Project description (Blogger Content Management System)

At the very beginning of the world wide web, everything was put down in static websites that served as content needless of any back end connected database. A series of websites acted & loaded very fast as there was no processing of requests raised by client-server and queries to connect and perform actions in the database. Because the static web content is utilized by only a few web surfers, they require very few computing resources and has less traffic in the network. As we trace back to the history of CMS, the first website is modeled based on the hypertext markup language system in 1990 by Tim Berners-Lee. These static contents, once created, are difficult to change and require HTML knowledge to keep the content updated and are not suitable for the long run. A naïve nontechnical student/researcher who wants to update their creative static content when there are any updates or new versions added, requires hiring a developer to modify the content with HTML and keep it fresh. The early systems were just started with text and links; later the features were added in stages to incorporate other kinds of data.

However, in the early 2000s the evolution of content management systems has begun and one of which WordPress became very famous. As we advance more in the technology field, there is a constant need to update the content frequently and the way of publishing content on the world wide web is also advanced to create digital content dynamically from static content. In the current world, data produced per minute is in very huge amounts like million data points per day and it is not clearly structured. Storing this wide range of content in relational databases is too expensive and requires more computational resources to process the data actions with more memory.

Think of a new content management technology proposed to pile up all kinds of intelligence, a smooth semantic structure, and metadata about your content. For such, it requires a database that provides the flexibility to adapt quickly and update the content structure that runs natively in the cloud without any impact and efficiently to automate the scaling of the data model on multiple servers as per the future needs.

A Blogging Content Management System (CMS) is an application that can be used by any individual to create their own web content with all the functionalities and features without knowing any programming or technical knowledge. It enables end users like students, researchers, creators, or any other authors to create, edit, and delete blogs, and add any functional pages designed by themselves without needing to write a single line of code. For any person who wants to learn, research, publish and promote any kind of creative information, this blogger will help them in organizing easily and avoid duplicates and inconsistencies in information. This blogger will also have a comment section, so other users can post their thoughts on their blog or article along with like/dislike functionality. Owner of the blog can block other users to stop them commenting abusive contents.

The main aim of our project is to develop a NoSQL database, a document-oriented data model that stores various types of unstructured data and can handle huge volumes in size so that it will incorporate and serve any kind of information like text, audio, links, images, articles in a single database with much faster and less expensive. In short, a NoSQL database (MongoDB) is required for a content platform that is cheaper, scalable, flexible, and faster than most of the current web-based systems with traditional relational databases suits best to this blogging content management system.

Database description

Blogging Content Management System lets end users to create and publish their creative ideas or experiences in the form of blogs through this application. Each post has a unique Id and an author that distinguishes between other posts in the application. A user can post more than one blog and can comment in other stories as well. Our Blog application has mainly three collections Users, Stories, and Comments to store the required data respectively.

For each user, we must store this information to identify and authorize such as, username, email, password with hashed, timestamps created and modified dates. Each user will have unique email & username and they will be able to use email and password to be able to login to the system. optionally we can also store the profile image of user by storing the user photo. All the user data is tracked with help of created date and updated date to find out when the record is created or last updated dates. Additionally, we can store the id of users of those who are blocked by story owner to comment on his own stories in the user collection document blocked Users field array of ObjectId. Unique key of the user document is ObjectId (_Id) stored automatically on inserting. For each user, there is read List field which we store the blog ids of the stories which user likes to read later or wants to read again and again.

Each blog/story will have title, content, poster (image source link), author fields to uniquely define the stories of a user. For each post, created by details of author id which is referenced to the user document link will automatically get filled up in backend based on the logged in user. All post will store the author details from user collections along with timestamps created and updated dates of blog in story collection. For each post, all the likes and unlike by users will be stored as user ObjectId in form of array list referencing to user collection documents. Each post will have the comment field which will store all the comments ids in form of array which references to the comment collection. Optionally story collection has read time field which will display the time takes to read the blog based on the count of words. Post will also contain the slug which contains unique id of story with hyphen separated words. The content field in story collection will store all kinds of html tags with content like bullets, image links, font styling etc. so the user can make his blog colorful and organized if needed rather than pure text content

Each Comment collection will store the story id and author id along with content of the comment. Comment collection also have likes/ dislikes array to store the user ids of those users who likes or dislikes the array. Optionally we also added star field to have rating of the comments or with 5 stars. Comment section also has timestamps to store created and updated dates and unique _id which automatically created on insertion of document

Functionalities of Blog Application:

- On landing page, all end users can see the stories/blogs on our application
- User can only read stories/comments on landing page
- To write / post comment user needs to be registered and logged in
- User can register and login with email, password
- Once logged in user can create his story, update and able to delete his story
- User can only read other users' stories and be able to like/dislike and can comment on other blogs/stories
- User can block other users, if he founds abusive comments on his story so that other user can't post comments on his story in future.
- User can add comments and delete his comments, user can like/dislike other user comments
- User can edit his profile, update password, can add profile photo
- Blog/Story may contain any type of text like bullets, hyperlinks, image files, table, text fonts styling etc. and can also add blog header image

Database diagram & Data Dictionary

Our database contains three collections

- User
- Story
- Comments

NoSQL Database Model Document-Based Store (JSON File): All data is stored in a document, so there's no need for cross-referencing and instead of storing information in a table, it's stored in a document. The database we used is MongoDB which stores data in form of documents.

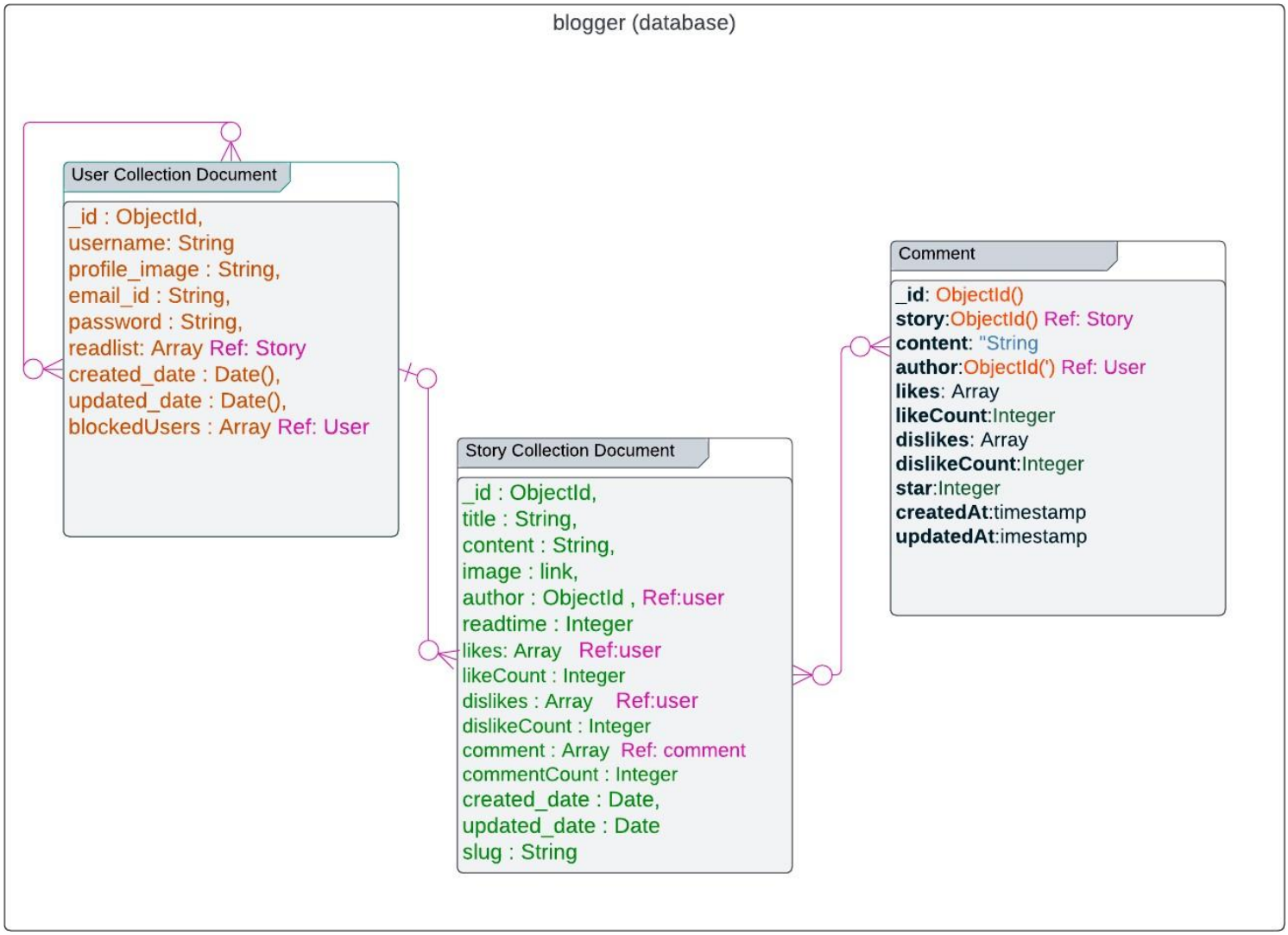
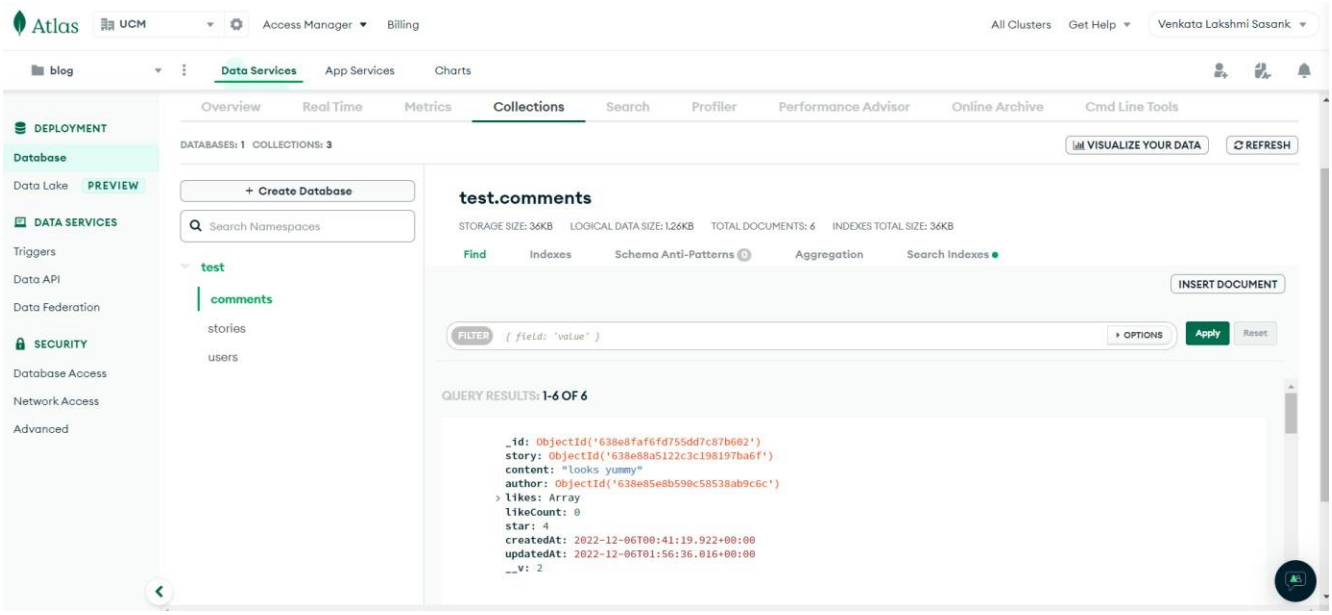


