we made sure to update the data in Redis after getting a successful response from the database. Also, instead of writing each vote, we can save them and do batch processing of votes on the database. We will update only the updated fields in the database whenever there is an update query involved.

Screen Captures:

Landing Page:



Sign up & Login Page:



My Profile Page:

Reddit localhost:3000/userprofile

Home Search user3

Your account



Your name: user3
Your email address: user3@gmail.com
Location (City/State):
Password:

Your gender: Male Female Other
Select topic: []

Bio: []

Save

Reddit localhost:3000/userprofile

Home Search user3

Your account



Your name: user3
Your email address: user3@gmail.com
Location (City/State): san jose
Password:

Your gender: Male Female Other
Select topic: Addiction Support [] music []

Bio: Hi I am user 3

Save

User Profile Updated.

A screenshot of a web browser showing a Reddit user profile editing interface. The main background shows a user account with a circular profile picture of a character. A modal window titled "Edit Topics" is open in the foreground. The modal contains a form with a "New topic name" input field containing "Algorithm". Below it is a list of existing topics: Politics, Activism, Addiction Support, dance, music, and Algorithm. Each topic has a pencil icon for edit and a red X icon for delete. To the right of the list are gender selection buttons: Male, Female, and Other. Below the gender buttons is a "Select topic" dropdown menu. At the bottom of the modal is a "Save" button.

Create Community Page

A screenshot of a web browser showing a "Create New Community" page. The left side features a large, colorful illustration of the Reddit logo character, Snoo, standing on a hill. The right side contains the form fields for creating a new community. The "Create a Community" section includes fields for "Name*" (with a note that names cannot be changed), "Topics" (with a "Select topic" dropdown), "Description*", and file upload fields for "Choose Files" and "Upload". Below these is a "Create Community" button. The right side also includes a "Rules" section with "Title*" and "Description*" fields, an "Add Rule" button, and a "List of Rules" section which is currently empty.

Reddit Create New Community Search

Edit a Community

Name* Community names including capitalization cannot be changed.

Topics
Select topic

Description*

Choose Files Upload

Rules
Title* Description*

List of Rules

Rule1	Rule1	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
Rule2	Rule2	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
Rule3	Rule3	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
Rule4	Rule4	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
Rule5	5th rule	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
6th rule	6th rule	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>
rule7	rule7 and final rule	<input type="button" value="Edit"/>	<input type="button" value="Delete"/>

My Communities Page:

Reddit My Communities Search

Sort By Descending

	Community Name: newcomm Description: as Total Users: 1 No of Post: 1 <input type="button" value="View More Details"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>
	Community Name: one_community Description: One Community description Total Users: 6 No of Post: 4 <input type="button" value="View More Details"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/>

Rows per page: 2 1-2 of 2 < >

Create Community
Name*

Reddit × +

localhost:3000/community/609f4ce013e08a4a6badd7f1

reddit Home Search

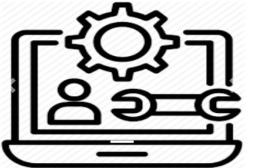
SoftwareEngineering Computer engineering Department Join

 r/SoftwareEngineering is a private community

You need to be a part of this community to see the posts.

[Home](#)

 r/SoftwareEngineering's images

 r/SoftwareEngineering's Rules

Rule 1: Enroll in the courses

 r/SoftwareEngineering's Interested topics

Algorithm

This screenshot shows the landing page of a private Reddit community named 'SoftwareEngineering'. It features a cartoon character icon, a message indicating it's a private community, and links to its images, rules, and interested topics.

Reddit × +

localhost:3000/createPost/609f634c759587644396c88a

reddit Home Search

Back to community 

[Post](#) [Images](#) [Link](#) [Poll](#)

TEXT BOX

desc

[Post](#)

 r/one_community's Rules

Rule1: Rule1
Rule2: Rule2
Rule3: Rule3
Rule4: Rule4
Rule5: 5th rule

 Posting to Reddit

1. Remember the human
2. Behave like you would in real life
3. Look for the original source of content
4. Search for duplicates before posting
5. Read the community's rules

This screenshot shows the interface for creating a post in a community. It includes fields for text, images, links, and polls, along with a 'Post' button. To the right, there are sections for the community's rules and guidelines for posting to Reddit.

Reddit localhost:3000/createPost/609f634c759587644396c88a

reddit Home Search

Back to community ↗

Post Images Link Poll

IMAGE POST

Choose File profile picture.png

Post

a potato

r/one_community's Rules

Rule1: Rule1
Rule2: Rule2
Rule3: Rule3
Rule4: Rule4
Rule5: 5th rule
6th rule: 6th rule
rule7: rule7 and final rule

Posting to Reddit

1. Remember the human
2. Behave like you would in real life
3. Look for the original source of content
4. Search for duplicates before posting
5. Read the community's rules

localhost:3000/community/609f634c759587644396c88a

Reddit localhost:3000/createPost/609f634c759587644396c88a

reddit Home Search

Back to community ↗

Post Images Link Poll

Title

url

Please enter a URL.

Post

r/one_community's Rules

Rule1: Rule1
Rule2: Rule2
Rule3: Rule3
Rule4: Rule4
Rule5: 5th rule
6th rule: 6th rule
rule7: rule7 and final rule

Posting to Reddit

1. Remember the human
2. Behave like you would in real life
3. Look for the original source of content
4. Search for duplicates before posting
5. Read the community's rules

localhost:3000/createPost/609f634c759587644396c88a#

Reddit × +

localhost:3000/community/609f634c759587644396c88a

reddit Home Search

one_community One Community description Moderator

Create Post

posted by u/one a few seconds ago
[LINK POST IS HERE]
<http://localhost:3000>

0 Award Share Save Hide Report

posted by u/one 8 minutes ago
[IMAGE POST]

Her: what social media do you have? 

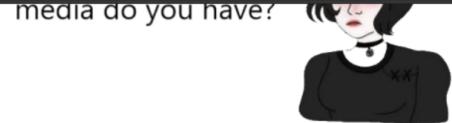
r/one_community's images

a potato

r/one_community's Rules

Rule1: Rule1
Rule2: Rule2
Rule3: Rule3
Rule4: Rule4
Rule5: 5th rule
6th rule: 6th rule
rule7: rule7 and final rule

r/one_community's interested topics

media do you have? 

Me: 

Rows per page: 2 1-2 of 4

0 Award Share Save Hide Report

Rule5: 5th rule

r/one_community's interested topics

Politics
Addiction Support
Activism
dance
music

r/one_community's Stats

Total Posts: 4
Total Users: 7
List of Users:

- u/one
- u/user3
- u/user4
- u/user7
- u/user8
- u/user9
- u/user10

Reddit × +

localhost:3000/community/609f634c759587644396c88a

reddit Home Search

one_community One Community description Moderator

Create Post

posted by u/one a few seconds ago
[LINK POST IS HERE]
<http://localhost:3000>

0 Award Share Save Hide Report

posted by u/one 8 minutes ago
[IMAGE POST]

Her: what social media do you have? 

