BLOG APPLICATION

Submitted by

LAVANYA D

Reg No: 1P23MC033

In partial fulfillment of the requirement for the award of the Degree of

Master of Computer Applications

Bharathiar University

Coimbatore

Under the Internal Supervision of

Ms. K Narmatha MCA.,

Assistant Professor

School of Computer Studies-MCA

Master of Computer Applications,

RVS College of Arts and Science (Autonomous),

Sulur, Coimbatore – 641 402

Nov 2024

RVS COLLEGE OF ARTS AND SCIENCE(AUTONOMOUS)

AFFLIATED TO BHARATHIAR UNIVERSITY

NAAC Re-accredited & ISO

9000:2008 Certified SULUR,

COIMBATORE-641402.

Nov 2024

Register Number – 1P23MC033

Certified *bonafide* original record of work done by **LAVANYA D**

Internal Supervisor **Director**

Submitted for the project Evaluation and *Viva-Voce* held on**………………….**

Internal Examiner **External Examiner**

DECLARATION

I, **LAVANYA D**, hereby declare that the project entitled **BLOG APPLICATION,** submitted to the School of Computer Studies, RVS College of Arts and Science, in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications is a record of original project work done by me during the period Jun 2024 to Nov 2024 under the internal supervision of **MS.K.NARMATHA MCA., ASSISTANT PROFESSOR, RVS COLLEGE OF ARTS AND SCIENCE (AUTONOMOUS)** From Bharathiar University, Coimbatore.

LAVANYA D
Signature of the Candidate

ACKNOWLEDGEMENTS

I express my sincere thanks to our Managing trustee **Dr.K.Senthil Ganesh MBA (USA)., MS (UK)., Ph.D.,** for providing us with the adequate faculty and laboratory resources for completing my project successfully.

I take this as a fine opportunity to express my sincere thanks to **Dr. T. Sivakumar M.Sc., M. Phil., Ph.D., Principal,** Rathnavel Subramaniam College of Arts and Science (Autonomous) for giving me the opportunity to undertake this project.

I express my sincere thanks to **Dr. P. Navaneetham M.Sc., M.Phil., Ph.D., Director (Administration), Department of Computer Science** for the help and advice throughout the project.

I express my sincere thanks to **Dr. S. Yamini M.Sc., (CC)., M. Phil., Ph.D., Director (Academic), Department of Computer Science** for her valuable guidance and prompt correspondence throughout the curriculum to complete the project.

I express my gratitude to **Ms. K. Narmatha MCA., Assistant Professor** for his valuable guidance, support, encouragement and motivation rendered by her throughout this project.

Finally, I express my sincere thanks to all other staff members and my dear friends, and all dear and near for helping me to complete this project.

LAVANYA D

ABSTRACT

Abstracting a blog app with React Native, Node.js, and MongoDB requires a well-structured full-stack approach that facilitates user-friendly content management. On the frontend, you'll develop a React Native app that allows users to view, create, and publish blog posts. Start by setting up a React Native project using Expo or React Native CLI. Implement user authentication to enable user sign-up and log-in, which can be achieved through third-party libraries. Following this, create components for viewing a list of blog posts, creating new posts, and editing/deleting existing posts. Additionally, develop a user profile page that displays user information and authored posts, and use navigation libraries like React Navigation to facilitate seamless navigation.

To complete this app, it's crucial to connect the frontend to the backend built with Node.js. In the backend, establish a Node.js project using a framework like Express.js and configure a MongoDB database for storing blog posts and user data. Create RESTful API routes for various functionalities, including user registration, authentication, CRUD operations for blog posts, fetching user-specific posts, and retrieving all blog posts for viewing. Ensure that user authentication middleware is in place to protect specific routes, such as editing or deleting posts.

In the MongoDB database, define data models for users and blog posts, preferably using an Object Data Modeling (ODM) library like Mongoose. Develop a well-defined schema for these data models

CONTENTS

CHAPTER-1

1.Introduction

This document outlines the development and architecture of BLOG APP, a real-time helping application designed to facilitate seamless communication among users. In an era where the digital landscape thrives on content and connectivity, the creation of a versatile and engaging blog application has become more compelling than ever. In this introduction, we embark on a journey to explore the art of crafting a blog app using the powerful trio of technologies: React Native, Node.js, and MongoDB.

React Native forms the backbone of our frontend, offering the ability to develop cross-platform mobile applications from a single codebase. This ensures a consistent and high-quality user experience for both iOS and Android users. At the core of our backend lies Node.js, renowned for its event-driven, non-blocking I/O model. It powers a responsive server that adeptly handles user authentication, content management, and much more. Our data finds a home in MongoDB. With its schema-less design, it perfectly aligns with the dynamic nature of blog content, allowing for easy adaptation to evolving requirements.

1.1 An Overview of the Project

A blog app is a versatile software application designed to simplify the process of creating, managing, and sharing written content in the form of blog posts. It serves as a platform for bloggers to express their thoughts and experiences while offering readers a convenient way to access and engage with this content. At the core of a blog app is user authentication, enabling users to create accounts, log in securely, and manage their profiles. Authentication is a crucial element, as it ensures that only authorized individuals have the ability to create, edit, or delete blog posts within the platform.

1.2 Mission of the Project

The mission of the blog app project is to provide a versatile and user-friendly platform that empowers bloggers and readers to connect and engage in the digital realm. Our project aims to facilitate the creation, management, and sharing of blog content, enhancing the experience for both authors and readers. We seek to foster a sense of community and knowledge sharing while prioritizing user security and data integrity. To provide bloggers with an intuitive and efficient tool for expressing their thoughts, experiences, and creativity, thereby enabling them to reach a wider audience and grow their online presence.

Problem:

- As the app's user base grows, it may face performance challenges, including slow loading times and unresponsiveness.
- The app may struggle to scale to meet the demands of a growing user base and increasing content.

Protecting user data and ensuring data privacy is crucial. Data breaches or vulnerabilities could pose a

- significant risk.
- Maintaining user engagement and ensuring users return to the app can be challenging.

Solution:

- Optimize the app's frontend and backend code, implement caching mechanisms, use a Content Delivery Network (CDN) for assets, and consider serverless computing to handle increased loads more efficiently.
- Implement load balancing and auto-scaling on the server-side, utilize a cloud-based database service that can handle scalability (e.g., MongoDB Atlas), and use a scalable file storage solution for media files.
- Employ strong encryption for data in transit and at rest, implement access controls and user authentication, regularly update and patch software components, and conduct security audits and penetration testing.
- Implement push notifications to notify users of new content or interactions, encourage user-generated content and interactions, and use data analytics to understand user behavior and preferences for content recommendations.

1.3 Background Study

- Study competitors and similar blog apps to understand their features, strengths, and weaknesses.
- Identify the target audience for your blog app. Consider demographics, interests, and user behaviors.
- Conduct surveys, interviews, or user research to understand what features potential users expect from a blog app.
- Prioritize features and functionalities based on user needs and market demand

1.3.1 A Study of the Existing System

1. Begin by identifying and documenting existing blog apps, websites, or platforms that offer similar features and cater to the same or a similar target audience.
2. Analyze the features and functionalities offered by these existing systems. Identify common and unique features, including user registration, content creation, content discovery, user engagement, and monetization strategies.
3. Evaluate the user experience of these existing systems. Consider the ease of navigation,

content readability, user interface design, and overall user satisfaction.

4. Investigate the monetization methods employed by existing systems. This may include analyzing their advertising models, premium subscriptions, affiliate marketing, or any other revenue-generating strategies.
5. Study how content is managed in these systems. This includes how authors create and edit posts, how content is categorized or tagged, and how users interact with and comment on posts.

CHAPTER-2

2. System Analysis

Blog app using React Native, Node.js, and MongoDB, involves a comprehensive examination of the project's requirements, functionalities, and architecture. It begins with meticulous requirements gathering, where both functional and non-functional aspects are identified and documented. Use cases and user stories are crafted to delineate how various user roles, including authors, readers, and administrators, will interact with the application, outlining their specific tasks and goals.

2.1 A Study of the Proposed System

A study of the proposed system involves a detailed examination of the features, functionalities, and architectural components that will be implemented in the blog app developed using React Native, Node.js, and MongoDB. Firstly, it entails the specification of core features that will define the app, encompassing user registration and authentication, content creation and management, content discovery, user engagement, and potential monetization strategies

2.2 User Requirement Specification

It involves understanding and documenting the specific needs and expectations of the users who will interact with the application.

