

CODECOM

**A PROJECT REPORT
for
Project (KCA451)
Session (2024-25)**

Submitted by

**SPARSH CHAUHAN
(2300290140185)**

**TANYA OBEROI
(2300290140192)**

**VISHWAS KUMAR VERMA
(2300290140207)**

**Submitted in partial fulfilment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATION

**Under the Supervision of
Dr. AMIT KUMAR
Assistant Professor**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(APRIL 2025)**

DECLARATION

I hereby declare that the work presented in this report entitled “CODECOM”, was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

SPARSH CHAUHAN
(2300290140185)

TANYA OBEROI
(2300290140192)

VISHWAS KUMAR VERMA
(2300290140207)

CERTIFICATE

Certified that **Sparsh Chauhan 2300290140185, Tanya Oberoi 2300290140192, Vishwas Kumar Verma 2300290140207** has/ have carried out the project work having “CODECOM” (Project-KCA451) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Sparsh Chauhan (2300290140185)

Tanya Oberoi (2300290140192)

Vishwas Kumar Verma (2300290140207)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Dr. Amit Kumar
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Akash Rajak
Dean
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

CODECOM

ABSTRACT

In the modern fast-paced digital age, learning and collaboration go hand in hand. This project presents a Community-Based Learning and Collaboration Platform, where individuals can sign up and create their own private communities. Each community is accessed using a special private code, with members able to join and contribute to group chats, task management, and live coding sessions

One of the main features of this platform is live coding collaboration, which allows newbies to acquire knowledge effectively by interacting with fellow learners and professional developers. Such an interactive system not only maximizes learning but also encourages collaboration and skill acquisition. In the next version, a Resources Section will be included, offering links to crucial documentation, tutorials, and video tutorials to further aid users in their learning process.

Through the integration of community involvement, collaborative tasks, and live coding, this project seeks to build a nurturing environment for learners to develop, exchange information, and advance their technical competencies interactively.

This will be an all-encompassing knowledge base where learners will have access to everything they need to deepen their understanding.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr. Amit Kumar** for his/ their guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Akash Rajak, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kinds of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Sparsh Chauhan (2300290140185)

Tanya Oberoi (2300290140192)

Vishwas Kumar Verma (2300290140207)

TABLE OF CONTENT

Declaration.....	ii
Acknowledgements.....	iii
Certificate.....	iv
Abstract.....	v
List of Figures.....	ix
Chapter 1 – Introduction.....	9-11
1.1 Overview.....	9
1.2 Project Description.....	9
1.3 Project Scope.....	10
1.4 Hardware/Software used in project.....	10-11
1.4.1 Hardware.....	10
1.4.2 Backend.....	10-11
1.4.3 Frontend.....	11
Chapter 2 – Feasibility Study.....	12-13
2.1 Economical Feasibility.....	12
2.2 Technical Feasibility.....	12-13
2.3 Operational Feasibility.....	13
Chapter 3 – Design and Planning.....	14- 22
3.1 SDLC Model (Prototype Model).....	14
3.2 Requirement Gathering.....	15
3.3 System Design.....	16
3.4 Development.....	17
3.5 Testing.....	17-18
3.6 Deployment.....	18
3.7 Maintenance and Support.....	18
3.8 Use Case Diagram.....	18-20
3.9 E-R Diagram.....	20-21
3.10 Data Flow Diagram.....	21-22

Chapter 4 – Project Screenshots & Coding.....	23-80
4.1.1 Home Page & code.....	23-29
4.1.2 Profile page & code.....	29-32
4.1.3 Community page & code.....	32-40
4.1.4 Community form & code.....	40-45
4.1.5 Backend code.....	45-80
Chapter 5 – Testing.....	81-84
5.1 Unit testing.....	81-82
5.2 Integration testing.....	82
5.3 Functional testing.....	82-83
5.4 Regression testing.....	83
5.5 User acceptance testing.....	83-84
5.6 Performance testing.....	84
Chapter 6– Conclusion and future scope.....	85-88
Chapter 7 – Bibliography.....	89

LIST OF FIGURES

3.1	Prototype Model	15
3.2	Use Case Diagram	20
3.3	E-R Diagram	21
3.4	Data Flow Diagram	22
4.1	Home Page	23
4.2	Profile Page	29
4.3	Community Page	33
4.4	Community Form	41

CHAPTER -1

INTRODUCTION

1.1 Overview

In today's fast-paced digital era, collaborative learning has become essential for skill development, particularly in the field of coding and software development. The CodeCom platform aims to address this need by creating a community-based learning environment where individuals can collaborate, share knowledge, and grow their skills together.

CodeCom is designed to provide a seamless and interactive experience for both beginners and experienced programmers.

1.2 Project Description

CodeCom is a community-based learning platform designed to enhance coding skills through real-time collaboration, task management, and resource sharing. It allows users to create or join private coding communities where they can chat, collaborate on live coding sessions, and manage tasks efficiently. The platform features real-time messaging for seamless communication, enabling members to discuss ideas and solve problems together.

The live coding environment allows multiple users to code simultaneously, promoting hands-on learning. Additionally, the task management system helps in organizing and tracking project activities. A resource section provides access to documentation, tutorials, and educational videos to support continuous learning.

A resource section provides access to documentation, tutorials, and educational videos to support continuous learning. Built using React.js, Node.js, and Firebase, CodeCom offers a dynamic and user-friendly interface with secure authentication and data storage. The platform fosters collaborative skill development by enabling learners to share knowledge, solve coding challenges, and grow together in a practical and interactive environment.

1.3 Project Scope

The CodeCom project aims to create a community-based learning platform that promotes collaborative coding, real-time interaction, and skill development. Its scope covers a range of functionalities designed to enhance the learning experience and foster teamwork among users.

The platform will enable users to create and manage private communities, allowing members to collaborate through live coding sessions and real-time chat. It will also include a task management system for organizing and tracking project activities, ensuring efficient teamwork. The resource section will provide access to documentation and educational materials, offering continuous learning support.

The project's scope covers the development, deployment, and ongoing improvement of a collaborative learning environment that empowers users to enhance their coding skills through practical, real-time interaction.

1.4 Hardware/Software used in Project

1.4.1 Hardware: Windows 10 or 11

1.4.2 Backend:

The backend of CodeCom is designed to handle the server-side operations, data management, and real-time communication. It is developed using **Node.js** with **Express.js** framework, allowing RESTful API development that interacts seamlessly with the frontend.

It is built using the following technologies:

1.4.2.1 Node.js & Express.js:

1. Used to create a scalable and efficient server.
2. Manages API routes, handles requests, and serves data to the frontend.

1.4.2.2 Socket.io:

- Enables real-time chat and live coding functionalities.
- Facilitates seamless communication between users in different communities.

1.4.2.3 JWT Authentication:

- Implements secure authentication using JSON Web Tokens (JWT).
- Ensures that only authorized users can access their respective communities.

1.4.3 Frontend:

User Interface: The frontend of CodeCom is built using React.js, Tailwind CSS, and Framer Motion to deliver a fast, responsive, and visually appealing user experience. React.js ensures efficient rendering and smooth navigation through its component-based architecture, making the platform dynamic and modular. The styling is managed using Tailwind CSS, a utility-first framework that provides a clean, consistent, and responsive design with minimal code.

To enhance the visual appeal, **Framer Motion** is used to add **smooth animations and transitions**, creating a more engaging and interactive user interface. Together, these technologies ensure that CodeCom offers a seamless and user-friendly experience for collaborative learning and real-time coding.

IDE (Integrated Development Environment): Visual Studio Code (VS Code) is the primary IDE used for the development of CodeCom.

Browser: Chrome, Edge

CHAPTER-2

FEASIBILITY STUDY

A feasibility study is crucial for evaluating whether a project is worth the investment of substantial resources and time. Here is an assessment of the CodeCom system based on three important aspects: Economic, Technical, and Operational.

2.1 Economical Feasibility

Development Costs: The development of CodeCom involves **designing the platform, building the frontend and backend**, and integrating **real-time collaboration features**. Since **open-source technologies** like **React.js, Node.js, and Socket.io** are used, the initial development costs are minimal. However, expenses may include **cloud services** (e.g., Firebase or MongoDB Atlas) for database management and **domain registration**.

Hardware/Software Costs: The project requires monthly cloud hosting services (e.g., Firebase, MongoDB Atlas, or Render) to store user data and enable real-time communication. Additionally, the use of third-party APIs or libraries (e.g., for authentication, real-time features, or notifications) may incur minor costs. Since the platform uses open-source frameworks, software licensing costs are minimal.

Maintenance Costs: Includes bug fixes, system updates, server to be Management and ensuring security.

