Term Project – REDDIT APPLICATION

Team Members

- Het Brahmbhatt
- Monil Sakhidas
- Priyansh Manish Patel
- Poonam Yadav
- Shakthivel Ramesh Nirmala
- Sahil Bhagchandani

Team Members Contribution

Shakthivel:

- Backend: Signup and login APIs with passportjs, User profile page APIs, Chat Module APIs, Community Analytics APIs, Migration from MongoDB to Amazon RDS for implementation of sign up and login features using MySQL.

Monil:

- Backend: Community Moderation APIs, Invitations and Notifications APIs, Posts CRUD APIs, Nested Comments CRUD APIs, Post and Comment upvoting and downvoting APIs

Poonam:

- Backend: Kafka Setup, S3 setup, Redis Setup, Community CRUD APIs, Community Search APIs, Community upvoting and downvoting, Dashboard APIs
- Frontend: Community Analytics, Notifications

- Project Management: scoped the tasks, used agile methodology to track the progress in Jira

Priyansh:

- Frontend and UI: Navbar, Integrated profile UI and profile backend, My Communities page, Community Moderation module along with allowing to accept multiple users' request or remove a user from multiple communities at once, all through multiple modals and tab view. And developed Create Post page to create posts of three different kinds with tab view implementation.
- Created modular UI components to share between different screens.
- Solved react router redirecting issues.
- Redux implementation at each level.

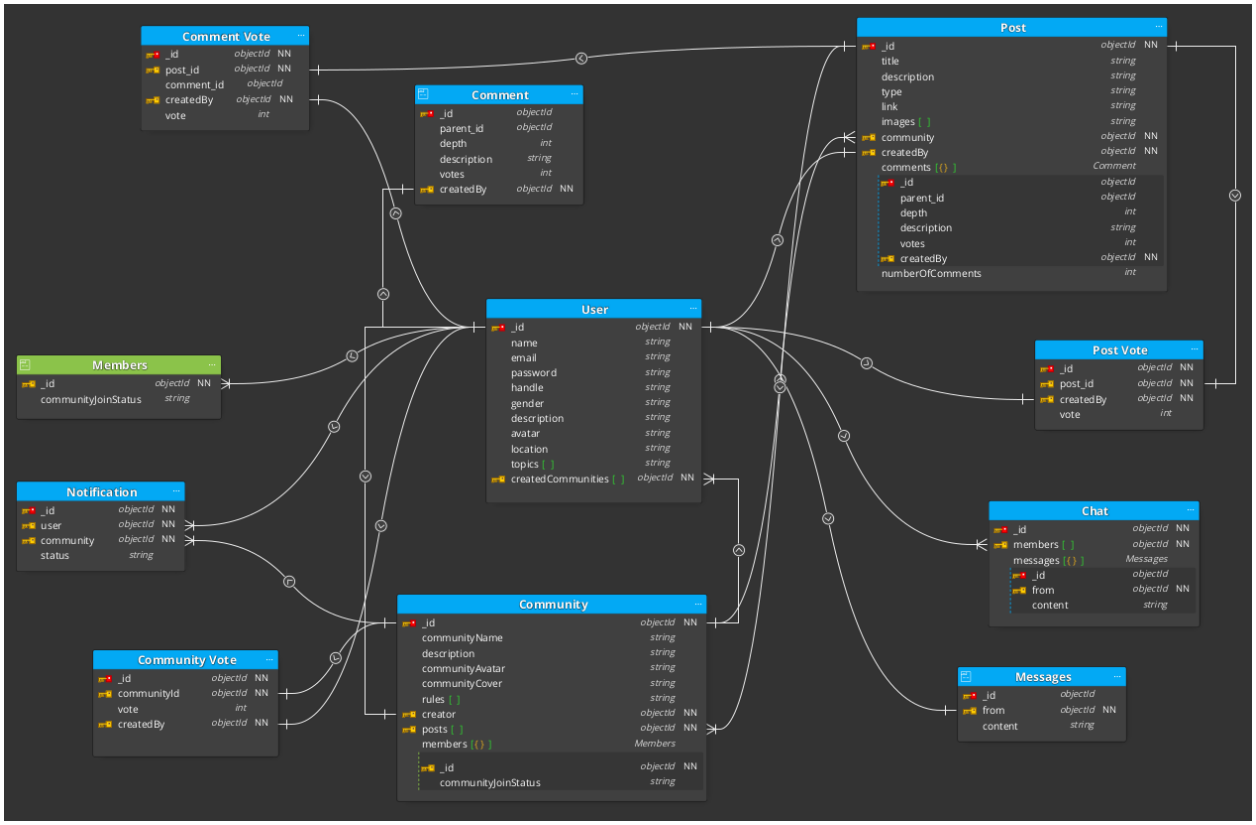
HET:

- Frontend and UI:
    - Login and SignUp Modal,Profile page basic,Create Community and Edit Community (both in same page based on params in frontend),
    - Community Search Page (Pagination,Dropdown,filter), Community Home Page (Upvote, downvote,nested comments), Dashboard, My messages(Chat Modal ,used react-chat-elements to display messages like web application), Invitations.
    - Redux implementation at each level

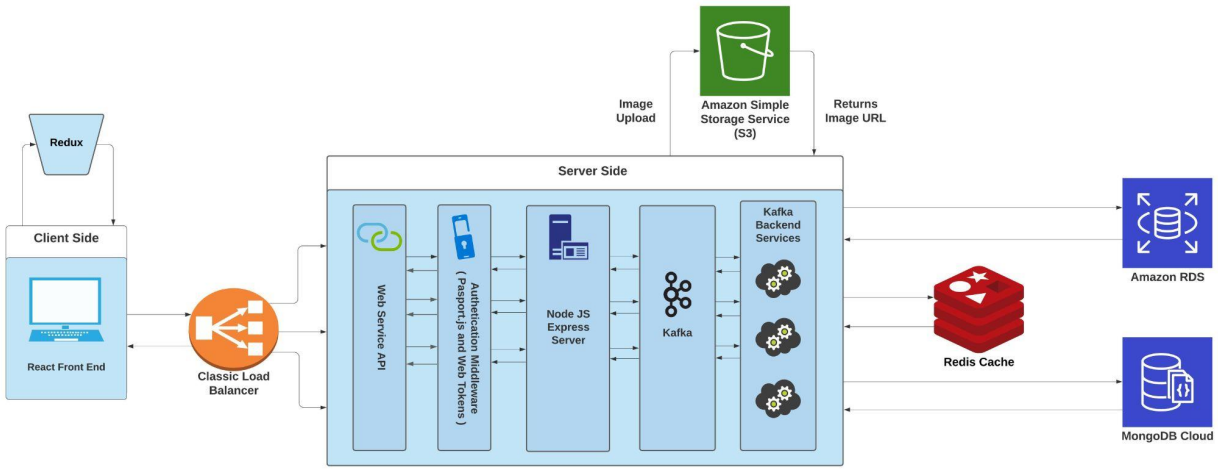
SAHIL:

- Frontend and UI:
    - User Profile Page (list of communities they joined), Community Analytics Page (used react-google-charts), Community Moderation Page (pagination,search).
    - Frontend testing, and created modular UI components such as filter drop downs. Implemented and enforced visually similarity between components.

Database Schema



System Architecture Diagram



Object Management Policy

- Direct state changes were avoided in all the react components.
- Redis Cache was used for frequently fetched data.
- Multiple state changes were batched in setState that calls in the update only once and avoids the performance issues.
- The state was passed from parent to child components only if it was required.
- Top down or unidirectional data flow was maintained and any changes done in the data or state of any particular component in a tree was known only to the child components.

Handle Heavy Weight Resources

- APIs were specific designed for the use cases separately to return the minimal possible data so that response doesn't become big
- Lodash was used to handle heavy data objects for iteration.
- Component states were managed in such a way that they were updated only when it was required.
- `Eslint-Plugin-react` was used in order to forcefully adapt a lot of rules while building the application so that it does not cause any problems in the long run.
- Functions were binded early to handle the events of child components.

