

# **ScrollIn (Blogging Website)**

**A PROJECT REPORT  
for  
Mini Project (KCA353)  
Session (2024-25)**

**Submitted by**

**Adarsh Dubey  
University Roll No  
2300290140011  
Avinashkumar Singh  
University Roll No  
2300290140183**

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

## **MASTER OF COMPUTER APPLICATION**

**Under the  
Supervision of  
Dr. Ankit Verma  
(Associate Professor)**



**Submitted to**

**DEPARTMENT OF COMPUTER  
APPLICATIONS  
KIET Group of Institutions, Ghaziabad  
Uttar Pradesh-201206**

**DECEMBER 2024**

## **CERTIFICATE**

Certified that **Adarsh Dubey (2300290140011)**, **Avinashkumar Singh (2300290140183)** have carried out the project work having “**ScrollIn**” (**Mini Project KCA353**) 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 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:**

**Adarsh Dubey (2300290140011)**

**Avinashkumar Singh (2300290140183)**

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

Date:

**Dr. Ankit Verma**  
**Associate Professor**  
**Department of Computer Applications**  
**KIET Group of Institutions, Ghaziabad**

**Dr. Arun Kumar Tripathi**  
**Dean**  
**Department of Computer Applications**  
**KIET Group of Institutions, Ghaziabad**

**ScrollIn**  
**(Adarsh Dubey)**  
**(Avinashkumar Singh)**

**ABSTRACT**

In today's digital era, the need for efficient, scalable, and user-friendly web applications is ever-increasing. *ScrollIn* is a dynamic web application built on the MERN stack (MongoDB, Express.js, React.js, Node.js), designed to provide seamless content management and user interaction. It addresses the limitations of traditional systems, such as poor scalability and inefficient user experience, through a combination of modern technologies and robust architecture.

The primary objective of *ScrollIn* is to enhance user experience while enabling administrators to efficiently manage content, user data, and system configurations. It offers secure user authentication using JSON Web Tokens (JWT), real-time content updates, and administrative tools for streamlined operations. The modular architecture allows flexibility and easy integration of future features.

*ScrollIn* leverages React.js for its dynamic, responsive frontend, MongoDB for a scalable NoSQL database, and Node.js with Express.js for a fast and efficient backend. This ensures high performance under heavy loads and seamless data communication between components. The application's features include user management, secure login, dynamic content interaction, and a scalable structure to accommodate growing demands.

*ScrollIn* is a versatile, cost-effective solution tailored for dynamic content management. Its robust design, modern technology stack, and focus on performance and security make it an ideal choice for individuals and organizations. With potential for future enhancements, *ScrollIn* is positioned as a cutting-edge platform capable of adapting to evolving digital requirements.

**Keywords :-** Web Application, MERN Stack, Content Management, Dynamic Interface, User Authentication, Scalability, MongoDB, React.js, Node.js, Express.js, Real-Time Updates, JWT Security.

## ACKNOWLEDGEMENTS

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

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi**, 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 kind 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.

**Adarsh Dubey**

**Avinashkumar Singh**

## TABLE OF CONTENTS

	Certificate	i
	Abstract	ii
	Acknowledgements	iii
	Table of Contents	iv-v
	List of Figures	vi
1	Introduction	1-2
	1.1 Background	1
	1.2 Project overview	1
	1.3 Objective	1
	1.4 Key features	2
	1.5 Scope of the project	2
2	Problem identification & feasibility study	3-4
	2.1 Problem Identification	3
	2.2 Feasibility Study	4
3	Requirement Analysis	5
	3.1 Introduction	5
	3.2 Functional Requirements	5
	3.3 Non-Functional Requirements	5
4	Hardware and Software Specification	6
	4.1 Hardware Specification	6
	4.2 Software Specification	6
5	Choice of Tools & Technology	7-9
	5.1 React	7
	5.2 MongoDB	7
	5.3 Data Flow Diagram	8-9

6	ER-Diagram	10
	6.1 Entity-relationship model	10
7	Database	11
8	Form Design	12-13
	8.1 Login	12
	8.2 Signup	12
	8.3 Home Page	13
	8.4 All Authors	13
9	Coding	14-44
10	Testing	45-46
	10.1 Introduction	45
	10.2 Types of Testing	46

Future Scope and Further Enhancement of the Project

Conclusion & References

## LIST OF FIGURES

Figure No.	Name of Figure	Page No.
5.1	0 Level DFD	9
5.2	1 Level DFD	9
6.1	Entity-relationship Model	10
7.1	Users	11
7.2	Blogs	11
8.1	Sign-up	12
8.2	Login	12
8.3	Home Page	13
8.4	All Authors	13

# CHAPTER 1

## INTRODUCTION

### 1.1 Background

The internet has evolved from static websites to dynamic platforms capable of serving interactive content tailored to users' needs. In this context, web applications have become an essential part of our daily lives. They provide a seamless experience for activities ranging from shopping and entertainment to content consumption and management. However, there is a growing demand for systems that not only deliver dynamic content but also ensure scalability, responsiveness, and security. *ScrollIn* is designed to address these needs by leveraging the latest web technologies. Its goal is to provide a robust platform that caters to both end-users and administrators, offering an intuitive interface, reliable performance, and secure data management.

### 1.2 Project Overview

*ScrollIn* is a modern web application that simplifies the process of browsing, managing, and interacting with digital content. Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), it integrates a highly responsive frontend with a powerful backend. The platform is structured around a modular architecture, allowing users to navigate content effortlessly while enabling administrators to maintain control over the system. Whether it's a personal blog, an e-commerce site, or a content management system, *ScrollIn* adapts to diverse use cases, making it versatile and scalable.

### 1.3 Objective

The primary objective of *ScrollIn* is to create a dynamic platform that bridges the gap between user experience and administrative control. It aims to:

- Enhance user interaction through a responsive and feature-rich interface.
- Provide administrators with tools for efficient content and user management.
- Ensure data security and system reliability for all stakeholders.



## 1.4 Key Features

- **Secure Authentication:** Implements JWT-based authentication for secure user access.
- **Content Management:** Allows administrators to organize, update, and control content dynamically.
- **User-Friendly Design:** Features a responsive and visually appealing UI using React.js.
- **Scalable Architecture:** Built to handle a growing user base without compromising performance.
- **Real-Time Updates:** Provides live feedback and data synchronization.

## 1.5 Scope of the project

- *ScrollIn* has a wide application scope, ranging from small-scale personal projects to large-scale enterprise solutions.
- Its modularity and scalability allow for integration with other tools, such as analytics and AI-powered services.
- The system's flexibility ensures that it can evolve alongside user requirements, incorporating advanced features and technologies in the future.

## CHAPTER 2

### PROBLEM IDENTIFICATION & FEASIBILITY STUDY

#### 2.1 Problem Identification

Many existing content management systems face challenges such as limited scalability, poor user experience, and complex administrative processes. For end-users, navigating poorly designed interfaces or experiencing slow performance can be frustrating. On the other hand, administrators struggle with outdated tools that lack flexibility and efficiency. *ScrollIn* is designed to overcome these challenges by integrating modern technologies, focusing on both user satisfaction and administrative ease.

#### 2.2 Feasibility Study

##### 2.2.1 Technical Feasibility

The project utilizes the MERN stack, a combination of MongoDB, Express.js, React.js, and Node.js. These technologies are well-suited for building high-performance and scalable web applications. The use of open-source tools reduces dependency on proprietary software, ensuring flexibility and community support.

##### 2.2.2 Operational Feasibility

The intuitive design of *ScrollIn* ensures ease of use for both end-users and administrators. Minimal training is required for administrators to manage the system. Its responsive interface caters to users on various devices, enhancing accessibility and usability.

### **2.2.3 Economic Feasibility**

By leveraging open-source tools and technologies, the development and maintenance costs of *ScrollIn* are significantly reduced. This economic advantage allows the system to deliver high-quality features while remaining cost-effective

## CHAPTER 3

### REQUIREMENT ANALYSIS

#### 3.1 Introduction

The development of *ScrollIn* is guided by a detailed analysis of its functional and non-functional requirements. This ensures that the system meets user expectations and performs reliably under various conditions.

#### 3.2 Functional Requirements

1. **Smooth Scrolling:** Seamless user experience with adjustable speed.
2. **Infinite Scroll:** Dynamic content loading on scroll.
3. **Lazy Loading:** Load media only when in the viewport.
4. **Scroll Indicator:** Visual progress bar for scroll position.
5. **Touch and Mouse Support:** Compatibility with gestures and devices.