First and foremost, it's essential to define the various user roles and their associated profiles. In this context, we typically identify authors who create and manage blog posts, readers who browse and engage with content, and administrators who oversee the platform. Each of these user roles has unique requirements and expectations that need to be clearly defined to create a tailored user experience.

For authors and readers, a streamlined and secure user registration and authentication process is imperative. This involves email verification, password reset mechanisms, and often the option for users to log in using their social media credentials. The specification should detail the workflows for registration and login, emphasizing both ease of use and robust security measures.

2.2.1 Major Modules

- **Authentication Module**

This module encompasses user registration, login, and profile management. It ensures secure authentication and user data management, allowing users to create accounts, sign in, and customize their profiles.

- **Content Module**

Content management is a core module that enables authors to create, edit, and publish blog posts. It includes features for formatting text, uploading media, and categorizing posts. Readers can view and interact with blog posts in this module.

- **User Engagement and Interaction Module**

This module facilitates user engagement and interaction. It includes features for commenting on posts, upvoting or downvoting comments, sharing content on social media, and following authors. Push notifications are often part of this module to keep users updated

- **Performance Optimization Module**

This module is focused on optimizing the app's performance and ensuring its responsiveness. It includes strategies for load balancing, caching, and other techniques to handle increased user loads efficiently.

2.2.2 Sub Modules Signup

- **Sub-Module**

This submodule allows users to create new accounts by providing necessary information such as username, email, and password. It involves user registration forms and validation

- **Login Sub-Module**

This submodule handles user authentication, ensuring that users can securely log in using their credentials. It includes user sessions and authentication tokens.

- **User Profile Management Sub-Module**

User profile management enables users to edit and update their profiles, including profile pictures, personal information, and preferences.

- **Blog Post Creation and Editing Sub-Module**

This submodule enables authors to create and edit blog posts, providing tools for text input, formatting, and content organization.

- **Rich Text Editor or Markdown Support Sub-Module**

It may include a rich text editor with formatting options or support for Markdown, which allows authors to structure their content with plain text.

2.3 Software Requirement Specification

The Software Requirement Specification (SRS) is a vital document that serves as the foundation for the development of a blog app using React Native, Node.js, and MongoDB. It outlines both the functional and non-functional requirements that the software must meet. The following paragraphs break down the key sections of the SRS.It begins with an introduction that provides a comprehensive overview of the document's purpose and scope. It sets the stage for the entire project, describing the software's intended functionality and what it aims to achieve. It serves as a reference point for all stakeholders involved in the development process. The SRS delves into the functional requirements of the software. It meticulously describes the features, use cases, and user interactions that the system must support. This includes user registration, content management, interaction features, and more. It outlines the expected behavior and outcomes of the software, guiding developers in implementing these functionalities. Non-functional requirements encompass the quality attributes of the system. This section of the SRS outlines how the software should perform in terms of scalability, security, performance, and usability.

2.4 System Specification

System specification, often referred to as system design, plays a pivotal role in the development of a blog app using React Native, Node.js, and MongoDB. It involves a detailed description of how the software will be structured, architected, and implemented. The system architecture defines the high-level structure of the blog app. In this section, the system specification outlines how the frontend and backend components interact. It may detail the use of React Native for building the cross-platform mobile app, Node.js for the backend server, and MongoDB as the database. The architecture may describe the separation of concerns, API design, and communication protocols between these components

2.4.1 React Native

In a MERN stack project to React Native involves transitioning from a web-based application to a mobile app. React Native allows you to build mobile apps for iOS and Android using JavaScript and React Native.

2.4.2 Node.js

In the context of the Blog app, Node.js serves as the JavaScript runtime environment responsible for constructing and operating the server-side functionalities. It enables real-time communication, data processing, and user interactions within the app. The choice of Node.jsversion is crucial, aligning with the project's specific dependencies and libraries

2.4.3 Express.js

In Express.js is a widely-used framework for Node.js that provides a streamlined way to buildAPIs and manage HTTP requests in web applications. It provides a set of features for buildingweb and mobile applications, including robust routing, middleware support, and easy integration with various template engines.

2.4.4 MongoDB

In the context of the Blog app, specifying the version of MongoDB is vital to maintain compatibility with the database driver and Node.js. MongoDB, a NoSQL database, is employed to store and manage user profiles, blog articles, and multimedia content in a document-oriented structure.

2.4.5 Hardware Configuration

The minimum hardware requirements for a MERN stack project will vary based on thescope and intricacy of the application.

- Processor: Intel Core i5 or above
- RAM: 8GB or more
- Storage: 500GB SSD or more

2.4.6 Software Configuration

- **Operating Systems**: Windows 10/11 for development.
- **Development Tools**:
  - **Expo** for building and testing mobile applications.
  - **Visual Studio Code** for code development.
- **Version Control**: Git for code management and collaboration.
- **Package Managers**: npm (Node Package Manager) for handling dependencies such as Express, Mongoose, and React Native libraries.
- **Database**: MongoDB installed locally or remotely via MongoDB Atlas

CHAPTER-3

3. System Design and Development

The System design is the phase where the architectural blueprint of the blog app takes shape. It involves the creation of detailed technical specifications, including database schemas, system components, and the user interface design. The frontend, developed with React Native, is meticulously designed to offer an engaging and intuitive user experience. This phase also involves the design of the backend, where Node.js serves as the foundation for building a robust and scalable server-side infrastructure. Database design for MongoDB is refined, ensuring efficient data storage, retrieval, and management. Authentication and authorization mechanisms are implemented to secure user access, and the integration of third-party services, such as social media APIs or analytics tools, is carefully planned. The design phase is a critical step in ensuring that all elements of the blog app align with the software requirements and function seamlessly together.

3.1 Fundamentals of Design Concepts

This approach places the user at the forefront of the design process. It involves understanding the needs, preferences, and behaviors of both authors and readers. User personas are created to represent the different user groups, aiding in the development of interfaces and features that cater to their specific requirements.

Consistency in design is another fundamental concept. This ensures that the user interface maintains a cohesive look and feel throughout the app. Consistency extends to elements like color schemes, typography, navigation patterns, and the placement of buttons and features. This uniformity enhances user familiarity and comfort, reducing the learning curve for new users.

Effective information architecture is vital for organizing content and navigation. The design should present content in a logical and easily navigable manner. The categorization of blog posts, tags, and search functionalities should align with user expectations and make content discovery straightforward.

Incorporating responsive design is essential to ensure the app adapts to various screen sizes and devices. Given the cross-platform nature of React Native, responsive design principles must be applied for both Android and iOS devices.

3.1.1 Abstraction

In the context of software design is a fundamental concept that involves simplifying complex systems or processes to their essential, high-level components. It allows developers to focus on the most critical aspects

of a system while hiding the intricate details. In the design of a blog app using React Native, Node.js, and MongoDB, abstraction might involve creating user-friendly interfaces that shield users from the underlying complexities of the technology stack.

3.1.2 Refinement

Refinement in the context of software design is the iterative process of continuously improving and enhancing the design of a system or application. It involves the careful review and optimization of various design elements, including user interfaces, architecture, and features, to ensure they meet evolving requirements and user expectations.

3.1.3 Modularity

Modularity is a design principle in software development that involves breaking down a complex system into smaller, self-contained, and independent modules or components. Each module serves a specific function or task, and they can be developed, tested, and maintained separately. Modularity enhances the flexibility, maintainability, and scalability of a software system.

