

A COMPARATIVE ANALYSIS OF CODE AND NO-CODE/LOW-CODE PLATFORMS

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

in

Computer Science and Engineering/Information Technology

By

Akshat Gopal Sundriyal (191410)

Under the supervision of

**Dr. Saurabh Srivastava
Dr. Ruchi Verma**

to



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled “**A COMPARATIVE ANALYSIS OF CODE AND NO-CODE/LOW-CODE PLATFORMS**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** 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 Feb 2023 to May 2023 under the supervision of **Dr Saurabh Srivastava** , Assistant Professor (Mathematics) and **Dr Ruchi Verma**, Assistant Professor (CSE/IT).

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

Akshat Gopal Sundriyal, 191410

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

Dr Saurabh Srivastava
Assistant Professor (SG)
Mathematics
Dated:

Dr Ruchi Verma
Assistant Professor (SG)
CSE/IT
Dated:

PLAGIARISM CERTIFICATE

report

ORIGINALITY REPORT

5%

SIMILARITY INDEX

5%

INTERNET SOURCES

1%

PUBLICATIONS

%

STUDENT PAPERS

PRIMARY SOURCES

1	www.ir.juit.ac.in:8080 Internet Source	1%
2	www.coursehero.com Internet Source	1%
3	www.hfma.org Internet Source	1%
4	mafiadoc.com Internet Source	<1%
5	dzone.com Internet Source	<1%
6	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1%
7	www.businesswire.com Internet Source	<1%
8	www.timesjobs.com Internet Source	<1%
9	"Proceedings of the 8th International Conference on Advanced Intelligent Systems"	<1%

ACKNOWLEDGEMENT

I would want to take this opportunity to show my sincere gratitude to everyone who offered their unwavering cooperation and assisted me in successfully finishing my internship.

I want to start by thanking Miss Shivani Premi and Mr. Utkarsh for being my mentors, guides, and challenges. I also want to express my gratitude to my family and friends for their unwavering support while I was doing my internship.

I also appreciate the constructive criticism and assistance from my project supervisors, Drs. Saurabh Srivastava and Ruchi Verma, as well as the other staff members of the Computer Science and Engineering department at Jaypee University of Information Technology.

Akshat Gopal Sundriyal
191410

TABLE OF CONTENT

1	INTRODUCTION	1-15
	i. Introduction	
	ii. Description of Industry	
	iii. Problem Statement	
	iv. Objective	
	v. Methodology	
2	LITERATURE SURVEY	16-17
3	SYSTEM DESIGN AND DEVELOPMENT	18-34
	i. System Design Diagram	
	ii. System Design Implementation	
4	PERFORMANCE ANALYSIS & RESULTS	35-40
	i. Output using Traditional MERN Stack	
	ii. Output using NoCode Platforms	
5	CONCLUSION	41
6	REFERENCES	42

LIST OF FIGURES

S.NO	Figure Detail	Page Number
1	Company Overview	3
2	MERN Diagram	7
3	3 Tier Diagram	10
4	Appsmith Logo	13
5	Apache Hop Architecture	14
6	Frontend Design	18
7	Backend Design	19
8	Hardware Requirements	20
9	Backend Folder Structure	20
10	Frontend Folder Structure	21
11	Index File of Server	22
12	Index File of MongoDB	23
13	Input Component	24
14	Main Controller File	25
15	CSS File	26
16	Get Pipeline	27
17	Post Pipeline	28
18	Update Data Pipeline	29
19	Fetch Webservice	30
20	Update Webservice	30
21	Save data Webservice	31
22	Appsmith Dashboard	32
23	Ngrok Setup	33
24	Update API	33
25	Save API	34
26	MERN Form	35
27	Dynamic Inputs	36

28	Table with Pagination	37
29	Appsmith Form	38
30	Appsmith Table	39

ABSTRACT

This project involves a comparison between the traditional coding platforms, that is MERN stack with no-code/low-code platforms that are a buzz in the current industrial scenario. The project was assigned to me by the company in order to help me get a better understanding of the skills and the development tools that would be required in the company's live project.

We start the project with creating a dynamic data capturing platform using the conventional coding tools. We observed the advantages and disadvantages faced while making the platform according to our needs. The similar process is repeated, now using the no-code/low-code platform. It was ensured that the end product from both the techniques, should be as similar as possible.