#### 3.3 NON-FUNCTIONAL REQUIREMENTS

- **Performance:** The application should maintain optimal performance with at least 100 concurrent users.
- **Security:** Implements robust security measures, including data encryption and secure authentication.
- **Scalability:** Designed to accommodate increasing traffic and expanding functionality.
- **Usability:** Offers a user-friendly interface that enhances navigation and interaction.

## CHAPTER 4

### HARDWARE & SOFTWARE SPECIFICATION

#### 4.1 Hardware Specification

- **Processor:** Intel i5 (or equivalent) with multi-core support.
- **Memory:** Minimum 4 GB of RAM to support smooth development and runtime environments.
- **Storage:** SSD with a capacity of at least 256 GB for faster data access and application deployment.

#### 4.2 Software Specification

- **Operating System:** Windows 11 or Ubuntu 20.04.
- **Development Tools:** Visual Studio Code for code editing, Node.js for backend development, MongoDB for database management, and React for frontend development.

## **CHAPTER 5**

### **CHOICE OF TOOLS & TECHNOLOGY**

#### **6.1 React**

React is a JavaScript library used for building user interfaces. React allows developers to describe the UI's appearance at any point in time, and it automatically updates and renders the right components when the data changes.

React organizes the UI into reusable components, making it easier to manage and maintain complex applications. React uses a virtual DOM to optimize and update the actual DOM efficiently, improving performance by minimizing unnecessary re-rendering.

React uses JSX, a syntax extension that looks similar to XML or HTML, to describe the structure of UI components in a more concise and readable way. React components can manage their internal state, and data can be passed to components through props, allowing for dynamic and interactive UIs

#### **6.2 MongoDB**

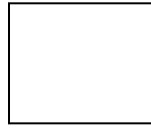
MongoDB is a NoSQL (non-relational) database management system that stores data in a flexible, JSON-like format known as BSON (Binary JSON). It is designed to handle large volumes of unstructured or semi-structured data, making it particularly suitable for applications with evolving and dynamic data requirements.

#### **6.3 Data Flow Diagram**

The data flow diagram shows the flow of data within any system. It is an important tool for designing phase of software engineering. Larry Constantine first developed it. It represents graphical view of flow of data. It's also known as BUBBLE CHART. The purpose of DFD is major transformation that will become in system design symbols used in DFD: -

In the DFD, four symbols are used and they are as follows.

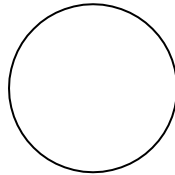
1. A square defines a source (originator) or destination of system data.



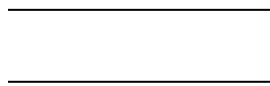
2. An arrow identifies data flow-data in motion. It is a pipeline through which information flows.



3. A circle or a "bubble" (Some people use an oval bubble) represents a process that transfers incoming data flows into outgoing data flows.



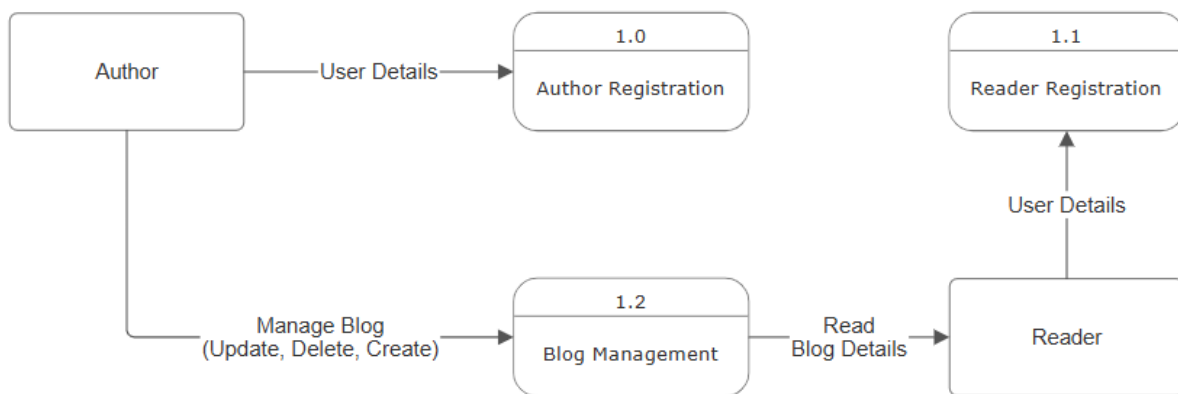
4. An open rectangle is a data store-data at rest, or a temporary repository of data.





### **0 Level DFD**

Fig 6.1



### **1<sup>st</sup> Level DFD**

Fig 6.2

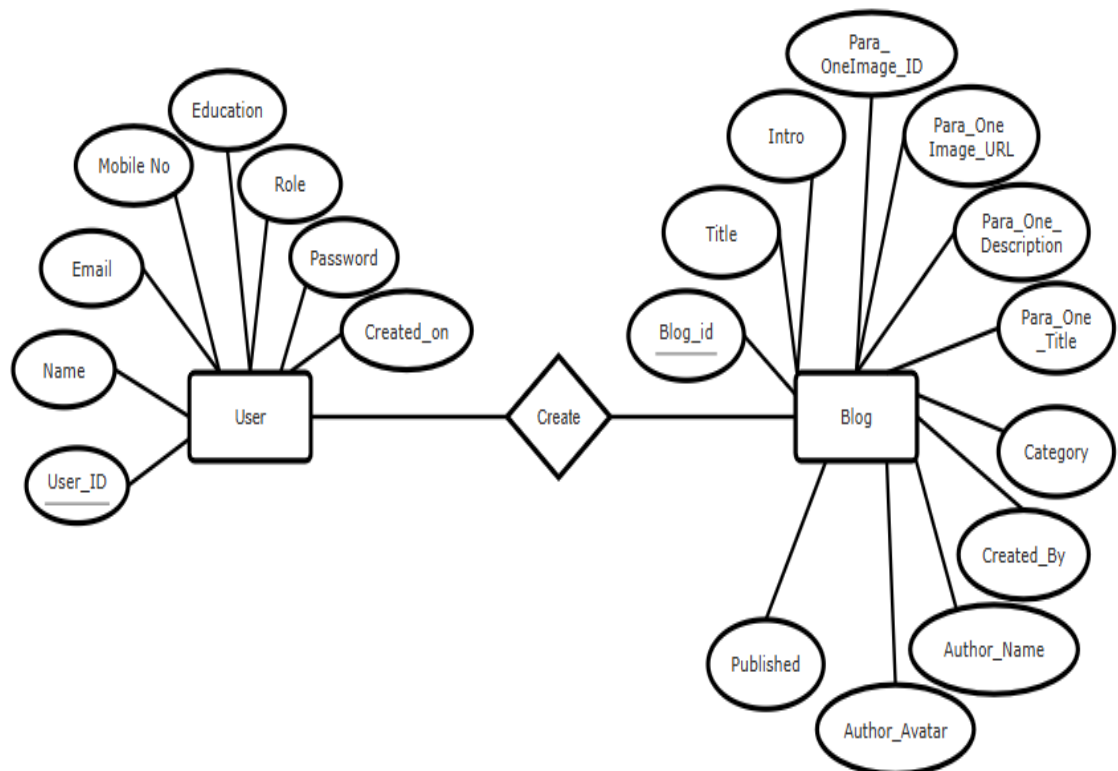


## CHAPTER 6

### ER-DIAGRAM

#### 7.1 Entity-relationship model: -

The entity-relationship model or entity-relationship diagram (ERD) is a data model or diagram for high-level descriptions of conceptual data model, and it provides a graphical notation for representing such data models in the form of entity- relationship diagrams.

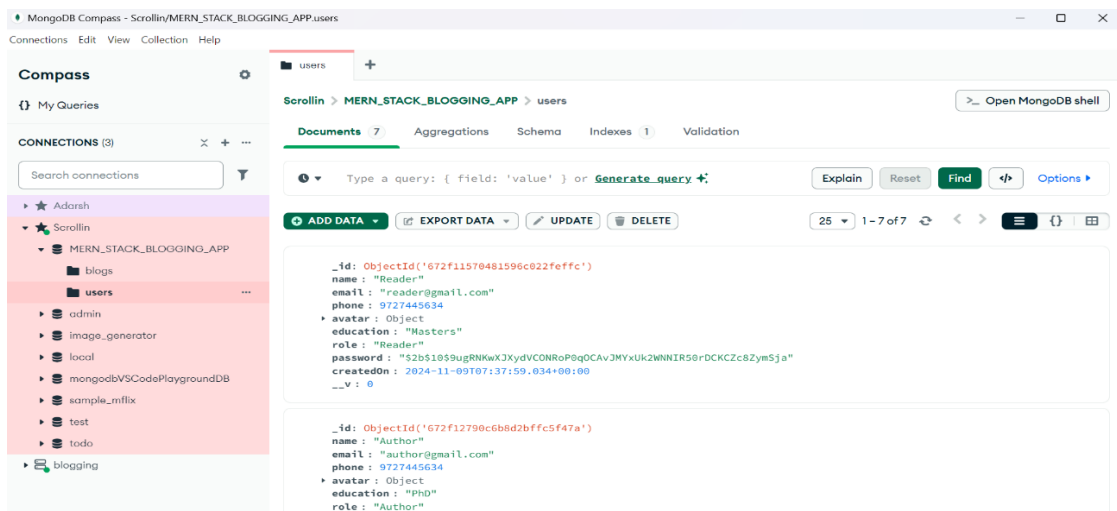


**ER Diagram**

Fig 7.

