**CONTENT MANAGEMENT SYSTEM**

*A Major Project thesis submitted in partial fulfillment of requirements for the*

*award of degree for VIII semester*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

By

**ALAPATI SRIVARDHANI (20131A0506)**

**DASARI VENKATA GNANESWAR (20131A0556)**

**DANDAMUDI SHALEM RAJA (20131A0553)**

**AVANIGADDA DHANVESH NAGA SAI (20131A0519)**



Under the esteemed guidance of

**Dr. CH. SITA KUMARI**

**Associate Professor**

### Department of Computer Science and Engineering

### GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)

### (Affiliated to JNTU-K, Kakinada)

### VISAKHAPATNAM

### 2023 – 2024

**Gayatri Vidya Parishad College of Engineering (Autonomous)**

**Visakhapatnam**

****

**CERTIFICATE**

This is to certify that the major project entitled **“Content Management System”** being submitted by

**ALAPATI SRIVARDHANI (20131A0506)**

**DASARI VENKATA GNANESWAR (20131A0556)**

**DANDAMUDI SHALEM RAJA (20131A0553)**

**AVANIGADDA DHANVESH NAGA SAI (20131A0519)**

in partial fulfillment for the award of the degree “Bachelor of Technology” in Computer Science and Engineering to the Jawaharlal Nehru Technological University, Kakinada is record of bonafide work done under my guidance and supervision during VIII semester of the academic year 2023-2024.

**Project Guide Head of the Department**

Dr. CH. SITA KUMARI Dr. D. UMA DEVI

Associate Professor H.O.D and Associate Professor

Department of CSE Department of CSE

GVPCE(A) GVPCE(A)

**DECLARATION**

We hereby declare that this Major Project entitled **“CONTENT MANAGEMENT SYSTEM”** is a bonafide work done by us and submitted to **Department of Computer Science and Engineering , Gayatri Vidya Parishad College of Engineering (Autonomous) Visakhapatnam** , in partial fulfilment for the award of the degree of B.Tech is of our own and it is not submitted to any other university or has been published any time before.

PLACE: VISAKHAPATNAM A. SRIVARDHANI (20131A0506)

DATE: 26-03-2024 D. V. GNANESWAR (20131A0556)

D. SHALEM RAJA (20131A0553)

A. DHANVESH NAGA SAI (20131A0519)

**ACKNOWLEDGEMENT**

We thank **Dr. A.B. KOTESWARA RAO** principal, **Gayatri Vidya Parishad College of Engineering (Autonomous)** for extending his utmost support and cooperation in providing all the provisions for the successful completion of the project.

We consider it our privilege to express our deepest gratitude to **Dr. D. UMA DEVI, Associate professor and Head of the Department of Computer Science and Engineering**,

for her valuable suggestions and constant motivation that greatly helped the project work to get successfully completed.

We thank our project coordinator and guide, **Dr. CH. SITA KUMARI**, **Associate Professor, Department of Computer Science and Engineering**, for the kind suggestions and guidance for the successful completion of our project work.

We also thank all the members of the staff in Computer Science and Engineering for their sustained help in our pursuits. We thank all those who contributed directly or indirectly in successfully carrying out this work.

A. SRIVARDHANI (20131A0506)

D. V GNANESWAR (20131A0556)

D. SHALEM RAJA (20131A0553)

A. DHANVESH NAGA SAI (20131A0519)

**ABSTRACT**

In the contemporary digital landscape, the proliferation of information sharing and community engagement is paramount, and our Content Management System (CMS) emerges as a dynamic and inclusive solution. Tailored to accommodate a multitude of domains, including but not limited to sports, interview experiences, college reviews, and more, our CMS establishes itself as a centralized hub fostering seamless content access, creation, and contribution. The fundamental principle of our CMS lies in creating a unified platform where users can effortlessly explore and contribute to a diverse array of topics.A distinctive feature of our CMS is its versatility, providing a comprehensive one-stop destination for individuals seeking information and engagement across various domains. Whether one is interested in the latest sports updates, gleaning insights from interview experiences, or researching colleges through authentic reviews, our CMS serves as an all-encompassing repository of knowledge and experiences.[8] To enhance user experience, our CMS is equipped with intuitive and user-friendly tools for content creation, editing, and formatting. This ensures that contributors, regardless of their technical expertise, can effortlessly articulate their thoughts and experiences, thereby democratizing the content creation process. our Content Management System redefines the paradigm of information sharing and community engagement in the digital age. Through its user-centric design, diverse domain support, rewards system, and collaborative environment, the CMS empowers individuals to share, discover, and connect over a myriad of topics, fostering a rich and inclusive online community.

**Keywords:** Information sharing, formatting tools, user-friendly, domains.

**INDEX**

**CHAPTER 1. INTRODUCTION 7**   
 1.1 Problem definition 7

1.2 Motivation 8

1.3 Purpose 8

1.4 Scope 8

**CHAPTER 2. SRS SOCUMENT 9**

2.1 Functional requirements 9

2.2 Non-functional requirements 10

2.3 Minimum hardware requirements 10

2.4 Minimum software requirements 11

**CHAPTER 3. ANALYSIS 12**

3.1 Existing methods and researches 12

3.2 Proposed method 13

3.3 Feasibility study 14

3.4 Cost benefit analysis 17

**CHAPTER 4. SOFTWARE DESCRIPTION 18**

4.1 Programming languages 18

4.2 Databases 19

4.2 Technologies 19

4.4 Platform 20

**CHAPTER 5. SYSTEM DESIGN 21**

5.1 Flow chart 21

5.2 Introduction to UML 22

5.3 Building blocks of UML 24

**CHAPTER 6. PROJECT DEVELOPMENT 33**

6.1 Overview 33

6.2 Code templates 34

**CHAPTER 7. RESULTS 52**

**CHAPTER 8. CONCLUSION 56**

**CHAPTER 9. FUTURE SCOPE 57**

**CHAPTER 10. REFERENCE LINKS 58**

1. **INTRODUCTION**

In the ever-evolving digital landscape, where information is abundant and diverse, Content Management Systems (CMS) play a pivotal role in orchestrating the creation, organization, and dissemination of digital content. A CMS is a robust and user-friendly platform designed to streamline the often complex process of managing and publishing content on the internet.[1]

At its core, a Content Management System serves as the digital engine powering websites, blogs, and various online platforms. It provides individuals and organizations with the tools and infrastructure necessary to create, edit, organize, and publish content without requiring extensive technical expertise. CMS platforms have become indispensable for businesses, bloggers, and content creators seeking an efficient and centralized solution for content governance.

CMS features vary widely. Most CMSs include the following [3]

* Web-based publishing,
* Format management,
* Revision control (version control),
* Indexing, search, and retrieval.
  1. **PROBLEM DEFINITION:**

In the contemporary digital landscape, the escalating volume and diversity of content creation have underscored the need for an efficient and adaptable Content Management System (CMS). Recognizing the challenges and limitations of existing systems, the proposed CMS project aims to address specific issues encountered by users, content creators, and organizations in managing and disseminating digital content effectively. The identified problems that motivate the development of this CMS project like complexity in content creation, limited collaboration and workflow management.

* 1. **MOTIVATION:**

The motivation behind undertaking the Content Management System (CMS) project stems from the increasing need for a robust and versatile solution in the dynamic digital landscape. As information sharing and collaboration become integral components of online interactions, the existing CMS platforms often fall short of addressing the diverse and evolving requirements of users. This project is driven by the aspiration to create a CMS that not only streamlines the process of content creation, organization, and distribution but also fosters a sense of community and engagement. The aim is to empower individuals and organizations by providing an intuitive, user-centric platform that accommodates various domains, including sports, interview experiences, college reviews, and more.

* 1. **PURPOSE:**

The main purpose of a Content Management System (CMS) is to facilitate the creation, organization, management, and publication of digital content in a collaborative and user-friendly manner. The key objectives of CMS include content creation and editing, content organization, publishing and distribution, collaboration and workflow management.

* 1. **SCOPE:**

The scope of our Content Management System (CMS) project is defined by its overarching goal to revolutionize how digital content is created, managed, and shared across various domains. This project envisions a versatile CMS that caters to the diverse needs of users, spanning realms such as sports, interview experiences, college reviews, and beyond. The CMS will offer an intuitive interface, supporting easy-to-use tools for content creation, editing, and formatting, thereby democratizing the content generation process. Furthermore, the project aims to address the limitations of existing CMS platforms. Our CMS project encompasses the development of a feature-rich, user-friendly, and adaptable platform that empowers users to share, discover, and connect over a broad spectrum of topics.

1. **SRS DOCUMENT**

A software requirements specification (SRS) is a document that outlines the functions and performance standards for the software.

* 1. **FUNCTIONAL REQUIREMENTS**

In software engineering, a functional requirement defines a function of a software system or its component. A function is described as a set of inputs and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). How a system implements functional requirements is detailed in the system design. In some cases, a requirements analyst generates use cases after gathering and validating a set of functional requirements. Each use case illustrates behavioral scenarios through one or more functional requirements.[12]

The functional requirements of the system is as follows:

* Register to the portal
* Login to the portal
* Create user for admin
* Create content to publish
* Upload image in the blog
* Publish the blog
* Edit the blog
* Delete the blog
* Like the blog
* Dislike the blog
* Add comment to others blog
* Search for required blog
* Logout
  1. **NON-FUNCTIONAL REQUIREMENTS**

The NON-FUNCTIONAL REQUIREMENT (NFR) standardizes a software system's

quality attribute. Based on responsiveness, usability, security, and portability, they assess the

software system. The following are qualities of a system, sometimes known as non-functional

requirements:

 **Performance:** The website responds quickly on average.

 **Reliability -** Our metric for reliability is the system's capacity to recognize and reject

invalid credentials or keys. Currently, the system rejects erroneous credentials in 99 out of 100 instances.

 **Operability-** The website's user interface is very dependable and clear, making it easy to use.

 **Efficiency -** The website design is very simple and intended to be direct, therefore working efficiency is excellent.

 **Understandability**- Users can quickly understand how it functions thanks to its user-friendly interfaces.

* 1. **SYSTEM REQUIREMENTS:**

To be used efficiently, all computer software needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as guidelines as opposed to an absolute rule. Most software defines two sets of system requirements minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades in existing computer systems than technological advancements.[12]

**2.3 MINIMUM HARDWARE REQUIREMENTS:**

* Processor - Intel Core
* i5Hard Disk - 512GB
* RAM - 4GB

**2.4 MINIMUM SOFTWARE REQUIREMENTS:**

* Programming Languages - HTML, CSS, Java Script, ReactJS, NodeJS
* Databases – MongoDB Atlas
* Technologies – MongoDB Cloud Services
* Platforms - Visual Studio Code

1. **ANALYSIS**
   1. **EXISTING SYSTEMS AND RESEARCHES:**

Understanding the landscape of existing systems and relevant research is crucial for informing the development and design decisions in the creation of a new Content Management System (CMS).[4]

* Medium is a content platform that focuses on providing a clean and distraction-free environment for writers. Research on user engagement, content curation algorithms, and the impact of design simplicity on reader experience, as observed in Medium, can inform decisions related to user interface and engagement strategies.
* WordPress stands out as one of the most widely used and versatile CMS platforms globally. Its open-source nature, extensive plugin ecosystem, and user-friendly interface have contributed significantly to its popularity. Research exploring the usability, extensibility, and security features of WordPress can provide valuable insights for enhancing the proposed CMS.