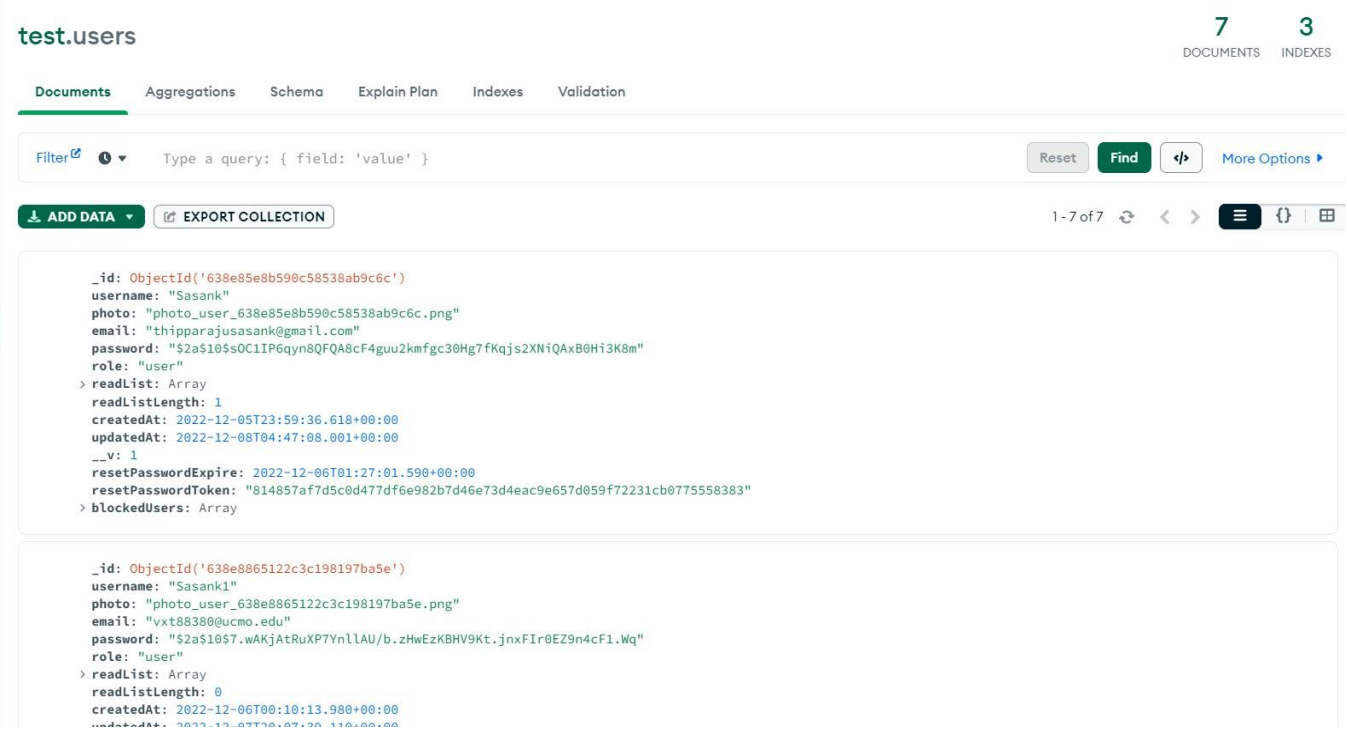
Fig 1. Database Design with data types

Sample Data Documents:

Collections:



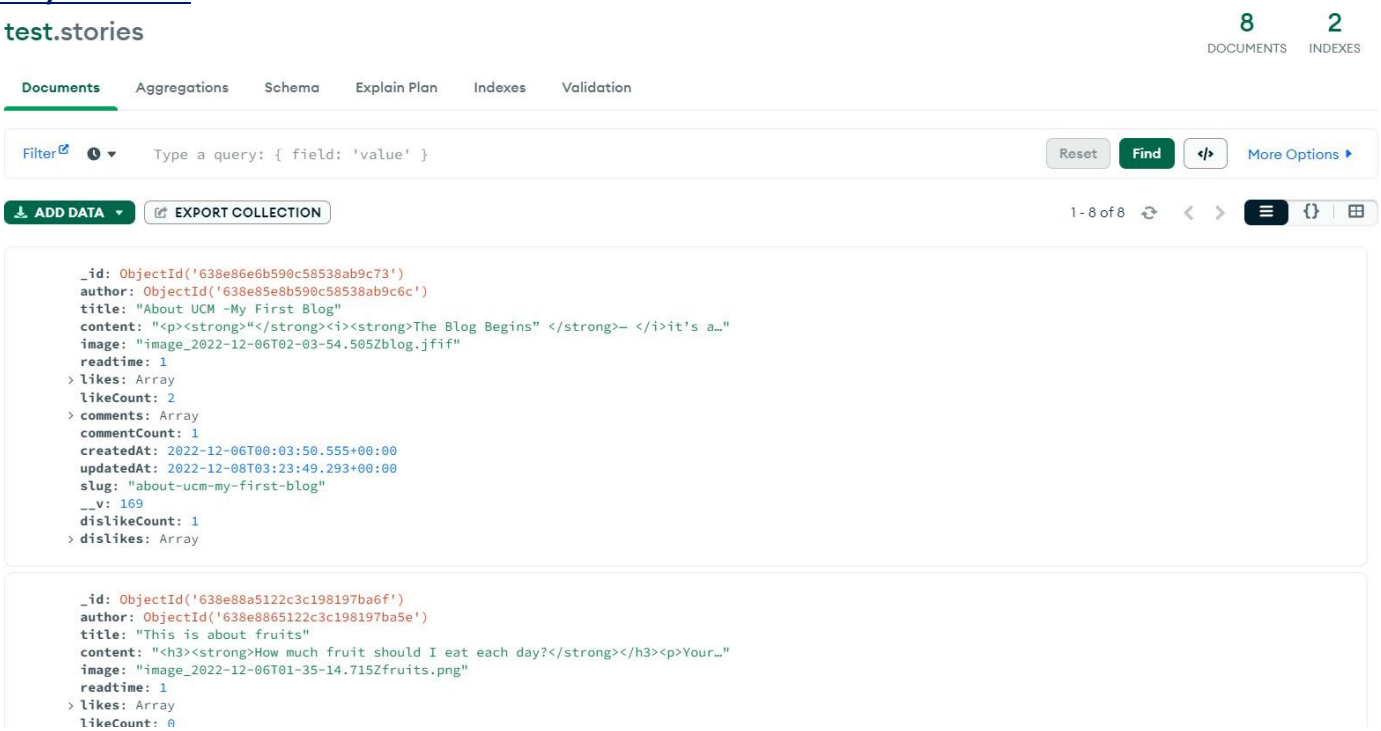
User Collection:



User document:


```
{
    "_id": {
      "$oid": "638f6a1230bfb02d7c76477f"
    },
    "username": "jeevan",
    "photo": "user.png",
    "email": "jeevan@gmail.com",
    "password": "$2a$10$ks8ERzLNAtEk.vjYBj4y2.Cmcfk3MD/xmbqM5LGbIUEyPbU4rFCu.",
    "role": "user",
    "readList": [
      {
        "$oid": "638f6a4630bfb02d7c76478f"
      },
      {
        "$oid": "638e86e6b590c58538ab9c73"
      }
    ],
    "readListLength": 2,
    "createdAt": {
      "$date": {
        "$numberLong": "1670343186510"
      }
    },
    "updatedAt": {
      "$date": {
        "$numberLong": "1670529726699"
      }
    },
    "__v": 153,
    "blockedUsers": [
      {
        "$oid": "638e8865122c3c198197ba5e"
      }
    ]
  }
```

Story Collection:

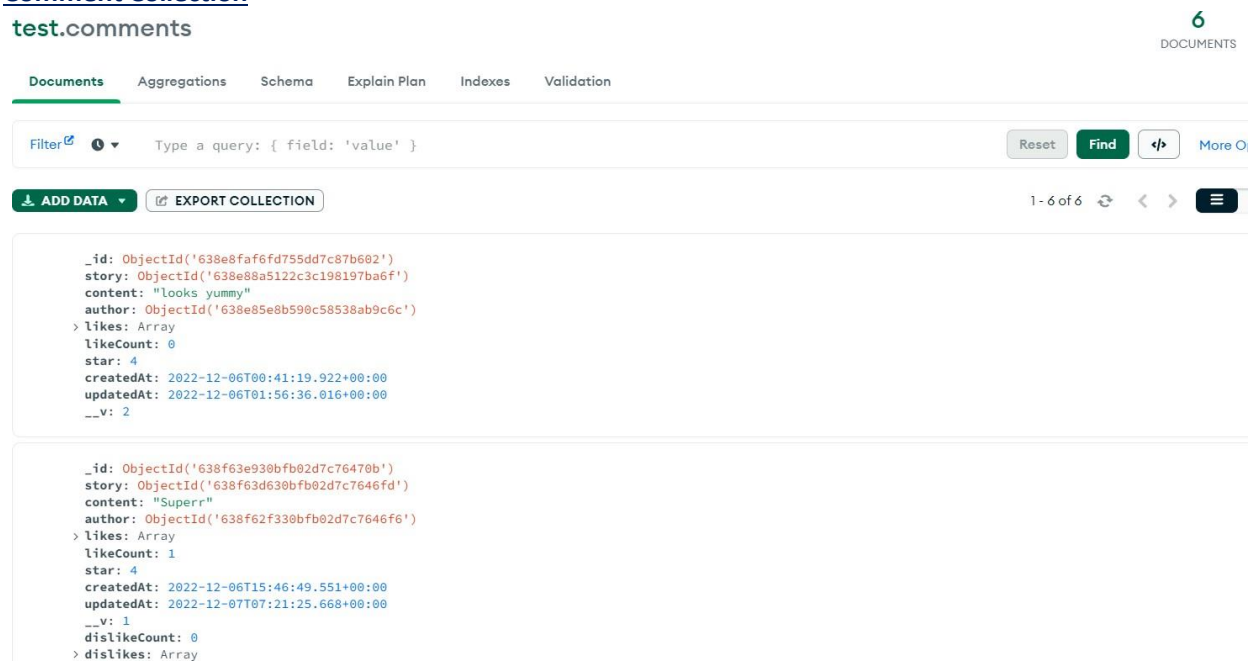


Story Document:


```
{
    "_id": {
      "$oid": "638e86e6b590c58538ab9c73"
    },
    "author": {
      "$oid": "638e85e8b590c58538ab9c6c"
    },
    "title": "About UCM -My First Blog",
    "content": "<p><strong>“</strong><i><strong>The Blog Begins” </strong>—
</i>it’s a great first blog post example that starts with a catchy headline, and
quickly molds into telling the story and setting expectations for the future
updates.</p><h2>The University of Central Missouri&amp;nbsp;</h2><p>UCM is an
innovative school that offers a high-quality education while remaining one of the
most affordable universities in the country.</p><p>&amp;nbsp;</p><p>Here, your
education extends beyond books to include service-learning projects, study abroad
opportunities and real-world experiences. Plus, UCM is the state leader in degree
completion among public universities. Statistics recently released by the
Missouri Department of Higher Education (MDHE) show that UCM’s degree completion
rate per full-time equivalent (FTE) is the highest among all of Missouri’s public
universities and more than double the state benchmark.</p><p>&amp;nbsp;</p><p>At UCM
we strive to give you the student services, resources and support you need to
experience academic success and earn your degree on time. Engaged learning.
Future-focused academics. Culture of service. Worldly perspective. These are the
```

```
building blocks that will allow you to thrive at UCM—and into your future. Take classes online or on our campuses in Warrensburg and Lee’s Summit, Missouri. Take the <a href=\"https://www.massinteract.com/university-of-central-missouri/\"><strong>Virtual Self-Guided Tour</strong></a> to view the Warrensburg and Lee's Summit campuses before <a href=\"https://www.ucmo.edu/future-students/visit-ucm/index.php\"><strong>visiting in person</strong></a>.</p><p> </p><p>The choice of words is simple. The length of the intro is fairly short and to the point, one of the things that makes the intro unique and showcases the personality of the blog.</p>",
    "image": "image_2022-12-06T02-03-54.505Zblog.jfif",
    "readtime": 1,
    "likes": [
      {
        "$oid": "638e85e8b590c58538ab9c6c"
      },
      {
        "$oid": "638e8865122c3c198197ba5e"
      }
    ],
    "likeCount": 2,
    "comments": [
      {
        "$oid": "6390d4235ad0ddfbc48c35a6"
      }
    ],
    "commentCount": 1,
    "createdAt": {
      "$date": {
        "$numberLong": "1670285030555"
      }
    },
    "updatedAt": {
      "$date": {
        "$numberLong": "1670469829293"
      }
    },
    "slug": "about-ucm-my-first-blog",
    "__v": 169,
    "dislikeCount": 1,
    "dislikes": [
      {
        "$oid": "638f6a1230bfb02d7c76477f"
      }
    ]
  }
```

Comment Collection



Comment Document


```
{
    "_id": {
      "$oid": "638f6b4f30bfb02d7c7647f8"
    },
    "story": {
      "$oid": "638f6a4630bfb02d7c76478f"
    },
    "content": "Its a good one",
    "author": {
      "$oid": "638e85e8b590c58538ab9c6c"
    },
    "likes": [
      {
        "$oid": "638e8865122c3c198197ba5e"
      },
      {
        "$oid": "638e85e8b590c58538ab9c6c"
      }
    ],
    "likeCount": 2,
    "star": 4,
    "createdAt": {
      "$date": {
```

```
        "$numberLong": "1670343503130"
      }
    },
    "updatedAt": {
      "$date": {
        "$numberLong": "1670474363947"
      }
    },
    "__v": 15,
    "dislikeCount": 1,
    "dislikes": [
      {
        "$oid": "638f6a1230bfb02d7c76477f"
      }
    ]
  }
```


User Interfaces and forms

Technologies Utilized:

- **Frontend**: ReactJS, NextJS, CSS, JavaScript, Bootstrap
- **Backend**: NodeJS
- **Database**: MongoDB

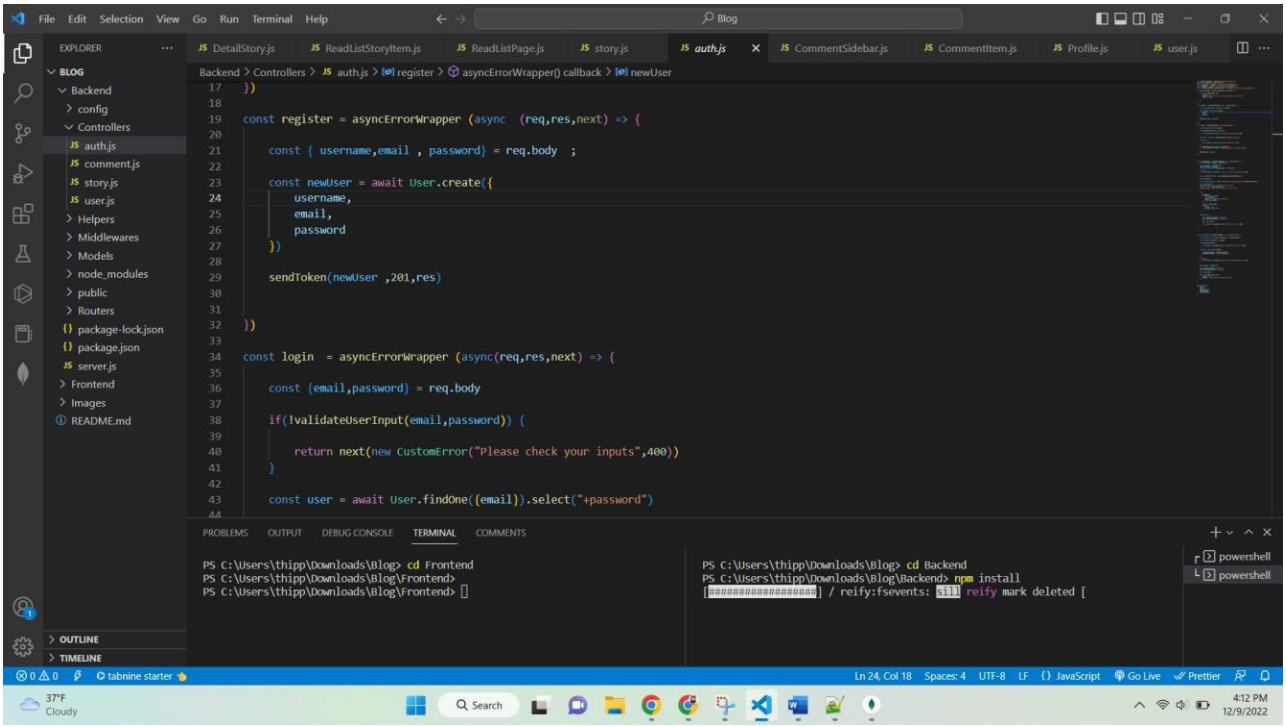
To Run the Application **Open two terminals:**

In One Terminal: from project folder location, use the below commands to start mongo DB connection

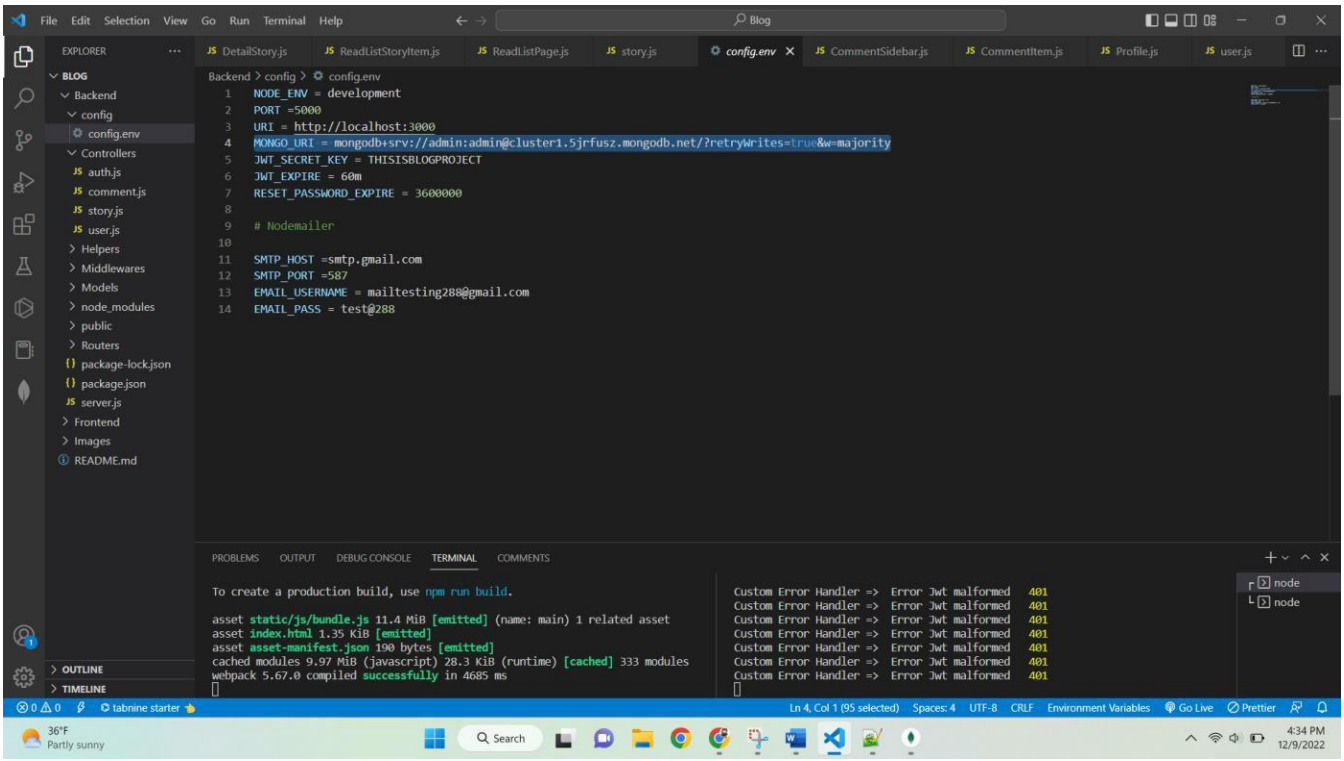
- cd Backend
- npm install
- npm start

In Other Terminal: from project folder location, use the below commands to start Frontend react connection