In the end, we explored the benefits and limitations of both, MERN stack and No-code/Low-code, platforms we faced and listed them out. In the conclusion, we find that these No-code/Low-code platforms have the potential to provide a streamlined experience given more resources and better customization as well as community support.

CHAPTER 1

INTRODUCTION

1.1 Introduction

The project's goal is to assess how no-code/low-code development and conventional coding fare when it comes to platform interaction. Platforms for no-code/low-code development have become increasingly popular in the software development sector in recent years. Platforms that require little or no coding allow people to construct software applications. Instead, users can design their apps using visual interfaces and pre-built modules, greatly decreasing development time and the requirement for technical skills. Faster development cycles, lower costs, and greater development process efficiency have all contributed to this transition. Additionally, no-code/low-code platforms help companies get around the problem of the industry's lack of qualified developers. Businesses of all kinds, from small startups to major corporations, are drawn to this new trend because they want to create specialised apps fast and affordably. No-code/low-code platforms are positioned to be crucial in the software development business as the demand for customised software applications keeps rising.

No-code/low-code development platforms have become more popular recently, enabling users to construct software applications without needing to write conventional code. These platforms aim to democratise software creation by increasing accessibility while also accelerating development, cutting costs, and lowering costs. The most typical method of creating software applications is still traditional coding, which many developers still like because of its control and flexibility.

Platform integration is one area where choosing between standard coding and no-code/low-code development is very crucial. Platform integration entails linking several software programmes and services so they can operate together without any glitches. The success and effectiveness of the integration can be greatly impacted by the development method that is used for this process, which can be complicated.

On the one hand, traditional coding entails the creation of software programmes using programming languages and custom code. While traditional coding has long been the standard, it can also be a costly and time-consuming procedure. Additionally, creating custom applications is challenging for non-technical people because it requires technical knowledge. Traditional coding offers the freedom to interact with other systems as well as the capacity to construct highly sophisticated and customised applications.

For some activities, such as creating intricate algorithms, designing systems, or low-level programming, traditional coding is still required. Traditional coding, however, is frequently viewed as a bottleneck in the development process, resulting in extended development periods and higher expenses. Businesses are increasingly looking into new ways to enhance their software development processes because traditional coding is no longer the only choice for creating software applications since the introduction of no-code/low-code platforms.

1.2 Description Of Industry

Paxcom

PAXCOM provides cloud-based multichannel process management services for Commerce (hereafter referred to as “Services”).

The Services are facilitated by a team of 400+ Ecommerce enthusiasts at PAXCOM, who love to utilize their technology and knowledge for enabling Digital Commerce of products in all global markets.

Data, statistics, consultation, and marketing solutions provided by PAXCOM are trusted and implemented by some of the world's leading brands like PepsiCo, Mondelez, Britannia, Lenovo, Wipro, and Abbott for their ecommerce-driven marketing strategy.

Our principles:

1. Honesty

Nothing is more important to Paxcomer than honesty and integrity in our work, and in the people, we work with. We value authenticity and are committed to delivering real results.



Figure 1: Company Overview

2. Customer attraction

Our customers have made a significant contribution to what we are today. We work with our customers as partners who care about their success, and we work hard to meet their needs and gain their trust by making them happy.

3. Long-distance travel

When it comes to innovating, discovering new opportunities, simplifying things, and continuing to learn and improve, we think a lot! We have never learned to ignore, either to our customers or to ourselves.

4. Speed

Paxcomers believe in anticipation, planning, preparation, and action rather than simply responding to circumstances.

5. Collaboration

We all work together to achieve the same goals as a large family. We respect and trust each other, and we recognize the importance of the individual's role in our success.

6. Build a Paxcom logo

Every Paxcomer is a product representative and Paxcom agent. In all that we do, each one of us integrates and reflects the values we stand for, and we strive to advance and strengthen the brand.

7. Have fun

We believe the work can be fun and rewarding. As a result, we strive to create a work environment where Paxcomers can be fun and productive at the same time.

Paymentus

Paxcom is part of the Paymentus team - the world's leading electronic payment and payment solutions.

Paymentus (NYSE: PAY) is a leading provider of cloud-based payment technology solutions and solutions. We present our next-generation product list using a state-of-the-art technology stack for more than 1,300 bill payers across North America. Our omni-channel platform provides consumers with an easy-to-use, flexible and secure electronic payment billing experience with their favorite payment channel and brand.