**3.1.1 DRAWBACKS OF EXISTING SYSTEMS:**

* **Lack of Categorized Search:**   
  -->Existing website lacks a feature to search for blogs based on specific concepts or categories such as interviews, sports, education, etc.  
  --> Users are limited to searching for specific topics, and the categories are not effectively organized for easy exploration.
* **Limited Search Functionality:**   
  --> The search functionality is constrained to specific entities like people and publications, restricting users from conducting more nuanced searches based on broader themes or subjects.
* **Irreversible Likes:**    
  --> Once a user likes a blog on Medium, there is no option to undo or retract the like.   
  --> This absence of a 'dislike' or undo feature could lead to an inaccurate representation of a blog's popularity, as the like count accumulates without the ability to revise or adjust.
* **Image Upload Capability:**  
  -->Authors currently lack the ability to complement their textual content with images. This absence restricts authors from providing visual aids that could significantly enhance readers' understanding of their ideas.
  1. **PROPOSED METHOD:**

The envisioned content management system introduces a diverse array of features aimed at elevating the overall user experience. Users are empowered to establish accounts with secure passwords, ensuring the safeguarding of their personal data. The system prioritizes the secure storage of user information within the database, emphasizing confidentiality and integrity. Upon login, users are greeted with meticulously organized categories such as education, interviews, sports, and more, facilitating easy navigation. Authors can align their content with specific categories, promoting a structured and thematic presentation of blogs. The inclusion of image uploads enhances the visual appeal of authored content. A dynamic feedback mechanism allows readers to express opinions through likes and dislikes, adding an interactive dimension to the platform. Further enriching user interaction, the system incorporates a robust search functionality, enabling swift content discovery based on individual interests or specific topics. Collectively, these integrated features contribute to the creation of a comprehensive and user-friendly content management system, cultivating a seamless and engaging environment for both authors and readers alike.

* + 1. **ADAVANTAGES OF PROPOSED SYSTEM:**
* The proposed content management system offers a range of features to enhance user experience.
* Users can create an account with a secure password, ensuring the protection of their personal information.
* User data is securely stored in the database, prioritizing the confidentiality and integrity of user information.
* Upon logging in with their email and password, users encounter neatly organized categories such as education, interviews, sports, and more.
* Authors can select a specific category aligned with the content they wish to post, allowing for organized and thematic structuring of blogs.
* Authors have the ability to enhance their blogs by uploading images, providing a visual dimension to their content.
* Readers can express their opinions through likes and dislikes, offering a dynamic and interactive element to the platform.
* The system boasts a robust search functionality, enabling users to quickly locate content based on their interests or specific topics of choice.
* These combined features contribute to a comprehensive and user-friendly content management system, fostering a seamless and engaging environment for both authors and readers.
  1. **FEASABILITY STUDY:**

A feasibility study is an analysis that determines the chance of successfully completing a project by taking into account all pertinent elements, including economic, technical, legal, and scheduling issues. A feasibility study is essential to evaluate whether any proposed project is feasible or not. A feasibility study evaluates the practicality of a given plan or project.

The main objectives of feasibility are mentioned below:

To determine if the product is technically and financially feasible to develop, is the main aim

of the feasibility study activity.[6] A feasibility study should give management with adequate information to make the following decisions:

 Whether the project can be done.

 To determine how successful your proposed action will be.

 Whether the final product will benefit its intended users.

 To describe the nature and complexity of the project.

 What are the alternatives among which a solution will be chosen

(During subsequent phases)

 To analyze if the software meets organizational requirements.

The results of our feasibility study are:

 This project is viable and implementable.

 This proposed project is designed to provide a secure and efficient solution for storing user login information.

 The CMSs can adapt to growing demands, both in terms of content volume and user engagement.

 Further development of this project will meet organizational requirements.

There are various types of feasibility that can be determined. They are:

**Operational** - Define the urgency of the issue and the acceptability of any solution, taking into account people- and social-oriented issues. These include internal issues like staffing shortages, labor disputes, manager resistance, organizational conflicts, and policies, as well as external issues like social acceptance, legal considerations, and governmental regulations.

**Technical**: Is the feasibility within the limits of current technology? Does the technology

exist at all? Is it available within a given resource?

**Economic** - Given the resource limitations, is the project feasible? Are the costs associated

with the new system worth the potential rewards? What will the system save in terms of both monetary and non-monetary savings? What are the operating and development costs?

**Schedule -** The project's workflow constraints and if they might be reasonably satisfied.

**3.3.1 ECONOMIC FEASIBILITY:**

Economic analysis could also be referred to as cost/benefit analysis. It is the most

frequently used method for evaluating the effectiveness of a new system. In economic analysis the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. Economic feasibility study related to price, and all kinds of expenditure related to the scheme before the project starts. This study also improves project reliability. It is also helpful for the decision-makers to decide the planned scheme processed latter or now, depending on the financial condition of the organization.

This evaluation process also studies the price benefits of the proposed scheme. Economic feasibility also performs the following tasks.

  Cost of packaged software/ software development.

  Cost of doing full system study.

  Is the system cost Effective?

The results of economic feasibility are:

  The positive outcomes from the analysis indicate that the project aligns with economic objectives and has the potential to provide a favorable return on investment while offering sustainable revenue generation opportunities.

**3.3.2 TECHNICAL FEASIBILITY:**

A large part of determining resources has to do with assessing technical feasibility. It considers the technical requirements of the proposed project. The technical requirements are then compared to the technical capability of the organization. The systems project is considered technically feasible if the internal technical capability is sufficient to support the project requirements. The analyst must find out whether current technical resources can be where the expertise of system analysts is beneficial, since using their own experience and their contact with vendors they will be able to answer the question of technical feasibility.

Technical feasibility also performs the following tasks.

  Is the technology available within the given resource constraints?

  Is the technology have the capacity to handle the solution

  Determines whether the relevant technology is stable and established.

  Does the technology chosen for software development has a large number of users so that they can be consulted when problems arise, or improvements are required?

The technical feasibility results are:

  The required technology is freely available with the given resource constraints.

  The required technologies, such as web development frameworks and database systems, are readily available and well-supported.

**3.3.3 OPERATIONAL FEASIBILITY:**

Operational feasibility is a measure of how well a proposed system solves the problems and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. The operational feasibility refers to the availability of the operational resources needed to extend research results beyond on which they were developed and for which all the operational requirements are minimal and easily accommodated. In addition the operational feasibility would include any rational compromises farmers make in adjusting the technology to the limited operational resources available to them.

The operational Feasibility also perform the tasks like

  Does the current mode of operation provide adequate response time?

  Does the current of operation make maximum use of resources?

  Determines whether the solution suggested by the software development team

is acceptable.

  Does the operation offer an effective way to control the data?

The operational feasibility results are:

 The operational feasibility evaluation affirms that the CMS project can be seamlessly integrated into existing operations.

* 1. **COST BENEFIT ANALYSIS**

The preliminary investigation's financial and economic queries are confirmed to estimate the following:

* A thorough cost-benefit analysis has been conducted, considering development costs, infrastructure investments, and potential operational expenses.
* The benefits, including increased efficiency in content management, enhanced user engagement, and potential monetization avenues, outweigh the incurred costs.

1. **SOFTWARE DESCRIPTION**

**4.1 PROGRAMMING LANGUAGES**

**4.1.1 HTML:**

HTML is the foundational language for creating and structuring content on the World Wide Web. It utilizes markup tags to define the elements of a web page, such as headings, paragraphs, links, images, and more. HTML provides the structure and semantics that web browsers interpret to render the content for users. As a core technology in web development, HTML is essential for creating accessible and well-organized documents, forming the backbone of virtually every webpage on the internet.

**4.1.2 CSS:**

CSS complements HTML by providing a styling mechanism to enhance the visual presentation of web pages. It allows developers to control the layout, color, typography, and other stylistic aspects of HTML elements. CSS follows a cascading style rule, enabling the definition of styles at various levels, from inline styles to external style sheets. This separation of content (HTML) and presentation (CSS) promotes maintainability and consistency across a website. CSS is instrumental in creating visually appealing and responsive web designs, ensuring a positive user experience across different devices and screen sizes.

**4.1.3 JAVA SCRIPT:**

JavaScript is a versatile and dynamic programming language that adds interactivity and functionality to web pages. It is primarily executed on the client-side (in the user's browser), allowing developers to create dynamic content, handle user input, and manipulate the Document Object Model (DOM) in real-time. JavaScript is an essential component of modern web development, enabling the creation of interactive features like sliders, form validations, and asynchronous communication with servers. Its widespread use across web browsers makes it a cornerstone for building engaging and user-friendly web applications.

**4.1.4 REACT JS:**

ReactJS, developed by Facebook, is a declarative and efficient JavaScript library for building user interfaces. It allows developers to create reusable UI components that update in response to data changes, providing a seamless and efficient way to build complex user interfaces. React employs a virtual DOM (Document Object Model) for optimal rendering performance, minimizing the need to directly manipulate the actual DOM. Its component-based architecture and one-way data binding simplify the development process, making it easier to maintain and scale large applications. React is widely adopted for building interactive and dynamic user interfaces in single-page applications.

**4.1.5 NODE JS:**

Node.js is a server-side JavaScript runtime environment that enables the execution of JavaScript code outside of a web browser. It uses the V8 JavaScript engine from Google Chrome to execute code efficiently. Node.js is known for its non-blocking, event-driven architecture, making it well-suited for building scalable and high-performance server applications. It has a vast ecosystem of modules available through the Node Package Manager (NPM), facilitating the development of a wide range of applications, including web servers, APIs, and real-time applications. Node.js has become a popular choice for developers seeking a unified language (JavaScript) for both client and server-side development, streamlining the overall development process.

**4.2 DATABASES**

**4.2.1 MONGO DB ATLAS:**

MongoDB Atlas is a fully managed and cloud-based database service that provides users with a convenient and scalable solution for deploying, managing, and scaling MongoDB databases. Developed by MongoDB, Inc., Atlas allows users to enjoy the benefits of MongoDB, a NoSQL database, without the operational complexities of traditional self-hosted databases. Atlas offers automated backups, built-in security features, and seamless scaling, making it an ideal choice for developers who want to focus on building applications rather than managing database infrastructure. It supports various cloud providers such as AWS, Azure, and Google Cloud Platform, providing flexibility in choosing the preferred cloud environment for hosting MongoDB databases.

**4.3 TECHNOLOGIES**

**4.3.1 MONGO DB CLOUD SERVICES**

MongoDB Cloud Services collectively refer to a suite of cloud-based offerings provided by MongoDB, Inc., designed to enhance the deployment, management, and optimization of MongoDB databases in the cloud. This includes services like MongoDB Atlas, MongoDB Realm, and MongoDB Charts. MongoDB Cloud Services are crafted to streamline the development and operational aspects of working with MongoDB databases, offering features such as automatic scaling, data visualization, and serverless functions through MongoDB Realm. These services cater to the evolving needs of modern applications by providing a comprehensive and integrated ecosystem for building, deploying, and scaling MongoDB-based solutions on the cloud.

**4.4 PLATFORMS**

**4.4.1 VISUAL STUDIO CODE**

Visual Studio Code (VS Code) is a lightweight, open-source code editor developed by Microsoft. It has gained widespread popularity among developers for its versatility, extensibility, and robust feature set. VS Code supports a wide range of programming languages and offers features like syntax highlighting, code completion, integrated Git control, and debugging support. Its modular architecture allows users to enhance functionality through extensions, making it adaptable to various development workflows. VS Code also provides seamless integration with cloud services, source code repositories, and various programming frameworks. Its user-friendly interface, combined with a thriving extension ecosystem, makes it a preferred choice for developers across different domains for writing, debugging, and maintaining code efficiently.

1. **SYSTEM DESIGN**

System design is the phase that bridges the gap between problem domain and the existing system in a manageable way. It is the process of defining elements of a system like modules, architecture, components and their interfaces and data for a system based on the specified requirements. This involves defining, developing and designing systems which satisfies the specific needs and requirements.

In the phase of system design, the SRS document is converted into a format that can be implemented and decides how the system will operate. A designer uses the modelling languages to express the information and knowledge in a structure of system that is defined by a consistent set of rules and definitions. The designs can be defined in graphical or textual modelling languages.

* 1. **FLOWCHART**

A flowchart is a type of diagram that represents a workflow or process. It is a simple graphical representation of steps in sequential order and is widely used in presenting the flow of algorithms, workflow or processes. A flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields. They use special shapes to represent different types of actions or steps in a process. Lines and arrows show the sequence of the steps, and the relationships among them.

Most common types of flowcharts include

▪ Process flowchart

▪ Swimlane flowchart

▪ Dataflow diagram

▪ Workflow diagram

A flowchart representing the process flow of our project is shown below.