Me: 

and 

Rows per page: 2 1-2 of 4

0 Award Share Save Hide Report

Rule5: 5th rule

r/one_community's interested topics

Politics
Addiction Support
Activism
dance
music

r/one_community's Stats

Total Posts: 4
Total Users: 7
List of Users:

- u/one
- u/user3
- u/user4
- u/user7
- u/user8
- u/user9
- u/user10

A screenshot of a Reddit post from the subreddit r/one_community. The post has been upvoted 4 times and is a [IMAGE POST]. The main image shows a woman with dark hair and a black top, looking slightly to the side. Below her is a smaller image of a man with blonde hair and a beard, also looking to the side. To the right of the man's image is the text "Me:" followed by a black cat icon and three orange arrows pointing upwards. At the bottom of the post, there is a text input field asking "What are your thoughts?" and a "Comment" button.

A screenshot of a web browser window titled "Reddit" showing a comment thread. The URL is "localhost:3000/community/609f634c759587644396c88a". The interface includes a header with "Comment" and "Report" buttons, and a sidebar with "Red topics". The main content area shows a comment by "u/one" with three replies, each with its own replies. The comments are timestamped and have upvote and reply buttons.

My Communities Moderation Page:

A screenshot of a web browser window titled "Reddit" showing the "My Communities" moderation page. The URL is "localhost:3000/mycommunitiesmoderation". The page has two main sections: "Communities" and "Users". The "Communities" section lists "r/one_community" with 3 requests. The "Users" section displays the message "No users to display!". Both sections include search bars and pagination controls.

A screenshot of a Reddit moderation interface. The window title is "r/one_community". The left sidebar shows "Community" and "1. r/one_community". The main content area has two sections: "User requests for joining the community" and "Users already part of the community".

User requests for joining the community:

- u/user3
- u/user4
- u/user7

Users already part of the community:

--- no data to display ---

Buttons at the bottom: "Accept" and "Reject".

A screenshot of a Reddit moderation interface, identical to the one above but with a different state. The "User requests for joining the community" section is now empty, displaying "--- no data to display ---". The "Users already part of the community" section now lists three users:

1. u/user3
2. u/user4
3. u/user7

Buttons at the bottom: "Accept" and "Reject".

Reddit

localhost:3000/mycommunitiesmoderation

My Communities

Search

Communities

1. r/one_community Number of requests: 0

Rows per page: 10 ▾ 1 of 1 << < 1 > >>

Users

1. u/user3

2. u/user4

3. u/user7

Rows per page: 10 ▾ 1 - 3 of 3 << < 1 > >>

Reddit

localhost:3000/mycommunitiesmoderation

My Communities

Search

Communities

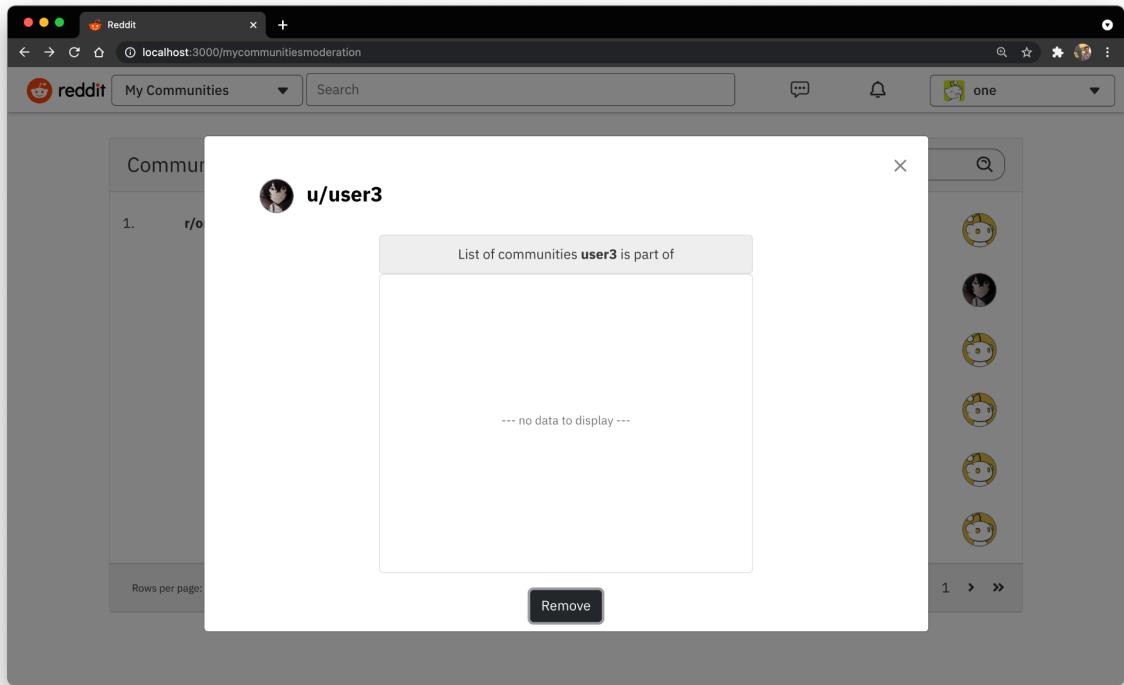
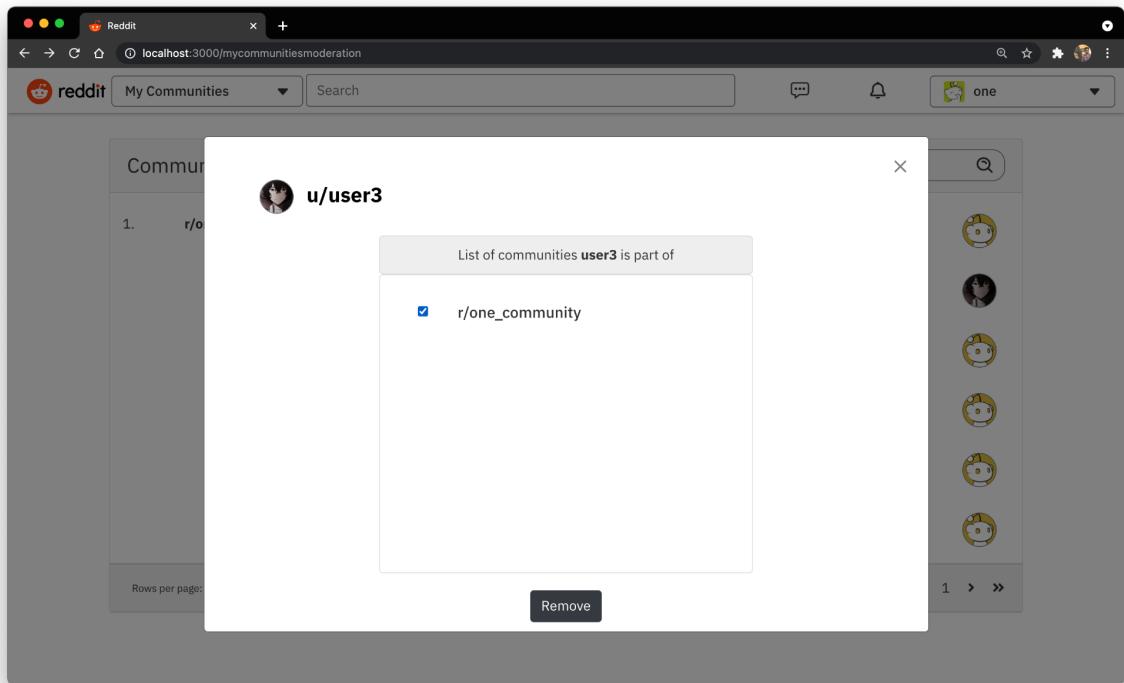
1. r/one_community