2.2 Technical Feasibility

System Architecture:

Frontend (Vercelli): Use React.js for a dynamic user interface. Vercel Handles Automatic scaling and fast deployment.

Backend (Netlify): The backend uses Node.js and Express.js with Netlify serverless functions to handle API requests and process real-time operations. The serverless architecture allows for automatic scaling based on user demand.

Database: MongoDB Atlas is used to store user data, messages, tasks, and community details. It ensures high availability, security, and scalability, making data management efficient.

2.2.1 Technology Stack:

Frontend: React.js (or Next.js), Axios for API calls Tailwind CSS

Backend: Node.js, Express.js, Netlify serverless functions for API handling.

Database: MongoDB Atlas for storing data.

2.3 OPERATIONAL FEASIBILITY

User Acceptance: CodeCom offers real-time chat, live coding, and task management, making it highly interactive and valuable for collaboration learning. Users are likely to appreciate its ease of use and the ability to improve their coding skills through shared tasks and discussions.

Operational Impact: the platform seamlessly integrates into user's learning routines, offering reliable performance with automated scaling and low maintenance. Its user-friendly interface ensures smooth operation, even with increasing user activity.

CodeCom is engineered to operate efficiently within a collaborative environment, supporting both novice and experienced users. Its robust backend, powered by Node.js and MongoDB, ensures smooth handling of multiple concurrent tasks and real-time communication. The platform's scalability allows it to adapt to increasing user loads without performance degradation. Integration of authentication, role-based access, and secure data handling enhances operational trust and compliance.

CHAPTER-3

DESIGN AND PLANNING

3.1 Software development life cycle model prototype model

The Prototype model requires that before carrying out the development software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possibly exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software.

In many instances, the client only has general view of what is expected from the software product, the processing needs, and the output requirement, the prototyping model may be employed.

For CodeCom, the prototype includes basic modules such as user registration/login, community creation, task posting, and basic chat functionality. Stakeholders can interact with this simplified version to provide feedback, which helps in refining requirements and adjusting the final design before full-scale development begins.

This iterative process minimizes the risk of misunderstandings and leads to the development of a more user-centric and functionally rich platform. The development team builds a quick version of the core functionalities—like creating a coding room, assigning roles, or sending messages in a community—so users and project stakeholders can evaluate the flow, usability, and interface design.

Based on their feedback, improvements are made in subsequent versions of the prototype. During the Implementation phase, these components were developed and integrated to provide a seamless user experience.

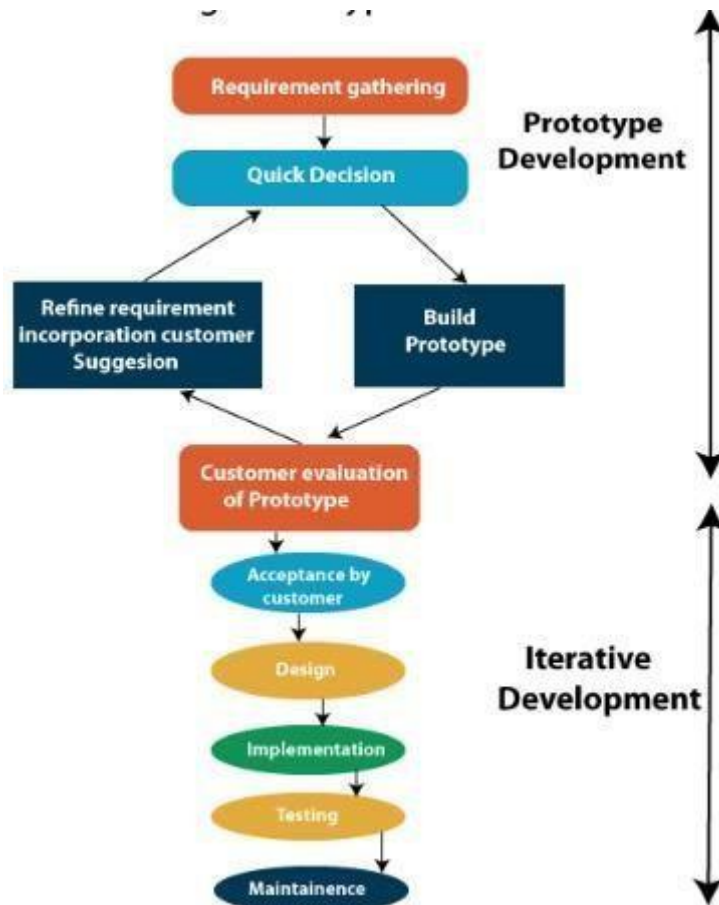


Fig 3.1 Prototype Model

1. Requirement Gathering:

In the initial phase of **CodeCom development**, the primary focus is on **gathering detailed requirements** from potential users, including **coders, learners, and mentors**. Various methods, such as **surveys, interviews, and feedback sessions**, are employed to understand user needs and expectations.

The key requirements include:

User Registration & Profiles: Users should be able to create accounts and profiles with details like name, skills, and interests.

Community Creation: Users can create and join private or public coding communities with unique access codes.

Real-Time Chat & Collaboration: The system should enable instant messaging, task sharing, and collaboration on coding projects.

Live Coding Environment: Integration of a **real-time code editor** to facilitate collaborative programming.

Dashboard: An intuitive dashboard for users to manage their communities, tasks, and interactions.

Security and Privacy: Ensure the platform uses JWT authentication and encryption to protect user data and privacy.

2. System Design

The **System Design** phase of **CodeCom** focuses on creating a **scalable, efficient, and secure architecture** to meet both functional and non-functional requirements. It includes designing the **frontend, backend, database, and UI/UX** to ensure seamless performance.

Technical level and will cover:

Architecture Design: CodeCom follows a client-server model, where the React.js frontend interacts with the Node.js backend through RESTful APIs. The backend manages features such as user authentication, community creation, real-time chat, task collaboration, and live coding.

The backend will be the utilize Express.js to set up a server and manage API requests for the features such as user authentication, idea at Submission CodeCom retrieval. The Security and measures like JWT Authentication and data to be validation will be implemented to guarantee. Those only authorized users can access certain data.

Database Design: The platform uses MongoDB Atlas to store user profiles, community data, chat messages, and coding projects. The schema is designed for efficient data retrieval and management, ensuring scalability and reliability.

Security Measures: JWT authentication ensures that only authorized users can access sensitive data. Additionally, data validation and encryption enhance security.

UI/UX Design: Comprehensive wireframes and mockups are created to provide an intuitive and smooth user experience. The Tailwind CSS framework is used for styling, ensuring a modern and responsive design.

This will include designing screens for:

- 2.1 User registration/login
- 2.2 Create Community Management
- 2.3 Join Community
- 2.4 Real-time Messaging
- 2.5 Real-time Coding Environment
- 2.6 Dashboard for Task and Progress Monitoring

3. Development

The Development Phase of CodeCom involves transforming the design into a fully functional platform through frontend, backend, and database integration. This stage also includes coding collaboration features, testing, and iterative improvements.

Real-time communication was achieved by integrating Socket.io, allowing users within a community to engage in live chat and collaborative coding sessions. Special attention was given to the community code logic, enabling users to securely create or join private communities using unique invite codes.

4. Testing

The Testing Phase of CodeCom ensures the platform is stable, secure, and ready for public use. This stage involves thorough testing to identify and fix bugs, optimize performance, and guarantee a smooth user experience before final deployment.

The primary goal was to ensure the platform's stability, functionality, and user experience before deployment. Various types of testing were conducted, starting with **unit testing** to validate individual functions such as user registration, community creation, and task assignment.

5. Deployment

The Deployment Phase of CodeCom focuses on making the platform available for public use after successful development and testing. This phase ensures that the system is properly configured, accessible, and scalable for real-time collaboration. It involves setting up the production environment, hosting the frontend and backend, and configuring the database.

6. Maintenance and Support

The Maintenance and Support Phase of CodeCom focuses on ensuring the platform remains reliable, secure, and efficient after deployment. This phase involves regular monitoring, updating, and providing technical assistance to users. It includes bug fixes, patches, and performance optimization to enhance the platform's stability. Regular security audits and vulnerability patching are performed to protect user data and prevent unauthorized access. The backend and database undergo continuous maintenance to ensure data integrity, with regular backups and performance enhancements. Additionally, user support is provided through a help desk and ticketing system to quickly resolve issues and gather feedback for future improvements. The platform is continuously refined by introducing new features, scaling capabilities, and optimizing performance based on user needs. This phase guarantees that CodeCom remains secure, scalable, and user-friendly, delivering a seamless and reliable experience over time.

