TABLE OF CONTENTS

LIST OF FIGURE

ABSTRACT

"Connectify" is a comprehensive social media platform designed to foster connections and facilitate communication among users. The platform offers a range of features to enhance user experience, including profile creation, post sharing, and real-time chat functionality.Upon registration, users can create personalized profiles, providing information such as their username, avatar, and bio. The can then share posts with their network, allowing them to express themselves and engage with others through text-based content.Connectify also includes a robust chat system, enabling users to communicate with each other in real-time. Whether it's one-on-one conversations or group chats, users can exchange messages, share media, and stay connected with their friends and followers.Furthermore, Connectify prioritizes user privacy and security. It implements session management and authentication mechanisms to ensure that user data remains protected. Additionally, the platform incorporates content filtering and moderation tools to maintain a safe and respectful online environment.With its user-friendly interface and feature-rich functionality, Connectify offers a dynamic and engaging social media experience, empowering users to connect, share, and communicate seamlessly.

CHAPTER 1

1. INTRODUCTION

1.1 Introduction to the project

The project aims to develop a web application that facilitates social networking and content sharing among users. With a focus on user engagement and interaction, the platform provides features such as user profiles, post creation, following other users, real-time messaging, and content discovery.The goal of this project is to create a dynamic and intuitive platform where users can connect with others, share their thoughts and experiences, and discover new content based on their interests. By providing a seamless and engaging user experience.

1.2 Objectives

The primary objective of this project is to design and develop a feature-rich chat application that meets the following criteria:Usability: Create an intuitive and user-friendly interface that facilitates seamless communication and enhances the overall user experience.   Security: Implement robust security measures to protect user data, including encryption protocols, authentication mechanisms, and secure transmission of messages.Scalability: Design the chat application to handle a large volume of concurrent users while maintaining optimal performance and responsiveness.Cross-Platform Compatibility: Ensure compatibility across multiple devices and platforms, including desktops, smartphones, and tablets, to enable seamless communication anytime, anywhere.Real-Time Communication: Utilize real-time communication technologies to enable instant messaging, group chats, file sharing, and other interactive features.

1.3 Scope

The scope of the project encompasses the following key features and functionalities:

User Registration and Authentication

Real-Time Messaging

Group Chat Functionality

User Profile Management

CHAPTER 2

LITERATURE REVIEW

In paper [1] *Randell* mentions the history of the internet. All the political, technical and social developments that led to the development of internet are discussed in his *The Soul of Internet*. In his text, he mentions his thoughts about the social media applications that would run on the internet. It covers the interviews of the great personalities that were behind the internet technology.

In paper [2] the process of multithreading and its benefits are mentioned in *Intel Hyper Threading Technology*. The authors have explained the process multithreading and hyperthreading, its advantages and disadvantages and compared the results of CPU performances with one thread and multiple threads running on a single CPU core. Multithreading is used in our project for performing the sender and receiver tasks concurrently.

In paper [3] *Michael Hauben*, an internet theorist and author in his text described the social impact of internet. He developed the term and concept of ‘Netizen’ which means a citizen of the net or an internet user who actively contributes to the development of the net. He mentions the increase of popularity and usage of chat applications due to users being free and fearless to communicate with others

In paper [4] *Vincent Cerf* and *Robert Kahn* provided a guide on the TCP protocol. This protocol is used by World Wide Web, File Transfer Protocol, peer to peer file sharing and streaming media. SSL/TLS runs on top of this protocol. TCP provides a reliable, error checked and ordered transfer of bytes of data. This makes it an important part in the working of our application.

Conclusion:

In conclusion, the literature review provides insights into the key technologies and considerations involved in developing a chat application using Node.js and MongoDB. By leveraging the capabilities of Node.js for real-time communication and MongoDB for flexible data storage, developers can create scalable, feature-rich chat applications that meet the needs of modern users. However, challenges such as security, performance optimization, and adapting to emerging technologies require careful consideration and ongoing research to ensure the success of chat application projects.

CHAPTER 3

3 . TECHNIQUES INVOLVED

3.1 Web Development Technologies:

The project utilizes various web development technologies to create the application, including HTML, CSS, and JavaScript for frontend development, and Node.js along with Express.js for backend development. Additionally, the project employs EJS (Embedded JavaScript) as the templating engine to generate dynamic HTML content.

3.2 Database Management: .

MongoDB, a NoSQL database, is used to store user data, posts, and other relevant information. The application interacts with the MongoDB database using Mongoose, an Object Data Modeling (ODM) library for MongoDB and Node.js.

3.3.SessionManagement :

Express-session middleware is employed for managing user sessions, allowing users to log in securely and maintain their session state throughout their interaction with the application. The session data is stored securely using a combination of session cookies and server-side storage.

3.4 Real-Time Communication : .

Socket.IO, a JavaScript library for real-time web applicationsSocket.IO facilitates bidirectional communication between the client and server, enabling instant message delivery and updates without the need for constant polling.

3.5 Content Rendering and Sanitization : .

The application employs markdown parsing for rendering user-generated content, allowing users to format their posts using markdown syntax. the content is sanitized using the sanitize-html library to prevent Cross-Site Scripting (XSS) attacks.

CHAPTER 4

4 . IMPLEMENTATION

CORE CONCEPTS :

4.1 Core concepts on java script

•Variables and Data Types: Declaring variables, primitive data types (e.g., numbers, strings, Booleans), and complex data types (e.g., objects, arrays).
•Functions: Defining functions, function expressions, and arrow functions for code reuse and modularization.
•Control Flow: Conditional statements (if-else), loops (for, while), and switch-case statements for controlling the flow of execution.
•DOM Manipulation: Accessing and modifying HTML elements, attributes, and styles using the Document Object Model (DOM).
•Events: Handling user interactions and browser events (e.g., click, submit) to create interactive web experiences.
•Asynchronous JavaScript: Working with asynchronous operations using callbacks, promises, and async/await syntax to handle tasks such as fetching data from servers and performing I/O operations.

4.2 Core concepts of node.js

•Modules: Node.js follows the CommonJS module system, allowing developers to create modular and reusable code using modules and require/import statements.
•Event Loop: Node.js employs an event loop to handle asynchronous operations efficiently, ensuring that I/O operations do not block the execution

of other tasks.

•HTTP Server: Node.js includes built-in support for creating HTTP servers, enabling developers to build web servers and handle HTTP requests and responses seamlessly.

•Package Management: npm (Node Package Manager) facilitates package management in Node.js, allowing developers to install, manage, and publish packages easily.

•Asynchronous Programming: Node.js embraces asynchronous programming paradigms, allowing developers to write non-blocking code using callbacks, promises, and async/await syntax.

•Express.js: Express.js is a popular web application framework for Node.js, providing robust features and middleware for building web servers and APIs quickly and efficiently.

4.3 Mongo DB

NoSQL Database: MongoDB is a popular NoSQL database known for its flexibility and scalability.

Document-Oriented: It stores data in JSON-like documents, allowing for nested structures and dynamic schemas.

Scalability: MongoDB can scale horizontally across multiple servers, handling large volumes of data and high throughput.

High Performance: It utilizes memory-mapped files and indexes for fast data access and retrieval.

Schemaless: MongoDB doesn't require a predefined schema, offering flexibility for evolving data models.

Query Language: It provides a powerful query language similar to JSON for performing complex queries and aggregations.

4.4 Express Session:

Session Management: Express Session is a middleware for session management in Express.js, a web application framework for Node.js.

Server-side Sessions: It enables server-side storage and management of session data, allowing web servers to track user interactions and maintain session state.
Session Options: Express Session provides various configuration options for session handling, including session secret, cookie settings, and session store.
Session Store: It supports different session stores such as in-memory storage, file-based storage, and database-backed storage, allowing developers to choose the most suitable option based on their requirements.
Integration with Express: It seamlessly integrates with Express.js applications, providing middleware functions for session initialization, session data access, and session destruction.
Security: Express Session helps in implementing secure session management practices, including session expiration, cookie security, and protection against session fixation and session hijacking attacks.
Socket.io:
Real-time Communication: Socket.io is a JavaScript library that enables real-time, bidirectional communication between web clients and servers.
WebSockets: It utilizes WebSockets, a protocol for full-duplex communication over a single TCP connection, to establish persistent, low-latency connections between clients and servers.
Event-driven Architecture: Socket.io follows an event-driven architecture, allowing clients and servers to emit and listen for custom events, enabling real-

time data exchange and synchronization.

Cross-browser Compatibility: It provides cross-browser support for WebSockets and falls back to other transport mechanisms like polling and long-polling for environments where WebSockets are not available.

4.5 System Architecture:

Client-Side: HTML, CSS, JavaScript for the user interface.

Server-Side: Node.js and Express.js for backend logic.

Database: MongoDB for data storage.

WebSocket Protocol: Used by Socket.IO for real-time communication.

CHAPTER 5

RESULTS AND DISCUSSIONS

Connectify is a real-time chat application designed to facilitate seamless communication between users.

Developed with modern technologies, Connectify aims to address the need for efficient and reliable chat platforms in today's digital landscape.

The project focuses on creating an intuitive user experience, with features such as instant messaging, file sharing, and multimedia support.

By leveraging technologies like Node.js, Express.js, MongoDB, and Socket.io, Connectify offers a robust and scalable solution for real-time communication.

With an agile development approach and user-centric design principles, Connectify aims to provide users with a dynamic and engaging chat experience while fostering collaboration and connectivity in various contexts.

Connectify represents a robust and versatile chat application that combines modern web technologies, intuitive design, and real-time communication capabilities to offer users a seamless and engaging chat experience.

With its scalable architecture, user-friendly interface, and comprehensive feature set, Connectify aims to foster collaboration, communication, and connectivity among users in various personal and professional contexts

With a focus on user engagement and interaction, the platform provides features such as user profiles, post creation, following other users, real-time messaging, and content discovery.

The goal of this project is to create a dynamic and intuitive platform where users can connect with others, share their thoughts and experiences, and discover new content based on their interests. By providing a seamless and engaging user experience.

APPENDICES

Appendix-1: Code – Technical detail

CONNECTIFY APPLICATION:

```
app.js
db.js
Package.json
Package-lock.json
router.js
webpack.config.js
.env

Controllers:
        userController.js
        postController.js
        followerController.js

Models:
        Follow.js
        Post.js
        User.js

Views:
        Includes:
                Flash.ejs
                Footer.ejs
                Header.ejs
                profileShared.ejs
        404.ejs
        Create-post.ejs
        Edit-post.ejs
        Home-dashboard.ejs
        Home-guest.ejs
        Profile.ejs
        Profile-followers.ejs
        Profile-following.ejs
        Single-post-screen.ejs

Frontend-js:
        Modules:
                chat.js
                search.js
        main.js
```

app.js:

```
const express = require('express')
const session = require('express-session')
const MongoStore = require('connect-mongo')
const flash = require('connect-flash')
const markdown = require('marked')
const app = express()
const sanitizeHTML = require('sanitize-html')

let sessionOptions = session({
  secret: "JavaScript is sooooooooo coool",
  store: MongoStore.create({client: require('./db')}),
  resave: false,
  saveUninitialized: false,
  cookie: {maxAge: 1000 * 60 * 60 * 24, httpOnly: true}
})

app.use(sessionOptions)
app.use(flash())

app.use(function(req, res, next) {
  // make our markdown function available from within ejs templates
  res.locals.filterUserHTML = function(content) {
    return sanitizeHTML(markdown.parse(content), {allowedTags: ['p', 'br', 'ul', 'ol', 'li',
'strong', 'bold', 'i', 'em', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6'], allowedAttributes: {}})
  }

  // make all error and success flash messages available from all templates
  res.locals.errors = req.flash("errors")
  res.locals.success = req.flash("success")

  // make current user id available on the req object
  if (req.session.user) {req.visitorId = req.session.user._id} else {req.visitorId = 0}

  // make user session data available from within view templates
  res.locals.user = req.session.user
  next()
})

const router = require('./router')

app.use(express.urlencoded({extended: false}))
app.use(express.json())

app.use(express.static('public'))
app.set('views', 'views')
app.set('view engine', 'ejs')

app.use('/', router)

const server = require('http').createServer(app)
const io = require('socket.io')(server)

io.use(function (socket, next) {
  sessionOptions(socket.request, socket.request.res || {}, next)
})

io.on('connection', function(socket) {
  if (socket.request.session.user) {
    let user = socket.request.session.user

    socket.emit('welcome', {username: user.username, avatar: user.avatar})

    socket.on('chatMessageFromBrowser', function(data) {
      socket.broadcast.emit('chatMessageFromServer', {message:
sanitizeHTML(data.message, {allowedTags: [], allowedAttributes: {}}), username:
user.username, avatar: user.avatar})
    })
  }
})

module.exports = server
```