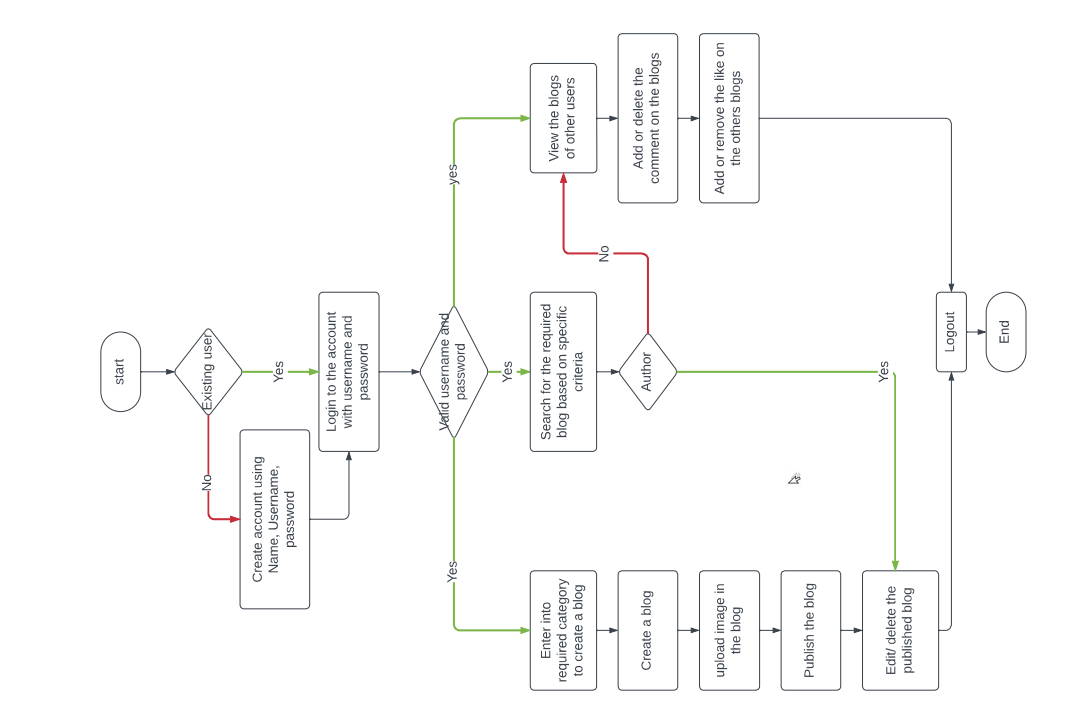


Fig 5.1 Flowchart

* 1. **INTRODUCTION TO UML:**

Unified Modeling Language **(UML)** is a general-purpose modeling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite like blueprints used in other fields of engineering. UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modeling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. It's been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.[13]

**5.2.1 WHY WE NEED UML?**

  Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate among them.

 Businessmen do not understand code. So, UML becomes essential to communicate with non-programmer’s essential requirements, functionalities and processes of the system.

  A lot of time is saved down the line when teams can visualize processes, user

interactions and static structure of the system.

UML is linked with **object-oriented** design and analysis. UML makes the use of elements

and forms associations between them to form diagrams. Diagrams in UML can be broadly

classified as:

 **Structural Diagrams –** Capture static aspects or structure of a system. Structural

Diagrams include Component Diagrams, Object Diagrams, Class Diagrams and

Deployment Diagrams.



**Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior

diagrams include Use Case Diagrams, State Diagrams, Activity Diagrams and

Interaction Diagrams.

Building Blocks of the UML Building Blocks of the UML Building Blocks of the UML

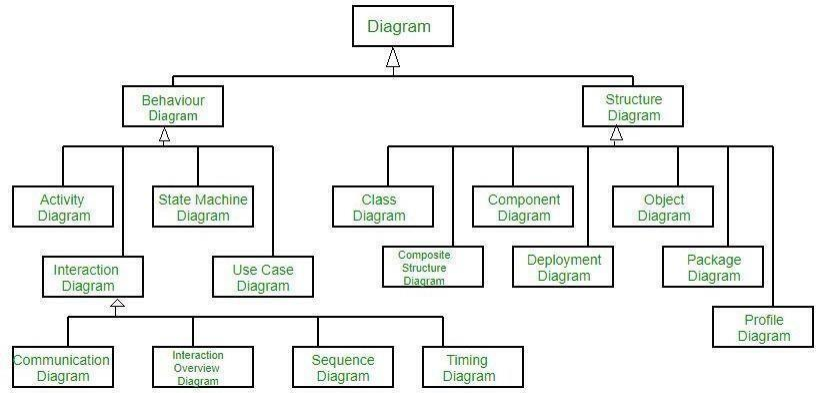


Fig 5.2 Building blocks of UML

**5.3 BUILDING BLOCKS OF UML**

The vocabulary of the UML encompasses three kinds of building blocks:

  Things

 Relationships

  Diagrams

Things are the abstractions that are first-class citizens in a model; relationships tie these

things together; diagrams group interesting collections of things.[13]

**5.3.1 THINGS IN UML**

There are four kinds of things in the UML:

➢ Structural things

➢ Behavioral things

➢ Grouping things

➢ Annotational things

These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models.[13]

**Structural Things**

Structural things are the nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical. Collectively, the structural things are called classifiers.

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations.

**Class** - A Class is a set of identical things that outlines the functionality and properties of an object. It also represents the abstract class whose functionalities are not defined.

Its notation is as follows



Fig 5.3 Class Representation

**Interface** - A collection of functions that specify a service of a class or component, i.e. externally visible behavior of that class.



Fig 5.4 Interface Representation

**Collaboration** - A larger pattern of behaviors and actions.

Example: All classes and behaviors that create the modeling of a moving tank in a simulation.



Fig 5.5 Collaboration Representation

**Use Case** - A sequence of actions that a system performs that yields an observable result. Used to structure behavior in a model. Is realized by collaboration.



Fig 5.6 Use-case Representation

**Component** - A physical and replaceable part of a system that implements a number of interfaces.

Example: a set of classes, interfaces, and collaborations.



Fig 5.7 Component Representation

**Node** - A physical element existing at run time and represents a computational source.



Fig 5.8 Node Representation

**Behavioral Things**

Behavioral things are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In all, there are two primary kinds of behavioral things

  Interaction

  State machine

**Interaction:**

It is a behavior that comprises a set of messages exchanged among a set of objector roles within a particular context to accomplish a specific purpose. The behavior of a society of objects or of an individual operation may be specified with an interaction. An interaction involves a number of other elements, including messages, actions, and connectors (the connection between objects). Graphically, a message is rendered as a directed line, almost always including the name of its operation.



Fig 5.9 Interaction Representation

**State machine:**

State machine is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events. The behavior of an individual class or a collaboration of classes may be specified with a state machine. A state machine involves a number of other elements, including states, transitions (the flow from state to state), events (things that trigger a transition), and activities (the response to a transition). Graphically, a state is rendered as a rounded rectangle, usually including its name and its sub states.



Fig 5.10 State Representation

**Grouping Things**

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available.

**Package −** Package is the only one grouping thing available for gathering structural and behavioral things.



Fig 5.11 Package Representation

**Annotational Things**

Annotational things are the explanatory parts of UML models. These are the comments you may apply to describe, illuminate, and remark about any element in a model. There is one primary kind of annotational thing, called a note. A note is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.



Fig 5.12 Note Representation

**5.3.2 RELATIONSHIPS IN UML:**

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships in the UML:

  Dependency

  Association

  Generalization

  Realization

**Dependency**

It is an element (the independent one) that may affect the semantics of the other element (the dependent one). Graphically, a dependency is rendered as a dashed line, possibly directed, and occasionally including a label.



Fig 5.13 Dependency Representation

**Association**

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.



Fig 5.14 Association Representation

**Generalization**

It is a specialization/generalization relationship in which the specialized element (the child) builds on the specification of the generalized element (the parent). The child shares the structure and the behavior of the parent. Graphically, a generalization relationship is rendered as a solid line with a hollow arrowhead pointing to the parent.



Fig 5.15 Generalization Representation31

**Realization**

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



Fig 5.16 Realization Representation

**5.3.3 UML DIAGRAMS**

UML is a modern approach to modeling and documenting software. It is based on **diagrammatic representations** of software components. It is the final output, and the diagram represents the system.

**UML includes the following:**

➢ Class diagram

➢ Object diagram

➢ Component diagram

➢ Composite structure diagram

➢ Use case diagram

➢ Sequence diagram

➢ Communication diagram

➢ State diagram

➢ Activity diagram

**SEQUENCE DIAGRAM**

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios. The horizontal axis shows the elements involved in the interaction and the vertical axis shows the progress of time or the ordering of messages. The figure below shows the sequence diagram of our model.

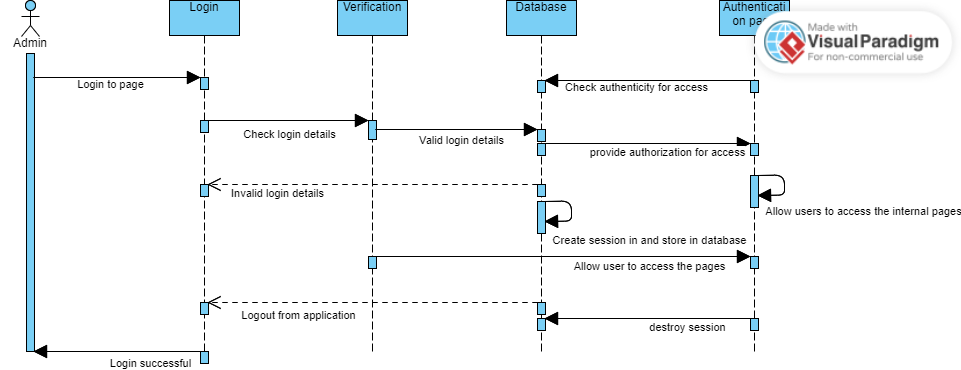


Fig 5.17 Sequence diagram for login

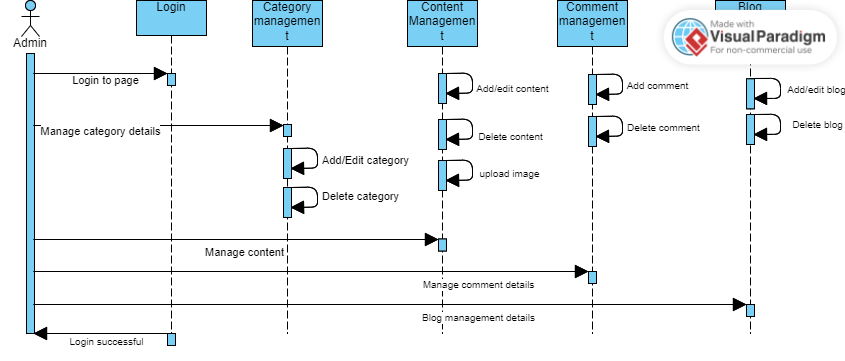


Fig 5.18 Sequence diagram after login

**USE CASE DIAGRAM**

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures. A use case diagram depicts the functionality of a system. It consists of actors, use cases and their relationships. It is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. The use case diagrams represent the core parts of a system and the workflow between them. A use case diagram for our model is shown the figure below.

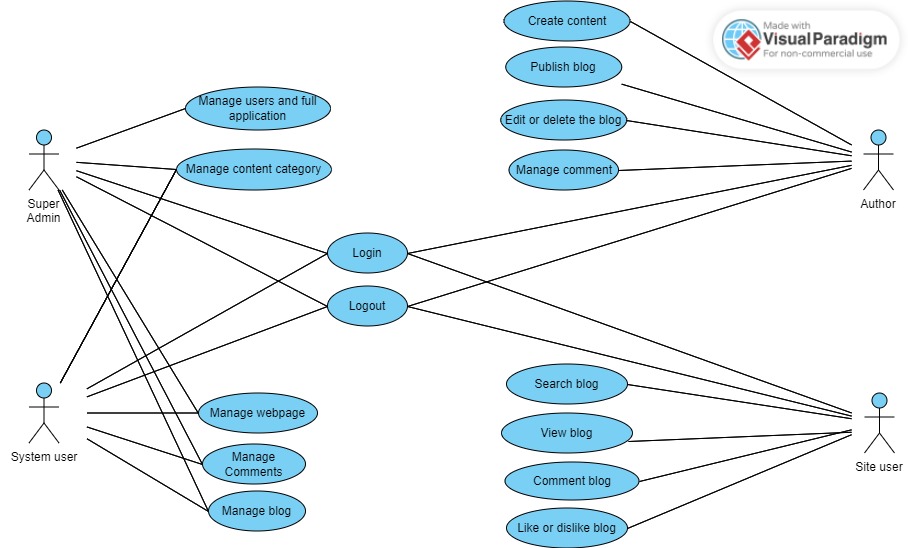


Fig 5.19 Use-Case diagram

**ACTIVITY DIAGRAM**

Activity diagram is an important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc. An activity specifies a particular operation of the system.

Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine diagrams. The below figure shows the activity diagram for our project.

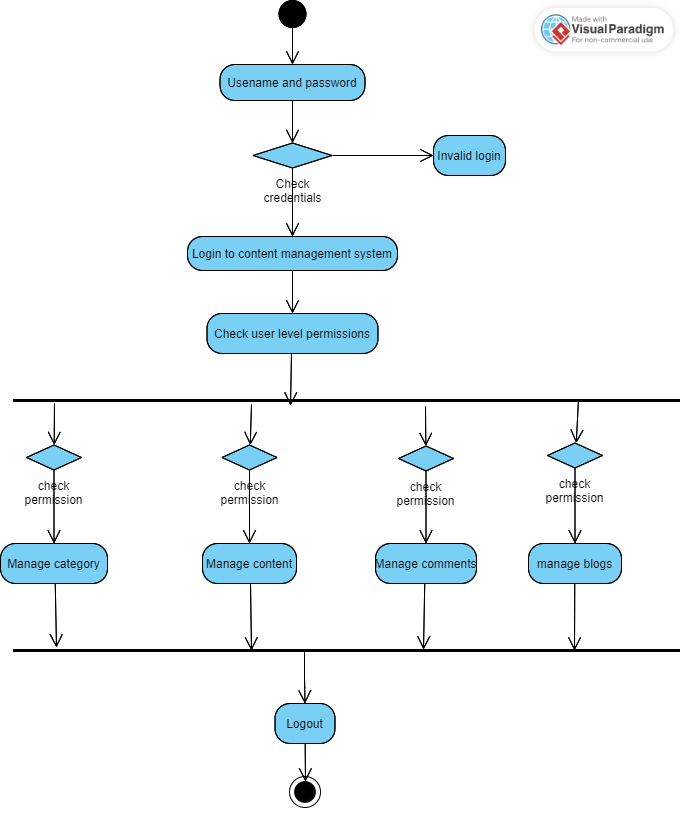


Fig 5.20 Activity diagram

1. **PROJECT DEVELOPMENT**
   1. **OVERVIEW:**

This project encompasses a frontend, backend, and database structure. The website's frontend is built with HTML, CSS, and React.js, while the backend utilizes Node.js and MongoDB for the database. Upon user registration, their credentials are securely stored in the database. Similarly, when a user publishes a blog, details such as creation date, author, likes, comments, and the blog's title are recorded in the database.[5] Only administrators and the original commenters have the authority to delete comments. Users can like a blog once and have the option to remove their like. This design ensures the website is intuitive and accessible to all users, including newcomers.

* 1. **CODE TEMPLATES:**
     1. **FRONT END :**

**LOGIN PAGE:** This file contains functionality to facilitate user authentication and registration. It includes options for users to log in using their existing credentials or to register by providing their name, username, and password. Additionally, it features a button that enables users to either log in or register.

import React, { useState, useEffect, useContext } from 'react';

import { TextField, Box, Button, Typography, styled } from '@mui/material';

import { useNavigate } from 'react-router-dom';

import { API } from '../../service/api';

import { DataContext } from '../../context/DataProvider';

const Component = styled(Box)`

    width: 400px;

    margin: auto;

    box-shadow: 5px 2px 5px 2px rgb(0 0 0/ 0.6);

`;

const Image = styled('img')({

    width: 100,

    display: 'flex',

    margin: 'auto',

    padding: '50px 0 0'

});

const Wrapper = styled(Box)`

    padding: 25px 35px;

    display: flex;

    flex: 1;

    overflow: auto;

    flex-direction: column;

    & > div, & > button, & > p {

        margin-top: 20px;

    }

`;

const LoginButton = styled(Button)`

    text-transform: none;

    background: #FB641B;

    color: #fff;

    height: 48px;

    border-radius: 2px;

`;

const SignupButton = styled(Button)`

    text-transform: none;

    background: #fff;

    color: #2874f0;

    height: 48px;

    border-radius: 2px;

    box-shadow: 0 2px 4px 0 rgb(0 0 0 / 20%);

`;

const Text = styled(Typography)`

    color: #878787;

    font-size: 12px;

`;

const Error = styled(Typography)`

    font-size: 10px;

    color: #ff6161;

    line-height: 0;

    margin-top: 10px;

    font-weight: 600;

`

const loginInitialValues = {

    username: '',

    password: ''

};

const signupInitialValues = {

    name: '',

    username: '',

    password: '',

};

const Login = ({ isUserAuthenticated }) => {

    const [login, setLogin] = useState(loginInitialValues);

    const [signup, setSignup] = useState(signupInitialValues);

    const [error, showError] = useState('');

    const [account, toggleAccount] = useState('login');

    const navigate = useNavigate();

    const { setAccount } = useContext(DataContext);

    const imageURL = 'https://www.sesta.it/wp-content/uploads/2021/03/logo-blog-sesta-trasparente.png';

  useEffect(() => {

        showError(false);

    }, [login])

    const onValueChange = (e) => {

        setLogin({ ...login, [e.target.name]: e.target.value });

    }

  const onInputChange = (e) => {

        setSignup({ ...signup, [e.target.name]: e.target.value });

    }

    const loginUser = async () => {

        let response = await API.userLogin(login);

        if (response.isSuccess) {

            showError('');

          sessionStorage.setItem('accessToken', `Bearer ${response.data.accessToken}`);

            sessionStorage.setItem('refreshToken', `Bearer ${response.data.refreshToken}`);

            setAccount({ name: response.data.name, username: response.data.username });

            isUserAuthenticated(true)

            setLogin(loginInitialValues);

            navigate('/');

        } else {

            showError('Something went wrong! please try again later');

        }

    }

    const signupUser = async () => {

        let response = await API.userSignup(signup);

        if (response.isSuccess) {

            showError('');

            setSignup(signupInitialValues);

            toggleAccount('login');

        } else {

            showError('Something went wrong! please try again later');

        }

    }

    const toggleSignup = () => {

        account === 'signup' ? toggleAccount('login') : toggleAccount('signup');

    }

  return (

        <Component>

            <Box>

                <Image src={imageURL} alt="blog" />

                {

                    account === 'login' ?

                        <Wrapper>

                            <TextField variant="standard" value={login.username} onChange={(e) => onValueChange(e)} name='username' label='Enter Username' />

                            <TextField variant="standard" value={login.password} onChange={(e) => onValueChange(e)} name='password' label='Enter Password' />

                          {error && <Error>{error}</Error>}

                            <LoginButton variant="contained" onClick={() => loginUser()} >Login</LoginButton>

                            <Text style={{ textAlign: 'center' }}>OR</Text>

                            <SignupButton onClick={() => toggleSignup()} style={{ marginBottom: 50 }}>Create an account</SignupButton>

                        </Wrapper> :

                        <Wrapper>

                            <TextField variant="standard" onChange={(e) => onInputChange(e)} name='name' label='Enter Name' />

                            <TextField variant="standard" onChange={(e) => onInputChange(e)} name='username' label='Enter Username' />

                            <TextField variant="standard" onChange={(e) => onInputChange(e)} name='password' label='Enter Password' />

                            <SignupButton onClick={() => signupUser()} >Signup</SignupButton>

                            <Text style={{ textAlign: 'center' }}>OR</Text>

                            <LoginButton variant="contained" onClick={() => toggleSignup()}>Already have an account</LoginButton>

                        </Wrapper>

                }

            </Box>

        </Component>

    )

}

export default Login;

**FIRST PAGE:** This code provides the landing page of our content management system website with heading BLOG and sub heading lets explore.

import { styled, Box, Typography } from '@mui/material';

const Image = styled(Box)`

    width: 100%;

    background: url(https://images.pexels.com/photos/1714208/pexels-photo-1714208.jpeg) center/55% repeat-x #000;

    height: 50vh;

    display: flex;

    flex-direction: column;

    align-items: center;

    justify-content: center;

`;

const Heading = styled(Typography)`

    font-size: 70px;

    color: #FFFFFF;

    line-height: 1

`;

const SubHeading = styled(Typography)`

    font-size: 20px;

    background: #FFFFFF;

`;

const Banner = () => {

    return (

        <Image>

            <Heading>BLOG</Heading>

            <SubHeading>Let's Explore</SubHeading>

        </Image>

    )

}

export default Banner;

**CATEGORIES BAR:** This code creates a table on the lest hand side of our webpage with all the categories of blogs like movies, sports, interviews etc.

import { Button, Table, TableHead, TableRow, TableCell, TableBody, styled } from '@mui/material';

import { Link, useSearchParams } from 'react-router-dom';

import { categories } from '../../constants/data';

// import SearchBar from './SearchBar';

const StyledTable = styled(Table)`

    border: 1px solid rgba(224, 224, 224, 1);

`;

const StyledButton = styled(Button)`

    margin: 20px;

    width: 85%;

    background: #6495ED;

    color: #fff;

    text-decoration: none;

`;

const StyledLink = styled(Link)`

    text-decoration: none;

    color: inherit;

`;

const Categories = () => {

    const [searchParams] = useSearchParams();

    const category = searchParams.get('category');

    return (

        <>

            <Link to={`/create?category=${category || ''}`} style={{ textDecoration: 'none' }}>

                <StyledButton variant="contained">Create Blog</StyledButton>

            </Link>

            <StyledTable>

                <TableHead>

                    <TableRow>

                        <TableCell>

                            <StyledLink to={"/"}>

                                All Categories

                            </StyledLink>

                        </TableCell>

                    </TableRow>

                </TableHead>

                <TableBody>

                    {

                        categories.map(category => (

                            <TableRow key={category.id}>

                                <TableCell>

                                    <StyledLink to={`/?category=${category.type}`}>

                                        {category.type}

                                    </StyledLink>

                                </TableCell>

                            </TableRow>

                        ))

                    }

                </TableBody>

            </StyledTable>

        </>

    )

}

export default Categories;

**CREATE POST:** This code is designed to generate a post, complete with various fields such as title, description, and image. Additionally, it incorporates a "publish" button to make the content available to viewers. Furthermore, it includes a button for users to express their appreciation by liking the content.

import React, { useState, useEffect, useContext } from 'react';

import { styled, Box, TextareaAutosize, Button, InputBase, FormControl  } from '@mui/material';

import { AddCircle as Add } from '@mui/icons-material';

import { useNavigate, useLocation } from 'react-router-dom';

import { API } from '../../service/api';

import { DataContext } from '../../context/DataProvider';

const Container = styled(Box)(({ theme }) => ({

    margin: '50px 100px',

    [theme.breakpoints.down('md')]: {

        margin: 0

    }

}));

const Image = styled('img')({

    width: '100%',

    height: '50vh',

    objectFit: 'cover'

});

const StyledFormControl = styled(FormControl)`

    margin-top: 10px;

    display: flex;

    flex-direction: row;

`;

const InputTextField = styled(InputBase)`

    flex: 1;

    margin: 0 30px;

    font-size: 25px;

`;

const Textarea = styled(TextareaAutosize)`

    width: 100%;

    border: none;

    margin-top: 50px;

    font-size: 18px;

    &:focus-visible {

        outline: none;

    }

`;

const initialPost = {

    title: '',

    description: '',

    picture: '',

    username: '',

    categories: '',

    createdDate: new Date(),

    likes:0,

}

const CreatePost = () => {

    const navigate = useNavigate();

    const location = useLocation();

  const [post, setPost] = useState(initialPost);

    const [file, setFile] = useState('');

    const { account } = useContext(DataContext);

    const url = post.picture ? post.picture : 'https://images.unsplash.com/photo-1543128639-4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0dXB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

    useEffect(() => {

        const getImage = async () => {

            if(file) {

                const data = new FormData();

                data.append("name", file.name);

                data.append("file", file);

                const response = await API.uploadFile(data);

                post.picture = response.data;

            }

        }

        getImage();

        post.categories = location.search?.split('=')[1] || 'All';

        post.username = account.username;

    }, [file])

    const savePost = async () => {

        await API.createPost(post);

        navigate('/');

    }

    const handleChange = (e) => {

        setPost({ ...post, [e.target.name]: e.target.value });

