

# **TASK MANAGER WEB APPLICATION USING MICROSERVICES**

Project report submitted in partial fulfilment of the  
requirement for the degree of Bachelor of Technology  
in

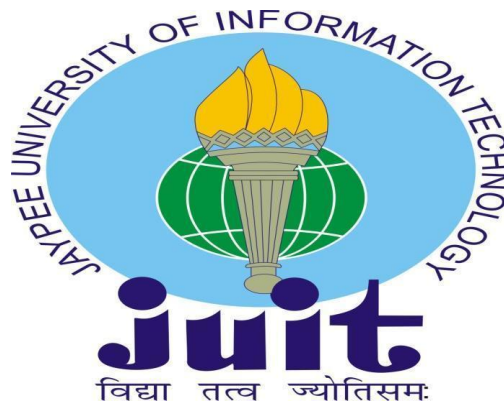
**Computer Science and Engineering**

By

**NARENDRA BAHADUR VERMA (191337)**

Under the supervision of  
Mr. Nishant Sharma

To



Department of Computer Science & Engineering and  
Information Technology

**Jaypee University of Information Technology**

**Waknaghat, Solan-173234, Himachal Pradesh**

## Candidate's Declaration

I hereby declare that the work presented in this report entitled “**A Task Manager Web Application Using Microservices.**” in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of **Mr. Nishant Sharma** (Assistant Professor Grade-II - CSE Dept.).

I also authenticate that I have carried out the above mentioned project work under the proficiency stream **Cloud Computing.**

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Narendra Bahadur Verma 191337

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

(Supervisor Signature)

Supervisor Name: Mr. Nishant Sharma

Designation: Assistant Professor Grade II

Department name: Computer Science

Dated:

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT**  
**PLAGIARISM VERIFICATION REPORT**

Date: .....

Type of Document (Tick):  PhD Thesis  M.Tech Dissertation/ Report  B.Tech Project Report  Paper

Name: \_\_\_\_\_ Department: \_\_\_\_\_ Enrolment No \_\_\_\_\_

Contact No. \_\_\_\_\_ E-mail. \_\_\_\_\_

Name of the Supervisor: \_\_\_\_\_

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): \_\_\_\_\_

**UNDERTAKING**

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

**Complete Thesis/Report Pages Detail:**

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

**FOR DEPARTMENT USE**

We have checked the thesis/report as per norms and found **Similarity Index** at .....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

**FOR LRC USE**

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none"> <li>• All Preliminary Pages</li> <li>• Bibliography/Images/Quotes</li> <li>• 14 Words String</li> </ul>		Word Counts	
<b>Report Generated on</b>			Character Counts	
		<b>Submission ID</b>	Total Pages Scanned	
			File Size	

Checked by  
Name & Signature

Librarian

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at [plagcheck.juit@gmail.com](mailto:plagcheck.juit@gmail.com)**

## ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing that made it possible for us to complete the project work successfully.

I am really grateful and wish my profound indebtedness to supervisor **Mr. Nishant Sharma, Assistant Professor**, Department of CSE, Jaypee University of Information Technology, Waknaghat. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Mr. Nishant Sharma**, Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straight forward or in a roundabout way in making this project a win. In this unique situation, I also want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Narendra Bahadur Verma 191337

# TABLE OF CONTENT

Candidate's Declaration.....	i
Plagiarism Certificate.....	ii
Acknowledgement.....	iii
Abstract.....	v
1. INTRODUCTION.....	1-7
1.1 Problem Statement.....	2
1.2 Objectives.....	3
1.3 Methodology.....	3-4
1.4 Technology Used.....	5-7
2. LITERATURE SURVEY.....	8-13
3. SYSTEM DEVELOPMENT.....	14-42
4. EXPERIMENT & RESULT ANALYSIS.....	43-51
5. CONCLUSION.....	52-55
References.....	56-57

## **ABSTRACT**

With improved state management and asynchronous operations, the Task Manager web application using the MERN (MongoDB, Express, React, and Node.js) stack with microservices, Passport local strategy, Redux, and Redux-Thunk is a strong and effective tool created to assist individuals and teams in managing their tasks and projects. The programme may be divided into smaller, independent services thanks to the microservices architecture, which makes it simpler to create, deploy, and maintain. Users can safely access their tasks and data thanks to the authentication and authorisation provided by the Passport local approach.

Performance and scalability are increased by using Redux and Redux-Thunk, respectively, for state management and asynchronous operations. Task creation, assignment, and tracking, user authentication, task prioritisation, notifications, and reminders are just a few of the capabilities offered by the application.

React and Redux were used to build the frontend of the programme, which offers a responsive and interactive user interface. Redux-Thunk middleware enables seamless management of asynchronous activities, while the Redux state management library enables efficient handling of complicated application state. The Passport local strategy guarantees secure user authentication and authorisation, giving users a convenient and safe experience. Users' personal information is kept secure while they effortlessly create accounts and log in to access their chores and data.

A robust and effective solution that can assist individuals and teams in managing their tasks and projects with improved state management and asynchronous operations is the Task Manager web application, which uses the MERN stack with microservices, Passport local strategy, Redux, and Redux-Thunk. The application's solid and durable basis is provided by the microservices architecture and Passport local strategy, while better performance and scalability are made possible by the Redux and Redux-Thunk framework.

## CHAPTER 1

### INTRODUCTION

The Task Manager web application was created to help the teams and individuals in managing their tasks and projects with better state and time management and asynchronous operations. It is a complete, good and feature-rich solution. Microservices, Passport, Passport local strategy, Redux, Redux-Thunk, and at last the MERN stack (MongoDB, Express, React, and Node.js) all of these technologies are used in this project. The programme is divided into smaller, independent services by using the microservices concept, which makes it simpler to create, deploy, and maintain. The Passport local strategy, which enables secure user authentication and authorisation, gives users a convenient and safe experience.

The Redux state management library and Redux-Thunk middleware are used to successfully handle complicated application state and successfully manage asynchronous operations. Basically redux is used for better state management than normal react. Also Redux and React were used to create the frontend, which offers a responsive and interesting user experience. The application's features include task creation, assignment, and tracking, user authentication, task prioritisation, notifications, and reminders, to name a few. Users may easily create accounts and log in to access their tasks and data, and their personal information is kept private at all times.

The Task Manager web application is a reliable and efficient solution that may help individuals and teams manage their tasks and projects with better state management, time management and asynchronous operations. It uses the MERN stack with microservices, Passport and the Passport local approach, Redux for state management, and Redux-Thunk for communicating with the server. Greater performance and scalability are provided by the Redux and Redux-Thunk frameworks, while the microservices architecture and Passport local strategy give

The application has a stable, secure and long-lasting foundation. It also provides easy scalability and high security. The program's high degree of versatility allows for quick customization to match the particular requirements of particular people or groups.

### **1.1 Problem Statement:**

The task management issue that the MERN (MongoDB, Express, React, and Node.js) stack-based Task Manager web application attempts to address is the ineffective and disorganised management of tasks and projects among individuals and teams. Spreadsheets, emails, and paper-based systems are examples of traditional task management techniques that can be time-consuming, error-prone, and challenging to collaborate on.

Expanding and modifying traditional task management techniques to suit the specific requirements of multiple users or teams may at times be challenging. As a result, the project's overall performance may suffer from a lack of productivity and efficiency.

Features like job prioritisation, notifications, reminders, and centralised data storage are typically absent from these techniques. Tasks could be neglected as a result, crucial deadlines might be missed, and teamwork might suffer.

The Task Manager web application uses the MERN stack in conjunction with microservices, Passport and Passport local strategy, Redux, and Redux-Thunk to produce a comprehensive and effective tool for managing tasks and projects for both people and teams. The Task Manager online tool takes care of this issue. The programme can be efficiently developed, deployed, and maintained thanks to the microservices design. Secure user authentication and authorisation are provided by the Passport local method, and state management and asynchronous actions are efficiently supported by the Redux and Redux-Thunk packages.

Our project provides a scalable, and adaptable task and project management tool that can improve teamwork and productivity on both the individual and team levels.



## **1.2 Objective:**

The Task Manager web application uses improved state management and asynchronous activities to provide individuals and teams with a comprehensive and useful solution for handling their tasks and projects. Microservices, the Passport local method, Redux, and Redux-Thunk are all used along with MERN (MongoDB, Express, React, and Node.js).

The microservices design enables the programme to be split into smaller, independent services, which makes it easier to build, deploy, and administer. The secure user authentication and authorisation provided by the Passport local strategy enables a seamless and secure user experience.

The application's features include task creation, assignment, and tracking, user authentication, task prioritisation, notifications, and reminders, to name a few. Delivering a practical tool that is simple to adapt to the particular needs of individual users or teams is the aim.

Redux-Thunk middleware and the Redux state management library are used to successfully manage asynchronous operations and handle complex application state. The frontend, which provides a responsive and engaging user experience, was made using Redux and React.

The main goal of the Task Manager web application is to provide a strong and practical task and project management tool that might boost team and individual productivity and collaboration. Microservices, the Passport local approach, Redux, and Redux-Thunk are combined with the MERN stack to accomplish this. The programme excels for a wide range of customers and use cases because to its scalability, robustness, and adaptability.

## **1.3 Methodology:**

The MERN (MongoDB, Express, React, and Node.js) stack with microservices, Passport local approach, Redux, and Redux-Thunk were used in the creation of the Task Manager web application.[11] In order to consistently provide usable

solutions, the Agile methodology significantly encourages incremental and iterative development. This approach promotes ongoing input and cooperation between programmers, stakeholders, and end users, allowing the programme to change and improve over time to satisfy changing user needs.

The Task Manager web application, which utilises the MERN stack, is created in the manner described below:

- The requirements collecting and analysis process includes identifying the features and functionalities that the application should have as well as determining the project's scope. Understanding user and stakeholder needs and desires is necessary for this.
- Design: This encompasses developing the application's architecture, which includes the microservices architecture, database schema, and user interface and user experience. The design should be built upon the specifications listed in step 1 of the process.
- The MERN stack, microservices, the Passport local approach, Redux, and Redux-Thunk will be used to implement the architecture. The foundation for the execution should be the design and specifications found in phases 1 and 2.[11]
- Testing involves putting an application through its paces to ensure that it complies with requirements and operates as intended. At different testing levels, unit testing, integration testing, and acceptability testing should all be conducted.[6]
- Deployment and maintenance: This involves running the application in a real environment, monitoring its usage, and sustaining it over time.