u/user3

List of communities user3 is part of

r/one_community

Remove



This screenshot shows the 'My Communities' moderation interface. On the left, under 'Communities', there is one entry: 'r/one_community' with 0 requests. On the right, under 'Users', there is a list of 5 users: u/user10, u/user4, u/user7, u/user8, and u/user9. Each user has a small profile icon next to their name.

Community	Number of requests:
r/one_community	0

User
u/user10
u/user4
u/user7
u/user8
u/user9

Community Search Page:

This screenshot shows the 'Community Search' page. It displays a list of communities sorted by 'Created At' in descending order. The communities listed are: SJSU, newcomm1, newcomm, one_community, and Community1. Each community entry includes its name, a thumbnail image, member count, post count, and creation details.

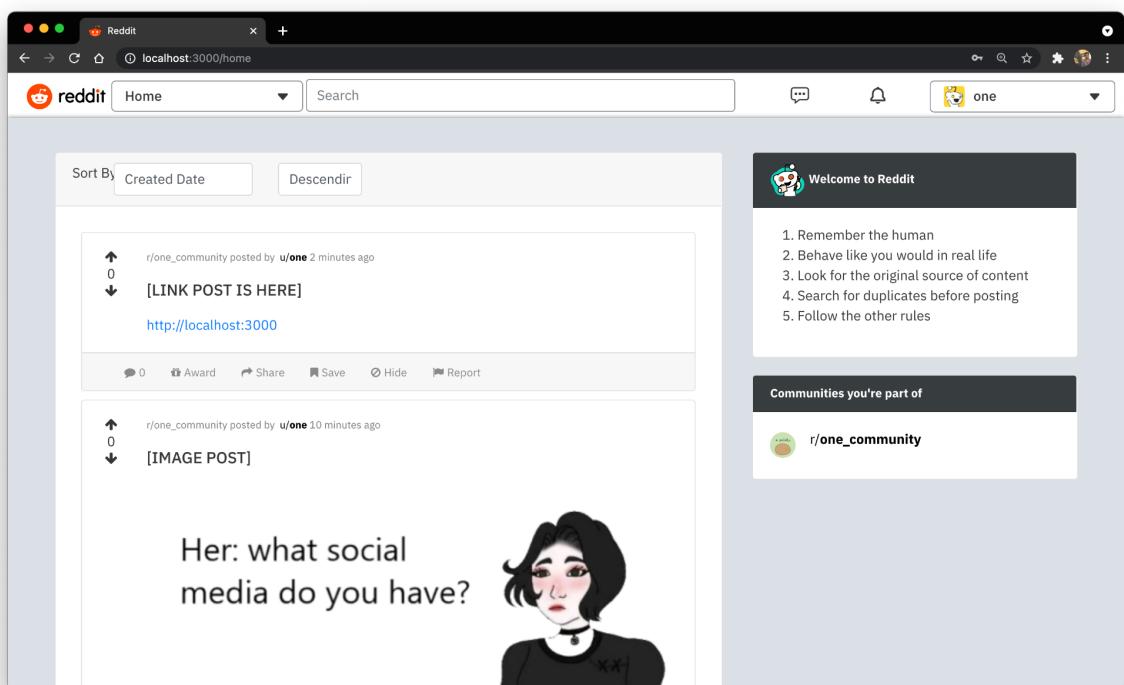
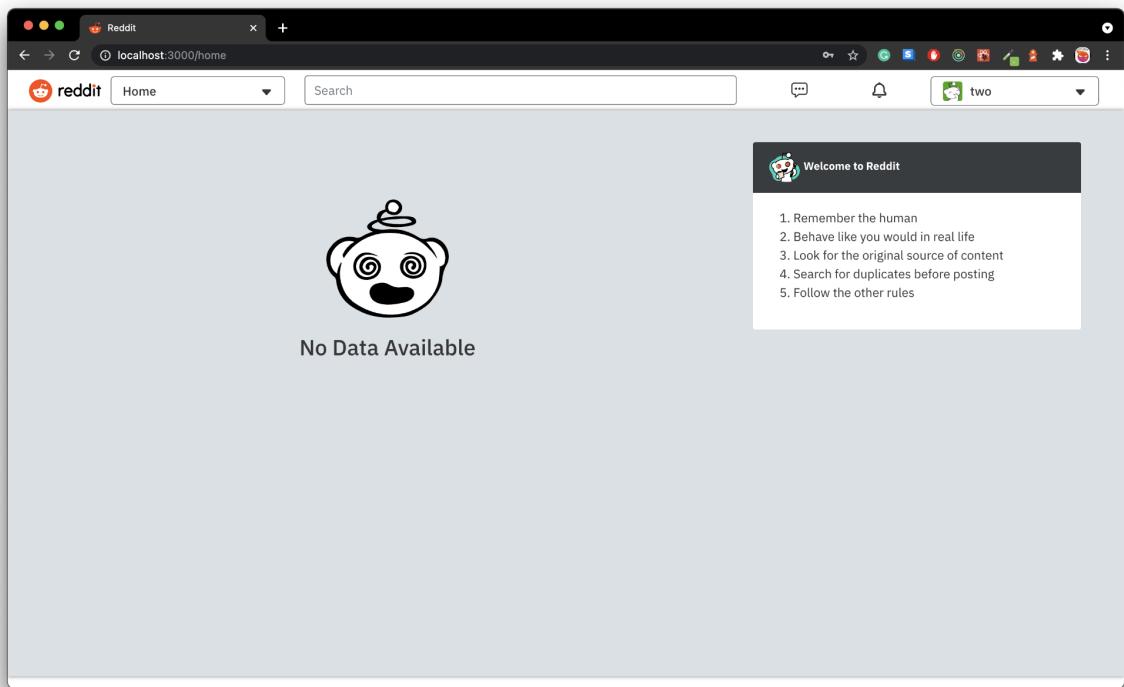
Community	Members	Posts	Created By	Created At
SJSU	1	0	user4	about a minute ago
newcomm1	1	0	user7	3 minutes ago
newcomm	1	1	one	11 minutes ago
one_community	6	4	one	34 minutes ago
Community1	2	0	user3	40 minutes ago

A screenshot of a local Reddit instance running on port 3000. The URL in the address bar is `localhost:3000/communitysearch?q=SJSU`. The search results for "SJSU" show one result: a post titled "SJSU" with a thumbnail image of a building, 0 upvotes, and 0 downvotes. It was created by user4 2 minutes ago. The interface includes a "Sort By" dropdown set to "Created At" and a "Descending" button. Below the results, it says "Rows per page: 5" and "1-1 of 1". The top navigation bar shows the Reddit logo, a "Community Search" dropdown, and a search input field containing "SJSU". The top right corner shows a user icon for "user4".