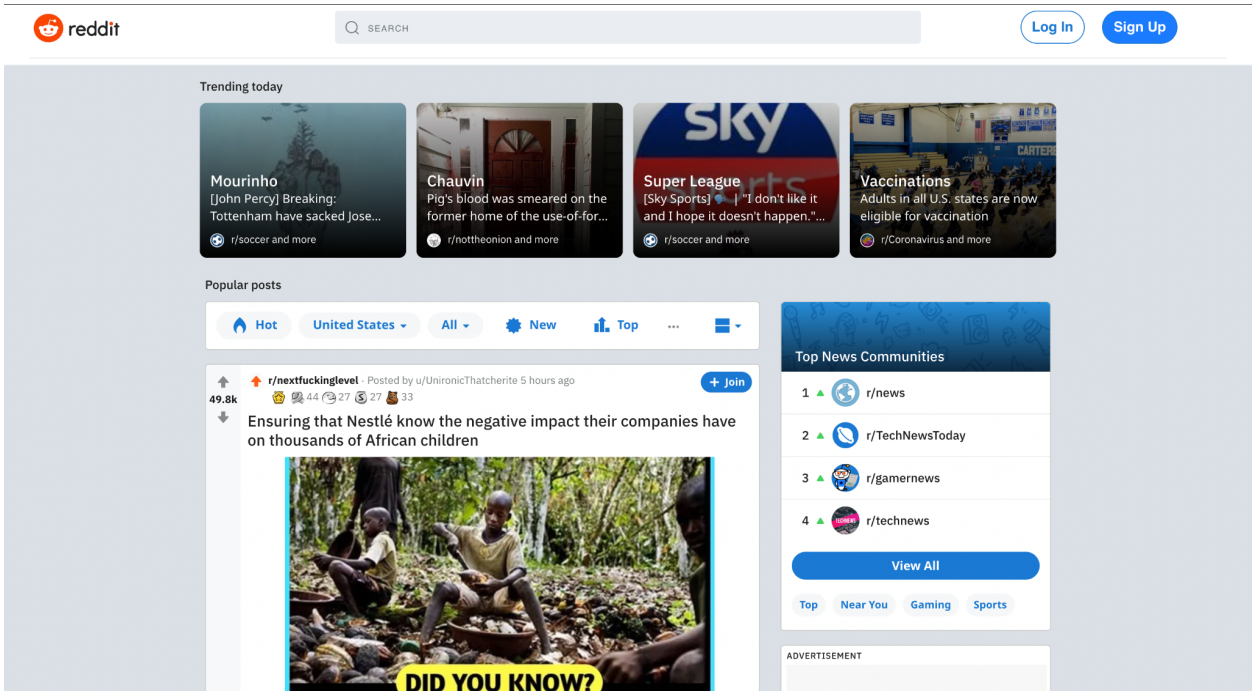
- Object.assign was used in complex data objects in order to copy the changes from one object to another object.This made detecting changes in data as simple as comparing the reference of the two objects.
- shouldComponentUpdate was used in order to avoid the render diff-process from triggering for those components whose data was getting changed frequently.
- Components structure was created carefully in order to manage the state changes in a better way and eventually avoiding the re-rendering of those components that were not needed to be re rendered.
- Route based code splitting was done to break down the whole application into chunks.

Policy Used To Decide When To Write Data Into Database

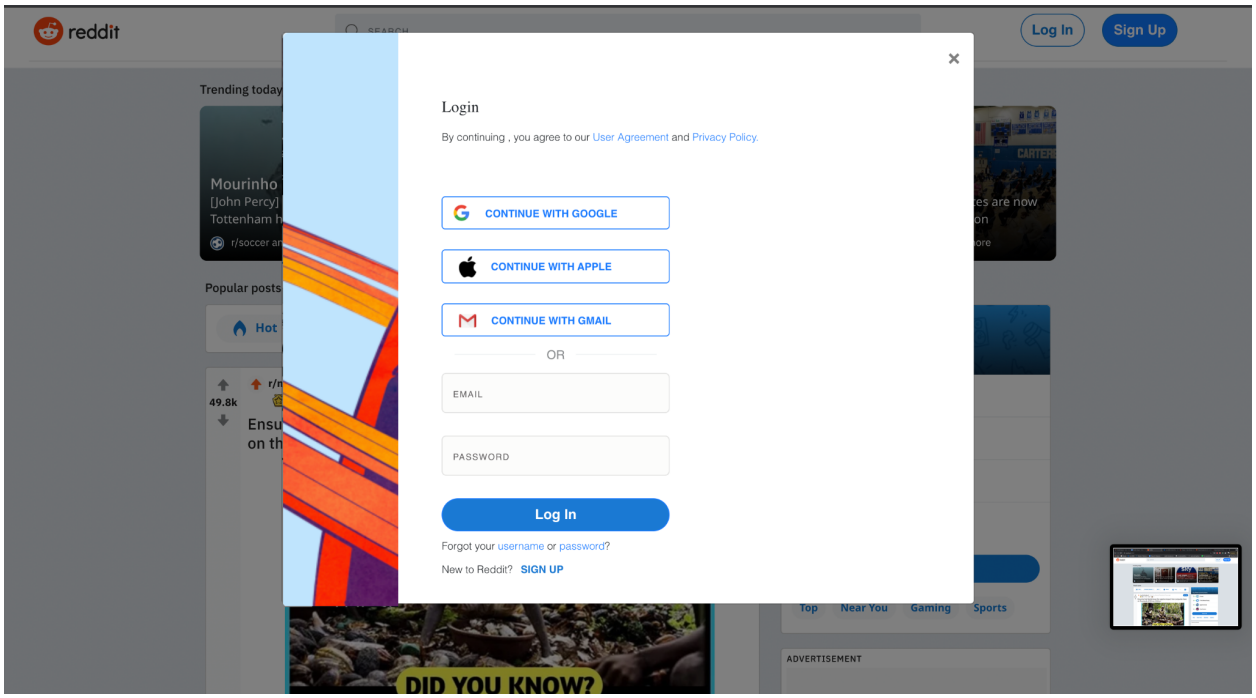
- The data was broken into logical pieces.
- Duplicate data was avoided in all the collections if performance was not the main issue.
- The data which was not changing a lot during a long period of time was stored in Redis Cache to enhance the performance overall.
- If the data was in bulk then upsert many was used in order to avoid the multiple handshake time.