Throughout the development process, the Agile methodology strongly emphasises collaboration and communication between developers, stakeholders, and end users. This method enables the programme to change over time to satisfy the users' evolving needs and expectations, ensuring that it continues to be valuable and helpful to them.

## **1.4 Technology Used:**

- **ReactJs:**

React is a free and open source JavaScript tool used to design user interfaces (UIs) for internet applications. It was developed by Facebook, and at the moment it is supported by Facebook as well as a community of independent developers and companies.

React gives developers the ability to create UI components that can be applied to many application regions, simplifying maintenance and updating. Furthermore, it offers a declarative syntax for specifying the organisation and behaviour of the user interface, which facilitates comprehension and debugging.[17]

Redux for state management and React Native for developing mobile applications are two examples of other libraries and frameworks that are frequently integrated with React. Due to the enormous and vibrant community of developers and businesses who actively participate in its development and upkeep, it has emerged as one of the most well-known UI libraries for creating online applications.

React uses a thin replica of the real DOM called a virtual DOM (Document Object Model) to refresh the user interface when updates are made more rapidly and effectively. Furthermore, it offers server-side rendering, a capability that lets software create user interface elements on the server before transmitting them to clients, enhancing performance and SEO (Search Engine Optimisation).

- **NodeJs:**

JavaScript code can run outside of a web browser thanks to a free, open-source runtime environment called Node.js. By allowing them to use JavaScript on the server-side, it enables programmers to build scalable and rapid network applications.

The V8 JavaScript engine, which powers the Google Chrome browser, is the foundation upon which Node.js is constructed. The event-driven, non-blocking I/O (input/output) approach it uses makes it ideal for creating scalable, real-time systems like web servers, APIs, and microservices.[18]

The Node.js package manager, npm (Node Package Manager), which can be quickly and simply integrated into Node.js applications, gives users access to a sizable variety of open-source, reusable packages. Instead of starting from scratch, developers may now use existing code, which makes it easier for them to quickly and efficiently design applications.

Node.js is widely used by developers and businesses of all sorts, from small startups to large enterprises, and it has a robust and active developer community that contributes to its development and maintenance. Due to its widespread adoption and usefulness, it is the favoured choice for creating web applications, APIs, and microservices.

- **ExpressJs:**

Express.js is a well-liked open-source Node.js web application framework. It includes a range of tools and resources that make it simple and quick to build web applications and APIs.[19]

Express.js is renowned for its brevity and straightforward architecture, which makes it easy for developers to get started. It offers a variety of middleware modules that can be used to enhance an application's capabilities, including managing HTTP requests and answers, parsing request bodies, controlling sessions and cookies, and more.

Additionally, Express.js supports a broad variety of templating engines, enabling developers to quickly create dynamic HTML content using information from a server or database. Additionally, it includes a strong routing system that enables developers to define URL routes and map them to certain functions or controllers, making it simpler to create scalable and maintainable systems.

Express.js may be combined with other Node.js frameworks and plugins to build complex web apps and APIs. It is extensively utilised by programmers and companies of all sizes to build web applications and APIs, and it has a substantial and active developer community that actively contributes to its development and maintenance.

- **MongoDB:**

A popular choice for building scalable and flexible applications is the widely used open-source MongoDB document-oriented database, which stores data as adaptable, JSON-like documents with configurable schemas.

MongoDB offers developers a great lot of flexibility and scalability because, unlike traditional relational databases, it stores data in collections rather than tables and allows documents within those collections to have numerous formats and fields.[16]

A few of the features that MongoDB provides to make it easier for developers to deal with data include automatic sharding, built-in replication for high availability and fault tolerance, support for indexing, and aggregation. In addition, MongoDB features a powerful query language and enables distributed transactions, making it easier for programmers to build complex, scalable applications.

Developers and organisations of all sizes regularly utilise MongoDB when building web applications, mobile apps, and other kinds of software that require adaptable and scalable data storage. It has a huge and vibrant developer community that actively participates in its development and upkeep. It is highly known for its ease of use, scalability, and flexibility.

### LITERATURE SURVEY

The usage of effective design is now essential to retaining visitors of websites and mobile applications. To specify the precise components needed in successful website and mobile application design, however, little study has been done. In our study and synthesis of the literature on effective design, we establish a concise set of components that are widely employed in studies. Navigation, graphical representation, organisation, content utility, purpose, simplicity, and readability were the design features that were most frequently referenced in the literature that was studied. We go over the definitions and assessments of these seven characteristics from earlier studies. Design professionals and researchers may find this review and the ensuing short list of design components useful in operationalizing best practices for facilitating and forecasting user interaction.

#### **1) Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js:[1]**

Hoque, S. (2020). Full - Stack React Projects: Learn MERN Stack Development by Building Modern Web Apps Using MongoDB, Express, React, and Node.js, 2nd Edition. United Kingdom: Packt Publishing. Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js" by Shama Hoque is a comprehensive guide for developers wishing to learn how to create full-stack web applications using the MERN stack. The book has a decent layout and provides concise explanations of the theories and procedures necessary to develop modern web apps. Before discussing the tools and technologies required to develop web applications using the MERN stack, the book opens with an overview of the MERN stack and its components. The author then goes into detail about how to build a useful task management application, providing step-by-step instructions for building the application's front end, back end, and database. One of the book's main focuses is on exercises and real-world examples. Exercises are included in each chapter to help readers put what they've learnt into practise and create their own applications.

## **2) Learning React JavaScript Library from Scratch:[2]**

Sidelnikov, G. (2017). React. Js Book: Learning React JavaScript Library from Scratch. (n.p.): Independently Published. Greg Sidelnikov wrote a self-published book titled "React.js Book: Learning React JavaScript Library from Scratch" that serves as an introduction to the React.js library. The step-by-step organisation of the book provides a concise and clear explanation of the concepts required in developing modern web apps with React.

Before discussing the tools and technologies required to build web apps using React, the book opens with an overview of React and its principles. After that, the author delves into developing a practical application, laying out step-by-step instructions for developing both the front-end and back-end of the programme. Throughout the entire book, the author provides useful tips and time-saving techniques for using React to build scalable and maintainable online apps. Thanks to the book's covering of complex topics like routing and data management, readers will have a solid understanding of the entire development process.

## **3) Mastering Node.js.United Kingdom:Packt Publishing:[3]**

Pasquali,S.(2013).Mastering Node.js.United Kingdom:Packt Publishing. Developers who want to learn how to use Node.js to create scalable and fast web apps should read "Mastering Node.js" by Sandro Pasquali. The book is well-structured and gives succinct, straightforward explanations of the technology and ideas related to developing contemporary web apps. An overview of the tools and technologies needed for developing web applications using Node.js follows an introduction to Node.js and its principles in the book's opening chapter. After that, the author delves into creating a practical application, outlining how to create each of its parts, including the front end, back end, and database.

## **4) Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker:[4]**

Zammetti,F.(2020). Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker. United States:Apress. Using TypeScript, React, Node.js, Webpack, and Docker, "Modern Full-Stack Development" by Frank

Zammetti is a comprehensive guide for developers interested in learning how to create modern online apps. Excellent organisation and explanations of the concepts and technology involved in building full-stack web apps are provided in the book.

Before discussing the concepts and tools needed in full-stack web development, TypeScript and React are briefly discussed in the book's opening pages. After that, the author delves into building a real application, laying down step-by-step instructions for building the front end, back end, and database of the software.

### **5) Ten simple rules for researchers who want to develop web apps.[5]**

REVIEW OF Saia, S. M., Nelson, N. G., Young, S. N., Parham, S., & Vandegrift, M. (2022). Ten simple rules for researchers who want to develop web apps. *PLoS Computational Biology*, 18(1). Saia et al.'s article "Ten Simple Rules for Researchers Who Want to Develop Web Apps" (2022) is a useful manual for researchers who want to create web applications for their research. The paper provides ten simple recommendations that researchers can apply to produce effective and practical web apps. This essay's aim is to outline the benefits of developing web apps for research, including improved accessibility and a platform for cooperation. The authors then provide 10 clear suggestions for researchers to follow when developing web apps, including starting with a detailed problem statement and being aware of the needs of the users. One of the paper's highlights is how strongly it emphasises the importance of user-centered design.

### **6) Modern web-development using ReactJS:[6]**

Aggarwal, S. (2018). Modern web-development using ReactJS. *International Journal of Recent Research Aspects*, 5(1). The building of interactive user interfaces uses ReactJS, a component-based technology. It currently holds the top spot among front-end JS libraries. The M-V-C (Model View Controller) pattern is used, which incorporates the view (V) layer. Along with a community of independent developers and organisations, Facebook, Instagram, and other platforms support it. Fundamentally, React makes it possible to create intricate and substantial online applications that may update their data without requiring



further page refreshes. With lightning-fast and reliable web app development, it seeks to offer improved user experiences. Other JavaScript libraries or frameworks, such as AngularJS, can be combined with ReactJS in MVC.

### **7) Towards a Systematic Approach for the Credibility of Human-Centric Web Applications:[7]**

Kamthan P. (2007). Towards a systematic approach for the credibility of human-centric web applications. *Journal of Web Engineering*. 6:2. (99-120). Online publication date: 1-June-2007. The issue of web application credibility in light of growing human engagement and collaboration is taken into consideration in this article. The several stakeholder categories that trustworthiness of web applications is pertinent to are identified. Based on a credibility taxonomy, the causes of the issue of credibility particular to human-centric web applications are investigated, and examples supporting this are shown. It is stressed how important dealing with credibility is while using flexible and iterative development procedures. It is suggested as a framework for methodically analysing and addressing the credibility of human-centric web applications. This framework comprises qualities that are important to stakeholders as well as workable, process- and product-oriented solutions to meet them.

### **8) Secure business application logic for e-commerce systems:[8]**

Nabi F. (2005). *Internet Technology & E-Commerce*. Computers and Security. 24:3. (208-217). Online publication date: 1-May-2005. The business application logic is a significant weak link in e-commerce systems, and it is discussed in this article, *Secure Business Application Logic for e-commerce Systems*. This research is focused on the security of the middle tier of the e-commerce server, which implements the business application logic. Traditionally, e-commerce sites implemented the middle tier of software on the web server using CGI, even though the security issues of the front-end and back-end software systems in e-commerce applications warrant equal attention. Additionally, we discuss methods for securing business application logic, including safe configuration, protective programming, secure server-side software wrappers, and smart design and engineering.

