San Jose State University

CMPE 273 - Enterprise Distributed systems

Simulation of Stackoverflow.com

Instructor: Dr Simon Shim

Submitted By

Group 5

Aniruddha A Narkhede	015451267
Sakshi Kasat	015880254
Siddhant Parmar	015742389
Soham Kasar	015899442
Sudheendra Katikar	015948465
Unmesh Padhye	015928471
Vineet Karmiani	015363530

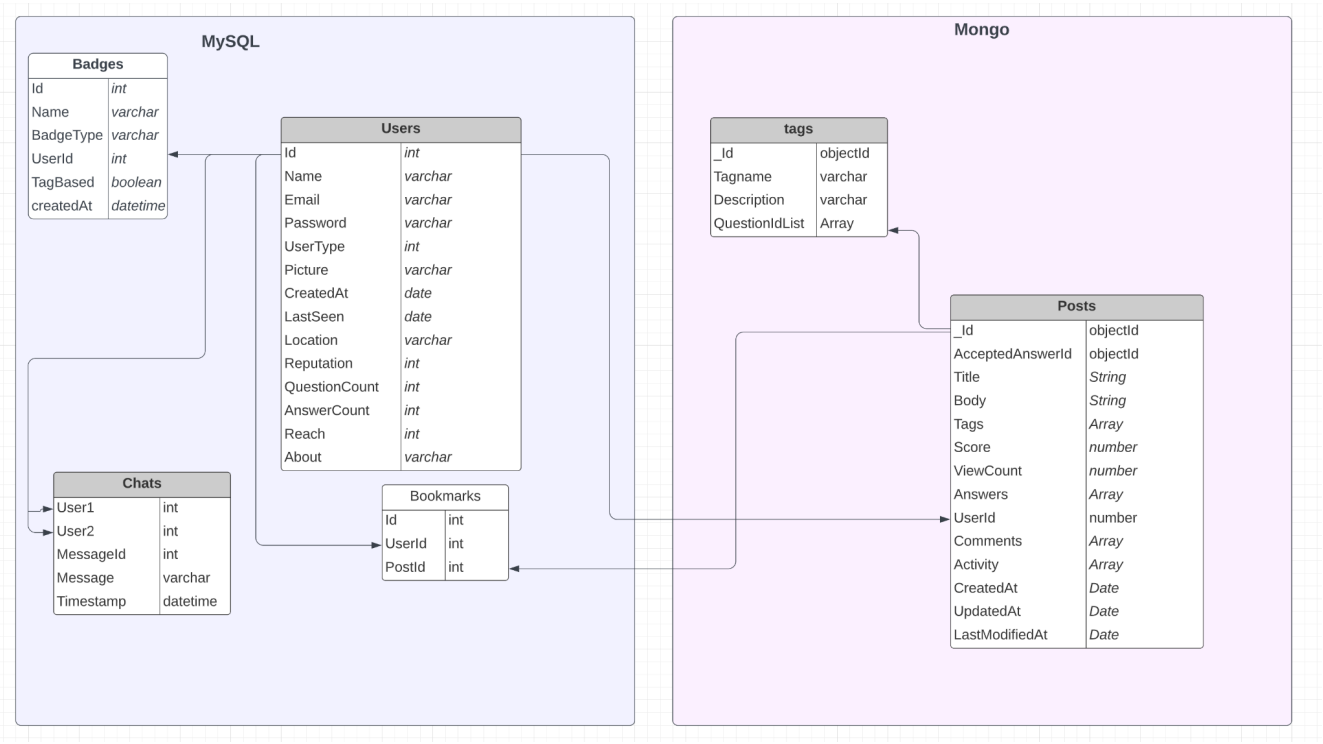
Individual Contribution:

- Aniruddha
    - Contributed to database design and system architecture
    - Implemented all the APIs persisting admin related activities like, get unapproved questions, approve/disapprove question, and data required for analytical dashboard
    - Setup JMeter for performance and stress testing and collated necessary graphs and comparisons
    - Contributed toward the deployment of the project on AWS using EC2 utilizing services like Load balancer and Auto Scaling
- Sakshi
    - Contributed to DB design.
    - Model creation of Posts and tags.
    - Implemented Create, Read and update APIs related to Posts, answers, comments and score.
    - Worked on maintaining user reputations, and score calculations for questions, answers, Upvotes and approvals.
    - Implemented Create and Read APIs related to tags.
    - Worked on dashboard APIs to get questions based on all types of filters.
    - API testing using Mocha and Chai.
- Siddhant
    - Implemented component of User Profile basic details, Profile tab and Activity tab.
    - Implemented UI for badges, questions by user, answers by user, tags and reputation.
    - Enhanced UI for Users tab and Tags tab.
    - Profile Update page and profile image upload to cloud.
- Soham
    - Contributed to DB design
    - Model creation of Messages
    - Implemented APIs required for messaging
    - Implemented APIs related to Users
    - Implemented Components for Navbar, Messaging, SearchBar, SearchPage, Admin, AskQuestion component
    - Contributed to Integrated APIs for the above components

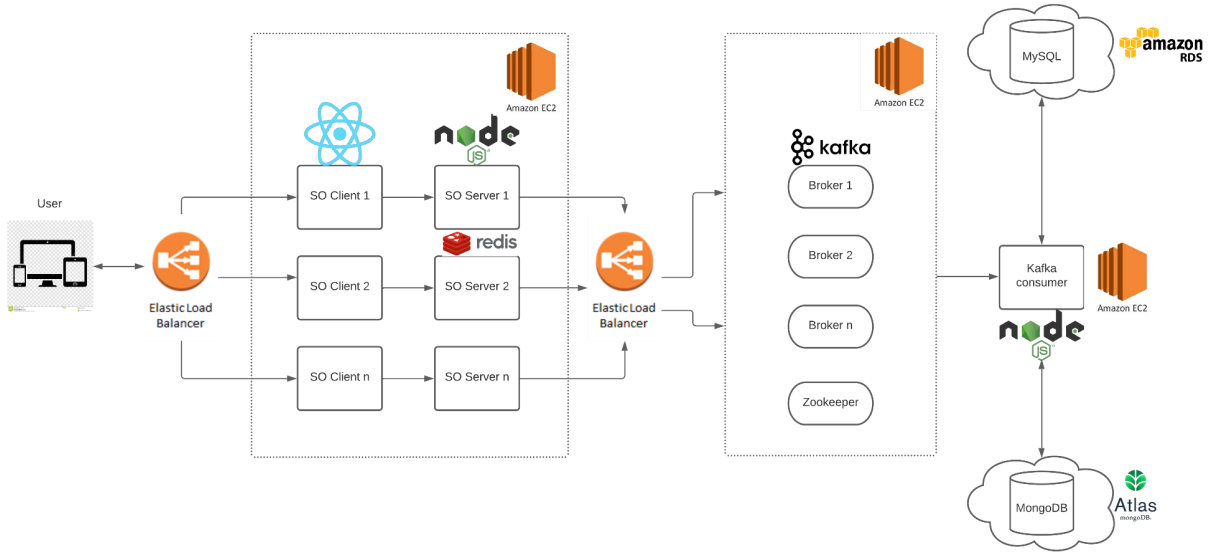
    - Contributed in showing badges on navbar.
- Sudheendra
    - Contributed to database design
    - Set up Kafka producers/consumers and logging
    - Created models for user, bookmarks
    - Implemented user authentication and profile APIs and services
    - DB deployment on the cloud, setup Redis cloud cache
    - Worked on deployment of app on EC2
- Unmesh
    - Contributed to database design
    - Additions to Badge and Post models
    - Designed and created APIs related to Badges and Searching
    - Creating routes for searching, badges and users
    - Migrated some of the backend APIs through kafka for ‘posts’ topic
    - Implemented Redis cache to store posts data

- Vineet
    - Contributed to DB design
    - Implemented Components for Home, Login, Signup, Tags, Users, AskQuestion, Questions for all selected tag and Question Overview page(showing the questions, answers, comments, upvoted, bookmarks, and approving the question), Admin component.
    - Integrated all the api’s for the above component.
    - Contributed in showing badges on navbar.

Database Schema



System Architecture Design



Notes:

Object Management Policy:

Post objects which are dynamic are stored in mongodb to achieve scalability. All the structured data like the user credentials, user information, badges, chats , bookmarks are all stored in MySQL.

MongoDB has higher throughput in terms of read operations which is why it made the obvious choice to store all the post related data. Also the data inside a post document has fields like answers, comments. These are arrays of complex objects which can be stored effectively in MongoDB. Thus MongoDB is the best option to handle this type of ever increasing data.

User related information which is sensitive which is why it is stored in MySQL. Also the schema for badges, bookmarks, users is very structured and hence it is better handled using MySQL.