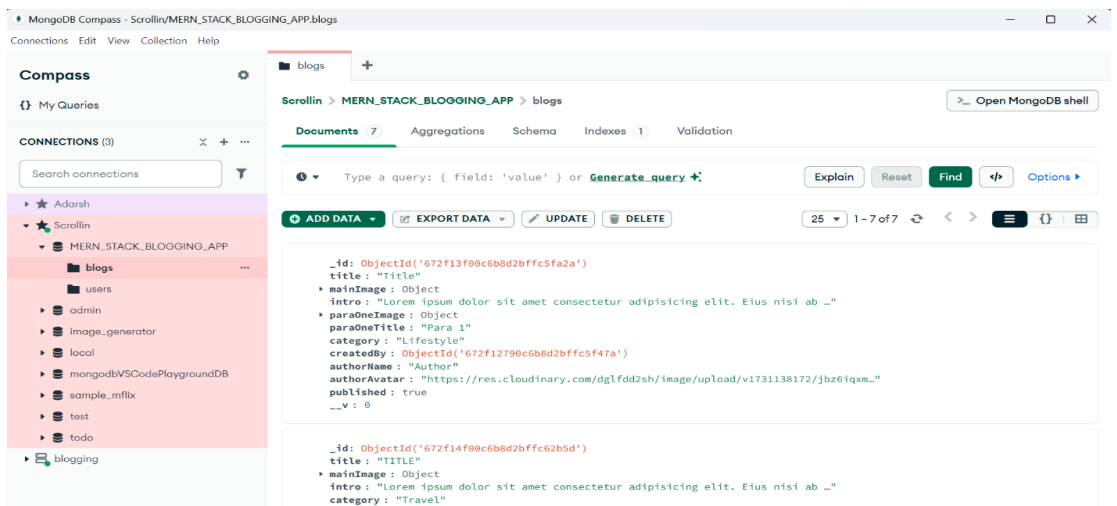
# CHAPTER 7

## DATABASE



User Database

Fig 8.1



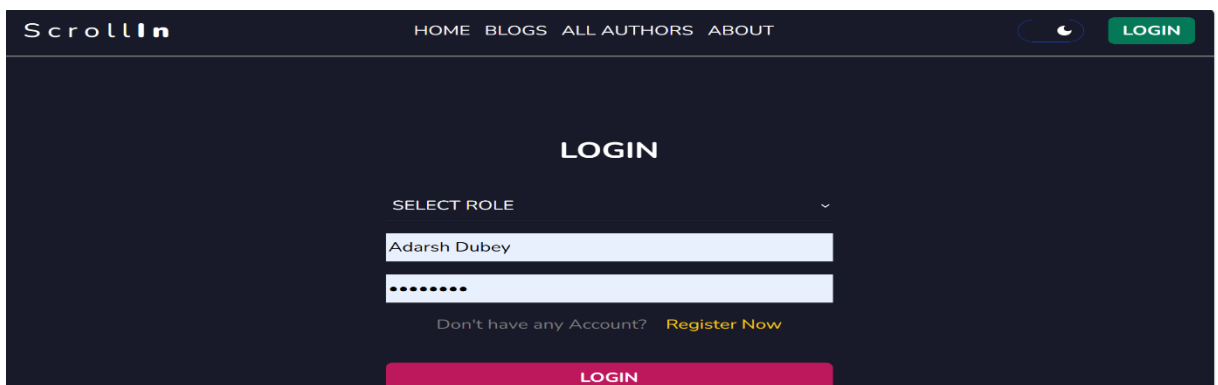
Blogs Database

Fig 8.2

## CHAPTER 8

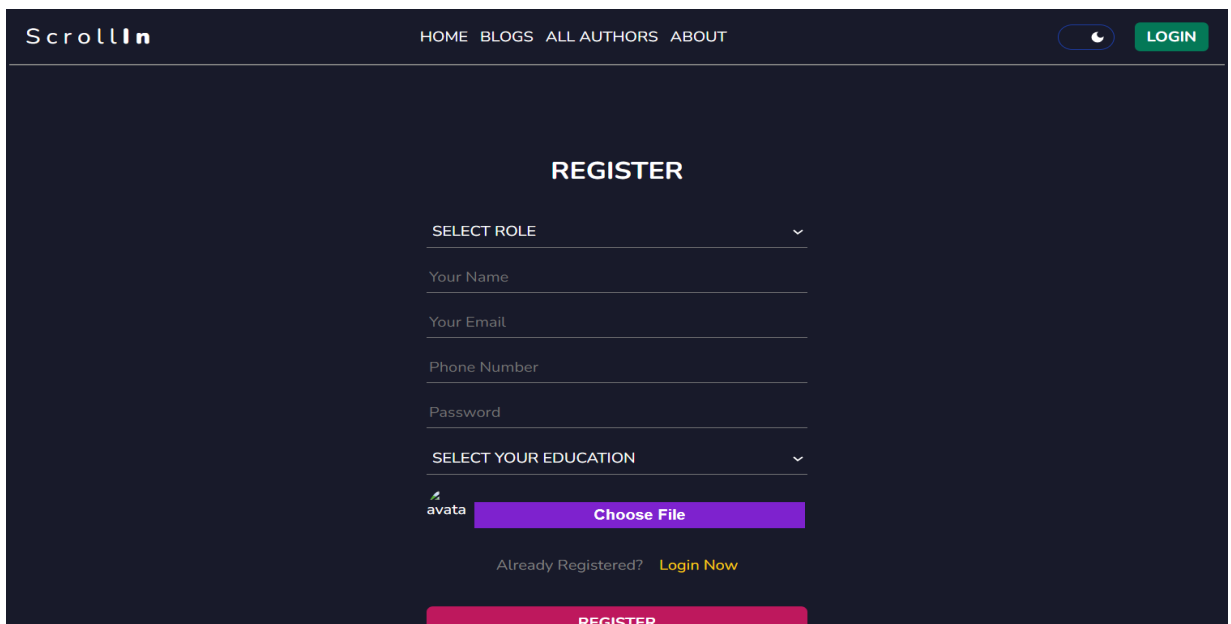
### FORM DESIGN

#### Login Form :-



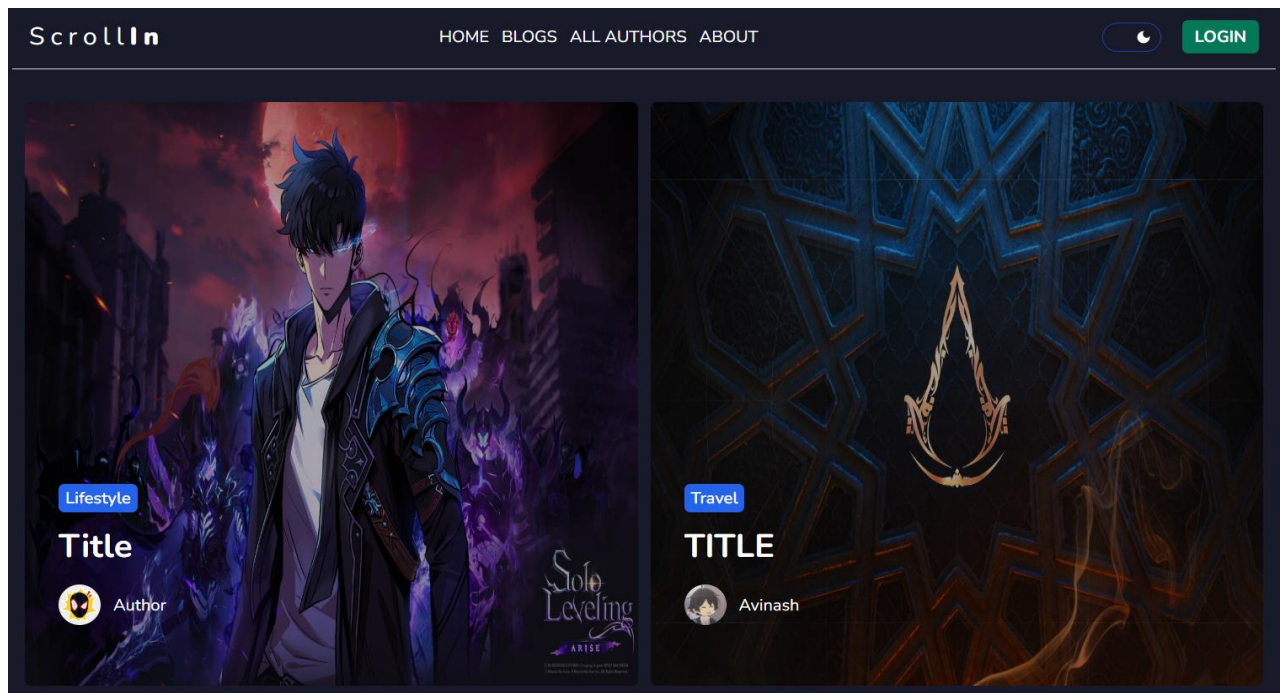
The screenshot shows the 'ScrollIn' website's login page. The header includes the brand name 'ScrollIn', navigation links for 'HOME', 'BLOGS', 'ALL AUTHORS', and 'ABOUT', a dark mode toggle, and a 'LOGIN' button. The main content area is titled 'LOGIN'. It features a 'SELECT ROLE' dropdown menu, a text input field containing 'Adarsh Dubey', and a password input field with masked characters. Below the password field is a link that says 'Don't have any Account? Register Now'. At the bottom of the form is a large pink 'LOGIN' button.

#### SignUp Form :-

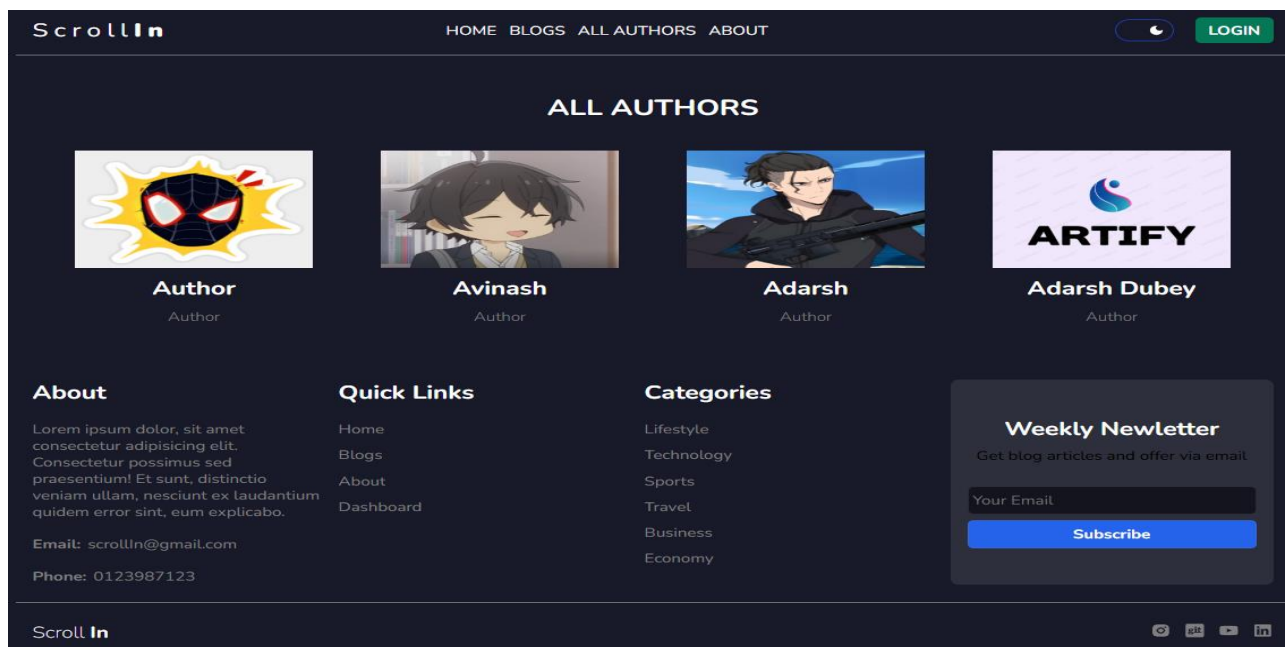


The screenshot shows the 'ScrollIn' website's register page. The header is identical to the login page. The main content area is titled 'REGISTER'. It includes a 'SELECT ROLE' dropdown, followed by text input fields for 'Your Name', 'Your Email', 'Phone Number', and 'Password'. Below these is a 'SELECT YOUR EDUCATION' dropdown. There is an 'avata' logo and a 'Choose File' button for profile picture selection. A link 'Already Registered? Login Now' is positioned above the final pink 'REGISTER' button.

## Home Page :-



## All Authors :-



## CHAPTER 9

### CODING

#### App.Jsx :-

```
import React, { useContext, useEffect } from "react";
import "./App.css";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Home from "../src/components/pages/Home";
import About from "../src/components/pages/About";
import Blogs from "../src/components/pages/Blogs";
import SingleBlog from "../src/components/pages/SingleBlog";
import Navbar from "../src/components/layout/Navbar";
import Footer from "../src/components/layout/Footer";
import { Toaster } from "react-hot-toast";
import Dashboard from "../components/pages/Dashboard";
import Register from "../components/pages/Register";
import Login from "../components/pages/Login";
import AllAuthors from "../components/pages/AllAuthors";
import { Context } from "../main";
import axios from "axios";