- cd Frontend
- npm install
- npm start

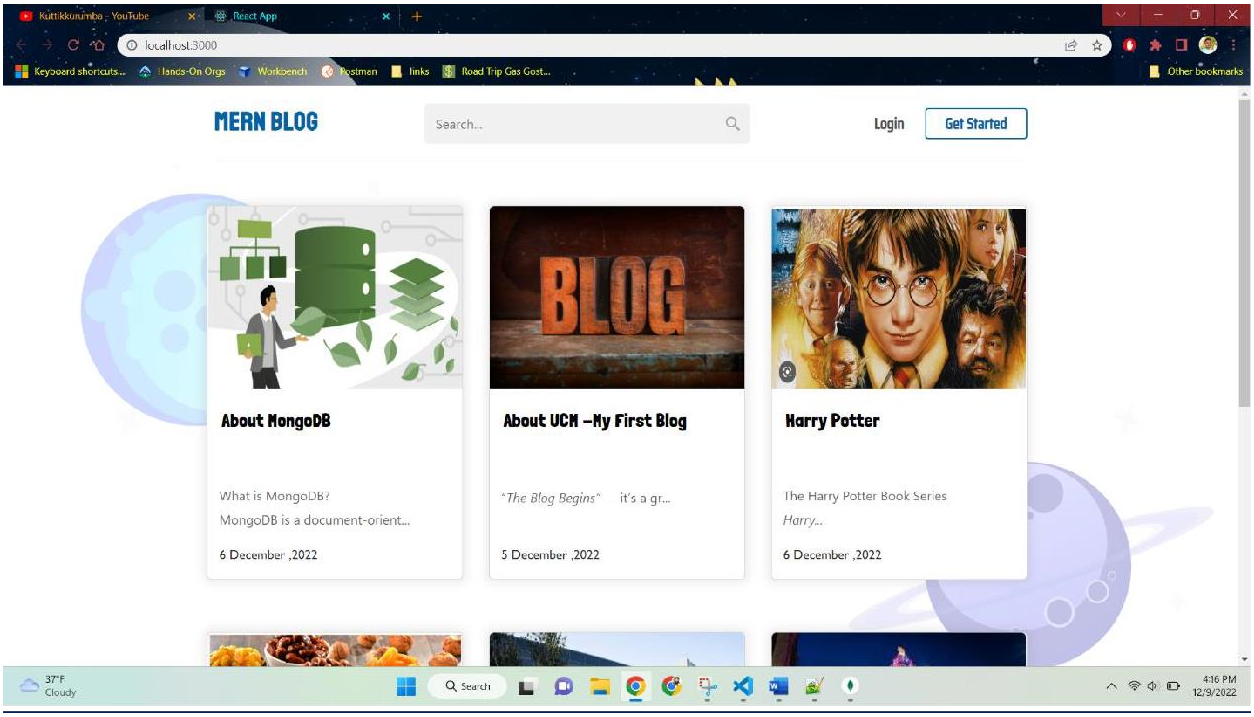


To Change Database Connection URL : Blog -> Backend -> Config -> config.env file Update mongo DB URI

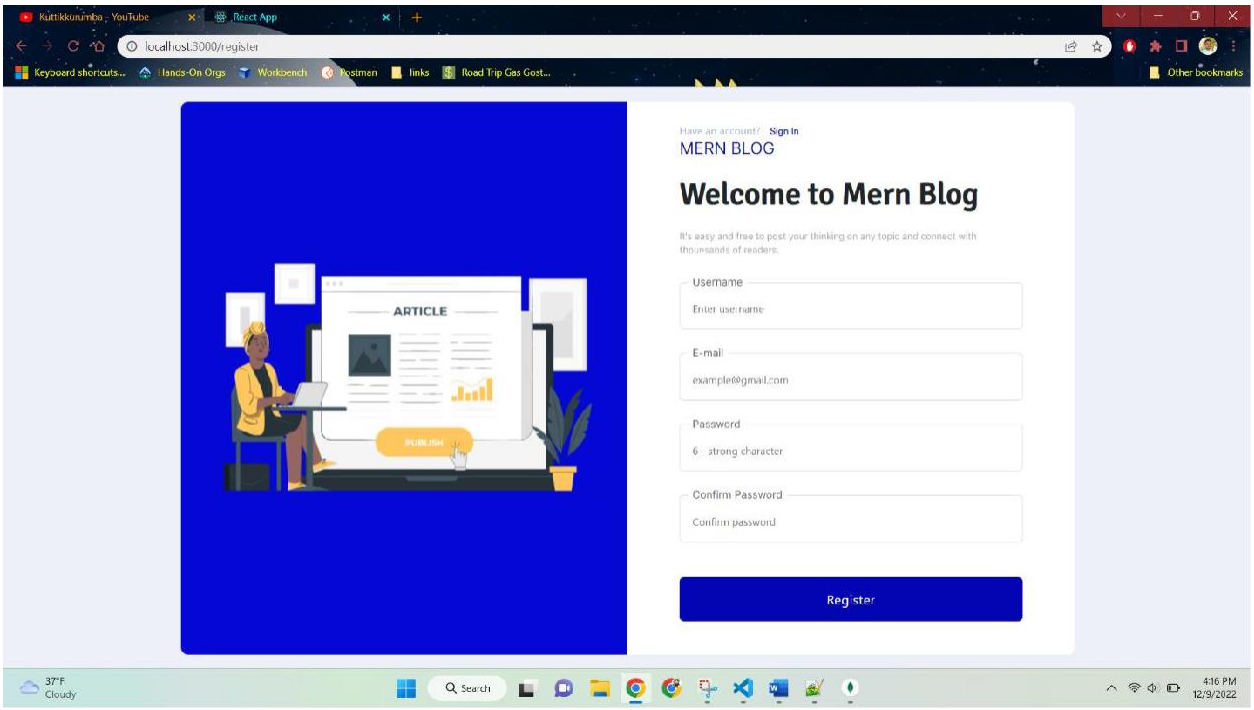


Screenshots of User Interface

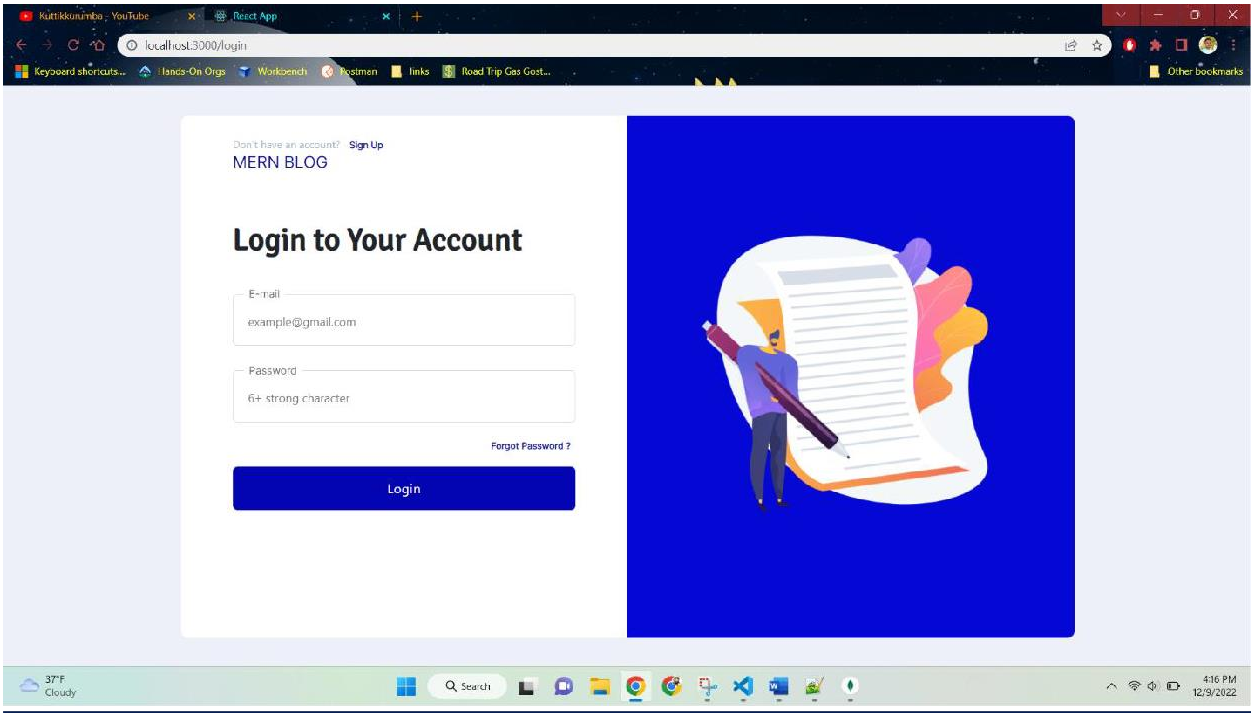
Home Page:



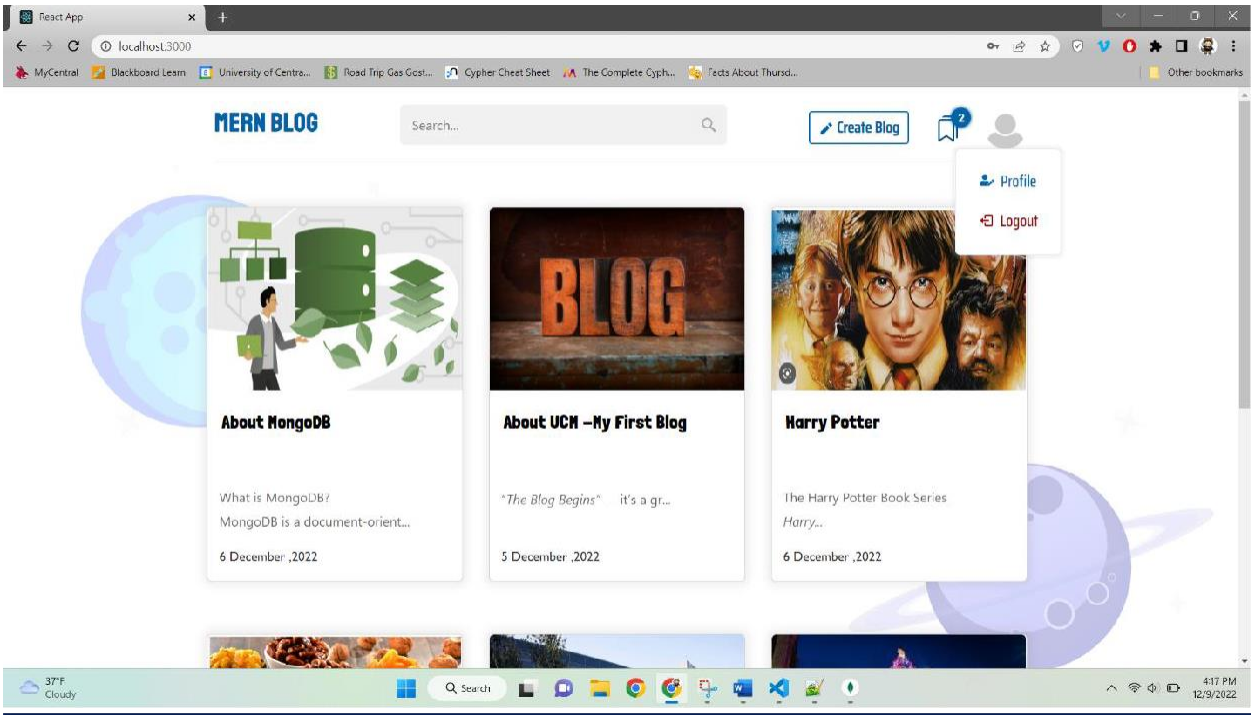
Registration Page



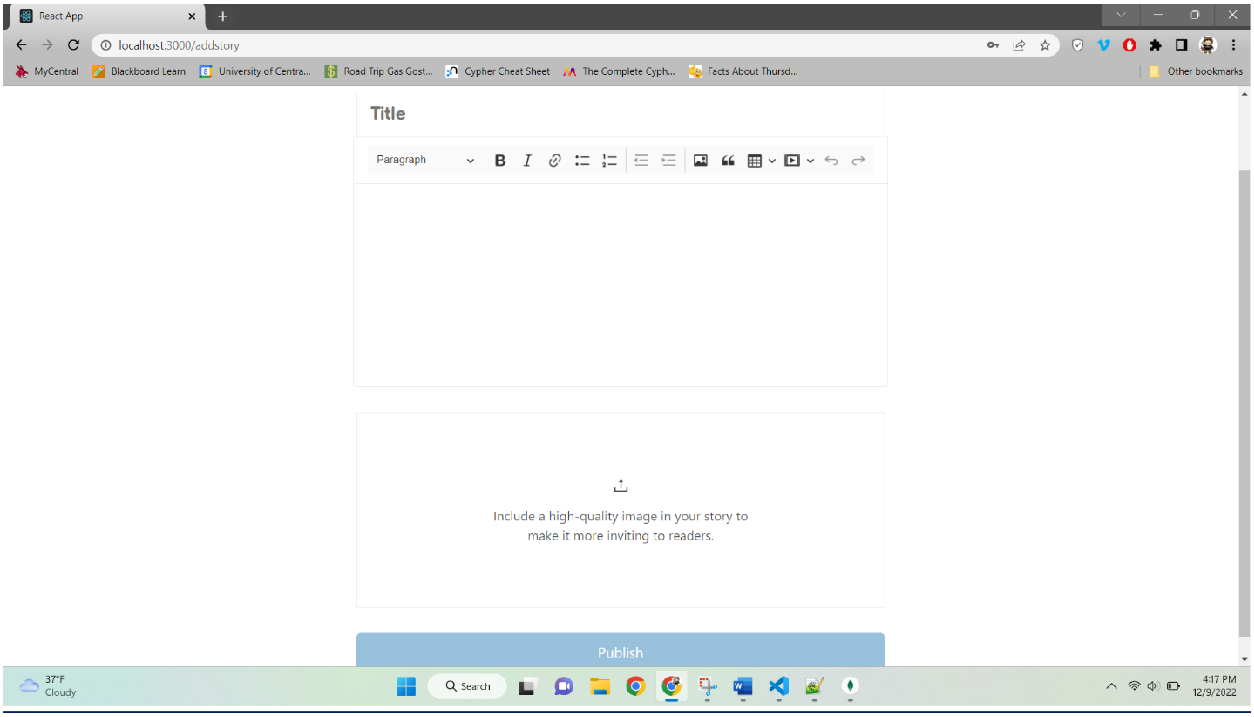
Login Page



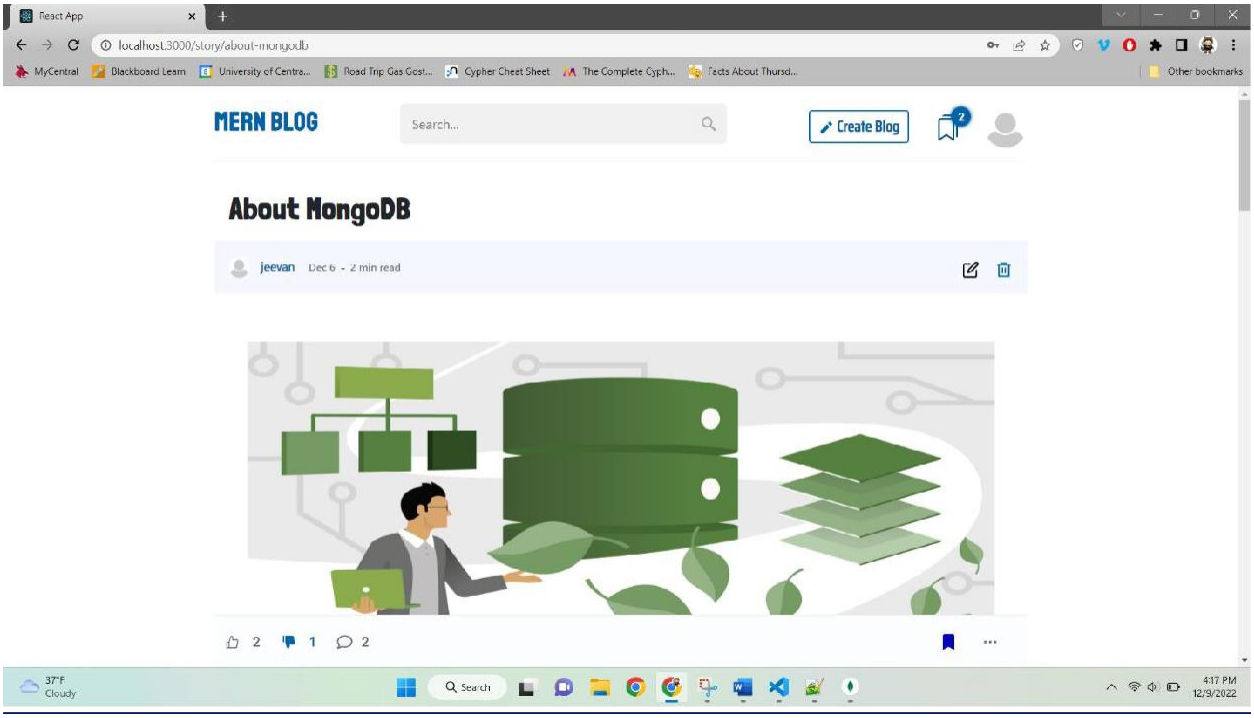
After Login Page



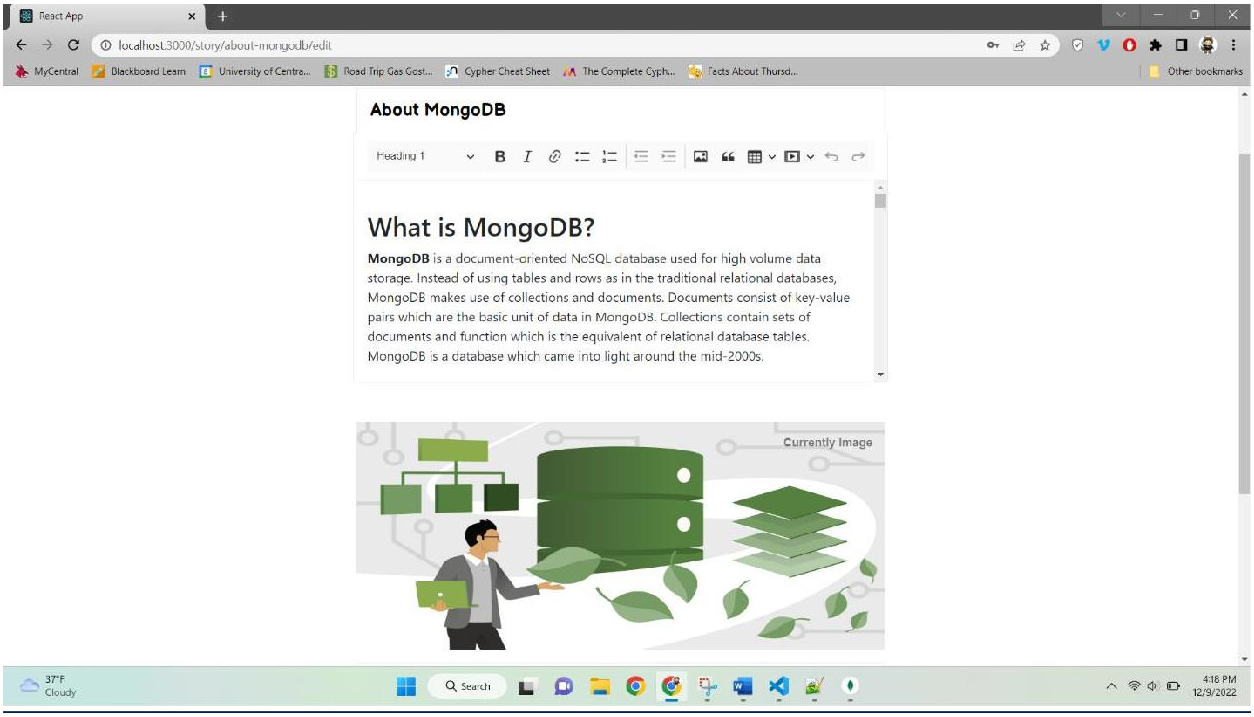
Blog Creation



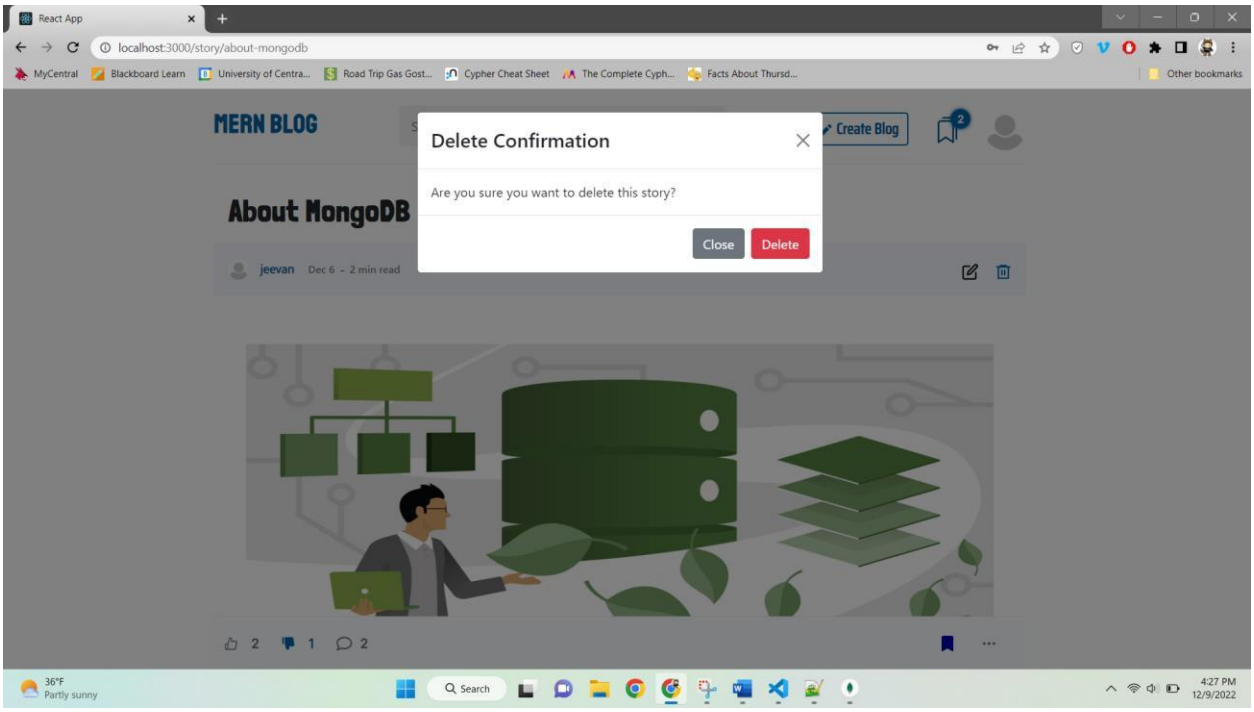
Blog View Page