Handling of heavyweight resources

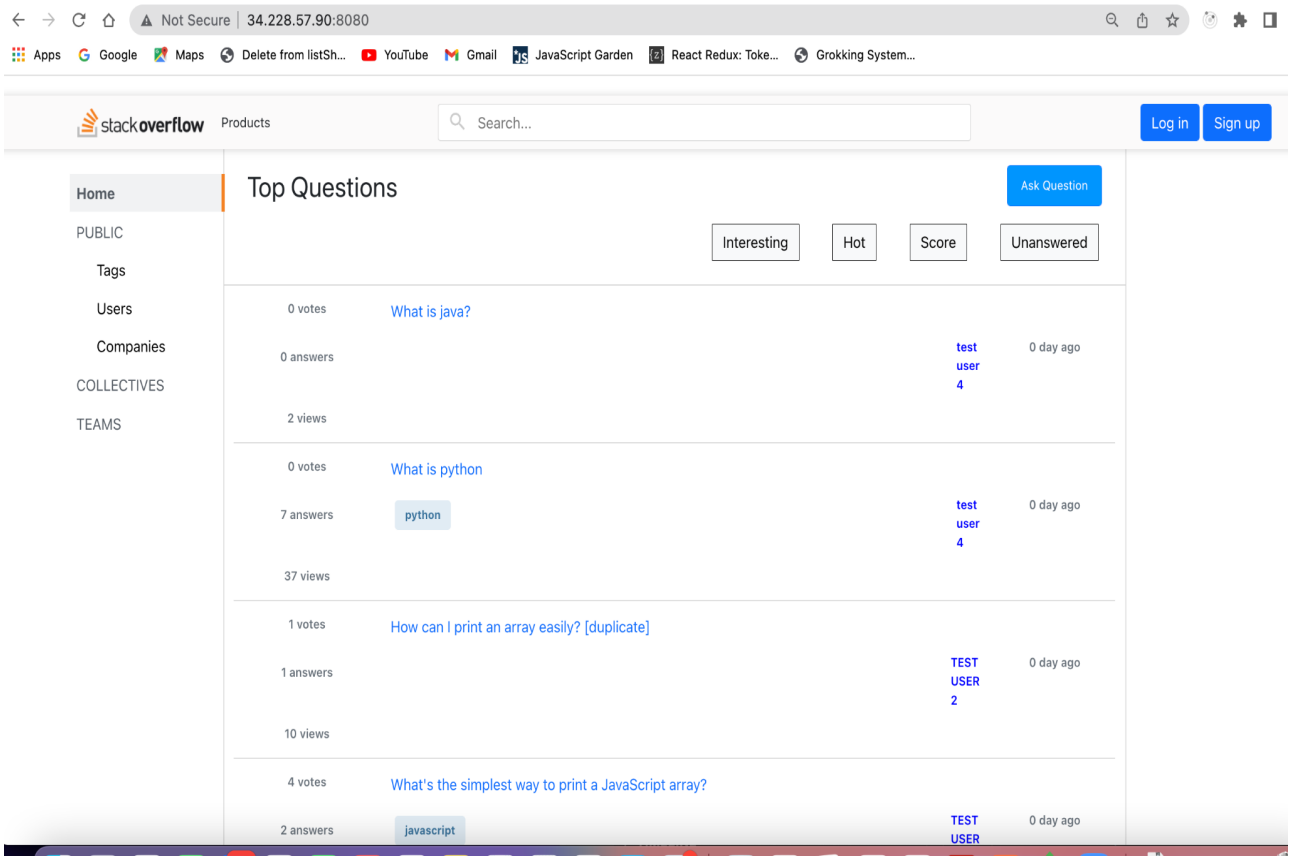
In order to handle latency caused by various heavyweight resources, we have adopted a few techniques to handle them. Firstly, we are making use of connection pooling which helps in increasing the system performance. We have also used Redis to store data which is being fetched constantly and that is user information. This data from redis is removed when user information is updated so as to avoid incorrect information from being displayed.

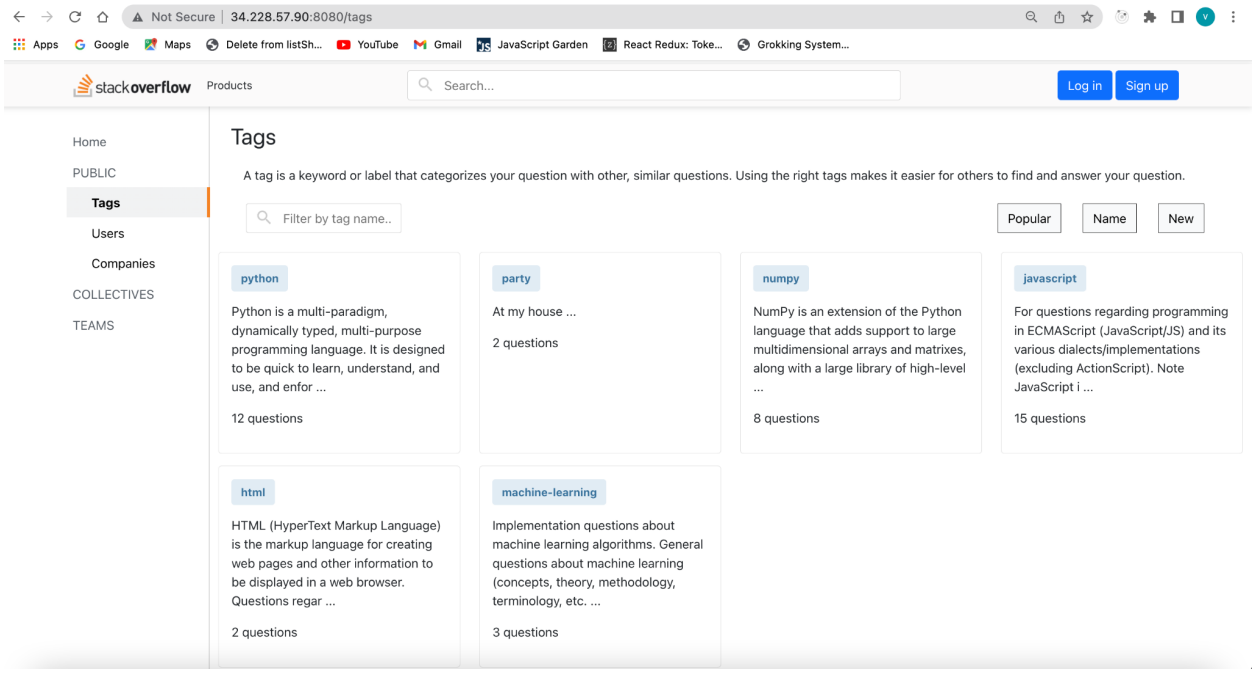
We have also stored data in Mongodb in a nested fashion so as to avoid aggregation and complex referencing. This has also helped in adding or updating data to a post ACID compliant.

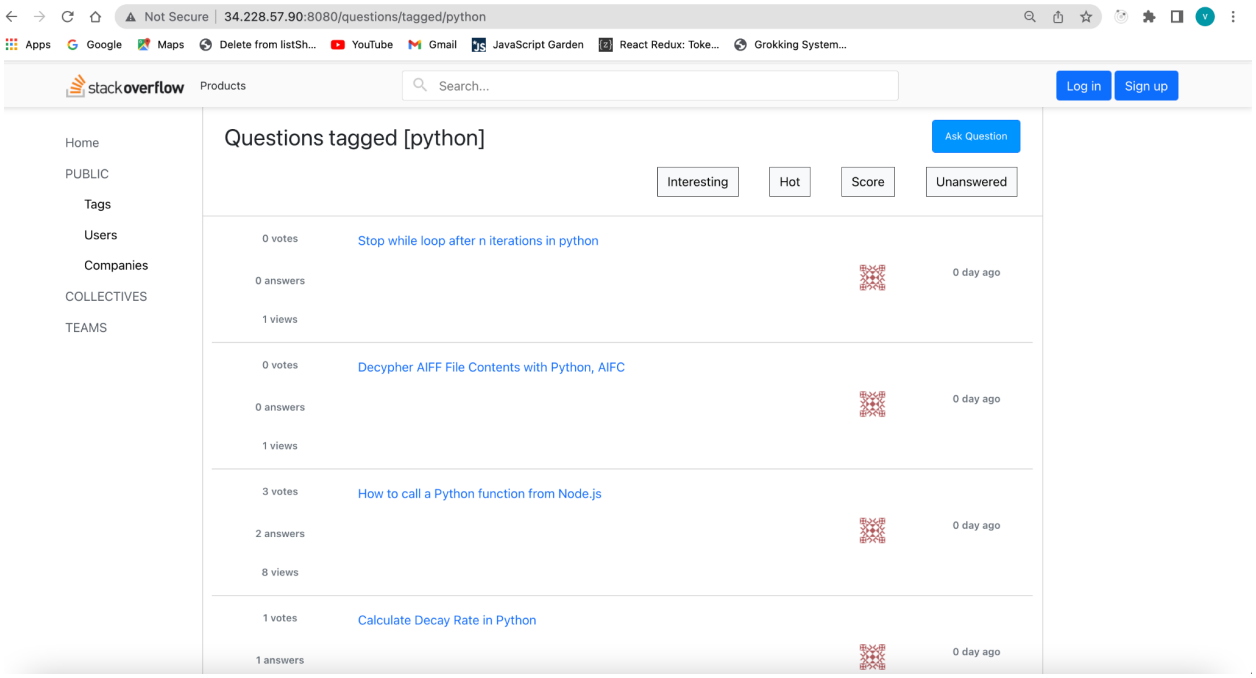
Policy to decide when to write data into database

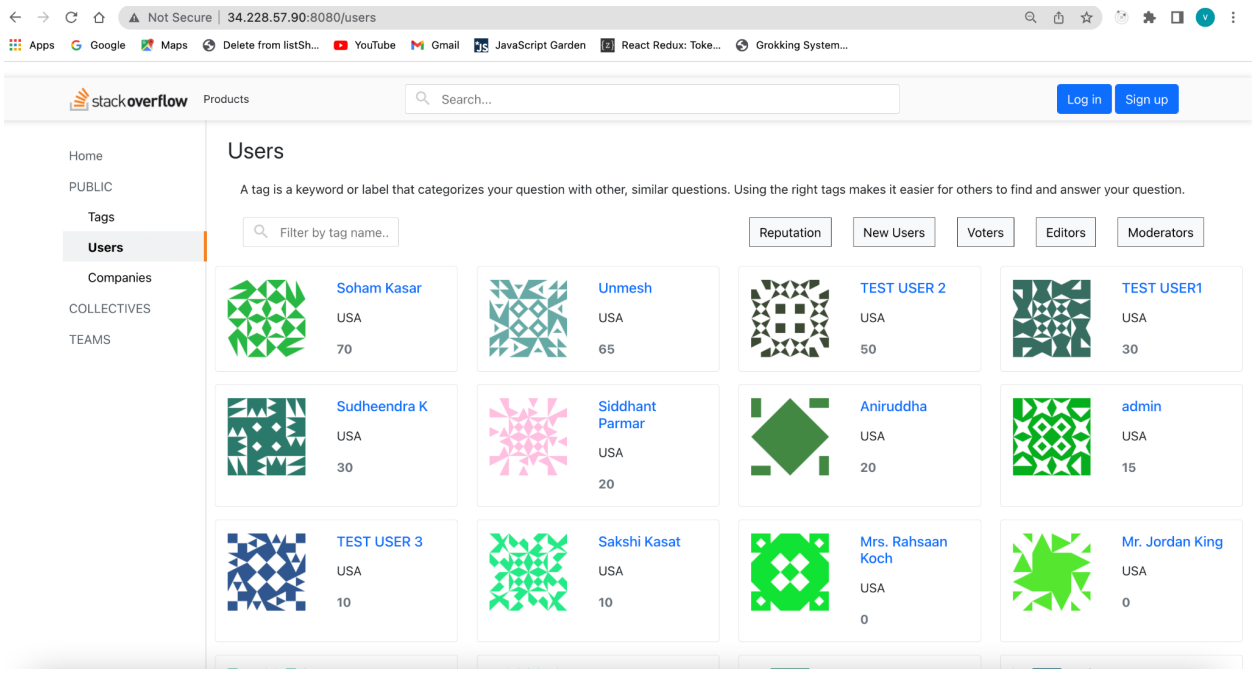
Only authenticated and authorized users were permitted to add questions, answers and comments. The bookmark and upvote/downvote feature also has restricted access. In our project, we implemented JWT authentication to authorize the API calls. We created private routes for admin and only admin can access those routes.