### **9) Protecting user accounts from password database leaks:[9]**

Kontaxis, G., Athanasopoulos, E., Portokalidis, G., and Keromytis, A. D. Sauth:Protecting user accounts from password database leaks. In ACM Conference on Computer and Communications Security (CCS) (2013). This paper discusses the password-based authentication of an application. The most common type of access control in online services is password-based authentication. Sadly, every year it shows itself to be getting worse. Even if users create lengthy, difficult passwords, a service's management of them may contain flaws that allow an attacker to obtain them. Recent occurrences on well-known platforms like LinkedIn and Twitter show the impact that such an event could have.

The development of hardware that makes it possible for potent password-cracking platforms to operate is a direct challenge to the usage of one-way hash algorithms to address the issue. In this article, we suggest SAuth, a protocol that makes use of authentication synergy across several services. Users must authenticate for their account on service V, which serves as a vouching party, in order to access their account on service S. Users frequently visit both services S and V on sites like Twitter, Facebook, and Gmail on a daily basis. If a hacker manages to get the password for service S, he won't be able to access it until he also manages to get the password for service V and perhaps more vouching services. SAuth is an addition to current authentication techniques, not their replacement. It functions at a layer above without being bound to a particular technique, allowing various services to use heterogeneous systems. To secure customers who use the same password across many services, we also use password decoys.

### **10) Encountering stronger password requirements: user attitudes and behaviors:[10]**

Shay, R., Komanduri, S., Kelley, P. G., Leon, P. G., Mazurek, M. L., Bauer, L., Christin, N., and Cranor, L. F. Encountering stronger password requirements: user attitudes and behaviors. In Proceedings of the Symposium on Usable Privacy and Security (SOUPS) (2010). Text-based passwords are still the most commonly used authentication mechanism in information systems. We took advantage of a rare chance offered by a substantial change in the Carnegie Mellon University (CMU) computing services password policy that forced users to reset their passwords. In

our study of 470 CMU computer users, we gathered information on habits and routines related to password use and creation. We also obtained the views of the public regarding the new, stricter policy criteria. According to our analysis, even though most users disliked having to establish complicated passwords, they feel more protected now. In addition, we do an entropy analysis and discuss how our results relate to NIST's recommendations for formulating a password policy. We also look at how users respond to particular inquiries about their passwords. Our findings may be useful in creating better password policies that take into account both the technological and behavioural responses of users to certain policy restrictions.

SYSTEM DEVELOPMENT

3.1 Why do we need a Task Manager Web Application?

A Task Manager Web Application is a useful tool for individuals and organisations to manage their tasks and projects efficiently. Tools for task management enable users to work more effectively, complete more tasks, and achieve more success. Here are some reasons why we need a Task Manager Web Application:

**Organisation:** Users can arrange their jobs and projects in a systematic, understandable manner using a task manager web application. This promotes effective task management and guarantees that nothing is missed.

**Time Management:** Effective time management is facilitated by work prioritisation and deadline setting capabilities provided by task manager web applications. Users can keep tabs on the status of their activities and projects, which is useful for spotting bottlenecks and making necessary adjustments.[6]

**Collaboration:** Within a company, several team members may be engaged on the same task or project. Users can communicate and share information quickly with one another using a task manager web application. This lessens the possibility of misunderstandings and guarantees that everyone is on the same page.

**Accountability:** Users of a task manager web application are able to assign tasks to particular team members and monitor their progress. By making sure everyone is responsible for their tasks, this helps to prevent delays or missed deadlines.

**Accessibility:** As long as there is an internet connection, a Task Manager Web Application can be accessible from any location. This enables users to access their assignments and projects when they are on the go, which boosts productivity and efficiency.

**Enhance productivity and efficiency:** Shorter production cycle turnaround times result from allocating the right mix of time and resources to each task.

**Reduce waste:** Stop wasting time pondering what to do next or redoing work that wasn't done right the first time.

**Meet deadlines:** Missed deadlines become a thing of the past when you and your team use an organised task management system.

Overall, a Task Manager Web Application can help individuals and organisations in managing their tasks and projects efficiently, saving time and improving productivity.

### **Step Of development of task manager app:**

In order to create a Task Manager Web Application using the MERN stack,[15] microservices, passport local approach, Redux, and Redux-thunk, the following general procedures must be followed:

**Gathering needs:** Specify the features and requirements for the Task Manager Web Application.

**Architecture design:** Design the application architecture, including the API endpoints, database schema, and microservices.

**Environment setup:** Installation and configuration of the development environment, which includes Node.js, MongoDB, and other required tools.

**Backend development:** Node.js and Express.js are used to create the application's backend. This entails creating the API endpoints and establishing a database connection.

**Microservices development:** Develop the microservices needed for the application using the microservices architecture.

**Frontend development:** React.js is used to create the application's front end. Included in this are the UI design and implementation, backend API integration, and the addition of Redux for state management.[17]

**Authentication and authorization:** An application might require data from a variety of third-party providers. In this case, the application will prompt the user to "connect" with several accounts, such as Facebook and Twitter. Once this

occurs, the user will already be authenticated with the application, therefore any further third-party accounts only need to be authorised and connected to the user once.[21] Given that authentication and authorisation are equivalent in this situation, Passport provides a mechanism to manage them both. The authorization will be carried out by calling `passport.authorize()`. If authorisation is granted, the result of the strategy's verify callback will be assigned to `req.account`. The current login session and `Req.user` won't be affected.

**Testing:** A functional or requirement specification contains this information. It is a document that outlines what a user is allowed to do so that he can assess if the system or application complies with it.[14] In some cases, this may also require the validation of real-world business side scenarios.

**Deployment:**A deployment pipeline in software development is a set of automated procedures intended to transport new code additions and updates from version control to production with speed and accuracy. Writing, creating, and deploying code required manual procedures in earlier development environments.

**Maintenance:** The constant re-evaluation, re-analysis, and modification of your current software applications is known as application maintenance. Application maintenance must be a continual effort if you want to ensure that your applications are always working as effectively as possible. It is crucial to adapt and embrace new tactics in [11] light of shifting customer expectations, the struggle to survive in an existing market, and technological advancements in order to maintain sustainability and remain competitive. Any competitive firm must constantly maintain and develop the established IT systems in order to remain relevant and satisfy shifting client needs. In this case, application support and maintenance are required.

### **3.2 Technologies/Libraries Used:**

#### **BootStrap:**

The free, open-source front-end framework known as Bootstrap enables the development of responsive and mobile-first websites and online applications. Twitter founded it, and now a developer community looks after it.

A collection of CSS, JavaScript, and HTML components offered by Bootstrap simplify the process of designing and building web pages. Fonts, forms, buttons, navigation bars, modal windows, carousels, and other components make up these elements. Bootstrap includes a grid framework that enables programmers to create flexible layouts that adjust to the size of the screen.

The ability to rapidly and simply create applications and websites with a polished appearance using Bootstrap is one of the main advantages of utilising it. This is possible without requiring a deep understanding of CSS and HTML. It can help developers save time and effort by providing a standardised and consistent approach to web design and development.

Developers can expand and tweak Bootstrap's components to meet their own requirements because of its great degree of customization. Sass, a well-liked CSS preprocessor, can be used to quickly alter the framework, as can custom CSS to replace the pre-installed CSS styles.

Along with its built-in elements, Bootstrap also comes with a number of JavaScript plugins that add further functionality. Examples of these are dropdown menus, tooltips, and scrollspy.

In conclusion, Due to its usability, adaptability, and comprehensive feature set, Bootstrap has become a popular front-end framework among web developers. Due to its popularity, there is a substantial developer community that supports other developers that use Bootstrap by providing support and tools.

Additionally, Bootstrap offers a collection of pre-made CSS classes and JavaScript plugins that may be used to add interactive and dynamic features to websites. For example, a carousel plugin from Bootstrap may be used to create image slideshows, and a modal plugin from Bootstrap can be used to display pop-up windows with more information or forms.

A further advantage of Bootstrap is the sizeable and vibrant developer community that supports and helps other developers who use the framework. This community has produced a tonne of third-party assets, like as templates, themes, and plugins, that can be utilised to enhance the functionality of Bootstrap or quickly construct unique websites and applications.

## **ReactJs:**

ReactJS, also referred to as React, is a free JavaScript package used to create user interfaces (UIs) or other front-end elements for online applications. Facebook created it, and businesses all across the world are now using it extensively.

Declarative design refers to the idea that the developer informs React how they want the user interface to look, and React handles the rest. By dividing them into discrete, reusable components that can be maintained individually, it enables developers to design complicated user interfaces. Developers can put HTML-like code directly in their JavaScript using the JSX syntax, which can be used to create React components.

One of React's unique features is support for the virtual DOM (Document Object Model). The virtual DOM, a slender approximation of the real DOM, allows React to change the UI successfully without having to directly alter the real one. This makes React incredibly speedy and efficient, especially when compared to other traditional front-end frameworks.[17]

In order to increase the effectiveness of the application, React also provides server-side rendering, which enables designers to develop the first user interface (UI) on the server and provide it to the client. Numerous back-end frameworks are compatible with React, and it typically partners with Node.js and Express.js.

Another benefit of React is its sizable and vibrant developer community, which supports and helps other React users while also making improvements to the framework. This community has produced a large number of third-party libraries and tools that may be used to enhance React's features or fast construct original applications. Redux, React Router, and React Native are three well-known React-based libraries.

React is widely used by organisations of all sizes and in many sectors, such as Facebook, Airbnb, Uber, and Netflix. It is frequently combined with other front-end technologies like Bootstrap and Material UI and is a popular choice for creating complicated user interfaces and single-page applications (SPAs).



Finally, ReactJS is a strong and flexible front-end toolkit that enables programmers to design intricate and dynamic user interfaces for online applications. It is quick and effective thanks to its declarative approach, virtual DOM, and server-side rendering, and any project can rely on it because of its big and active development community.

### **Redux:**

For JavaScript applications, Redux is a predictable state container that is frequently used with React. Redux gives developers a means to control an application's state in a single, immutable store, making it simpler to analyse how the state evolves over time. Because it clearly separates presentation logic from business logic, Redux has grown to be a popular solution for creating large-scale apps. This can make it simpler to maintain and test code over time.

Redux's central idea is the store, a JavaScript object that contains all of an application's state. Since the storage is immutable, no changes can be made to it directly. Instead, dispatching actions—basic JavaScript objects that explain what transpired in the application—is used to make changes to the store. Action makers, functions that produce action objects, are responsible for creating actions. Reducers are pure functions that take the current state and the action as input and return a new state based on the action. Once an action is dispatched, it is processed by reducers.