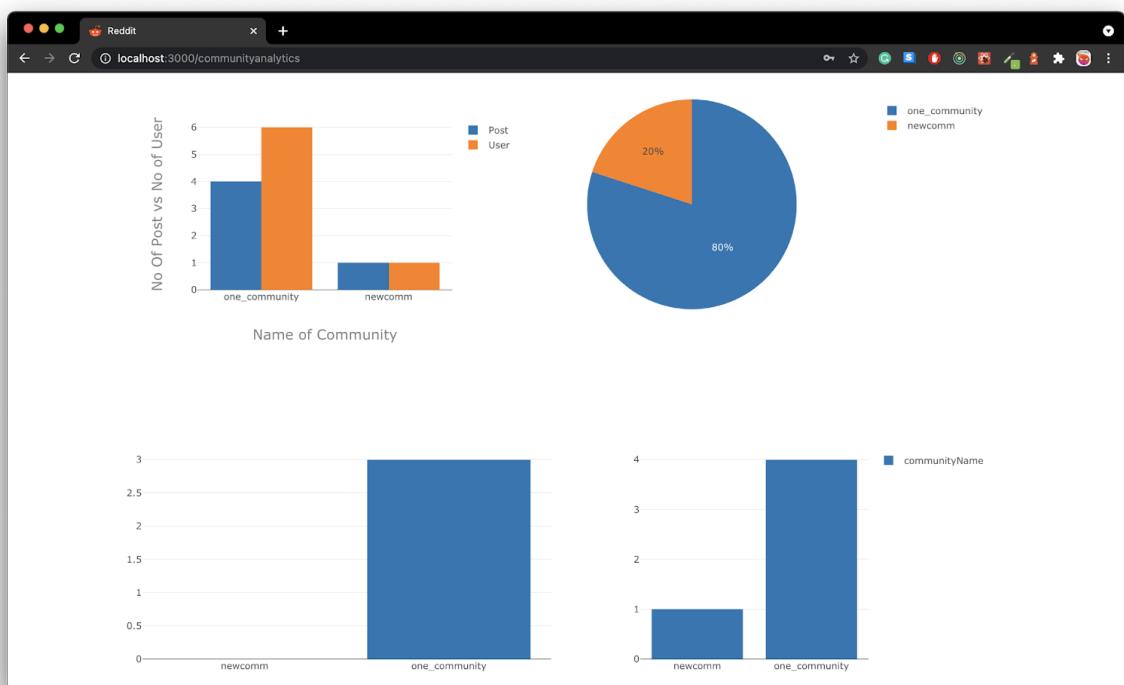
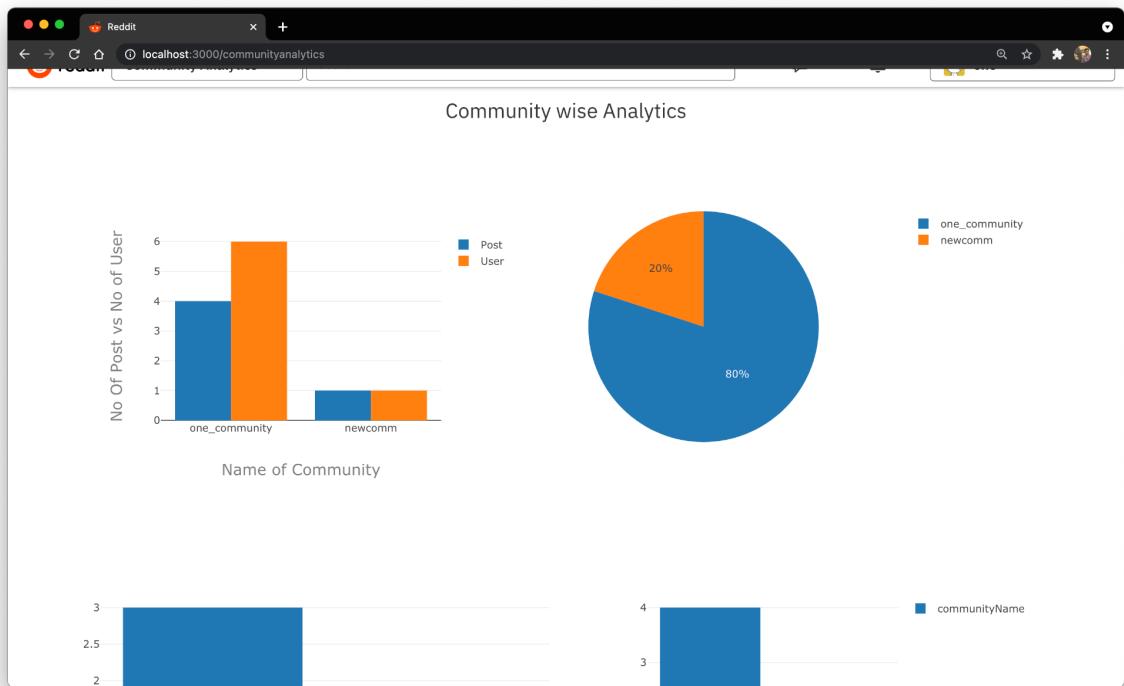
Community Home Page:

A screenshot of a local Reddit instance running on port 3000. The URL in the address bar is `localhost:3000/community/609f634c759587644396c88a`. The page shows the "one_community" subreddit. On the left, there's a post by u/one titled "[LINK POST IS HERE]" with a link to <http://localhost:3000>. On the right, there's a post titled "a potato" with an image of a potato. Below these, there's a post by u/one titled "[IMAGE POST]" with an illustration of a woman with short black hair and a white collar. The top navigation bar shows the Reddit logo, a "Home" dropdown, and a search input field. The top right corner shows a user icon for "one". A "Moderator" button is visible on the right side of the screen.