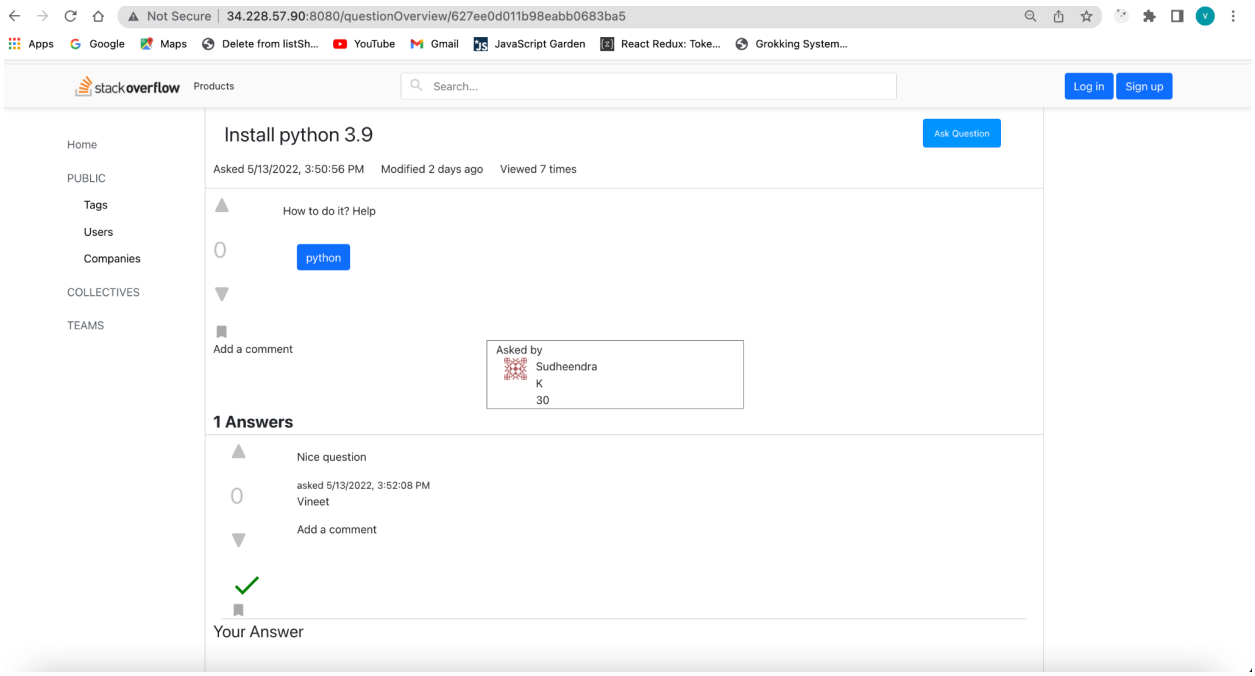
Screen Captures of important pages

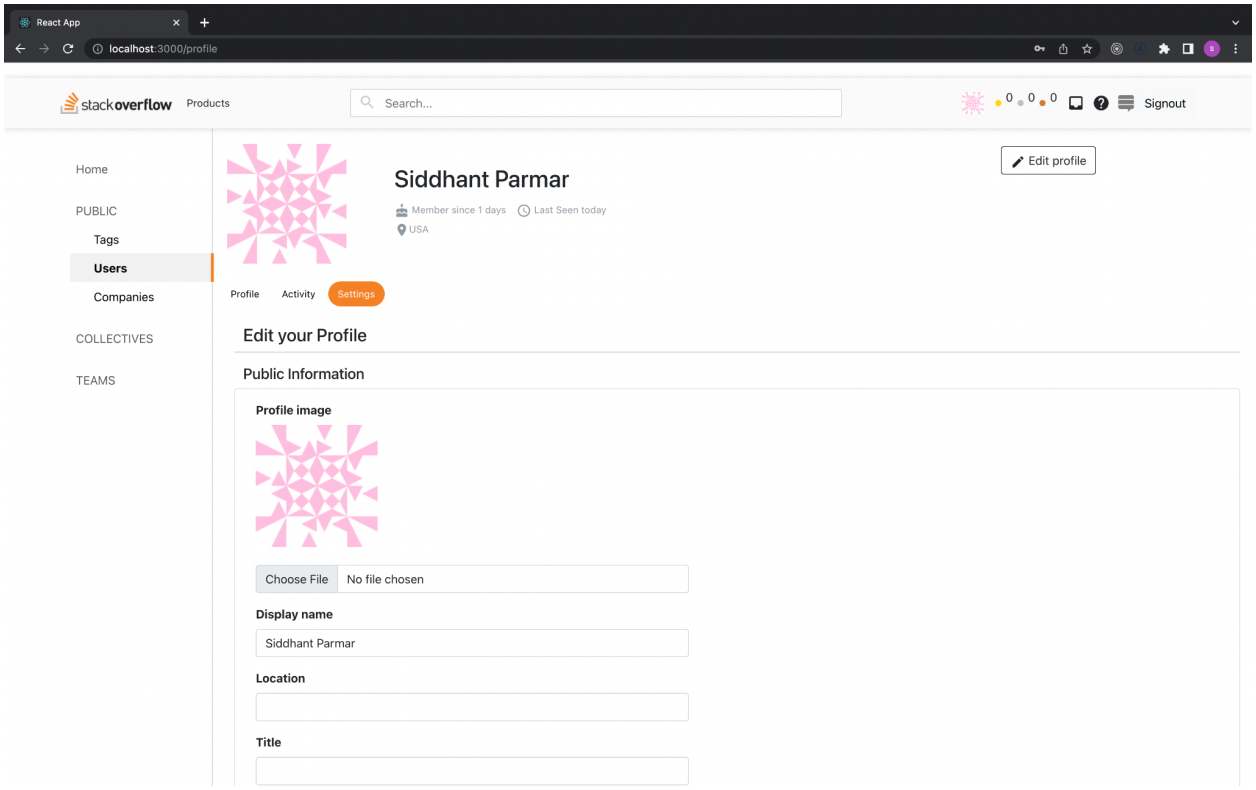


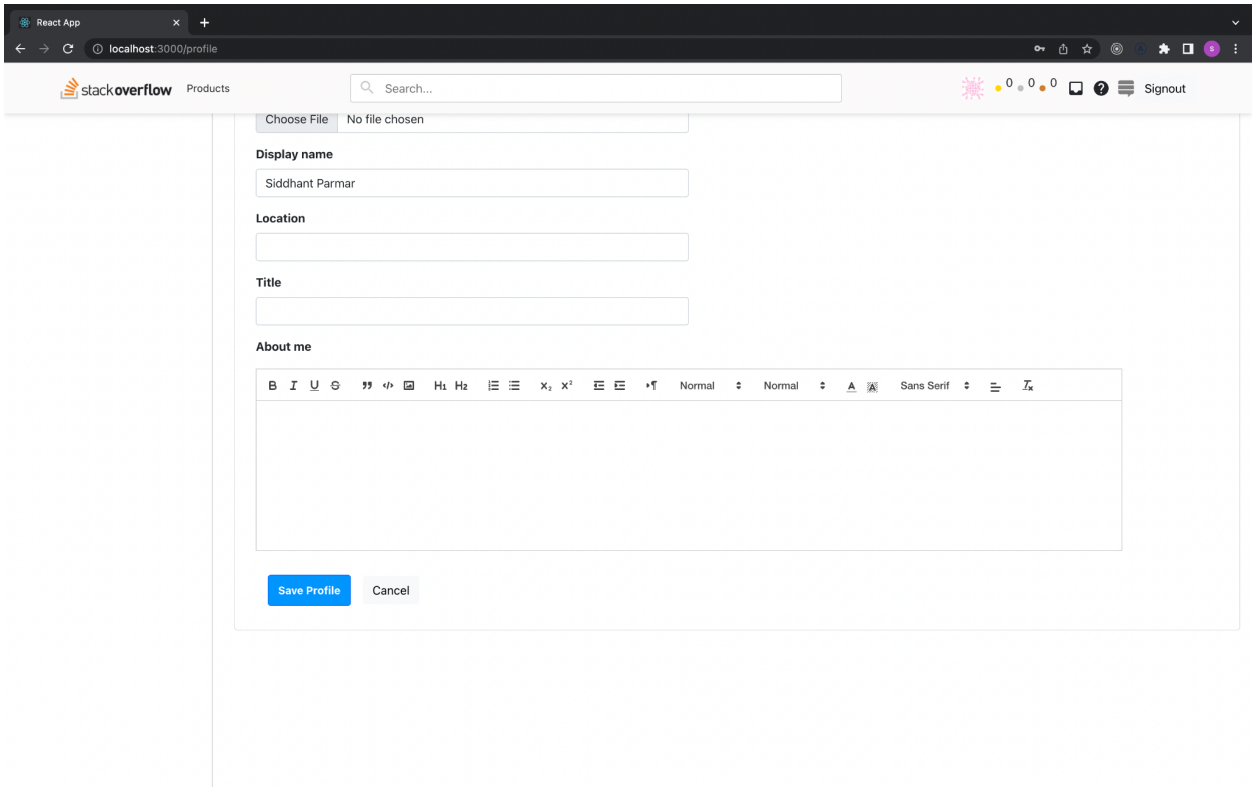


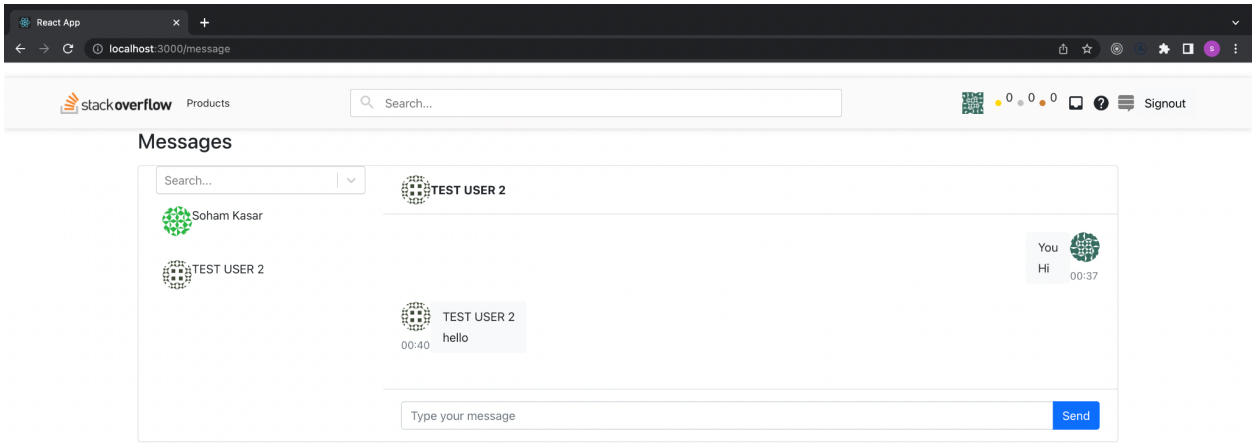

Not Secure | 34.228.57.90:8080/questions/tagged/python
Apps Google Maps Delete from listSh... YouTube Gmail JavaScript Garden React Redux: Toke... Grokking System...
stackoverflow Products
Search...
Log in
Sign up
Home
PUBLIC
Tags
Users
Companies
COLLECTIVES
TEAMS
Questions tagged [python]
Ask Question
Interesting
Hot
Score
Unanswered
0 votes
0 answers
1 views
Stop while loop after n iterations in python
0 day ago
0 votes
0 answers
1 views
Decypher AIFF File Contents with Python, AIFC
0 day ago
3 votes
2 answers
8 views
How to call a Python function from Node.js
0 day ago
1 votes
1 answers
Calculate Decay Rate in Python
0 day ago


Not Secure | 34.228.57.90:8080/users
Apps Google Maps Delete from listSh... YouTube Gmail JavaScript Garden React Redux: Toke... Grokking System...
stackoverflow Products
Search...
Log in
Sign up
Home
PUBLIC
Tags
Users
Companies
COLLECTIVES
TEAMS
Users
A tag is a keyword or label that categorizes your question with other, similar questions. Using the right tags makes it easier for others to find and answer your question.
Filter by tag name..
Reputation
New Users
Voters
Editors
Moderators
Soham Kasar
USA
70
Unmesh
USA
65
TEST USER 2
USA
50
TEST USER1
USA
30
Sudheendra K
USA
30
Siddhant Parmar
USA
20
Aniruddha
USA
20
admin
USA
15
TEST USER 3
USA
10
Sakshi Kasat
USA
10
Mrs. Rahsaan Koch
USA
0
Mr. Jordan King
USA
0

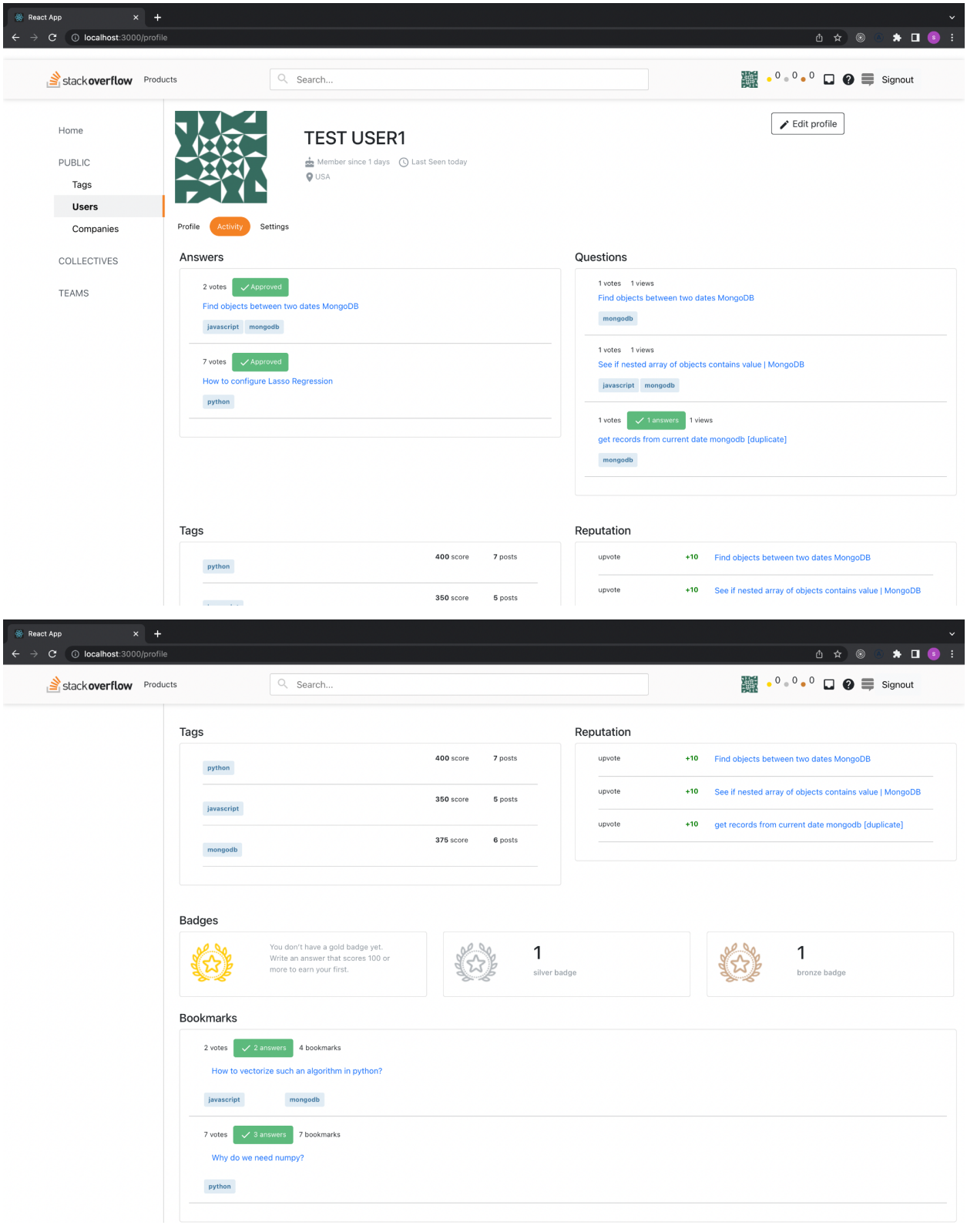

Not Secure | 34.228.57.90:8080/questionOverview/627ee0d011b98eabb0683ba5
Apps
Google
Maps
Delete from listSh...
YouTube
Gmail
JavaScript Garden
React Redux: Toke...
Grokking System...
stackoverflow
Products
Search...
Log in
Sign up
Home
PUBLIC
Tags
Users
Companies
COLLECTIVES
TEAMS
Install python 3.9
Ask Question
Asked 5/13/2022, 3:50:56 PM
Modified 2 days ago
Viewed 7 times
How to do it? Help
0
python
Add a comment
Asked by
Sudheendra
K
30
1 Answers
Nice question
0
asked 5/13/2022, 3:52:08 PM
Vineet
Add a comment
Your Answer


React App
localhost:3000/profile
stackoverflow
Products
Search...
Signout
Home
PUBLIC
Tags
Users
Companies
COLLECTIVES
TEAMS
Siddhant Parmar
Member since 1 days
Last Seen today
USA
Edit profile
Profile
Activity
Settings