3.2 Design Notations

3.2.1 System Structure Chart

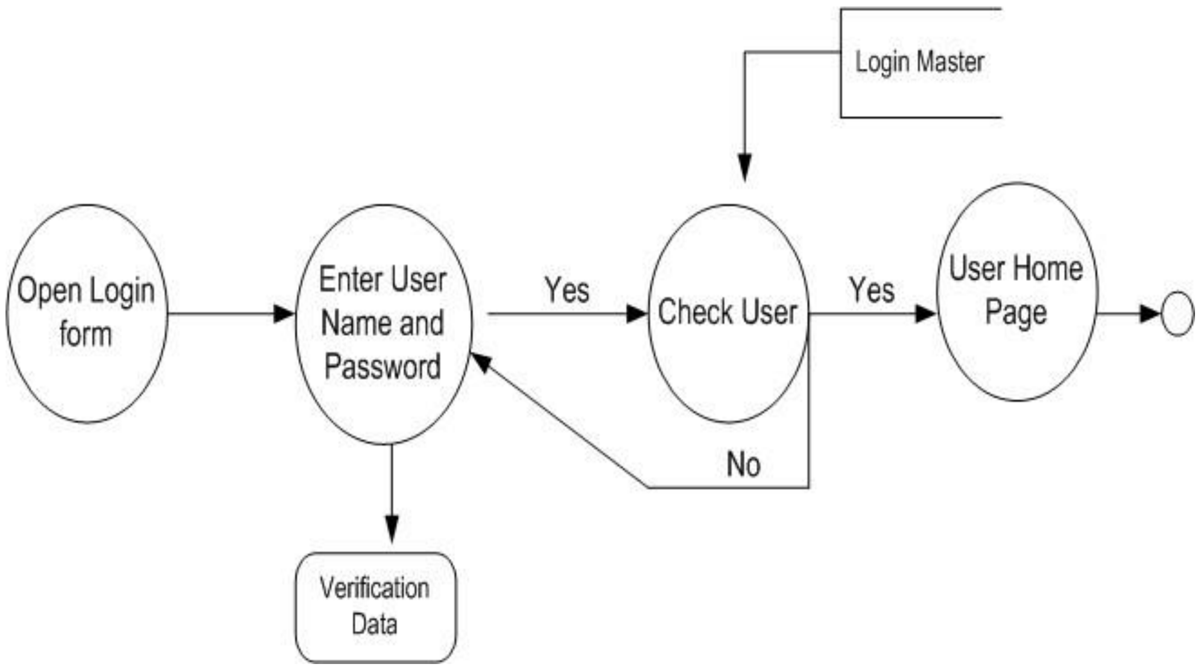


Figure 3.1 System Structure Chart

3.2.2 System Flow Diagram

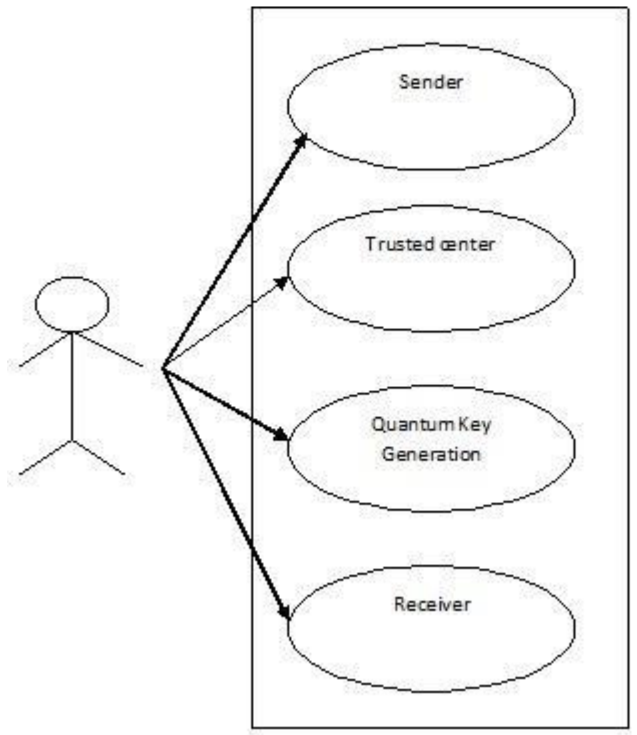


Figure 3.2 System Structure Chart

3.2.3 Data Flow Diagram / UML

A Data Flow Diagram (DFD) and Unified Modeling Language (UML) are two distinct modeling techniques used in software development to visually represent system processes, data flow, and relationships. A Data Flow Diagram (DFD) is a graphical representation of a system that illustrates the flow of data and processes within that system. DFDs are typically used to model information systems, software systems, and business processes. They consist of various elements, including processes, data stores, data flow, and external entities.

External Entities:

An external entity is a source or destination of a data flow. Only those entities which originate orreceive data are represented on a data flow diagram. The symbol used is a rectangular box.

Processes:

A process shows a transformation or manipulation of data flow within the system. The symbol used is an oval shape.

Data Flows:

The data flow shows the flow of information from a source to its destination. Dataflow is representedby a line, with arrowheads showing the direction of flow. Information always flows to or from a process and may be written, verbal or electronic. Each data flow may be referenced by the processes or data storesat its head and tail, or by a description of its contents.

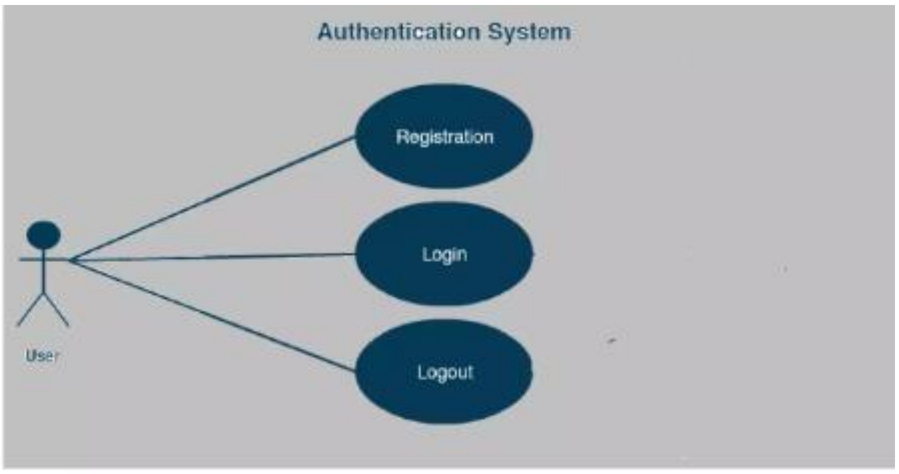


Figure 3.3 Data Flow Diagram

3.2.4 Software Engineering Model

A software process model is an abstraction of the software development process. The models specifythe stages and order of a process. So, think of this as a representation of the order of activities of theprocess and the sequence in which they are performed.

In the system, we can observe that the user interacts with the application through a graphical user interface. The inputs to the system are the Search and Filter criteria provided by the user and a newreview written by the user. Also, the output is in the form of Repeater and grid views which presentthe donars with list. The users can view complete specification, view Images.

3.3 Design Process

3.3.1 Software Architecture

- Software architecture is the set of structures needed to reason about a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.
- The architecture of a software system is a metaphor, analogous to the architecture of a building. It functions as the blueprints for the system and the development project, which project management can later use to extrapolate the tasks necessary to be executed by the teams and people involved.
- Software architecture is about making fundamental structural choices that are costly to change once implemented. Software architecture choices include specific structural options from

possibilities in the design of the software.

3.3.2 Control Hierarchy

- Control hierarchy, also called program structure, represents the organization of program components (modules) and implies a hierarchy of control. It does not represent procedural aspects of software such as sequence of processes, occurrence or order of decisions, or repetition of operations; nor is it necessarily applicable to all architectural styles.
- The control hierarchy also represents two subtly different characteristics of the software architecture: visibility and connectivity. Visibility indicates the set of program componentsthat may be invoked or used as data by a given component, even when this is accomplishedindirectly.
- A control system manages, commands, directs, or regulates the behavior of other devices or systems using control loops. It can range from a single home heating controller using a thermos state controlling a domestic boiler to large industrial control systems which are used for controlling processes or machines. The control systems are designed via control engineering process.

3.3.3 Structural Partitioning

- The program structure should be partitioned both vertically and horizontally. As describe in horizontal partitioning describe separate branches of the modular hierarchy of reach major program function. The Control modules, represented in a darker shade are used to coordinate communication among and execution of program functions. The easiest approach to horizontal partitioning describes 3 partitions - input, data transformation often called processing and output.
- The nature of modification in program architectures justifies the requirement for vertical partitioning. The change in a control module high in the architecture will have a higher possibility of propagating side effects to modules which are subordinate to it. The change toa worker module given its low level in the structure is less likely to cause the propagation ofside effects. In common changes to computer programs revolve around changes to input.

3.3.4 Data Structure

- Data structure is a storage that is used to store and organize data. It is a way of arranging dataon a computer so that it can be accessed and updated efficiently.
- Depending on your requirement and project, it is important to choose the right data structurefor your project. For example, if you want to store data sequentially in the memory, then you can go for the Array data structure.
- Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer—a bit string, representing a memory address,that can

be itself stored in memory and manipulated by the program. Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations, while the linked data structures are based on storing addresses of data items within the structure itself.