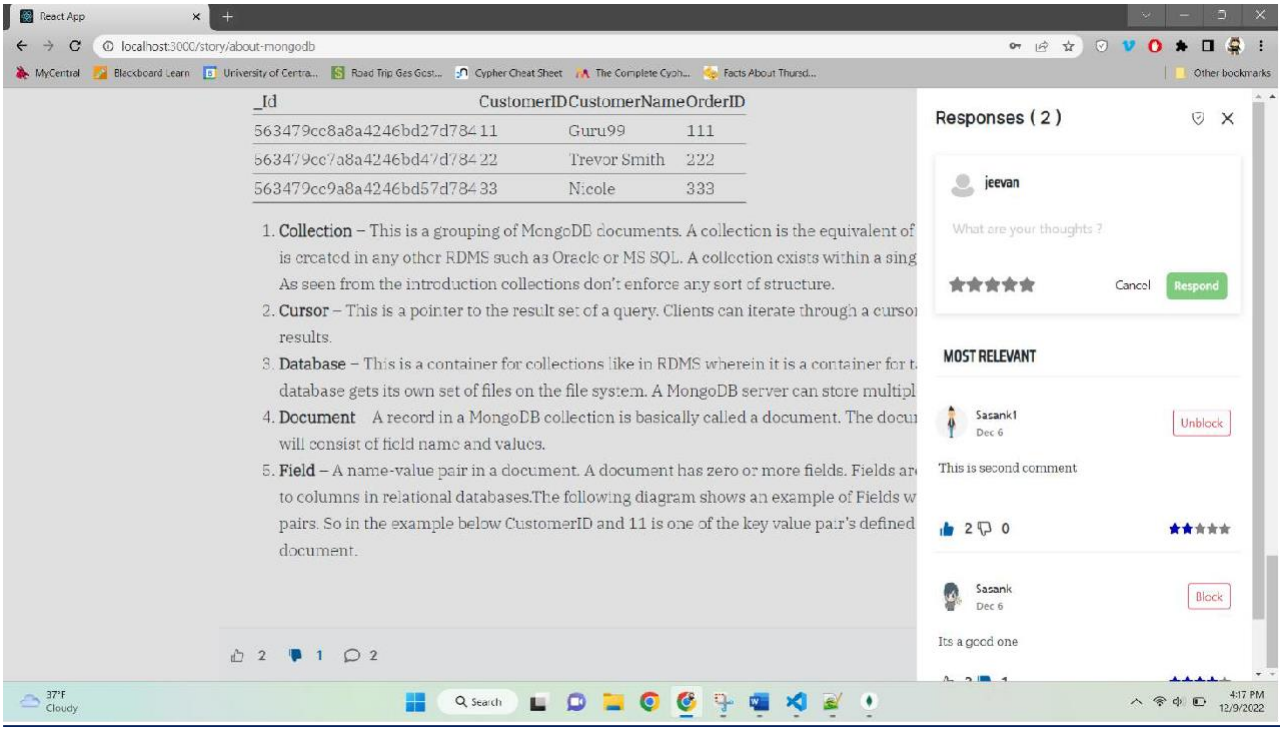
Blog Edit Page:



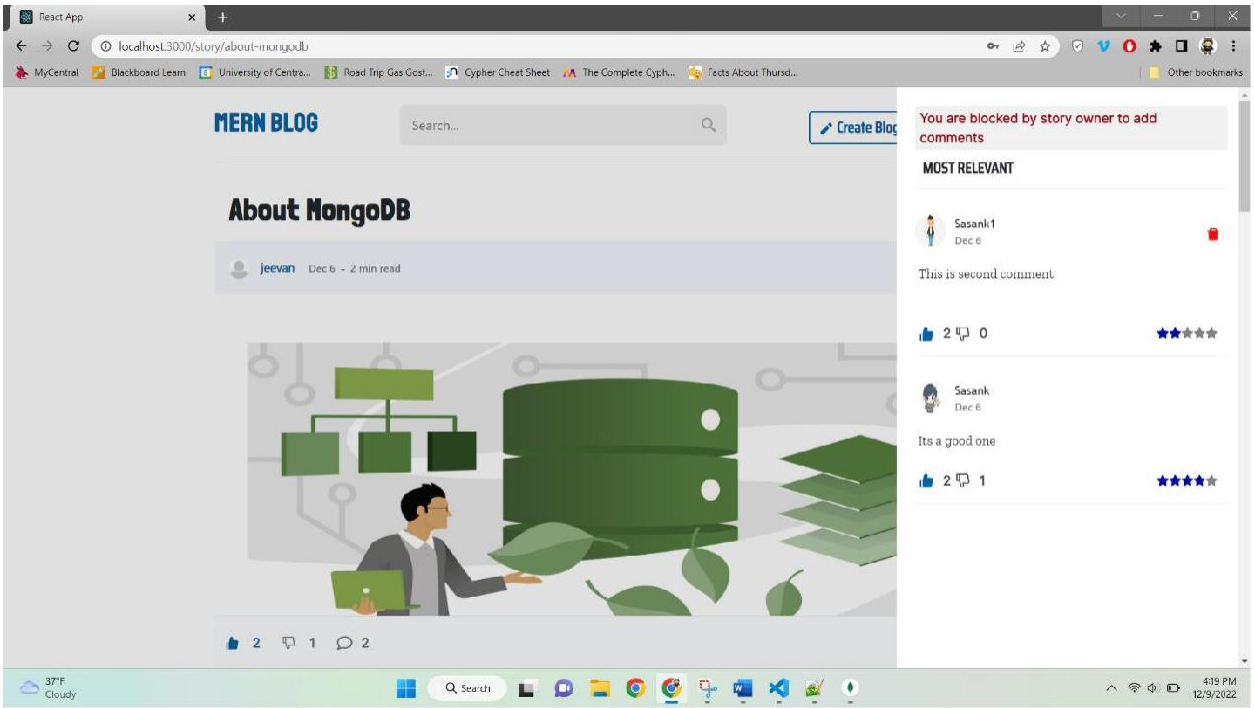
Blog Delete Page:



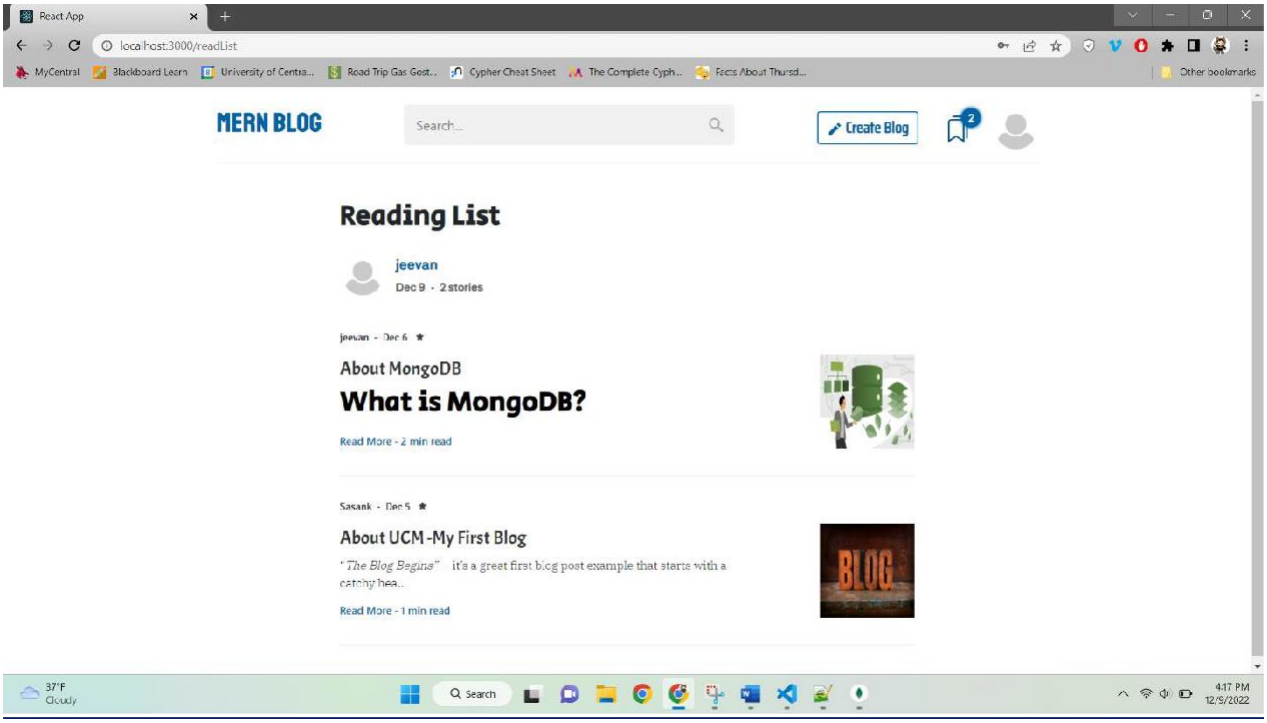
Blog Commenting / Like Dislike / Block Unblocks by Owner:



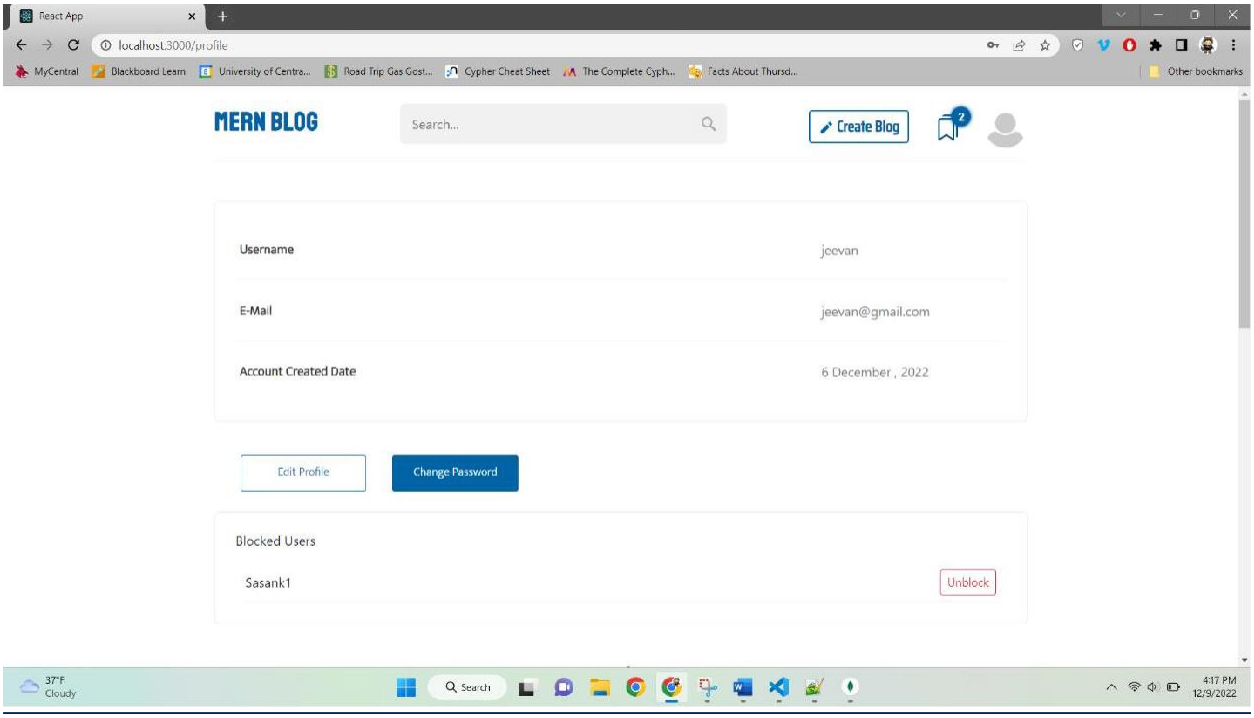
Blog Comment / Like Dislike: Blocked User can't add comments