Due to their responsibility for managing state changes, reducers serve as the foundation of the Redux architecture. Reducers are pure functions, which implies they don't actually alter the state. Instead, they make a fresh copy of the state and deliver it, keeping the original state unaltered. Due of reducers' predictability and lack of negative side effects, it is simpler to reason about state changes that occur over time.

Middleware is a crucial idea in Redux because it gives the dispatching mechanism more functionality. Prior to being handled by reducers, middleware can be used to manage asynchronous actions, log actions, or alter actions. As actions are dispatched, middleware is composed of a sequence of functions that can pass the

action on to the subsequent middleware in the chain, change the action, or prevent it from being handled at all.

The view layer, which is in charge of showing the state of the application to the user, is the last component of the Redux architecture. React commonly implements the display layer using components, which are reusable UI building blocks that may be combined to construct complex user interfaces. By mapping the state and actions from the store to the component's props, the `connect()` function allows React components to be linked to the Redux store.

Redux offers a strong and adaptable method for controlling the state of complicated applications overall. It is a popular option for developing large-scale apps with React due to its predictable and immutable architecture, which makes it simpler to reason about changes to the state over time. Its middleware and view layer integration is another benefit. Redux is not a panacea, though, and it might not be suitable for all projects, like any technology. In order to select the right tools and architecture for your unique use case, it is crucial to assess the requirements of your application.

### **Redux-Thunk:**

You can create asynchronous logic that communicates with a Redux store using the middleware for Redux called Redux-Thunk. It offers a method to deal with asynchronous operations like AJAX calls and other side effects that are frequently needed in contemporary web applications.

Redux uses a store to keep track of the application's state, and reducers to update that state in response to dispatched actions. Redux, however, does not have a built-in method for dealing with asynchronous activities. Redux-Thunk steps in to help with it. Redux-Thunk enables the creation of action producers that return functions as opposed to simple action objects. These functions can then carry out asynchronous activities and send out commands in response to the outcomes of those operations. This technique is known as "thunking" an action.

Simply said, a thunk is a function that returns another function that can be used later. A thunk is a function that returns another function that can communicate with the Redux store in the context of Redux-Thunk.

```

const fetchData = () => {
  return (dispatch) => {
    dispatch({ type: 'FETCH_DATA_START' });
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => dispatch({ type: 'FETCH_DATA_SUCCESS', payload: data }))
      .catch(error => dispatch({ type: 'FETCH_DATA_FAILURE', payload: error }));
  };
};

```

**Fig 3.1 : Use of Redux-Thunk Concept**

In this case, the function returned by the thunk `fetchData` takes a `dispatch` argument. To signal that we are beginning to fetch data, we dispatch an `FETCH_DATA_START` action inside that method. Then, we use the `fetch` function to initiate an AJAX call. Depending on whether the AJAX request succeeded or failed, we send either a `FETCH_DATA_SUCCESS` or `FETCH_DATA_FAILURE` action after it has finished.

We would generally import and dispatch this thunk using the `useDispatch` hook from the `react-redux` library in order to use it in a component:

```

import { useDispatch } from 'react-redux';
import { fetchData } from './actions';

const MyComponent = () => {
  const dispatch = useDispatch();

  useEffect(() => {
    dispatch(fetchData());
  }, []);

  return (
    // ...
  );
};

```

**Fig 3.2 : Use of Redux-Thunk Concept(useDispatch and useEffect)**

In this illustration, the `fetchData` thunk is sent when the component mounts using the `useEffect` hook. To actually dispatch the action, we are utilising the `dispatch` method supplied by the `useDispatch` hook.

Redux-Thunk, as mentioned before, is a middleware for Redux that enables you to include asynchronous logic in your action creators. It functions by enabling you to return functions from your action creators as opposed to straightforward action objects. These functions have the ability to carry out asynchronous operations and dispatch actions in response to the outcomes of those operations. As a result, managing side effects like AJAX calls and other asynchronous actions in your Redux store is made simpler.

### **NodeJs:**

Using JavaScript, developers may create server-side apps with the help of the open-source, cross-platform runtime environment Node.js. It supports JavaScript execution outside of the browser and is developed on top of the Google Chrome V8 engine. Node.js has an event-driven, non-blocking I/O architecture that makes it lightweight and effective, making it the best option for developing scalable and high-performance network applications.

Node.js has become into one of the most well-liked frameworks for building server-side applications since Ryan Dahl first introduced it in 2009. Developers from all around the world routinely create web applications, APIs, command-line tools, real-time applications, and other things.

One of Node.js's key characteristics is its ability to manage numerous requests and large amounts of data concurrently without causing other processes to halt. The architecture's event-driven design, which executes code without blocking while each I/O action starts an event, enables this. Because of this, Node.js is excellent for building real-time applications like chat programmes and online gaming platforms.[18]

The enormous library of modules and packages that Node.js offers, accessible via the Node Package Manager (npm), is another benefit of the software. The usage of pre-built modules for frequent activities like managing HTTP requests, parsing

JSON data, interacting with databases, and more is made simple for developers thanks to this.

As a result, programmers may create applications utilising a unified language and paradigm thanks to Node.js's support for JavaScript on both the client and server sides. Application development becomes less difficult as a result, and client and server code sharing is made simpler.

The flexibility to add or remove modules based on the particular requirements of the application makes Node.js another highly adaptable platform. As a result, apps load more quickly and perform better because of developers' ability to build lightweight programmes that only contain necessary features.

Node.js has a robust ecosystem of tools and frameworks that expand its capability in addition to the essential features it already offers. Among the most well-known Node.js frameworks are Express.js, which is used to create web apps, Socket.io, which is used to create real-time applications, and Nest.js, which is used to create scalable server-side applications.

In general, Node.js gives developers a strong and adaptable framework for creating JavaScript-based server-side applications. It is a preferred option for creating scalable, high-performance applications because of its event-driven, non-blocking I/O style, large module library, and support for client-side JavaScript.

### **ExpressJs:**

On top of Node.js, Express.js is a well-liked online application framework. It offers a comprehensive collection of capabilities for building web apps and APIs. Express.js makes it easier to create web apps by giving users access to a small but mighty set of tools for managing HTTP requests and answers as well as middleware for adding new features.

Developers can build scalable and maintainable apps with Express.js, which is renowned for its simplicity and versatility. It offers a straightforward and opinionated core, thus it won't push a specific application structure on you. This makes it simple to modify and fit to the particular requirements of your project.

A key element of the Express.js framework is middleware. Middleware functions are those that have access to the request and response objects in an application's request-response cycle. They are capable of handling faults and logging data. By allowing middleware routines to be added to the chain of requests being processed, the `app.use()` method makes it easy to change an application's behaviour.[19]

Express.js's routing feature enables programmers to offer a number of routes for handling HTTP requests. Routes can be built using `App.get()`, `App.post()`, `App.put()`, `App.delete()`, and other methods with a similar syntax. This makes it simple to group your application's functionality into logical sections and to respond to different request types in different ways.

Express.js' support for templates is a crucial component. By inserting data into a preset template, templates enable developers to create dynamic HTML pages. Jade, EJS, and Handlebars are a few of the template engines supported by Express.js.

Working with databases such as MongoDB, MySQL, and PostgreSQL is also supported natively by Express.js. In addition to support for connection pooling and transaction management, it offers a straightforward and consistent API for carrying out CRUD (Create, Read, Update, Delete) operations on the database.

In conclusion, Express.js is an effective and adaptable online application framework that offers a wealth of functionality for creating web apps and APIs. It is the perfect option for developers who wish to quickly and simply create scalable and maintainable applications due to its simplicity and flexibility. It is a popular option for developing web applications of various shapes and sizes because of its middleware architecture, support for routing and templates, and integrated database support.

### **MongoDB:**

The widely used open-source MongoDB document-oriented database stores data as adaptable, JSON-like documents with changeable schemas, making it a popular option for creating scalable and adaptable applications.

MongoDB, in contrast to conventional relational databases, stores data in collections rather than tables, and documents within those collections can have various forms and fields, giving developers a great deal of flexibility and scalability.[16]

Automatic sharding, built-in replication for high availability and fault tolerance, support for indexing and aggregation are just a few of the features that MongoDB offers to make it simpler for developers to interact with data.

Additionally, MongoDB includes a robust query language and supports distributed transactions, which makes it simpler for developers to create intricate, scalable systems.

When creating web applications, mobile apps, and other types of software that need flexible and scalable data storage, developers and businesses of all sizes frequently use MongoDB. It is well known for its simplicity of use, scalability, and flexibility and has a sizable and active community of developers who contribute to its development and maintenance.

### **Redis:**

Redis is an open-source, in-memory data structure store that can be used as a database, cache, and message broker. It was first created by Salvatore Sanfilippo in 2009 and has since grown to be one of the most well-liked NoSQL databases. Redis is well renowned for its great performance, versatility, and scalability and is frequently used in real-time analytics, online applications, and other applications where high-throughput data storage and retrieval is essential.

Since Redis is a key-value store, data is kept in key-value pairs. Values can include texts, hashes, lists, sets, and ordered sets among other data structures. Redis offers a wide range of data manipulation commands for various types of data structures, enabling programmers to create intricate data structures and carry out complex operations on them.

Redis's speed is one of its primary characteristics. Redis is perfect for applications that require quick data access since it saves data in memory, enabling extremely rapid read and write rates. Redis also provides data persistence, which enables

periodic or on-demand disc storage of data for durability and data recovery in the event of a system crash or power loss.

Redis is incredibly scalable both horizontally and vertically. It has the potential to process huge datasets with millions or even billions of keys and can be set up as a distributed cluster setup for even more scalability. Redis has replication and sharding capability, which enables it to manage extremely heavy write loads and keep excellent availability.

Redis is frequently utilised as a web application caching layer in addition to being a database. Web applications can increase performance and lessen the stress on their main database by caching frequently used data in Redis. Additionally, Redis has a variety of capabilities that are intended expressly for caching, including as support for time-to-live (TTL) key expiration and LRU key eviction.

As a message broker for real-time applications, Redis is another frequent application. Due to Redis' support for pub/sub messaging, several clients can subscribe to a channel and instantly receive messages. Redis is the best option for creating real-time applications because of this, including chat systems, live sports updates, and stock market apps.