import UpdateBlog from "../components/pages/UpdateBlog";

const App = () => {
  const { setUser, isAuthenticated, setIsAuthenticated, user, setBlogs } =
    useContext(Context);
  useEffect(() => {
    const fetchUser = async () => {
      try {
        const { data } = await axios.get(
          "http://localhost:4000/api/v1/user/myprofile",
```

```

        {
            withCredentials: true,
        }
    );
    setUser(data.user);
    setIsAuthenticated(true);
} catch (error) {
    console.log(error);
    setIsAuthenticated(false);
    setUser({});
}
};

const fetchBlogs = async () => {
    try {
        const { data } = await axios.get(
            "http://localhost:4000/api/v1/blog/all",
            { withCredentials: true }
        );
        setBlogs(data.allBlogs);
    } catch (error) {
        setBlogs([]);
    }
};

fetchUser();
fetchBlogs();
}, [isAuthenticated, user]);

return (
    <>
    <BrowserRouter>
    <Navbar />
    <Routes>
    <Route path="/" element={ <Home /> } />
    <Route path="/register" element={ <Register /> } />
    <Route path="/login" element={ <Login /> } />
    <Route path="/blogs" element={ <Blogs /> } />
    <Route path="/blog/:id" element={ <SingleBlog /> } />

```

```

    <Route path="/about" element={<About />} />
    <Route path="/authors" element={<AllAuthors />} />
    <Route path="/dashboard" element={<Dashboard />} />
    <Route path="/blog/update/:id" element={<UpdateBlog />} />
  </Routes>
  <Footer />
  <Toaster />
</BrowserRouter>
</>
);
};
export default App;

```

### **Home.Jsx :-**

```

import React, { useContext, useState } from "react";
import LatestBlogs from "../miniComponents/LatestBlogs";
import HeroSection from "../miniComponents/HeroSection";
import TrendingBlogs from "../miniComponents/TrendingBlogs";
import PopularAuthors from "../miniComponents/PopularAuthors";
import { Context } from "../main";

const Home = () => {
  const { mode, blogs } = useContext(Context);
  const filteredBlogs = blogs.slice(0, 6);
  return (
    <>
      <article className={mode === "dark" ? "dark-bg" : "light-bg"}>
        <HeroSection />
        <TrendingBlogs />
        <LatestBlogs heading="Latest Blogs" blogs={filteredBlogs} />
        <PopularAuthors />
      </article>
    </>
  );
};