3.2 Use Case Diagram

The Use Case Diagram for CodeCom models the dynamic behavior of the platform by capturing the interactions between the system and its actors. It represents the high-level functionalities and illustrates how users engage with the system. The diagram consists of actors (users, community members, and

admins) and their interactions with the system's features, such as user authentication, community management, task collaboration, real-time messaging, and live coding.

On the other hand, the Admin plays a more supervisory role, with capabilities such as **managing users** and, in the future, **managing educational resources** like documentation and tutorial videos.

These use cases are represented as actions connected to their respective actors, highlighting the functional requirements of the system. The diagram helps clarify system behavior, guides the development team, and ensures that all user needs and administrative controls are properly addressed in the design and implementation of CodeCom.

Key Use Cases in CodeCom:

1. **User Registration & Login:** Users and admins can create accounts, log in, and manage their profiles.
2. **Community Creation & Management:** Users can create their own **communities** and invite members using a private code.
3. **Task Collaboration:** Community members can **assign, manage, and track tasks** collaboratively.
4. **Real-Time Messaging:** Users can **chat with community members** to collaborate effectively.
5. **Live Coding:** Members can engage in **live coding sessions** to enhance their skills and solve problems together.
6. **Admin Management:** Admins can **oversee the platform**, manage user data, and ensure security and smooth operations.

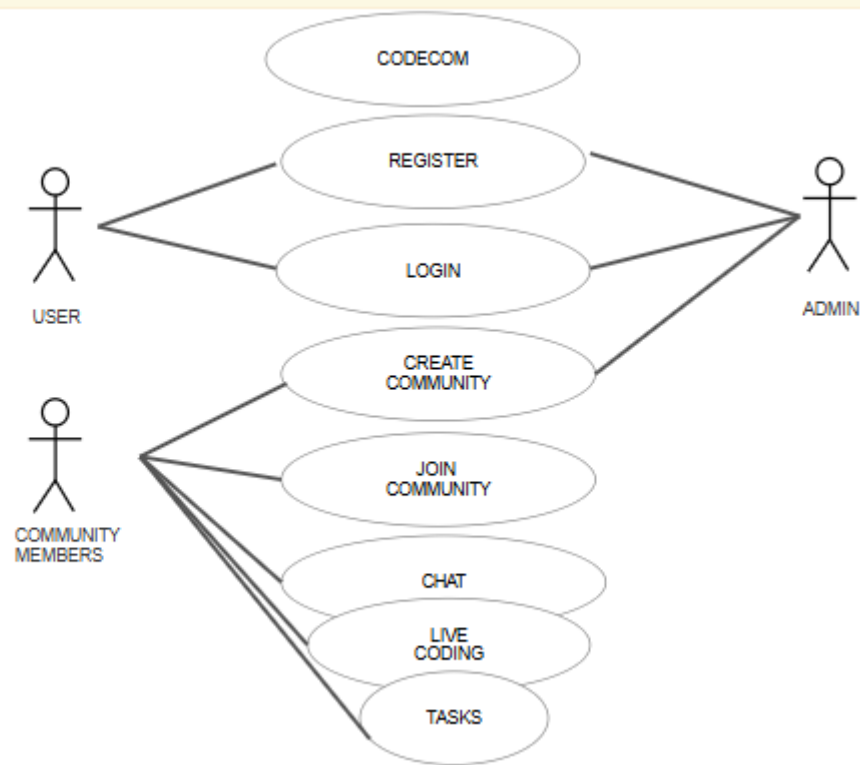


Fig 3.2 USE CASE DIAGRAM

3.3 E-R(ENTITY-RELATIONSHIP) DAIGRAM

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements relation for a specified system. It develop conceptual design for the database. It also develop simple and design view of data.

In ER modeling, the data base structure portrayed as a diagram call an entity- relationship diagram. The primary entities in the system include **User**, **Community**, **Message**, **Task**, and optionally, **Resource** for future expansion. Each **User** has attributes such as user ID, name, email, and password, and can either **create** or **join** a Community.

A **Community** entity includes attributes like community ID, name, and unique join code, and it maintains a one-to-many relationship with both Users and Tasks—indicating that one community can have many users and many tasks.

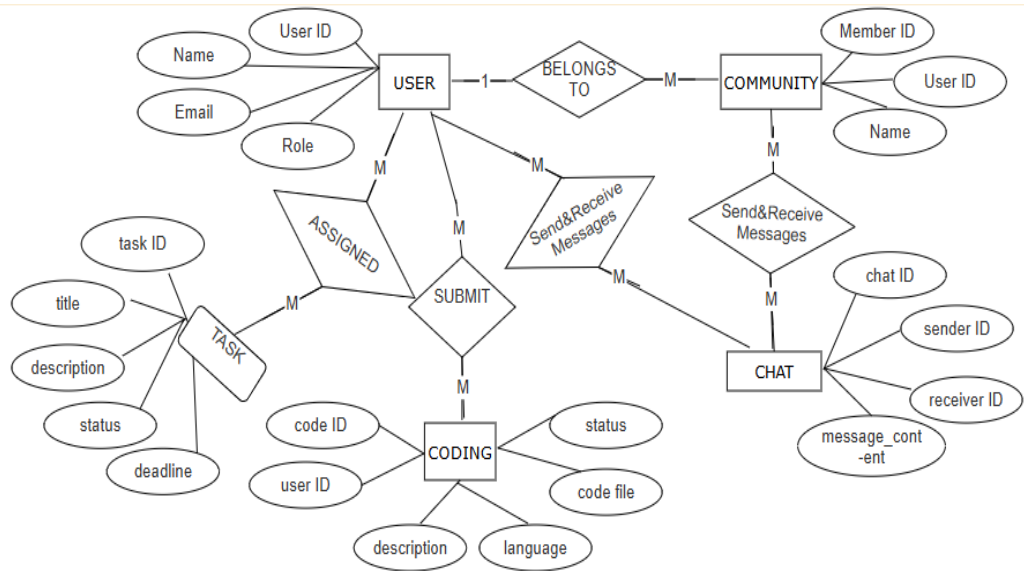


Fig 3.3 E-R DIAGRAM

3.4 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both. It show how data centers and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between as system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The central system processes this input and communicates with internal data stores such as the User Database, Community Database, Chat and Coding Data, and Task Data. As the DFD is further detailed in Level 1, it breaks down into key functional modules including User Authentication, Community Management, Chat and Live Coding, Task Collaboration, and a future module for Resource Sharing.