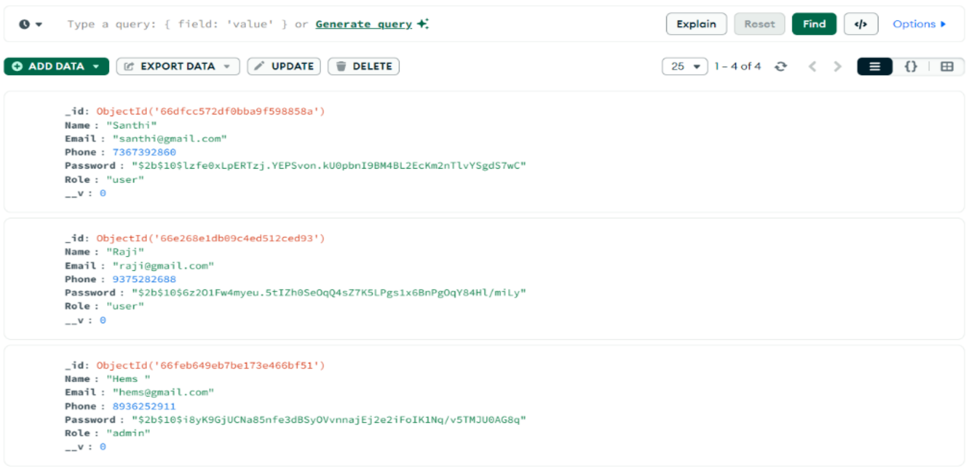
3.3.5 Software Procedure

The development process for a blog app using React Native, Node.js, and MongoDB. They encompass a series of well-defined steps and actions that developers follow to create, deploy, and maintain the software. The development phase entails a set of procedures to build the app's frontend and backend. This involves writing code, creating user interfaces, implementing features, and ensuring the app aligns with the software requirements. Developers follow best practices and coding standards, conduct code reviews, and collaborate on version control systems to maintain code quality. Implementing user authentication and authorization involves specific procedures, including user registration, login mechanisms, password reset procedures, and user role management. Developers create secure authentication flows, handle user sessions, and enforce access control rules.

Procedures related to database management include data modeling, schema design, and database interactions with MongoDB. Developers define data structures, establish indexes, and implement data retrieval and storage procedures. Maintenance procedures, such as data backups and migrations, are also essential

3.4 Database Design

Databases are the storehouses of data used in the software systems. The data is stored in collection inside the database. Several collections are created for the manipulation of the datafor the system. MongoDB is a NoSQL database, which means it doesn't use the traditional relational table-based structure. Instead, it stores data in flexible, JSON-like documents.

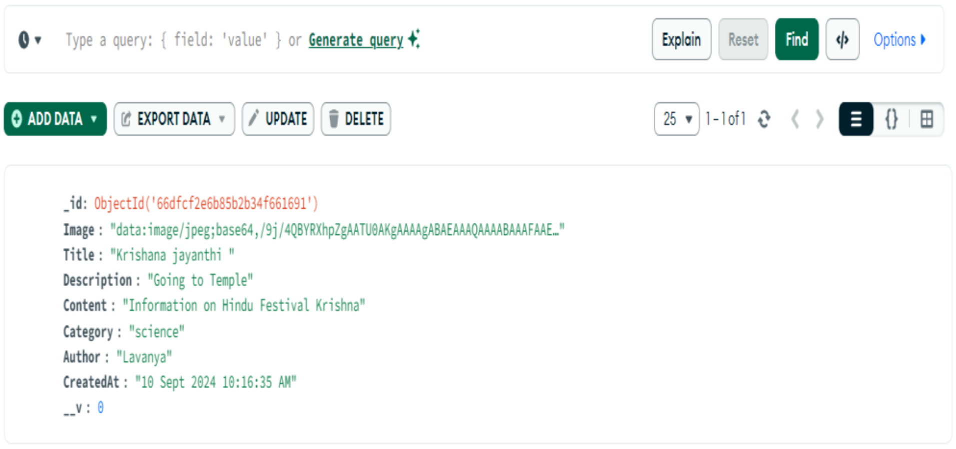


```
_id: ObjectId('66dfcc572df0bba9f598858a')
Name : "Santhi"
Email : "santhi@gmail.com"
Phone : 7367392860
Password : "$2b$10$lzfe0xLpERTzj.YEPSvon.kU0pbnI9BM4BL2EcKm2nTlvYSgdS7wC"
Role : "user"
__v : 0

_id: ObjectId('66e268e1db09c4ed512ced93')
Name : "Raji"
Email : "raji@gmail.com"
Phone : 9375282688
Password : "$2b$10$6z2O1Fw4myeu.5tIZh0SeOqQ4sZ7K5LPgs1x6BnPgOqY84Hl/miLy"
Role : "user"
__v : 0

_id: ObjectId('66feb649eb7be173e466bf51')
Name : "Hems "
Email : "hems@gmail.com"
Phone : 8936252911
Password : "$2b$10$i8yK9GjUCNa85nfe3dBSyOVvnnajEj2e2iFoIK1Nq/v5TMJU0AG8q"
Role : "admin"
__v : 0
```

Figure 3.5 user database



```
_id: ObjectId('66dfcf2e6b85b2b34f661691')
Image : "data:image/jpeg;base64,/9j/4QBYRXhpZgAATU0AKgAAAAgABAEAAAQAAAABAAAFAAE…"
Title : "Krishana jayanthi "
Description : "Going to Temple"
Content : "Information on Hindu Festival Krishna"
Category : "science"
Author : "Lavanya"
CreatedAt : "10 Sept 2024 10:16:35 AM"
__v : 0
```

Figure 3.6 Admin Database

.

3.5 Input Design

In input design is a critical aspect of user interface (UI) and user experience (UX) design in the development of a blog app using React Native, Node.js, and MongoDB. It focuses on creating intuitive and user-friendly methods for users to input information, interact with the app, and provide content. Creating and editing blog posts involves rich text input, multimedia uploads, and category selection. The design should incorporate a user-friendly editor with formatting options, content preview, and media upload functionality. User guidance and immediate feedback are crucial for error-free content creation.

Input design should prioritize the principles of simplicity, clarity, and user guidance. It should ensure that input fields are visually distinct, responsive, and provide immediate feedback on user actions, such as highlighting active fields, displaying character counts, or offering suggestions. Additionally, input fields should be designed to be accessible and responsive to various screen sizes, making them usable on both mobile and desktop platforms.

A well-designed input system enhances the user experience, reduces input errors, and fosters user engagement, making it an essential component in the development of a successful blog app.

3.6 Output Design

Output design is a crucial component of user interface (UI) and user experience (UX) design in the development of a blog app using React Native, Node.js, and MongoDB. It focuses on presenting information, feedback, and content to users in a visually appealing, understandable, and engaging manner. The design should ensure that blog posts, comments, and other content are presented in a clear and readable format. This includes defining typography, text layout, and image placement to make the content visually appealing and easy to consume.

Output design in a blog app aims to create a visually pleasing and user-centric presentation of content and information. It should prioritize clarity, readability, and an engaging visual aesthetic to enhance the user experience and encourage user interaction with the app's content and features.

3.7 Development Approach

The development approach for creating a blog app using React Native, Node.js, and MongoDB requires a systematic and well-structured process to ensure a successful outcome. An Agile development approach is well-suited for building a blog app. Agile emphasizes iterative development, collaboration, and flexibility. Development is divided into sprints or iterations, each focused on a specific set of features or user stories. This approach allows for frequent updates and feedback loops, making it easier to adapt to changing requirements and user needs. A cross-functional development team is composed of individuals with diverse skills, including front-end and back-end developers, UI/UX designers, quality assurance testers, and possibly DevOps engineers for deployment. This team structure ensures that all aspects of the app, from user interface design to server-side logic, are effectively addressed.

This development approach combines agile principles with a user-centric focus, modularity, and modern development practices to create a blog app that is not only feature-rich but also responsive to user feedback and adaptable to evolving requirements. It places the user experience at the forefront and emphasizes collaboration and flexibility to achieve a successful end result.

CHAPTER-4

4. Testing and Implementation

4.1 Testing

Software testing involves the process of evaluating and verifying that a software product or system meets the specified requirements. It includes the act of examining the artifacts and behavior of the software through validation and verification. Testing offers an independent assessment, allowing businesses to understand the risks involved in deploying the software. The primary goal of testing is to identify errors and ensure that the system operates as intended.

Testing is closely intertwined with the implementation process. As the system is implemented, tests ensure that it functions without errors, validating that the implementation was correctly executed. In practice, these two phases work in tandem to deliver a successful product.

Implementation, in this context, refers to executing a plan or process and ensuring that the system runs as expected. Each step of the implementation must be tested, a process known as implementation testing.