Edit your Profile
Public Information
Profile image
Choose File
No file chosen
Display name
Siddhant Parmar
Location
Title


React App
localhost:3000/profile
stackoverflow
Products
Search...
Signout
Choose File
No file chosen
Display name
Siddhant Parmar
Location
Title
About me
Normal
Normal
Sans Serif
Save Profile
Cancel



Note: Reputation and Bookmarks implemented on Localhost (screenshots as requested by TA)


React App
localhost:3000/profile
stackoverflow
Products
Search...
Signout
Home
PUBLIC
Tags
Users
Companies
COLLECTIVES
TEAMS
TEST USER1
Member since 1 days
Last Seen today
USA
Edit profile
Profile
Activity
Settings
Answers
2 votes
Approved
Find objects between two dates MongoDB
javascript
mongodb
7 votes
Approved
How to configure Lasso Regression
python
Questions
1 votes
1 views
Find objects between two dates MongoDB
mongodb
1 votes
1 views
See if nested array of objects contains value | MongoDB
javascript
mongodb
1 votes
1 answers
1 views
get records from current date mongodb [duplicate]
mongodb
Tags
python
400 score
7 posts
350 score
5 posts
Reputation
upvote
+10
Find objects between two dates MongoDB
upvote
+10
See if nested array of objects contains value | MongoDB
Badges
You don't have a gold badge yet.
Write an answer that scores 100 or more to earn your first.
1
silver badge
1
bronze badge
Bookmarks
2 votes
2 answers
4 bookmarks
How to vectorize such an algorithm in python?
javascript
mongodb
7 votes
3 answers
7 bookmarks
Why do we need numpy?
python

Code listing implementation of entity objects

Posts:

```
const postSchema = new mongoose.Schema({
 title: {
   type: String,
   required: true,
 },
 body: {
   type: String,
   required: true,
 },
 tags: {
   type: Array,
   default: [],
 },
 ownerId: {
   type: Number,
   required: true,
 },
 answers: {
   type: Array,
   default: [],
 },
 score: {
   type: Number,
   default: 0,
 },
 viewCount: {
   type: Number,
   default: 0,
 },
 approved: {
   type: Boolean
 },
 answerApproved: {
   type: Boolean,
   default: false
 },
 comment: {
   type: Array,
   default: [],
```

```
 },
 activities: {
   type: Array,
   default: [],
 },
 createdAt: {
   type: Date,
   default: Date.now,
 },
 updatedAt: {
   type: Date,
   default: Date.now,
 },
 lastModifiedAt: {
   type: Date,
   default: Date.now,
 },
})

export default mongoose.model('posts', postSchema);
```