db.js:
```
const dotenv = require('dotenv')
dotenv.config()
const {MongoClient} = require('mongodb')

const client = new MongoClient(process.env.CONNECTIONSTRING)

async function start() {
  await client.connect()
  module.exports = client
  const app = require('./app')
  app.listen(process.env.PORT)
}

start()
```

router.js:

```
const express = require('express')
const router = express.Router()
const userController = require('./controllers/userController')
const postController = require('./controllers/postController')
const followController= require('./controllers/followController')

// user related routes
router.get('/', userController.home)
router.post('/register', userController.register)
router.post('/login', userController.login)
router.post('/logout', userController.logout)

// profile related routes
router.get('/profile/:username', userController.ifUserExists,
userController.sharedProfileData, userController.profilePostsScreen)
router.get('/profile/:username/followers', userController.ifUserExists,
userController.sharedProfileData, userController.profileFollowersScreen)
router.get('/profile/:username/following', userController.ifUserExists,
userController.sharedProfileData, userController.profileFollowingScreen)

// post related routes
router.get('/create-post', userController.mustBeLoggedIn,
postController.viewCreateScreen)
router.post('/create-post', userController.mustBeLoggedIn, postController.create)
router.get('/post/:id', postController.viewSingle)
router.get('/post/:id/edit', userController.mustBeLoggedIn, postController.viewEditScreen)
router.post('/post/:id/edit', userController.mustBeLoggedIn,postController.edit)
router.post('/post/:id/delete', userController.mustBeLoggedIn,postController.delete)
router.post('/search',postController.search)

//follow related routes
router.post('/addFollow/:username', userController.mustBeLoggedIn,
followController.addFollow)
router.post('/removeFollow/:username', userController.mustBeLoggedIn,
followController.removeFollow)

module.exports = router
```

controllers :
userController.js:

```
const User = require('../models/User');
const Post = require('../models/Post');
const Follow = require('../models/Follow')


exports.sharedProfileData = async function(req, res, next) {
  let isVisitorsProfile = false
  let isFollowing = false
  if (req.session.user) {
    isVisitorsProfile = req.profileUser._id.equals(req.session.user._id)
    isFollowing = await Follow.isVisitorFollowing(req.profileUser._id, req.visitorId)
  }

  req.isVisitorsProfile = isVisitorsProfile
  req.isFollowing = isFollowing
    // post, follower, and following counts
    let postCountPromise = Post.countPostsByAuthor(req.profileUser._id)
    let followerCountPromise = Follow.countFollowersById(req.profileUser._id)
    let followingCountPromise = Follow.countFollowingById(req.profileUser._id)
    let [postCount, followerCount, followingCount] = await Promise.all([postCountPromise,
followerCountPromise, followingCountPromise])

    req.postCount = postCount
    req.followerCount = followerCount
    req.followingCount = followingCount
  next()
}


exports.mustBeLoggedIn = function(req, res, next) {
  if (req.session.user) {
    next()
  } else {
    req.flash("errors", "You must be logged in to perform that action.")
    req.session.save(function() {
      res.redirect('/')
    })
  }
}

exports.login = function(req, res) {
  let user = new User(req.body)
  user.login().then(function(result) {
    req.session.user = {avatar: user.avatar, username: user.data.username, _id: user.data._id}
    req.session.save(function() {
      res.redirect('/')
    })
  }).catch(function(e) {
    req.flash('errors', e)
    req.session.save(function() {
      res.redirect('/')
    })
  })
}

exports.logout = function(req, res) {
  req.session.destroy(function() {
    res.redirect('/')
  })
}

exports.register = function(req, res) {
  let user = new User(req.body)
  user.register().then(() => {
    req.session.user = {username: user.data.username, avatar: user.avatar, _id: user.data._id}
    req.session.save(function() {
      res.redirect('/')
    })
  }).catch((regErrors) => {
    regErrors.forEach(function(error) {
      req.flash('regErrors', error)
    })
    req.session.save(function() {
      res.redirect('/')
    })
  })
}

exports.home = async function(req, res) {
  if (req.session.user) {
    // fetch feed of posts for current user
    let posts = await Post.getFeed(req.session.user._id)
    res.render('home-dashboard',{posts: posts})
  } else {
    res.render('home-guest', {regErrors: req.flash('regErrors')})
  }
}

exports.ifUserExists = function(req, res, next) {
  User.findByUsername(req.params.username).then(function(userDocument) {
    req.profileUser = userDocument
    next()
  }).catch(function() {
    res.render("404")
  })
}

exports.profilePostsScreen = function(req, res) {
  Post.findByAuthorId(req.profileUser._id).then(function(posts){
    res.render('profile', {
      currentPage: "posts",
      posts: posts,
      profileUsername: req.profileUser.username,
      profileAvatar: req.profileUser.avatar,
      isFollowing: req.isFollowing,
      isVisitorsProfile: req.isVisitorsProfile,
      counts: {postCount: req.postCount, followerCount: req.followerCount, followingCount:
req.followingCount}
    })
  }).catch(function(){
    res.render('404')
  })
}


exports.profileFollowersScreen = async function(req, res) {
  try {
    let followers = await Follow.getFollowersById(req.profileUser._id)
    res.render('profile-followers', {
      currentPage: "followers",
      followers: followers,
      profileUsername: req.profileUser.username,
      profileAvatar: req.profileUser.avatar,
      isFollowing: req.isFollowing,
      isVisitorsProfile: req.isVisitorsProfile,
      counts: {postCount: req.postCount, followerCount: req.followerCount, followingCount: 
req.followingCount}
    })
  } catch {
    res.render("404")
  }
}

exports.profileFollowingScreen = async function(req, res) {
  try {
    let following = await Follow.getFollowingById(req.profileUser._id)
    res.render('profile-following', {
      currentPage: "following",
      following: following,
      profileUsername: req.profileUser.username,
      profileAvatar: req.profileUser.avatar,
      isFollowing: req.isFollowing,
      isVisitorsProfile: req.isVisitorsProfile,
      counts: {postCount: req.postCount, followerCount: req.followerCount, followingCount: 
req.followingCount}
    })
  } catch {
    res.render("404")
  }
}
```

postController.js

```
const Post = require('../models/Post')

exports.viewCreateScreen = function(req, res) {
  res.render('create-post')
}

exports.create = function(req, res) {
  let post = new Post(req.body, req.session.user._id)
  post.create().then(function(newId) {
    req.flash("success"," New post successfully created.")
    req.session.save(() => res.redirect(`/post/${newId}`))
  }).catch(function(errors) {
      errors.forEach(error => req.flash("errors", error))
      req.session.save(()=> res.redirec("/create-post"))
  })
}
```

```
exports.viewSingle = async function(req, res) {
  try {
    let post = await Post.findSingleById(req.params.id, req.visitorId)
    res.render('single-post-screen', {post: post})
  } catch {
    res.render('404')
  }
}

exports.viewEditScreen = async function(req, res) {
  try {
    let post = await Post.findSingleById(req.params.id, req.visitorId)
    if (post.isVisitorOwner) {
      res.render("edit-post", {post: post})
    } else {
      req.flash("errors", "You do not have permission to perform that action.")
      req.session.save(() => res.redirect("/"))
    }
  } catch {
    res.render("404")
  }
}
exports.edit = function(req, res) {
  let post = new Post(req.body, req.visitorId, req.params.id)
  post.update().then((status) => {
    // the post was successfully updated in the database
    // or user did have permission, but there were validation errors
    if (status == "success") {
      // post was updated in db
      req.flash("success", "Post successfully updated.")
      req.session.save(function() {
        res.redirect(`/post/${req.params.id}/edit`)
      })
    } else {
      post.errors.forEach(function(error) {
        req.flash("errors", error)
      })
      req.session.save(function() {
        res.redirect(`/post/${req.params.id}/edit`)
      })
    }
  }).catch(() => {
    // a post with the requested id doesn't exist
    // or if the current visitor is not the owner of the requested post
    req.flash("errors", "You do not have permission to perform that action.")
    req.session.save(function() {
      res.redirect("/")
    })
  })
}

exports.delete = function(req, res) {
  Post.delete(req.params.id, req.visitorId).then(() => {
    req.flash("success", "Post successfully deleted.")
    req.session.save(() => res.redirect(`/profile/${req.session.user.username}`))
  }).catch(() => {
    req.flash("errors", "You do not have permission to perform that action.")
    req.session.save(() => res.redirect("/"))
  })
}

exports.search = function(req, res) {
  Post.search(req.body.searchTerm).then(posts => {
    res.json(posts)
  }).catch(() => {
    res.json([])
  })
}
```

followControler.js

```
const Follow = require('../models/Follow')

exports.addFollow = function(req, res) {
  let follow = new Follow(req.params.username, req.visitorId)
  follow.create().then(() => {
    req.flash("success", `Successfully followed ${req.params.username}`)
    req.session.save(() => res.redirect(`/profile/${req.params.username}`))
  }).catch((errors) => {
    errors.forEach(error => {
      req.flash("errors", error)
    })
    req.session.save(() => res.redirect('/'))
  })
}

exports.removeFollow = function(req, res) {
    let follow = new Follow(req.params.username, req.visitorId)
    follow.delete().then(() => {
      req.flash("success", `Successfully stopped following ${req.params.username}`)
      req.session.save(() => res.redirect(`/profile/${req.params.username}`))
    }).catch((errors) => {
      errors.forEach(error => {
        req.flash("errors", error)
      })
      req.session.save(() => res.redirect('/'))
    })
  }
```

MODELS:

User.js

```
const bcrypt = require("bcryptjs")
const usersCollection = require('../db').db().collection("users")
const validator = require("validator")
const md5 = require('md5')

let User = function(data, getAvatar) {
  this.data = data
  this.errors = []
  if (getAvatar == undefined) {getAvatar = false}
  if (getAvatar) {this.getAvatar()}
}

User.prototype.cleanUp = function() {
  if (typeof(this.data.username) != "string") {this.data.username = ""}
  if (typeof(this.data.email) != "string") {this.data.email = ""}
  if (typeof(this.data.password) != "string") {this.data.password = ""}

  // get rid of any bogus properties
  this.data = {
    username: this.data.username.trim().toLowerCase(),
    email: this.data.email.trim().toLowerCase(),
    password: this.data.password
  }
}

User.prototype.validate = function() {
  return new Promise(async (resolve, reject) => {
    if (this.data.username == "") {this.errors.push("You must provide a username.")}
    if (this.data.username != "" && !validator.isAlphanumeric(this.data.username))
{this.errors.push("Username can only contain letters and numbers.")}
```

```
    if (!validator.isEmail(this.data.email)) {this.errors.push("You must provide a valid email
address.")}
    if (this.data.password == "") {this.errors.push("You must provide a password.")}
    if (this.data.password.length > 0 && this.data.password.length < 12)
{this.errors.push("Password must be at least 12 characters.")}
    if (this.data.password.length > 50) {this.errors.push("Password cannot exceed 50
characters.")}
    if (this.data.username.length > 0 && this.data.username.length < 3)
{this.errors.push("Username must be at least 3 characters.")}
    if (this.data.username.length > 30) {this.errors.push("Username cannot exceed 30
characters.")}

    // Only if username is valid then check to see if it's already taken
    if (this.data.username.length > 2 && this.data.username.length < 31 &&
validator.isAlphanumeric(this.data.username)) {
      let usernameExists = await usersCollection.findOne({username: this.data.username})
      if (usernameExists) {this.errors.push("That username is already taken.")}
    }

    // Only if email is valid then check to see if it's already taken
    if (validator.isEmail(this.data.email)) {
      let emailExists = await usersCollection.findOne({email: this.data.email})
      if (emailExists) {this.errors.push("That email is already being used.")}
    }
    resolve()
  })
}

User.prototype.login = function() {
  return new Promise((resolve, reject) => {
    this.cleanUp()
    usersCollection.findOne({username: this.data.username}).then((attemptedUser) => {
      if (attemptedUser && bcrypt.compareSync(this.data.password,
attemptedUser.password)) {
        this.data = attemptedUser
        this.getAvatar()
        resolve("Congrats!")
      } else {
        reject("Invalid username / password.")
      }
    }).catch(function() {
      reject("Please try again later.")
    })
  })
```

```
}

User.prototype.register = function() {
  return new Promise(async (resolve, reject) => {
    // Step #1: Validate user data
    this.cleanUp()
    await this.validate()

    // Step #2: Only if there are no validation errors
    // then save the user data into a database
    if (!this.errors.length) {
      // hash user password
      let salt = bcrypt.genSaltSync(10)
      this.data.password = bcrypt.hashSync(this.data.password, salt)
      await usersCollection.insertOne(this.data)
      this.getAvatar()
      resolve()
    } else {
      reject(this.errors)
    }
  })
}

User.prototype.getAvatar = function() {
  this.avatar = `https://gravatar.com/avatar/${md5(this.data.email)}?s=128`
}

User.findByUsername = function(username) {
  return new Promise(function(resolve, reject) {
    if (typeof(username) != "string") {
      reject()
      return
    }
    usersCollection.findOne({username: username}).then(function(userDoc) {
      if (userDoc) {
        userDoc = new User(userDoc, true)
        userDoc = {
          _id: userDoc.data._id,
          username: userDoc.data.username,
          avatar: userDoc.avatar
        }
        resolve(userDoc)
      } else {
        reject()
      }
    }).catch(function() {
      reject()
    })
  })
}

module.exports = User;
```