My Dashboard Page:



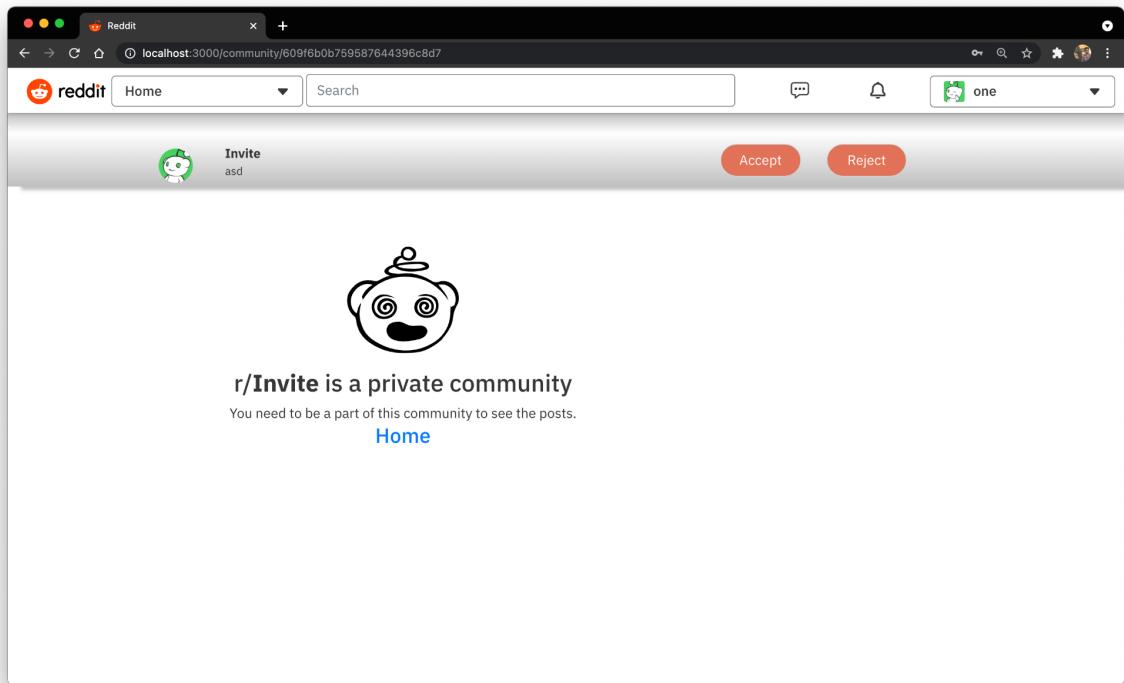
My Communities Analytics Page:



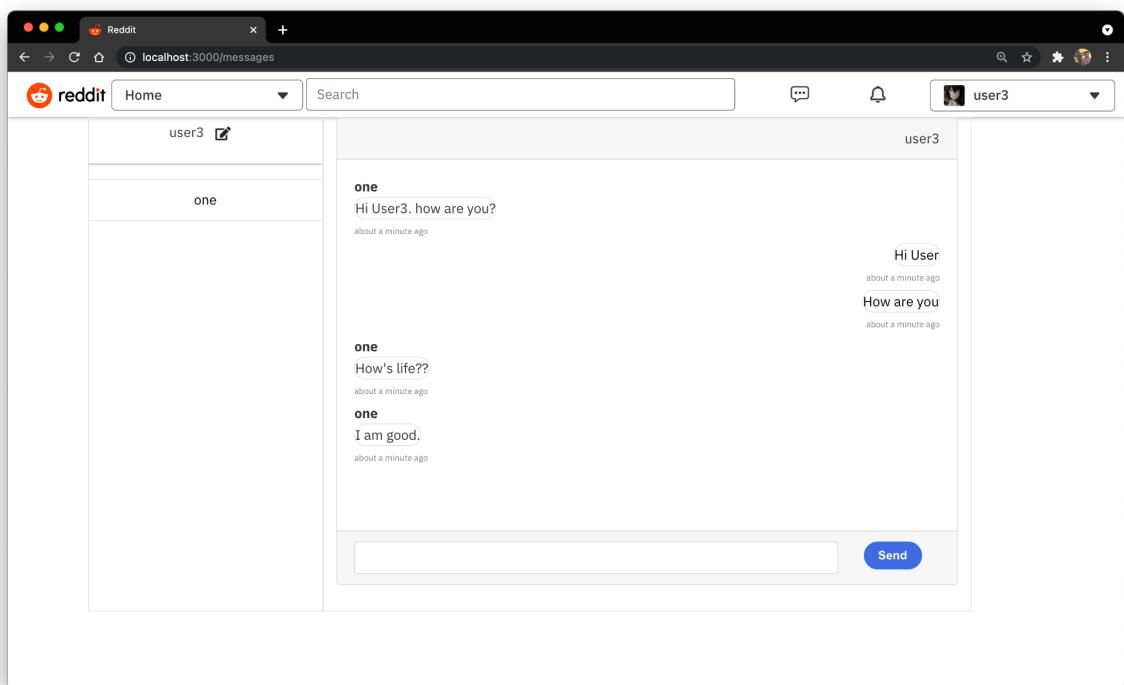
Invitation:

A screenshot of a web browser displaying a Reddit clone interface. A modal window titled "Notification" is open, listing two items: "newcomm1" and "Invite". Both items have green checkmarks and red X's next to them, indicating they can be accepted or declined. "newcomm1" was posted "2 minutes ago". "Invite" was posted "a few seconds ago". Below the modal, the main content area shows a post from "r/one_community" with the link "[LINK POST IS HERE]" and the URL "http://localhost:3000". Another post from "r/one_community" is visible below it, with the link "[IMAGE POST]".

A screenshot of a web browser displaying the "Invitation" management page. The search bar contains the text "newcomm1". The main content area is titled "Approved Users" and lists four users: "one", "two", "User2", and "user3", each with a small profile icon. To the right of each user is an upvote arrow icon and the text "a few seconds ago". At the bottom of the list, there are pagination controls: "Rows per page: 10", "1-4 of 4", and navigation arrows. The top of the page shows the Reddit logo and the user "user7".



My Messages:



User Profile Page:

The screenshot shows a user profile page for a user named "one" on a local instance of Reddit at `localhost:3000/user/74`. The interface is styled like the official Reddit website.

Left Sidebar (Subreddits):

- newcomm**: 0 posts, 1 member, created by **one** 7 minutes ago.
- one_community**: 6 members, 4 posts, created by **one** 30 minutes ago.

User Profile (Right Side):

Name	Location
one	N/A
Bio	N/A

Code listing of entity objects:

Comment:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const CommentSchema = new mongoose.Schema(
{
    communityID: { type: Schema.Types.ObjectId, ref: "Community" },
    postID: { type: Schema.Types.ObjectId, ref: "Post" },
    description: { type: String, required: true },
    parentCommentID: { type: Schema.Types.ObjectId, ref: "Comment" },
    userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
    upvotedBy: [
        {
            userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
        },
    ],
    downvotedBy: [
        {
            userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
        },
    ],
    score: { type: Number, default: 0 },
    parentCommentID: { type: Schema.Types.ObjectId, ref: "Comment" },
    isParentComment: { type: Boolean, defaultValue: true },
},
{
    timestamps: true,
}
);

module.exports = mongoose.model("Comment", CommentSchema);
```

Community:

```
const mongoose = require("mongoose");
const mongooseAggregatePaginate = require("mongoose-aggregate-paginate-v2");

const Schema = mongoose.Schema;
const CommunitySchema = new mongoose.Schema(
{
    communityName: { type: String, required: true, unique: true },
    communityDescription: { type: String },
    imageURL: [
        {
            url: { type: String },
        },
    ],
    topicSelected: [
        {
            topic: { type: String },
        },
    ],
    listOfUsers: [
        {
            userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
            isAccepted: { type: Number, default: 0 },
            isModerator: { type: Boolean, defaultValue: false },
        },
    ],
    ownerID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
    NoOfPost: { type: Number },
    upvotedBy: [
        {
            userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
        },
    ],
    downvotedBy: [
        {
            userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
        },
    ],
    score: { type: Number, default: 0 },
    sentInvitesTo: [
        {
            userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
        },
    ],
}
```

```

        isAccepted: { type: Number, default: 0 },
        dateTme: { type: Date, default: Date.now },
    },
],
rules: [
{
    title: { type: String },
    description: { type: String },
},
],
},
{
    timestamps: true,
}
);

CommunitySchema.plugin(mongooseAggregatePaginate);

module.exports = mongoose.model("Community", CommunitySchema);

```

Message:

```

module.exports = (sequelize, DataTypes) => {
  const Message = sequelize.define("Message", {
    message_id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      autoIncrement: true,
      primaryKey: true
    },
    message: {
      type: DataTypes.STRING(255),
      allowNull: false
    }
  });

  Message.associate = models => {
    Message.belongsTo(models.User, {
      as: "sentByUser",
      foreignKey: {
        allowNull: false,
        name: "sent_by"
      }
    });
  };
}

```

```

        }
    }) ,
    Message.belongsTo(models.User, {
        as: "sentToUser",
        foreignKey: {
            allowNull: false,
            name: "sent_to"
        }
    });
}

return Message;
};


```

Post:

```

const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const PostSchema = new mongoose.Schema(
{
    communityID: { type: Schema.Types.ObjectId, ref: "Community" },
    userID: { type: Schema.Types.ObjectId, ref: "UserProfile" },
    type: {
        type: Number,
        required: true,
    },
    link: { type: String }, // if post type is link, save link
    otherwise save url of image if post type is image
    postImageUrl: { type: String },
    description: { type: String },
    NoOfComments: { type: Number },
    score: { type: Number, default: 0 },
    title: { type: String, required: true },
},
{
    timestamps: true,
}
);

module.exports = mongoose.model("Post", PostSchema);

```

UserProfile:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const UserProfileSchema = new mongoose.Schema({
  userIDSQL: {
    type: String,
    required: true,
    unique: true
  },
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  gender: {
    type: String
  },
  location: {
    type: String
  },
  bio: {
    type: String
  },
  profile_picture_url: {
    type: String
  },
  listOfTopics: [
    {
      topic: { type: String }
    }
  ],
  communityInvites: [
    {
      communityID: {
        type: Schema.Types.ObjectId,
        ref: "Community"
      },
    }
  ]
});
```

```

        isAccepted: {
            type: Boolean,
            defaultValue: false
        },
        invitedBy: {
            type: Schema.Types.ObjectId,
            ref: "UserProfile"
        },
        dateTime: { type: Date, default: Date.now }
    }
]
} );
}

module.exports = mongoose.model("UserProfile", UserProfileSchema);

```

Vote:

```

const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const VoteSchema = new mongoose.Schema(
{
    entityId: { type: Schema.Types.ObjectId, ref: "Community" },
    userId: { type: Schema.Types.ObjectId, ref: "User" },
    voteDir: { type: Number },
},
{
    timestamps: true,
}
);

module.exports = mongoose.model("Vote", VoteSchema);

```

Topic:

```

module.exports = (sequelize, DataTypes) => {
    const Topic = sequelize.define('Topic', {
        topic_id: {
            type: DataTypes.INTEGER,
            allowNull: false,
            autoIncrement: true,
            primaryKey: true,
        },
        topic: {

```

```

        type: DataTypes.STRING(150),
        allowNull: false,
        unique: true
    }
})
return Topic;
}

```

User:

```

module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define("User", {
    user_id: {
      type: DataTypes.INTEGER,
      allowNull: false,
      autoIncrement: true,
      primaryKey: true
    },
    email: {
      type: DataTypes.STRING(150),
      allowNull: false,
      unique: true
    },
    password: {
      type: DataTypes.STRING(200),
      allowNull: false
    },
    name: {
      type: DataTypes.STRING(150),
      allowNull: false
    },
    profile_picture_url: {
      type: DataTypes.STRING(255)
    }
  });

  User.associate = models => {
    UserhasMany(models.Message, {
      onDelete: "cascade",
      foreignKey: {
        allowNull: false,
        name: "sent_by"
      }
    })
  }
}

```

```

        } ) ,
        User.hasMany(models.Message, {
            onDelete: "cascade",
            foreignKey: {
                allowNull: false,
                name: "sent_to"
            }
        ) ;
    } ;

    return User;
} ;

```

Code listing of the security and session objects:

Authentication using JWT-Token:

```

const db = require("./../models/sql");

const jwtAuthHandler = async (msg, callback) => {
    res = {};
    try {
        console.log(msg, "msg");
        db.User.findOne({ where: { user_id: msg.user_id } }).then(result =>
{
            res.status = 200;
            res.data = result;
            callback(null, res);
        }).catch(err => {
            res.status = 404;
            res.data = err;
            callback(null, res);
        });
    } catch (err) {
        res.status = 500;
        callback(null, res);
    }
};

handle_request = (msg, callback) => {
    if (msg.path === "jwt-auth") {
        jwtAuthHandler(msg, callback);
    }
};

```

```

        }
    };

exports.handle_request = handle_request;

```

Hashing user Password:

```

const db = require("../models/sql");
const passwordHash = require('password-hash');

const login = async (msg, callback) => {
    res = {};
    db.User.findOne({
        where: {
            email: msg.email
        }
    })
    .then(user => {
        if (user === null) {
            res.status = 404;
            callback(null, res);
        } else {
            if (passwordHash.verify(msg.password, user.password)) {
                user.password = "";
                res.status = 200;
                res.data = user;
                callback(null, res);
            } else {
                res.status = 401;
                callback(null, res);
            };
        }
    })
    .catch(err => {
        res.status = 500;
        callback(null, res);
    });
};

exports.login = login;
const db = require("../models/sql");
const passwordHash = require('password-hash');

```

```

const signUp = async (msg, callback) => {
  msg.password = passwordHash.generate(msg.password)
  db.User.create({
    email: msg.email,
    password: msg.password,
    name: msg.name
  })
  .then(user => {
    callback(null, user);
  })
  .catch(err => {
    callback(err, null);
  });
};

exports.signUp = signUp;

```

Main Server Code:

```

const path = require("path");
const cookieParser = require("cookie-parser");
const logger = require("morgan");
const express = require("express");
const cors = require("cors");
const app = express();
const { frontEnd } = require("./Util/config");
// DB connection
// require("./dbConnection");

// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");
app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));

const session = require("express-session");

// use session to store user data between HTTP requests
app.use(