Users:

```
const User = sequelize.define('User', {
 id: {
   type: Sequelize.INTEGER,
   autoIncrement: true,
   primaryKey: true,
   allowNull: false
 },
 full_name: {
   type: Sequelize.STRING,
   allowNull: false
 },
 email: {
   type: Sequelize.STRING,
   allowNull: false
 },
 password: {
   type: Sequelize.STRING,
   allowNull: false
 },
 user_type: {
```

```
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
picture: {
  type: Sequelize.TEXT,
  allowNull: true
},
last_seen: {
  type: Sequelize.DATE,
  allowNull: false
},
reputation: {
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
question_count: {
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
answer_count: {
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
comment_count: {
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
upvotes: {
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
downvotes: {
  type: Sequelize.INTEGER,
  allowNull: false,
  defaultValue: 0
},
```

```
 reach: {
   type: Sequelize.INTEGER,
   allowNull: false,
   defaultValue: 0
 },
 location: {
   type: Sequelize.STRING,
   allowNull: true,
   defaultValue: ""
 },
 about: {
   type: Sequelize.STRING,
   allowNull: true,
   defaultValue: ""
 },
},
 {
   timestamps: true,
   updatedAt: false
 }
)

User.hasMany(Chat)

export default User
```

Chat:

```
const Chat = sequelize.define(
 "chat",
 {
   message_id: {
     type: Sequelize.INTEGER,
     autoIncrement: true,
     primaryKey: true,
     allowNull: false,
   },
   sender_id: {
     type: Sequelize.INTEGER,
     allowNull: false,
   },
   reciever_id: {
     type: Sequelize.INTEGER,
```

```
    allowNull: false,
  },
  message: {
    type: Sequelize.STRING,
    allowNull: false,
  },
 },
 {
  timestamps: true,
  updatedAt: false,
 }
);

export default Chat;
```

Tags:

```
const TagSequelize = sequelize.define('tags', {
 id: {
  type: DataTypes.UUID,
  allowNull: false,
  primaryKey: true,
  defaultValue: DataTypes.UUIDV4,
 },
 tagname: {
  type: DataTypes.STRING(255),
  allowNull: false,
  unique: 'tagname',
 },
 description: {
  type: DataTypes.TEXT,
  allowNull: false,
 },
 questionCount: {
  type: DataTypes.INTEGER,
  allowNull: false,
  defaultValue: 0,
 },
}, {
 sequelize,
 tableName: 'tags',
 underscored: true,
```

```
 timestamps: true,
 indexes: [
   {
     name: 'PRIMARY',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'id' },
     ],
   },
   {
     name: 'tagname',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'tagname' },
     ],
   },
 ],
});

export default TagSequelize;
```

Badge:

```
const Badge = sequelize.define('Badge', {
 id: {
   type: DataTypes.INTEGER,
   autoIncrement: true,
   primaryKey: true,
   allowNull: false
 },
 badge_name: {
   type: DataTypes.STRING,
   allowNull: false
 },
 badge_type: {
   type: DataTypes.STRING,
   allowNull: false
 },
 user_id: {
   type: DataTypes.INTEGER,
```

```
    allowNull: false
  },
},
 {
    timestamps: true,
    updatedAt: false
  }
)

export default Badge;
```

Implementation of security and session objects

```
import bcrypt from 'bcrypt'
import { v4 as uuid } from 'uuid';

const register = async (payload, cb) => {
  const { full_name, email, password } = payload
  try {
    const user = await User.findOne({ where: { email } })
    if (user) {
      return cb("Already registered", null)
    }
    const salt = await bcrypt.genSalt(10)
    const encrypted = await bcrypt.hash(password, salt)

    const newUser = new User({
      id: uuid(),
      full_name: full_name,
      email: email,
      password: encrypted,
      last_seen: Date.now()
    })
    const result = await newUser.save()      Sudheendra Katikar, 6 days ago • User auth flow …
    return cb(null, result)
  } catch (error) {
    console.log(error)
    return cb(error, null)
  }
```

```
const login = async (payload, cb) => {
  const { email, password } = payload
  try {
    const user = await User.findOne({ where: { email } })
    if (!user) {
      return cb("Email not found", null)
    }
    const isMatch = await bcrypt.compare(password, user.password)
    if (!isMatch) {
      return cb("Invalid credentials", null)
    }

    const payload = {
      user: {
        id: user.id
      }
    }

    jwt.sign(
      payload,
      process.env.SECRET_KEY,
      {
        expiresIn: 3600
      },
      (err, token) => {
        if (err) throw err
        return cb(null, {
          user: {
            id: user.id,
            role: user.user_type
          }, "token": "Bearer " + token
        })
```

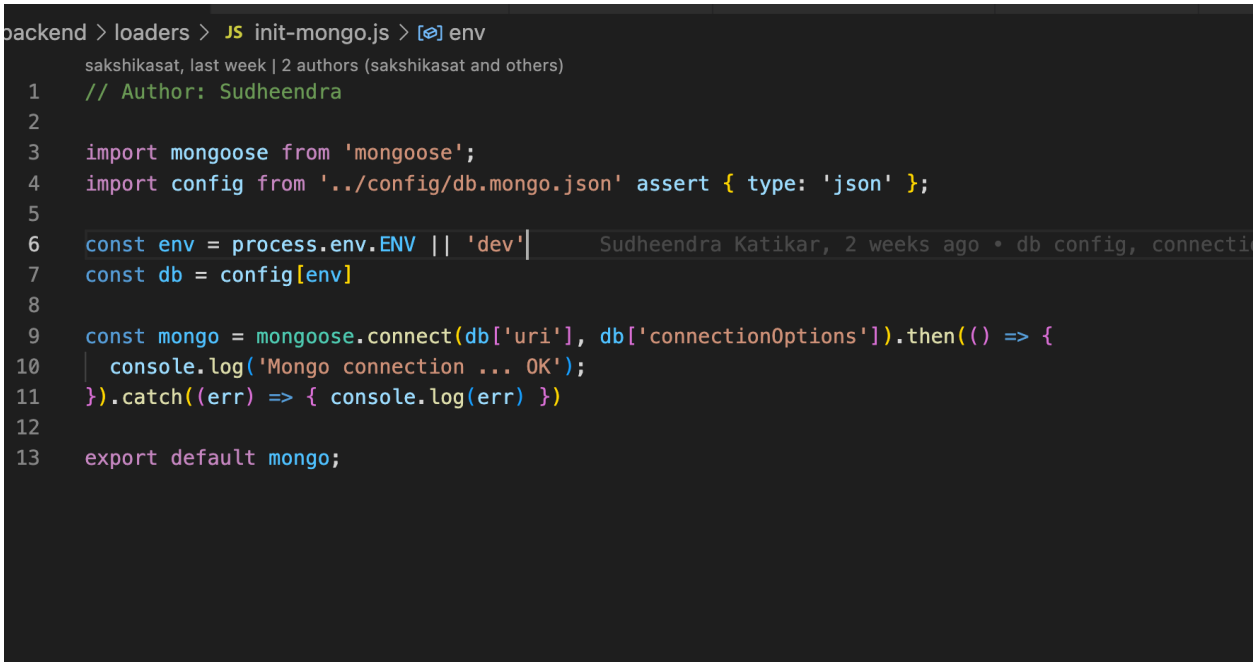
Main server code

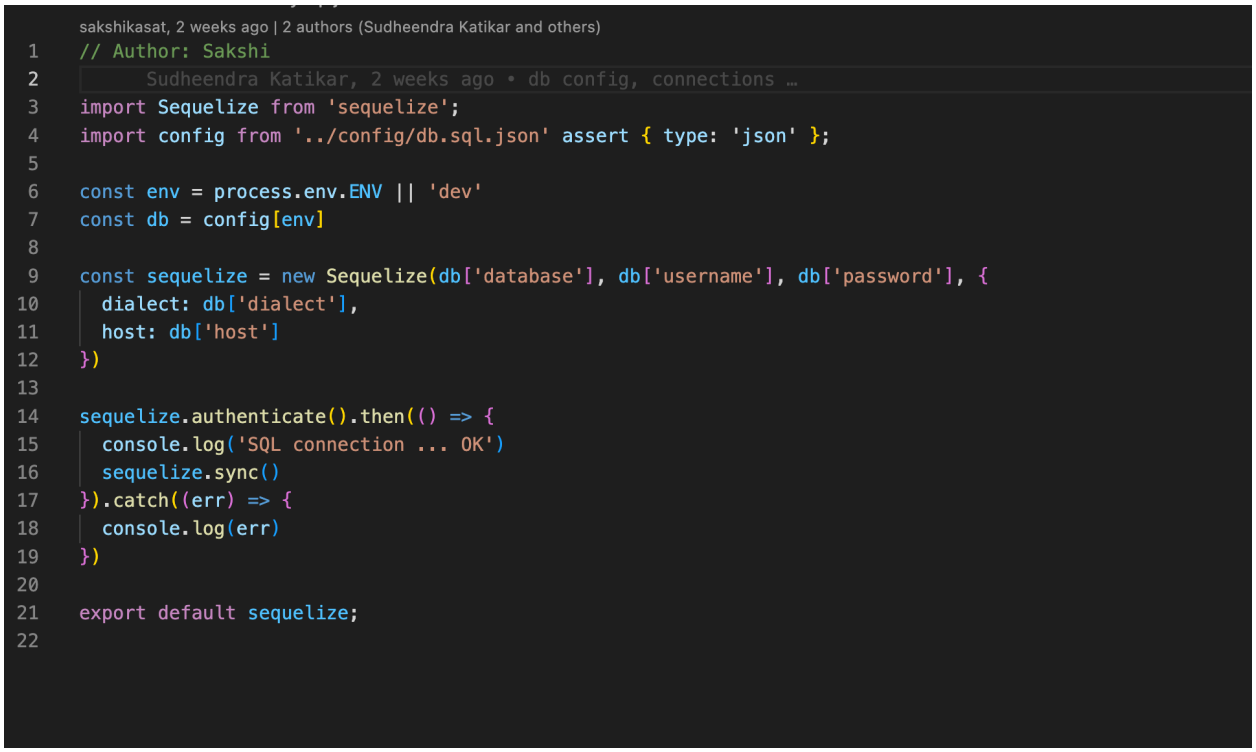
```
import express from 'express';
import cors from 'cors';
import routes from './routes';
import { sql, mongo } from './loaders/db';

const app = express();

const corsOptions = { origin: true, credentials: true };
app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*"); // update to match the domain you will make the request
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  next();
});
app.use('/api', routes);

// app.use(express.static('../frontend/build'))

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`StackOverflow server running on port ${PORT}`);      sakshikasat, 2 weeks ago • API for cr
});
```