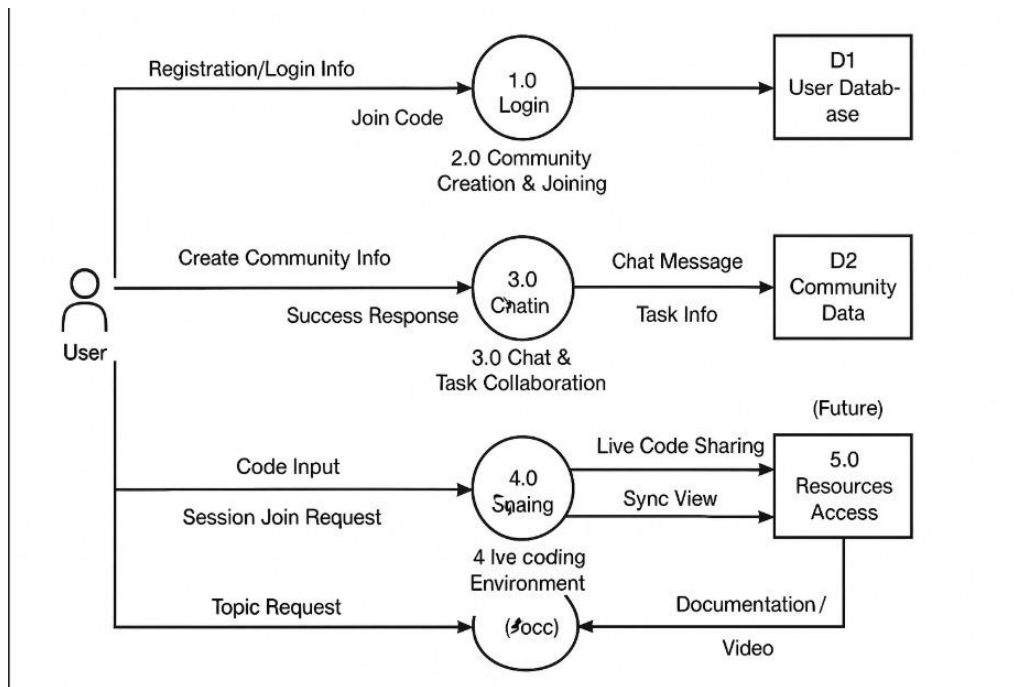


Fig 3.4 DATA FLOW DIAGRAM

CHAPTER-4

PROJECT SCREENSHOTS & CODING

4.1 FRONTEND

4.1.1 HOME PAGE



Fig 4.1.1 HOME PAGE

CODE:-

```
import { useEffect, useRef, useState } from "react";
import Frame from "../Frame";
import { motion } from "framer-motion";
import comImage from "../images/com-image.png";
import SignUpAndLogin from "../SignUpAndLogin";
function Home() {
  const [mousePosition, setMousePosition] = useState({
    x: 0,
    y: 0,
  });
  const [cursorVariant, setCursorVariant] = useState("default");
  useEffect(() => {
    const mouseMove = (e) => {
```

```

setMousePosition({
  x: e.clientX,
  y: e.clientY,
});

};

window.addEventListener("mousemove", mouseMove);

return () => {
  window.removeEventListener("mousemove", mouseMove);
};

}, []);

const variants = {
  default: {
    x: mousePosition.x - 16,
    y: mousePosition.y - 16,
  },
  text: {
    height: 150,
    width: 150,
    x: mousePosition.x - 75,
    y: mousePosition.y - 75,
    backgroundColor: "purple",
    opacity: 0.3,
    transition:
      "height 0.3s ease-in-out, width 0.3s ease-in-out, background-color 0.3s ease-in-out, opacity 0.3s ease-in-out",
  },
};

const textEnter = () => setCursorVariant("text");
const textLeave = () => setCursorVariant("default");
const [showSignUp, setShowSignUp] = useState(false);

```



```

/>

<Frame
color="bg-green-400"
size="w-24 md:w-48 h-24 md:h-48"
top="60%"
left="5%"
delay={1}
/>

<Frame
color="bg-red-400"
size="w-24 md:w-48 h-24 md:h-48"
top="60%"
left="80%"
delay={5}
/>

</div>

<div className="border border-orange-200 w-full min-h-screen md:min-h-[500px] rounded-xl relative">

  <div className="flex justify-between items-center mt-4 px-4 md:px-10">

    <div>

      <img src={comImage} alt="" className="h-10 w-10 md:h-14 md:w-14" />

    </div>

    <button
      onClick={() => setShowSignUp(true)}

      className="relative group border-none bg-transparent p-0 outline-none cursor-pointer font-mono font-light uppercase text-sm md:text-base"

      >

      <span className="absolute top-0 left-0 w-full h-full bg-black bg-opacity-25 rounded-lg transform translate-y-0.5 transition duration-[600ms] ease-[cubic-bezier(0.3,0.7,0.4,1)] group-hover:translate-y-1 group-hover:duration-[250ms] group-active:translate-y-px"></span>
    
```

>

CodeCom

</h1>

<p className="px-4 md:px-16 text-base md:text-lg text-center mt-4 font-semibold bg-gradient-to-r from-orange-400 to-red-600 bg-clip-text text-transparent">

Join a vibrant community of coders, collaborate on projects, and level up your skills together. Whether you are a beginner or an expert, connect with like-minded learners, share knowledge, and grow in a supportive environment. Start coding, learning, and building today!

</p>

</motion.div>

{showSignUp && (

<motion.div

className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50 p-4"

initial={{ opacity: 0 }}

animate={{ opacity: 1 }}

onClick={e => {

if (e.target === e.currentTarget) {

setShowSignUp(false);

}

}}

>

<motion.div

className="bg-white rounded-xl w-full max-w-md p-6"

ref={formRef}

initial={{ opacity: 0, scale: 0.9 }}

animate={{ opacity: 1, scale: 1 }}

transition={{ duration: 0.3, ease: "easeOut" }}

```

>
<SignUpAndLogin />
</motion.div>
</motion.div>
)}
</div>
</div>
</>
);
}
export default Home;

```

4.1.2 PROFILE PAGE

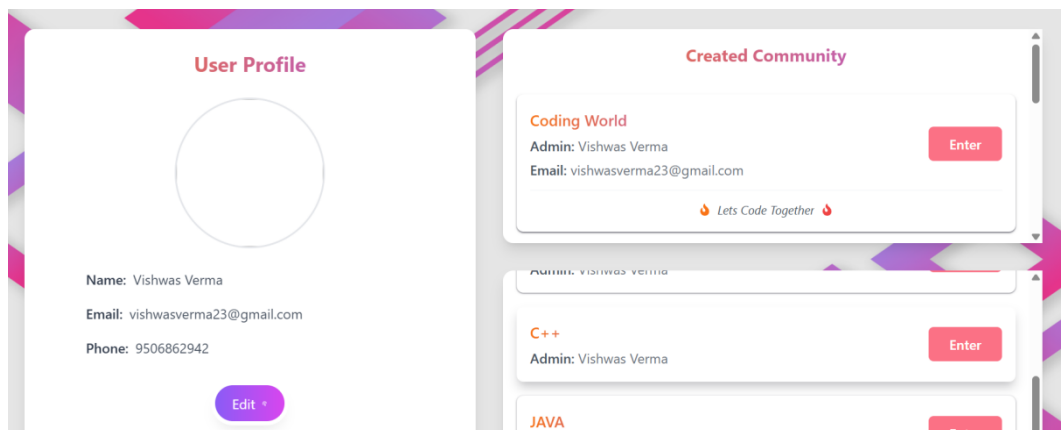


Fig 4.1.2 PROFILE PAGE

CODE:-

```

import { useEffect, useState } from "react";
import userBack from "../images/userBack.png";
import axios from "axios";
import UserEditCard from "./UserEditCard";
import UserProfileCardd from "./UserProfileCardd";
import JoinedCommunity from "./JoinedCommunity";

```

```

import CreatedCommunity from "../CreatedCommunity";

function UserProfile() {
  const [visible, setVisible] = useState(true);
  const [userData, setUserData] = useState({
    name: "",
    email: "",
    number: "",
    communityList: [],
  });
  const fetchData = async () => {
    try {
      console.log("hello");
      const resp = await axios.get("http://localhost:2024/api/user/userProfile", {
        withCredentials: true,
      });
      console.log(resp.data);
      setUserData({
        name: resp.data.name,
        email: resp.data.email,
        number: resp.data.number,
        communityList: resp.data.communityList,
      });
    } catch (error) {
      console.log(error);
    }
  };
  useEffect(() => {
    fetchData();
  }, []);
  const handleUpdate = () => {

```

Created Community

</h1>

<div className="mt-4 sm:mt-6">

<CreatedCommunity />

</div>

</div>

<div className="bg-white shadow-lg p-4 w-full rounded-xl h-[250px] md:h-58 overflow-y-auto">

<h1 className="font-bold text-center text-lg sm:text-xl bg-gradient-to-r from-orange-500 to-purple-500 bg-clip-text text-transparent top-0 bg-white pb-2">

Joined Community

</h1>

<div className="mt-4 sm:mt-6">

<JoinedCommunity />

</div>

</div>

</div>

</div>

</div>

</div>

);

}

export default UserProfile;

4.1.3 COMMUNITY PAGE



Fig 4.1.3 COMMUNITY PAGE

CODE:-

```
import axios from "axios";
import { useEffect, useState } from "react";
import { NavLink, useParams } from "react-router-dom";
import { FaGithub, FaLinkedin } from "react-icons/fa";
import Chat from "../chat/Chat";
import CodeTogether from "../chat/CodeTogether";
function CommunityPage() {
  const [visible, setVisible] = useState("");
  const [infoVisible, setInfoVisible] = useState(false);
  const [membersVisible, setMembersVisible] = useState(false);
  const [isAdmin, setIsAdmin] = useState(false);
  const { name } = useParams();
  const decodeName = decodeURIComponent(name);
  const [community, setCommunity] = useState(() => {
    const savedCommunity = localStorage.getItem(decodeName);
    return savedCommunity ? JSON.parse(savedCommunity) : { members: [] };
  });
```

```

xmlns="http://www.w3.org/2000/svg"
width="40px"
height="40px"
viewBox="0 0 24 24"
className="stroke-blue-300"
>
<path
strokeLinejoin="round"
strokeLinecap="round"
strokeWidth="1.5"
d="M11 6L5 12M5 12L11 18M5 12H19"
></path>
</svg>
</button>
</NavLink>
</div>
<div className="flex flex-col text-center">
<h1 className="text-3xl h-16 md:text-5xl font-extrabold bg-gradient-to-r
from-orange-500 to-purple-500 bg-clip-text text-transparent ">
{community.name}
</h1>
<p className="text-zinc-600 mb-2 text-sm md:text-lg">
{community.description}
</p>
</div>
<div className="relative">
<section
onClick={() => setInfoVisible(!infoVisible)}
className="relative flex cursor-pointer justify-center items-center"
>
<div className="group flex justify-center transition-all hover:scale-125
rounded-full bg-gray-200 p-1">