    }

    return (

        <Container>

            <Image src={url} alt="post" />

          <StyledFormControl>

                <label htmlFor="fileInput">

                    <Add fontSize="large" color="action" />

                </label>

                <input

                    type="file"

                    id="fileInput"

                    style={{ display: "none" }}

                    onChange={(e) => setFile(e.target.files[0])}

                />

                <InputTextField onChange={(e) => handleChange(e)} name='title' placeholder="Title" />

                <Button onClick={() => savePost()} variant="contained" color="primary">Publish</Button>

            </StyledFormControl>

            <Textarea

                rowsMin={5}

                placeholder="Tell your story..."

                name='description'

                onChange={(e) => handleChange(e)}

            />

        </Container>

    )

}

export default CreatePost;

**UPDATE POST:** This code serves the purpose of allowing users to update their previously published blog posts. It provides functionality for users to modify the content of their existing blogs as needed.

import React, { useState, useEffect } from 'react';

import { Box, styled, TextareaAutosize, Button, FormControl, InputBase } from '@mui/material';

import { AddCircle as Add } from '@mui/icons-material';

import { useNavigate, useParams } from 'react-router-dom';

import { API } from '../../service/api';

const Container = styled(Box)(({ theme }) => ({

    margin: '50px 100px',

    [theme.breakpoints.down('md')]: {

        margin: 0

    }

}));

const Image = styled('img')({

    width: '100%',

    height: '50vh',

    objectFit: 'cover'

});

const StyledFormControl = styled(FormControl)`

    margin-top: 10px;

    display: flex;

    flex-direction: row;

`;

const InputTextField = styled(InputBase)`

    flex: 1;

    margin: 0 30px;

    font-size: 25px;

`;

const StyledTextArea = styled(TextareaAutosize)`

    width: 100%;

    border: none;

    margin-top: 50px;

    font-size: 18px;

    &:focus-visible {

        outline: none;

    }

`;

const initialPost = {

    title: '',

    description: '',

    picture: '',

    username: 'codeforinterview',

    categories: 'Tech',

    createdDate: new Date()

}

const Update = () => {

    const navigate = useNavigate();

    const [post, setPost] = useState(initialPost);

    const [file, setFile] = useState('');

    const [imageURL, setImageURL] = useState('');

  const { id } = useParams();

    const url = 'https://images.unsplash.com/photo-1543128639-4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0dXB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

    useEffect(() => {

        const fetchData = async () => {

            let response = await API.getPostById(id);

            if (response.isSuccess) {

                setPost(response.data);

            }

        }

        fetchData();

    }, []);

    useEffect(() => {

        const getImage = async () => {

            if(file) {

                const data = new FormData();

                data.append("name", file.name);

                data.append("file", file);

                const response = await API.uploadFile(data);

                if (response.isSuccess) {

                    post.picture = response.data;

                    setImageURL(response.data);

                }

            }

        }

        getImage();

    }, [file])

  const updateBlogPost = async () => {

        await API.updatePost(post);

        navigate(`/details/${id}`);

    }

    const handleChange = (e) => {

        setPost({ ...post, [e.target.name]: e.target.value });

    }

  return (

        <Container>

            <Image src={post.picture || url} alt="post" />

            <StyledFormControl>

                <label htmlFor="fileInput">

                    <Add fontSize="large" color="action" />

                </label>

                <input

                    type="file"

                    id="fileInput"

                    style={{ display: "none" }}

                    onChange={(e) => setFile(e.target.files[0])}

                />

                <InputTextField onChange={(e) => handleChange(e)} value={post.title} name='title' placeholder="Title" />

                <Button onClick={() => updateBlogPost()} variant="contained" color="primary">Update</Button>

            </StyledFormControl>

            <StyledTextArea

                rowsMin={5}

                placeholder="Tell your story..."

                name='description'

                onChange={(e) => handleChange(e)}

                value={post.description}

            />

        </Container>

    )

}

export default Update;

**LIKE A POST:** This functionality of this code is to like and dislike a blog.

import { useState, useEffect, useContext } from 'react';

import { Box, Typography, IconButton, styled } from '@mui/material';

import { Favorite, FavoriteBorder, Delete, Edit } from '@mui/icons-material';

import { Link, useNavigate, useParams } from 'react-router-dom';

import { API } from '../../service/api';

import { DataContext } from '../../context/DataProvider';

import Comments from './comments/Comments';

const Container = styled(Box)(({ theme }) => ({

    margin: '50px 100px',

    [theme.breakpoints.down('md')]: {

        margin: 0

    },

}));

const Image = styled('img')({

    width: '100%',

    height: '50vh',

    objectFit: 'cover'

});

const EditIcon = styled(Edit)`

    margin: 5px;

    padding: 5px;

    border: 1px solid #878787;

    border-radius: 10px;

`;

const DeleteIcon = styled(Delete)`

    margin: 5px;

    padding: 5px;

    border: 1px solid #878787;

    border-radius: 10px;

`;

const Heading = styled(Typography)`

    font-size: 38px;

    font-weight: 600;

    text-align: center;

    margin: 50px 0 10px 0;

`;

const Author = styled(Box)(({ theme }) => ({

    color: '#878787',

    display: 'flex',

    margin: '20px 0',

    [theme.breakpoints.down('sm')]: {

        display: 'block'

    },

}));

const DetailView = () => {

    const url = 'https://images.unsplash.com/photo-1543128639-4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0dXB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

    const [post, setPost] = useState({});

    const [likes, setLikes] = useState(0);

    const [liked, setLiked] = useState(false); // State for tracking if post is liked

    const { account } = useContext(DataContext);

    const navigate = useNavigate();

    const { id } = useParams();

  useEffect(() => {

        const fetchData = async () => {

            const response = await API.getPostById(id);

            if (response.isSuccess) {

                setPost(response.data);

                setLikes(response.data.likes || 0);

                // Check if current user has liked the post

                setLiked(response.data.likes && response.data.likes > 0);

            }

        };

        fetchData();

    }, [id]);

    const deleteBlog = async () => {

        await API.deletePost(post.\_id);

        navigate('/');

    };

  const handleLike = async () => {

        let updatedLikes = likes;

        if (!liked) {

            updatedLikes += 1;

        } else {

            updatedLikes -= 1;

        }

        setLikes(updatedLikes);

        setLiked(!liked); // Toggle liked state

        // Update likes count in the database

        await API.updatePost({ ...post, likes: updatedLikes });

    };

    return (

        <Container>

            <Image src={post.picture || url} alt="post" />

            <Box style={{ float: 'right' }}>

                {account.username === post.username && (

                    <>

                        <Link to={`/update/${post.\_id}`}>

                            <EditIcon color="primary" />

                        </Link>

                        <DeleteIcon onClick={() => deleteBlog()} color="error" />

                    </>

                )}

            </Box>

            <Heading>{post.title}</Heading>

            <Author>

                <Link to={`/?username=${post.username}`} style={{ textDecoration: 'none', color: 'inherit' }}>

                    <Typography>

                        Author: <span style={{ fontWeight: 600 }}>{post.username}</span>

                    </Typography>

                </Link>

                <Typography style={{ marginLeft: 'auto' }}>{new Date(post.createdDate).toDateString()}</Typography>

            </Author>

            <Typography>{post.description}</Typography>

            {/\* Like/Dislike button \*/}

            <IconButton onClick={handleLike} color={liked ? "secondary" : "primary"}>

                {liked ? <Favorite /> : <FavoriteBorder />}

            </IconButton>

            {/\* Display the number of likes \*/}

            <Typography>{likes} {likes === 1 ? 'like' : 'likes'}</Typography>

            <Comments post={post} />

        </Container>

    );

};

export default DetailView;

**ADD COMMENT:** This file enables users to interact with other people's blogs by adding comments .

import { useState, useEffect, useContext } from 'react';

import { Box, TextareaAutosize, Button, styled } from '@mui/material';

import { DataContext } from '../../../context/DataProvider';

import { API } from '../../../service/api';

//components

import Comment from './Comment';

const Container = styled(Box)`

    margin-top: 100px;

    display: flex;

`;

const Image = styled('img')({

    width: 50,

    height: 50,

    borderRadius: '50%'

});

const StyledTextArea = styled(TextareaAutosize)`

    height: 100px !important;

    width: 100%;

    margin: 0 20px;

`;

const initialValue = {

    name: '',

    postId: '',

    date: new Date(),

    comments: ''

}

const Comments = ({ post }) => {

    const url = 'https://static.thenounproject.com/png/12017-200.png'

    const [comment, setComment] = useState(initialValue);

    const [comments, setComments] = useState([]);

    const [toggle, setToggle] = useState(false);

    const { account } = useContext(DataContext);

    useEffect(() => {

        const getData = async () => {

            const response = await API.getAllComments(post.\_id);

            if (response.isSuccess) {

                setComments(response.data);

            }

        }

        getData();

    }, [toggle, post]);

    const handleChange = (e) => {

        setComment({

            ...comment,

            name: account.username,

            postId: post.\_id,

            comments: e.target.value

        });

    }

    const addComment = async() => {

        await API.newComment(comment);

        setComment(initialValue)

        setToggle(prev => !prev);

    }

    return (

        <Box>

            <Container>

                <Image src={url} alt="dp" />

                <StyledTextArea

                    rowsMin={5}

                    placeholder="what's on your mind?"

                    onChange={(e) => handleChange(e)}

                    value={comment.comments}

                />

                <Button

                    variant="contained"

                    color="primary"

                    size="medium"

                    style={{ height: 40 }}

                    onClick={(e) => addComment(e)}

                >Post</Button>

            </Container>

            <Box>

                {

                    comments && comments.length > 0 && comments.map(comment => (

                        <Comment comment={comment} setToggle={setToggle} />

                    ))

                }

            </Box>

        </Box>

    )

}

export default Comments;

**DELETE COMMENT:** This file enables users to delete the comment they posted.

import { useContext } from "react";

import { Typography, Box, styled } from "@mui/material";

import { Delete } from '@mui/icons-material';

import { API } from '../../../service/api';

import { DataContext } from "../../../context/DataProvider";

const Component = styled(Box)`

    margin-top: 30px;

    background: #F5F5F5;

    padding: 10px;

`;

const Container = styled(Box)`

    display: flex;

    margin-bottom: 5px;

`;

const Name = styled(Typography)`

    font-weight: 600,

    font-size: 18px;

    margin-right: 20px;

`;

const StyledDate = styled(Typography)`

    font-size: 14px;

    color: #878787;

`;

const DeleteIcon = styled(Delete)`

    margin-left: auto;

`;

const Comment = ({ comment, setToggle }) => {

    const { account } = useContext(DataContext)

    const removeComment = async () => {

       await API.deleteComment(comment.\_id);

       setToggle(prev => !prev);

    }

    return (

        <Component>

            <Container>

                <Name>{comment.name}</Name>

                <StyledDate>{new Date(comment.date).toDateString()}</StyledDate>

                { comment.name === account.username && <DeleteIcon onClick={() => removeComment()} /> }

            </Container>

            <Typography>{comment.comments}</Typography>

        </Component>

    )

}

export default Comment;

**SEARCH POST:** This code facilitates users in searching for specific blogs of their choice using the username and post title as search criteria.

import React, { useEffect, useState } from 'react';

import { Grid, Box, TextField } from '@mui/material';

import { Link, useSearchParams } from 'react-router-dom';

import { API } from '../../../service/api';

import Post from './Post';

const Posts = () => {

    const [posts, setPosts] = useState([]);

    const [filteredPosts, setFilteredPosts] = useState([]);

    const [searchTerm, setSearchTerm] = useState('');

    const [searchParams] = useSearchParams();

    const category = searchParams.get('category');

    useEffect(() => {

        const fetchData = async () => {

            let response = await API.getAllPosts({ category: category || '' });

            if (response.isSuccess) {

                setPosts(response.data);

                setFilteredPosts(response.data); // Initialize filtered posts with all posts

            }

        }

        fetchData();

    }, [category]);

    const handleSearch = (searchTerm) => {

        const filtered = posts.filter(post =>post.title.toLowerCase().includes(searchTerm.toLowerCase())|| post.username.toLowerCase().includes(searchTerm.toLowerCase()));

        setFilteredPosts(filtered);