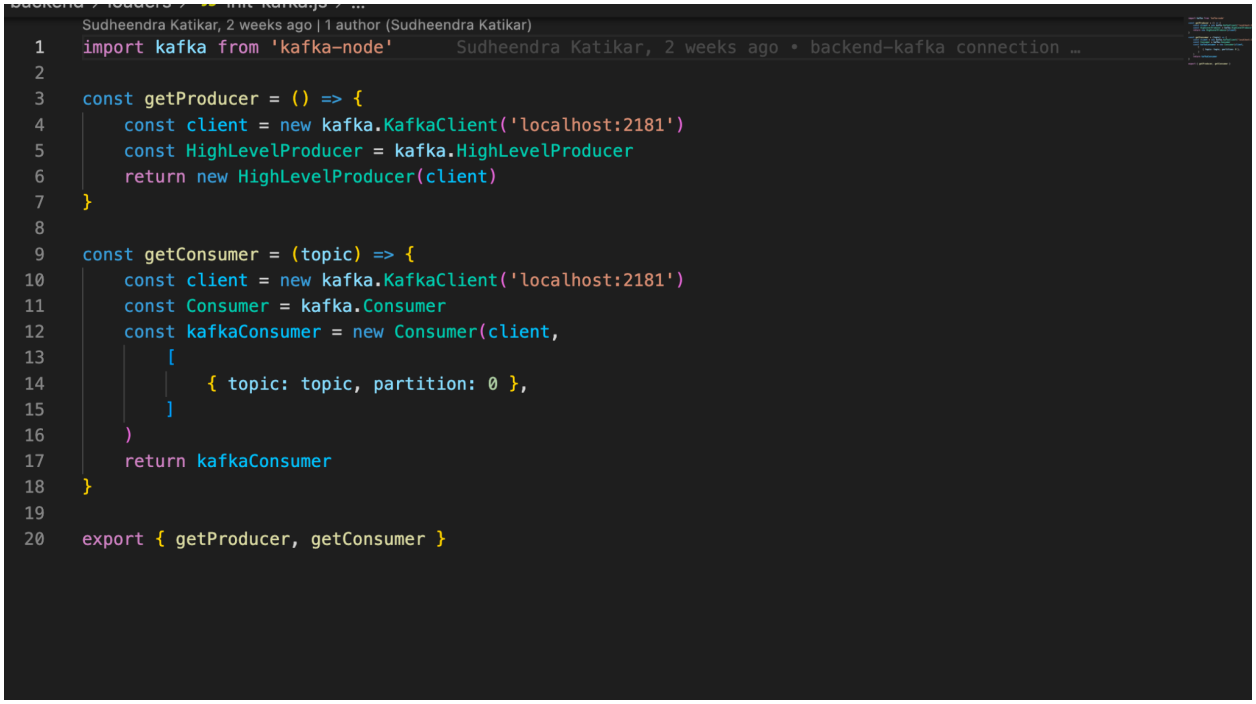
Kafka server code

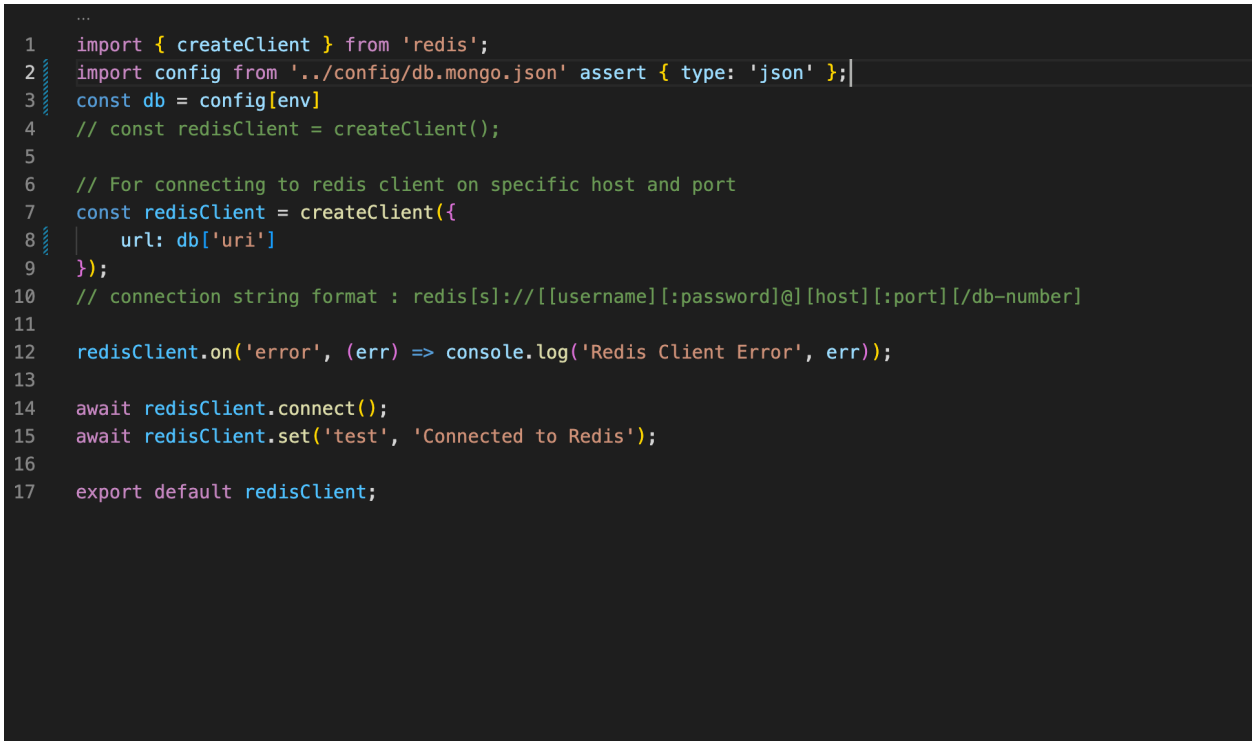
```
import express from 'express'          Sudheendra Katikar,
import dotenv from 'dotenv'

dotenv.config()

import { sql, mongo } from './loaders/db'

import './consumers/authConsumer'
import './consumers/tagConsumer'
import './consumers/postConsumer'
import './consumers/userConsumer'
import './consumers/adminConsumer'

const app = express();

const PORT = process.env.PORT || 8081;
app.listen(PORT, () => {
  console.log(`Kafka server running on port ${PORT}`);
});
```

Database access or connection code


backend > loaders > JS init-mongo.js > env
sakshikasat, last week | 2 authors (sakshikasat and others)
// Author: Sudheendra
import mongoose from 'mongoose';
import config from '../config/db.mongo.json' assert { type: 'json' };
const env = process.env.ENV || 'dev'
Sudheendra Katikar, 2 weeks ago • db config, connecti
const db = config[env]
const mongo = mongoose.connect(db['uri'], db['connectionOptions']).then(() => {
console.log('Mongo connection ... OK');
}).catch((err) => { console.log(err) })
export default mongo;


sakshikasat, 2 weeks ago | 2 authors (Sudheendra Katikar and others)
// Author: Sakshi
Sudheendra Katikar, 2 weeks ago • db config, connections …
import Sequelize from 'sequelize';
import config from '../config/db.sql.json' assert { type: 'json' };
const env = process.env.ENV || 'dev'
const db = config[env]
const sequelize = new Sequelize(db['database'], db['username'], db['password'], {
dialect: db['dialect'],
host: db['host']
})
sequelize.authenticate().then(() => {
console.log('SQL connection ... OK')
sequelize.sync()
}).catch((err) => {
console.log(err)
})
export default sequelize;





Mocha test and output

- Mocha Code:

```
import chai, { expect } from 'chai';
import chaiHttp from 'chai-http';
import casual from 'casual';
import server from '../index';

chai.use(chaiHttp);

describe('API testing', () => {
 const email = casual.email;
 const password = casual.password;
 const full_name = casual.name;

 let userId, postId, answerId;

 const title = casual.word;
 const body = casual.description;
 const tags = ['python', 'javascript'];
 const approved = false;

 it('Checks Register API and returns status code', (done) => {
   chai.request(server)
     .post('/api/auth/register')
     .send({
       full_name,
       email,
       password,
     })
     .end((err, res) => {
       expect(res.statusCode).to.eq(200);
       expect(res.body).to.have.property('id');
       expect(res.body).to.have.property('full_name');
       expect(res.body).to.have.property('email');
       expect(res.body.full_name).to.equal(full_name);
       expect(res.body.email).to.equal(email);
       done();
     });
 });

 it('Checks Login API and returns status code', (done) => {
   chai.request(server)
     .post('/api/auth/login')
     .send({
```

```
      email,
      password,
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      expect(res.body).to.have.property('user');
      expect(res.body.user).to.have.property('id');
      userId = res.body.user.id;
      done();
    });
});

it('Checks Create posts API and returns status code', (done) => {
  chai.request(server)
    .post('/api/posts')
    .send({
      title,
      body,
      tags,
      ownerId: userId,
      approved,
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      expect(res.body).to.have.property('_id');
      expect(res.body).to.have.property('title');
      expect(res.body).to.have.property('body');
      expect(res.body).to.have.property('tags');
      expect(res.body).to.have.property('ownerId');
      expect(res.body).to.have.property('approved');
      expect(res.body).to.have.property('answers');
      expect(res.body.title).to.equal(title);
      expect(res.body.body).to.equal(body);
      expect(res.body.tags).to.deep.equal(tags);
      expect(res.body.ownerId).to.equal(userId);
      expect(res.body.approved).to.equal(approved);
      postId = res.body._id;
      done();
    });
});

it('Checks Get all interesting posts API and returns status code', (done) => {
```

```
  chai.request(server)
    .get('/api/posts/getInteresting')
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      res.body.forEach(element => {
        expect(element.post).to.have.property('_id');
        expect(element.post).to.have.property('title');
        expect(element.post).to.have.property('body');
        expect(element.post).to.have.property('tags');
        expect(element.post).to.have.property('ownerId');
        expect(element.post).to.have.property('approved');
        expect(element.post).to.have.property('answers');
      });
      done();
    });
});

it('Checks posts answer API and returns status code', (done) => {
  chai.request(server)
    .post('/api/posts/answer')
    .send({
      questionId: postId,
      body: casual.description,
      ownerId: userId
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      done();
    });
});

it('Checks Get single posts API and returns status code', (done) => {
  chai.request(server)
    .get(`/api/posts/${postId}`)
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      expect(res.body).to.have.property('_id');
      expect(res.body).to.have.property('title');
      expect(res.body).to.have.property('body');
      expect(res.body).to.have.property('tags');
      expect(res.body).to.have.property('approved');
      expect(res.body).to.have.property('ownerId');
```

```
      expect(res.body).to.have.property('answers');
      expect(res.body.title).to.equal(title);
      expect(res.body.body).to.equal(body);
      expect(res.body.tags).to.deep.equal(tags);
      expect(res.body.approved).to.equal(approved);
      expect(res.body.ownerId).to.equal(userId);
      answerId = res.body.answers[0].id;
      done();
    });
});

it('Checks posts comment API and returns status code', (done) => {
  chai.request(server)
    .post('/api/posts/comment')
    .send({
      parentId: postId,
      comment: casual.description,
      userId,
      userName: full_name
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      done();
    });
});

it('Checks posts vote API and returns status code', (done) => {
  chai.request(server)
    .post('/api/posts/voteQuestion')
    .send({
      questionId: postId,
      userId,
      value: 5
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      done();
    });
});

it('Checks mark posts accepted API and returns status code', (done) => {
  chai.request(server)
```

```
    .post('/api/posts/markAccepted')
    .send({
      questionId: postId,
      userId,
      answerId,
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      done();
    });
});

it('Checks posts bookmark API and returns status code', (done) => {
  chai.request(server)
    .post('/api/user/bookmark')
    .send({
      postId,
      userId,
    })
    .end((err, res) => {
      expect(res.statusCode).to.eq(200);
      done();
    });
});
});
```