Overall, Redis is a strong and adaptable technology that can be applied to a variety of applications. For developers looking for a quick and dependable data store, cache layer, or message broker for their applications, it is a compelling option because to its speed, scalability, and broad feature set.

### **Passport:**

Passport is a middleware for Node.js that offers a quick and adaptable foundation for web application authentication. It enables programmers to quickly create a number of authentication methods, including OAuth, OpenID, and local authentication with a username and password.

The primary benefit of Passport is its modularity, which enables developers to employ only the authentication methods necessary for their application. Through the use of "strategies" in Passport, which are effectively pluggable authentication



modules, this modularity is accomplished. Developers can choose from a variety of preexisting strategies or design their own unique ones.

Passport offers two main methods for authentication: `passport.authenticate()` and `passport.serializeUser()`, which together make up a straightforward yet effective API. A user is authenticated using the `passport.authenticate()` function, and a user object is serialised into a session using `passport.serializeUser()`. The user object from the session can also be deserialized using the `passport.deserializeUser()` method.

Additionally, Passport offers middleware features for securing routes that demand authentication. To make sure that only authenticated users have access to them, these middleware functions can be added to routes.

OAuth authentication, which is utilised by many well-known websites and services, like Facebook, Google, and Twitter, is supported by Passport and is one of its core features. It is simple to incorporate OAuth authentication into your application because Passport offers pre-built OAuth strategies for these providers. Passport's support for OpenID authentication, which enables users to authenticate using their current OpenID credentials, is another significant feature. Numerous well-known OpenID providers, including Yahoo, Google, and PayPal, are supported by pre-built OpenID strategies from Passport.[21]

Additionally supported by Passport is local authentication, which is frequently used for login and password security. The local strategy offered by Passport, referred to as `passport-local`, offers a simple structure for including local authentication in your application. The "verify functions" concept—functions that accept a username and password as input and return either a user object or an error—is the foundation of the `passport-local` strategy.

Passport is an all-around strong and adaptable authentication framework for Node.js that offers a user-friendly and straightforward API for integrating different authentication schemes in your online applications.

## **Passport-Local:**

With Passport.js, a Node.js authentication package, Passport-Local is a middleware that can be utilised. Based on a username and password combination, it is used to authenticate users. Passport-Local offers a method for Passport that confirms the user's credentials using a local database or another data source.[21]

Users can be verified using a username and password that are kept locally by using the Passport-Local Strategy. It is a module that may be added to a Node.js application along with Passport.js to perform local authentication. Two functions are needed for the strategy: a verification function and a serialisation function.

The verification function is used to check the user's credentials. After receiving the user's username and password, it checks the local database to see if the user is there. The function verifies the password against the database's password once the user has been located. The function returns the user object in the event that the password is accurate. The function returns a false value if the password entered is wrong.

To save the user object in the session, use the serialisation function. The user object is kept in the session after the user has been authenticated. To extract the user object from the session and attach it to the request object, use the serialisation function. Databases like MySQL, PostgreSQL, MongoDB, and many others can be used with the Passport-Local Strategy. Based on a username and password combination, it offers a quick and efficient method of authenticating users.

Passport-Local Strategy can be used with a variety of authentication techniques, including multi-factor authentication, two-factor authentication, and social authentication (Facebook, Google, and Twitter), in addition to the standard username/password authentication. It gives your Node.js application a flexible and modular solution to integrate authentication.

The ease with which Passport-Local Strategy can be connected with other Passport.js strategies is one of its benefits. For local and social authentication, respectively, you can utilise the Passport-Local Strategy and the Passport-Facebook Strategy. You may now give your users a variety of authentication choices thanks to this.

The high level of security offered by the Passport-Local Strategy is another benefit. The database contains the passwords in an encrypted format. This makes sure that the passwords won't be available to the attacker even if the database is compromised.

As a result, the Passport-Local Strategy is a well-liked and successful method of integrating local authentication into a Node.js application. It offers a quick and adaptable method for authenticating users using a username and password combination. It can be simply combined with other Passport.js techniques and is also quite safe.

### **Bcrypt:**

We have used this library for the encryption and decryption of the passwords using the salt. Popular password-hashing technology called Bcrypt is used to store passwords securely in computer systems. As a more secure substitute for previous password-hashing procedures at the time, Niels Provos and David Mazières first presented it in 1999. As a result of its extensive acceptance, many computer languages now consider Bcrypt to be a standard tool for password hashing.

To safeguard user credentials in the event of a data breach, password hashing is used. Passwords are the most private data in a system, and if they are kept in plaintext or with insufficient encryption, they can be readily hacked. A password is inputted into a one-way procedure called hashing, which outputs a fixed-length string of characters that the system stores. In order to confirm a user's identity, their password is hashed and compared to the hash that has been previously recorded.

It is more challenging for attackers to break passwords using brute-force techniques since Bcrypt is intended to be slow and computationally costly. The Blowfish cypher is used to hash passwords, and the system's security can be increased by modifying the number of computation repetitions. This indicates that even if an attacker obtained the hashed passwords, it would be difficult to decipher them.

Another feature of Bcrypt is salt, which involves pre-hashing each password with a random string of characters. Salting increases the difficulty of password

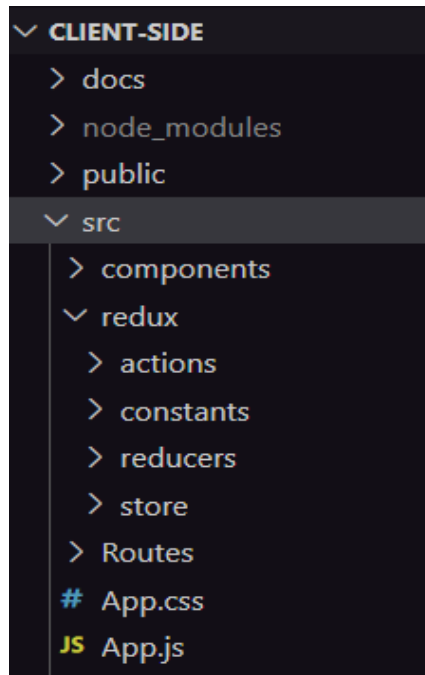
cracking by attackers using pre-generated tables of hashes. Attackers would have to recompute each password's hash with a different salt, which would take a lot longer and be more difficult. Bcrypt has evolved into the de facto industry standard for password hashing in online applications, mobile applications, and other systems that necessitate safe password storage. There are numerous libraries that make using bcrypt in your application straightforward. Several programming languages, including Python, Ruby, Java, and JavaScript, have all used it.

In conclusion, the passwords of users are protected in the event of a data breach thanks to the secure and reliable password-hashing function known as bcrypt. It is intended to be slow and computationally demanding, which makes it harder for attackers to break passwords using brute-force techniques. It is frequently utilised in web and mobile applications to safely store passwords and contains features like salt to boost system security.

### **3.3 Folder Structure:**

#### **Frontend:**

Firstly we have our root folder named as client-side. Inside this root folder we have a public folder which contains an src folder which has stored all the code's folder and files inside it. Then we have separate folders for componentes, redux, and routes.



**Fig 3.3 : Folder structure Of Client-side**

**Components:**

Inside the components folder we have all listed the components files which have been used in the projects. Components folder also contains another folder named as CSS which has all the css code required in the project with respect to every component.

**Redux:**

Redux folder contains three folder inside it named as Actions, Constants, Reducers and Store.

**Action:**

Action folder contains all the files which have been used for the communication purposes with the server side using the redux-thunk.

**Constant:**

The constant folder contains a file which have all the constant value used in the project.

### **Reducers:**

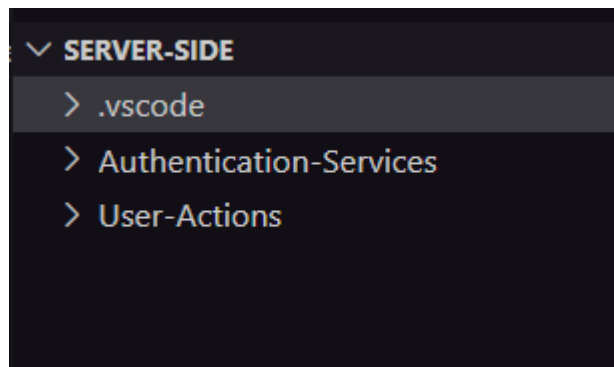
The Reducer folder contains all the files which helps the React for state management and better flow of the application. It also has an index.js file which combines all the reducers together.

### **Store:**

The Store folder has an index.js file which creates a store and exports it. This is also used to use the middleware like thunk in the project.

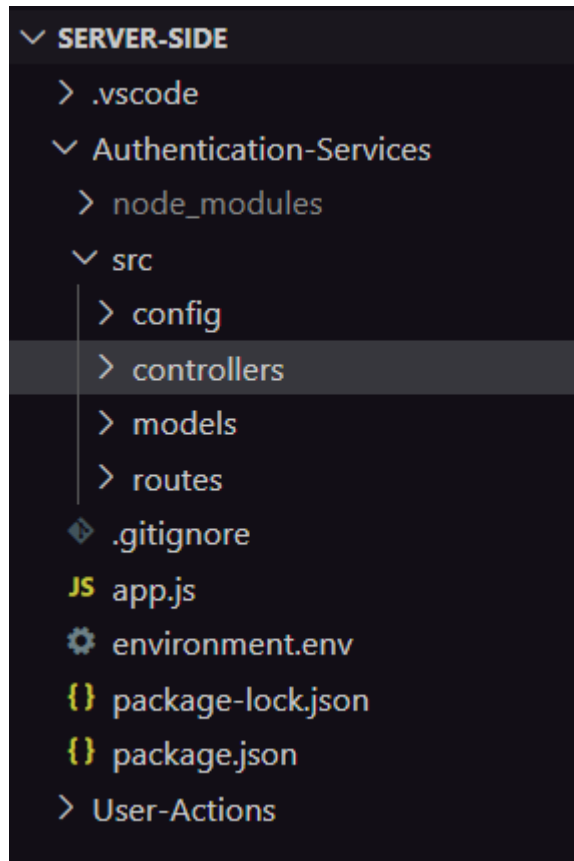
### **Backend:**

Firstly we have our root folder named as server-side. Inside this root folder we have two microservices named as Authentication-Services and User-Actions.



**Fig 3.4 : Folder structure Of Microservices**

These two microservices have separate folder structures. Inside each microservice which contains the src folder. Inside this src it has config, controllers, models and routes folder.