```

```

session({
  secret: "cmpe273_kafka_passport_mongo",
  resave: false,
  saveUninitialized: true,
  duration: 60 * 60 * 1000, // Overall duration of Session : 30
  minutes : 1800 seconds
  activeDuration: 5 * 60 * 1000,
})
);

app.set("view engine", "ejs");

// use cors to allow cross origin resource sharing
app.use(
  cors({
    origin: frontEnd,
    credentials: true,
  })
);

// Allow Access Control
app.use(function (req, res, next) {
  res.setHeader("Access-Control-Allow-Origin", frontEnd);
  res.setHeader("Access-Control-Allow-Credentials", "true");
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET,HEAD,OPTIONS,POST,PUT,DELETE"
  );
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Access-Control-Allow-Headers, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, Access-Control-Request-Headers"
  );
  res.setHeader("Cache-Control", "no-cache");
  next();
});

// error handler
app.use(function (err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get("env") === "development" ? err : {};
});

```

```

    // render the error page
    res.status(err.status || 500);
    res.render("error");
} );

module.exports = app;

```

Kafka backend server:

```

var connection = new require("./kafka/connection");
//const messageSQL = require("./services/Message/message_sql");
const message = require("./services/Message/topicMapping");
const user_info = require("./services/userSQL/topicMapping");
const jwt_auth = require("./services/jwt_auth");

const search_mongo = require("./services/search_mongo");

const user_mongo = require("./services/userMongo/topicMapping");
const userAuth_sql = require("./services/userAuth/topicMapping");
const vote_mongo = require("./services/Vote/topicMapping");
const community_mongo = require("./services/community/topicMapping");
const manage_community =
require("./services/community/manageCommunity/topicMapping");
const community_analytics =
require("./services/community/communityAnalytics/topicMapping");
const getTopic = require("./services/getTopic");
const post = require("./services/post/topicMapping");
const comment = require("./services/comment/topicMapping");
const community_user =
require("./services/community/communityUser/topicMapping");
const topic_sql = require("./services/Topic/topicMapping");

require("./dbConnection");
const sqldb = require("./models/sql");
sqldb.sequelize.sync().then(() => {
  console.log("sequelize is running");
});

function handleTopicRequest(topic_name, fname) {
  var consumer = connection.getConsumer(topic_name);
  var producer = connection.getProducer();

```

```

console.log("Kafka Server is running ");
consumer.on("message", function (message) {
  console.log("Message received for " + topic_name);
  var data = JSON.parse(message.value);

  fname.handle_request(data.data, function (err, res) {
    console.log("HANDLED", err, res);
    var payloads = [
      {
        topic: data.replyTo,
        messages: JSON.stringify({
          correlationId: data.correlationId,
          data: res,
        }) ,
        partition: 0,
      },
    ];
    producer.send(payloads, function (err, data) {
      console.log("payload sent:", data);
    });
    return;
  });
});

handleTopicRequest("sql_message", message);
handleTopicRequest("JWT_auth", jwt_auth);
handleTopicRequest("mongo_user", user_mongo);
handleTopicRequest("sql_user_auth", userAuth_sql);
handleTopicRequest("search_mongo", search_mongo);
handleTopicRequest("mongo_community", community_mongo);
handleTopicRequest("user_info", user_info);
handleTopicRequest("manage_community", manage_community);
handleTopicRequest("community_analytics", community_analytics);
handleTopicRequest("get_topic", getTopic);
handleTopicRequest("post", post);
handleTopicRequest("comment", comment);
handleTopicRequest("community_user", community_user);
handleTopicRequest("vote_mongo", vote_mongo);
handleTopicRequest("manage_topic", topic_sql);

```

Database Connection code:

```
const {
  SQLdb,
  SQLusername,
  SQLpassword,
  SQLhost,
  mongoURI,
} = require("./Util/config");
const mongoose = require("mongoose");

mongoose.connect(mongoURI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useCreateIndex: true,
});
const connection = mongoose.connection;
console.log("MongoDB Connected !!!");

// SQL Connection
const Sequelize = require("sequelize");
const sqldb = new Sequelize(SQLdb, SQLusername, SQLpassword, {
  host: SQLhost,
  dialect: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 1000000,
    idle: 10000,
  },
});
sqldb
  .authenticate()
  .then(() => {
    console.log("SQL Connection has been established successfully.");
  })
  .catch((err) => {
    console.error("Unable to connect to the SQL database:", err);
  });

module.exports = sqldb;
```

testpages.js — Untitled (Workspace)

```
273-Reddit > backend > test > JS testpages.js > ...
```

```
You, 2 hours ago | author (You)
1 const app = require("../index.js");
2 var chai = require("chai");
3 chai.use(require("chai-http"));
4 var expect = require("chai").expect;
5
6 var agent = require("chai").request.agent(app);
7 const auth =
8 "JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpVJCJ9eyJpZCI6NTESImIhdCI6MTYyMTA0OTM3OCwiZhwIjoxNjIyMDU3Mzc4fQ.GBTpxFxBdRXHEvil84sJgzNRTM9306ozj80MLR";
10 > it("POST /loginUser", function (done) {
11   agent
12     .get("/getUserProfileByMongoID?ID=609f4022e8ab974852c6020a")
13     .set("Authorization", auth)
14     .then(function (res) {
15       expect(res).to.have.status(200);
16       done();
17     })
18     .catch(e => {
19       done(e);
20     });
21   });
22
23 describe("Reddit App", function () {
24   it("GET ", function (done) {
25     agent
26       .get("/getCommunitiesForOwner - get Community", function (done) {
27         agent
28           .get(
29             "/getCommunitiesForOwner?ID=609f4022e8ab974852c6020a&page=0&size=2&search=''"
30           )
31           .set("Authorization", auth)
32           .then(function (res) {
33             expect(res).to.have.status(200);
34             done();
35           })
36           .catch(e => {
37             done(e);
38           });
39         });
40       });
41     });
42     });
43     });
44     });
45     });
46     });
47     });
48     });
49     });
50     });

Ln 188, Col 1 Spaces: 2 UTF-8 LF JavaScript Prettier ✓ Prettier
```

EXPLORER ... JS testpages.js X JS user.js .../sql JS post.js JS comment.js JS user.js .../mongo JS router.js JS vote.

