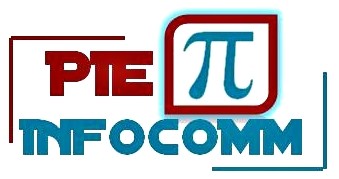
****

**SESSION – 2024-2025**

**Project Report**

**On**

**Create X (DIY Platform) For Sharing and Discovering Do It Yourself Ideas**

*Submitted in fulfillment of requirement of the internship of*

**Full Stack Web Development using MERN Stack**

*To*

**PIE INFOCOMM PVT. LTD**.

*by*

**Kushagra Chaturvedi**

*Under the guidance of*

**Ms. Aishwarya Saxena**

**(Sr. Software Developer)**

**TABLE OF CONTENTS**

1. Introduction
2. Objective
3. Background
4. Hardware and Software Requirement
5. ERD and DFD
6. Form Layout
7. Database Schema
8. Coding
9. Output Screenshot
10. Future Scope
11. Conclusion

References and Bibliography

**INTRODUCTION**

CreateX is an innovative web platform built on the MERN stack, combining MongoDB, Express.js, React.js, and Node.js. This platform is designed to cater to DIY enthusiasts by providing a space where users can share, discover, and engage with a wide range of DIY project ideas and tutorials. The goal of CreateX is to foster a community where creativity and knowledge about DIY projects can be exchanged freely. The platform's user-friendly interface ensures that users can easily navigate through various project categories, submit their own projects, and interact with other members of the community. By leveraging modern web technologies, CreateX aims to deliver a seamless and interactive experience that meets the needs of a diverse audience interested in DIY projects. The platform also supports essential features such as user authentication, project submission, and community feedback, making it a comprehensive tool for both beginners and experienced DIY enthusiasts.

**OBJECTIVE**

The primary objective of CreateX is to establish a comprehensive, user-centric platform dedicated to sharing and discovering DIY (Do It Yourself) project ideas and tutorials. This objective is underpinned by several key goals and functionalities designed to enhance the user experience and foster a vibrant DIY community.

**1. User-Friendly Interface:**

CreateX aims to provide an intuitive and engaging user interface that simplifies the process of discovering, sharing, and managing DIY projects. The platform is designed with user experience at its core, featuring a clean, organized layout that allows users to easily navigate through various sections, such as project categories, search functions, and user profiles. By prioritizing ease of use, CreateX ensures that users, regardless of their technical expertise, can interact with the platform effectively.

**2. Project Submission and Management:**

One of the fundamental objectives of CreateX is to enable users to submit their own DIY projects seamlessly. The platform provides a structured submission form where users can enter detailed information about their projects, including titles, descriptions, tags, and multimedia files. This feature allows users to showcase their creativity and expertise, contributing valuable content to the platform. Additionally, users have the ability to manage their submitted projects, including editing and deleting entries, which provides them with control over their contributions.

**3. Community Engagement:**

CreateX is designed to foster a sense of community among DIY enthusiasts. To achieve this, the platform includes features that facilitate interaction between users. Comments and feedback mechanisms allow users to engage with project creators, ask questions, and provide suggestions. By enabling these interactions, CreateX promotes a collaborative environment where users can share knowledge, offer support, and build connections with others who share similar interests.

**4. Discoverability and Search Functionality:**

Another key objective is to enhance the discoverability of DIY projects through robust search and filtering options. CreateX includes advanced search features that allow users to find projects based on specific criteria such as keywords, categories, difficulty levels, and tags. This functionality ensures that users can quickly locate projects that match their interests and skill levels, making it easier for them to find relevant content and inspiration.

**5. User Authentication and Security:**

User authentication is a critical component of CreateX, ensuring that the platform is secure and that user data is protected. The objective is to implement a reliable authentication system that allows users to create accounts, log in, and manage their profiles securely. By utilizing modern authentication techniques, such as password hashing and session management, CreateX aims to safeguard user information and provide a secure environment for interaction and content sharing.

**6. Responsive Design:**

CreateX is committed to delivering a seamless experience across various devices and screen sizes. The platform’s responsive design ensures that it functions effectively on desktops, tablets, and smartphones. By optimizing the user interface for different devices, CreateX aims to accommodate users’ preferences and browsing habits, making it accessible to a broader audience and enhancing overall user satisfaction.