- Mocha output screenshots:

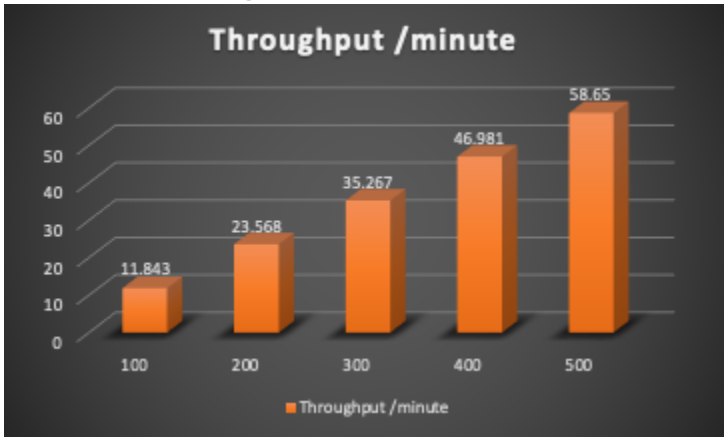
```
> stackoverflow@1.0.0 test
> mocha --experimental-specifier-resolution=node --exit

(node:17804) ExperimentalWarning: The Node.js specifier resolution flag is experimental. It could change or be removed at any time.
(Use `node --trace-warnings ...` to show where the warning was created)
(node:17804) ExperimentalWarning: Importing JSON modules is an experimental feature. This feature could change at any time
StackOverflow server running on port 8080


  API testing
POST /register => {"full_name":"Dr. Giovanny Howell","email":"Kadin.Bashirian@Predovic.ca","password":"5Diego52"}
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'auth',
    messages: '{"payload":{"full_name":"Dr. Giovanny Howell","email":"Kadin.Bashirian@Predovic.ca","password":"5Diego52","action":"REGISTER"},"correlationId":"fafdd428-4ca5-4b11-a5b4-87cb
354bf6d"}',
    partition: 0
  }
]
ERR null
DATA { auth: { '0': 137 } }
    ✔ Checks Register API and returns status code (231ms)
POST /login => {"email":"Kadin.Bashirian@Predovic.ca","password":"5Diego52"}
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
Executing (default): SELECT 1+1 AS result
Executing (default): SELECT 1+1 AS result
3 -> Producer write to topic ... [
  {
    topic: 'auth',
    messages: '{"payload":{"email":"Kadin.Bashirian@Predovic.ca","password":"5Diego52","action":"LOGIN"},"correlationId":"8bda2b36-78d3-40d0-8652-7b75f0ae1cba"}',
    partition: 0
  }
]
ERR null
DATA { auth: { '0': 138 } }
SQL connection ... OK
Executing (default): CREATE TABLE IF NOT EXISTS `Badges` (`id` INTEGER NOT NULL auto_increment , `badge_name` VARCHAR(255) NOT NULL, `badge_type` VARCHAR(255) NOT NULL, `user_id` INTEGER
OT NULL, `createdAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
SQL connection ... OK
Executing (default): SELECT 1+1 AS result
Executing (default): SHOW INDEX FROM `Badges`
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER NOT NULL auto_increment , `full_name` VARCHAR(255) NOT NULL, `email` VARCHAR(255) NOT NULL, `password` VARCHAR(255) N
T NULL, `user_type` INTEGER NOT NULL DEFAULT 0, `picture` TEXT, `last_seen` DATETIME NOT NULL, `reputation` INTEGER NOT NULL DEFAULT 0, `question_count` INTEGER NOT NULL DEFAULT 0, `answe
_count` INTEGER NOT NULL DEFAULT 0, `comment_count` INTEGER NOT NULL DEFAULT 0, `upvotes` INTEGER NOT NULL DEFAULT 0, `downvotes` INTEGER NOT NULL DEFAULT 0, `reach` INTEGER NOT NULL DEFA
LT 0, `location` VARCHAR(255) DEFAULT '', `about` VARCHAR(255) DEFAULT '', `createdAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `Users`
    ✔ Checks Login API and returns status code (94ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"title":"ipsum","body":"Dolor quidem pariatur asperiores. Delectus repellendus quidem vitae laudantium aut laboriosam enim. Voluptas assumenda atque. Repellendu
 eius ab nulla dolor sit ea vel sed expedita.","tags":["python","javascript"],"ownerId":85610374,"approved":false,"action":"ADD_POST"},"correlationId":"b35e99f0-8fba-4a96-9dd5-fdb10c4c539
```

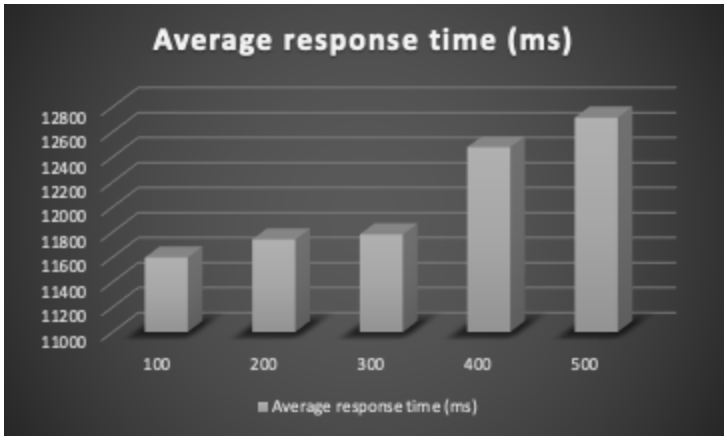
```
  {
    topic: 'posts',
    messages: '{"payload":{"title":"ipsum","body":"Dolor quidem pariatur asperiores. Delectus repellendus quidem vitae laudantium aut laboriosam enim. Voluptas assumenda atque. Repellendus
 eius ab nulla dolor sit ea vel sed expedita.","tags":["python","javascript"],"ownerId":85610374,"approved":false,"action":"ADD_POST"},"correlationId":"b35e99f0-8fba-4a96-9dd5-fdb10c4c5394
"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 416 } }
    ✔ Checks Create posts API and returns status code (164ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"action":"GET_INTERESTING"},"correlationId":"00ce5ed2-f8b9-438d-9e33-37c1369dacd7"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 417 } }
Mongo connection ... OK
    ✔ Checks Get all interesting posts API and returns status code (413ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"questionId":"627f209073cdb158062cd052","body":"Sint aliquid quia fugiat tempora rerum. Vel mollitia aut et. Ipsum sit quia provident perspiciatis porro officia.
","ownerId":85610374,"action":"ADD_ANSWER"},"correlationId":"2cdd0428-e0df-499b-a1d0-c7585260l023"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 418 } }
    ✔ Checks posts answer API and returns status code (70ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"id":"627f209073cdb158062cd052","action":"GET_SINGLE_POST"},"correlationId":"830dc3db-e985-431c-b252-8545128a8d5b"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 419 } }
    ✔ Checks Get single posts API and returns status code (78ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"parentId":"627f209073cdb158062cd052","comment":"Et eos possimus error iure doloribus maxime optio. Et quia ipsam ut. Dolores magni est est nesciunt aliquid veri
catis possimus eligendi.","userId":85610374,"userName":"Dr. Giovanny Howell","action":"ADD_COMMENT"},"correlationId":"8a612d80-fe22-4e84-b67d-b61232d94bea"}',
    partition: 0
  }
```

```
ERR null
DATA { posts: { '0': 419 } }
    ✔ Checks Get single posts API and returns status code (78ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"parentId":"627f209073cdb158062cd052","comment":"Et eos possimus error iure doloribus maxime optio. Et quia ipsam ut. Dolores magni est est nesciunt aliquid ve
tatis possimus eligendi.","userId":85610374,"userName":"Dr. Giovanny Howell","action":"ADD_COMMENT"},"correlationId":"8a612d80-fe22-4e84-b67d-b61232d94bea"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 420 } }
    ✔ Checks posts comment API and returns status code (77ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"userId":85610374,"questionId":"627f209073cdb158062cd052","value":5,"action":"VOTE_QUESTION"},"correlationId":"d26609fe-2510-45a9-b8eb-f8aa0252c178"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 421 } }
    ✔ Checks posts vote API and returns status code (126ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'posts',
    messages: '{"payload":{"userId":85610374,"questionId":"627f209073cdb158062cd052","answerId":"627f209173cdb158062cd055","action":"MARK_ACCEPTED"},"correlationId":"01a44755-2db7-4845-8
7-e4cf98aa2b43"}',
    partition: 0
  }
]
ERR null
DATA { posts: { '0': 422 } }
    ✔ Checks mark posts accepted API and returns status code (215ms)
1 -> Sending request to Kafka consumer ...
2 -> CorrelationID and requests ...
3 -> Producer write to topic ... [
  {
    topic: 'users',
    messages: '{"payload":{"userId":85610374,"postId":"627f209073cdb158062cd052","action":"BOOKMARK_POST"},"correlationId":"60379b32-8c6c-42ce-9e62-fdd73e18d5e0"}',
    partition: 0
  }
]
ERR null
DATA { users: { '0': 33 } }
    ✔ Checks posts bookmark API and returns status code (89ms)


  10 passing (2s)

sakshi@Sakshis-MacBook-Pro backend %
```