4.1.1 System Testing

System testing is a comprehensive software testing approach that evaluates the complete and fully integrated system to ensure it meets the specified requirements. This type of testing occurs after integration testing and before acceptance testing, confirming that the system is suitable for release to end-users.

Key characteristics of system testing include:

- **Scope**: It tests both functional and non-functional aspects of the software.
- **Objective**: To detect defects in integrated units and the overall system.
- **Process**: Takes input from passed integration tests and checks for irregularities across the system components.
- **Method**: System testing is usually a black-box testing method, where the tester does not need to know the internal workings of the application. It focuses on validating the system against the Software Requirement Specifications (SRS) and functional requirement specifications.

System testing ensures that the entire software system operates as designed. The goal is to ensure that the system performs tasks as required, aligning with the expectations of both developers and end-users. This stage of testing is typically performed by an independent testing team to maintain objectivity and impartiality in evaluating system quality.

4.1.2 White Box Testing

White box testing, also known as clear box or glass box testing, is a method where the internal structure of the software is known to the tester. Unlike black box testing, it focuses on validating internal processes and logic by examining the codebase.

- **Goal:** To test the internal workings of the software, including paths, conditions, loops, and logic.
- **Scope:** Requires the tester to have programming skills and knowledge of the internal code to test the functionality accurately.

4.1.3 Unit Testing

- **Definition**: Unit testing involves testing individual components or modules of the application to ensure they perform as expected.
- **Objectives**:
  - Isolate each part of the program and show that the individual parts are correct.
  - Facilitate easier debugging and code refactoring.
- **Approach**:
  - Write test cases for every function, method, or class to validate their correctness.
  - Utilize mocks and stubs for dependent components to focus on the unit being tested.

4.2 System Implementation

System implementation refers to executing a well-formulated plan for deploying a system or process. The success of implementation hinges on clear objectives, predefined actions, and thorough testing to ensure the system operates correctly. Each action in the implementation process undergoes testing, known as implementation testing, to confirm that the system performs as intended.

Key Aspects of Implementation:

1. **Testing Technological Specifications**: It is critical to verify the implementation of specific technological requirements to ensure they align with user and system expectations.
2. **Improving Interfaces**: Through testing, implementation can improve user interfaces to make them more understandable and user-friendly for developers and users alike.
3. **Action Confirmation**: Implementation testing ensures that identified actions can be successfully executed without issues.
4. **Identifying Issues**: Ambiguous or conflicting actions are easily identified during implementation testing, allowing for quick resolution.
5. **Quality Control**: Testing during implementation acts as a quality check, ensuring that the system's features are correctly implemented and operational.

During this phase, the test manager collaborates with the IT infrastructure team to ensure the availability of the appropriate test environment and its configuration. Implementation also involves the realization of system components, such as:

- **System Classes**: The blueprint for how the system behaves.
- **User Interface**: Ensuring users interact with the system intuitively and effectively.
- **Database Structures**: Ensuring the data storage and retrieval system operates as designed.

4.2.1 System Maintenance

System maintenance involves ensuring the system remains operational and efficient over time. Maintenance includes two primary activities:

1. **System Maintenance**: The process of restoring the system to its original functionality by fixing bugs, improving performance, and maintaining operational consistency.
2. **System Enhancement**: The addition or modification of features based on evolving user needs or updated specifications. Enhancement expands the system's capabilities, making it adaptable to new requirements.

Types of Maintenance:

- **Corrective Maintenance**: Fixing errors found in the system after it has been deployed.
- **Adaptive Maintenance**: Modifying the system to cope with changes in the environment (e.g., new hardware or software platforms).

- **Perfective Maintenance**: Improving system performance or maintainability, even when no defects are present.
- **Preventive Maintenance**: Identifying and correcting potential issues before they occur, ensuring the system remains stable over time.

Importance of System Maintenance:

- Keeps the system functioning at an optimal level by addressing potential faults before they impact operations.
- Ensures system security and compliance with current technological standards.
- Enhances the system to meet new user requirements, extending its lifecycle and value.
- **Documentation**: Maintain comprehensive documentation that includes installation procedures, user guides, and troubleshooting instructions to assist users and future developers.

For businesses, investing in proper maintenance results in long-term cost savings. Regular updates and fixes prevent the need for expensive overhauls or replacements, keeping the system efficient and relevant.

CHAPTER-5

5. Conclusion

In summary, creating a log app using React Native is an exciting and fulfilling endeavor that opens the doors to a world of possibilities. One of the key advantages of using React Native is its cross-platform compatibility, allowing you to develop for both iOS and Expo simultaneously. This can significantly save development time and resources, making it a practical choice for reaching a broader audience.

User-friendliness is paramount in a blog app. React Native's component-based approach and extensive library of UI components make it a powerful tool for crafting an intuitive and visually appealing user interface. This enables you to design an engaging and user-friendly environment that encourages users to explore and contribute to the platform.

Performance is a critical aspect of any mobile application, and a blog app is no exception. While React Native offers good performance out of the box, it's essential to optimize your app to ensure swift navigation, quick loading times, and responsive interactions. This requires regular profiling and performance tuning to provide a seamless user experience.

API integration is fundamental for a blog app. Incorporating APIs for user authentication, content management, and social sharing is a must. Users should be able to create, edit, and publish their blog posts effortlessly. Consider implementing features like text formatting, image uploading, and categorization to enhance the content creation process.

5.1 Directions for future enhancements references:

The following things can be done in future.

The integration of self-service portals stands as a promising feature. By allowing users to manage their profiles, update personal details, and access essential chat-related documents, the application empowers users, granting them greater control over their experience. This not only enhances user satisfaction but also reduces the administrative burden on the system.

A forward-thinking addition would be the incorporation of real-time help alerts. Users could opt-in for notifications that trigger when blog fall below specified thresholds. This feature not only provides users with valuable information but also enrichestheir blog experience, enabling them to make well-informed decisions.

The current system, primarily focused on the blog, could be extended to include auser-friendly checkout process. Allowing users to store multiple blog and addresses and facilitating selection through intuitive drag-and-drop functionality can significantly improve the efficiency of the checkout process. This streamlined approach ensures a seamless experience, enhancing user satisfaction and encouraging help usage.

CHAPTER-6

BIBLIOGRAPHY:

- **Official React Native Documentation**: https://reactnative.dev/
- **Node.js Documentation**: https://nodejs.org/en
- **All about npm** http://www.npmjs.com
- **MongoDB Documentation**: https://www.mongodb.com/
- **TMDB API Documentation**: https://www.themoviedb.org/
- **Wikipedia for various diagrams & testing methods** http://www.wikipedia.org/

BOOK REFERENCE:

1."Learning React: A Hands-On Guide to Building Web Applications Using React and Redux", Author: Kirupa Chinnathambi, Publisher: Addison-Wesley Professional, Year: 2020, Edition: 2nd Edition

2."Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node", Author: Vasan Subramanian, Publisher: Apress, Year: 2019, Edition: 2nd Edition

3."Full-Stack React Projects: Learn MERN Stack Development by Building Modern Web Apps Using MongoDB, Express, React, and Node.js", Author: Shama Hoque
, Publisher: Packt Publishing, Year: 2020, Edition: 2nd Edition

4."The Road to React: Your Journey to Master React.js in JavaScript",Author: Robin Wieruch,Publisher: Independently Published, Year: 2020, Edition: 4th Edition

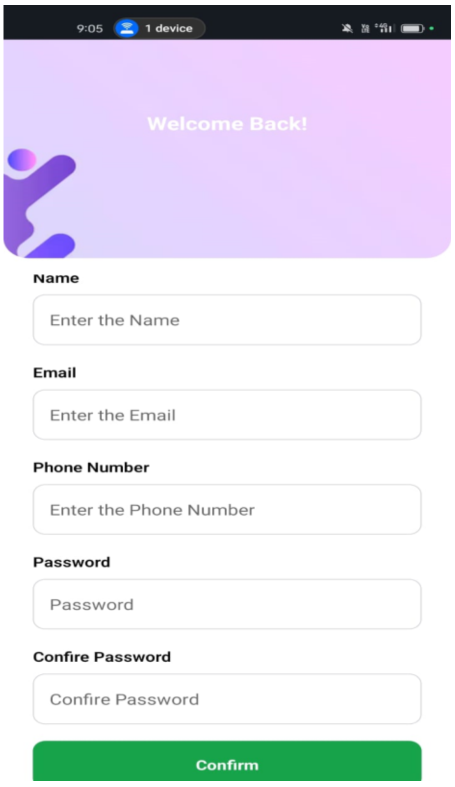
5."MongoDB: The Definitive Guide: Powerful and Scalable Data Storage", Author: Shannon Bradshaw, Eoin Brazil, Kristina Chodorow, Publisher: O'Reilly Media, Year: 2019, Edition:

ANNEXURE

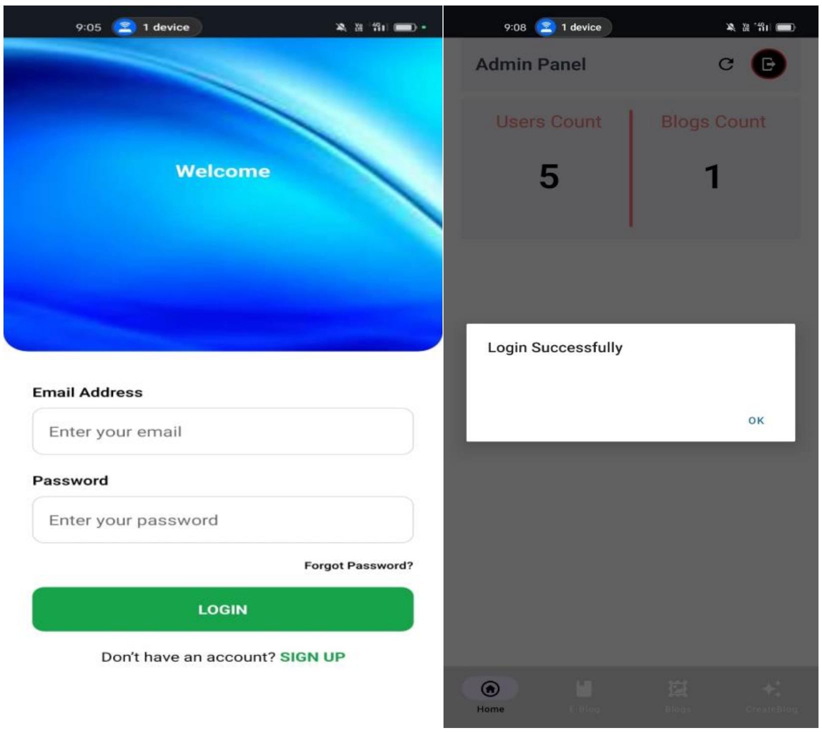
ANNEXURE –A-OUTPUT DESIGN

ADMIN_PANEL: -

REGISTER

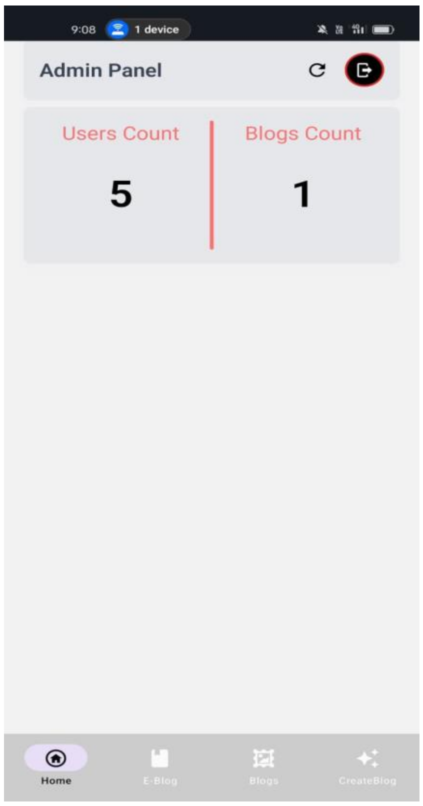


LOGIN

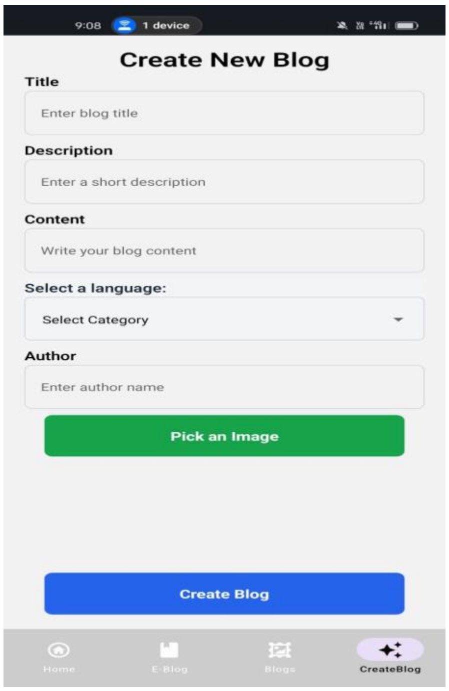


Login Successfully

DASHBOARD



CREATE PAGE



9:08 1 device

Create New Blog

Title

Enter blog title

Description

Enter a short description

Content

Write your blog content

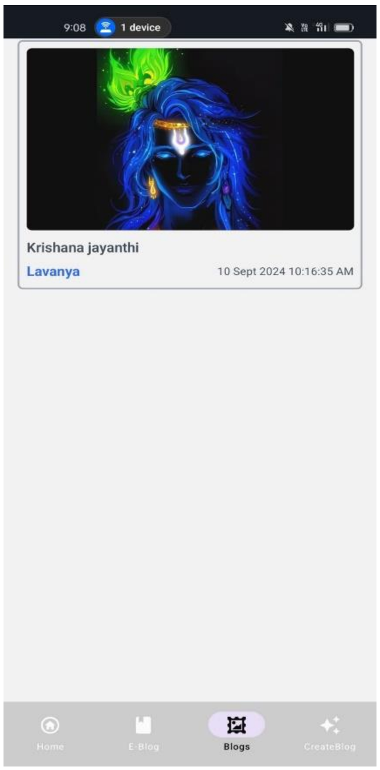
Select a language:

Select Category

Author

Enter author name

UPLOAD BLOG:



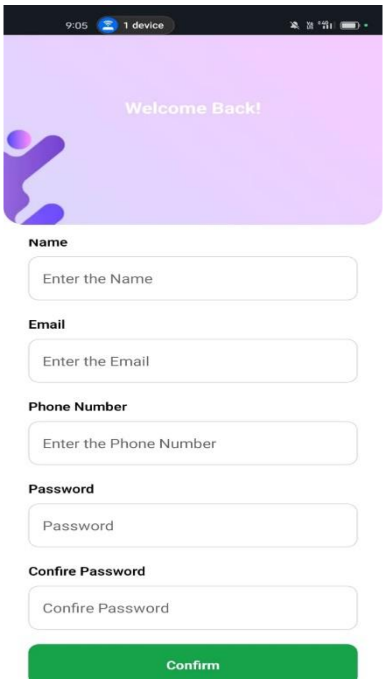
9:08 1 device

Krishana jayanthi

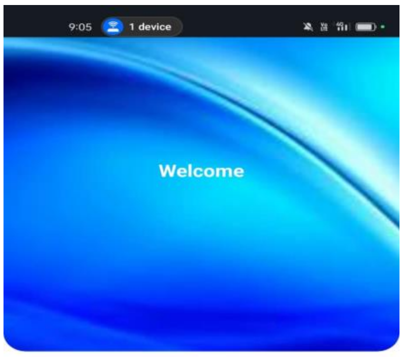
Lavanya 10 Sept 2024 10:16:35 AM

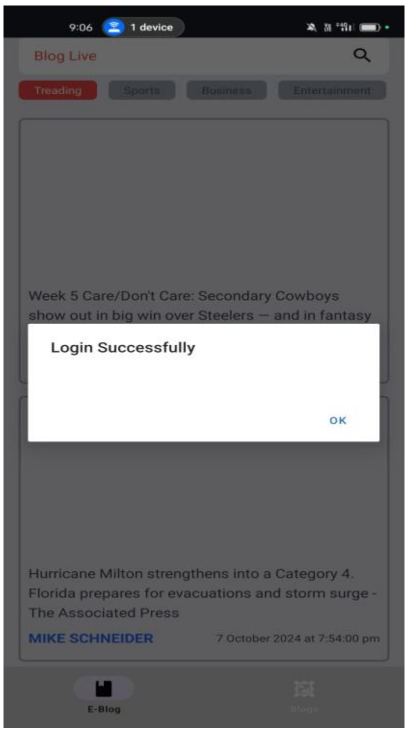
USER _- REACT NATIVE APP: -

REGISTER



LOGIN:





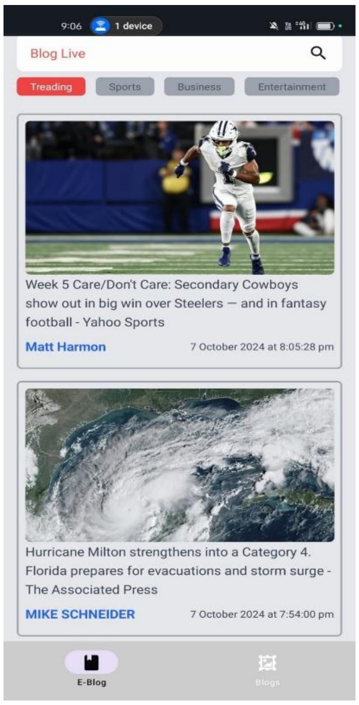
Week 5 Care/Don't Care: Secondary Cowboys show out in big win over Steelers — and in fantasy

Login Successfully

Hurricane Milton strengthens into a Category 4. Florida prepares for evacuations and storm surge - The Associated Press

MIKE SCHNEIDER 7 October 2024 at 7:54:00 pm

TRENDING BLOG



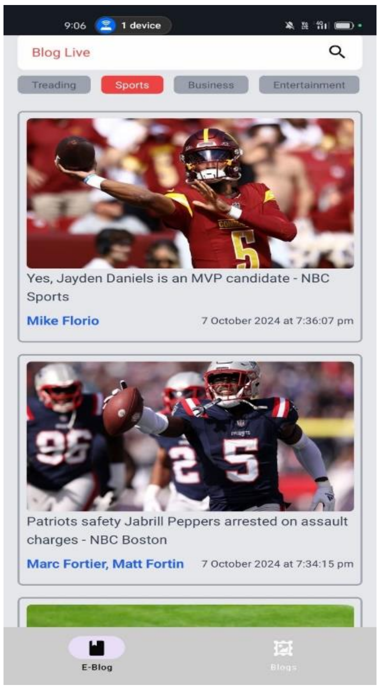
Week 5 Care/Don't Care: Secondary Cowboys show out in big win over Steelers — and in fantasy football - Yahoo Sports

Matt Harmon 7 October 2024 at 8:05:28 pm

Hurricane Milton strengthens into a Category 4. Florida prepares for evacuations and storm surge - The Associated Press

MIKE SCHNEIDER 7 October 2024 at 7:54:00 pm

SPORTS BLOG



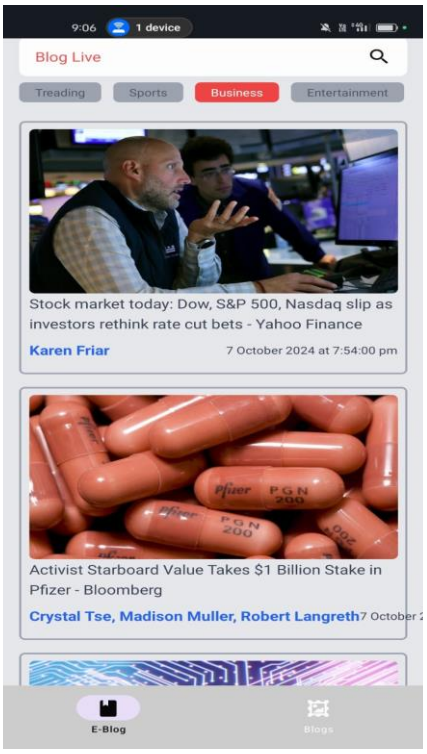
Yes, Jayden Daniels is an MVP candidate - NBC Sports

Mike Florio 7 October 2024 at 7:36:07 pm

Patriots safety Jabrill Peppers arrested on assault charges - NBC Boston

Marc Fortier, Matt Fortin 7 October 2024 at 7:34:15 pm

BUSINESS BLOG



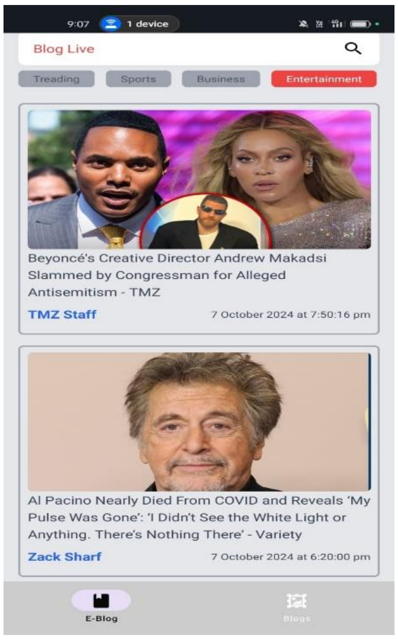
Stock market today: Dow, S&P 500, Nasdaq slip as investors rethink rate cut bets - Yahoo Finance

Karen Friar 7 October 2024 at 7:54:00 pm

Activist Starboard Value Takes $1 Billion Stake in Pfizer - Bloomberg

Crystal Tse, Madison Muller, Robert Langreth 7 October

ENTERTAINMENT BLOG



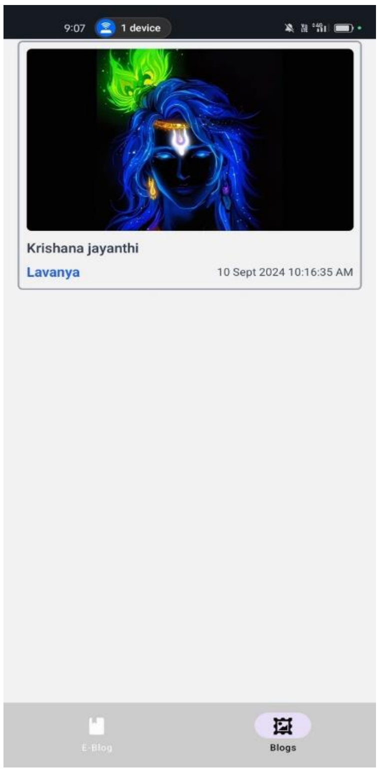
Beyoncé's Creative Director Andrew Makadsi Slammed by Congressman for Alleged Antisemitism - TMZ

TMZ Staff 7 October 2024 at 7:50:16 pm

Al Pacino Nearly Died From COVID and Reveals 'My Pulse Was Gone': 'I Didn't See the White Light or Anything. There's Nothing There' - Variety

Zack Sharf 7 October 2024 at 6:20:00 pm

POST BLOG



Krishana jayanthi

Lavanya

10 Sept 2024 10:16:35 AM

ANNEXURE –B-SOURCE CODE:

REACT NATIVE APP – JAIFLIX

APP.JS

```
import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import Splash from "./src/screens/Splash";
import Login from "./src/screens/Login";
import Register from "./src/screens/Register";
import Home from "./src/screens/Home";
import Admin from "./src/screens/Admin";
import AdminPanel from "./src/screens/AdminPanel";
import NewsViewer from "./src/Components/NewsViewer";
import Search from "./src/Components/Search";
import UserPanel from "./src/screens/UserPanel";
import ViewCard from "./src/Components/ViewCard";
import DetailsScreen from "./src/screens/DetailsScreen";
export default function App() {
const Stack = createNativeStackNavigator();
return (
<NavigationContainer>
<Stack.Navigator screenOptions={{ headerShown: false }}>
<Stack.Screen name="Splash" component={Splash} />
<Stack.Screen name="Login" component={Login} />
<Stack.Screen name="Register" component={Register} />
<Stack.Screen name="Home" component={Home} />
<Stack.Screen name="Admin" component={Admin} />
<Stack.Screen name="Adminpanel" component={AdminPanel} />
<Stack.Screen name="Userpanel" component={UserPanel} />
<Stack.Screen name="NewsViewer" component={NewsViewer} />
<Stack.Screen name="Search" component={Search} />
<Stack.Screen name="Viewcard" component={ViewCard} />
<Stack.Screen name="Detailscreen" component={DetailsScreen} />
</Stack.Navigator>
</NavigationContainer>
);
}
```