Screen Captures Of The Application

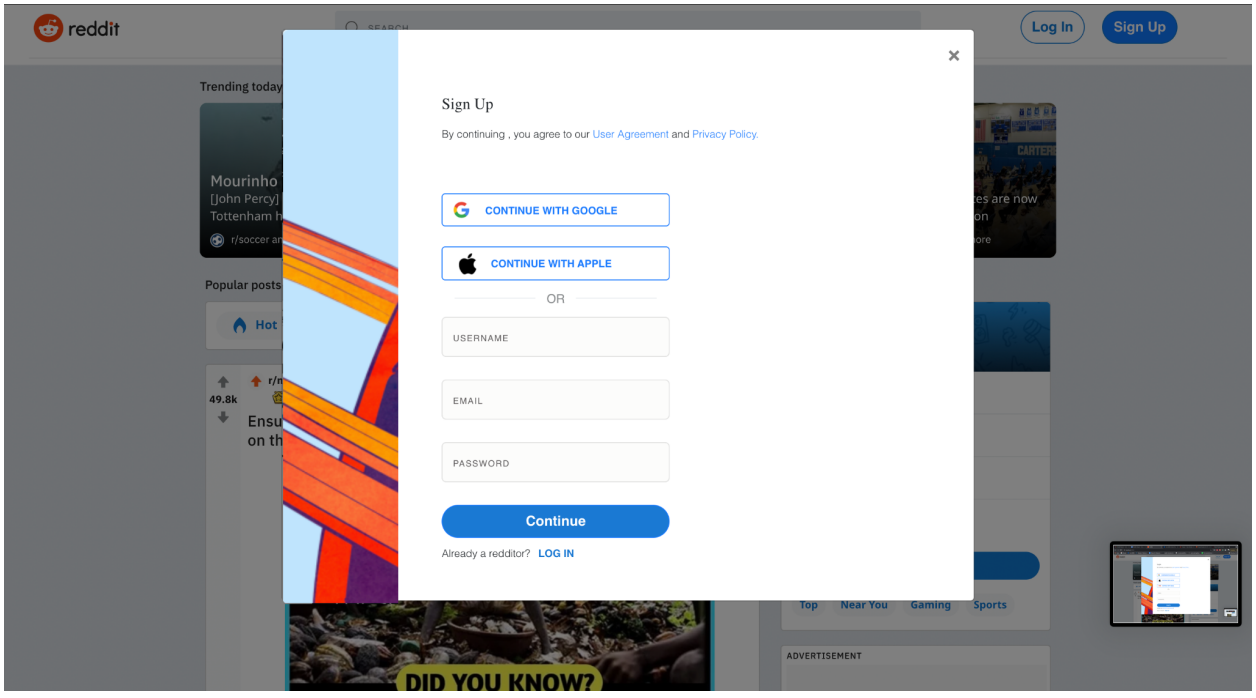
Landing Page



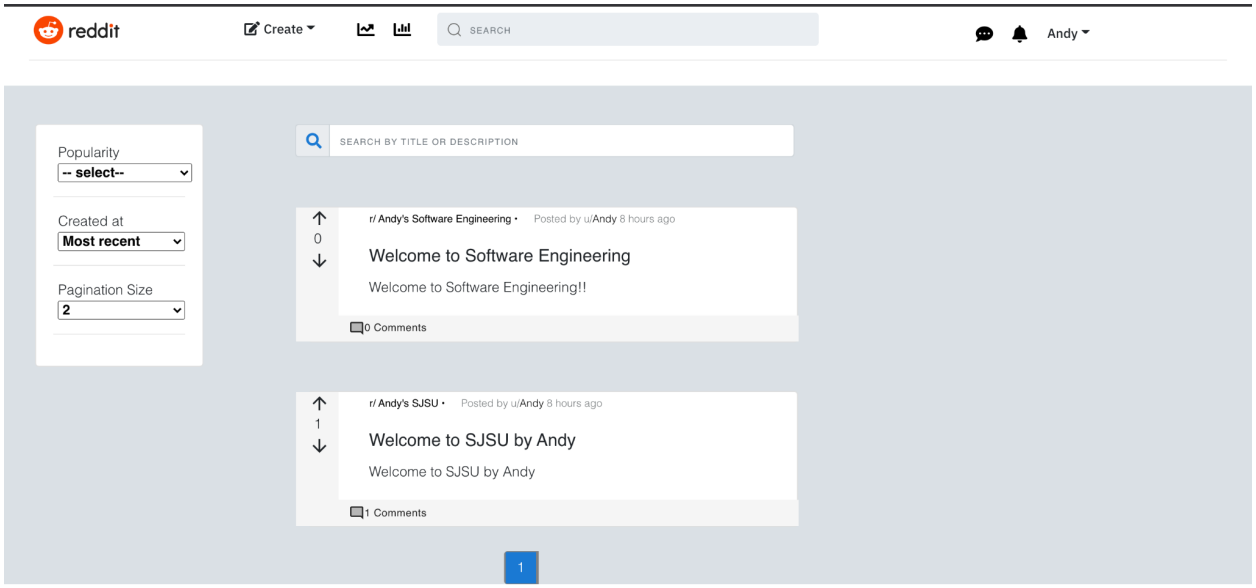
Login



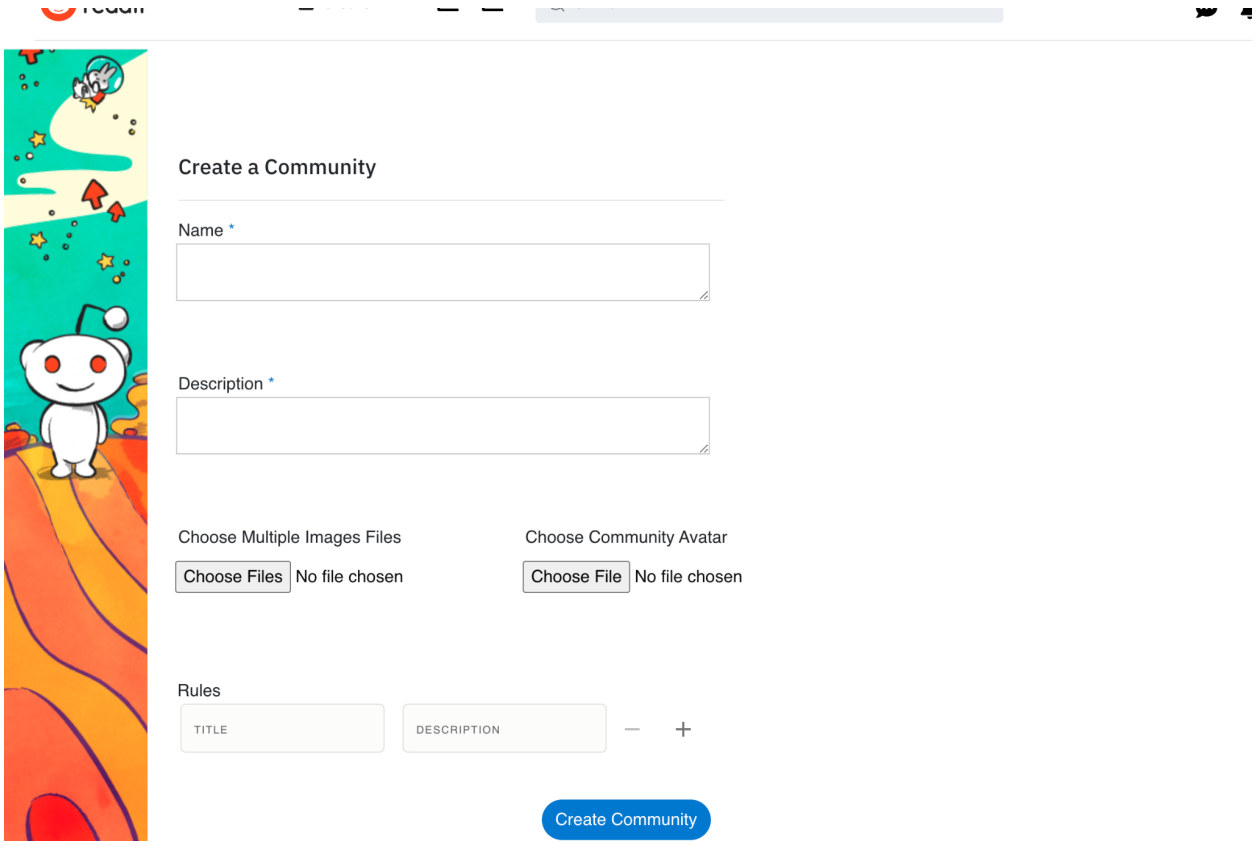
Sign Up



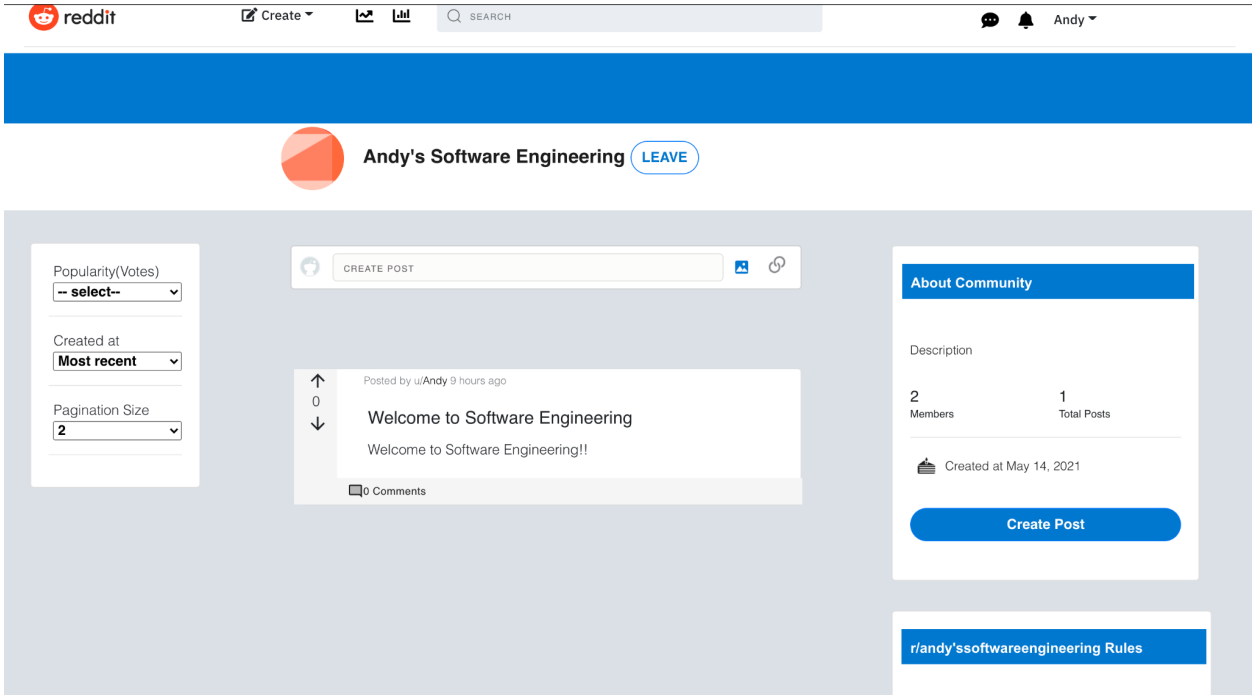
Dashboard



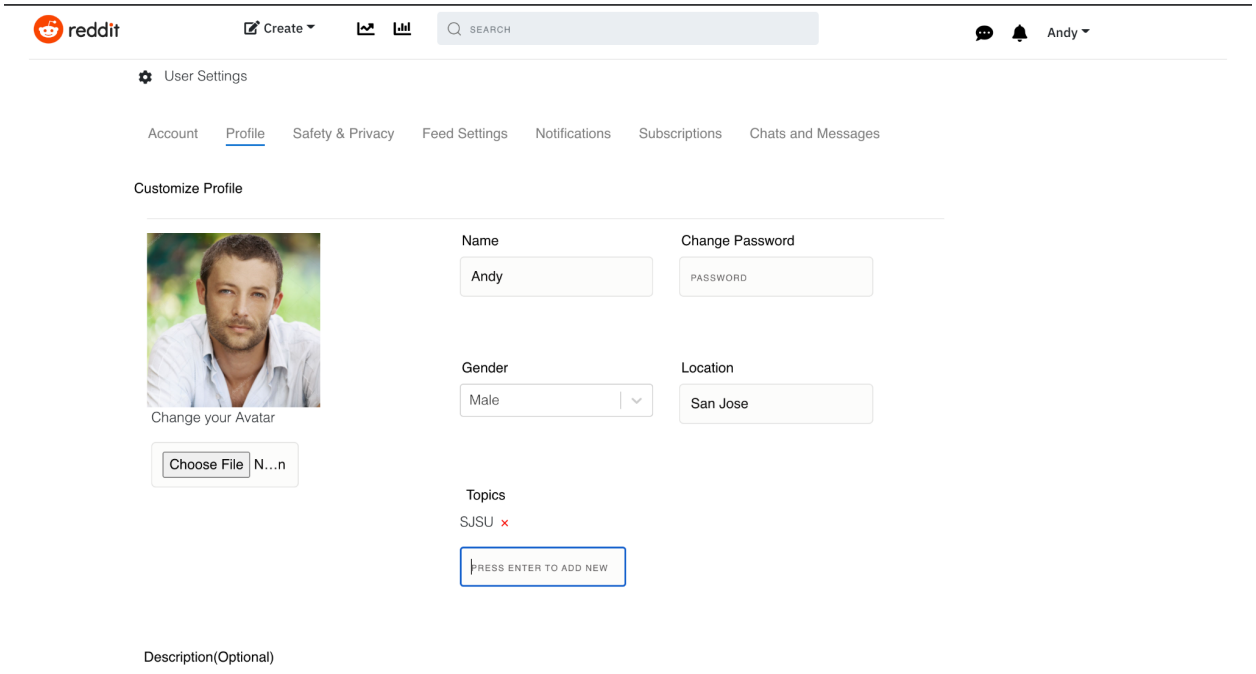
Create Community



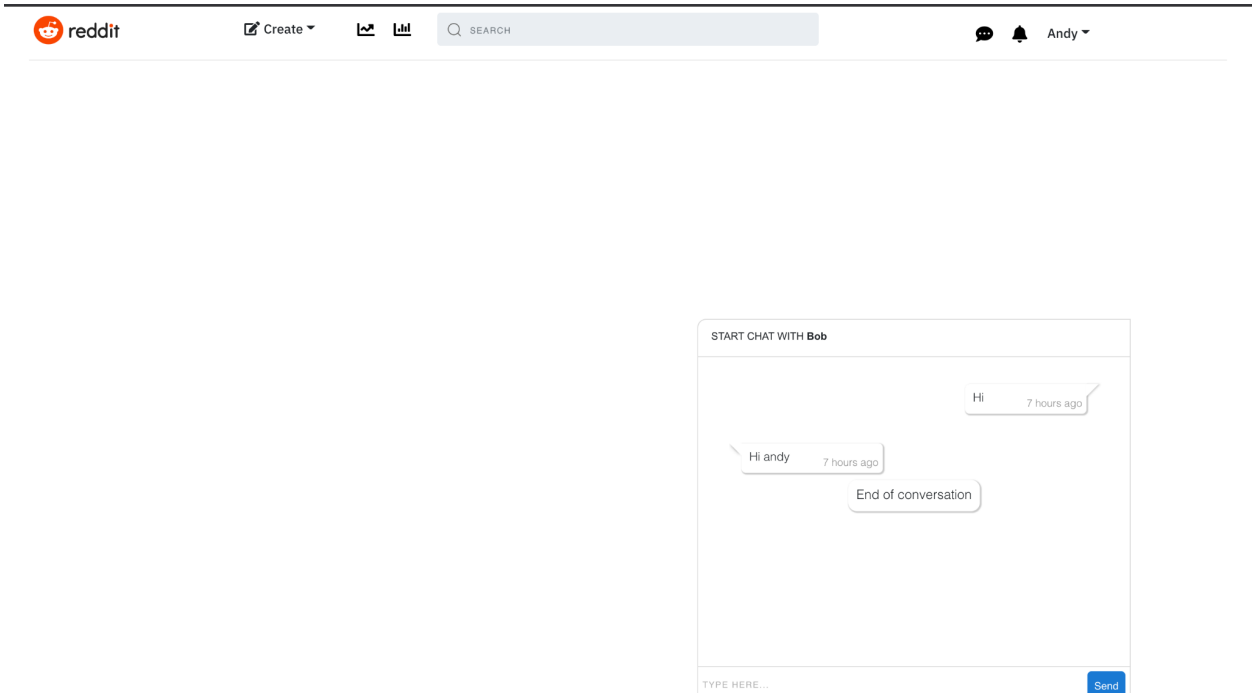
Community Home Page



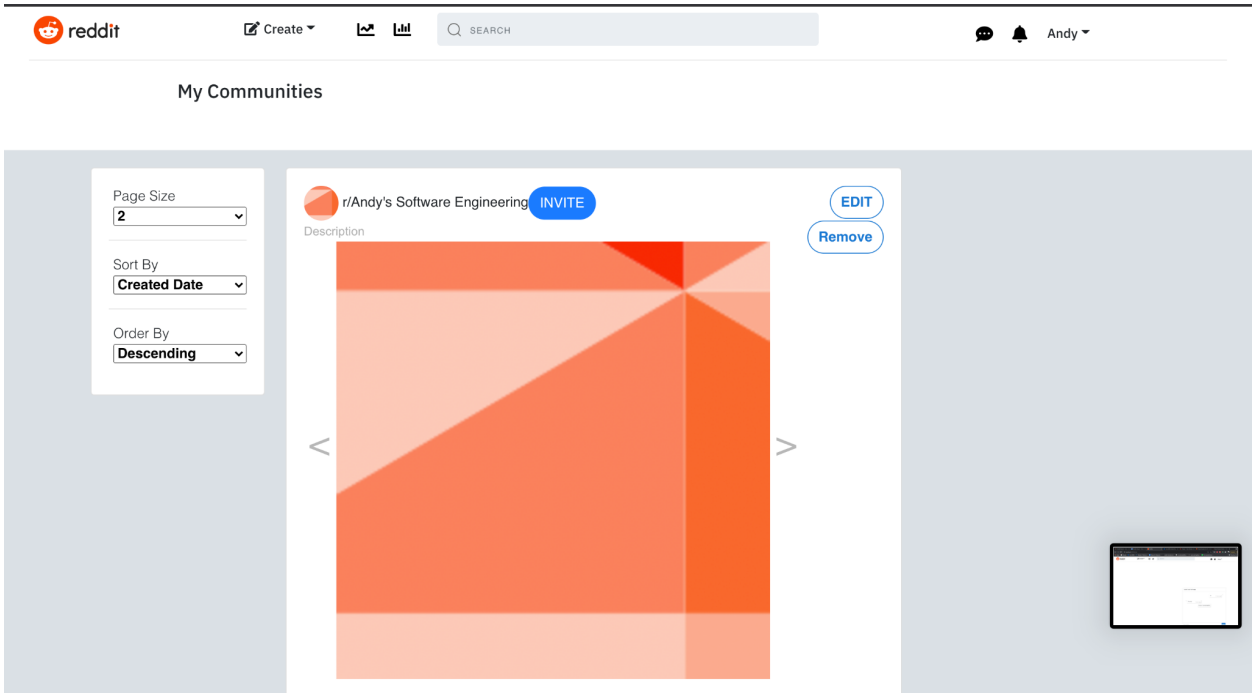
Profile Page



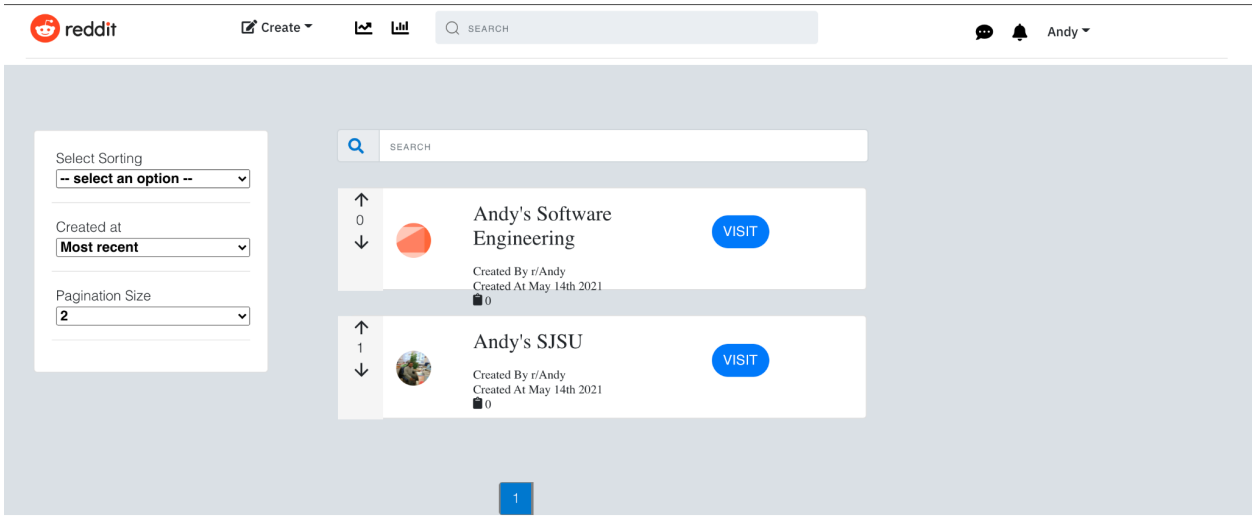
Messages



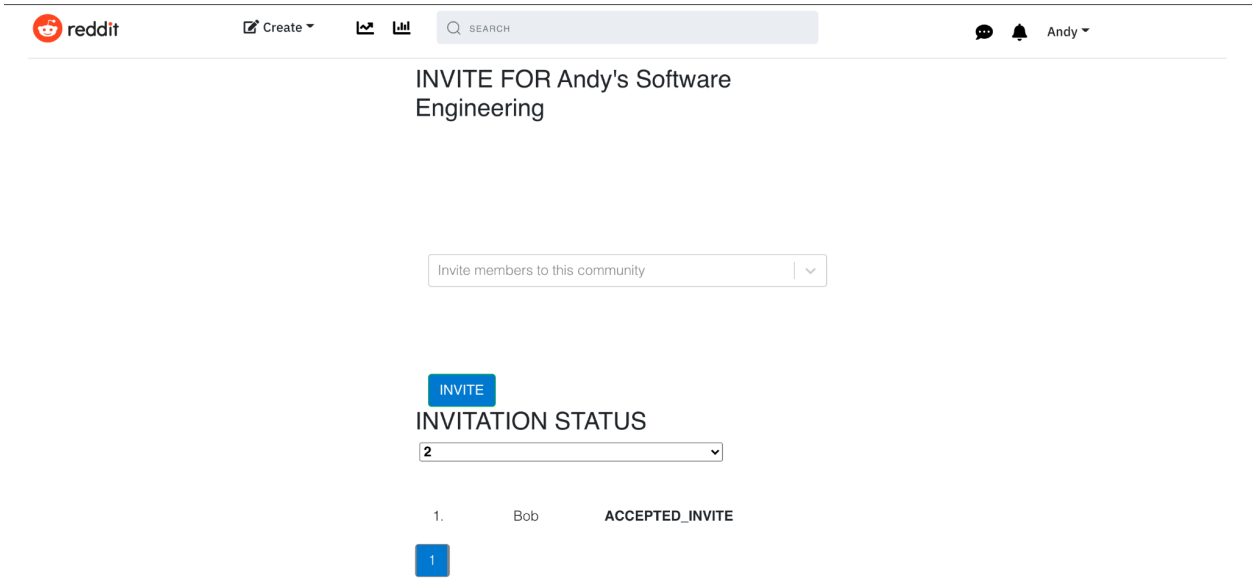
My Communities



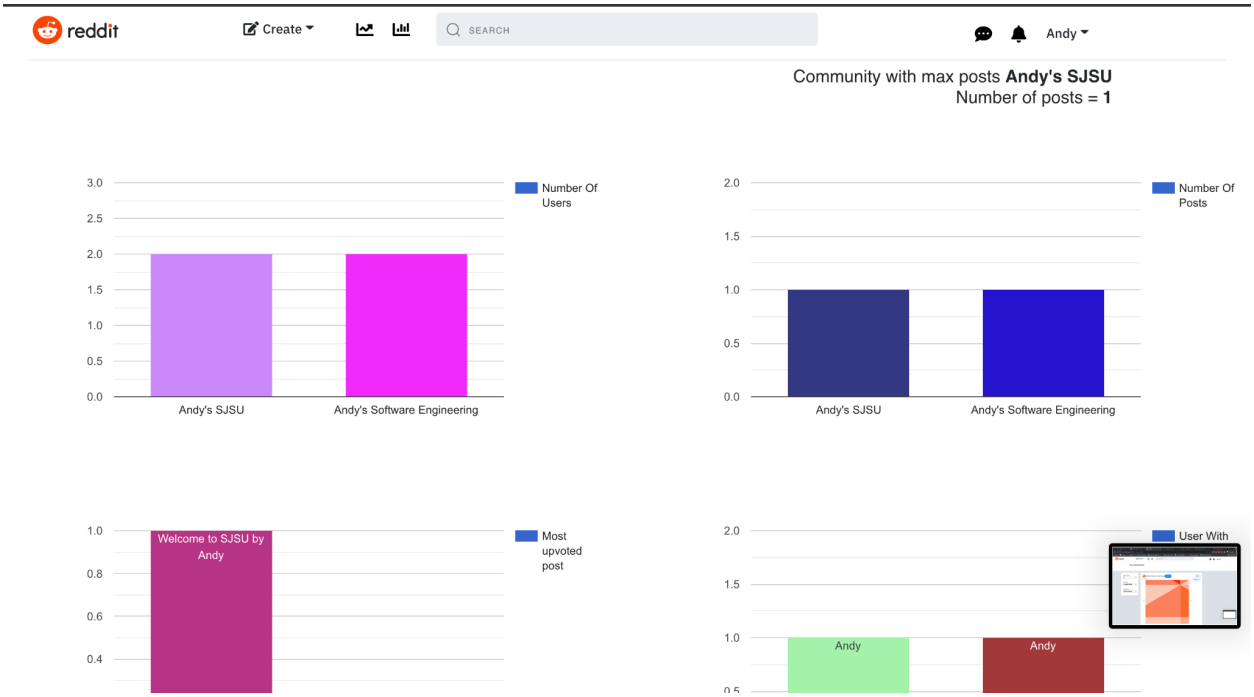
Community Search Page



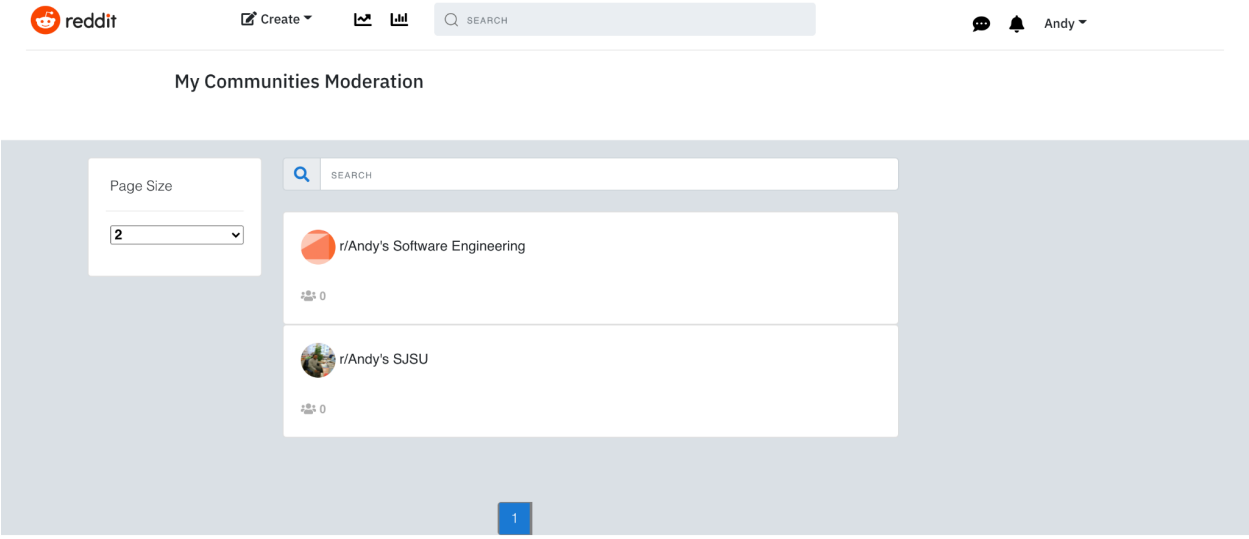
Invitations



Community Analytics



My Communities Moderation



Project reports

Projects / Reddit / Reports

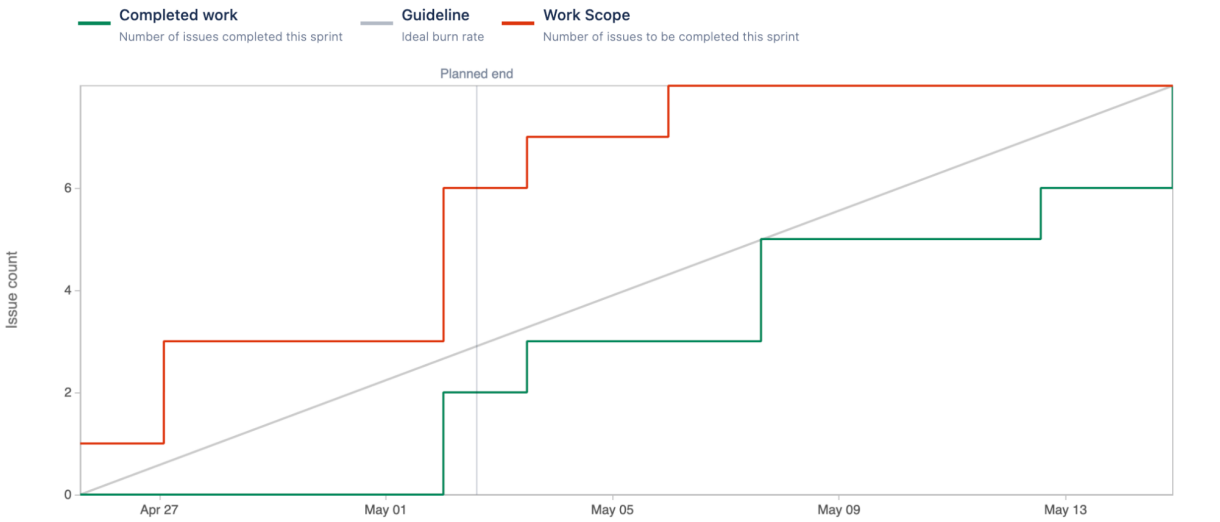
Burnup report

› How to read this report

Sprint: Sprint2

Estimation field: Issue count

Date - April 25, 2021 - May 2, 2021



Projects / Reddit / Reports

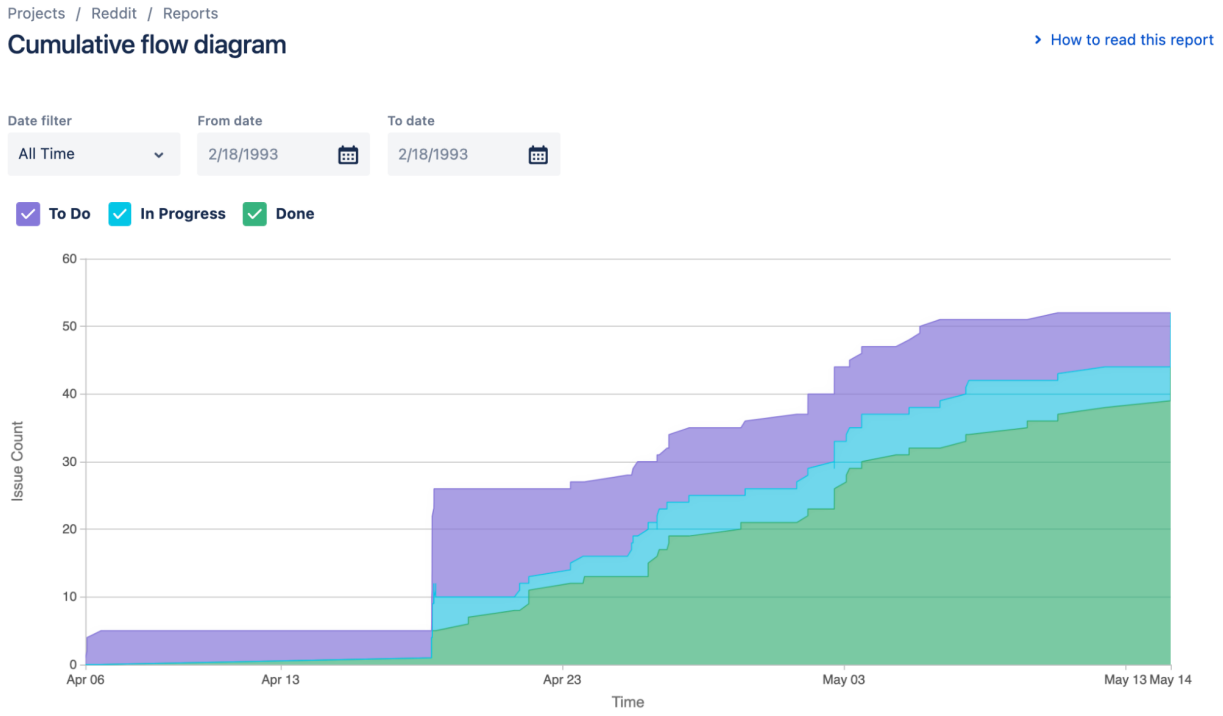
Sprint burndown chart

› How to read this report

Sprint: Sprint2

Estimation field: Issue count

Date - April 25, 2021 - May 2, 2021



Server Implementation For Entity Objects

User Model

```
const usersSchema = new Schema(
 {
   name: {
     type: String,
     required: true,
   },
   email: {
     type: String,
     required: true,
     unique: true,
   },
   password: {
     type: String,
     required: true,
   },
```

```
  handle: {
    type: String,
  },
  gender: {
    type: String,
  },