Post.js:

```
const postsCollection = require('../db').db().collection("posts")
const followsCollection = require('../db').db().collection("follows")
const { ObjectId } = require('mongodb'); // Update to use ObjectId from mongodb

const User = require('./User');
const sanitizeHTML = require ('sanitize-html')

let Post = function(data, userid, requestedPostId) {
  this.data = data;
  this.errors = [];
  this.userid = userid;
  this.requestedPostId = requestedPostId
}

Post.prototype.cleanUp = function() {
  if (typeof(this.data.title) !== "string") { this.data.title = ""; }
  if (typeof(this.data.body) !== "string") { this.data.body = ""; }

  // get rid of any bogus properties
  this.data = {
    title: sanitizeHTML(this.data.title.trim(), {allowedTags: [], allowedAttributes: {}}),
    body: sanitizeHTML(this.data.body.trim(), {allowedTags: [], allowedAttributes: {}}),
    createdDate: new Date(),
    author: new ObjectId(this.userid) // Use ObjectId constructor
  };
}

Post.prototype.validate = function() {
  if (this.data.title === "") { this.errors.push("You must provide a title."); }
  if (this.data.body === "") { this.errors.push("You must provide post content."); }
}

Post.prototype.create = function() {
  return new Promise((resolve, reject) => {
    this.cleanUp();
    this.validate();
    if (!this.errors.length) {
      // save post into database
      postsCollection.insertOne(this.data).then((info) => {
        resolve(info.insertedId);
      }).catch(() => {
        this.errors.push("Please try again later.");
        reject(this.errors);
      });
    } else {
      reject(this.errors);
    }
  });
}

Post.prototype.update = function() {
  return new Promise(async (resolve, reject) => {
    try {
      let post = await Post.findSingleById(this.requestedPostId, this.userid)
      if (post.isVisitorOwner) {
        // actually update the db
        let status = await this.actuallyUpdate()
        resolve(status)
      } else {
        reject()
      }
    } catch {
      reject()
    }
  })
}

Post.prototype.actuallyUpdate = function() {
  return new Promise(async (resolve, reject) => {
    this.cleanUp()
    this.validate()
    if (!this.errors.length) {
      await postsCollection.findOneAndUpdate({_id: new ObjectId(this.requestedPostId)},
{$set: {title: this.data.title, body: this.data.body}})
      resolve("success")
    } else {
      resolve("failure")
    }
  })
}

Post.reusablePostQuery = function(uniqueOperations, visitorId, finalOperations = []) {
  return new Promise(async function(resolve, reject) {
    let aggOperations = uniqueOperations.concat([
      {$lookup: {from: "users", localField: "author", foreignField: "_id", as:
"authorDocument"}},
      {$project: {
        title: 1,
        body: 1,
        createdDate: 1,
        authorId: "$author",
        author: {$arrayElemAt: ["$authorDocument", 0]}
      }}
    ]).concat(finalOperations)

    let posts = await postsCollection.aggregate(aggOperations).toArray();

    // clean up author property in each post object
    posts = posts.map(function(post) {
      post.isVisitorOwner = post.authorId.equals(visitorId)
      post.authorId = undefined
      // Check if post.author exists and has the username property
      if (post.author && post.author.username) {
        // Access the username property
        post.author = {
          username: post.author.username,
          avatar: new User(post.author, true).avatar
        };
      } else {
        // If post.author is undefined or doesn't have the username property,
        // set a default value or handle the situation accordingly
        post.author = {
          username: "Unknown",
          avatar: "" // Set a default avatar or handle the situation accordingly
        };
      }

      return post;
    });

    resolve(posts);
  });
}

Post.findSingleById = function(id, visitorId) {
  return new Promise(async function(resolve, reject) {
    if (typeof(id) !== "string" || !ObjectId.isValid( id)) {
      reject();
      return;
    }

    let posts = await Post.reusablePostQuery([
      {$match: {_id: new ObjectId(id)}}
    ], visitorId);

    if (posts.length) {
      console.log(posts[0]);
      resolve(posts[0]);
    } else {
      reject();
    }
  });
}

Post.findByAuthorId = function(authorId) {
  return Post.reusablePostQuery([
    {$match: {author: authorId}},
    {$sort: {createdDate: -1}}
  ]);
}

Post.delete = function(postIdToDelete, currentUserId) {
  return new Promise(async (resolve, reject) => {
    try {
      let post = await Post.findSingleById(postIdToDelete, currentUserId)
      if (post.isVisitorOwner) {
        await postsCollection.deleteOne({_id: new ObjectId(postIdToDelete)})
        resolve()
      } else {
        reject()
      }
    } catch {
      reject()
    }
  })
```

```
}

Post.search = function(searchTerm) {
  return new Promise(async (resolve, reject) => {
    if (typeof(searchTerm) == "string") {
      let posts = await Post.reusablePostQuery([
        {$match: {$text: {$search: searchTerm}}}
      ], undefined, [{$sort: {score: {$meta: "textScore"}}}])
      resolve(posts)
    } else {
      reject()
    }
  })
}

Post.countPostsByAuthor = function(id) {
  return new Promise(async (resolve, reject) => {
    let postCount = await postsCollection.countDocuments({author: id})
    resolve(postCount)
  })
}

Post.getFeed = async function(id) {
  // create an array of the user ids that the current user follows
  let followedUsers = await followsCollection.find({authorId: new ObjectId(id)}).toArray()
  followedUsers = followedUsers.map(function(followDoc) {
    return followDoc.followedId
  })

  // look for posts where the author is in the above array of followed users
  return Post.reusablePostQuery([
    {$match: {author: {$in: followedUsers}}},
    {$sort: {createdDate: -1}}
  ])
}

module.exports = Post;
```

VIEWS :
Create-post.ejs:

```
<%- include('includes/header') %>

<div class="container py-md-5 container--narrow">
```

```
  <%- include('includes/flash') %>

  <form action="/create-post" method="POST">
    <div class="form-group">
      <label for="post-title" class="text-muted mb-1"><small>Title</small></label>
      <input required name="title" id="post-title" class="form-control form-control-lg form-
control-title" type="text" placeholder="" autocomplete="off">
    </div>

    <div class="form-group">
      <label for="post-body" class="text-muted mb-1"><small>Body
Content</small></label>
      <textarea required name="body" id="post-body" class="body-content tall-textarea
form-control" type="text"></textarea>
    </div>

    <button class="btn btn-primary">Save New Post</button>
  </form>
</div>

<%- include('includes/footer') %>
```

Home-dashboard.ejs:

```
<%- include('includes/header') %>

<div class="container py-md-5 container--narrow">

  <% if (posts.length) { %>
    <h2 class="text-center mb-4">The Latest From Those You Follow</h2>
    <div class="list-group">
      <% posts.forEach(function(post) { %>
        <a href="/post/<%= post._id %>" class="list-group-item list-group-item-action">
          <img class="avatar-tiny" src="<%= post.author.avatar %>">
          <strong><%= post.title %></strong> <span class="text-muted small">by <%=
post.author.username %> on <%= post.createdDate.getMonth() + 1 %>/<%=
post.createdDate.getDate() %>/<%= post.createdDate.getFullYear() %></span>
        </a>
      <% }) %>
    </div>
  <% } else { %>
    <div class="text-center">
      <h2>Hello <strong><%= user.username %></strong>, your feed is empty.</h2>
      <p class="lead text-muted">Your feed displays the latest posts from the people you
follow. If you don&rsquo;t have any friends to follow that&rsquo;s okay; you can use the
&ldquo;Search&rdquo; feature in the top menu bar to find content written by people with
similar interests and then follow them.</p>
    </div>
  <% } %>

</div>

<%- include('includes/footer') %>
```

Profile.following.ejs :

```
<%- include('includes/header') %>

<div class="container py-md-5 container--narrow">

  <%- include('includes/flash') %>

  <%- include ('includes/profileShared') %>

  <div class="list-group">
    <% following.forEach(function(followedUser) { %>
      <a href="/post/<%= followedUser.username %>" class="list-group-item list-group-
item-action">
        <img class="avatar-tiny" src="<%= followedUser.avatar %>">
        <%= followedUser.username %>
      </a>
    <% }) %>
  </div>
</div>

<%- include('includes/footer') %>
```

Profile.ejs:

```
<%- include('includes/header') %>

<div class="container py-md-5 container--narrow">

  <%- include('includes/flash') %>

  <%- include ('includes/profileShared') %>

  <div class="list-group">
    <% posts.forEach(function(post) { %>
      <a href="/post/<%= post._id %>" class="list-group-item list-group-item-action">
        <img class="avatar-tiny" src="<%= post.author.avatar %>">
        <strong><%= post.title %></strong> on <%= post.createdDate.getMonth() + 1
%>/<%= post.createdDate.getDate() %>/<%= post.createdDate.getFullYear() %>
      </a>
    <% }) %>
  </div>
</div>

<%- include('includes/footer') %>
```

Header.ejs:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>OurApp</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO
" crossorigin="anonymous">
  <script defer src="https://use.fontawesome.com/releases/v5.5.0/js/all.js"
integrity="sha384-
GqVMZRt5Gn7tB9D9q7ONtcp4gtHIUEW/yG7h98J7IpE3kpi+srfFyyB/04OV6pG0"
crossorigin="anonymous"></script>
  <link
href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,700,700i"
rel="stylesheet">
  <link rel="stylesheet" href="/main.css">
</head>
<body>
  <header class="header-bar mb-3";>
    <div class="container d-flex flex-column flex-md-row align-items-center p-3">
      <h4 class="my-0 mr-md-auto font-weight-normal"><a href="/" class="text-
white">CONNECTIFY</a></h4>

      <% if (user) { %>
        <div class="flex-row my-3 my-md-0">
            <a href="#" class="text-white mr-2 header-search-icon" title="Search" data-
toggle="tooltip" data-placement="bottom"><i class="fas fa-search"></i></a>
```

```
        <span class="text-white mr-2 header-chat-icon" title="Chat" data-toggle="tooltip"
data-placement="bottom"><i class="fas fa-comment"></i></span>
        <a href="/profile/<%= user.username%>" class="mr-2"><img title="My Profile"
data-toggle="tooltip" data-placement="bottom" style="width: 32px; height: 32px; border-
radius: 16px;" src="<%= user.avatar %>"></a>
        <a class="btn btn-sm btn-success mr-2" href="/create-post">Create Post</a>
        <form action="/logout" method="POST" class="d-inline">
          <button class="btn btn-sm btn-secondary">Sign Out</button>
        </form>
      </div>
    <% } else { %>
      <form action="/login" method="POST" class="mb-0 pt-2 pt-md-0">
        <div class="row align-items-center">
          <div class="col-md mr-0 pr-md-0 mb-3 mb-md-0">
            <input name="username" class="form-control form-control-sm input-dark"
type="text" placeholder="Username" autocomplete="off">
          </div>
          <div class="col-md mr-0 pr-md-0 mb-3 mb-md-0">
            <input name="password" class="form-control form-control-sm input-dark"
type="password" placeholder="Password">
          </div>
          <div class="col-md-auto">
            <button class="btn btn-primary btn-sm">Sign In</button>
          </div>
        </div>
      </form>
    <% } %>
  </div>
</header>
```
Flash.ejs:

```
<% errors.forEach(function(message) { %>
    <div class="alert alert-danger text-center"><%= message %></div>
  <% }) %>

  <% success.forEach(function(message) { %>
    <div class="alert alert-success text-center"><%= message %></div>
  <% }) %>
```

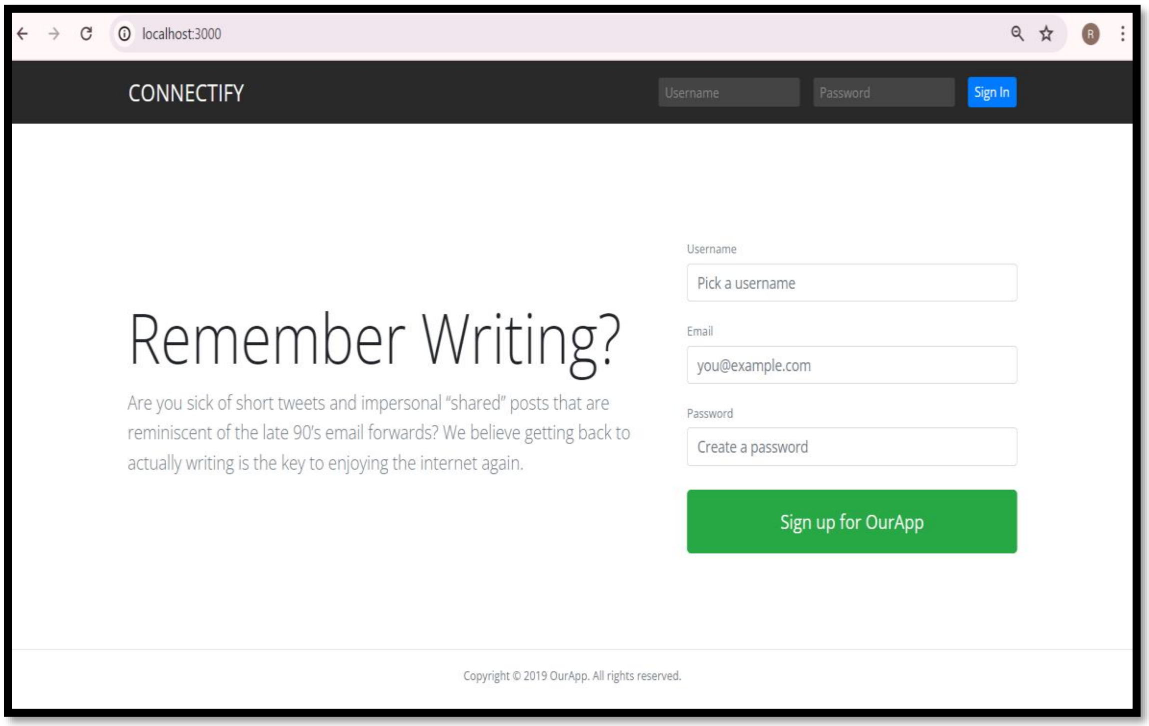
profileShared.ejs:

```
<h2><img class="avatar-small" src="<%= profileAvatar %>"> <%= profileUsername %>
    <% if (user && !isVisitorsProfile) {
      if (isFollowing) { %>
```

```
        <form class="ml-2 d-inline" action="/removeFollow/<%= profileUsername %>"
method="POST">
          <button class="btn btn-danger btn-sm">Stop Following <i class="fas fa-user-
times"></i></button>
        </form>
     <% } else { %>
        <form class="ml-2 d-inline" action="/addFollow/<%= profileUsername %>"
method="POST">
          <button class="btn btn-primary btn-sm">Follow <i class="fas fa-user-
plus"></i></button>
        </form>
     <% }
    } %>
  </h2>