**7. Future Scalability and Expansion:**

An important objective for CreateX is to ensure that the platform can scale and adapt to future needs. This involves designing the system architecture to handle increased traffic, additional features, and expanding user base. Future scalability considerations include integrating advanced functionalities, such as multimedia support and personalized recommendations, to enhance the platform’s capabilities and relevance over time.

**8. Encouraging Innovation and Creativity:**

Ultimately, CreateX aims to be a catalyst for innovation and creativity within the DIY community. By providing a platform that supports user-generated content and community interaction, CreateX encourages individuals to explore new ideas, share their knowledge, and contribute to a collective pool of DIY resources. This objective aligns with the broader goal of fostering a culture of creativity and self-sufficiency.

In summary, the objectives of CreateX revolve around creating a user-friendly, engaging, and secure platform for sharing and discovering DIY projects. By focusing on key areas such as user experience, community engagement, search functionality, and content quality, CreateX aims to build a valuable resource for DIY enthusiasts and promote a collaborative and innovative environment.

**BACKGROUND**

The concept of DIY projects has seen a remarkable rise in popularity in recent years, driven by an increasing interest in self-sufficiency, creativity, and cost-effectiveness. As more individuals look to personalize their spaces, develop new skills, and save money, the DIY movement has grown into a vibrant community. DIY platforms serve as a critical resource in this trend, offering users a vast array of projects that span various categories such as home improvement, gardening, arts and crafts, electronics, and more. These platforms enable users to not only find inspiration and ideas but also access detailed tutorials and guides to help them complete their projects successfully.

Traditional DIY platforms, however, often face limitations in terms of user engagement and modern features. Many existing platforms struggle with outdated interfaces, lack of interactive elements, and limited functionality for user interaction and content management. This is where leveraging the MERN stack (MongoDB, Express.js, React.js, Node.js) offers a significant advantage. The MERN stack provides a comprehensive suite of technologies that collectively enhance the functionality, responsiveness, and overall user experience of the platform.

* **MongoDB**

MongoDB's flexible, schema-less data structure is ideal for handling the diverse and complex data associated with DIY projects. Its ability to store data in JSON-like documents makes it easy to manage various project details, user profiles, comments, and other related information. MongoDB’s scalability and performance are crucial for ensuring that the platform can handle a growing number of users and an expanding database of DIY projects.

* **Express.js and Node.js**

On the backend, Express.js and Node.js offer robust and efficient handling of server-side operations. Express.js simplifies the process of creating and managing routes, handling HTTP requests, and integrating with databases, while Node.js provides a powerful runtime environment for executing server-side code. Together, they form a solid foundation for building a fast, secure, and scalable backend. This combination enables seamless user authentication, real-time data updates, and efficient handling of CRUD (Create, Read, Update, Delete) operations for DIY projects.

* **React.js**

React.js is essential for developing a dynamic and responsive frontend. Its component-based architecture allows for the creation of reusable UI components, which not only enhances development efficiency but also ensures a consistent user interface across the platform. React’s virtual DOM significantly improves the performance of dynamic web applications, allowing users to interact with the platform smoothly and efficiently. Features like real-time project updates, interactive commenting, and responsive design are easily implemented with React.js, ensuring a rich and engaging user experience.

By integrating these modern web development technologies, the DIY platform can overcome the limitations of traditional platforms and offer a more interactive, user-friendly, and scalable solution. This approach not only meets the current needs of DIY enthusiasts but also positions the platform to adapt and grow with future trends and user demands.