  description: {
    type: String,
  },
  avatar: {
    type: String,
  },
  location: {
    type: String,
  },
  topics: {
    type: Array,
  },
  createdCommunities: [
    {
      ref: "community",
      type: Schema.Types.ObjectId,
    },
  ],
  memberships: [
    {
      ref: "community",
      type: Schema.Types.ObjectId,
    },
  ],
},
{
  timestamps: true,
  versionKey: false,
}
);
```

Community Model

```
const communitySchema = new Schema(
 {
   communityName: { type: String, required: true, unique: true },
   description: { type: String, required: true },
   communityAvatar: [{ type: String, required: true }],
   communityCover: { type: String, required: true },
   rules: { type: Array },
   creator: { type: Schema.Types.ObjectId, ref: "user" },
   members: [
     {
       _id: { type: Schema.Types.ObjectId, ref: "user" },
       communityJoinStatus: {
         type: String,
         enum: [
           config.REQUESTED_TO_JOIN_COMMUNITY,
           config.INVITED_TO_JOIN_COMMUNITY,
           config.REJECTED_REQUEST_TO_JOIN_COMMUNITY,
           config.ACCEPTED_REQUEST_TO_JOIN_COMMUNITY,
         ],
       },
     },
   ],
   posts: [{ type: Schema.Types.ObjectId, ref: "post" }],

 },
 {
   timestamps: true,
   versionKey: false,
 }
);
communitySchema.plugin(aggregatePaginate);
communitySchema.pre('remove', function (next) {
 Post.remove({ community: this._id }).exec();
 next();
});
```

Community Vote Model

```
const communityVoteSchema = new Schema({
 communityId: {