```

```
export default Home;
```

### **Login.Jsx :-**

```
import React, { useContext, useState } from "react";
import { Link, Navigate, useNavigate } from "react-router-dom";
import { Context } from "../main";
import axios from "axios";
import toast from "react-hot-toast";
```

```
const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [role, setRole] = useState("");
  const { mode, isAuthenticated } = useContext(Context);
  const navigateTo = useNavigate();
```

```
  const handleLogin = async (e) => {
    e.preventDefault();
    await axios
      .post(
        "http://localhost:4000/api/v1/user/login",
        { email, password, role },
        {
          withCredentials: true,
          headers: { "Content-Type": "application/json" },
        }
      )
      .then((res) => {
```



```

        toast.success(res.data.message);

        setEmail("");
        setPassword("");
        setRole("");
        navigateTo("/");
    })
    .catch((error) => {
        toast.error(error.response.data.message);
    });
};

if(isAuthenticated){
    return <Navigate to={'/'}/>
}

return (
    <article className={mode === "dark" ? "dark-bg" : "light-bg"}>
        <section className="auth-form">
            <form onSubmit={handleLogin}>
                <h1>LOGIN</h1>
                <div>
                    <select value={role} onChange={(e) => setRole(e.target.value)}>
                        <option value="">SELECT ROLE</option>
                        <option value="Reader">READER</option>
                        <option value="Author">AUTHOR</option>
                    </select>
                </div>
                <div>
                    <input
                        type="email"
                        placeholder="Your Email"

```

```

        value={email}
        onChange={(e) => setEmail(e.target.value)}
    />
</div>
<div>
    <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
    />
</div>
<p>
    Don't have any Account? <Link to={"/register"}>Register Now</Link>
</p>
<button className="submit-btn" type="submit">
    LOGIN
</button>
</form>
</section>
</article>
);
};

export default Login;

```

### **Login.jsx :-**

```

import React, { useContext, useState } from "react";
import { Link, Navigate, useNavigate } from "react-router-dom";
import { Context } from "../../main";

```

```

import axios from "axios";
import toast from "react-hot-toast";

const Register = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [password, setPassword] = useState("");
  const [role, setRole] = useState("");
  const [education, setEducation] = useState("");
  const [avatar, setAvatar] = useState("");
  const [avatarPreview, setAvatarPreview] = useState("");

  const changeAvatarHandler = (e) => {
    const file = e.target.files[0];
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => {
      setAvatarPreview(reader.result);
      setAvatar(file);
    };
  };

  const { mode, isAuthenticated } = useContext(Context);

  const navigateTo = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();
    const formData = new FormData();
    formData.append("name", name);
    formData.append("email", email);
    formData.append("phone", phone);
    formData.append("password", password);
    formData.append("education", education);
    formData.append("role", role);
    formData.append("avatar", avatar);
  };

```

```

try {
  const { data } = await axios.post(
    "http://localhost:4000/api/v1/user/register",
    formData,
    {
      withCredentials: true,
      headers: {
        "Content-Type": "multipart/form-data",
      },
    }
  );
  setName("");
  setEmail("");
  setEducation("");
  setPassword("");
  setPhone("");
  setRole("");
  setAvatar("");
  setAvatarPreview("");
  toast.success(data.message);
  navigateTo("/");
} catch (error) {
  toast.error(error.response.data.message);
}
};

if (isAuthenticated) {
  return <Navigate to={"/"} />;
}

return (
  <article className={mode === "dark" ? "dark-bg" : "light-bg"}>
    <section className="auth-form">
      <form onSubmit={handleRegister}>
        <h1>REGISTER</h1>
        <select value={role} onChange={(e) => setRole(e.target.value)}>

```

```

    <option value="">SELECT ROLE</option>
    <option value="Reader">READER</option>
    <option value="Author">AUTHOR</option>
</select>
<div>
  <input
    type="text"
    placeholder="Your Name"
    value={ name }
    onChange={ (e) => setName(e.target.value) }
  />
</div>
<div>
  <input
    type="email"
    placeholder="Your Email"
    value={ email }
    onChange={ (e) => setEmail(e.target.value) }
  />
</div>
<div>
  <input
    type="number"
    placeholder="Phone Number"
    value={ phone }
    onChange={ (e) => setPhone(e.target.value) }
  />
</div>
<div>
  <input
    type="password"
    placeholder="Password"
    value={ password }
    onChange={ (e) => setPassword(e.target.value) }
  />
</div>
<select

```

```

value={education}
onChange={ (e) => setEducation(e.target.value)}
>
<option value="">SELECT YOUR EDUCATION</option>
<option value="Matric">Matric</option>
<option value="Intermediate">Intermediate</option>
<option value="Graducation">Graducation</option>
<option value="Masters">Masters</option>
<option value="PhD">PhD</option>
</select>
<div
  style={{
    display: "flex",
    flexDirection: "row",
    alignItems: "center",
  }}
>
  <div className="avatar">
    <img
      src={avatarPreview ? `${avatarPreview}` : "/pic.jpg"}
      alt="avatar"
    />
  </div>
  <input
    type="file"
    onChange={changeAvatarHandler}
    className="avatar_input_tag"
    style={{ border: "none" }}
  />
</div>
<p>
  Already Registered? <Link to={"/login"}>Login Now</Link>
</p>
<button className="submit-btn" type="submit">
  REGISTER
</button>
</form>

```

```

    </section>
  </article>
);
};

export default Register;

```

### **AllAuthor.Jsx :-**

```

import React, { useContext, useEffect, useState } from "react";
import { Context } from "../main";
import axios from "axios";
import { BeatLoader, BounceLoader } from "react-spinners";

const AllAuthors = () => {
  const [authors, setAuthors] = useState([]);
  const { mode } = useContext(Context);
  useEffect(() => {
    const fetchAuthors = async () => {
      const { data } = await axios.get(
        "http://localhost:4000/api/v1/user/authors",
        { withCredentials: true }
      );
      setAuthors(data.authors);
    };
    fetchAuthors();
  }, []);

  return (
    <article
      className={
        mode === "dark" ? "dark-bg all-authors" : "light-bg all-authors"
      }
    >
    <h2>ALL AUTHORS</h2>

```

```

<div className="container">
  {authors && authors.length > 0 ? (
    authors.map((element) => {
      return (
        <div className="card" key={element._id}>
          { /* {authors && authors.avatar && ( */ }
            <img src={element.avatar.url} alt="author_avatar" />
          { /* ) } */ }
          <h3>{element.name}</h3>
          <p>{element.role}</p>
        </div>
      );
    })
  ) : (
    <BeatLoader color="gray" size={50} style={{ padding: "200px 0" }} />
  ) }
</div>
</article>
);
};

export default AllAuthors;

```

### **UpdateBlog.jsx :-**

```

import axios from "axios";
import React, { useContext, useEffect, useState } from "react";
import toast from "react-hot-toast";
import { useParams } from "react-router-dom";
import { Context } from "../main";

const UpdateBlog = () => {
  const { id } = useParams();
  const [category, setCategory] = useState("");
  const [mainImage, setMainImage] = useState("");
  const [intro, setIntro] = useState("");

```



```

const [paraOneTitle, setParaOneTitle] = useState("");
const [paraOneImage, setParaOneImage] = useState("");
const [paraOneDescription, setParaOneDescription] = useState("");
const [paraTwoTitle, setParaTwoTitle] = useState("");
const [paraTwoImage, setParaTwoImage] = useState("");
const [paraTwoDescription, setParaTwoDescription] = useState("");
const [paraThreeTitle, setParaThreeTitle] = useState("");
const [paraThreeImage, setParaThreeImage] = useState("");
const [paraThreeDescription, setParaThreeDescription] = useState("");
const [mainImagePreview, setMainImagePreview] = useState("");
const [paraOneImagePreview, setParaOneImagePreview] = useState("");
const [paraTwoImagePreview, setParaTwoImagePreview] = useState("");
const [paraThreeImagePreview, setParaThreeImagePreview] = useState("");
const [title, setTitle] = useState("");
const [published, setPublished] = useState(true);

useEffect(() => {
  const fetchBlog = async () => {
    try {
      const { data } = await axios.get(
        `http://localhost:4000/api/v1/blog/singleblog/${id}`,
        { withCredentials: true }
      );
      setTitle(data.blog.title);
      setIntro(data.blog.intro);
      setCategory(data.blog.category);
      setPublished(data.blog.published);
      setMainImage(data.blog.mainImage.url);
      setParaOneTitle(data.blog.paraOneTitle);
      setParaOneDescription(data.blog.paraOneDescription);
      data.blog.paraOneImage &&
      setParaOneImage(data.blog.paraOneImage.url);
      setParaTwoTitle(data.blog.paraTwoTitle);
      setParaTwoDescription(data.blog.paraTwoDescription);
      data.blog.paraTwoImage &&
      setParaTwoImage(data.blog.paraTwoImage.url);
      setParaThreeTitle(data.blog.paraThreeTitle);
    }
  };
  fetchBlog();
}, [id]);