  <div class="profile-nav nav nav-tabs pt-2 mb-4">
    <a href="/profile/<%= profileUsername %>" class="profile-nav-link nav-item nav-link
<%= currentPage == 'posts' ? 'active' : '' %>">Posts: <%= counts.postCount %></a>
    <a href="/profile/<%= profileUsername %>/followers" class="profile-nav-link nav-item
nav-link <%= currentPage == 'followers' ? 'active' : '' %>">Followers: <%=
counts.followerCount %></a>
    <a href="/profile/<%= profileUsername %>/following" class="profile-nav-link nav-item
nav-link <%= currentPage == 'following' ? 'active' : '' %>">Following: <%=
counts.followingCount %></a>
  </div>
```

Appendix-2: Screenshots



Remember Writing?

Are you sick of short tweets and impersonal "shared" posts that are reminiscent of the late 90's email forwards? We believe getting back to actually writing is the key to enjoying the internet again.

Username

Pick a username

Email

you@example.com

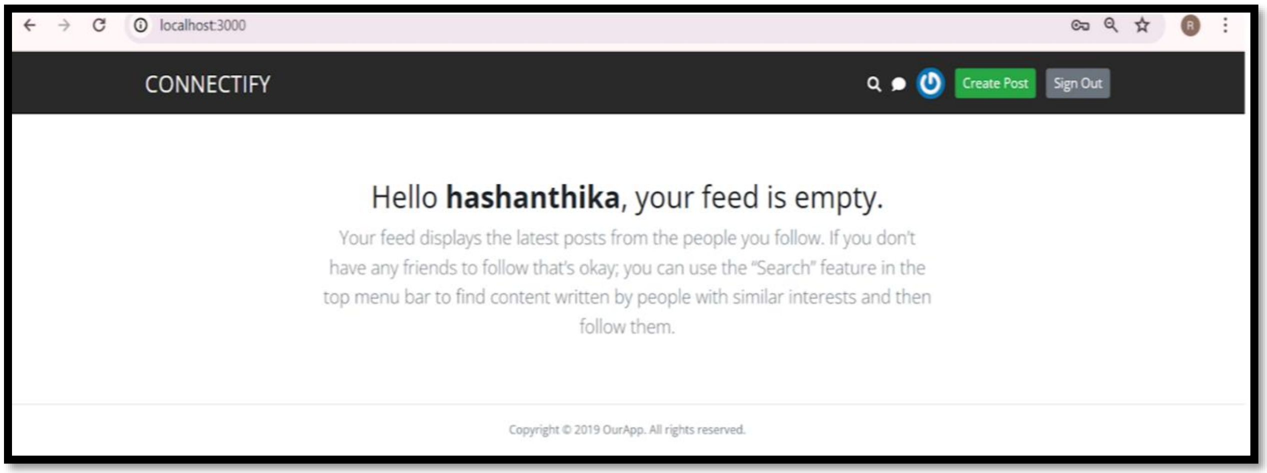
Password

Create a password

Sign up for OurApp

Copyright © 2019 OurApp. All rights reserved.

Figure 1: home page styling

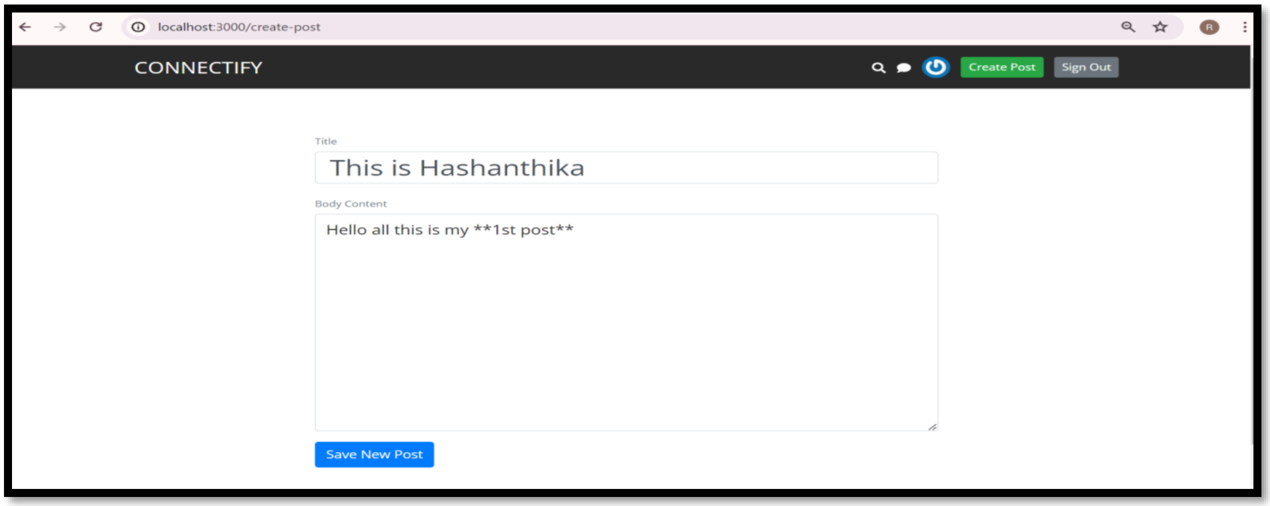


Hello **hashanthika**, your feed is empty.

Your feed displays the latest posts from the people you follow. If you don't have any friends to follow that's okay; you can use the "Search" feature in the top menu bar to find content written by people with similar interests and then follow them.

Copyright © 2019 OurApp. All rights reserved.

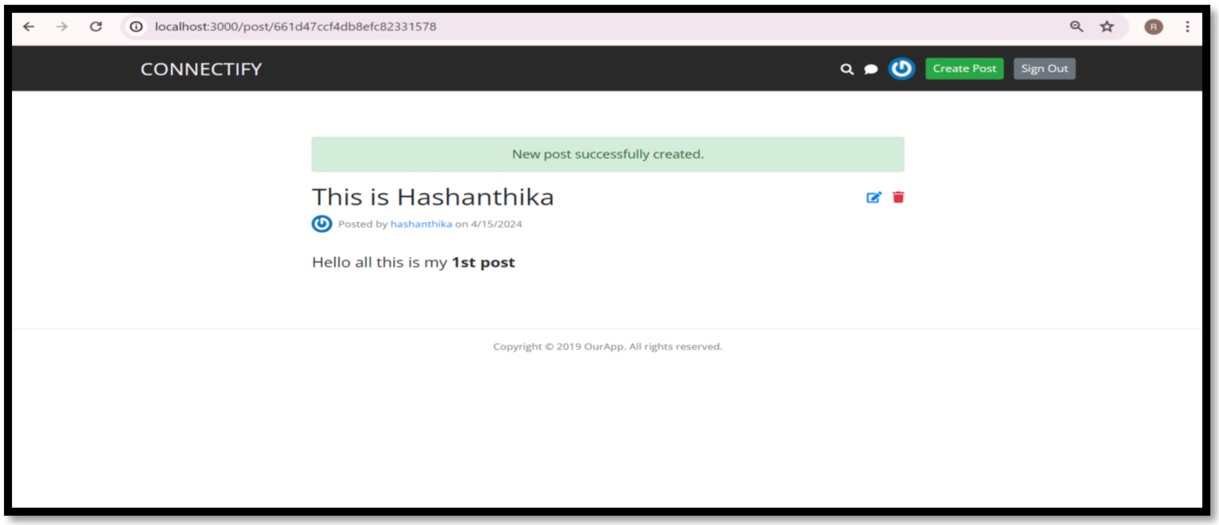
Figure 2 : page after registration or login



This is Hashanthika

Hello all this is my **1st post**

Figure 3: creating post



New post successfully created.

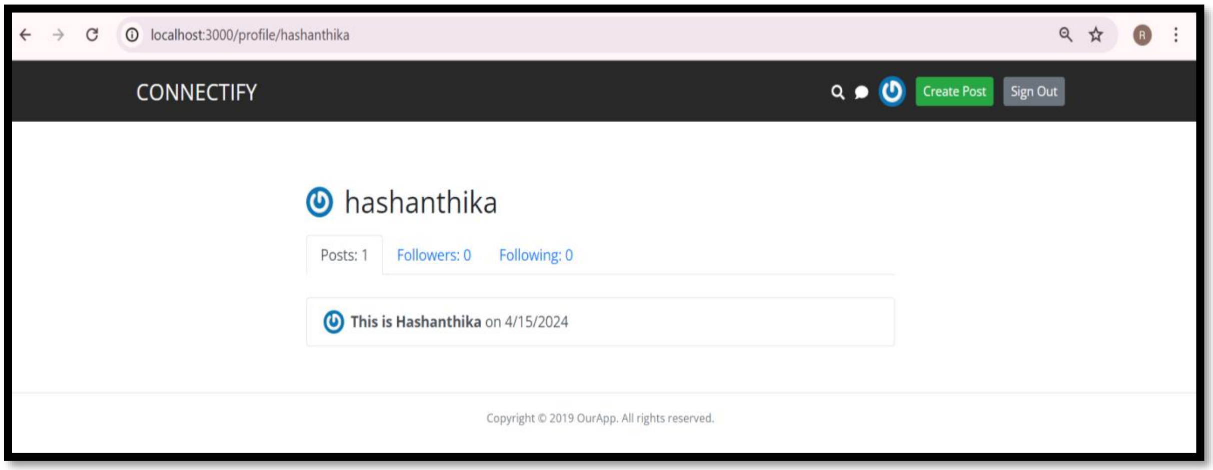
This is Hashanthika

Posted by hashanthika on 4/15/2024

Hello all this is my **1st post**

Copyright © 2019 OurApp. All rights reserved.

Figure 4: page after creating post with a success flash message



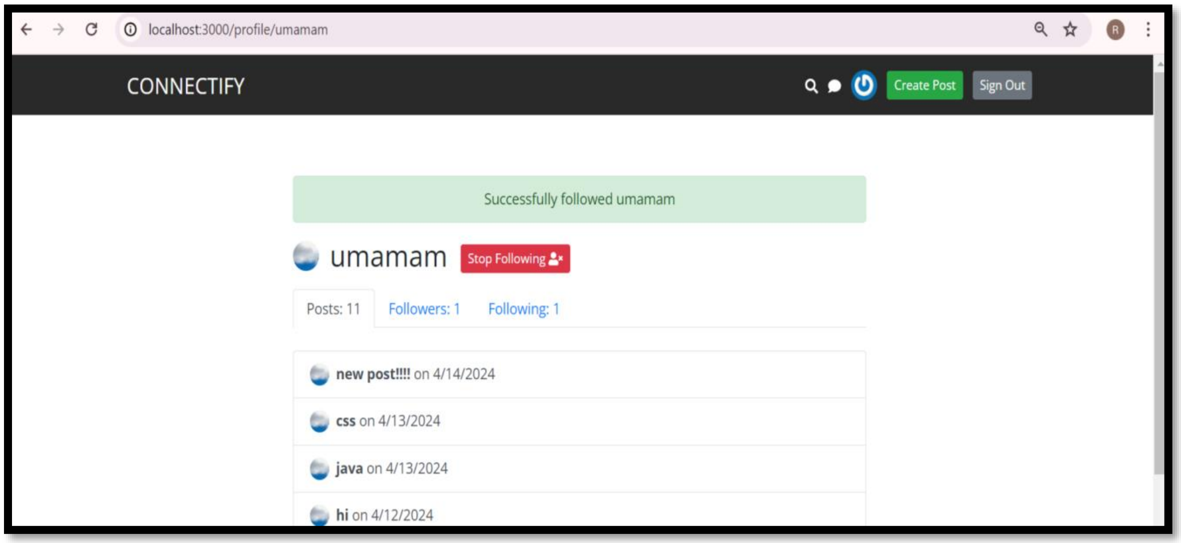
hashanthika

Posts: 1 Followers: 0 Following: 0

This is Hashanthika on 4/15/2024

Copyright © 2019 OurApp. All rights reserved.

Figure 5: users profile page


localhost:3000/profile/umamam

CONNECTIFY

Create Post Sign Out

Successfully followed umamam

umamam Stop Following

Posts: 11 Followers: 1 Following: 1

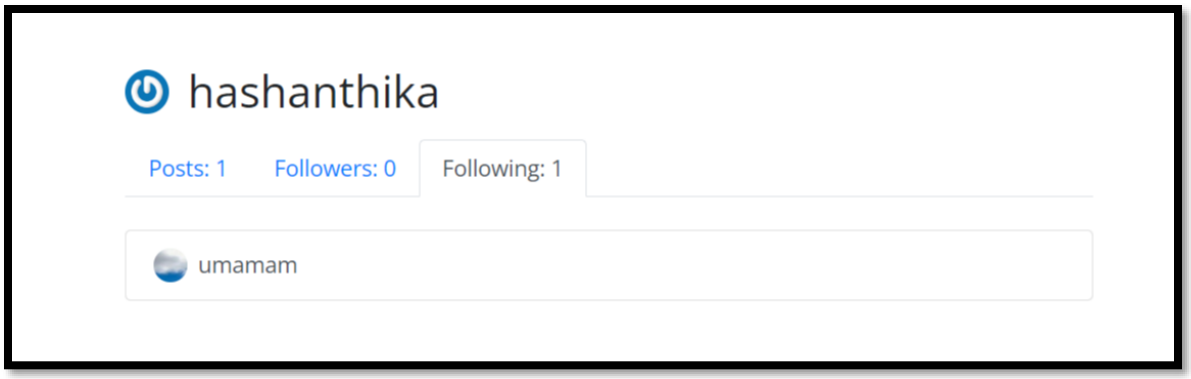
new post!!!! on 4/14/2024

css on 4/13/2024

java on 4/13/2024

hi on 4/12/2024

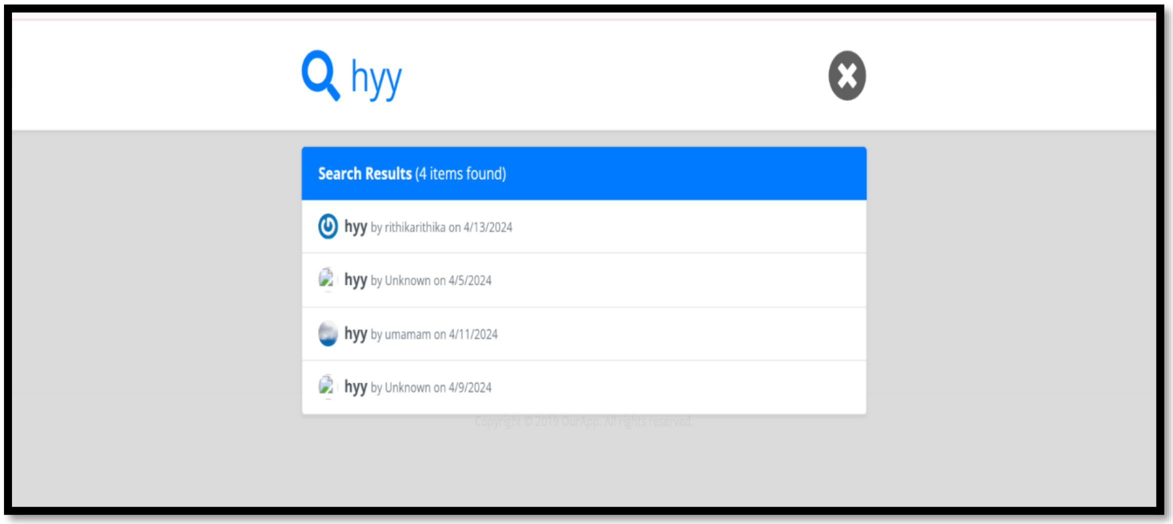
Figure 6: page after following an other user


hashanthika

Posts: 1 Followers: 0 Following: 1

umamam

Figure 7: entering number of follower


hyy

Search Results (4 items found)

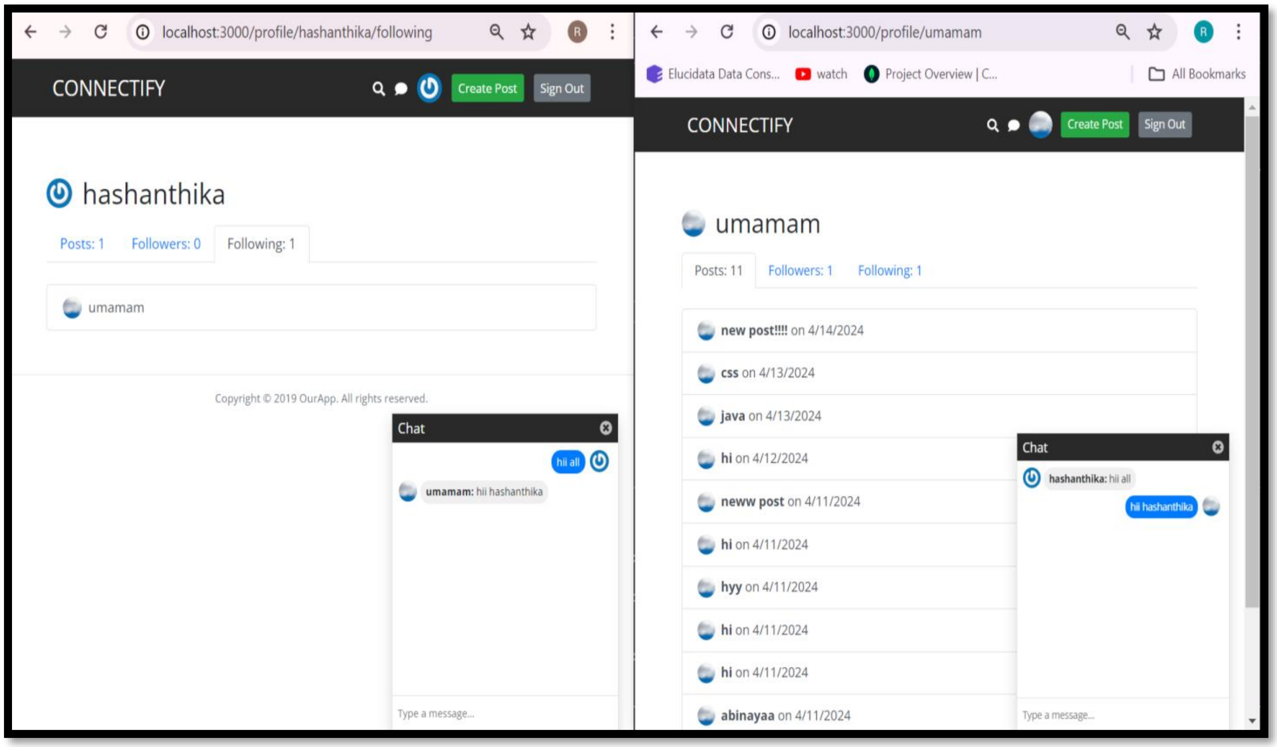
hyy by rithikarithika on 4/13/2024

hyy by Unknown on 4/5/2024

hyy by umamam on 4/11/2024

hyy by Unknown on 4/9/2024

Figure 8: searching feature



localhost:3000/profile/hashanthika/following

CONNECTIFY

hashanthika

Posts: 1 Followers: 0 Following: 1

umamam

Copyright © 2019 OurApp. All rights reserved.

Chat

hii all

umamam: hii hashanthika

localhost:3000/profile/umamam

CONNECTIFY

umamam

Posts: 11 Followers: 1 Following: 1

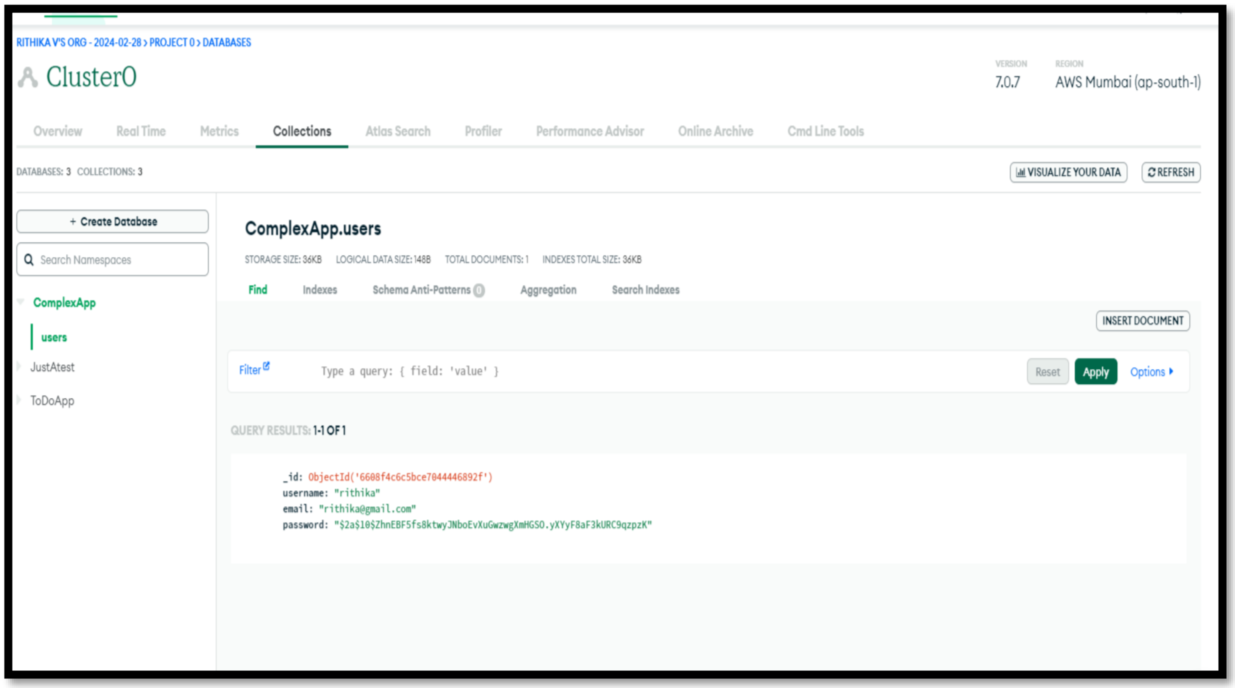
new post!!!! on 4/14/2024
css on 4/13/2024
java on 4/13/2024
hi on 4/12/2024
neww post on 4/11/2024
hi on 4/11/2024
hyy on 4/11/2024
hi on 4/11/2024
hi on 4/11/2024
abinayaa on 4/11/2024

Chat

hashanthika: hii all

hii hashanthika

Figure 9: chatting feature



RITHIKA V'S ORG - 2024-02-28 > PROJECT 0 > DATABASES

Cluster0

VERSION 7.0.7 REGION AWS Mumbai (ap-south-1)

DATABASES: 3 COLLECTIONS: 3

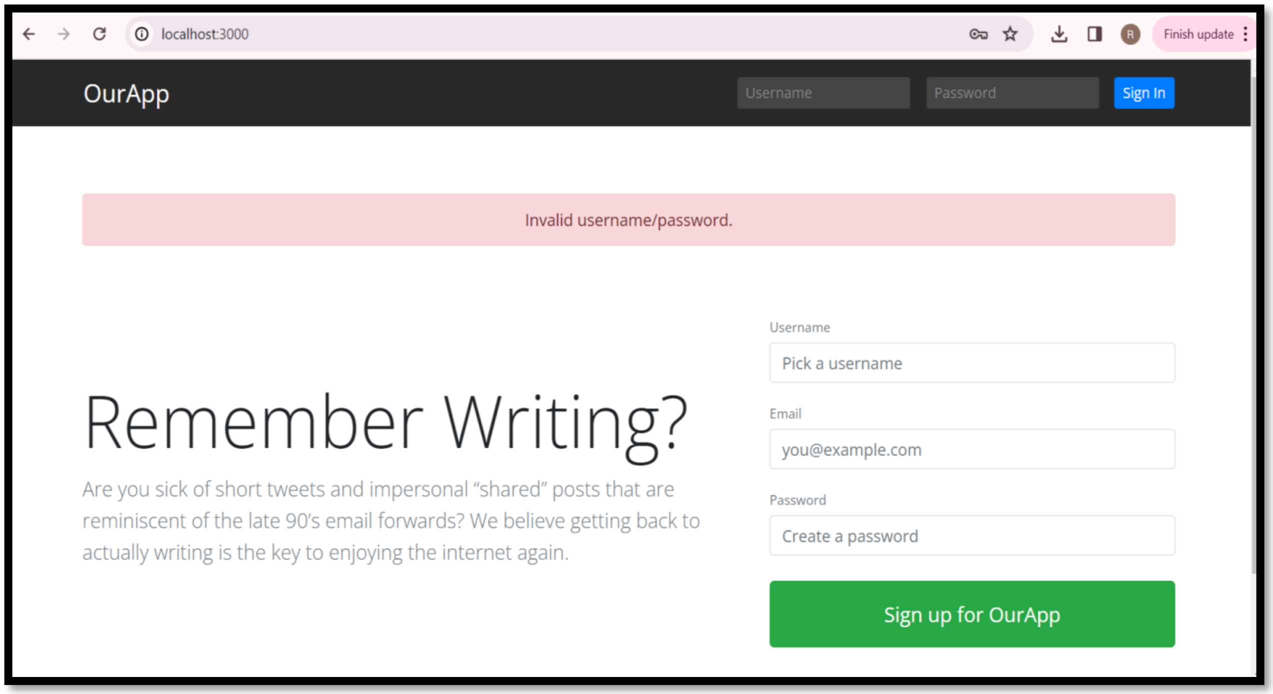
ComplexApp.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 148B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('6608f4c6c5bce7044446892f')
username: "rithika"
email: "rithika@gmail.com"
password: "$2a$10$ZhnEBF5fs8ktwyJNboEvXuGwzwgXmHGSO.yXYyF8aF3kURC9qzpzK"
```