**HARDWARE AND SOFTWARE REQUIREMENTS**

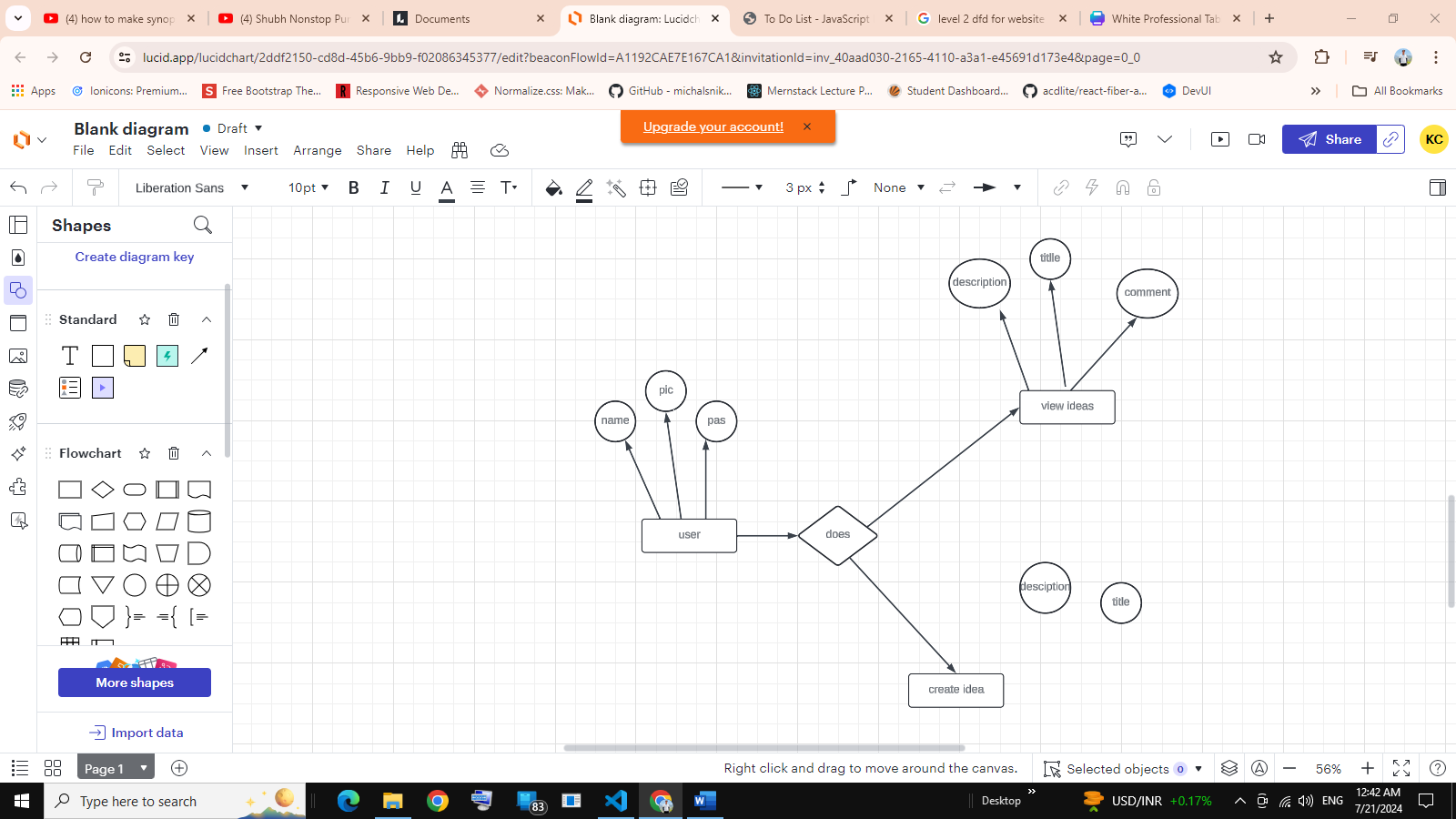
* **Hardware Requirements:**

|  |  |
| --- | --- |
| Minimum Ram | 8 GB DDR4 |
| Processor | i5 or equivalent |
| System Type | 64-bit Operating System |
| Windows Version | Window 8 or above |

* **Software Requirements:**

|  |  |
| --- | --- |
| Frontend Tools | HTML5 , CSS3, JavaScript |
| Frontend Scripting Language | ReactJS |
| Backend Scripting Language | NodeJs , ExpressJs |
| Database | MongoDB Atlas |
| Code Editor | VS CODE |

**ERD AND DFD**

Entity-Relationship Diagram (ERD):

**Data Flow Diagram (DFD):**

1. LEVEL – 0

A screenshot of a computer

Description automatically generated

1. LEVEL – 1

A screenshot of a computer screen

Description automatically generated

1. LEVEL–2

Check Roles of Access







Manage Blog Details

Manage Blog Details

Login

Admin

Manage Modules

Check Credentials

Create Account

Manage Blogs

Manage Blogs

Id , Password Created

**FORM LAYOUTS**

The form layouts define the user interface for various functionalities within the application. Key forms include:

**- Registration Form:** Captures user information such as username, email, and password.

LOGO

SIGNUP

User Name

Password

SIGNUP

Already have an account ? Login

**- Login Form:** Authenticates users with their email and password.

LOGIN

User Name 

Password

LOGIN

New here? Register

LOGO

New Here ? Register

Login

**- Idea Submission Form:** Allows users to submit new DIY projects, including fields for title, description, and image upload.