**Fig 3.5 : Folder structure Of Server-side**

**Config:**

Config folder contains a file named passport.js which have the code for the authentication purpose.

**Controller:**

Controller folder contains files which have the API function.

**Models:**

Models have a file name as connection.js which has code for the connection of the server with the database. It also contains the file which has the schema code of the database.

**Routes:**

The Route folder contains a route.js file which contains all the API routes.

## Schema of the Database:

### Organization Schema:

```
const { ObjectId } = require('mongodb');
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);

const OrganizationSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  details: {
    type: String,
    required: true
  }
})

const Organization = new
mongoose.model("Organizations", OrganizationSchema);
module.exports = Organization;
```



## User Schema:

```
const { ObjectId } = require('mongoose');
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);
const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true
  },
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  password: {
    type: String,
    required: true,
  }, role: {
    type: String,
    required: true,
  },
  salt: {
    type: String
  },
  orgId: {
    type: ObjectId,
  },
  reportingTo: {
    type: ObjectId
  },
  createdBy: {
    type: ObjectId
  },
  updatedBy: {
    type: ObjectId
  }
})
const User = new mongoose.model("Users", UserSchema);
module.exports = User;
```

## Task Schema:

```
const { ObjectId } = require('mongodb');
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);
const TaskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  userId:{
    type:ObjectId,
  },
  status:{
    type:String
  },
  createdBy:{
    type:ObjectId
  },
  updatedBy:{
    type:ObjectId
  }
});

const Task = new mongoose.model("tasks", TaskSchema);
module.exports = Task;
```

## Connection File:

```
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);
const uri = 'mongodb://127.0.0.1:27017/taskManager'
mongoose.connect(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).
then(()=>{
  console.log("Connected to DB");
}).catch((err)=>{
  console.log(err);
});
```

## Organization Collection of MongoDB:

### taskManager.organizations

Documents Aggregations Schema Explain Plan Indexes Validation

Filter   Type a query: { field: 'value' }

 ADD DATA  EXPORT COLLECTION

```
_id: ObjectId('643f7d86069b771d43e6d189')
name: "Paxcom"
email: "paxcom2023@gmail.com"
details: "Paxcom India (P) Ltd - A Paymentus Company"
__v: 0
```

```
_id: ObjectId('643f7e0c069b771d43e6d18c')
name: "Amazon"
email: "amazon2023@gmail.com"
details: "AWS - A Cloud Service Provider"
__v: 0
```

```
_id: ObjectId('6450d2a16ce62a74d5be4408')
name: "Flipkart"
email: "flipkart2023@gmail.com"
details: "A Ecommerce Web Application"
__v: 0
```

```
_id: ObjectId('6450d2db6ce62a74d5be440d')
name: "Flipkart"
email: "flipkart2023@gmail.com"
details: "A Ecommerce Web Application"
```

## Task Collection of MongoDB:

### taskManager.tasks

Documents Aggregations Schema Explain Plan Indexes Validation

Filter Type a query: { field: 'value' }

ADD DATA EXPORT COLLECTION

```
_id: ObjectId('64465bfe0a34660325379ff7')
title: "First Task"
description: "This is Updated First Task"
userId: ObjectId('644621da67978b5a025fbef7')
status: "pending"
createdBy: ObjectId('644621da67978b5a025fbef7')
__v: 0
```

```
_id: ObjectId('64536691963ab259064b7132')
title: "Second Task"
description: "This is Second Task"
status: "pending"
createdBy: ObjectId('64461e0627222785716a61b5')
__v: 0
```

```
_id: ObjectId('6454963fbd3db0062aa629ff')
title: "First Admin todo"
description: "First Admin todo Description"
userId: ObjectId('644623895840d21edf5e561e')
status: "completed"
createdBy: ObjectId('64461e0627222785716a61b5')
__v: 0
```

## Users Collection of MongoDB:

### taskManager.users

Documents Aggregations Schema Explain Plan Indexes Validation

Filter Type a query: { field: 'value' }

ADD DATA EXPORT COLLECTION

```
_id: ObjectId('643ed8a7a4a9b63760a4b495')
name: "superAdmin"
email: "superAdmin2023@gmail.com"
password: "superAdmin2023"
__v: 0
role: "superAdmin"
username: "superadmin"
```

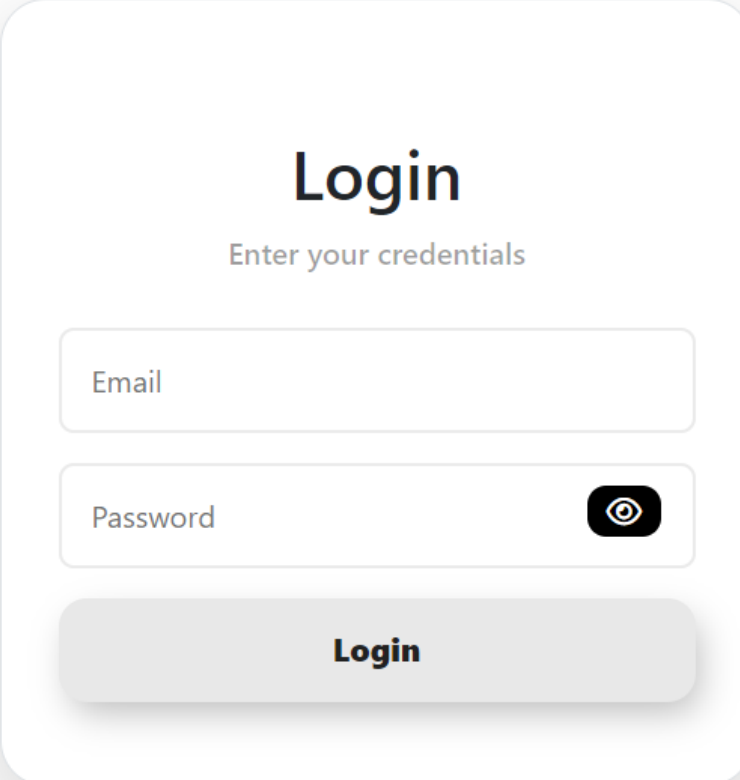
```
_id: ObjectId('64461e0627222785716a61b5')
username: "abcd"
name: "abcd"
email: "abcd@gmail.com"
password: "$2a$10$B3XfA8LFDZHo8nbczQ/Z70A.T1.234bQUmna6Xrgb5aUfscZVfvG"
role: "admin"
salt: "$2a$10$B3XfA8LFDZHo8nbczQ/Z70"
orgId: ObjectId('643f7d86069b771d43e6d189')
reportingTo: ObjectId('643ed8a7a4a9b63760a4b495')
createdBy: ObjectId('643ed8a7a4a9b63760a4b495')
updatedBy: ObjectId('643ed8a7a4a9b63760a4b495')
__v: 0
```

```
_id: ObjectId('644621da67978b5a025fbef7')
username: "abcdef"
name: "abcdef"
email: "abcdef@gmail.com"
password: "$2a$10$4o8qDk4U0sZGDY6RgNsFgu/18q/A7PQHlu1Pm/KBV/UotgCY2Gtjq"
role: "user"
salt: "$2a$10$4o8qDk4U0sZGDY6RgNsFgu"
orgId: ObjectId('643f7d86069b771d43e6d189')
reportingTo: ObjectId('64461e0627222785716a61b5')
createdBy: ObjectId('64461e0627222785716a61b5')
updatedBy: ObjectId('64461e0627222785716a61b5')
--v: 0
```

```
_id: ObjectId('644623895840d21edf5e561e')
username: "abcdefgh"
name: "abcdefgh"
email: "abcdefgh@gmail.com"
password: "$2a$10$g.bW0d/09wwf5wdOjwo.60fijsg0zq2f5GtwzjSDEQAfofi8HC0tm"
role: "user"
salt: "$2a$10$g.bW0d/09wwf5wdOjwo.60"
orgId: ObjectId('643f7e0c969b771d43e6d18c')
reportingTo: ObjectId('64461e0627222785716a61b5')
createdBy: ObjectId('64461e0627222785716a61b5')
updatedBy: ObjectId('64461e0627222785716a61b5')
--v: 0
```

#### 4) Flow Of The Application:

##### Authentication:



The image shows a login form with a white background and rounded corners. At the top, the word "Login" is written in a large, bold, black font. Below it, the text "Enter your credentials" is displayed in a smaller, gray font. There are two input fields: one for "Email" and one for "Password". The "Password" field has a black eye icon on the right side, indicating a toggle for password visibility. Below the input fields is a large, gray, rounded button with the word "Login" written in bold black text.

Fig 3.6 : Authentication Page

##### 2) Super Admin Interface:

Task Manager Create-Admin Create-organization Logout

username Name Email Password  
 Select Organization Submit

### Admin List

Search by Name : Select No. Of Item

Username	Name	Email	Delete
nverma	Narendra	narendra.verma8005@gmail.com	
jain	Atishya	atishyajain@gmail.com	
shaan	Shaan Srivastava	shaan@gmail.com	
nyadav	Navya Yadav	navyadav0417@gmail.com	

Previous 1 2 3 4 5 6 Next

**Fig 3.7 : Create Admin Page And Show Admin Data Table**

Task Manager Create-Admin Create-organization Logout

Organization Name Organization Email Organization Detail Submit

### Organization List

Search by Name : Select No. Of Item

Organization Name	Organization Email	Organization Details	Delete
Paxcom	paxcom2023@gmail.com	Paxcom India (P) Ltd - A Paymentus Company	
Amazon	amazon2023@gmail.com	AWS - A Cloud Service Provider	
Flipkart	fiipkart2023@gmail.com	A Ecommerce Web Application	
Myntra	myntra2023@gmail.com	A shopping Web App	
Nykaa	nykaa2023@gmail.com	A Ecommerce WebSite	

Previous 1 Next

**Fig 3.8 : Create Organization Page And Show Organization Data Table**

### 3) Admin Interface:

Task Manager Create-User Create-Task Logout

#### User List

Select No. Of Item 
Search by Name :

Name	Organization	Email	Delete
shaanG	Shaan Grover	shaan.grover143@gmail.com	<input type="button" value="Delete"/>
nverma	Narendra Verma	narendra.verma8005@gmail.com	<input type="button" value="Delete"/>
kbhandari	Kunal Bhandari	kunal.bhandari2001@gmail.com	<input type="button" value="Delete"/>