```

```

        setParaThreeDescription(data.blog.paraThreeDescription);
        data.blog.paraThreeImage &&
            setParaThreeImage(data.blog.paraThreeImage.url);
    } catch (error) {
        console.log(error);
    }
};
fetchBlog();
}, [id]);
const handleUpdate = async (e) => {
    e.preventDefault();
    const updatedBlog = new FormData();
    updatedBlog.append("title", title);
    updatedBlog.append("intro", intro);
    updatedBlog.append("category", category);
    console.log(published);
    updatedBlog.append("published", published);
    updatedBlog.append("mainImage", mainImage);
    if (paraOneTitle && paraOneTitle.length !== 0) {
        updatedBlog.append("paraOneTitle", paraOneTitle);
    } else {
        updatedBlog.append("paraOneTitle", "");
    }
    if (paraOneDescription && paraOneDescription.length !== 0) {
        updatedBlog.append("paraOneDescription", paraOneDescription);
    } else {
        updatedBlog.append("paraOneDescription", "");
    }
    if (paraOneImage) {
        updatedBlog.append("paraOneImage", paraOneImage);
    }
    if (paraTwoTitle && paraTwoTitle.length !== 0) {
        updatedBlog.append("paraTwoTitle", paraTwoTitle);
    } else {
        updatedBlog.append("paraTwoTitle", "");
    }
    if (paraTwoDescription && paraTwoDescription.length !== 0) {

```

```

        updatedBlog.append("paraTwoDescription", paraTwoDescription);
    } else {
        updatedBlog.append("paraTwoDescription", "");
    }
    if (paraTwoImage) {
        updatedBlog.append("paraTwoImage", paraTwoImage);
    }
    if (paraThreeTitle && paraThreeTitle.length !== 0) {
        updatedBlog.append("paraThreeTitle", paraThreeTitle);
    } else {
        updatedBlog.append("paraThreeTitle", "");
    }
    if (paraThreeDescription && paraThreeDescription.length !== 0) {
        updatedBlog.append("paraThreeDescription", paraThreeDescription);
    } else {
        updatedBlog.append("paraThreeDescription", "");
    }
    if (paraThreeImage) {
        updatedBlog.append("paraThreeImage", paraThreeImage);
    }
    try {
        const { data } = await axios.put(
            `http://localhost:4000/api/v1/blog/update/${id}`,
            updatedBlog,
            { withCredentials: true }
        );
        toast.success(data.message);
    } catch (error) {
        toast.error(error.response.data.message);
    }
};

const mainImagePreviewHandler = (e) => {
    const file = e.target.files[0];
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => {
        setMainImagePreview(reader.result);
    }
};

```

```

        setMainImage(file);
    };
};

const paraOneImagePreviewHandler = (e) => {
    const file = e.target.files[0];
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => {
        setParaOneImagePreview(reader.result);
        setParaOneImage(file);
    };
};

const paraTwoImagePreviewHandler = (e) => {
    const file = e.target.files[0];
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => {
        setParaTwoImagePreview(reader.result);
        setParaTwoImage(file);
    };
};

const paraThreeImagePreviewHandler = (e) => {
    const file = e.target.files[0];
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => {
        setParaThreeImagePreview(reader.result);
        setParaThreeImage(file);
    };
};

const { mode } = useContext(Context);
return (
    <article className={mode === "dark" ? "dark-bg" : "light-bg"}>
        <section className="update-blog">
            <h3>UPDATE BLOG</h3>
            <form>
                <div className="category-box">

```

```

<label>Category</label>
<select
  value={category}
  onChange={(e) => setCategory(e.target.value)}
>
  <option value="">Select Blog Category</option>
  <option value="Lifestyle">Lifestyle</option>
  <option value="Technology">Technology</option>
  <option value="Sports">Sports</option>
  <option value="Travel">Travel</option>
  <option value="Business">Business</option>
  <option value="Economy">Economy</option>
</select>
</div>
<input
  type="text"
  placeholder="BLOG MAIN TITLE"
  value={title}
  onChange={(e) => setTitle(e.target.value)}
/>
<div
  style={{ display: "flex", flexDirection: "column", gap: "20px" }}
>
  <label>BLOG MAIN IMAGE</label>
  <img
    src={
      mainImagePreview
      ? `${mainImagePreview}` // If paraOneImage exists, use it directly
      : mainImage // Otherwise, use paraOneImagePreview
      ? `${mainImage}`
      : "/imgPL.webp" // If neither paraOneImage nor
paraOneImagePreview exists, use an empty string
    }
    alt="subParaOneImg"
  />
  <input
    type="file"

```

```

        onChange={mainImagePreviewHandler}
        style={{ border: "none" }}
    />
</div>
<textarea
    rows="25"
    className="intro"
    placeholder="BLOG INTRO..... (Must contain at least 250 characters!)"
    value={intro}
    onChange={(e) => setIntro(e.target.value)}
/>
<div className="sub-para">
    <input
        type="text"
        placeholder="Paragraph one title"
        value={
            paraOneTitle && paraOneTitle.length > 0 ? paraOneTitle : ""
        }
        onChange={(e) => setParaOneTitle(e.target.value)}
    />
    <img
        src={
            paraOneImagePreview
            ? `${paraOneImagePreview}` // If paraOneImage exists, use it
            directly
            : paraOneImage // Otherwise, use paraOneImagePreview
            ? `${paraOneImage}`
            : "/imgPL.webp" // If neither paraOneImage nor
paraOneImagePreview exists, use an empty string
        }
        alt="subParaOneImg"
    />
    <input
        type="file"
        onChange={paraOneImagePreviewHandler}
        style={{ border: "none" }}
    />

```

```

<textarea
  rows="10"
  placeholder="Blog First Sub Paragraph Comes Here..."
  value={
    paraOneDescription && paraOneDescription.length > 0
    ? paraOneDescription
    : ""
  }
  onChange={(e) => setParaOneDescription(e.target.value)}
/>
</div>
<div className="sub-para">
  <input
    type="text"
    placeholder="Paragraph two title"
    value={
      paraTwoTitle && paraTwoTitle.length > 0 ? paraTwoTitle : ""
    }
    onChange={(e) => setParaTwoTitle(e.target.value)}
  />
  <img
    src={
      paraTwoImagePreview
      ? `${paraTwoImagePreview}` // If paraOneImage exists, use it
      directly
      : paraTwoImage // Otherwise, use paraOneImagePreview
      ? `${paraTwoImage}`
      : "/imgPL.webp" // If neither paraOneImage nor
      paraOneImagePreview exists, use an empty string
    }
    alt="subParaOneImg"
  />
  <input
    type="file"
    onChange={paraTwoImagePreviewHandler}
    style={{ border: "none" }}
  />

```

```

<textarea
  rows="10"
  placeholder="Blog Second Sub Paragraph Comes Here..."
  value={
    paraTwoDescription && paraTwoDescription.length > 0
    ? paraTwoDescription
    : ""
  }
  onChange={(e) => setParaTwoDescription(e.target.value)}
/>
</div>
<div className="sub-para">
  <input
    type="text"
    placeholder="Paragraph three title"
    value={
      paraThreeTitle && paraThreeTitle.length > 0
      ? paraThreeTitle
      : ""
    }
    onChange={(e) => setParaThreeTitle(e.target.value)}
  />
  <img
    src={
      paraThreeImagePreview
      ? `${paraThreeImagePreview}` // If paraOneImage exists, use it
      directly
      : paraThreeImage // Otherwise, use paraOneImagePreview
      ? `${paraThreeImage}`
      : "/imgPL.webp" // If neither paraOneImage nor
      paraOneImagePreview exists, use an empty string
    }
    alt="subParaOneImg"
  />
  <input
    type="file"
    onChange={paraThreeImagePreviewHandler}

```



```

        style={{ border: "none" }}
      />
      <textarea
        rows="10"
        placeholder="Blog Third Sub Paragraph Comes Here..."
        value={
          paraThreeDescription && paraThreeDescription.length > 0
            ? paraThreeDescription
            : ""
        }
        onChange={(e) => setParaThreeDescription(e.target.value)}
      />
    </div>
    <div className="publish-box">
      <label>Published?</label>
      <select
        value={published === null ? "" : published}
        onChange={(e) => setPublished(e.target.value === "true")}
      >
        <option value={true}>Yes</option>
        <option value={false}>No</option>
      </select>
    </div>
    <button className="update-btn" onClick={handleUpdate}>
      UPDATE
    </button>
  </form>
</section>
</article>
);
};

export default UpdateBlog;