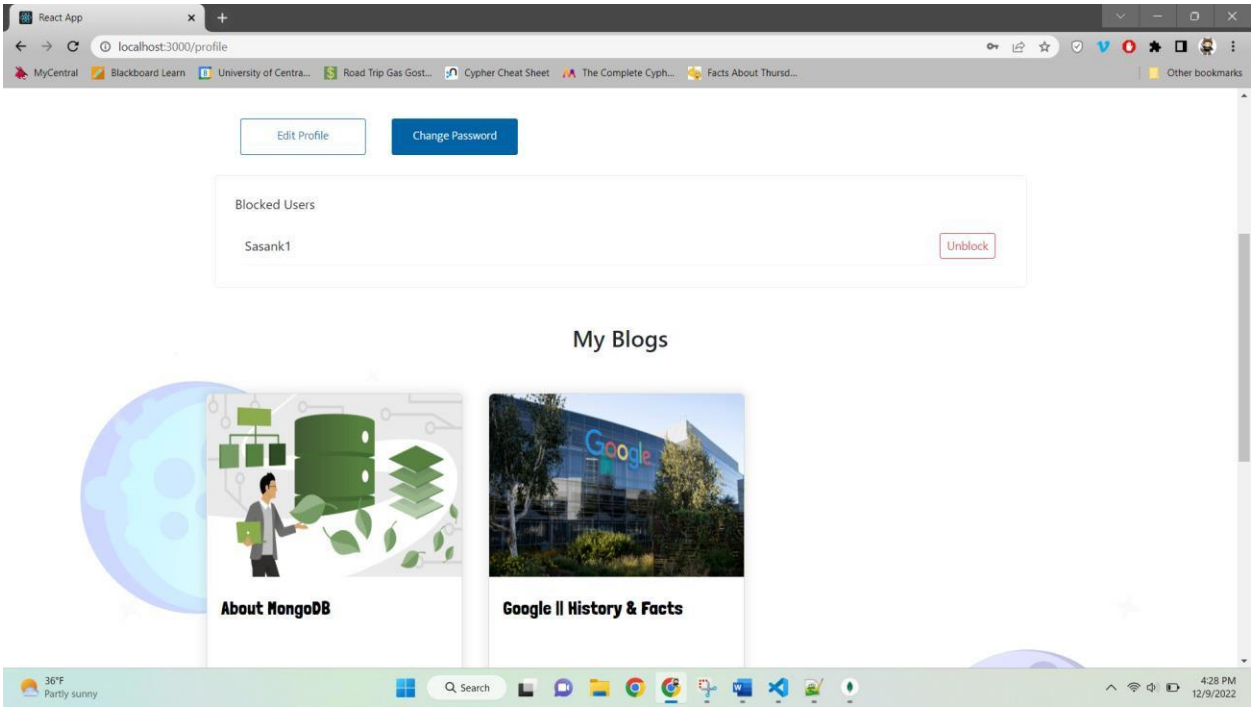
User Read List Page



User Profile Page



User Profile My Blogs:



Source codes

Source code is attached in zip file as it has many files and libraries

NoSQL commands

- **User.js NoSQL Commands files**

```
const asyncErrorWrapper = require("express-async-handler")
const User = require("../Models/user");
const Story = require("../Models/story");
const CustomError = require("../Helpers/error/CustomError");
const { comparePassword ,validateUserInput} =
require("../Helpers/input/inputHelpers");

const profile = asyncErrorWrapper(async (req, res, next) => {

    return res.status(200).json({
        success: true,
        data: req.user
    })

})


const editProfile = asyncErrorWrapper(async (req, res, next) => {

    const { email, username } = req.body

    const user = await User.findByIdAndUpdate(req.user.id, {
        email, username,
        photo : req.savedUserPhoto
    },
        {
            new: true,
            runValidators: true
        })

    return res.status(200).json({
        success: true,
        data: user

    })

})
```

```
const changePassword = asyncErrorWrapper(async (req, res, next) => {

    const  {newPassword , oldPassword}=req.body

    if(!validateUserInput(newPassword,oldPassword)){

        return next(new CustomError("Please check your inputs ", 400))

    }

    const user = await User.findById(req.user.id).select("+password")

    if(!comparePassword(oldPassword , user.password)){
        return next(new CustomError('Old password is incorrect ', 400))
    }

    user.password = newPassword

    await user.save() ;


    return res.status(200).json({
        success: true,
        message : "Change Password  Successfully",
        user : user

    })

})


const addStoryToReadList =asyncErrorWrapper(async(req,res,next) => {

    const {slug} =req.params
    const { activeUser} = req.body  ;

    const story = await Story.findOne({slug})

    const user = await User.findById(activeUser._id)

    if (!user.readList.includes(story.id)){

        user.readList.push(story.id)
```

```
        user.readListLength = user.readList.length
        await user.save() ;
    }

    else {
        const index = user.readList.indexOf(story.id)
        user.readList.splice(index,1)
        user.readListLength = user.readList.length
        await user.save() ;
    }

    const status =user.readList.includes(story.id)

    return res.status(200).json({
        success:true ,
        story  :story ,
        user : user,
        status:status
    })

})

const readListPage =asyncErrorWrapper(async (req,res,next) => {

    const user = await User.findById(req.user.id)
    const readList = []

    for (let index = 0; index < user.readList.length; index++) {

       var story = await Story.findById( user.readList[index]).populate("author")

       readList.push(story)

    }

    return res.status(200).json({
        success :true ,
        data :readList
    })

})

const getBlockedUsers = asyncErrorWrapper(async (req, res, next) => {
    const userId = req.query.user_id
    const user = await User.findById(userId)
```

```
    const blockedUserIds = user.blockedUsers
    const records = await User.find({ '_id': { $in: blockedUserIds } });
    return res.status(200).json({
        success :true ,
        data :records,
    })
})

const unBlockUser = asyncErrorWrapper(async(req, res, next) => {
    const currUserId = req.query.curr_userId
    const userId = req.query.blocked_userId
    const user = await User.findById(currUserId)
    if(userId) {
        const index = user.blockedUsers.indexOf(userId)
        user.blockedUsers.splice(index, 1)
        await user.save()
    }
    const records = await User.find({ '_id': { $in: user.blockedUsers } });
    return res.status(200).json({
        success :true ,
        blockedUsersList :records,
    })
})

module.exports = {
    profile,
    editProfile,
    changePassword,
    addStoryToReadList,
    readListPage,
    getBlockedUsers,
    unBlockUser
}
```

- **Story.js NoSQL Commands file**

```
const asyncErrorWrapper = require("express-async-handler")
const Story = require("../Models/story");
const deleteImageFile = require("../Helpers/Libraries/deleteImageFile");
const { searchHelper, paginateHelper } = require("../Helpers/query/queryHelpers")

const addStory = asyncErrorWrapper(async (req, res, next) => {
```

```
    const { title, content } = req.body

    var wordCount = content.trim().split(/\s+/).length;

    let readtime = Math.round(wordCount / 200);


    try {
        const newStory = await Story.create({
            title,
            content,
            author: req.user._id,
            image: req.savedStoryImage,
            readtime
        })

        return res.status(200).json({
            success: true,
            message: "created blog successfully ",
            data: newStory
        })
    }

    catch (error) {

        deleteImageFile(req)

        return next(error)

    }

})

const getAllStories = asyncErrorWrapper(async (req, res, next) => {

    let query = Story.find();

    query = searchHelper("title", query, req)

    const paginationResult = await paginateHelper(Story, query, req)

    query = paginationResult.query;

    query = query.sort("-likeCount -commentCount -createdAt")
```

```
    const stories = await query

    return res.status(200).json(
        {
            success: true,
            count: stories.length,
            data: stories,
            page: paginationResult.page,
            pages: paginationResult.pages
        })
})

const getMyStories = asyncErrorWrapper(async (req, res, next) => {

    let query = Story.find({author: req.query.author});

    query = query.sort("-likeCount -commentCount -createdAt")

    const stories = await query

    return res.status(200).json(
        {
            success: true,
            count: stories.length,
            data: stories
        })

})



const detailStory = asyncErrorWrapper(async (req, res, next) => {

    const { slug } = req.params;
    const { activeUser } = req.body

    const story = await Story.findOne({
        slug: slug
    }).populate("author likes dislikes")

    const storyLikeUserIds = story.likes.map(json => json.id)
    const storydisLikeUserIds = story.dislikes.map(json => json.id)
    var likeStatus = false
```

```
    var dislikeStatus = false
    if(storyLikeUserIds.includes(activeUser._id)) {
        likeStatus = storyLikeUserIds.includes(activeUser._id)
    }else if(storydisLikeUserIds.includes(activeUser._id)) {
        dislikeStatus = storydisLikeUserIds.includes(activeUser._id)
    }

    return res.status(200).
        json({
            success: true,
            data: story,
            likeStatus: likeStatus,
            dislikeStatus: dislikeStatus,
            storyLikeUserIds: storyLikeUserIds,
            storydisLikeUserIds : storydisLikeUserIds
        })

})

const likeStory = asyncErrorWrapper(async (req, res, next) => {

    const { activeUser } = req.body
    const { slug } = req.params;

    const story = await Story.findOne({
        slug: slug
    }).populate("author likes dislikes")

    const storyLikeUserIds = story.likes.map(json => json._id.toString())
    const storydisLikeUserIds = story.dislikes.map(json => json._id.toString())
    if (!storyLikeUserIds.includes(activeUser._id)) {
        if (storydisLikeUserIds.includes(activeUser._id)) {
            const index = storydisLikeUserIds.indexOf(activeUser._id)
            story.dislikes.splice(index, 1)
            story.dislikeCount = story.dislikes.length
            await story.save();
            var dislikeStatus = false
        }
        story.likes.push(activeUser)
        story.likeCount = story.likes.length
        await story.save();
        var likeStatus = true
    }
    else {
```

```
        const index = storyLikeUserIds.indexOf(activeUser._id)
        story.likes.splice(index, 1)
        story.likeCount = story.likes.length
        var likeStatus = false
        await story.save();
    }

    return res.status(200).
        json({
            success: true,
            data: story,
            likeStatus: likeStatus,
            dislikeStatus: dislikeStatus
        })

})

const dislikeStory = asyncErrorWrapper(async (req, res, next) => {

    const { activeUser } = req.body
    const { slug } = req.params;

    const story = await Story.findOne({
        slug: slug
    }).populate("author likes dislikes")

    const storydisLikeUserIds = story.dislikes.map(json => json._id.toString())

    const storyLikeUserIds = story.likes.map(json => json._id.toString())

    if (!storydisLikeUserIds.includes(activeUser._id)) {
        if (storyLikeUserIds.includes(activeUser._id)) {
            const index = storyLikeUserIds.indexOf(activeUser._id)
            story.likes.splice(index, 1)
            story.likeCount = story.likes.length
            var likeStatus = false
            await story.save();
        }
        story.dislikes.push(activeUser)
        story.dislikeCount = story.dislikes.length
        var dislikeStatus = true
        await story.save();
    }
    else {
        const index = storydisLikeUserIds.indexOf(activeUser._id)
```

```
        story.dislikes.splice(index, 1)
        story.dislikeCount = story.dislikes.length
        var dislikeStatus = false
        await story.save();
    }
    return res.status(200).
        json({
            success: true,
            data: story,
            likeStatus: likeStatus,
            dislikeStatus: dislikeStatus
        })

})



const editStoryPage = asyncErrorWrapper(async (req, res, next) => {
    const { slug } = req.params;

    const story = await Story.findOne({
        slug: slug
    }).populate("author likes dislikes")

    return res.status(200).
        json({
            success: true,
            data: story
        })

})


const editStory = asyncErrorWrapper(async (req, res, next) => {
    const { slug } = req.params;
    const { title, content, image, previousImage } = req.body;

    const story = await Story.findOne({ slug: slug })

    story.title = title;
    story.content = content;
    story.image = req.savedStoryImage;

    if (!req.savedStoryImage) {
        // if the image is not sent
```

```
        story.image = image
    }
    else {
        // if the image sent
        // old image location delete
        deleteImageFile(req, previousImage)

    }

    await story.save();

    return res.status(200).
        json({
            success: true,
            data: story
        })

})

const deleteStory = asyncErrorWrapper(async (req, res, next) => {

    const { slug } = req.params;

    const story = await Story.findOne({ slug: slug })

    deleteImageFile(req, story.image);

    await story.remove()

    return res.status(200).
        json({
            success: true,
            message: "Story delete succesfully "
        })

})


module.exports = {
    addStory,
    getAllStories,
    getMyStories,
    detailStory,
    likeStory,
    dislikeStory,
```

```
    editStoryPage,
    editStory,
    deleteStory
}
```