PROFILE

ALL BLOGS MY BLOGS CREATE

LOGO

**Create Your Idea**

Title

Description

Choose File

Upload Photo

Create Now

**- Comment Form:** Enables users to add comments to projects.

PROFILE

LOGO

ALL BLOGS MY BLOGS CREATE

TITLE

DESCRIPTION

IDEA PHOTO

XYZ

XYZ

COMMENTS

ADD COMMENT

Your comment……..

Add a comment

**DATABASE SCHEMA**

The database schema outlines the structure of the data stored in MongoDB. Key collections include:

**- Users:** Stores user information such as username, email, and hashed password.

const mongoose=require('mongoose')

const UserSchema=new mongoose.Schema({

    username:{

        type:String,

        required:true,

        unique:true

    },

    email:{

        type:String,

        required:true,

        unique:true

    },

    password:{

        type:String,

        required:true;

    }

},{timestamps:true})

module.exports=mongoose.model("User",UserSchema)

**- Ideas** Contains project details including title, description, image URL, and references to the user who created it.

const mongoose=require('mongoose')

const PostSchema=new mongoose.Schema({

    title:{

        type:String,

        required:true,

        unique:true

    },

    desc:{

        type:String,

        required:true,

        unique:true

    },

    photo:{

        type:String,

        required:false,

    },

    username:{

        type:String,

        required:true,

    },

    userId:{

        type:String,

        required:true,

    },

    categories:{

        type:Array,

    },

},{timestamps:true})

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

**- Comments:** Holds comments related to projects, including the content of the comment and references to the user and project.

const mongoose=require('mongoose')

const CommentSchema=new mongoose.Schema({

    comment:{

        type:String,

        required:true,

    },

    author:{

        type:String,

        required:true,

    },

    postId:{

        type:String,

        required:true,

    },

    userId:{

        type:String,

        required:true

    }

},{timestamps:true})

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

**CODING**

The coding section would include code snippets and explanations for key functionalities:

* **Backend (Node.js and Express.js):**
  + **User Authentication:** Using User.js for handling user sessions.

const express=require('express')

const router=express.Router()

const User=require('../models/User')

const bcrypt=require('bcrypt')

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

const Comment=require('../models/Comment')

const verifyToken = require('../verifyToken')

//UPDATE

router.put("/:id",verifyToken,async (req,res)=>{

    try{

        if(req.body.password){

            const salt=await bcrypt.genSalt(10)

            req.body.password=await bcrypt.hashSync(req.body.password,salt)

        }

        const updatedUser=await User.findByIdAndUpdate(req.params.id,{$set:req.body},{new:true})

        res.status(200).json(updatedUser)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//DELETE

router.delete("/:id",verifyToken,async (req,res)=>{

    try{

        await User.findByIdAndDelete(req.params.id)

        await Post.deleteMany({userId:req.params.id})

        await Comment.deleteMany({userId:req.params.id})

        res.status(200).json("User has been deleted!")

    }

    catch(err){

        res.status(500).json(err)

    }

})

//GET USER

router.get("/:id",async (req,res)=>{

    try{

        const user=await User.findById(req.params.id)

        const {password,...info}=user.\_doc

        res.status(200).json(info)

    }

    catch(err){

        res.status(500).json(err)

    }

})

module.exports=router

* + **API Routes:** RESTful API endpoints for CRUD operations.

const express=require('express')

const router=express.Router()

const User=require('../models/User')

const bcrypt=require('bcrypt')

const jwt=require('jsonwebtoken')

//REGISTER

router.post("/register",async(req,res)=>{

    try{

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

        const salt=await bcrypt.genSalt(10)

        const hashedPassword=await bcrypt.hashSync(password,salt)

        const newUser=new User({username,email,password:hashedPassword})

        const savedUser=await newUser.save()

        res.status(200).json(savedUser)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//LOGIN

router.post("/login",async (req,res)=>{

    try{

        const user=await User.findOne({email:req.body.email})

        if(!user){

            return res.status(404).json("User not found!")

        }

        const match=await bcrypt.compare(req.body.password,user.password)

        if(!match){

            return res.status(401).json("Wrong credentials!")