Figure 10: Mongo db connection


localhost:3000

OurApp

Invalid username/password.

Remember Writing?

Are you sick of short tweets and impersonal "shared" posts that are reminiscent of the late 90's email forwards? We believe getting back to actually writing is the key to enjoying the internet again.

Username

Pick a username

Email

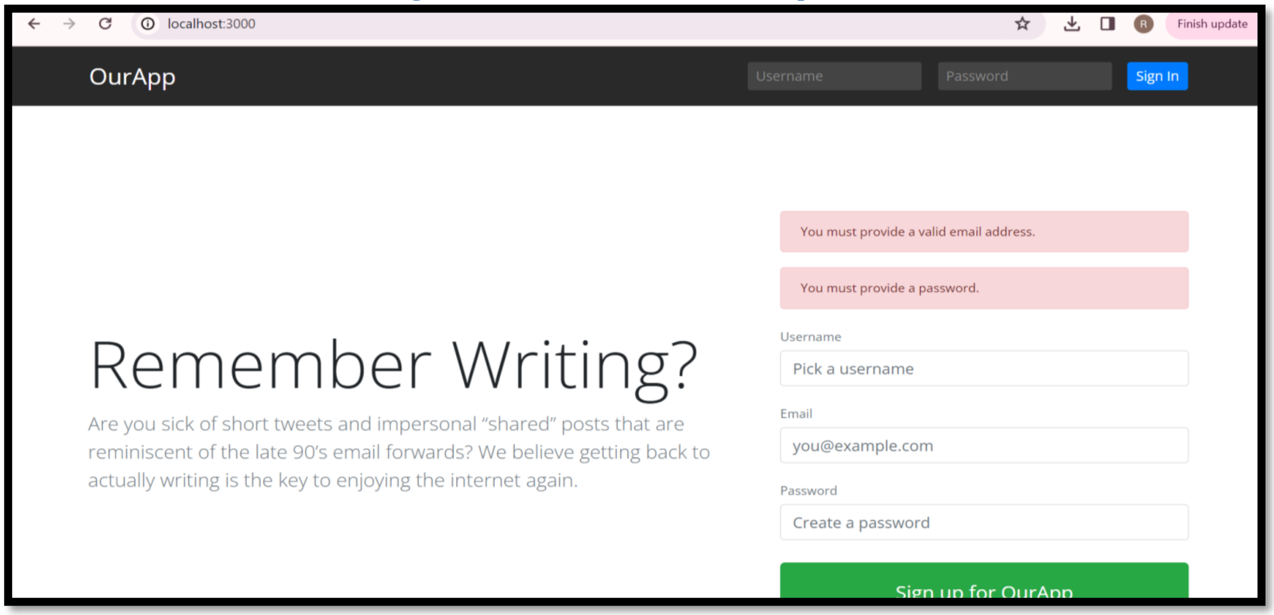
you@example.com

Password

Create a password

Sign up for OurApp

Figure 11: validation for username and password


localhost:3000

OurApp

You must provide a valid email address.

You must provide a password.

Remember Writing?

Are you sick of short tweets and impersonal "shared" posts that are reminiscent of the late 90's email forwards? We believe getting back to actually writing is the key to enjoying the internet again.

Username

Pick a username

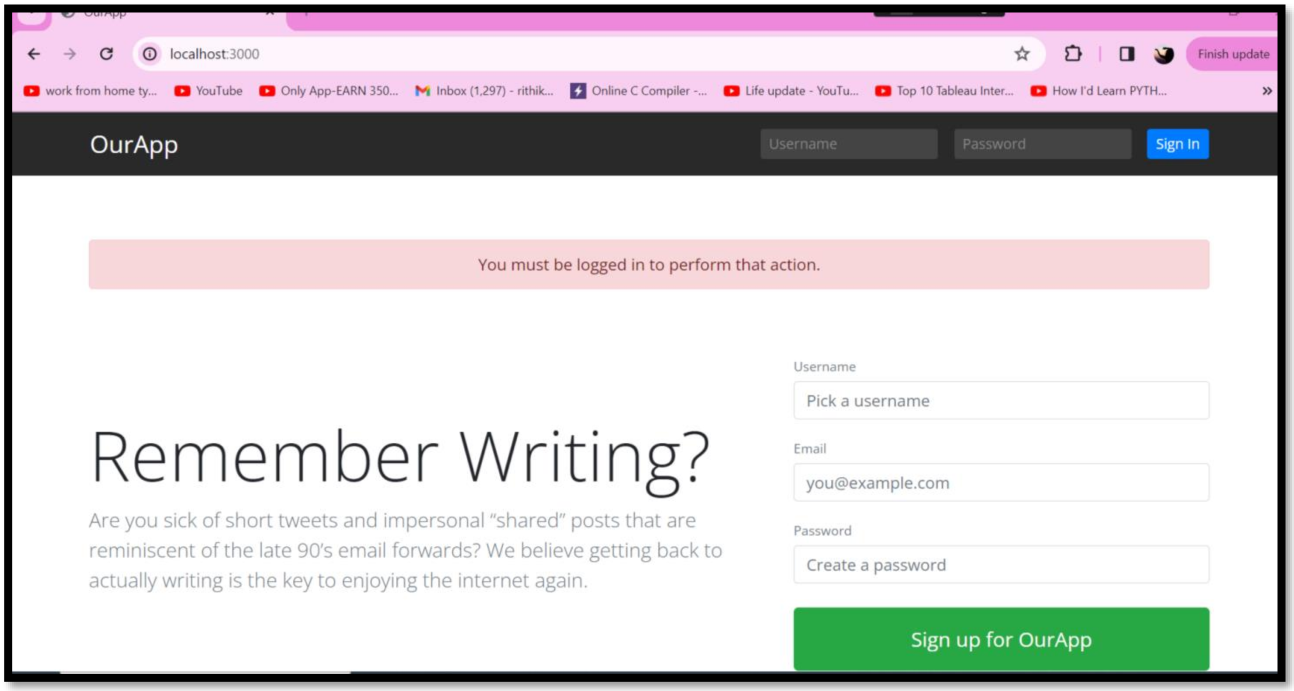
Email

you@example.com

Password

Create a password

Figure 12: validation for registration form


localhost:3000
work from home ty... YouTube Only App-EARN 350... Inbox (1,297) - rithik... Online C Compiler -... Life update - YouTu... Top 10 Tableau Inter... How I'd Learn PYTH...
Finish update
OurApp
Username
Password
Sign In