**Fig 3.19 : Create User Page And Show User DataTable**

Task Manager Create-User Create-Task Logout

#### Admin Todo

Select No. Of Item 
Search by Name :

	Title	Description	Status	
<input type="checkbox"/>	Study App Smith	Study and make notes of App Smith From documentation	Pending	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
<input checked="" type="checkbox"/>	Study About Apache hop	Study and make notes of Apache hop From documentation	Pending	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

**Fig 3.10 : Create Task for user and change status of the task**

Task Manager Create-User Create-Task Logout

#### Admin Todo List

Select No. Of Item 
Search by Name :

	Title	Description	Status	
<input type="checkbox"/>	Study App Smith	Study and make notes of App Smith From documentation	Pending	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
<input checked="" type="checkbox"/>	Study About Apache hop	Study and make notes of Apache hop From documentation	Pending	<input type="button" value="Edit"/> <input type="button" value="Edit Todo"/>

**Fig 3.11 : Edit A Particular Task**

### 3) User Interface:

Task Manager Logout

Todo Title:  Description:  Create Todo

Select No. Of Item:  Search by Name:

	Name	Description	Status	Action
<input type="checkbox"/>	Study About Passport Js	Study and Make notes of PassportJs	Pending	<a href="#">✎</a> <a href="#">🗑</a>
<input type="checkbox"/>	Complete Assignment	Complete the C++ Assignment	Completed	<a href="#">✎</a> <a href="#">🗑</a>

Previous Next

**Fig 3.12 : Create Task For User**

Task Manager Logout

Todo Title:  Description:  Create Todo

Select No. Of Item:  Search by Name:

	Name	Description	Status	Action
<input type="checkbox"/>	Study About Passport Js	Study and Make notes of PassportJs	Pending	<a href="#">✎</a> <a href="#">🗑</a>

Previous Next

**Fig 3.13 : Search Task by Name**



### PERFORMANCE ANALYSIS

The MERN (MongoDB, ExpressJS, ReactJS, and NodeJS) stack performance analysis of a task manager app comprises measuring numerous performance indicators, such as load time, response time, and scalability. When examining the functionality of a task management application using the MERN stack, keep the following points in mind:[1]

A website or web application's load time has a significant impact on the user experience. Slow load times can annoy users and lead to high bounce rates, as users abandon the website without interacting with it. Therefore, it is important to measure and improve the task management application's load time once it was built using the MERN stack.

To gauge how quickly a web page loads, utilise programmes like Google PageSpeed Insights or GTmetrix. These tools provide a complete analysis of the website's functionality and offer suggestions for improvements that can shorten load times.[12] Examples of usual optimisations include image compression, code minification, lowering server response times, and using a content delivery network (CDN) to provide static files.

A task management application developed utilising the MERN stack must also consider response time while evaluating its functionality and user interface. Response time is the amount of time it takes the server to respond to a client request. It includes the time it takes to process the request, extract the requested information from the database, and provide the client their response. User annoyance from slow response times can lead to high bounce rates and poor user engagement.

Tools like New Relic or AppDynamics can be used to measure response time. These tools offer thorough performance analytics for your application, such as the average response time for various queries. You can locate slow-running queries, enhance database indexes, and enhance server-side code by examining the response timings for various requests.

Any task manager application created utilising the MERN stack must be scalable. The ability of the software to accommodate growing user demand and traffic without compromising on performance is referred to as scalability. It's critical to confirm that the app can manage the additional load and deliver a consistent user experience as the user base and traffic to the app expand.

A task manager app's scalability can be evaluated using load testing software like JMeter or LoadRunner. The purpose of these tools is to stress-test the app and find any potential bottlenecks by simulating real-world traffic and user scenarios. You may find the parts of the programme that need to be optimised to handle additional traffic and users by looking at the performance data during load testing. One of the most important advantages of using a task manager app for businesses is improved communication.[14] Any team must have effective communication in order to collaborate effectively and accomplish shared objectives. Task managers may significantly enhance communication both within a team and beyond the organisation by offering a centralised platform for team collaboration.

Team members can communicate in real-time updates, comments, and progress reports thanks to task managers. This implies that everyone, even those who aren't physically present, can stay informed about what's going on a project or assignment. There are fewer risks of misconceptions or miscommunication when team members can effectively communicate, share ideas, and ask questions.

In general, task managers are crucial tools for businesses trying to increase output, teamwork, and communication. They aid in making sure that everyone is pursuing the same objectives and that resources are being utilised effectively. Task managers will be more crucial as businesses rely more on digital collaboration and remote work to enable teams to work together efficiently across all locations.

### **Response Time of the Website:**

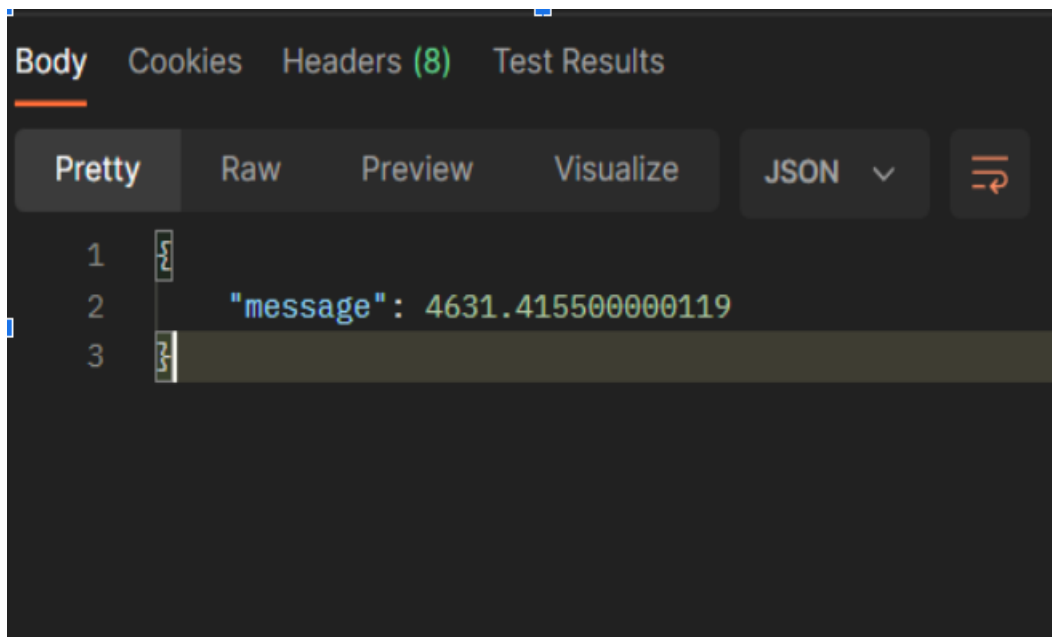
Perf\_hooks, a Node.js built-in module, provides a way to measure how well Node.js applications are performing. It is used to estimate how long the code's functions take to execute. [12]

By monitoring the time spent by a specific block of code or function using perf\_hooks, developers can identify bottlenecks and speed up their code.

## Code:

```
router.get("/performance/getPerformance", (req, res) => {
  const t1 = performance.now();
  const responseTime = t1 - t0;
  res.status(201).json({message: responseTime});
});
```

**Fig. 4.1: Response Time API**



**Fig. 4.2: Output of the response time API**

## React Hooks:

React version 16.8 introduced a set of functions called React Hooks as a new technique for controlling state and lifecycle processes in functional components. State and lifecycle methods could only be utilised in class components before the invention of hooks.[17]

By enabling developers to leverage state and other React capabilities within functional components, hooks offer an alternative to using classes. As a result, developers can now manage state and other React capabilities without the requirement for classes, giving functional components increased power and ease of use.

## Hooks in this project:

### useState:

One of the built-in Hooks in React that enables developers to control state in functional components is the `useState()` Hook, which was added in version 16.8. A state variable and a function that can be used to update that variable are declared using the `useState()` Hook. The following is the syntax for the `useState()` Hook:[17]

```
const [stateVariable, setStateFunction] = useState(initialValue);
```

**Fig. 4.3: useState Syntax**

Here, `setStateFunction` is the function that can be used to update the state variable, `initialValue` is the state variable's initial value, and `stateVariable` is the name of the state variable that is being declared. Consider the scenario where we want to use a functional component to build a counter. To control the counter variable's status, we can use the `useState()` Hook:[17]

```

import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    setCount(count + 1);
  }

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={incrementCount}>
        Click me
      </button>
    </div>
  );
}

```

**Fig. 4.4: useState Example**

### **useEffect:**

The `useEffect()` A further built-in Hook in React that enables developers to carry out side effects in functional components is called Hook. Side effects are operations that take place outside of the component, such as DOM updates, event listener configuration, or data retrieval from an API. Two arguments are required by the `useEffect()` Hook: a function and an optional array of dependencies. The array of dependents determines when the side effect should be re-run, and the function is the side effect that needs to be executed. [17]

A sample of how to employ the `useEffect()` Hook is provided below:

```

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

function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

```

**Fig. 4.5: useEffect Example**

### **useNavigate:**

The React Router v6 module makes useNavigate available as a React Hook. You can use it to programmatically travel through your application to a different route. The applicationFor travelling to a different route inside your application, Navigate Hook offers a simple function. Similar to any other React Hook, it can be used in functional components. Using the useNavigate Hook in a React functional component is demonstrated here:[17]

```

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

function HomePage() {
  const navigate = useNavigate();

  const handleClick = () => {
    navigate('/about');
  };

  return (
    <div>
      <h1>Welcome to the Home Page!</h1>
      <button onClick={handleButtonClick}>Go to About Page</button>
    </div>
  );
}

```

**Fig. 4.6: useNavigate Example**

### **Security:**

Some pages in the project can be accessed by admin only and some of them can be accessed by normal users.[8] Data of the user is being saved in the Local Storage after logging the user into the website.

### **LocalStorage:**

In order to save data locally on the user's browser, local storage is a crucial component of web development. This makes it possible for web apps to function without an internet connection and to retain user preferences or settings even when the user signs out or closes the browser. Small bits of information, such as user preferences, settings, or authentication tokens, are often stored in local storage.

Chrome, Firefox, Safari, and Edge are just a few of the contemporary web browsers that allow local storage. The global localStorage object offered by the browser's window object can be used by your application to access local storage.