        }

        const token=jwt.sign({\_id:user.\_id,username:user.username,email:user.email},process.env.SECRET,{expiresIn:"3d"})

        const {password,...info}=user.\_doc

        res.cookie("token",token).status(200).json(info)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//LOGOUT

router.get("/logout",async (req,res)=>{

    try{

        res.clearCookie("token",{sameSite:"none",secure:true}).status(200).send("User logged out successfully!")

    }

    catch(err){

        res.status(500).json(err)

    }

})

//REFETCH USER

router.get("/refetch", (req,res)=>{

    const token=req.cookies.token

    jwt.verify(token,process.env.SECRET,{},async (err,data)=>{

        if(err){

            return res.status(404).json(err)

        }

        res.status(200).json(data)

    })

})

module.exports=router

* **Frontend (React.js):**
  + **Components:** HomePost.js component

/\* eslint-disable react/prop-types \*/

import {IF} from '../url'

const HomePosts = ({post}) => {

  return (

    <div className="w-full flex mt-8 space-x-4">

    {/\* left \*/}

    <div className="w-[35%] h-[200px] flex justify-center items-center">

    <img src={IF+post.photo} alt="" className="h-full w-full object-cover"/>

    </div>

    {/\* right \*/}

    <div className="flex flex-col w-[65%]">

      <h1 className="text-xl font-bold md:mb-2 mb-1 md:text-2xl">

      {post.title}

      </h1>

      <div className="flex mb-2 text-sm font-semibold text-gray-500 items-center justify-between md:mb-4">

       <p>@{post.username}</p>

       <div className="flex space-x-2 text-sm">

       <p>{new Date(post.updatedAt).toString().slice(0,15)}</p>

       <p>{new Date(post.updatedAt).toString().slice(16,24)}</p>

       </div>

      </div>

      <p className="text-sm md:text-lg">{post.desc.slice(0,200)+" ...Read more"}</p>

    </div>

    </div>

  )

}

export default HomePosts

* + **State Management:** Using React Context API for managing application state.

import axios from "axios";

import { createContext, useState, useEffect } from "react";

import { URL } from "../url"; // Ensure this path is correct

export const UserContext = createContext({})

export function UserContextProvider({children}) {

    const [user, setUser] = useState(null)

    useEffect(() => {

        getUser()

    }, [])

    const getUser = async () => {

        try {

            const res = await axios.get(URL+"/api/auth/refetch",{withCredentials:true })

            setUser(res.data)

        } catch (err) {

            console.log(err)

        }

    }

    return (

        <UserContext.Provider value={{user,setUser }}>

            {children}

        </UserContext.Provider>)

    }

* **Database (MongoDB):**
  + **Schema Design:** Defining collections and relationships.

const mongoose=require('mongoose')