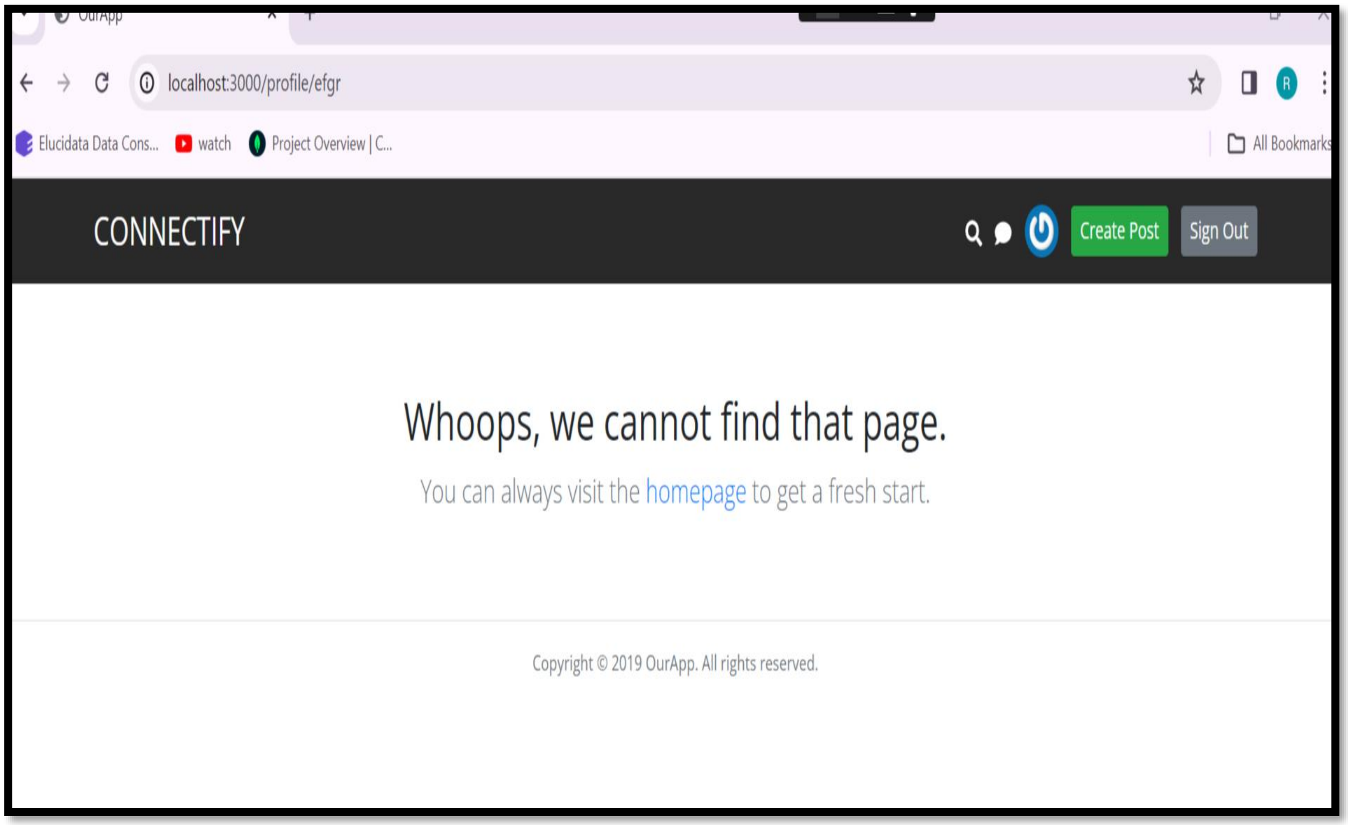
You must be logged in to perform that action.

Remember Writing?

Are you sick of short tweets and impersonal "shared" posts that are reminiscent of the late 90's email forwards? We believe getting back to actually writing is the key to enjoying the internet again.

Username
Pick a username
Email
you@example.com
Password
Create a password
Sign up for OurApp

Figure 13: Authentication of the user

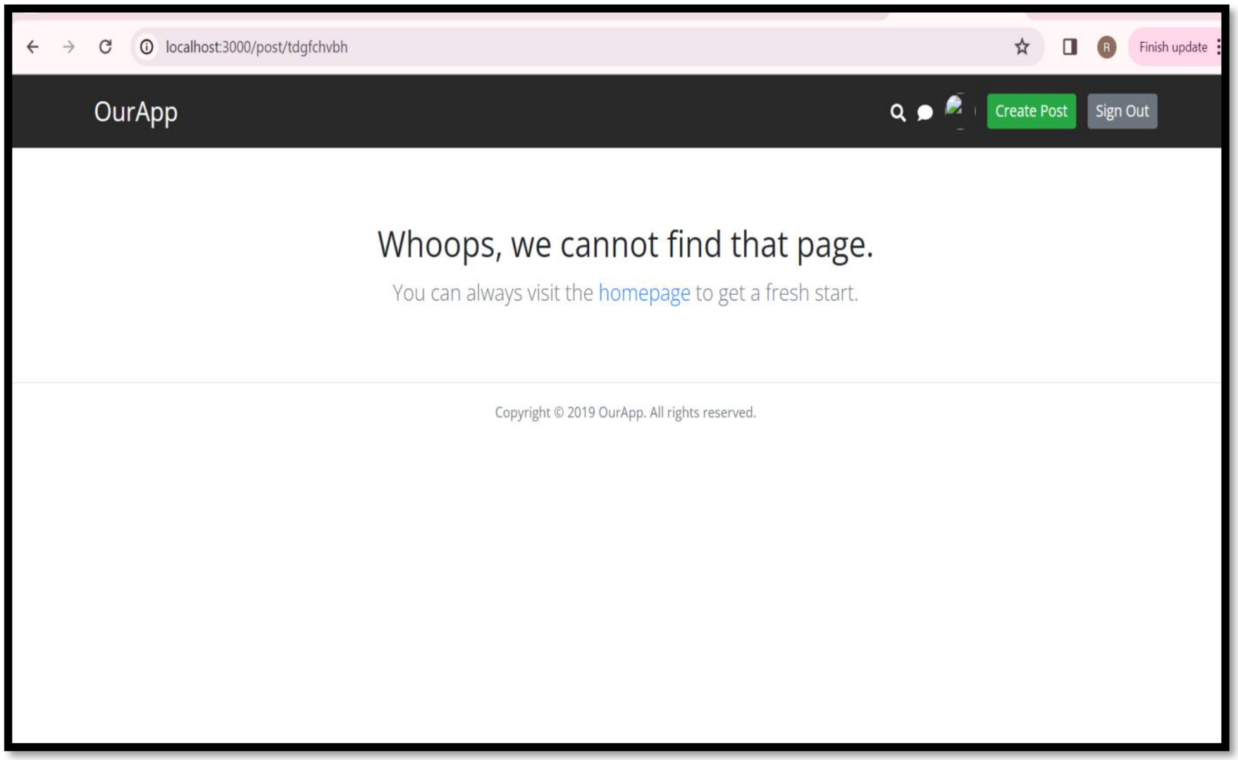

OurApp
localhost:3000/profile/efgr
Elucidata Data Cons... watch Project Overview | C...
All Bookmarks
CONNECTIFY
Create Post
Sign Out

Whoops, we cannot find that page.

You can always visit the homepage to get a fresh start.

Copyright © 2019 OurApp. All rights reserved.

Figure 14: profile Authentication

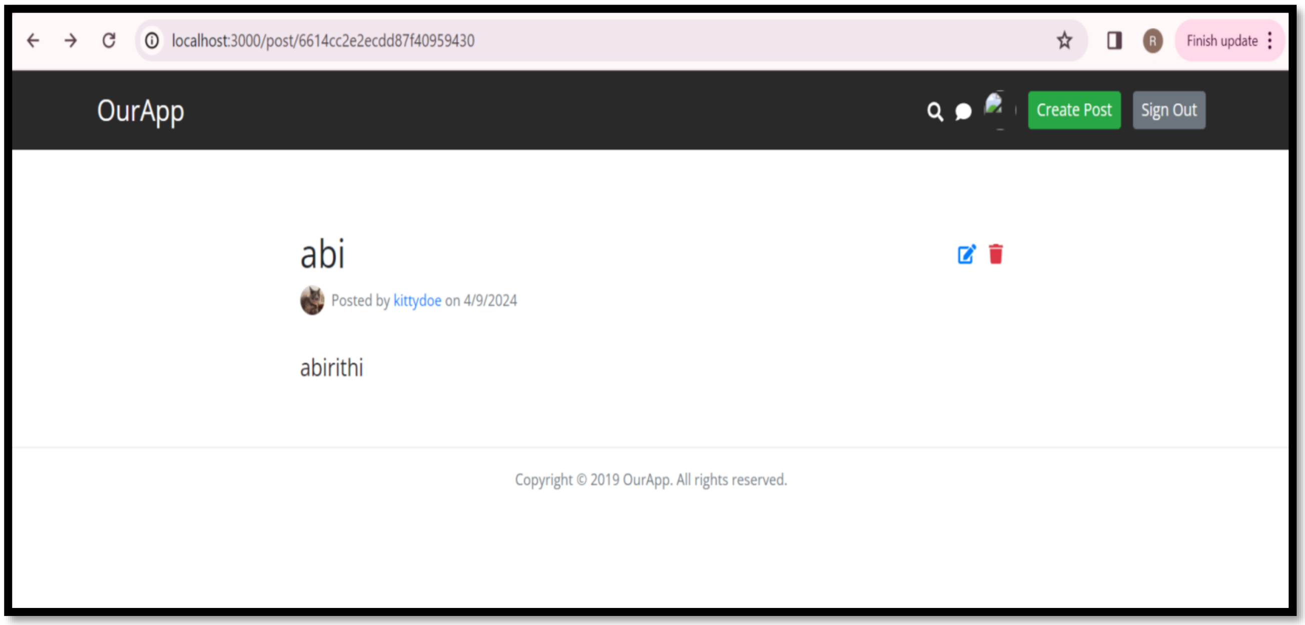


Whoops, we cannot find that page.

You can always visit the homepage to get a fresh start.

Copyright © 2019 OurApp. All rights reserved.

Figure 15: post Authentication



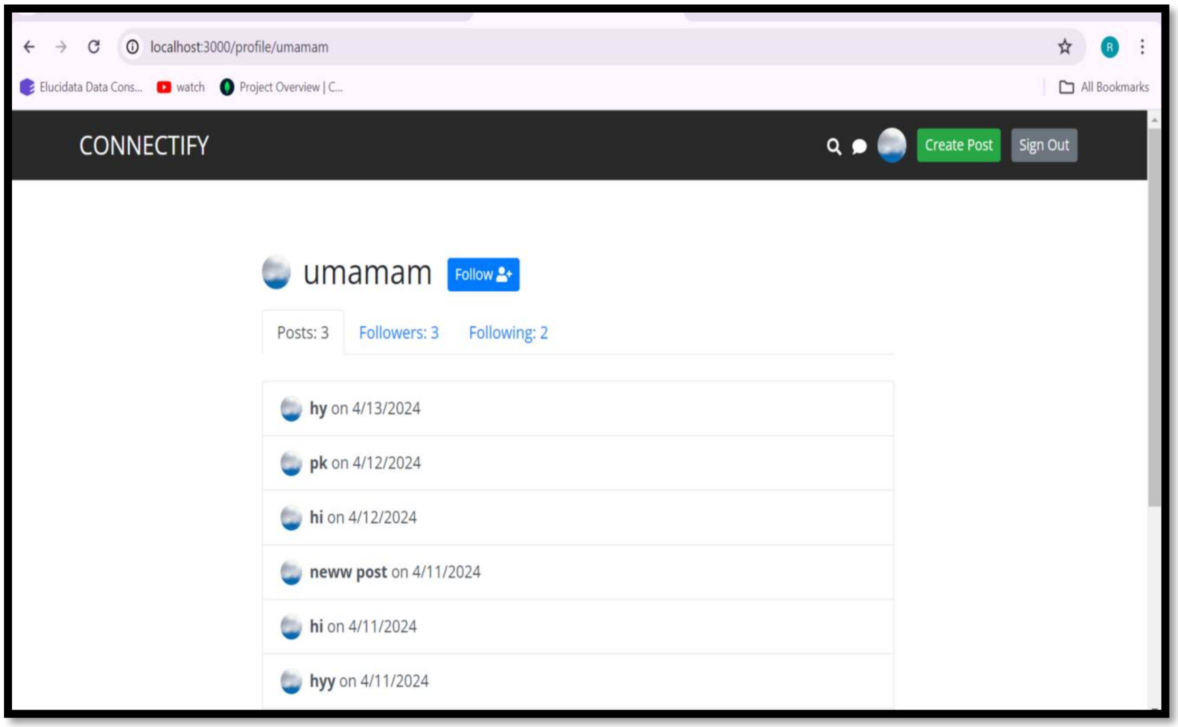
abi

Posted by kittydoe on 4/9/2024

abirithi

Copyright © 2019 OurApp. All rights reserved.