   type: Schema.Types.ObjectId,
   ref: "community",
   required: true,
 },
 vote: {
   type: Number,
   min: -1,
   max: 1,
 },
 createdBy: {
   type: Schema.Types.ObjectId,
   ref: "user",
   required: true,
 },
},
 {
   timestamps: false,
   versionKey: false,
 }
);
```

Post Model

```
const commentSchema = new Schema(
 {
   parent_id: {
     type: mongoose.ObjectId,
   },
   depth: {
     type: Number,
     default: 0,
   },
   description: {
     type: String,
     required: true,
```

```
    trim: true,
  },
  votes: {
    type: Number,
    default: 0,
  },
  createdBy: {
    type: Schema.Types.ObjectId,
    ref: "user",
    required: true,
  },
 },
 {
  timestamps: true,
  versionKey: false,
 }
);

// Posts Schema
const postSchema = new Schema(
 {
  type: {
    type: String,
    enum: ["text", "image", "link"],
    trim: true,
    required: true,
  },
  votes: {
    type: Number,
    default: 0,
  },
  numberOfComments: {
    type: Number,
    default: 0,
  },
  images: [
    {
      type: String,
    },
  ],
```

```
    description: {
      type: String,
    },
    title: {
      type: String,
      required: true,
      trim: true,
    },
    link: {
      type: String,
    },
    comments: [commentSchema],
    createdBy: {
      type: Schema.Types.ObjectId,
      ref: "user",
      required: true,
    },
    community: {
      type: Schema.Types.ObjectId,
      ref: "community",
      required: true,
    },
  },
  {
    timestamps: true,
    versionKey: false,
  }
);