const UserSchema=new mongoose.Schema({

    username:{

        type:String,

        required:true,

        unique:true

    },

    email:{

        type:String,

        required:true,

        unique:true

    },

    password:{

        type:String,

        required:true,

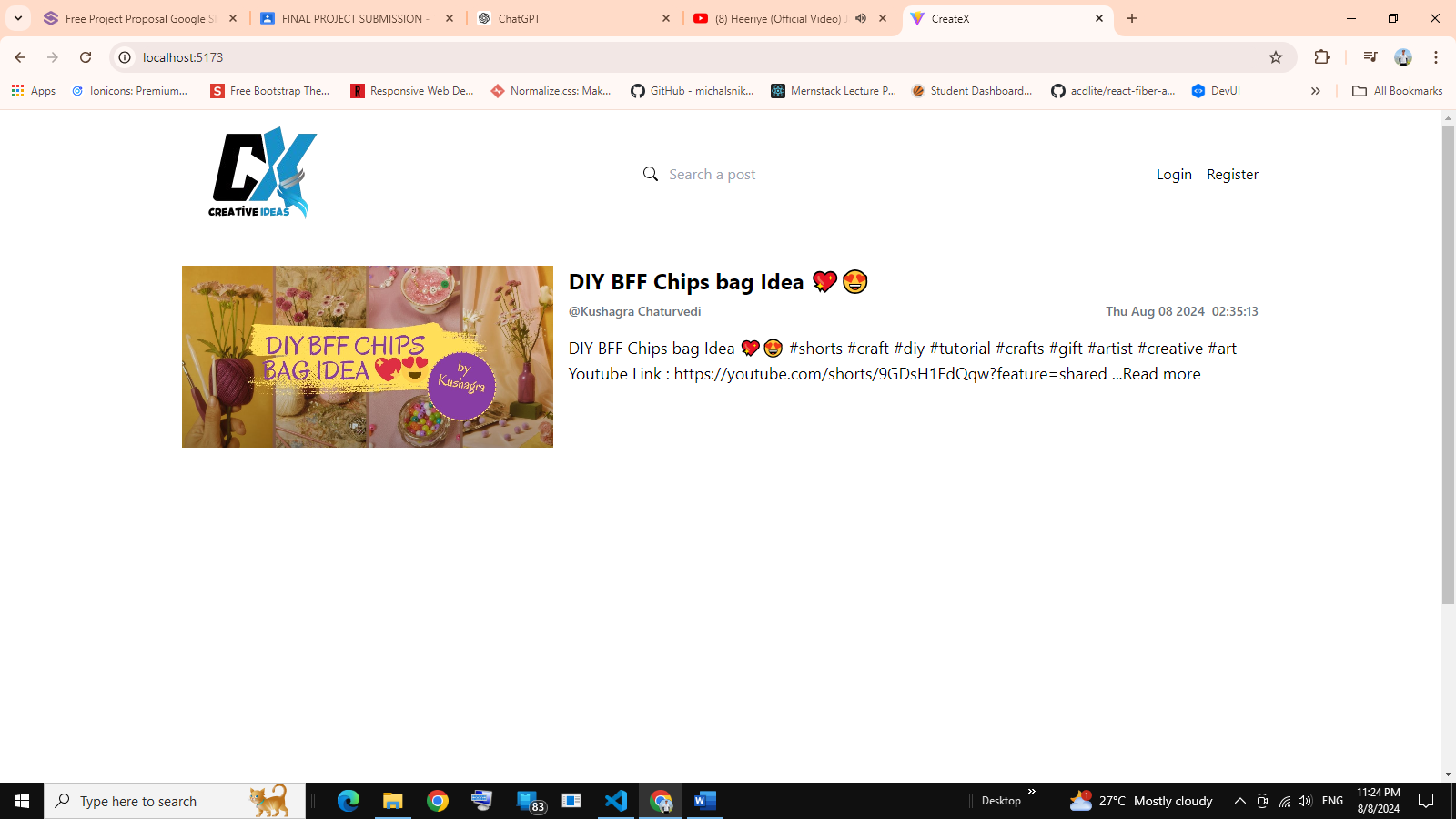
    }

},{timestamps:true})

module.exports=mongoose.model("User",UserSchema)