    };

    const handleChange = (e) => {

        const searchTerm = e.target.value;

        setSearchTerm(searchTerm);

        handleSearch(searchTerm);

    };

    return (

        <>

            <TextField

                label="Search by Username or title"

                variant="outlined"

                value={searchTerm}

                onChange={handleChange}

            />

            {

                filteredPosts?.length ? (

                    <Grid container spacing={2}>

                        {filteredPosts.map(post => (

                            <Grid item key={post.\_id} lg={3} sm={4} xs={12}>

                                <Link style={{ textDecoration: 'none', color: 'inherit' }} to={`details/${post.\_id}`}>

                                    <Post post={post} />

                                </Link>

                            </Grid>

                        ))}

                    </Grid>

                ) : (

                    <Box style={{ color: '878787', margin: '30px 80px', fontSize: 18 }}>

                        No posts found for the specified username or title

                    </Box>

                )

            }

        </>

    );

};

export default Posts;

* + 1. **BACKEND:**

**Category.js:**

import mongoose from 'mongoose';

const CategorySchema = mongoose.Schema({

    name: {

        type: String,

        required: true

    }

});

const category = mongoose.model('category', CategorySchema);

export default category;

**Comment.js**

import mongoose from 'mongoose';

const CommentSchema = mongoose.Schema({

    name: {

        type: String,

        required: true,

    },

    postId: {

        type: String,

        required: true

    },

    date: {

        type: String,

        required: true

    },

    comments: {

        type: String,

        required: true

    }

});

const comment = mongoose.model('comment', CommentSchema);

export default comment;

**Item.js**

import mongoose from 'mongoose';

// Define the schema for the Item model

const itemSchema =  mongoose.Schema({

  title: {

    type: String,

    required: true,

    unique: true

  },

  description: {

    type: String,

    required: true

  },

  picture: {

    type: String,

    required: false

  },

  username: {

    type: String,

    required: true

  },

  categories: {

    type: [String], // Assuming categories is an array of strings

    required: false

  },

  createdDate: {

    type: Date,

    default: Date.now

  },

  likes: {

    type: Number,

    default: 0 // Default value for likes is 0

}

});

// Create the Item model using the schema

const Item = mongoose.model('Item', itemSchema);

export default Item;

**Post.js**

import mongoose from 'mongoose';

const PostSchema = mongoose.Schema({

    title: {

        type: String,

        required: true,

        unique: true

    },

    description: {

        type: String,

        required: true

    },

    picture: {

        type: String,

        required: false

    },

    username: {

        type: String,

        required: true

    },

    categories: {

        type: Array,

        required: false

    },

    createdDate: {

        type: Date

    },

    likes: {

        type: Number,

        default: 0 // Default value for likes is 0

    }

});

const Post = mongoose.model('Post', PostSchema);

export default Post;

**Token.js**

import mongoose from 'mongoose';

const TokenSchema = mongoose.Schema({

    token: {

        type: String,

        required: true

    }

});

const token = mongoose.model('token', TokenSchema);

export default token;

**User.js**

import mongoose from 'mongoose';

const userSchema = mongoose.Schema({

    name: {

        type: String,

        required: true

    },

    username: {

        type: String,

        required: true,

        unique: true

    },

    password: {

        type: String,

        required: true

    }

});

const user = mongoose.model('user', userSchema);

export default user;

**Upload.js**

import multer from 'multer';

import { GridFsStorage } from 'multer-gridfs-storage';

const storage = new GridFsStorage({

    url: `mongodb://user:codeforinterview@ac-tz54972-shard-00-00.ta9g9bn.mongodb.net:27017,ac-tz54972-shard-00-01.ta9g9bn.mongodb.net:27017,ac-tz54972-shard-00-02.ta9g9bn.mongodb.net:27017/?ssl=true&replicaSet=atlas-4jt23a-shard-0&authSource=admin&retryWrites=true&w=majority`,

    options: { useNewUrlParser: true },

    file: (request, file) => {

        const match = ["image/png", "image/jpg"];

        if(match.indexOf(file.memeType) === -1)

            return`${Date.now()}-blog-${file.originalname}`;

        return {

            bucketName: "photos",

            filename: `${Date.now()}-blog-${file.originalname}`

        }

    }

});

export default multer({storage});

**Route.js:** This code sets up routes for various functionalities in an Express.js application. Here's a breakdown of what each route does:

* ./login: Handles user login.
* ./signup: Handles user signup.
* ./logout: Handles user logout.
* ./token: Creates a new authentication token.
* ./create: Creates a new post (requires authentication token).
* ./update/:id: Updates an existing post (requires authentication token).
* ./delete/:id: Deletes a post (requires authentication token).
* ./post/:id: Retrieves a specific post (requires authentication token).
* ./posts: Retrieves all posts (requires authentication token).
* ./file/upload: Uploads a file (requires authentication token).
* ./file/:filename: Retrieves an uploaded file.
* ./comment/new: Adds a new comment to a post (requires authentication token).
* ./comments/:id: Retrieves comments for a specific post (requires authentication token).
* ./comment/delete/:id: Deletes a comment (requires authentication token).

import express from 'express';

import { createPost , updatePost, deletePost, getPost, getAllPosts } from '../controller/post-controller.js';

import { uploadImage, getImage } from '../controller/image-controller.js';

import { newComment, getComments, deleteComment } from '../controller/comment-controller.js';

import { loginUser, singupUser, logoutUser } from '../controller/user-controller.js';

import { authenticateToken, createNewToken } from '../controller/jwt-controller.js';

import upload from '../utils/upload.js';

const router = express.Router();

router.post('/login', loginUser);

router.post('/signup', singupUser);

router.post('/logout', logoutUser);

router.post('/token', createNewToken);

router.post('/create', authenticateToken, createPost);

router.put('/update/:id', authenticateToken, updatePost);

router.delete('/delete/:id', authenticateToken, deletePost);

router.get('/post/:id', authenticateToken, getPost);

router.get('/posts', authenticateToken, getAllPosts);

router.post('/file/upload', upload.single('file'), uploadImage);

router.get('/file/:filename', getImage);

router.post('/comment/new', authenticateToken, newComment);

router.get('/comments/:id', authenticateToken, getComments);

router.delete('/comment/delete/:id', authenticateToken, deleteComment);

export default router;

* + 1. **DATABASE:**

**Db.js:** This code establishes a connection to a MongoDB database using the Mongoose library in a Node.js environment. It takes the username and password as parameters to authenticate the connection to the MongoDB Atlas cluster specified by the connection URL. Once the connection is established successfully, it logs a message confirming the successful connection to the database. If there's an error during the connection process, it logs an error message instead. This code essentially encapsulates the process of connecting to a MongoDB database using Mongoose into a reusable function.

import mongoose from 'mongoose';

const Connection = async (username, password) => {

    const URL = `mongodb://${username}:${password}@ac-tz54972-shard-00-00.ta9g9bn.mongodb.net:27017,ac-tz54972-shard-00-01.ta9g9bn.mongodb.net:27017,ac-tz54972-shard-00-02.ta9g9bn.mongodb.net:27017/?ssl=true&replicaSet=atlas-4jt23a-shard-0&authSource=admin&retryWrites=true&w=majority`;

    try {

        await mongoose.connect(URL, { useNewUrlParser: true })

        console.log('Database connected successfully');

    } catch (error) {

        console.log('Error while connecting to the database ', error);

    }

};

export default Connection;

1. **RESULTS**

This is the registration page where new users can create their account in our website with the credentials like name, username and password.

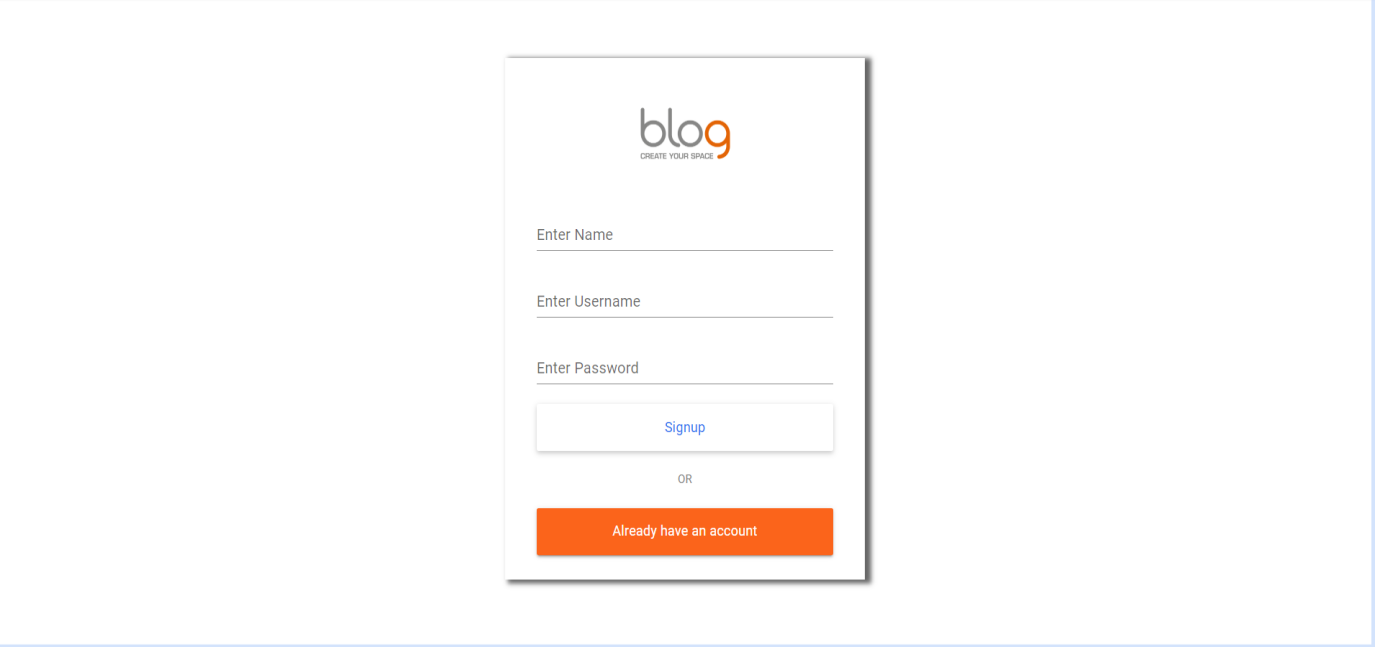
****

Fig 7.1 Registration page

This is the login page where already registered users can login into our website and can create blogs, search blogs, comment on others blogs etc.

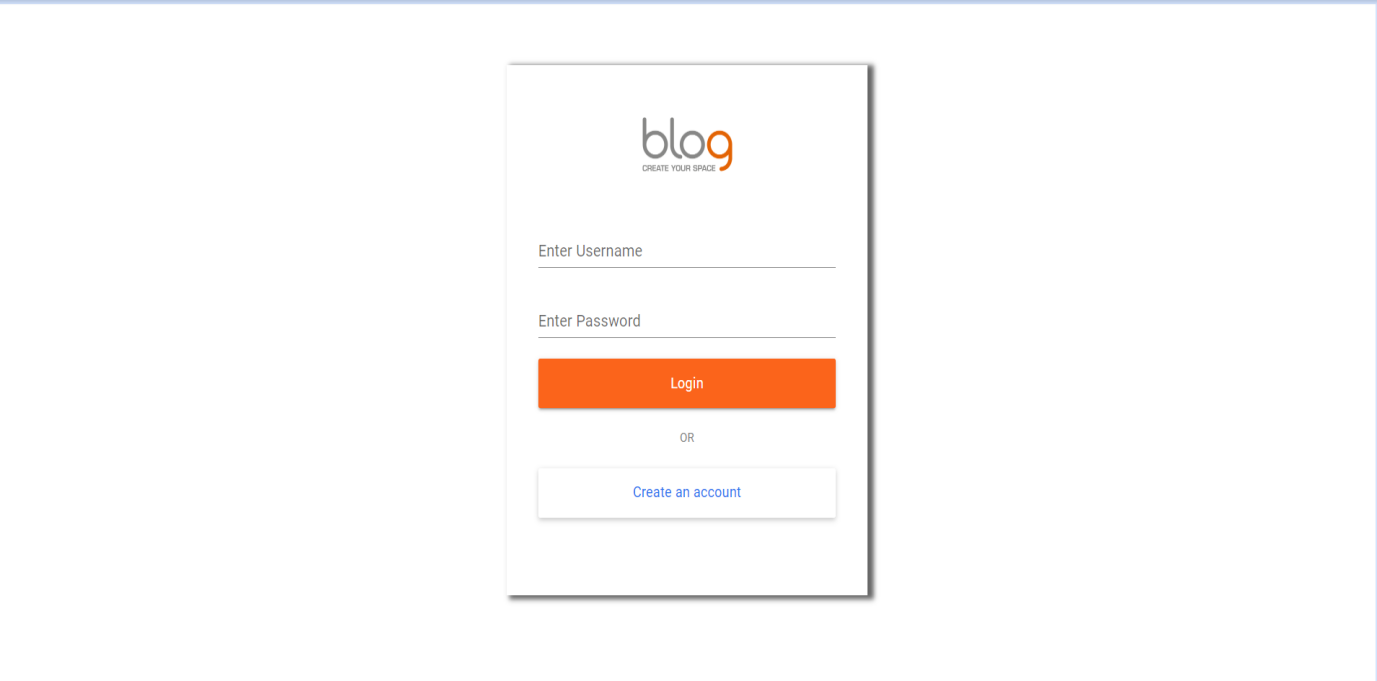


Fig 7.2 Login page

This is the landing page of our website. Upon user authentication, this is the view that greets them.