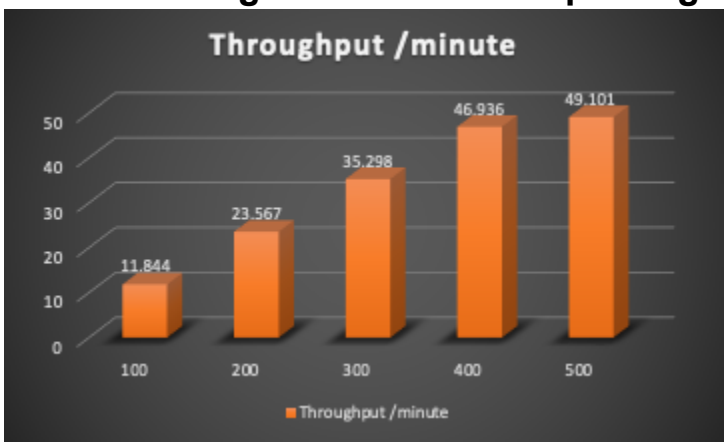
Performance graphs

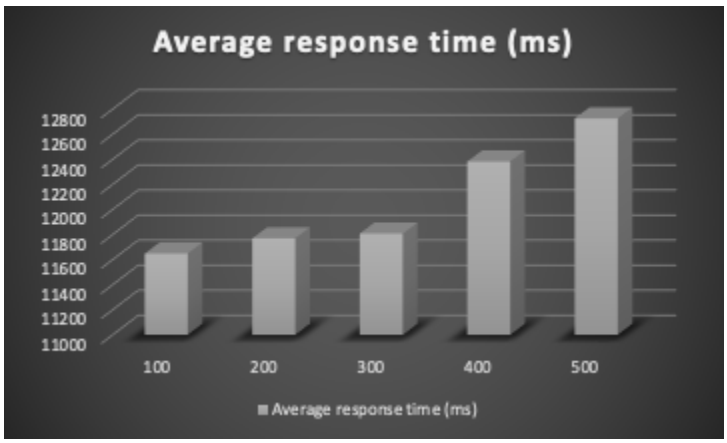
- **JMeter Testing B (Base)**



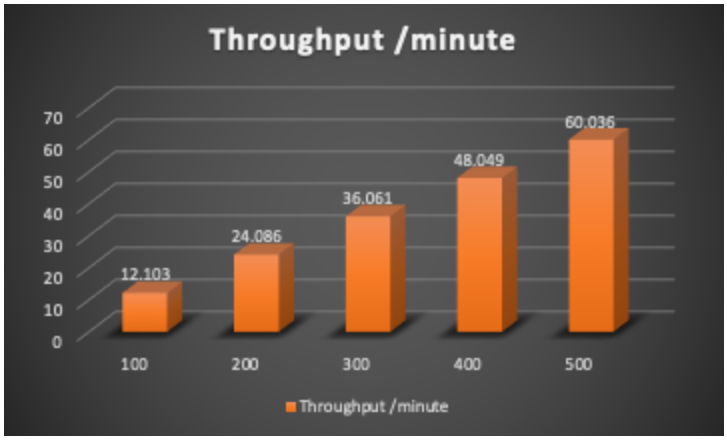


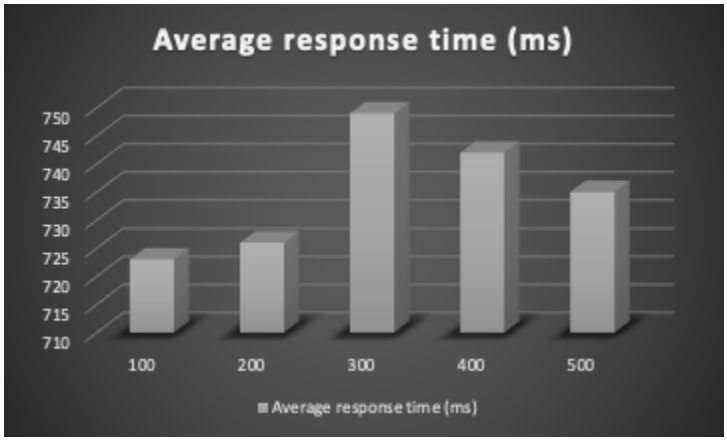
- **JMeter Testing with connection pooling B+D**



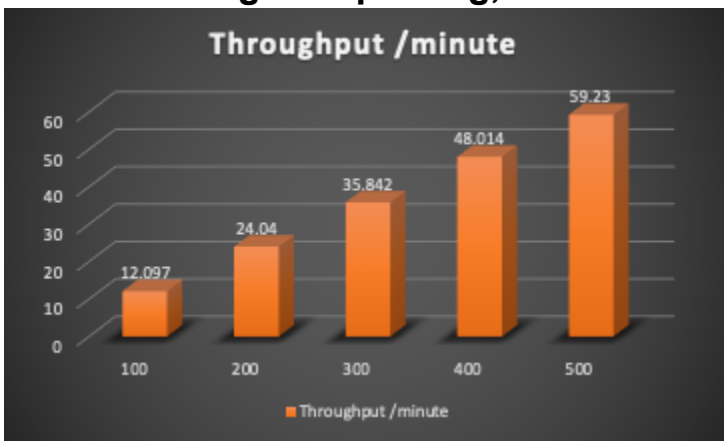


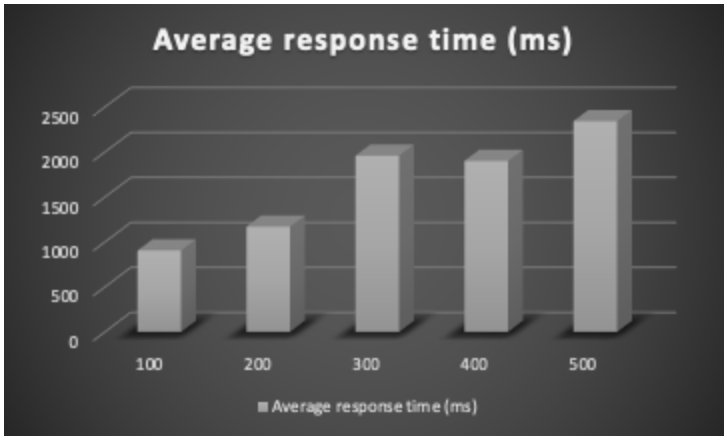
- **JMeter Testing with connection pooling and Redis (B + D + S)**



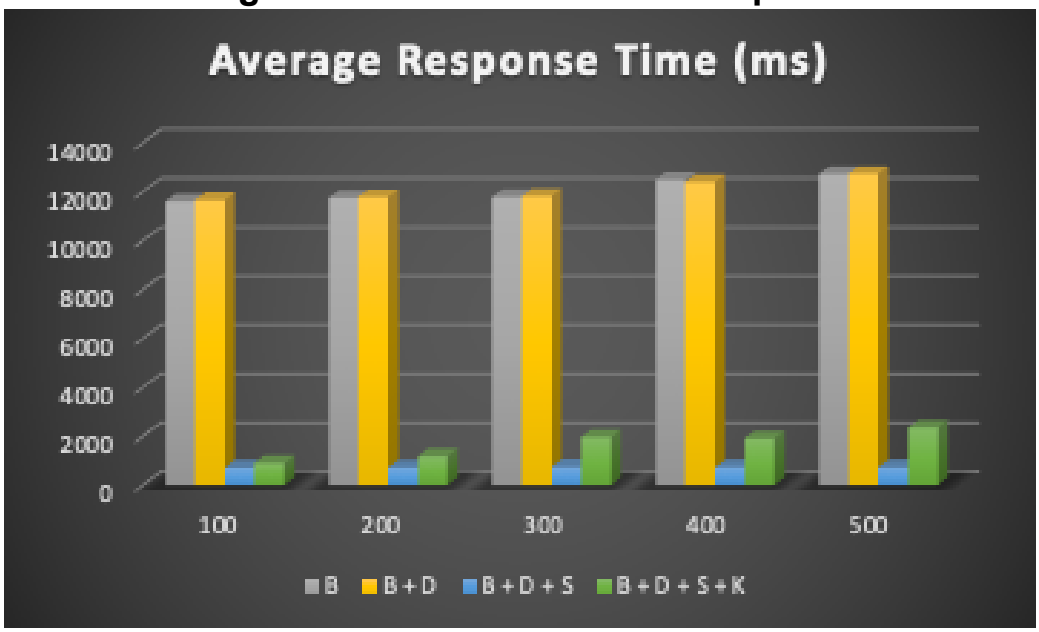


- **JMeter Testing with pooling, redis and Kafka (B + D + S + K)**

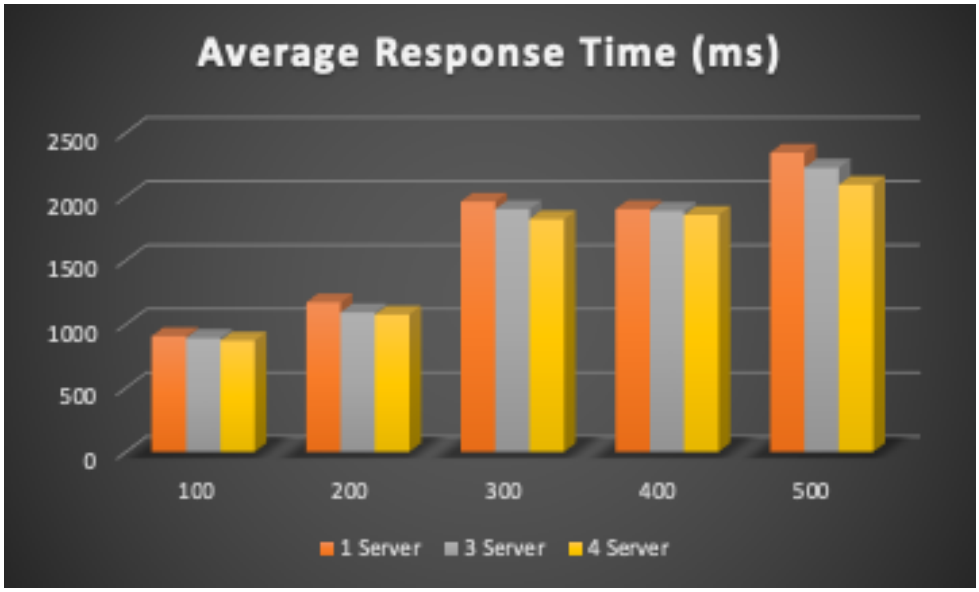




- **JMeter Testing - Overall Performance Comparison**



- **Performance Comparison with Load Balancer**



Github Link:
https://github.com/anirud55/cmpe273-stackoverflow-team-project