postSchema.plugin(aggregatePaginate);
```

Post Votes Model

```
const postsVotesSchema = new Schema(
  {
    post_id: {
      type: Schema.Types.ObjectId,
      ref: "post",
      required: true,
    },
```

```
  vote: {
    type: Number,
    min: -1,
    max: 1,
  },
  createdBy: {
    type: Schema.Types.ObjectId,
    ref: "user",
    required: true,
  },
},
{
  timestamps: false,
  versionKey: false,
}
);
```

Comments Votes Model

```
const commentsVotesSchema = new Schema(
{
  post_id: {
    type: Schema.Types.ObjectId,
    ref: "post",
    required: true,
  },
  comment_id: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  vote: {
    type: Number,
    min: -1,
    max: 1,
  },
  createdBy: {
    type: Schema.Types.ObjectId,
    ref: "user",
    required: true,
```

```
    },
  },
  {
    timestamps: false,
    versionKey: false,
  }
);
```

Invitation Model

```
const invitationSchema = new Schema(
  {
    user: {
      type: mongoose.SchemaTypes.ObjectId,
      ref: "user",
      required: true,
    },
    community: {
      type: mongoose.SchemaTypes.ObjectId,
      ref: "community",
      required: true,
    },
    status: {
      type: String,
      enum: [
        config.USER_ACCEPTED_INVITE,
        config.USER_PENDING_INVITE,
        config.USER_REJECTED_INVITE,
      ],
      trim: true,
      required: true,
      default: "PENDING_INVITE",
    },
  },
  {
    timestamps: true,
    versionKey: false,
  }
);
```

```
invitationSchema.plugin(mongoosePaginate);
```