Figure 16: create and update button of the post



hy on 4/13/2024

pk on 4/12/2024

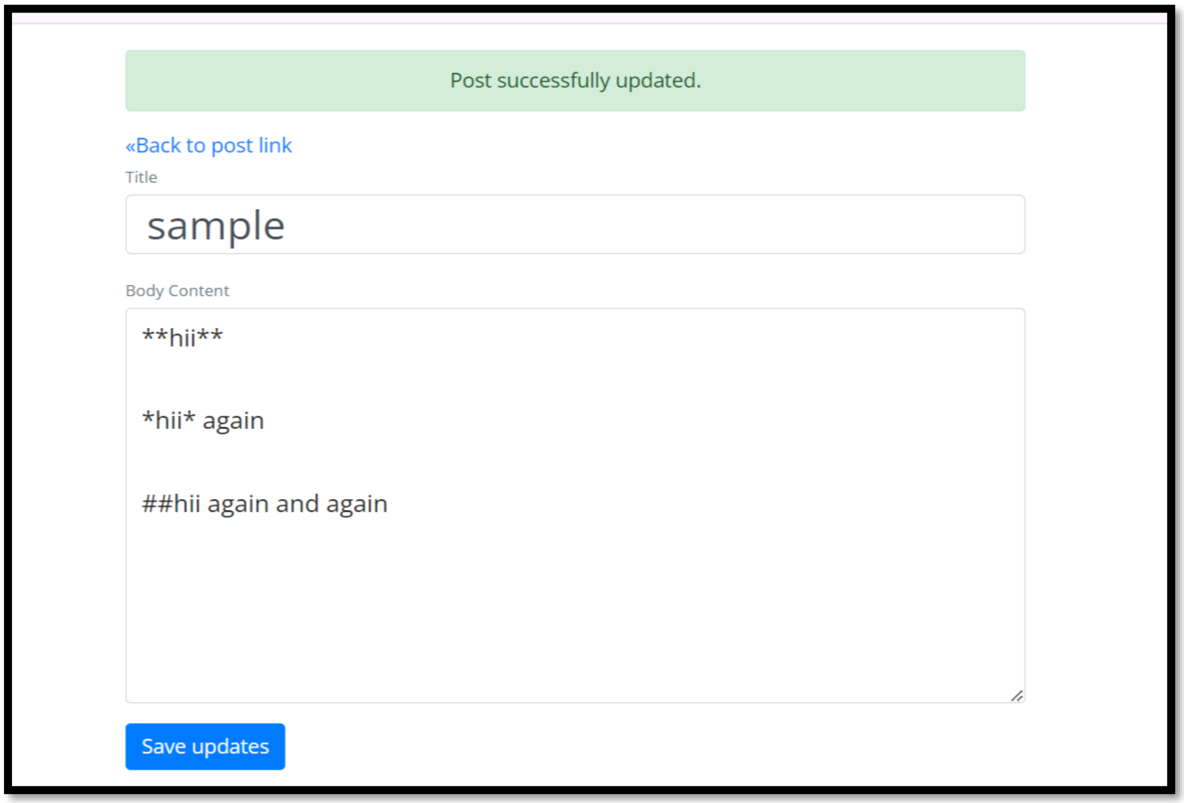
hi on 4/12/2024

neww post on 4/11/2024

hi on 4/11/2024

hyy on 4/11/2024

Figure 17: number of post posted by the user



Post successfully updated.

«Back to post link

Title

sample

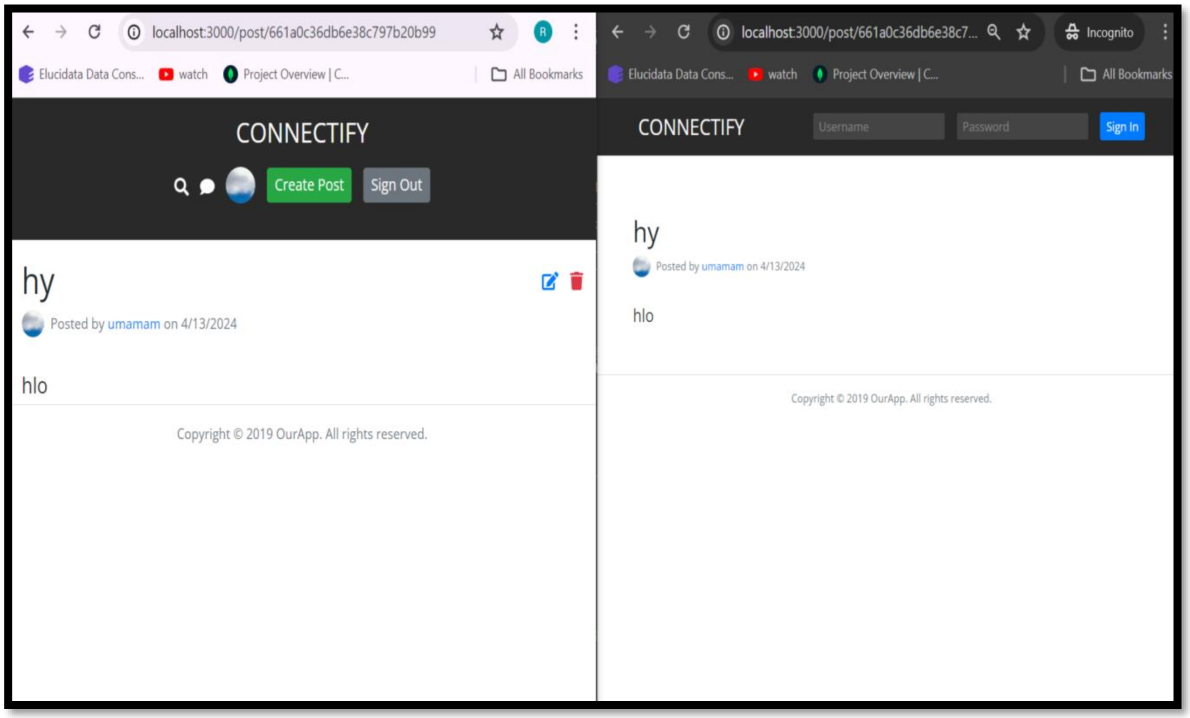
Body Content

```
**hii**

*hii* again

##hii again and again
```

Figure 18: updating the post



hy

Posted by umamam on 4/13/2024

hlo

Copyright © 2019 OurApp. All rights reserved.

Figure 19: update and delete button can be viewed and selected only based the account holder

REFERENCES

1. [1] Neil Randell “The Soul of the Internet”, 1997
2. [2] Cody Tinker, Christopher Valerino “Intel Hyper Threading Technology”
3. [3] Michael Hauben “The Net and the Netizens”, 1997
4. [5] Vincent Cerf and Robert Kahn “A Protocol for Packet Network Intercommunication”, 1974

WEB REFERENCE

1. Geeksforgeeks – learning nodejs and mongodb

   https://www.geeksforgeeks.org/introduction-about-node-js-and-mongodb/

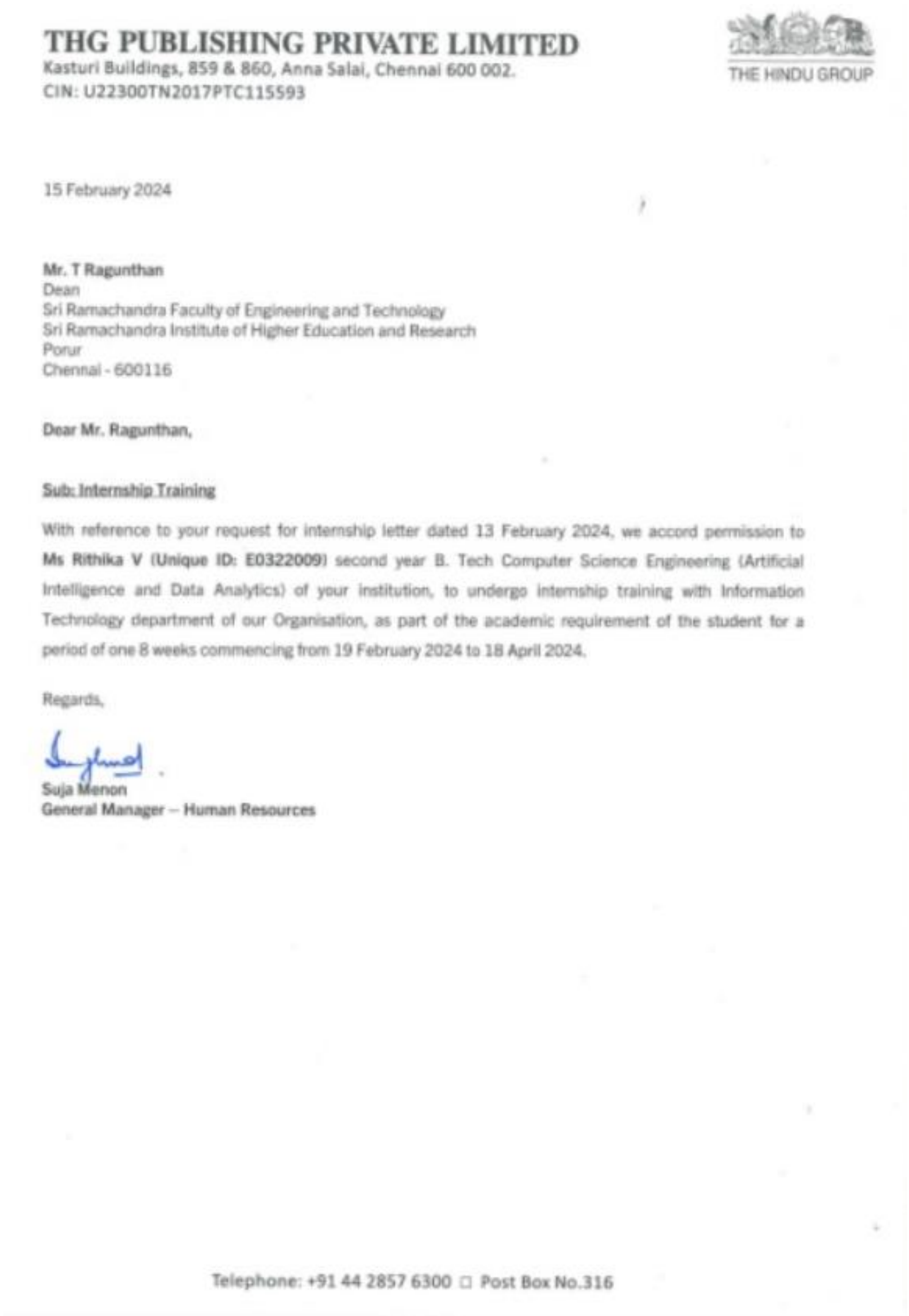
2. Learn JavaScript: Full-Stack from Scratch - udemy course

   https://www.udemy.com/course/learn-javascript-full-stack-from-scratch/?couponCode=24T4FS22124

WORKLOG

DAY	DATE	TASK DONE
1	20.2.24 - 23.2.24	Core concept of javascript, nodejs, expressjs
2	26.2.24 - 01.3.24	Concept of mongodb, connecting nodejs and mongodb and creating a basic CRUD operation application.
3	04.3.24 - 08.3.24	Starting with connectify application, ejs files, hashing password
4	11.3.24 - 15.3.24	Flash for validation, frontend ejs registration
5	18.3.24 - 22.3.24	Creating post, and saving in the database
6	25.3.24 - 29.3.24	Live search feature, sanitizing user generated Html
7	01.4.24 - 05.4.24	Letting user follow each other
8	08.4.24 - 12.4.24	Creating real time chat application

OFFER LETTER



THG PUBLISHING PRIVATE LIMITED
Kasturi Buildings, 859 & 860, Anna Salai, Chennai 600 002.
CIN: U22300TN2017PTC115593

THE HINDU GROUP

15 February 2024

Mr. T Ragunthan
Dean
Sri Ramachandra Faculty of Engineering and Technology
Sri Ramachandra Institute of Higher Education and Research
Porur
Chennai - 600116

Dear Mr. Ragunthan,

Sub: Internship Training

With reference to your request for internship letter dated 13 February 2024, we accord permission to **Ms Rithika V (Unique ID: E0322009)** second year B. Tech Computer Science Engineering (Artificial Intelligence and Data Analytics) of your institution, to undergo internship training with Information Technology department of our Organisation, as part of the academic requirement of the student for a period of one 8 weeks commencing from 19 February 2024 to 18 April 2024.

Regards,

[illegible]

Suja Menon
General Manager – Human Resources

Telephone: +91 44 2857 6300 □ Post Box No.316

COMPLETION CERTIFICATE