LOGINSCREEN.JS

```
import React, { useState } from "react";
import {
View,
Text,
TextInput,
TouchableOpacity,
ImageBackground,
Alert,
} from "react-native";
import axios from "axios";
import secOneBg from "../../assets/sec1-bg.png";
import login from"../../assets/login image.jpeg";
import { jwtDecode } from "jwt-decode";
export default function Login({ navigation }) {
const [fdata, setFdata] = useState({
Email: "",
Password: "",
});
```

```
const [userData, setUserData] = useState([]);
const [decodeData, setDecodeData] = useState([]);
const HandleLogin = async () => {
if (!fdata.Email || !fdata.Password) {
Alert.alert("All fields are required");
return;
}
try {
let res = await axios.post(
`http://192.168.135.44:8080/demo/user/login`,
{
...fdata,
}
);
setUserData(res.data);
Alert.alert(userData.message);
let decoded = jwtDecode(userData.token);
setDecodeData(decoded);
if (decodeData.role === "admin") {
navigation.navigate("Adminpanel");
} else {
navigation.navigate("Userpanel");
}
} catch (error) {
Alert.alert("Login Failed", error.response?.data || "Network Error");
}
};
return (
<View className="flex-1 overflow-hidden rounded-b-3xl">
<ImageBackground
source={login}
className="flex-1 justify-center items-center"
>
<View className="absolute inset-0 bg-black/40" />
<Text className="text-white text-2xl font-bold mb-5">
Welcome
</Text>
</ImageBackground>
</View>
<View className="flex-1 bg-white px-7 mt-10 rounded-t-3xl">
<Text className="text-black text-lg font-semibold mb-2">
Email Address
</Text>
<TextInput
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Enter your email"
onChangeText={(text) => setFdata({ ...fdata, Email: text })}
/>
<Text className="text-black text-lg font-semibold mb-2">Password</Text>
<TextInput
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Enter your password"
secureTextEntry
onChangeText={(text) => setFdata({ ...fdata, Password: text })}
/>
<View className="flex-row justify-end items-center mb-5">
<Text className="font-semibold text-black">Forgot Password?</Text>
```

```
</View>
<TouchableOpacity
className="bg-green-600 py-4 rounded-xl items-center mb-5"
onPress= {() => {
HandleLogin ();
}}
>
<Text className="text-white text-lg font-bold">LOGIN</Text>
</TouchableOpacity>
<Text className="text-center text-lg">
Don’t have an account?{" "}
<Text
className="text-green-600 font-bold"
onPress={() => navigation.navigate("Register")}
>
SIGN UP
</Text>
</Text>
</View>
</View>
);
}
```

REGISTERSCREEN.JS

```
import {
View,
Text,
TextInput,
TouchableOpacity,
ImageBackground,
ScrollView,
Alert,
} from "react-native";
import React, { useState } from "react";
import secOneBg from "../../assets/sec1-bg.png";
import axios from "axios";
export default function Register({ navigation }) {
const [fdata, setFdata] = useState({
Name: "",
Email: "",
Phone: "",
Password: "",
ConPassword: "",
});
const HandleRegister = async () => {
if (
!fdata.Name ||
!fdata.Email ||
!fdata.Phone ||
!fdata.Password ||
!fdata.ConPassword
) {
```

```
Alert.alert("All fields are required");
return;
}
if (fdata.Password !== fdata.ConPassword) {
Alert.alert("Password and confirm password must be the same");
return;
}
try {
let res = await axios.post(`http://192.168.135.44:8080/demo/user`, {
...fdata,
});
Alert.alert(res.data);
setFdata({
Name: "",
Email: "",
Phone: "",
Password: "",
ConPassword: "",
});
navigation.navigate("Login");
} catch (error) {
if (error.response.status >= 400 && 499 >= error.response.status) {
Alert.alert(error.response.data);
} else {
Alert.alert("Something Wrong / Network Error");
}
}
};
return (
<View className="flex-1 bg-white">
<View className="flex-[0.5] overflow-hidden rounded-b-3xl">
<ImageBackground
source={secOneBg}
className="flex-1 justify-center items-center"
>
<View className="absolute inset-0 bg-black/40" />
<Text className="text-white text-2xl font-bold mb-5">
Welcome Back!
</Text>
</ImageBackground>
</View>
<ScrollView className="flex-1 bg-white px-7 mt-5 rounded-t-3xl">
<Text className="text-black text-lg font-semibold mb-2">Name</Text>
<TextInput
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Enter the Name"
onChangeText={(text) => setFdata({ ...fdata, Name: text })}
/>
<Text className="text-black text-lg font-semibold mb-2">Email</Text>
<TextInput
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Enter the Email"
onChangeText={(text) => setFdata({ ...fdata, Email: text })}
/>
<Text className="text-black text-lg font-semibold mb-2">
Phone Number
</Text>
<TextInput
```

```
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Enter the Phone Number"
onChangeText={(text) => setFdata({ ...fdata, Phone: text })}
/>
<Text className="text-black text-lg font-semibold mb-2">Password</Text>
<TextInput
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Password"
onChangeText={(text) => setFdata({ ...fdata, Password: text })}
secureTextEntry
/>
<Text className="text-black text-lg font-semibold mb-2">
Confire Password
</Text>
<TextInput
className="border border-gray-300 rounded-xl p-4 text-xl mb-5"
placeholder="Confire Password"
onChangeText={(text) => setFdata({ ...fdata, ConPassword: text })}
secureTextEntry
/>
<TouchableOpacity
className="bg-green-600 py-4 rounded-xl items-center mb-5"
onPress={() => {
HandleRegister();
}}
>
<Text className="text-white text-lg font-bold">Confirm</Text>
</TouchableOpacity>
```

```
<Text className="text-center text-lg">
Already have an account?{" "}
<Text
className="text-green-600 font-bold"
onPress={() => navigation.navigate("Login")}
>
SIGN IN
</Text>
</Text>
</ScrollView>
<View></View>
</View>
);
}
```

ADMIN PANEL SOURCE CODE: -

```
import React from "react";
import { createMaterialBottomTabNavigator } from "@react-
navigation/material-bottom-tabs";
import MaterialCommunityIcons from "react-native-vector-
icons/MaterialCommunityIcons";
import Admin from "./Admin";
import BlogView from "./BlogView";
import CreateBlog from "./CreateBlog";
import Home from "./Home";
const Tab = createMaterialBottomTabNavigator();
const TabNav = () => {
return (
<Tab.Navigator
```

```
activeColor="#080808"
inactiveColor="#fcfcfc"
barStyle={{ backgroundColor: "#cccccc", paddingBottom: -5 }}
>
<Tab.Screen
name="Admin"
component={Admin}
options={{
tabBarLabel: "Home",
tabBarIcon: ({ color }) => (
<MaterialCommunityIcons
name="home-circle-outline"
color={color}
size={25}
/>
),
}}
/>
<Tab.Screen
name="e-blog"
component={Home}
options={{
tabBarLabel: "E-Blog",
tabBarIcon: ({ color }) => (
<MaterialCommunityIcons
name="book"
color={color}
size={25}
/>
),
}}
/>
<Tab.Screen
name="Blogview"
component={BlogView}
options={{
tabBarLabel: "Blogs",
tabBarIcon: ({ color }) => (
<MaterialCommunityIcons
name="postage-stamp"
color={color}
size={25}
/>
),
}}
/>
<Tab.Screen
name="Createblog"
component={CreateBlog}
options={{
tabBarLabel: "CreateBlog",
tabBarIcon: ({ color }) => (
<MaterialCommunityIcons name="creation" color={color} size={25} />
),
}}
/>
</Tab.Navigator>
);
```

```
};
export default function AdminPanel() {
return <TabNav />;
}
```

DASHBOARD.JS

```
import { View, Text, Image } from "react-native";
import React from "react";
export default function DetailsScreen({ route }) {
const { data } = route.params;
if (!data) {
return (
<View className="flex-1 justify-center items-center">
<Text>No data available</Text>
</View>
);
}
return (
<View className="flex-1 mt-10 px-4">
<View className="flex-row justify-between items-center mt-1">
<Text className="text-2xl text-gray-500">{data.Title}</Text>
<Text className="text-2xl text-gray-500">
Category : {data.Category}
</Text>
</View>
<Image
source={{
uri:
data.Image ||
"https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcTULSPiQKGEcCtCxrkr4t9Ub8U-
Jwzv3kXu2RMOzQoihg&s",
}}
className="h-[40vh] w-full rounded-md"
resizeMode="cover" // Use cover for better image fitting
/>
<View className="flex-row justify-between items-center mt-1">
<Text className="text-lg font-bold text-blue-600">{data.Author}</Text>
<Text className="text-md text-gray-700">{data.CreatedAt}</Text>
</View>
<Text className="text-lg underline text-gray-700 my-2">
{data.Description}
</Text>
<Text className="text-2xl text-gray-700 text-justify my-
2">{data.Content}</Text>
</View>
);
}
```