```

### **BlogController.js:-**

```

import { catchAsyncErrors } from "../middlewares/catchAsyncErrors.js";
import ErrorHandler from "../middlewares/error.js";
import { Blog } from "../models/blogSchema.js";
import cloudinary from "cloudinary";

export const blogPost = catchAsyncErrors(async (req, res, next) => {
  if (!req.files || Object.keys(req.files).length === 0) {
    return next(new ErrorHandler("Blog Main Image Is Mandatory!", 400));
  }
  const { mainImage, paraOneImage, paraTwoImage, paraThreeImage } =
    req.files;
  if (!mainImage) {
    return next(new ErrorHandler("Blog Main Image Is Mandatory!", 400));
  }
  const allowedFormats = ["image/png", "image/jpeg", "image/webp"];
  if (
    !allowedFormats.includes(mainImage.mimetype) ||
    (paraOneImage && !allowedFormats.includes(paraOneImage.mimetype)) ||
    (paraTwoImage && !allowedFormats.includes(paraTwoImage.mimetype)) ||
    (paraThreeImage && !allowedFormats.includes(paraThreeImage.mimetype))
  ) {
    return next(
      new ErrorHandler(
        "Invalid file type. Only JPG, PNG and WEBP Formats Are Allowed!",
        400
      )
    );
  }
  const {
    title,
    intro,
    paraOneDescription,
    paraOneTitle,
    paraTwoDescription,
    paraTwoTitle,
    paraThreeDescription,
    paraThreeTitle,
  } = req.body;

```

```

    category,
    published,
  } = req.body;

  const createdBy = req.user._id;
  const authorName = req.user.name;
  const authorAvatar = req.user.avatar.url;

  if (!title || !category || !intro) {
    return next(
      new ErrorHandler("Title, Intro and Category Are Required Fields!", 400)
    );
  }

  const uploadPromises = [
    cloudinary.uploader.upload(mainImage.tempFilePath),
    paraOneImage
      ? cloudinary.uploader.upload(paraOneImage.tempFilePath)
      : Promise.resolve(null),
    paraTwoImage
      ? cloudinary.uploader.upload(paraTwoImage.tempFilePath)
      : Promise.resolve(null),
    paraThreeImage
      ? cloudinary.uploader.upload(paraThreeImage.tempFilePath)
      : Promise.resolve(null),
  ];

  const [mainImageRes, paraOneImageRes, paraTwoImageRes,
    paraThreeImageRes] =
    await Promise.all(uploadPromises);
  if (
    !mainImageRes ||
    mainImageRes.error ||
    (paraOneImage && (!paraOneImageRes || paraOneImageRes.error)) ||
    (paraTwoImage && (!paraTwoImageRes || paraTwoImageRes.error)) ||
    (paraThreeImage && (!paraThreeImageRes || paraThreeImageRes.error))
  ) {

```

```

    return next(
      new ErrorHandler("Error occurred while uploading one or more images!",
500)
    );
  }
const blogData = {
  title,
  intro,
  paraOneDescription,
  paraOneTitle,
  paraTwoDescription,
  paraTwoTitle,
  paraThreeDescription,
  paraThreeTitle,
  category,
  createdBy,
  authorAvatar,
  authorName,
  published,
  mainImage: {
    public_id: mainImageRes.public_id,
    url: mainImageRes.secure_url,
  },
};
if (paraOneImageRes) {
  blogData.paraOneImage = {
    public_id: paraOneImageRes.public_id,
    url: paraOneImageRes.secure_url,
  };
}
if (paraTwoImageRes) {
  blogData.paraTwoImage = {
    public_id: paraTwoImageRes.public_id,
    url: paraTwoImageRes.secure_url,
  };
}
if (paraThreeImageRes) {

```

```

    blogData.paraThreeImage = {
      public_id: paraThreeImageRes.public_id,
      url: paraThreeImageRes.secure_url,
    };
  }
  const blog = await Blog.create(blogData);
  res.status(200).json({
    success: true,
    message: "Blog Uploaded!",
    blog,
  });
});

export const deleteBlog = catchAsyncErrors(async (req, res, next) => {
  const { id } = req.params;
  const blog = await Blog.findById(id);
  if (!blog) {
    return next(new ErrorHandler("Blog not found!", 404));
  }
  await blog.deleteOne();
  res.status(200).json({
    success: true,
    message: "Blog deleted!",
  });
});

export const getAllBlogs = catchAsyncErrors(async (req, res, next) => {
  const allBlogs = await Blog.find({ published: true });
  res.status(200).json({
    success: true,
    allBlogs,
  });
});

export const getSingleBlog = catchAsyncErrors(async (req, res, next) => {
  const { id } = req.params;
  const blog = await Blog.findById(id);
  if (!blog) {
    return next(new ErrorHandler("Blog not found!", 404));
  }

```

```

    }
    res.status(200).json({
      success: true,
      blog,
    });
  });
export const getMyBlogs = catchAsyncErrors(async (req, res, next) => {
  const createdBy = req.user._id;
  const blogs = await Blog.find({ createdBy });
  res.status(200).json({
    success: true,
    blogs,
  });
});
export const updateBlog = catchAsyncErrors(async (req, res, next) => {
  const { id } = req.params;
  let blog = await Blog.findById(id);
  if (!blog) {
    return next(new ErrorHandler("Blog not found!", 404));
  }
  const newBlogData = {
    title: req.body.title,
    intro: req.body.intro,
    category: req.body.category,
    paraOneTitle: req.body.paraOneTitle,
    paraOneDescription: req.body.paraOneDescription,
    paraTwoTitle: req.body.paraTwoTitle,
    paraTwoDescription: req.body.paraTwoDescription,
    paraThreeTitle: req.body.paraThreeTitle,
    paraThreeDescription: req.body.paraThreeDescription,
    published: req.body.published,
  };
  if (req.files) {
    const { mainImage, paraOneImage, paraTwoImage, paraThreeImage } =
req.files;
    const allowedFormats = ["image/png", "image/jpeg", "image/webp"];
    if (

```

```

    (mainImage && !allowedFormats.includes(mainImage.mimetype)) ||
    (paraOneImage && !allowedFormats.includes(paraOneImage.mimetype)) ||
    (paraTwoImage && !allowedFormats.includes(paraTwoImage.mimetype)) ||
    (paraThreeImage &&
!allowedFormats.includes(paraThreeImage.mimetype))
  ) {
    return next(
      new ErrorHandler(
        "Invalid file format. Only PNG, JPG and WEBp formats are allowed.",
        400
      )
    );
  }
  if (req.files && mainImage) {
    const blogMainImageId = blog.mainImage.public_id;
    await cloudinary.uploader.destroy(blogMainImageId);
    const newBlogMainImage = await cloudinary.uploader.upload(
      mainImage.tempFilePath
    );
    newBlogData.mainImage = {
      public_id: newBlogMainImage.public_id,
      url: newBlogMainImage.secure_url,
    };
  }
  if (req.files && paraOneImage) {
    if (blog.paraOneImage && blog.paraOneImage.public_id) {
      const blogParaOneImageId = blog.paraOneImage.public_id;
      await cloudinary.uploader.destroy(blogParaOneImageId);
    }
    const newBlogParaOneImage = await cloudinary.uploader.upload(
      paraOneImage.tempFilePath
    );
    newBlogData.paraOneImage = {
      public_id: newBlogParaOneImage.public_id,
      url: newBlogParaOneImage.secure_url,
    };
  }
}

```

```

if (req.files && paraTwoImage) {
  if (blog.paramTwoImage && blog.paramTwoImage.public_id) {
    const blogParamTwoImageId = blog.paramTwoImage.public_id;
    await cloudinary.uploader.destroy(blogParamTwoImageId);
  }
  const newBlogParamTwoImage = await cloudinary.uploader.upload(
    paraTwoImage.tempFilePath
  );
  newBlogData.paramTwoImage = {
    public_id: newBlogParamTwoImage.public_id,
    url: newBlogParamTwoImage.secure_url,
  };
}
if (req.files && paraThreeImage) {
  if (blog.paramThreeImage && blog.paramThreeImage.public_id) {
    const blogParamThreeImageId = blog.paramThreeImage.public_id;
    await cloudinary.uploader.destroy(blogParamThreeImageId);
  }
  const newBlogParamThreeImage = await cloudinary.uploader.upload(
    paraThreeImage.tempFilePath
  );
  newBlogData.paramThreeImage = {
    public_id: newBlogParamThreeImage.public_id,
    url: newBlogParamThreeImage.secure_url,
  };
}
blog = await Blog.findByIdAndUpdate(id, newBlogData, {
  new: true,
  runValidators: true,
  useFindAndModify: false,
});
res.status(200).json({
  success: true,
  message: "Blog Updated!",
  blog,
});