Chat Model

```
const messageSchema = new Schema({
   from: {
       type: mongoose.SchemaTypes.ObjectId,
       ref: 'user'
   },
   content: { type: String }
},
   {
       timestamps: true,
       versionKey: false,
   });
const chatSchema = new Schema({
   members: [
       {
           type: mongoose.SchemaTypes.ObjectId,
           ref: 'user'
       }
   ],
   messages: [messageSchema]
},
   {
       timestamps: true,
       versionKey: false,
   });
```

Server Implementation Of The Security And Session Objects

```
function auth() {
 var opts = {
   jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
   secretOrKey: process.env.JWT_SECRET,
```

```
};

passport.use(
  new JwtStrategy(opts, (decodedPayload, callback) => {
    const user_id = decodedPayload._id;
    client.get(user_id, function (err, user) {

      if (user) {
        callback(null, JSON.parse(user));
        return;
      }
      Users.findById(user_id, "name email", (error, user) => {
        console.log(user_id);
        if (error) {
          return callback(error, false);
        } else if (user) {
          console.log("Key not found ",user);
          client.set(user_id, JSON.stringify(user));
          callback(null, user);;
        } else {
          callback(null, false);
        }
      });
    });
  })
);
}

// Appending the user data in all incoming requests to backend
const checkAuth = (req, res, next) => {
 passport.authenticate("jwt", { session: false }, (error, user, info) => {
  if (error || !user) {
    const error = {
      errorMessage: "Please login to continue",
    };
    return res.status(401).json(error);
  } else {
    req.user = user;
  }
  return next();
```

```
 })(req, res, next);
};
```

Main Server Code

```
var connection = new require("./kafka/connection");

//topics files
const userService = require("./services/user");
const authService = require("./services/auth");
const postService = require("./services/post");
const communityService = require("./services/community");
const postVoteService = require("./services/postVote");
const commentVoteService = require("./services/commentVote");
const chatService = require("./services/chat");
const invitationService = require("./services/invitation");

function handleTopicRequest(topic_name, fname) {
 var consumer = connection.getConsumer(topic_name);
 var producer = connection.getProducer();
 //console.log(producer);
 console.log("server is running ");
 consumer.on("message", function (message) {
   console.log("message received for " + topic_name + " ", fname);
   console.log(JSON.stringify(message.value));
   var data = JSON.parse(message.value);
   fname.handle_request(data.data, (err, res) => {
     console.log("in callback, producer:");
     console.log("err", err);
     console.log("res ", res);
     //console.log(producer);
     //response(data, res, err, producer);
     var payloads = [
       {
         topic: data.replyTo,
         messages: JSON.stringify({
           correlationId: data.correlationId,
           data: { res, err },
```

```
        }),
        partition: 0,
      },
    ];
    producer.send(payloads, function (err, data) {
      //console.log('producer send', data);
      if (err) {
        console.log("Error when producer sending data", err);
      } else {
        console.log("responseX ", data);
        console.log(data);
      }
    });
    return;
  });
});
}

function response(data, res, producer) {
 console.log("after handle", res);
 var payloads = [
   {
     topic: data.replyTo,
     messages: JSON.stringify({
       correlationId: data.correlationId,
       data: res,
     }),
     partition: 0,
   },
 ];
 producer.send(payloads, function (err, data) {
   //console.log('producer send', data);
   if (err) {
     console.log("Error when producer sending data", err);
   } else {
     console.log(data);
   }
 });
 return;
}
```

```
// Add your TOPICs here
//first argument is topic name
//second argument is a function that will handle this topic request
handleTopicRequest("reddit-user-topic", userService);
handleTopicRequest("reddit-auth-topic", authService);
handleTopicRequest("reddit-post-topic", postService);
handleTopicRequest("reddit-community-topic", communityService);
handleTopicRequest("reddit-post-vote-topic", postVoteService);
handleTopicRequest("reddit-comment-vote-topic", commentVoteService);
handleTopicRequest("reddit-chat-topic", chatService);
handleTopicRequest("reddit-invitation-topic", invitationService);
```

Database Access Or Connection

Mongo DB

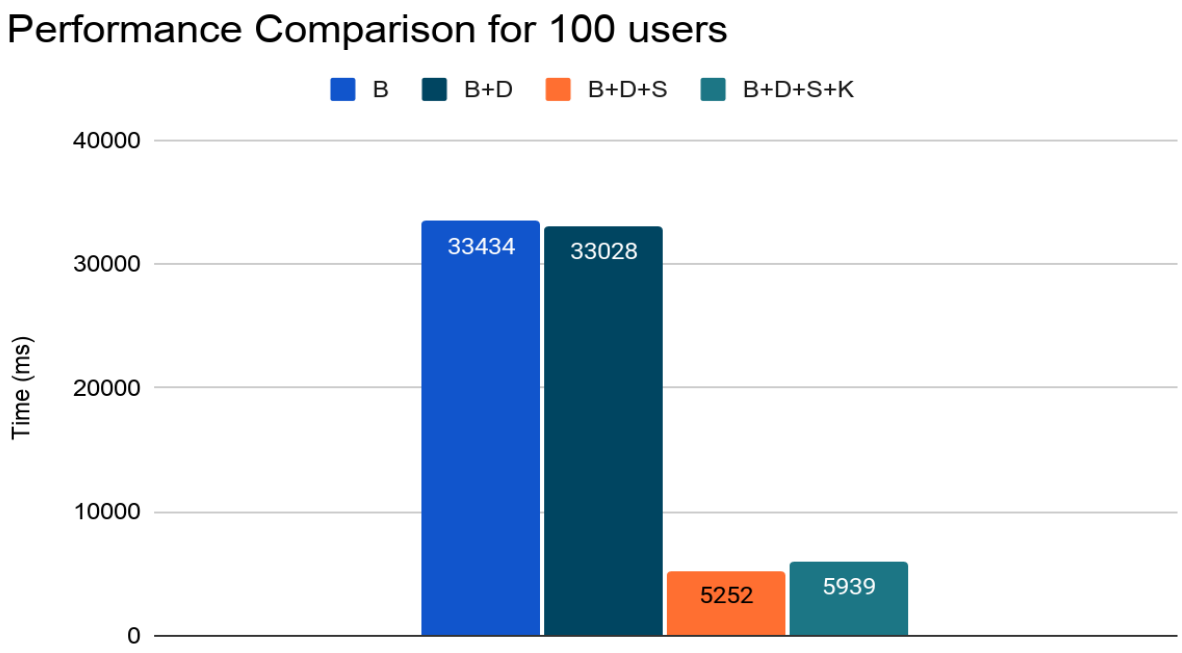
```
const { mongoDBURI } = require("./config/config");
//connect to mongoDB

const connectMongoDB = async () => {
 const options = {
   poolSize: 900,
   useNewUrlParser: true,
   useCreateIndex: true,
   useUnifiedTopology: true,
   useFindAndModify: false,
 };

 try {
   await mongoose.connect(mongoDBURI, options);
   console.log("MongoDB connected 3");
 } catch (err) {
   console.log("Could not connect to MongoDB", err);
 }
};
connectMongoDB();
```

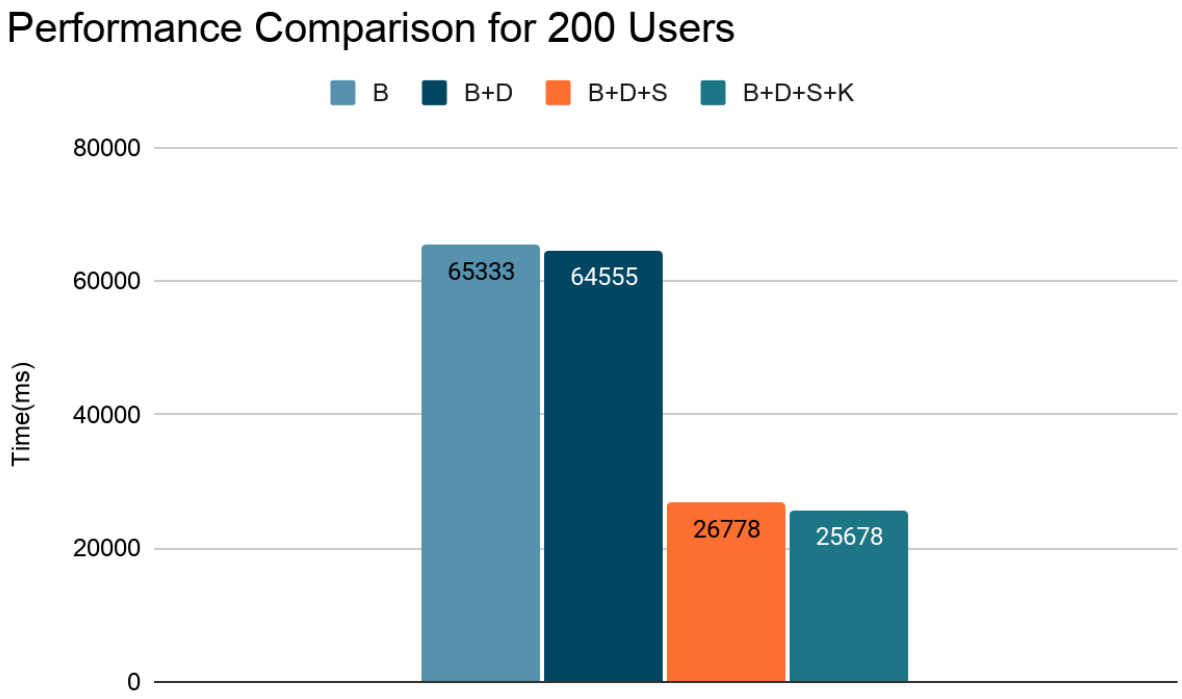
```
const Sequelize = require('sequelize');
const dbConfig = require('./mysqldb.config');

const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER,
dbConfig.PASSWORD, {
 host: dbConfig.HOST,
 dialect: dbConfig.dialect,
 pool: {
   max: 5,
   min: 0,
   idle: 10000,
 },
});

module.exports = sequelize;
```