**OUTPUT SCREENSHOT**

1. **Fig 1 : Home Page with a Private Route for non-registered users.**

****

1. **A screenshot of a computer

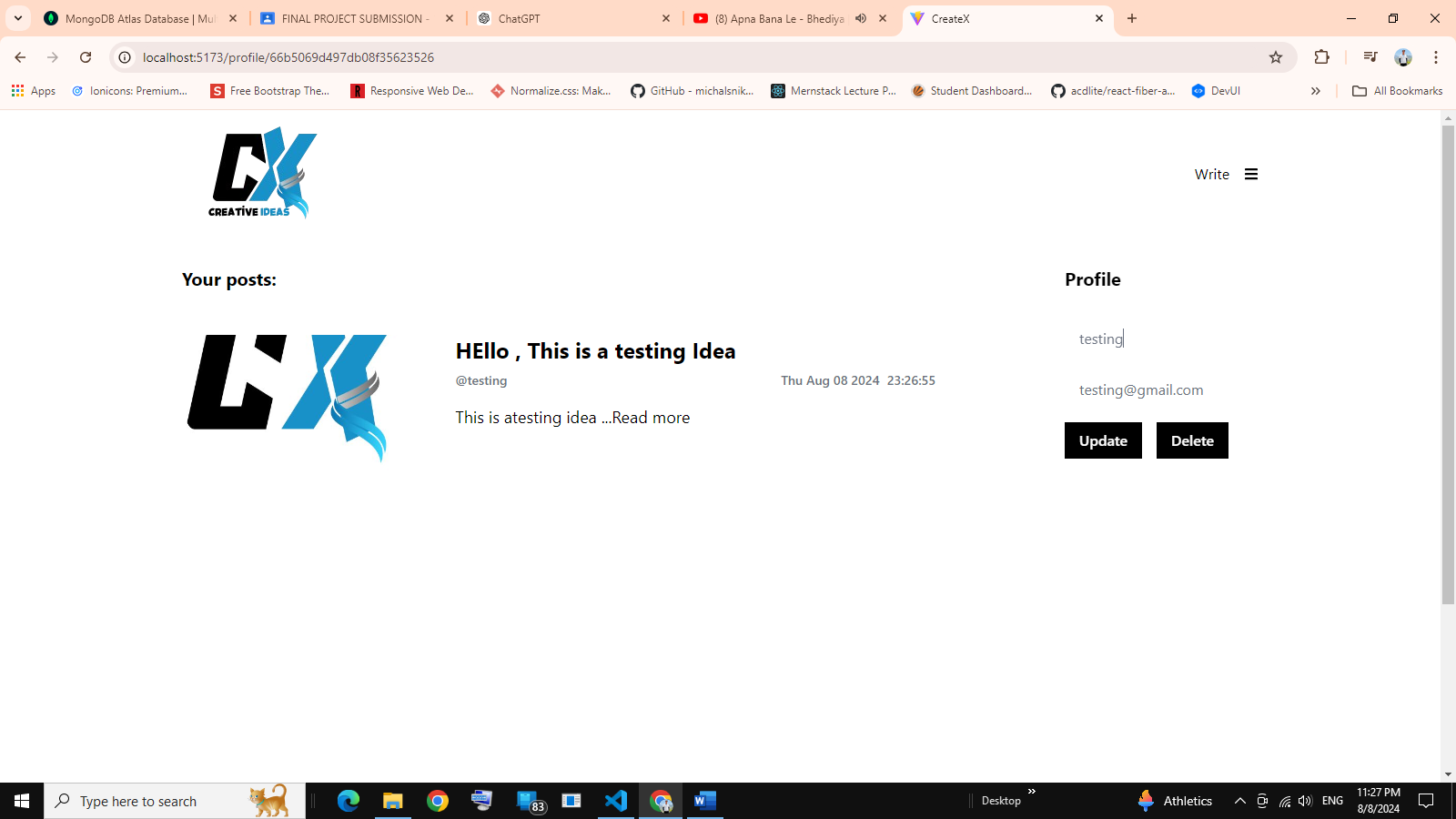
   Description automatically generatedFig 2 : Register Page with features including Show-Hide Password , a redirect link to a web page which is a developed for generating a random password , includes toast notification for better user experience**
2. **A screenshot of a computer

   Description automatically generatedFig 3 : Create Page with features including multer for uploading the poster /image/banner for the idea , includes adding of categories also.**
3. **A screenshot of a computer

   Description automatically generatedFig 4 : Redirection to Home Page after creating the idea.**
4. **Fig 5 : Integrated Comment adding feature on the DIY Ideas**

**A screenshot of a computer

Description automatically generated**

1. **Fig 6 : Integrated User Profile Page for accessing their login credentials and ideas.**
2. **A screenshot of a computer

   Description automatically generatedFig 7 : Dedicated My Ideas page for accessing the ideas only.**
3. **A screenshot of a computer

   Description automatically generatedFig 7 : Logout Feature for Logging Out of the CreateX Dashboard**

**FUTURE SCOPE**

The future scope of the DIY project platform is vast and promising, with numerous enhancements and expansions planned to further enrich the user experience and ensure the platform remains a vibrant and sustainable community for DIY enthusiasts. Key areas of development include:

1. **User Profiles**

Expanding user profiles is a critical next step for enhancing user engagement and personalization. By allowing users to create comprehensive profiles, the platform can offer features such as:

- **Portfolio of Projects:** Users can showcase a portfolio of their completed and ongoing projects, which not only serves as a personal achievement gallery but also inspires other users. This feature can include detailed project descriptions, images, and even video tutorials.

- **Follower/Following Features:** Implementing follower/following capabilities will enable users to connect with each other more meaningfully. Users can follow their favorite creators, receive updates on new projects, and build a network of like-minded DIY enthusiasts. This social aspect will foster a sense of community and encourage regular platform engagement.

1. **Mobile Application**

Developing a mobile application for iOS and Android is essential to cater to the growing number of users who prefer accessing content on their mobile devices. A mobile app will provide the following benefits:

- **On-the-Go Access:** Users can browse, create, and manage their DIY projects from anywhere, making the platform more convenient and accessible.

- **Enhanced User Experience:** A native mobile app can offer a more tailored and responsive user experience, utilizing device-specific features such as push notifications, camera integration for project documentation, and offline access to project guides.