```

```

<p className="text-sm">Admin: {community.admin.name}</p>
<p className="text-sm">Code: {community.code}</p>
<p className="text-sm">
Strength: {community.members.length}
</p>
</div>
{isAdmin && (
<div className="flex justify-between">
<button className="inline-block cursor-pointer font-mono text-sm font-bold
bg-slate-500 text-white py-2 px-4 rounded-full transition ease-in-out delay-75
duration-100 hover:-translate-y-1 hover:scale-110 transform hover:opacity-
75">
Edit
</button>
<button className="inline-flex items-center px-3 py-2 bg-red-600 transition
ease-in-out delay-75 hover:bg-red-700 text-white text-sm font-medium
rounded-full hover:-translate-y-1 hover:scale-110">
<svg
stroke="currentColor"
viewBox="0 0 24 24"
fill="none"
className="h-5 w-5 mr-2"
xmlns="http://www.w3.org/2000/svg"
>
<path
d="M19 7l-.867 12.142A2 2 0 0116.138 21H7.862a2 2 0 01-1.995-1.858L5
7m5 4v6m4-6v6m1-10V4a1 1 0 00-1-1h-4a1 1 0 00-1-1v3M4 7h16"
strokeWidth="2"
strokeLinejoin="round"
strokeLinecap="round"
></path>
</svg>
Delete
</button>

```



```

</div>
)}
</div>
</div>
</div>
)}
</div>
</div>
<div className="flex flex-col ">
  <div onClick={() => setMembersVisible(!membersVisible)}
    className=" cursor-pointer bg-gray-200 shadow-inner shadow-zinc-400 w-
    full">
    <div className="flex justify-between items-center px-4 py-3 border-b border-
    white">
      <h1 className="font-bold text-zinc-700">Members</h1>
      <button
        className="text-zinc-600 hover:text-zinc-800 focus:outline-none"
        >
        <svg
          xmlns="http://www.w3.org/2000/svg"
          className={`h-6 w-6 transition-transform duration-200 ${
            membersVisible ? "transform rotate-180" : ""
          }`}
          fill="none"
          viewBox="0 0 24 24"
          stroke="currentColor"
          >
            <path
              strokeLinecap="round"
              strokeLinejoin="round"
              strokeWidth={2}
              d="M19 9l-7 7-7-7"

```

```

</li>

<li>
  onClick={() => setVisible("code")}

  className="cursor-pointer border border-zinc-300 w-full text-center p-2
  hover:bg-gray-50"
>
  Code Together
</li>

</ul>

<div className="p-4">
  { visible === "chat" ? (
    <div>
      <Chat community={community}/>
    </div>
  ) : visible === "task" ? (
    <div>Task</div>
  ) : visible === "code" ? (
    <div><CodeTogether community={community}/></div>
  ) : (
    <div className="flex w-full h-[60vh] justify-center items-center">
      <h1 className="text-2xl md:text-5xl text-gray-300 font-bold text-center">
        What are you going to do Today?
      </h1>
    </div>
  )}
</div>

</div>

</div>

</div>

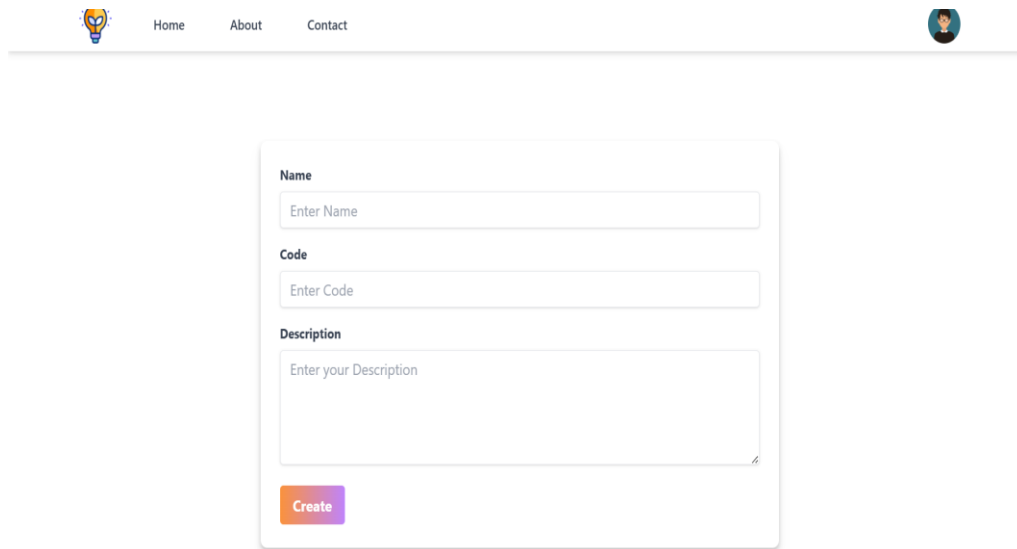
</div>

);
}

export default CommunityPage;

```

4.1.4 COMMUNITY FORM



The image shows a web application header and a community form. The header has a light blue background with a lightbulb icon, 'Home', 'About', and 'Contact' links, and a user profile icon. The form is a white box with a light blue border containing three input fields: 'Name' (placeholder 'Enter Name'), 'Code' (placeholder 'Enter Code'), and 'Description' (placeholder 'Enter your Description'). A 'Create' button is at the bottom left of the form.

Fig 4.1.4 COMMUNITY FORM

CODE:-

```
import { useState } from "react";
import { motion } from "framer-motion";
import { Slide, ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import { useNavigate } from "react-router-dom";
import axios from "axios";

function CommunityForm() {
  const navigate = useNavigate();
  const [input, setInput] = useState({
    name: "",
    code: "",
    description: "",
  });

  const handleSubmit = async(e) => {
```

```

e.preventDefault();

if(!input.name || !input.code || !input.description){
return toast.error("Please fill all the fields",{ autoClose:1000})

}

try {

const response= await
axios.post("http://localhost:2024/api/community/createCommunity",input,{ wit
hCredentials:true})

console.log(response.data);

toast.success("Account created successfully", { autoClose: 1000 });

setInput({
name: "",
code: "",
description: "",
});

const communityName=response.data.community.name;

setTimeout(() => {
navigate(`/Community/${communityName}`)
},1000)

} catch (error) {
console.log(error,"failed to create community");
toast.error("Failed to create community",{autoClose:1000})
}

};

const handleInput = (e) => {
const { name, value } = e.target;
setInput((prev) => ({
...prev,
[name]: value,
}));
}

```

```

};
return (
  <>
  <ToastContainer transition={Slide}/>
  <motion.div
    initial={{ opacity: 0, y: 30 }}
    animate={{ opacity: 1, y: 0 }}
    transition={{ duration: 1 }}
    className=" flex justify-center items-center h-screen"
  >
    <form className="bg-white w-[90vw] md:w-[50vw] p-6 rounded-lg shadow-
md drop-shadow-lg">
      <div className="mb-4">
        <label htmlFor="title" className="block text-gray-700 text-sm font-bold mb-
2">
          Name
        </label>
        <input
          placeholder="Enter Name"
          onChange={handleInput}
          value={input.name}
          name="name"
          type="text"
          className="shadow appearance-none border rounded w-full py-2 px-3 text-
gray-700 leading-tight focus:outline-none focus:shadow-outline"
        />
      </div>
      <div className="mb-4">
        <label htmlFor="title" className="block text-gray-700 text-sm font-bold mb-
2">
          Code
        </label>

```

```

onClick={handleSubmit}

type="submit"

className="bg-gradient-to-r from-orange-400 to-purple-400 text-white font-
bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"

>

Create

</button>

</div>

</form>

</motion.div>

</>

);

}

export default CommunityForm;

```

4.2 BACKEND

(codeAuthcontroller.js)

```

import { sendResetPasswordEmail, sendSuccessEmail, sendVerificationEmail,
sendWelcomeEmail } from "../Email/email.js";

import { User } from "../models/userModel.js";

import bcryptjs from 'bcryptjs';

import crypto from 'crypto';

import dotenv from 'dotenv'

import { generateTokenAndSetCookie } from
"../utils/generateTokenAndSetCookie.js";

dotenv.config()

export const signup=async(req,res)=>{