Vinay SJSU 04:22

Video Mute End

273-Reddit

- backend
- bin
- kafka
- kafkaRoutes
- mongo
- comment.js
- community.js
- post.js
- router.js
- search.js
- user.js
- vote.js
- sql
- message.js
- router.js
- user.js
- userAuth.js
- models
- node_modules
- public
- routes
- test
- JS testpages.js
- Util
- app.js
- dbConnection.js
- index.js
- Kafka scripts
- [1] package-lock.json

> OUTLINE

> TIMELINE

> NPM SCRIPTS

dev ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ tabnine ↗

community.js — Untitled (Workspace)

PROBLEMS OUTPUT TERMINAL

```
> < TERMINAL
GET /getCommunities 200 146.533 ms - 24
  ✓ Get /getCommunities - Get Communities (151ms)
    Get getNotification data
    User ID From Token :51
    { JWT_auth: { '0': 3056 } }
    { mongo_user: { '0': 1084 } }
    [
      {
        communityName: 'mocha',
        communityID: '609f42be94011d39c85c765f',
        time: '2021-05-15T03:40:57.837Z'
      }
    ]
    [
      {
        communityName: 'mocha',
        communityID: '609f42be94011d39c85c765f',
        time: '2021-05-15T03:40:57.837Z'
      }
    ]
POST /getNotificationData 200 233.924 ms - 102
  ✓ Get /getNotificationData (239ms)
    Get getCommunitiesCreatedByMe data
    User ID From Token :51
    { JWT_auth: { '0': 3057 } }
    { mongo_community: { '0': 1222 } }
POST /getCommunitiesCreatedByMe 200 176.173 ms - 69
  ✓ Get /getCommunitiesCreatedByMe (181ms)
    Post Comment
    User ID From Token :51
    { JWT_auth: { '0': 3058 } }
    { comment: { '0': 136 } }
POST /comment 200 246.510 ms - 0
  ✓ post /comment (255ms)
    Post Vote
    User ID From Token :51
    { JWT_auth: { '0': 3059 } }
    { comment: { '0': 137 } }
POST /comment 200 215.031 ms - 0
  ✓ post /vote (219ms)
    Post createPost
    User ID From Token :51
    { JWT_auth: { '0': 3060 } }
    { comment: { '0': 138 } }
POST /comment 200 221.184 ms - 0
  ✓ post /createPost (225ms)

10 passing (2s)
```

⌚ You, seconds ago Ln 131, Col 27 Spaces: 2 UTF-8 LF JavaScript Prettier ✓ Prettier

Load balancer:

Screenshot of the AWS EC2 Load Balancer configuration page:

The screenshot shows the AWS Management Console with the URL <https://us-west-2.console.aws.amazon.com/ec2/v2/home?region=us-west-2#LoadBalancers:sort=loadBalancerName>. The left sidebar is the New EC2 Experience interface.

Load balancer: reddit-loadbalancer

Basic Configuration

Name	Value
Name	reddit-loadbalancer
ARN	arn:aws:elasticloadbalancing:us-west-2:223198954103:loadbalancer/app/reddit-loadbalancer/697a7fc61728c174
DNS name	reddit-loadbalancer-1768064832.us-west-2.elb.amazonaws.com (A Record)
State	active
Type	application

Screenshot of the AWS EC2 Instances configuration page:

The screenshot shows the AWS Management Console with the URL <https://us-west-2.console.aws.amazon.com/ec2/v2/home?region=us-west-2#Instances:instanceState=running>. The left sidebar is the New EC2 Experience interface.

Welcome to the new instances experience!

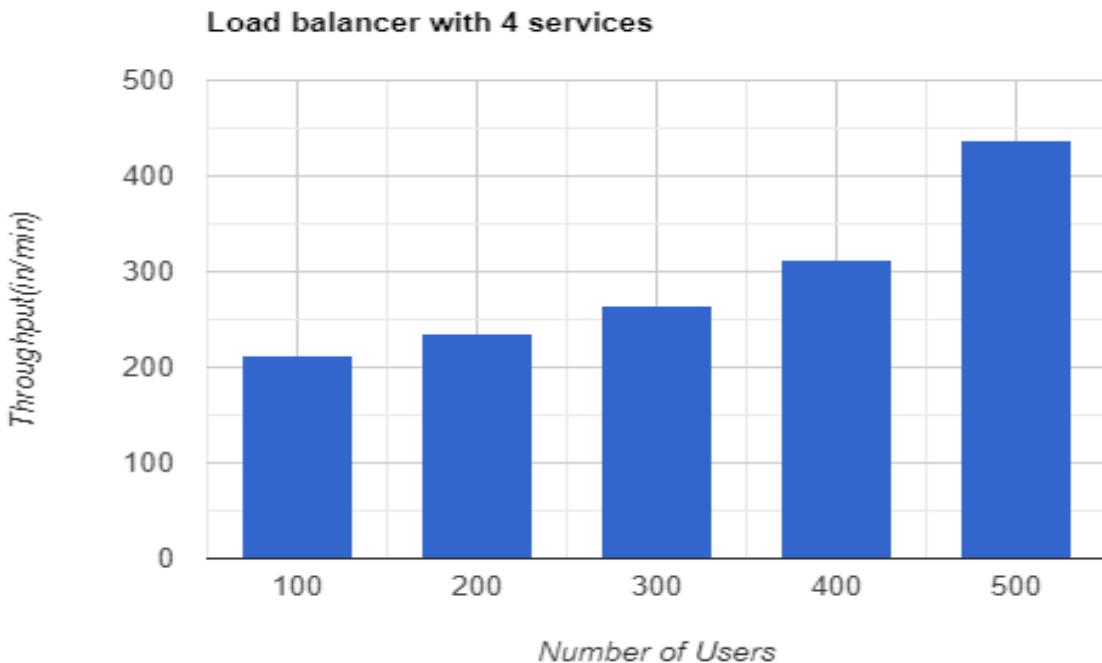
Instances (1/5) Info

Instance state: running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Publ
-	i-0d8683b8845d36a88	Running	t2.large	2/2 checks passed	No alarms	us-west-2b	ec2-
-	i-0ca483b108bf4a424	Running	t2.large	2/2 checks passed	No alarms	us-west-2b	ec2-
-	i-05550cf64a6c1db804	Running	t2.large	2/2 checks passed	No alarms	us-west-2b	ec2-
-	i-07f231cba1c971eda	Running	t2.large	2/2 checks passed	No alarms	us-west-2b	ec2-
<input checked="" type="checkbox"/>	i-002211e53eb82323f	Running	t2.large	2/2 checks passed	No alarms	us-west-2c	ec2-

Instance: i-002211e53eb82323f

Details



Observations for different deployment configurations

- **Caching** It delivers the best performance since it has a $O(1)$ look up time. APIs that involve looking up heavy queries or heavy computing can be cached to enhance performance. However, caching is expensive since it requires a large amount of RAM. Cache busting using time to live simplifies the problem but suffers from data consistency.
- **Load balancing** significantly reduces response time by distributing load across autoscaling groups in AWS across multiple EC2 machines. Auto scaling helps to handle the load in ad hoc manner. Auto scaling services need to be built in a stateless fashion since they can be brought down or spawned at anytime
- **Message queues** (Kafka) help decouple services and help make them truly distributed. With message queues, services can communicate in an asynchronous fashion, services can be replaced, and communication is not impacted even if one of the services is temporarily down. Messages in message queues like Kafka can be potentially stored for an infinite time which help it double up as a durable database store. Moreover, running Kafka on clusters enhances durability and scalability.
- **Connection pooling** enhances performance since a pool of TCP connections is kept alive which can be reused for subsequent requests thus avoiding overhead of reserving resources and time to establish a new TCP connection. By default, mongoose gives a size of 5 and when we increase it up to in the range of 15-20, we can see a lot of improvement in the way the results are retrieved. However, a very large connection pool is counterproductive since it leads to unnecessary reservation of resources thus choking memory.