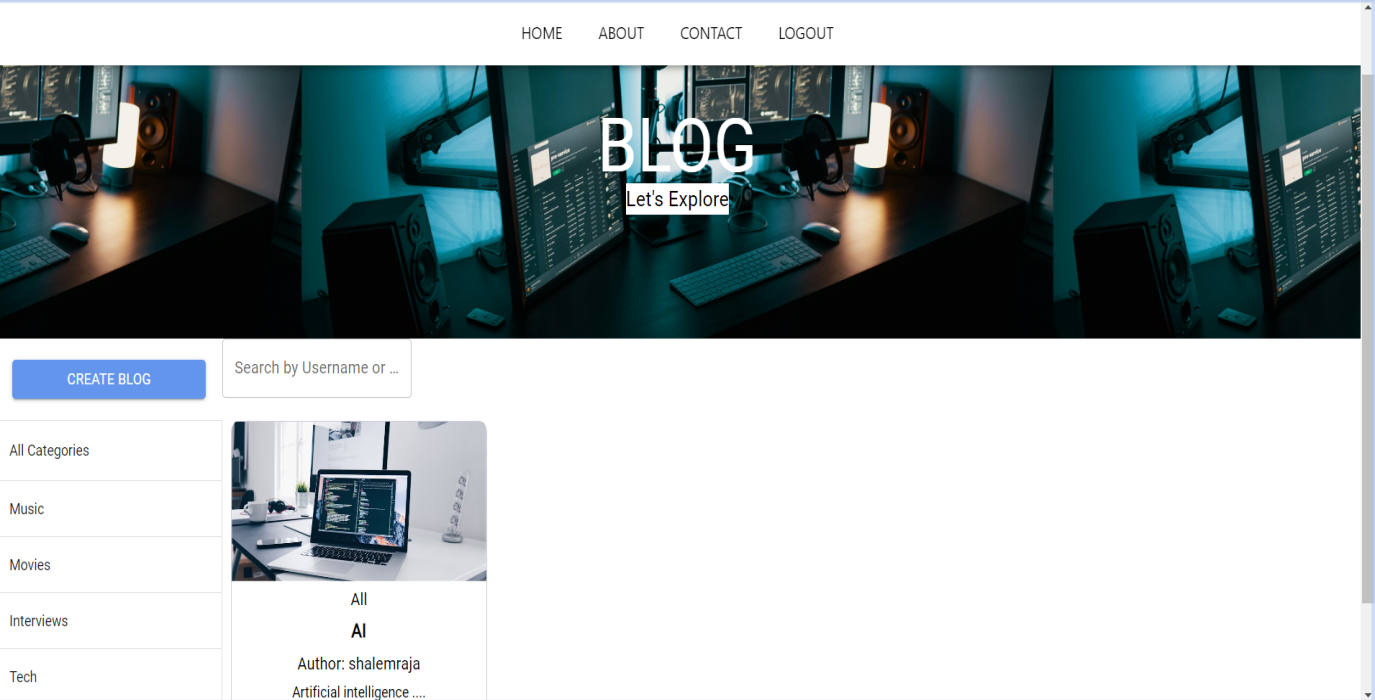
****

Fig 7.3 Landing page

This is the side bar where we can all the categories that we include in our page like sports, education etc.

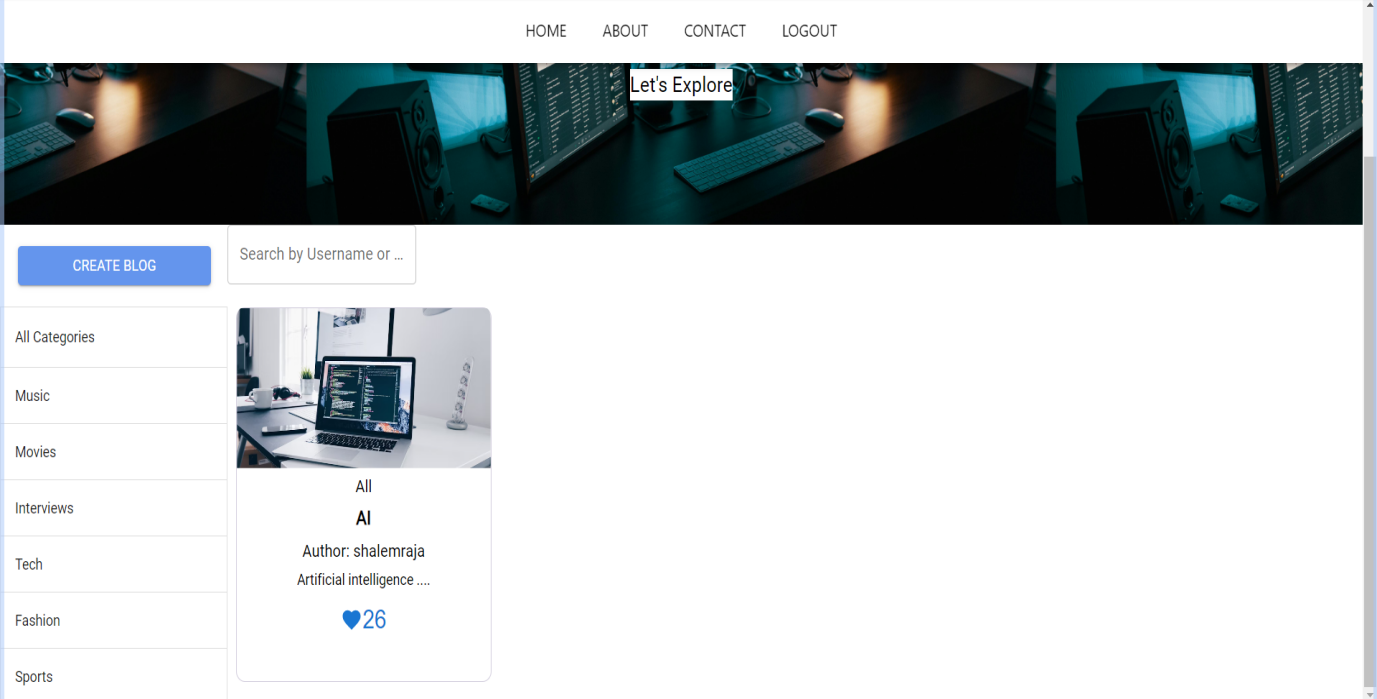


Fig 7.4 Categories bar

This page enable us to create a blog with the title and description where we can write what ever we want and it contains a publish button to publish our content.

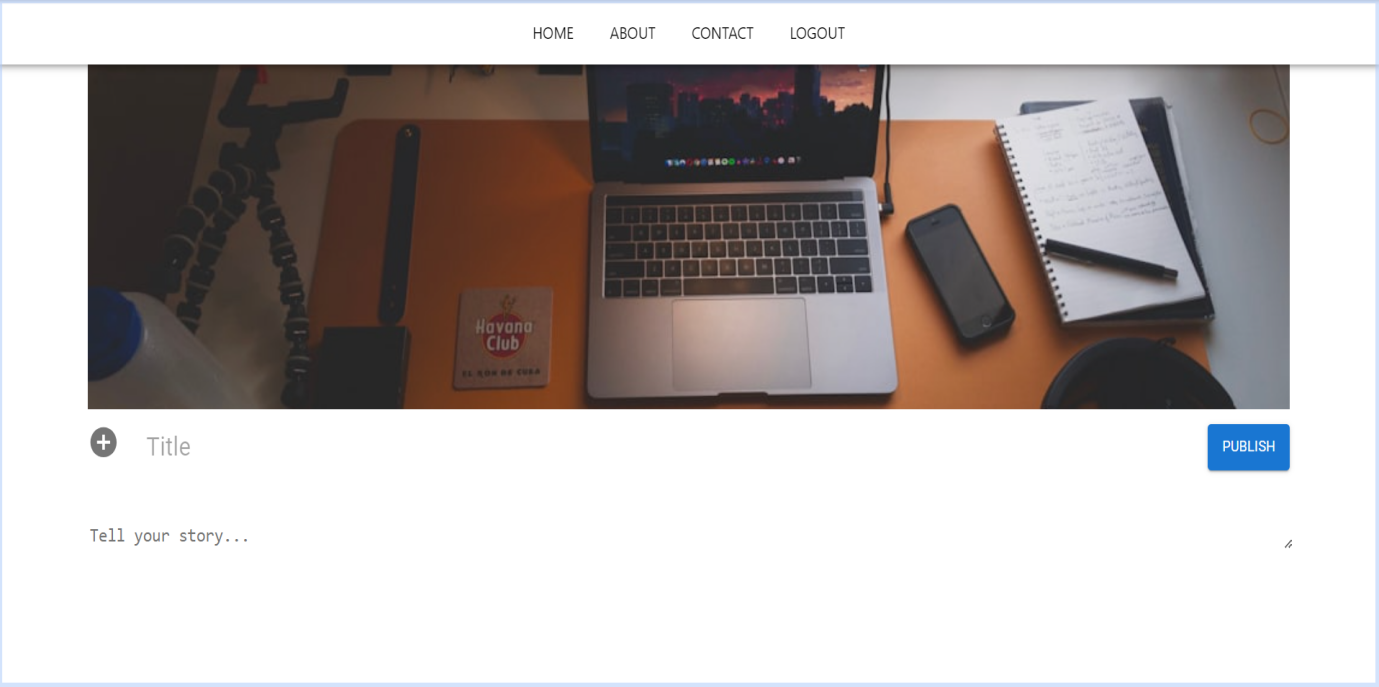


Fig 7.5 Content creation

This consists the comment section and liking section. This helps us to like and dislike the blog or to interact with the help of comments.