```

```

const { email,password,name } = req.body;

try {

  if(!email||!name||!password){

    throw new Error("All fields are required")

  }

const userAlreadyExist=await User.findOne({email})

  if(userAlreadyExist){

    return res.status(400).json({ success: false,message: "User already exists"})

  }

const hashedPassword = await bcryptjs.hash(password,10);

const verificationToken= Math.floor(

  100000 + Math.random() * 900000

).toString();

const user= new User(

  {

    email,

    password: hashedPassword,

    name,

    verificationToken:verificationToken,

    verificationTokenExpiresAt:Date.now() + 24 * 60 * 60 * 1000,

  }

)

await user.save();

```

```

    user.resetToken=undefined;

    user.resetTokenExpiration=undefined;

    await user.save();

    await sendSuccessEmail(user.email)

    res.status(200).json({ success:true, message:"Password reset successfully" })
  } catch (error) {

    console.log("error resetting password",error);

    res.status(400).json({ success: false,message:"Password reset failed" })

  }
}

export const verifyEmail=async(req,res)=>{

  const { code}=req.body;

  try {

    const user= await User.findOne(

      {

        verificationToken:code,

        verificationTokenExpiresAt:{$gt:Date.now()}

      }

    )

    if(!user){

      return res.status(400).json({ success: false,message:"Invalid or Expired Token" })

    }

    user.isVerified = true;

```



```

}

export const login=async(req,res)=>{

  const {email , password}=req.body;

  console.log(req.body);

  if(!email||!password){

    return res.status(400).json({ success: false,message:"All fields are
required" })

  }

  try {

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

    if(!user){

      console.log("user not found");

      return res.status(400).json({ success: false,message:"User not found" })

    }

    if(!user.isVerified){

      return res.status(400).json({ success: false,message:"Email is not
Verified" })

    }

    const isPasswordValid= await bcryptjs.compare(password,user.password);

    if(!isPasswordValid){

      console.log("not valid");

      return res.status(400).json({ success: false,message:"Invalid
credentials" })

    }

    console.log("pass check successful");

```

```

    await generateTokenAndSetCookie(res,user._id)

    user.lastLogin = new Date()

    await user.save();

    res.status(200).json({ success:true, message:"Logged in successfully",

      user:{

        ...user._doc,

        password:undefined

      }

    })

  }

  catch (error) {

    console.log("Login Failed",error);

    res.status(400).json({ success: false,message:error.message })

  }

}

export const logout=async(req,res)=>{

  res.clearCookie('authToken')

  res.json({ success:true, message:"Logged out successfully" })

}

```

(Communitycontroller.js)

```

import Community from "../models/communityModel.js";

import { User } from "../models/userModel.js";

import io from "../index.js"

export const createCommunity = async (req, res) => {

```

```

const { name, code, description } = req.body;

try {

  const communityExisted = await Community.findOne({ name });

  if (communityExisted) {

    return res.status(400).json({ success: false, message: "Community
already exists" })

  }

  const community = new Community({

    name,

    description,

    code,

    admin: req.userId,

    members: [req.userId]

  })

  const user = await User.findById(req.userId)

  user.createdCommunities.push(community._id)

  await user.save()

  await community.save()

  res.status(201).json({ success: true, message: "Community created
successfully", community })

} catch (error) {

  console.log(error, "failed to create community");

  res.status(400).json({ success: false, message: "Failed to create
community" })

}

```

```

        res.status(200).json({ success: true, message: "Community joined
successfully" })

    } catch (error) {

        console.log(error, "failed to join community");

        res.status(400).json({ success: false, message: "Failed to join
community" })

    }

}

export const getJoinedCommunity = async (req, res) => {

    try {

        const joinedCommunity = await Community.find({ members:
req.userId }).select('name code admin ').populate('admin', 'name')

        console.log(joinedCommunity);

        res.status(200).json(joinedCommunity)

    } catch (error) {

        console.log(error, "failed to get the joined community");

        res.status(400).json({ success: false, message: "Failed to get the
joined community" })

    }

}

export const getCreatedCommunity = async (req, res) => {

    try {

        const createdCommunity = await Community.find({ admin:
req.userId }).select('name code admin description ').populate('admin', 'name
email')

        console.log(createdCommunity);

```

```

    res.status(200).json(createdCommunity)

  }

  catch (error) {

    console.log(error, "failed to get the created community");

    res.status(400).json({ success: false, message: "Failed to get the
created community" })

  }

}

export const fetchCommunity = async (req, res) => {

  const { name } = req.params;

  try {

    const community = await Community.findOne({ name
}).select('admin name code description members').populate('members', 'name
email').populate('admin', 'name email')

    if (!community) {

      return res.status(400).json({ success: false, message: "Community
not found" })

    }

    res.status(200).json({ community, currentUserId: req.userId })

  } catch (error) {

    console.log(error, "failed to fetch community");

    res.status(400).json({ success: false, message: "Failed to fetch
community" })

  }

}

```

```

        console.log("community not found");

        return res.status(400).json({ success: false, message: "Community
not Found" })

    }

    const newMessage = { user: req.userId, message };

    community.messages.push(newMessage);

    await community.save();

    io.to(code).emit("recieveMessage", { user:req.userId, message });

    res.status(200).json({ success: true, message: "Message sent" })

    } catch (error) {

        res.status(400).json({ success: false, message: "Failed to send" })

    }

}

```

(Userprofilecontroller.js)

```

import { User } from "../models/userModel.js"

import jwt from 'jsonwebtoken'

export const updateUser=async(req,res)=>{

    const token =req.cookies.authToken;

    const {inputName,inputNumber}= req.body;

    if(!token){

        console.log("token is invalid");

    }

    try {

        const decoded= jwt.verify(token,process.env.JWT_SECRET_KEY)

```

```

const userId=decoded.userId

const user = await User.findByIdAndUpdate(userId,{
  name:inputName,
  number:inputNumber
},{ $set:req.body}, {new:true, runValidators:true})

if(!user){
  res.status(400).json("user not found")
}

await user.save()

res.status(200).json(user+" data updated successfully")
} catch (error) {
  console.log(error+" failed updating user data");
  res.status(400).json("update failed")
}
}

export const userProfile=async(req,res)=>{
  const token =req.cookies.authToken;
  if(!token){
    console.log("token is invalid ");
  }
  try {
    const decoded= jwt.verify(token,process.env.JWT_SECRET_KEY)
    const userId=decoded.userId;
    const user=await User.findById(userId)

```

```

    if(!user){

        res.status(400).json("user not found");

    }

    res.status(200).json(user);

} catch (error) {

    console.log(error,"error finding user");

    res.status(400).json("user data is not available")

}

}

```

(connectedDB.js)

```

import mongoose from "mongoose"

export const connectDb=async()=>{

    try {

        mongoose.connect(process.env.MONGO_URL)

        console.log("Mongodb Connected");

    } catch (error) {

        console.error(`Error connecting to MongoDB: ${error.message}`);

        process.exit(1);

    }

}

```



```

});

export const sender = process.env.MY_EMAIL;

export const sendEmail = async (recipient, subject, html) => {

  const mailOptions = {

    from: sender,

    to: recipient,

    subject,

    html,

  };

  try {

    const info = await transporter.sendMail(mailOptions);

    console.log('Email sent: ', info.response);

    return info;

  } catch (error) {

    console.error('Error sending email:', error);

    throw new Error('Error sending email');

  }

};

```

(mailtemplate.js)

```

export const VERIFICATION_EMAIL_TEMPLATE = `

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

```

```

</div>

</body>

</html>

`;

export const PASSWORD_RESET_SUCCESS_TEMPLATE = `

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Password Reset Successful</title>

</head>

<body style="font-family: Arial, sans-serif; line-height: 1.6; color: #333; max-width: 600px; margin: 0 auto; padding: 20px;">

  <div style="background: linear-gradient(to right, #4CAF50, #45a049); padding: 20px; text-align: center;">

    <h1 style="color: white; margin: 0;">Password Reset Successful</h1>

  </div>

  <div style="background-color: #f9f9f9; padding: 20px; border-radius: 0 0 5px 5px; box-shadow: 0 2px 5px rgba(0,0,0,0.1);">

    <p>Hello,</p>

    <p>We're writing to confirm that your password has been successfully reset.</p>

    <div style="text-align: center; margin: 30px 0;">