Mocha Testing

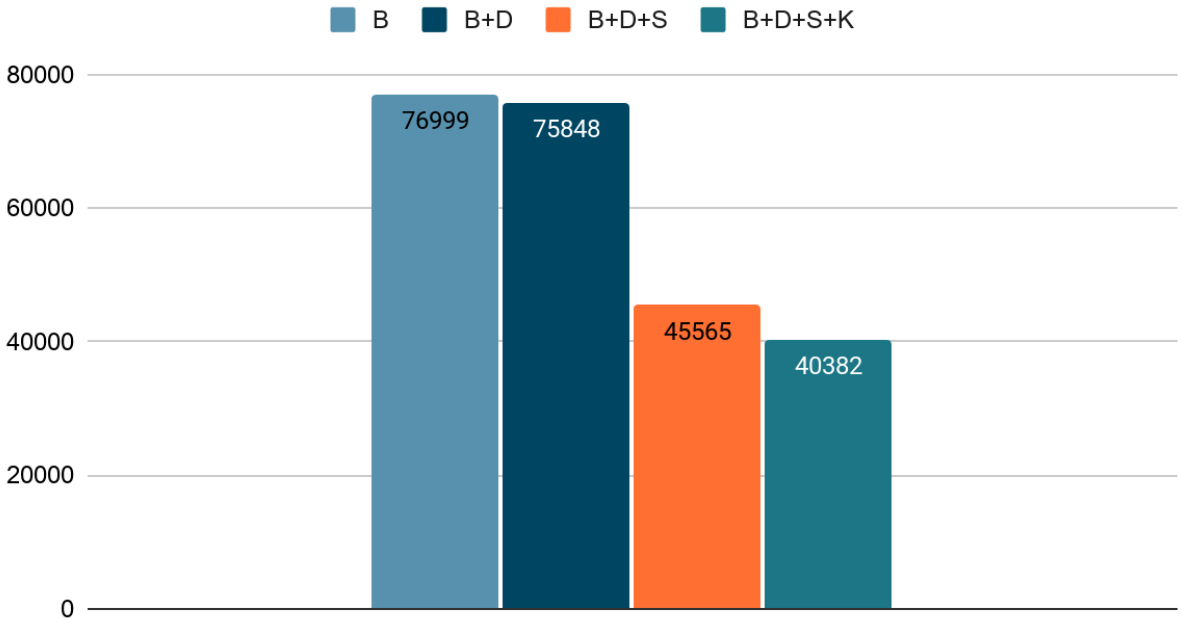
```
Community Search API testing
  Test GET route /community/search
    ✓ should return all tasks (257ms)

/POST login correct
  ✓ should return correct login user response (271ms)

/POST signup correct
  ✓ should return correct login user response (102ms)

/Get get all communities created by user
  ✓ should return correct community response (124ms)

/GET get communities detail request
  ✓ should return correct community details response (457ms)

/POST upvote community correct
  ✓ should return correct community upvote response (448ms)

/DELETE cascade delete community correct
  ✓ should delete the community

/POST community downvote correct
  ✓ should return correct login user response (627ms)

/POST community join
  ✓ should return correct community join response (95ms)

/GET chat get request
  ✓ should return correct chat get response


10 passing (2s)
```

Performance Of The Application

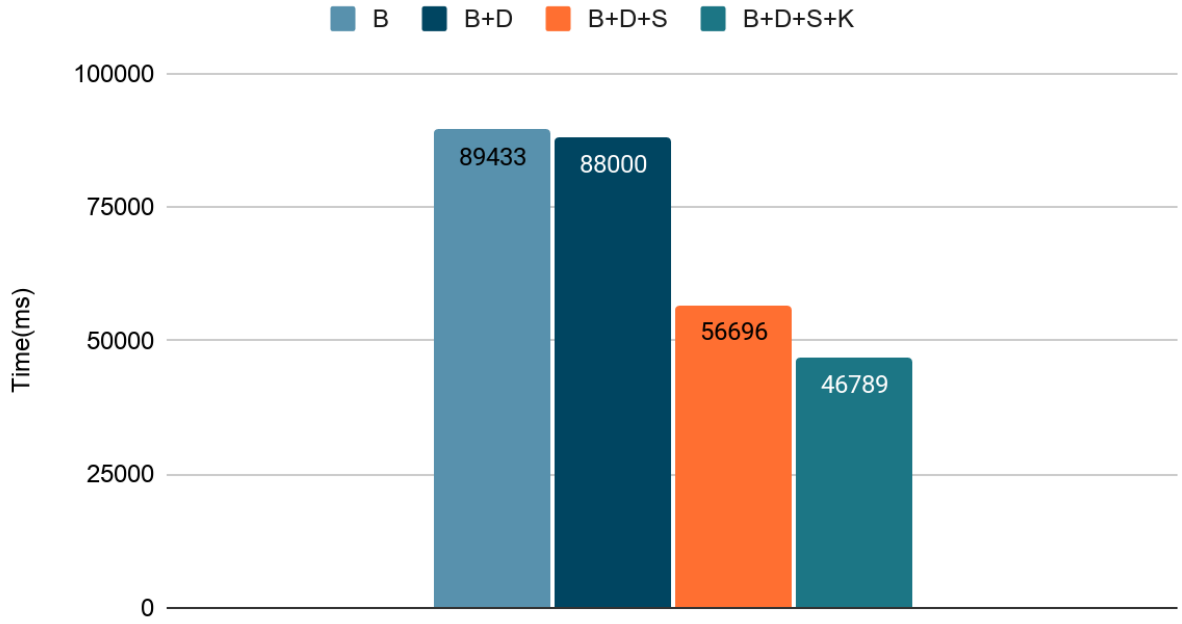




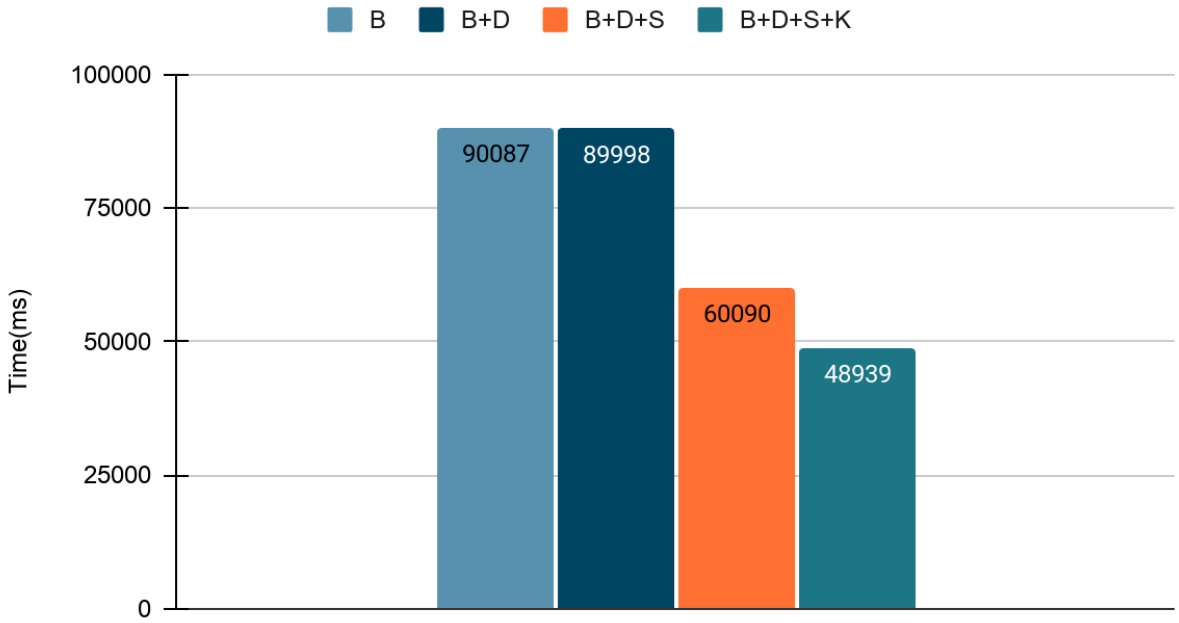
Performance Comparison for 300 Users



Performance Comparison for 400 Users



Performance Comparison for 500 users



Performance Comparison of Services With Load balancers