```



```
});
```

### **UserController.js:-**

```
import { catchAsyncErrors } from "../middlewares/catchAsyncErrors.js";
import ErrorHandler from "../middlewares/error.js";
import { User } from "../models/userSchema.js";
import { sendToken } from "../utils/jwtToken.js";
import cloudinary from "cloudinary";

export const register = catchAsyncErrors(async (req, res, next) => {
  if (!req.files || Object.keys(req.files).length === 0) {
    return next(new ErrorHandler("User Avatar Required!", 400));
  }
  const { avatar } = req.files;
  const allowedFormats = ["image/png", "image/jpeg", "image/webp"];
  if (!allowedFormats.includes(avatar.mimetype)) {
    return next(
      new ErrorHandler(
        "Invalid file type. Please provide your avatar in png, jpg or webp format.",
        400
      )
    );
  }
  const { name, email, password, phone, role, education } = req.body;
  if (
    !name ||
    !email ||
    !password ||
    !phone ||
    !role ||
    !education ||
    !avatar
  ) {
    return next(new ErrorHandler("Please fill full details!", 400));
  }
  let user = await User.findOne({ email });
  if (user) {
```

```

    return next(new ErrorHandler("User already exists", 400));
  }
  const cloudinaryResponse = await cloudinary.uploader.upload(
    avatar.tempFilePath
  );
  if (!cloudinaryResponse || cloudinaryResponse.error) {
    console.error(
      "Cloudinary error:",
      cloudinaryResponse.error || "Unknown cloudinary error!"
    );
  }
  user = await User.create({
    name,
    email,
    password,
    phone,
    role,
    education,
    avatar: {
      public_id: cloudinaryResponse.public_id,
      url: cloudinaryResponse.secure_url,
    },
  });
  sendToken(user, 200, "User registered successfully", res);
});
export const login = catchAsyncErrors(async (req, res, next) => {
  const { email, password, role } = req.body;
  if (!email || !password || !role) {
    return next(new ErrorHandler("Please fill full form!", 400));
  }
  const user = await User.findOne({ email }).select("+password");
  if (!user) {
    return next(new ErrorHandler("Invalid email or password!", 400));
  }
  const isPasswordMatched = await user.comparePassword(password);
  if (!isPasswordMatched) {
    return next(new ErrorHandler("Invalid email or password", 400));
  }

```

```

    }
    if (user.role !== role) {
      return next(
        new ErrorHandler(`User with provided role(${role}) not found`, 400)
      );
    }
    sendToken(user, 200, "User logged in successfully", res);
  });
export const logout = catchAsyncErrors((req, res, next) => {
  res
    .status(200)
    .cookie("token", "", {
      expires: new Date(Date.now()),
      httpOnly: true,
    })
    .json({
      success: true,
      message: "User logged out!",
    });
});
export const getMyProfile = catchAsyncErrors((req, res, next) => {
  const user = req.user;
  res.status(200).json({
    success: true,
    user,
  });
});

export const getAllAuthors = catchAsyncErrors(async (req, res, next) => {
  const authors = await User.find({ role: "Author" });
  res.status(200).json({
    success: true,
    authors,
  });
});

```

# CHAPTER 10

## TESTING

### Introduction :-

Testing is a fundamental aspect of the software development life cycle (SDLC), aimed at ensuring the quality, functionality, and reliability of software applications. It involves systematically examining a system or its components to detect errors, defects, or deviations from specified requirements. By identifying issues early in the development process, testing helps to deliver software that meets user expectations and business objectives.

The primary goal of testing is to validate that the software performs as intended under various conditions and adheres to its functional and non-functional requirements. Functional testing evaluates specific actions or outputs based on inputs, ensuring the application behaves correctly. Non-functional testing assesses attributes like performance, usability, scalability, and security, ensuring the software is robust and user-friendly.

Testing can be conducted manually or through automated tools, with each approach offering unique advantages. Manual testing allows for a human-centric perspective, while automation provides efficiency and precision, particularly for repetitive tasks. Testing is categorized into different levels, such as unit testing, integration testing, system testing, and acceptance testing, each focusing on a specific stage of the software's development and deployment.

Effective testing plays a vital role in minimizing risks, reducing maintenance costs, and improving customer satisfaction. It prevents critical errors that could lead to system failures, data breaches, or negative user experiences. Additionally, testing supports continuous improvement by providing feedback for developers to enhance code quality.

In a competitive software market, rigorous testing is indispensable for maintaining a product's reputation and trustworthiness. It ensures the software meets high standards of quality, reliability, and usability, ultimately contributing to its success in real-world scenarios.

### Types of Testing :-

- **Manual Testing:** Executing test cases manually without automation tools.
- **Automated Testing:** Using tools/scripts to automate test case execution.

- **Functional Testing:** Verifying software functionality against requirements.
- **Non-functional Testing:** Testing software's performance, usability, scalability, etc.
- **Unit Testing:** Testing individual components or modules.
- **Integration Testing:** Ensuring different modules work together seamlessly.
- **System Testing:** Validating the complete and integrated system.
- **Acceptance Testing:** Checking if the system meets business requirements (User and Alpha/Beta Testing).

## Future Scope and Further Enhancement of the Project

The ScrollIn Project, designed as a comprehensive blogging platform using the MERN stack, has ample opportunities for expansion and improvement:

1. **Advanced Editor Features:** Integrate a more sophisticated text editor with rich formatting options, including drag-and-drop media uploads and markdown support.
2. **User Engagement Tools:** Add features such as likes, shares, and social media integration to enhance user interaction and reach.
3. **Recommendation System:** Use machine learning to recommend blogs based on user preferences and browsing history.
4. **Mobile Application:** Develop a cross-platform mobile app for better accessibility and convenience.
5. **Analytics Dashboard:** Provide authors with insights into their blog performance, such as readership trends, engagement metrics, and geographic reach.
6. **Search and Filters:** Implement an advanced search functionality with filters for tags, categories, and authors.
7. **Monetization Features:** Introduce advertising spaces, subscription models, or a tipping system for authors.
8. **Enhanced Security:** Strengthen data protection with measures like role-based access control, enhanced encryption for sensitive information, and real-time threat monitoring.
9. **Localization:** Support multiple languages to cater to a global audience.
10. **Scalability:** Optimize the backend to handle more concurrent users and large-scale data efficiently.

## **CONCLUSION**

The ScrollIn Project achieves its goal of creating an interactive and dynamic blogging platform. By leveraging the MERN stack, it demonstrates seamless integration of modern technologies to provide a user-friendly experience. This project highlights the importance of efficient design, secure authentication, and responsive UI in delivering value to both authors and readers. While the current implementation fulfills its objectives, the project is well-positioned for future advancements that can expand its reach and functionality, making it a robust solution for content creators.

## REFERENCES

### Books

1. **Mario Casciaro, Luciano Mammino, *Node.js Design Patterns***, Packt Publishing, 2016.
  2. **Alex Banks, Eve Porcello, *Learning React: Functional Web Development with React and Redux***, O'Reilly Media, 2020.
  3. **Shannon Bradshaw, Eoin Brazil, *MongoDB: The Definitive Guide***, O'Reilly Media, 2019.
  4. **Ethan Brown, *Web Development with Node and Express: Leveraging the JavaScript Stack***, O'Reilly Media, 2019.
  5. **Frank Zammetti, *Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker***, Apress, 2020.
  6. **Heather Adkins et al., *Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems***, O'Reilly Media, 2020.
- 

### Research Papers

1. **"A Study on MERN Stack Development for Web Applications,"** International Journal of Computer Applications, 2021.
  2. **"JWT Security in Web Applications,"** IEEE Transactions on Information Forensics and Security, 2020.
  3. **"Modern NoSQL Databases and Their Role in Scalable Web Applications,"** Journal of Data Management, 2020.
  4. **"Scalable Web Applications Using MERN Stack,"** International Journal of Computer Science and Information Technologies, 2022.
  5. **"Comparative Study of Relational and NoSQL Databases in the Context of Modern Web Applications,"** ACM SIGMOD Record, 2019.
  6. **"User Experience Optimization through Modern Frontend Frameworks,"** International Journal of Human-Computer Interaction, 2021.
- 

### Online Resources

1. [React Official Documentation.](#)
  2. [Node.js Official Documentation.](#)
  3. [MongoDB Official Documentation.](#)
  4. [Express.js Guide.](#)
  5. [JWT Best Practices.](#)
  6. [Figma Design Resources.](#)
  7. [Mozilla Developer Network \(MDN\).](#)
  8. [ACM Digital Library.](#)
-