```

```
<div style="background-color: #4CAF50; color: white; width: 50px;
height: 50px; line-height: 50px; border-radius: 50%; display: inline-block;
font-size: 30px;">
```

```
✓
```

```
</div>
```

```
</div>
```

```
<p>If you did not initiate this password reset, please contact our support
team immediately.</p>
```

```
<p>For security reasons, we recommend that you:</p>
```

```
<ul>
```

```
<li>Use a strong, unique password</li>
```

```
<li>Enable two-factor authentication if available</li>
```

```
<li>Avoid using the same password across multiple sites</li>
```

```
</ul>
```

```
<p>Thank you for helping us keep your account secure.</p>
```

```
<p>Best regards,<br>Your App Team</p>
```

```
</div>
```

```
<div style="text-align: center; margin-top: 20px; color: #888; font-size:
0.8em;">
```

```
<p>This is an automated message, please do not reply to this email.</p>
```

```
</div>
```

```
</body>
```

```
</html>
```

```
`;
```

```
export const PASSWORD_RESET_REQUEST_TEMPLATE = `
```

```

</div>

<div style="text-align: center; margin-top: 20px; color: #888; font-size:
0.8em;">

  <p>This is an automated message, please do not reply to this email.</p>

</div>

</body>

</html>

`;

```

(ismember.js)

```

import Community from '../models/communityModel.js';

export const isMember = async (req, res, next) => {

  const { communityId } = req.params;

  try {

    const community = await Community.findById(communityId);

    if (!community) {

      return res.status(404).json({ message: "Community not found" });

    }

    if (!community.members.includes(req.user._id)) {

      return res.status(403).json({ message: "Access denied, you are not a
member of this community" });

    }

    next();

  } catch (error) {

    res.status(500).json({ message: "Server error" });

  }
}

```

```
};
```

(verifytoken.js)

```
import jwt from 'jsonwebtoken';
```

```
export const verifyToken=async(req,res,next)=>{
```

```
    const token=req.cookies.authToken;
```

```
    if(!token) {
```

```
        console.log("token not provided");
```

```
        return res.status(401).json({ success: false, message:"Token not provided"})
```

```
    }
```

```
    try {
```

```
        const decoded=jwt.verify(token,process.env.JWT_SECRET_KEY)
```

```
        if(!decoded) {
```

```
            return res.status(401).json({ success: false, message:"Token not valid"})
```

```
        }
```

```
        req.userId=decoded.userId;
```

```
        next();
```

```
    } catch (error) {
```

```
        console.log("error verifying token", error);
```

```
        res.status(500).json({ success: false, message:"Internal server error"})
```

```
    }
```

```
}
```

(communitymodel.js)

```
import mongoose from "mongoose";

const communitySchema = mongoose.Schema(

  {

    name: {

      type: String,

      required: true,

      unique: true,

    },

    description: {

      type: String,

      required: true,

    },

    code: {

      type: String,

      required: true,

      unique: true,

    },

    admin: {

      type: mongoose.Schema.Types.ObjectId,

      ref: "User",

    },

    members: [

      {
```

```

    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
  },
],
messages: [
  {
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
    },
    message: String,
    createdAt: { type: Date, default: Date.now }
  }
],
},
{ timestamps: true }
);

const Community = mongoose.model("Community", communitySchema);

export default Community;

```

(usermodel.js)

```

import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },

```

```

router.post('/signup',signup)

router.post('/login',login)

router.post('/logout',logout)

router.post('/verify-email',verifyEmail)

router.post('/forgot-password',forgotPassword)

router.post('/reset-password/:token',resetPassword)

export default router;

```

(communityroute.js)

```

import express from 'express';

import {
  createCommunity, fetchCommunity, getCreatedCommunity, getJoinedCommunity,
  joinCommunity, getMessages, sendMessages } from
'../controller/communityController.js';

import { verifyToken } from '../middleware/verifyToken.js';

const router = express.Router();

router.post('/createCommunity',verifyToken,createCommunity)

router.post('/joinCommunity',verifyToken, joinCommunity)

router.get('/getJoinedCommunity',verifyToken,getJoinedCommunity)

router.get('/getCreatedCommunity',verifyToken,getCreatedCommunity)

router.get('/:name',verifyToken,fetchCommunity)

router.get('/getMessages/:name',verifyToken,getMessages)

router.post('/sendMessages',verifyToken,sendMessages)

export default router;

```


(userroute.js)

```
import express from 'express';

import { updateUser, userProfile } from '../controller/userProfileController.js';

const router= express.Router()

router.get('/userProfile',userProfile)

router.post('/updateUser',updateUser)

export default router;
```

(generatetokenandsetcookies.js)

```
import jwt from 'jsonwebtoken'

export const generateTokenAndSetCookie=async(res,userId)=>{

  const token=jwt.sign({ userId},process.env.JWT_SECRET_KEY,{

    expiresIn:'7d'

  })

  res.cookie("authToken",token,{

    httpOnly:true,

    secure:process.env.NODE_ENV==='production'

    ||process.env.NODE_ENV==='development',

    sameSite:"none",

    maxAge: 7*24*60*60*1000

  })

  return token

}
```

(.env)

PORT=2024

MONGO_URL=mongodb+srv://vishwas_23a:wnSC74rtg4NGsYRt@mydara
base.jyzdgdzw.mongodb.net/codeComAuth?retryWrites=true&w=majority&a
ppName=myDaraBase

JWT_SECRET_KEY=mysecretkey

NODE_ENV=development

MAILTRAP_TOKEN=9a98eb63279f6c7b1f531902a999b374

MY_EMAIL=codecom03@gmail.com

MY_EMAIL_PASS=yfimdlbjztbsxczk

RESET_URL=http://localhost:2024

(index.js)

```
import express from 'express';  
  
import dotenv from 'dotenv';  
  
import { createServer } from 'node:http';  
  
import { Server } from "socket.io";  
  
import authRoutes from './router/codeAuth.js'  
  
import userRoute from './router/userRoute.js'  
  
import communityRoute from './router/communityRoute.js'  
  
import { connectDb } from './db/connectDb.js';  
  
import cookieParser from 'cookie-parser';  
  
import cors from 'cors'  
  
import Community from './models/communityModel.js';  
  
dotenv.config();
```

```

io.on('connection', (socket) => {

  console.log('a user connected');

  socket.on("joinCommunity", (communityCode) => {

    socket.join(communityCode);

    console.log(`User ${socket.id} joined community: ${communityCode}`);

  });

  socket.on("sendMessage", async ({ code, userId, message }) => {

    try {

      const community = await Community.findOne({ code });

      if (!community) return;

      const newMessage = { user: userId, message };

      community.messages.push(newMessage);

      await community.save();

      io.to(code).emit("receiveMessage", { user: userId, message });

    } catch (error) {

      console.error("Error handling sendMessage:", error);

    }

  });

  socket.on("disconnect", () => {

    console.log("A user disconnected:", socket.id);

  });

});

export default io;

```

```
"jsonwebtoken": "^9.0.2",  
"mailtrap": "^3.4.0",  
"mongoose": "^8.13.0",  
"nodemailer": "^6.10.0",  
"nodemon": "^3.1.9",  
"socket.io": "^4.8.1"  
}  
}
```

CHAPTER-5

TESTING

Testing plays a crucial role in ensuring the quality, functionality, and reliability of the CodeCom platform. Throughout the development lifecycle, various testing techniques are employed to validate different aspects of the system. Unit testing is used to verify individual components like user registration, task creation, and code execution modules.

Integration testing ensures that these components work seamlessly together—for instance, how the task management system interacts with user profiles and the coding environment. Functional testing confirms that the platform meets all defined requirements such as community interactions, real-time messaging, and collaborative features.

Regression testing is conducted after updates to ensure new changes do not break existing functionality. User Acceptance Testing (UAT) involves real users testing the system to provide feedback on usability and satisfaction. Furthermore, performance testing is used to measure system responsiveness and stability under different loads, especially when many users are active.

5.1 UNIT TESTING

This involves testing individual components or modules of CodeCom to ensure they function as expected in isolation. Examples include testing user registration, login, code submission, community post creation, and chat message handling separately to verify their correctness.

Unit Testing is the foundational level of software testing where individual components or functions of the application are tested in isolation to ensure they work as intended.

In the development of **CodeCom**, unit testing was employed to verify the correctness of modules such as user authentication, code compilation logic, chat messaging services, and task creation features. Each of these units was tested

separately using tools like Jest or Mocha to check their behavior with various input values and conditions.

This process helped identify bugs early in development, ensuring each module operated correctly before integrating it into the broader system. By thoroughly testing individual units, CodeCom maintained a high level of code reliability and reduced the chances of defects during later stages of integration and deployment.

5.2 Integration Testing

Integration testing checks that different modules within CodeCom interact correctly. For instance, it verifies the proper flow between the user authentication system, coding module, task assignment, and real-time chat, ensuring that user data and community interactions are seamlessly connected. For example, integration tests ensured that once a user logged in, their coding tasks, chat history, and community posts loaded correctly from the database.