This object offers ways to add, get, and delete data from local storage.

```
> localStorage
< Storage {user: '{"data":{"user":{"_id":"6457e6adf09e42bf687b4f47",...ail.com|",|"password|":|"shaan|"}}, "request":{}}', length: 1} ⓘ
  user: '{"data":{"user":{"_id":"6457e6adf09e42bf687b4f47",|"username|":"shaanG",|"name|'
  length: 1
```

**Fig. 4.7: user info using localhost**

Local storage is only usable from the same domain that produced it, which is another restriction. This implies that a website's local storage of data prevents access to such data by another website. Cross-site scripting attacks are guarded against by this security measure.

To sum up, local storage is an effective tool for web developers that offers a practical approach to store tiny quantities of data locally on the user's browser. All current web browsers support it, and it can be used to enhance the functionality and user experience of web applications. It's crucial to recognise the limitations of local storage, though, and to make the best use of it.

### **React Router:**

A well-liked library for creating single-page applications (SPAs) with React is called React Router. It enables you to build dynamic, multi-page apps that resemble conventional webpages by handling client-side routing declaratively and effectively.

You may define the routes for your application and traverse between them using the components and functions provided by React Router. Route and BrowserRouter are React Router's two most crucial parts. The Route component specifies a route and the component that should be rendered when that route is requested, but the BrowserRouter component covers your entire application and offers the routing capability.[17]



```

import React from 'react'
import { Outlet, Navigate } from 'react-router-dom'
const LogoutPrivateRoute = () => {
  let flag;
  if (JSON.parse(localStorage.getItem("user"))) {
    if (JSON.parse(localStorage.getItem("user"))['data']['user']['role'] === "user") {
      flag = 0;
    }
    else {
      flag = 1;
    }
  }
  else {
    flag = 2;
  }
  return (
    <>
    {flag === 0 ? <Outlet /> : flag === 1 ? <Navigate to="/adminTodo" /> : <Navigate to="/" />}
    </>
  )
}
export default LogoutPrivateRoute;

```

**Fig. 4.8: React Router**

### CONCLUSION

The MERN (MongoDB, ExpressJS, ReactJS, and NodeJS) stack has skyrocketed in popularity in recent years among web developers. Due to the stack's capacity to offer a complete solution for creating scalable, secure, and quick online applications, it has grown in favour among developers. Task managers are only one of the many applications that have been built using the stack.

The creation of a task manager utilising the MERN stack was covered in this article. The task management interface in the application we created allows users to add, remove, update, and complete tasks. Along with implementing authentication using JWT tokens, we also used MongoDB to store task data.

MongoDB, Express, React, and NodeJS were some of the MERN stack's individual elements that we looked at first. We talked about how they fit into the stack and work together to create a complete web application development solution.

We then got started on the task manager application's development from the backend up. A RESTful API with endpoints for task management was built using NodeJS and ExpressJS. User data was kept in MongoDB and authentication was implemented using JWT tokens.

To construct a responsive user interface for handling tasks, we used ReactJS in the frontend development phase. To construct a dynamic user interface, we used a variety of ReactJS functionalities, including event processing, conditional rendering, and state management. Additionally, we linked the UI with the backend API to retrieve and work with task data.

Best practices for creating scalable and secure online applications were emphasised frequently during the development process. We managed dependencies using package managers like NPM and Yarn, stored sensitive data in environment variables, and included error handling to make sure everything ran smoothly.

Finally, creating a task manager using the MERN stack is a great illustration of how the stack may be used to develop a reliable and scalable web application. We have shown off the different components of the stack, such as MongoDB, ExpressJS, ReactJS, and NodeJS. Authentication implementation and data storing in MongoDB have also been demonstrated.

Overall, the MERN stack offers programmers a complete method of creating web apps. The development process could be sped up and made more effective by its flexibility, scalability, and security. The MERN stack has evolved into a crucial tool for developers aiming to create high-quality web applications in response to the growing demand for them.

A task manager is a tool that assists teams and individuals in efficiently organising and managing their tasks. It offers a centralised location where jobs may be assigned, followed up on, and ranked. Task managers are crucial for businesses for a number of reasons, including:

**Increased productivity:** Using a task manager helps teams and individuals stay on top of deadlines, prioritise tasks, and maintain organisation. Because tasks are finished quickly and the team is concentrated on the most crucial activities, productivity increases.

**Better teamwork:** Task managers enable teams to collaborate more effectively. Team members can be given tasks to do, and real-time progress can be monitored. This makes it easier for team members to communicate, share knowledge, and work towards common objectives.

**Increased accountability:** Task managers enable team members to assume ownership of their work, which increases accountability. Each team member knows their individual responsibilities when assignments are assigned to them. Team members are more likely to take their duties seriously as a result, increasing accountability.

**Enhanced communication:** Task managers give team members a forum for efficient collaboration. Real-time communication of comments, updates, and progress reports makes it simple for everyone to keep informed.

**Effective resource management:** Task managers assist businesses in effectively managing their resources. Companies can find places where resources are being squandered and take corrective action by tracking tasks.

**Better decision-making:** Task managers give businesses data that can be utilised to make better judgements, which leads to better decision-making. Companies can increase performance by monitoring progress, locating bottlenecks, and examining patterns.

For businesses trying to increase efficiency, cooperation, and communication, task managers are crucial tools. They support ensuring that resources are used effectively and that everyone is working towards shared goals. Task managers will be more crucial as businesses depend more on remote work and online collaboration to enable teams to collaborate successfully no matter where they are located.

Building web apps requires careful consideration of security, particularly when it comes to MERN (MongoDB, ExpressJS, ReactJS, and NodeJS) stack websites. The following security precautions must to be applied to MERN websites:

**Input validation:** This crucial security feature makes sure that information entered into a web application is in the right format and is free of harmful code. To make sure that input data is sanitised and doesn't include hazardous code, use client-side validation as well as server-side validation.

**Secure data storage:** Use secure data storage techniques, such as encryption, to prevent unauthorised access to sensitive data. For the protection of data, MongoDB offers encryption at rest.

**Authentication and authorization:** Implement authentication and authorisation processes to make sure that only authorised users can access sensitive data. For ExpressJS authentication and permission, use passport.js.

**Use HTTPS:** To ensure that data exchanged between the client and server is encrypted, use HTTPS rather than HTTP. HTTPS connections require SSL/TLS certificates.

**Implement rate restriction:** Limit the amount of queries a user can make in a certain period of time and avoid brute-force assaults by implementing rate limiting.

**Sanitize user input:** User input should be cleaned up in order to thwart attacks like SQL injection and others.

**Use a Content Security Policy (CSP):** Put a Content Security Policy (CSP) to use Put a CSP in place to stop injection attacks and cross-site scripting (XSS) assaults.

**Update all software:** To prevent vulnerabilities, update all software and its dependencies.

**Be cautious while using third-party libraries and packages:** Use only packages and libraries from reliable sources. Avoid utilising vulnerable or out-of-date software.

**Conduct routine security audits:** Conduct routine security audits to find vulnerabilities and proactively fix them.

In conclusion, creating secure MERN stack websites necessitates using sound coding procedures, secure data storage, authentication and authorization, and frequent security assessments. It's critical to be informed about the most recent security risks and to take precautions to defend your website from them. By putting these security measures in place, you can create secure MERN websites that safeguard user information and guarantee a dependable user experience.

## REFERENCES

- [1] Hoque, S. Full - Stack React Projects: Learn MERN Stack Development by Building Modern Web Apps Using MongoDB, Express, React, and Node.js, 2nd Edition. United Kingdom: Packt Publishing, March 2020.
- [2] Sidelnikov, G. React. Js Book: Learning React JavaScript Library from Scratch. April 2017.
- [3] Pasquali, S. Mastering Node.js. United Kingdom: Packt Publishing, October 2013.
- [4] Zammetti, F. Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker, May 2020.
- [5] REVIEW OF Saia, S. M., Nelson, N. G., Young, S. N., Parham, S., & Vandegrift, M. Ten simple rules for researchers who want to develop web apps, April 2020.
- [6] Aggarwal, S. (2018). Modern web-development using ReactJS, April-2018.
- [7] Kamthan P. Towards a systematic approach for the credibility of humancentric web applications, December 2007.
- [8] Nabi F. Internet Technology & E-Commerce. Computers and Security. 24:3. (208-217). Online publication date: 1-May-2005. The business application logic is a significant weak link in e-commerce systems, May 2005.
- [9] Kontaxis, G., Athanasopoulos, E., Portokalidis, G., and Keromytis, A. D. South: Protecting user accounts from password database leaks. In ACM Conference on Computer and Communications Security (CCS), June 2013.
- [10] Shay, R., Komanduri, S., Kelley, P. G., Leon, P. G., Mazurek, M. L., Bauer, L., Christin, N., and Cranor, L. F. Encountering stronger password requirements: user attitudes and behaviors.
- [11] Sanchit Agarwal, Jyoti Verma, "Comparative Analysis of MEAN Stack and MERN Stack", International Journal of Recent Research Aspects, March 2018.

- [12] Mohanish Bawane , Ishali Gawande , Vaishnavi Joshi , Rujuta Nikam , Prof. Sudesh A. Bachwani, “A Review on Technologies used in MERN stack”, International Journal for Research in Applied Science & Engineering Technology (IJRASET), Jan 2022.
- [13] David Gillman, Yin Lin, Bruce Maggs, Ramesh K. Sitaraman, “Protecting Websites from Attack with Secure Delivery Networks”, IEEE, April 2015.
- [14] Timothy Kelley, Bennett I. Bertenthal, “Real-World Decision Making: Logging Into Secure vs. Insecure Websites”, Springer, June 2016.
- [15] Tien Pham, “Building an online shop application with MERN stack”, Bachelor’s Thesis, November 2020.
- [16] MongoDB official website for understanding what is MERN stack: <https://www.mongodb.com/mern-stack> [Access Date: 10-02-23]
- [17] Official Documentation for ReactJS: <https://react.dev/learn> [Access Date: 13-02-23]
- [18] NodeJS official documentation: <https://nodejs.org/en/docs> [Access Date: 15-02-23]
- [19] ExpressJS official documentation: <https://expressjs.com/> [Access Date: 24-02-23]
- [20] Getting started with Mongoose: <https://mongoosejs.com/> [Access Date: 06-03-23]
- [21] Passport JS documentation for Google Login: <https://www.passportjs.org/packages/passport-auth0/> [Access Date: 19-03-23]
- [22] Using CORS for secure connection between frontend and backend: <https://developer.mozilla.org/en-US/docs/Glossary/CORS> [Access Date: 04-04-23]