- **Broader Reach:** A mobile app will help the platform reach a wider audience, including those who primarily use smartphones and tablets for their online activities.

1. **Integration with Social Media**

Integrating the platform with popular social media networks will significantly increase its visibility and user engagement. Key integration features include:

**- Easy Sharing:** Users can share their projects directly to social media platforms like Facebook, Instagram, Pinterest, and Twitter. This will not only promote individual projects but also drive traffic back to the platform.

**- Social Media Logins:** Allowing users to sign in with their social media accounts will streamline the registration process and encourage more sign-ups.

**- Engagement Metrics:** Social media integration can provide valuable insights into how projects are being received and shared across different platforms, helping to refine content strategies and user engagement tactics.

1. **Monetization**

Introducing monetization strategies is crucial for sustaining the platform in the long term. Potential revenue models include:

**- Premium Features:** Offering premium features such as ad-free browsing, exclusive project tutorials, and advanced project management tools can generate subscription-based revenue. Premium users could also access features like direct messaging with project creators and personalized project recommendations.

**- Ad-Based Revenue:** Implementing targeted advertising can provide a steady revenue stream without compromising the user experience. Ads related to DIY supplies, tools, and relevant services can be integrated seamlessly into the platform.

**- Affiliate Marketing:** Partnering with DIY supply vendors and including affiliate links in project guides can generate additional income. When users purchase supplies through these links, the platform earns a commission.

**- Sponsored Content**: Collaborating with brands for sponsored projects and tutorials can offer another revenue stream. Brands can gain exposure to a targeted audience, while users benefit from high-quality content and potential discounts on supplies.

By pursuing these future developments, the DIY project platform can evolve into a comprehensive and thriving ecosystem for DIY enthusiasts. The enhancements in user profiles, the addition of a mobile application, social media integration, and strategic monetization efforts will not only improve the user experience but also ensure the platform's long-term sustainability and growth.

**CONCLUSION**

The DIY project platform represents a significant advancement in fostering an engaging and interactive community for DIY enthusiasts. By leveraging the power of the MERN stack, the platform is designed to be scalable, efficient, and highly responsive, offering a seamless user experience that can be continuously enhanced to meet evolving user needs. This technological foundation ensures that the platform is not only robust in handling the diverse and dynamic data associated with DIY projects but also flexible enough to adapt to future innovations and user demands.

One of the platform’s core strengths is its ability to create a vibrant space where users can share their creativity and passion for DIY projects. The platform offers a rich repository of project ideas and tutorials across a wide range of categories, catering to both novice and experienced DIYers. By enabling users to create, read, update, and delete project posts, the platform ensures that content remains fresh and relevant. Additionally, the browsing and searching capabilities allow users to quickly find projects that match their interests, making the platform a valuable resource for inspiration and practical guidance.

The integration of a commenting system further enhances the user experience by promoting interaction and feedback. Users can engage in discussions, share tips, and provide support to one another, fostering a collaborative and supportive community. This interactive element is crucial for building a sense of belonging and encouraging users to actively participate and contribute to the platform.

Looking ahead, the planned future enhancements will significantly expand the platform’s capabilities and user base. The development of comprehensive user profiles, a mobile application, and social media integration will not only improve user engagement and accessibility but also extend the platform’s reach and influence. These features will enable users to connect more deeply with the community, showcase their work, and access the platform from any device, at any time.

Moreover, the introduction of monetization strategies such as premium features, ad-based revenue, affiliate marketing, and sponsored content will ensure the platform’s long-term sustainability. These revenue models will provide the necessary financial support to maintain and grow the platform, allowing for continuous improvements and the addition of new features.

**REFERENCES AND BIBLIOGRAPHY**

- MongoDB Documentation. - Retrieved from <https://docs.mongodb.com>

- Express.js Documentation. - Retrieved from <https://expressjs.com>

- React.js Documentation. - Retrieved from <https://reactjs.org>

- Node.js Documentation. - Retrieved from <https://nodejs.org>

- JWT Documentation - Retrieved from <https://jwt.io>

- Toastify Notification - Retrieved from <https://www.npmjs.com/package/react-toastify>

- Password Strength Estimation – Retrieved from <https://www.npmjs.com/package/zxcvbn/v/2.0.1>

- Multer - Retrieved from <https://www.npmjs.com/package/multer>