Tools like Postman and automated test suites were used to simulate real-time interactions between frontend components and backend APIs. This phase ensured that data flowed smoothly across services and that functionalities dependent on multiple components (like submitting code in a community challenge and seeing feedback) worked without any issues.

5.3 Functional Testing

Functional testing validates that CodeCom meets all functional requirements. This includes verifying that users can register, join communities, submit tasks or code, participate in discussions, and receive feedback or ratings—exactly as defined in the system specifications. This included ensuring that all core features such as user registration and login, task assignment, real-time chat, code submission, and community interaction operated as expected.

Each function was tested against its specification—such as verifying whether a user could successfully join a coding community, submit a code snippet for review, or participate in a chat discussion without errors.

Edge cases were also tested, like handling invalid inputs during registration or ensuring proper error messages were shown when a task submission failed. The focus during this testing phase was on user interactions and outcomes—checking not just if actions could be performed, but if the system's response to those actions was correct and user-friendly.

5.4 Regression Testing

Regression testing ensures that newly added features or changes (e.g., a new coding editor or chat enhancement) do not break or degrade existing features. It involves re-testing existing modules like login, dashboards, or community views to ensure system stability after updates.

As new features like task tracking or real-time chat enhancements were added, regression testing was essential to verify that previously working modules—such as user authentication, community management, and code submission—continued to operate without issues.

Each time a code change was implemented, automated and manual tests were rerun on core functionalities to detect any unintended side effects. As a result, CodeCom maintained its integrity and performance, even as the system grew in complexity and user base.

This process is particularly important in a collaborative coding environment like CodeCom, where even small changes in one module can affect other interconnected components such as code sharing, task management, or community interaction features.

5.5 User Acceptance Testing (UAT)

UAT involves real users interacting with the CodeCom platform to validate whether it meets their needs. Users such as students, mentors, or contributors test key features like submitting code, joining community discussions, and tracking progress through dashboards to confirm the platform's usability and effectiveness. Their feedback helped identify any overlooked usability issues, interface glitches, or unexpected behavior in real-time scenarios.

UAT was crucial for validating whether features like the dashboard, chat functionality, and code-sharing mechanisms aligned with user intentions. Based on their input, minor adjustments were made to improve clarity, responsiveness, and navigation flow. This testing phase ensured that CodeCom was not only technically sound but also intuitive and engaging, paving the way for a successful deployment and broader adoption.

5.6 Performance Testing

Performance testing examines how CodeCom behaves under different load conditions. This includes testing how quickly coding tasks are saved, how the chat handles multiple users, and how the system performs during peak hours. Scalability and responsiveness are evaluated to ensure the platform works smoothly for all users.

Performance Testing for CodeCom focused on evaluating the system's speed, scalability, and stability under varying load conditions. This phase was essential to ensure the platform could handle multiple users accessing and interacting with the system simultaneously—such as joining communities, uploading code, messaging, and assigning tasks.

This testing helped identify necessary optimizations in backend services, database queries, and API responses. As a result, performance tuning measures were implemented to ensure that CodeCom remains fast and responsive, even with increased user activity.

CHAPTER-6

CONCLUSION AND FUTURE SCOPE

Conclusion

The CodeCom platform stands as a dynamic and collaborative community-based learning environment aimed at improving coding skills and fostering peer-to-peer interaction. By combining modern web technologies with an intuitive interface and community-driven features, CodeCom enables users to learn, code, share, and grow together. The CodeCom project successfully achieves its goal of creating a collaborative and interactive platform for coders, learners, and tech enthusiasts.

Through its user-friendly interface, real-time messaging, task management, and community-driven coding features, CodeCom fosters an engaging environment where users can learn, share, and grow together. The integration of modern technologies like React.js, Node.js, and MongoDB ensures scalability and robust performance. With a focus on clean UI/UX and essential functionality, CodeCom stands out as a productive space for collaborative software development.

It not only simplifies the way users interact and share ideas but also promotes a culture of continuous learning and teamwork. The project lays a strong foundation for future enhancements, with the potential to expand into an even more dynamic and versatile coding community platform. The development of CodeCom marks a significant step toward fostering a collaborative coding environment tailored for learners, developers, and community contributors.

Designed with a modern tech stack and a user-first approach, the platform enables smooth interaction between users through features like community discussions, real-time messaging, coding collaboration, and task assignments.

Key Benefits of the CodeCom project:

- **Collaborative Learning Environment:**
CodeCom enables users to engage in shared coding projects, promoting teamwork and peer-to-peer learning among community members.
- **Community Engagement:**
The platform fosters active participation through discussion forums, real-time chat, and community-based project contributions, encouraging users to share ideas and support each other.
- **Task Management and Productivity:**
With built-in task assignment and tracking features, users can efficiently manage coding projects, track progress, and stay organized.
- **User-Friendly Interface:**
A clean and intuitive UI ensures easy navigation, making the platform accessible for beginners while still being powerful for experienced developers.
- **Scalability and Flexibility:**
Built with scalable cloud technologies like MongoDB Atlas and Vercel, CodeCom can grow with its user base and adapt to new features as needed.
- **Secure and Efficient Backend:**
Using Node.js and Express.js, the backend is designed to handle data securely and respond quickly to user interactions, ensuring a smooth experience.
- **Real-Time Communication:**
Instant messaging features enhance collaboration, allowing users to discuss issues, solve bugs, or brainstorm ideas on the go.
- **Inclusive Development Platform:**
CodeCom supports users from various backgrounds—students, hobbyists, and professionals—by offering a supportive and dynamic learning space.

Future Scope

The future scope of the CodeCom project is both vast and promising, with several enhancements and expansions planned to elevate user experience and platform capabilities. One of the key developments is the integration of AI-based code assistance, enabling features like auto code completion, real-time bug detection, and intelligent suggestions to streamline the coding process.

The introduction of gamification elements—such as rewards, badges, and leaderboards—can enhance user engagement and motivation within the community. Expanding the platform to mobile applications will make CodeCom more accessible, allowing users to collaborate from anywhere. Additionally, multilingual support will cater to developers working with various programming languages and frameworks.

Integration with version control systems like GitHub will facilitate better project management and collaboration. An advanced analytics dashboard can be introduced to monitor user activity and track project progress, helping users stay organized. To ensure safety and trust, future updates will include stronger security measures such as two-factor authentication and role-based access. Lastly, transitioning CodeCom into an open-source initiative could attract a global community of developers to contribute, innovate, and expand the platform's functionality.

1. **Gamification and Skill Ranking:**

Introducing gamified elements like leaderboards, badges, and skill-level ranking can further boost user motivation and participation.

2. **Integration with Version Control Tools (e.g., GitHub):**

Allowing users to sync their coding tasks and projects with GitHub or GitLab will enhance collaborative workflows and project visibility.

3. **Advanced Task Recommendation System:**

AI-based recommendation engines can be added to suggest relevant coding tasks or communities to users based on their interests and activity.

4. **Mobile Application:**

Developing a mobile-friendly version of CodeCom would extend its accessibility, enabling users to stay connected and contribute on the go.

5. **Wider Community Support:**

Future versions can accommodate different user roles like mentors, educators, and professionals from various domains to expand learning and networking possibilities.

6. **Multilingual Code Support:**

Future versions can support additional programming languages and frameworks, making the platform more inclusive for developers working on diverse tech stacks.

7. **Enhanced Security Features:**

With the growth of the platform, stronger security protocols such as two-factor authentication, encrypted messaging, and role-based access control can be implemented to ensure data safety.

CHAPTER-7

BIBLIOGRAPHY

- Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.
- Wenger, E. (1998). *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press.
- Guo, P. J. (2013). *Online Learning and Real-Time Coding: Effects of Live Programming in Education*. In Proceedings of ACM Symposium on Learning Technologies.
- Johnson, D. W., Johnson, R. T., & Smith, K. A. (2014). *Cooperative Learning: Improving University Instruction by Basing Practice on Validated Theory*. Journal on Excellence in College Teaching.
- Cockburn, A. (2002). *Agile Software Development: The Cooperative Game*. Addison-Wesley.
- Stack Overflow Developer Survey. (2023). *Insights on Collaboration and Learning in Programming*. Retrieved from: <https://insights.stackoverflow.com/>
- GitHub Docs. (n.d.). *Collaboration and Version Control for Developers*. Retrieved from: <https://docs.github.com>
- Mozilla Developer Network (MDN). (n.d.). *WebSockets – Real-Time Communication*. Retrieved from: <https://developer.mozilla.org>
- Firebase Documentation. (n.d.). *Real-Time Database and Authentication Services*. Retrieved from: <https://firebase.google.com/docs>
- React Documentation. (n.d.). *Building Modern User Interfaces with React*. Retrieved from: <https://react.dev>