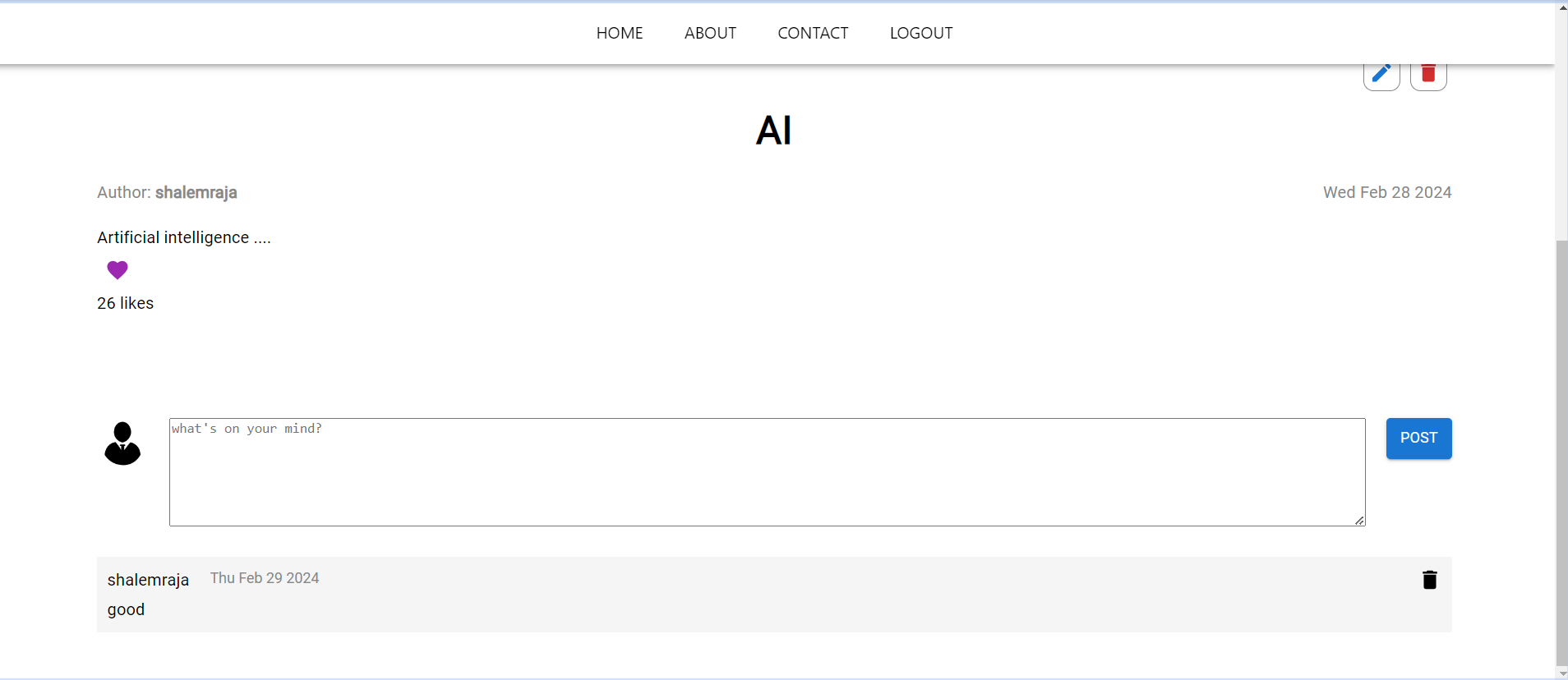


Fig 7.6 Comment and Like

This image shows the database when a new user registers in the website. The new user credentials will be stored in our database.

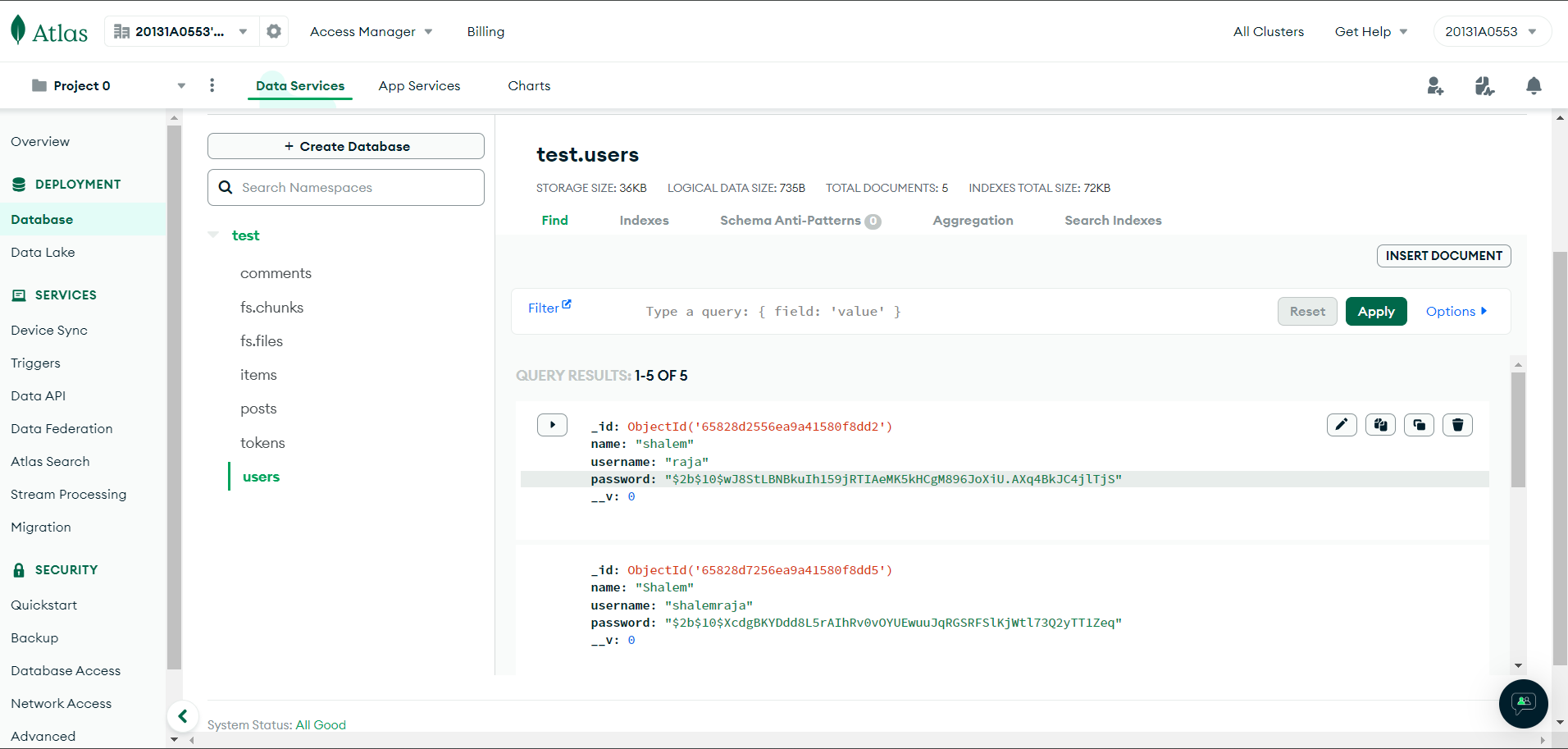


Fig 7.7 User credentials stored in database

The image shows that the blogs created in our website are stored in the database. The information like blog id, title, description, username, created date etc are stored in the database.

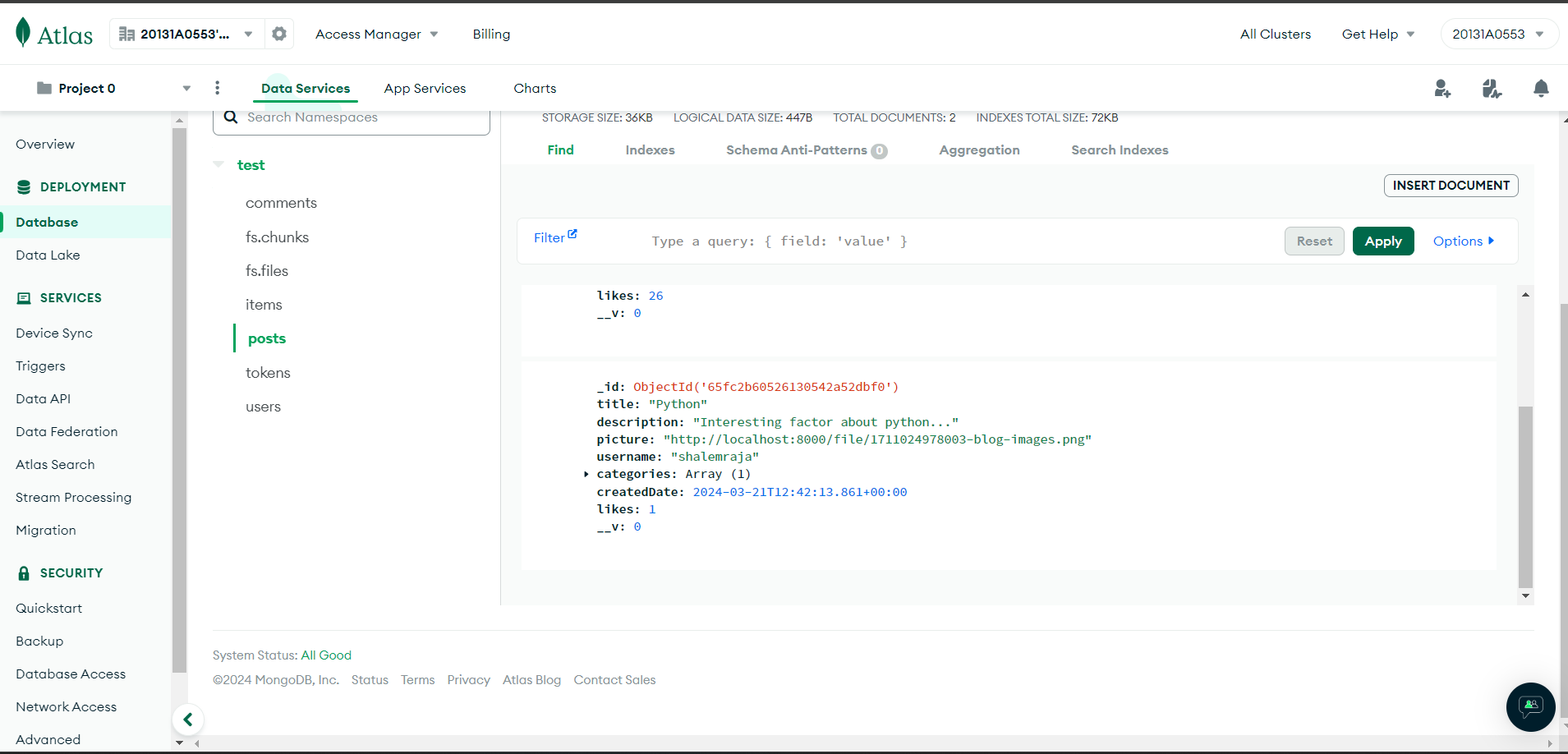


Fig 7.8 Blogs data stored in the database

1. **CONCLUSION**

This project furnishes a web application utilizing React.js for the frontend and Node.js for the backend. It aims to deliver a dependable and resilient website, enabling users to securely log in and register using their credentials. It offers a comprehensive solution for managing digital content effectively. Through features such as user authentication, post creation, editing, and deletion, image uploads, comment management, and user management, like management it provides a robust platform for content creators to publish and manage their content. Additionally, the integration of authentication tokens enhances security, ensuring that only authorized users can access and modify content. Overall, the project streamlines content management processes, fostering a user-friendly and organized environment for both content creators and consumers.

1. **FUTURE SCOPE**

The website developed in this project empowers users to create content in their preferred categories, allowing them to express themselves freely through blogs. Users have the flexibility to write about any topic they desire and can seamlessly include images to enrich their posts. Upon publishing, these blogs become accessible to other users of our platform, who can engage with them by liking, commenting, and sharing their thoughts. Additionally, users can easily search for blogs based on their interests or specific requirements. Looking ahead, there is immense potential for further enhancement through voice interfaces. As voice assistants and smart speakers gain popularity, integrating voice and conversational interfaces into the CMS can revolutionize how content is consumed and interacted with. Furthermore, incorporating AI-powered features like content personalization, automated tagging, and natural language processing holds promise for enhancing user experiences and streamlining content management processes.

1. **REFERENCE LINKS**

[1] <https://kinsta.com/knowledgebase/content-management-system/>

[2] <https://medium.com/>

[3] <https://wptravel.io/what-is-a-content-management-system-easy-beginners-guide/>

[4] <https://ieeexplore.ieee.org/document/9675311>

[5] <https://www.netsolutions.com/insights/content-management-system/>

[6] <https://www.optimizely.com/optimization-glossary/contentmanagement-system/>

[7] <https://www.joomla.org/core-features.html>

[8] <https://www.progress.com/blogs/digital-experience>

[9] <https://www.oracle.com/in/content-management/what-is-cms/>

[10] [https://www.techtarget.com/searchcontentmanagement/definition/content-management-systemCMS#:~:text=A%20CMS%20provides%20a%20graphical,content%20delivery%20application%20(CDA).](https://www.techtarget.com/searchcontentmanagement/definition/content-management-system-CMS#:~:text=A%20CMS%20provides%20a%20graphical,content%20delivery%20application%20(CDA).)

[11] <https://www.nibusinessinfo.co.uk/content/important-cms-features-and-functions>

[12] <https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document>

[13] <https://miro.com/diagramming/what-is-a-uml-diagram/>