- **Comments.js NoSQL Commands file**

```
const asyncErrorWrapper = require("express-async-handler")
const Story = require("../Models/story");
const User = require("../Models/user");
const Comment = require("../Models/comment");

const addNewCommentToStory  =asyncErrorWrapper(async(req,res,next)=> {

    const {slug} = req.params

    const {star , content } =req.body

    const story = await Story.findOne({slug :slug })

    const comment = await Comment.create({

        story :story._id ,
        content :content ,
        author : req.user.id ,
        star:star
    })

    story.comments.push(comment._id)

    story.commentCount = story.comments.length

    await story.save();

    return res.status(200).json({
        success :true  ,
        data : comment
    })

})


const getAllCommentByStory = asyncErrorWrapper(async(req, res, next) => {
```

```
    const { slug } = req.params
    const story = await Story.findOne({slug:slug})

    const commmentList =await Comment.find({
        story : story._id
    }).populate({
        path :"author",
        select:"username photo"
    }).sort("-createdAt")

    return res.status(200)
        .json({
            success: true,
            count: story.commentCount,
            data: commmentList,
            storyAuthor: story.author
        })

})

const commentLike = asyncErrorWrapper(async(req, res, next) => {

    const { activeUser} =  req.body
    const { comment_id} =  req.params


    const comment = await Comment.findById(comment_id)

    if (!comment.likes.includes(activeUser._id)) {
        if (comment.dislikes.includes(activeUser._id)) {
            const index = comment.dislikes.indexOf(activeUser._id)
            comment.dislikes.splice(index, 1)
            comment.dislikeCount = comment.dislikes.length
            await comment.save();
            var dislikeStatus = false
        }
        comment.likes.push(activeUser._id)
        comment.likeCount = comment.likes.length ;
        await comment.save()  ;
        var likeStatus = true
    }
    else {
        const index = comment.likes.indexOf(activeUser._id)
        comment.likes.splice(index, 1)
```

```
        comment.likeCount = comment.likes.length
        await comment.save();
        var likeStatus = false
    }

    return res.status(200)
        .json({
            success: true,
            data : comment,
            likeStatus:likeStatus,
            dislikeStatus:dislikeStatus
        })

})

const commentdisLike = asyncErrorWrapper(async(req, res, next) => {

    const { activeUser} =  req.body
    const { comment_id} =  req.params
    const comment = await Comment.findById(comment_id)

    if (!comment.dislikes.includes(activeUser._id)) {
        if (comment.likes.includes(activeUser._id)) {
            const index = comment.likes.indexOf(activeUser._id)
            comment.likes.splice(index, 1)
            comment.likeCount = comment.likes.length
            await comment.save();
            var likeStatus = false
        }
        comment.dislikes.push(activeUser._id)
        comment.dislikeCount = comment.dislikes.length ;
        await comment.save()  ;
        var dislikeStatus = true
    }
    else {

        const index = comment.dislikes.indexOf(activeUser._id)
        comment.dislikes.splice(index,  1)
        comment.dislikeCount = comment.dislikes.length
        var dislikeStatus = false
        await comment.save();
    }
    return res.status(200)
        .json({
            success: true,
```

```
            data  :  comment,
            likeStatus: likeStatus,
            dislikeStatus:dislikeStatus
        })

})


const getCommentLikeStatus = asyncErrorWrapper(async(req, res, next) => {

    const { activeUser} =  req.body
    const { comment_id} =  req.params

    const comment = await Comment.findById(comment_id)
    const likeStatus = comment.likes.includes(activeUser._id)
    const dislikeStatus = comment.dislikes.includes(activeUser._id)

    return res.status(200)
    .json({
        success: true,
        likeStatus:likeStatus,
        dislikeStatus:dislikeStatus
    })

})

const deleteComment = asyncErrorWrapper(async(req, res, next) => {
    const { comment_id} =  req.params
    const storyId = req.query.story_id

    const comment = await Comment.findById(comment_id)
    await comment.remove()

    const story = await Story.findById(storyId)
    const index = story.comments.indexOf(comment_id)
    story.comments.splice(index, 1)
    story.commentCount = story.comments.length
    await story.save();

    return res.status(200).
        json({
            success: true,
            message: "Comment delete succesfully ",
        })
```

```
})

const blockUser = asyncErrorWrapper(async(req, res, next) => {

    const { commentUserId} =  req.params
    const { activeUser} =  req.body
    const user = await User.findById(activeUser._id)

    if (!user.blockedUsers.includes(commentUserId)) {
        user.blockedUsers.push(commentUserId)
        await user.save();
        var blockStatus = 'Unblock'
        return res.status(200).
        json({
            blockStatus: blockStatus,
            message: "User Blocked Succesfully "
        })
    }
    else {
        const index = user.blockedUsers.indexOf(commentUserId)
        user.blockedUsers.splice(index, 1)
        await user.save();
        var blockStatus = 'Block'
        return res.status(200).
        json({
            blockStatus: blockStatus,
            message: "User Unblocked Succesfully "
        })
    }
})

const getUserBlockedToComment = asyncErrorWrapper(async(req, res, next) => {
    const { slug } = req.params
    const { activeUser} =  req.body
    const story = await Story.findOne({slug :slug })
    const storyAuthorDetails = await User.findById(story.author)

    if(storyAuthorDetails.blockedUsers.includes(activeUser._id)) {
        isUserBlocked = true
        return res.status(200).
        json({
            isUserBlocked: isUserBlocked,
        })
    }
    else {
```

```
        isUserBlocked = false
        return res.status(200).
        json({
            isUserBlocked: isUserBlocked,
        })
    }

})

module.exports ={
    addNewCommentToStory,
    getAllCommentByStory,
    commentLike,
    commentdisLike,
    getCommentLikeStatus,
    deleteComment,
    blockUser,
    getUserBlockedToComment
}
```

- **Auth.js for user authentication**

```
const asyncErrorWrapper = require("express-async-handler")

const User = require("../Models/user");
const CustomError = require("../Helpers/error/CustomError");
const { sendToken } = require("../Helpers/auth/tokenHelpers");
const sendEmail = require("../Helpers/Libraries/sendEmail");
const { validateUserInput,comparePassword } =
require("../Helpers/input/inputHelpers");

const getPrivateData = asyncErrorWrapper((req,res,next) =>{

    return res.status(200).json({
        success:true ,
        message : "You got access to the private data in this route ",
        user : req.user

    })

})

const register = asyncErrorWrapper (async  (req,res,next) => {
```

```
    const { username,email , password} = req.body  ;

    const newUser = await User.create({
        username,
        email,
        password
    })

    sendToken(newUser ,201,res)


})

const login  = asyncErrorWrapper (async(req,res,next) => {

    const {email,password} = req.body

    if(!validateUserInput(email,password)) {

        return next(new CustomError("Please check your inputs",400))
    }

    const user = await User.findOne({email}).select("+password")

    if(!user) {

        return next(new CustomError("Invalid credentials",404))
    }

    if(!comparePassword(password,user.password)){
        return next(new CustomError("Please chech your credentails",404))
    }

    sendToken(user ,200,res)  ;

})




const forgotpassword  = asyncErrorWrapper( async (req,res,next) => {
    const {URI,EMAIL_USERNAME} = process.env ;

    const resetEmail = req.body.email  ;
```

```
    console.log(EMAIL_USERNAME)
    const user = await User.findOne({email : resetEmail})

    if(!user ) {
         return next(new CustomError( "There is no user with that email",400))
    }

    const resetPasswordToken = user.getResetPasswordTokenFromUser();

    await user.save()  ;

    const resetPasswordUrl =
`${URI}/resetpassword?resetPasswordToken=${resetPasswordToken}`

    const emailTemplate = `
    <h3 style="color : red "> Reset Your Password </h3>
    <p> This <a href=${resetPasswordUrl}
     target='_blank'  >Link </a> will expire in 1 hours </p>
    `;

    try {

        sendEmail({
            from: EMAIL_USERNAME,
            to: resetEmail,
            subject: " ✓ Reset Your Password  ✓",
            html: emailTemplate
        })

        return res.status(200)
        .json({
            success: true,
            message: "Email Send"
        })

    }

    catch(error ) {

        user.resetPasswordToken = undefined ;
        user.resetPasswordExpire = undefined  ;

        await user.save();

        return next(new CustomError('Email could not be send ', 500))
```

```
    }



})

const resetpassword  =asyncErrorWrapper(  async (req,res,next) => {

    const newPassword = req.body.newPassword || req.body.password

    const {resetPasswordToken} = req.query

    if(!resetPasswordToken) {

        return next(new CustomError("Please provide a valid token ",400))
    }

    const user = await User.findOne({

        resetPasswordToken :resetPasswordToken ,
        resetPasswordExpire : { $gt: Date.now() }

    })

    if(!user) {
        return next(new CustomError("Invalid token or Session Expired" ,400))
    }


    user.password = newPassword ;

    user.resetPasswordToken = undefined
    user.resetPasswordExpire = undefined

    await user.save() ;

    return res.status(200).json({
        success :true ,
        message : "Reset Password access successfull"
    })

})


module.exports ={
```

```
    register,
    login,
    resetpassword,
    forgotpassword,
    getPrivateData
}
```

---------- THE END ----------