Paymentus's proprietary Instant Payment Network TM, or IPN, expands our reach by connecting IPN partner forums with tens of thousands of debt collectors in our integrated payment, billing, and reconciliation capabilities.

Paymentus offers taxpayers' costs for all sizes in a different vertical industry, including services, financial services, insurance, government, communications and health care.

In 2004, Paymentus was born out of a desire to improve the way debt is repaid. Vision, innovation and exemplary service have propelled Paymentus to be the leading electronic payment solution and payment solution in the market, resulting in 1,700 customers, including some of North America's largest taxpayers.

We know that in order to keep our solutions current and relevant, we need people with knowledge, driving and the ability to promote the most exciting customer experience. Our highly dedicated, intelligent staff have transformed the vision into a secure Customer Partnership, established by SAAS and Payment Forum; one that enables direct billing organizations to provide integrated customer information and improve the acceptance of electronic payment and payment services services.

Recognized by Deloitte as one of North America's fastest-growing companies for 4 years, Paymentus consistently strives to develop better, faster, more secure, less expensive payment and payment platforms. We are always looking for the highest number of our customers, in both solutions and service.

That is what has led to our amazing growth over the past decade. We succeed when our clients are successful. They thrive when their customer relationships are improved and, 25 subsequently, their customers participate in these cost-effective electrical services at higher prices.

Paymentus differences: a payment tech that increases your compliance, effectiveness, key and customer skills

1. MOST FUNNY

We believe in configuration, not customization. Our enhanced, law-based engine solves your business needs and payment rules without customization.

2. ACKNOWLEDGMENTS

Our cloud-based platform is accessible through a host of APIs, iFrames and fully customized solutions that offer 360 degrees of control over your user experience.

3. NEXT

As a compliant provider of PCI Level I, we help businesses reduce and eliminate the burden of PCI.

4. SAFE AND RELIABLE

Our system is designed for 100% downtime and high security. Multi-layer detection and blocking system, multi-level authentication, encryption and token processing ensures the reliability and security of the transaction.

5. VERY CONNECTED

We cover more than 350 key programs, including the CIS, accounting and ERP systems, which facilitate seamless payments, reporting and reconciliation to key financial systems and operations.

6. BUILT FOR THE FUTURE

It is built from the ground up. On the basis of a single code and no version, we are able to innovate constantly and stay ahead of ever-changing needs

1.3 Problem Statement

The software development industry has been undergoing a transformation in recent years with the emergence of no-code/low-code development platforms. These platforms aim to provide a solution to the shortage of skilled developers and enable non-technical users to create complex applications quickly and easily.

However, the integration capabilities of these platforms with traditional coding approaches are still uncertain.

- This project aims to address this problem by comparing traditional coding with no-code/low-code development platforms to determine their integration capabilities.
- The project seeks to evaluate whether no-code/low-code platforms can deliver results similar to traditional coding and provide an alternative approach for software development.
- The project also aims to identify the limitations and advantages of both traditional coding and no-code/low-code platforms, thus helping developers and businesses choose the most suitable approach for their software development needs.

1.4 Objective

This project compares the MERN stack's typical coding capabilities to those of newly developing no-code and low-code development platforms, notably AppSmith and Apache Hop. The goal of the project is to determine the benefits and drawbacks of both strategies for creating a dynamic data capture platform.

- The primary goal is to evaluate whether no-code/low-code platforms can deliver results similar to traditional coding and provide an alternative approach for software development. The project will develop a Dynamic Data Capture Platform using both traditional coding with the MERN stack and no-code/low-code platforms with AppSmith and Apache Hop. This will provide a basis for comparing the development time, ease of use, and overall effectiveness of both approaches. The project will also examine the level of customization and flexibility offered by each approach, along with the ability to integrate with other systems.
- The secondary objective of this project is to determine the potential of no-code/low-code platforms, such as AppSmith and Apache Hop, to transform the software development industry by enabling non-technical users to create complex applications quickly and easily. The project will also highlight the limitations of no-code/low-code platforms and the situations where traditional coding may still be necessary.

1.5 Methodology

1.5.1 Mern stack

- What is MERN Stack, and how does it work?



Figure 2: MERN Diagram

A whole suite of open-source software tools called the MERN Stack is used to develop online applications that are scalable, high-performing, and reliable. The four main components of the stack, referred to as MERN, are MongoDB, Express.js, React, and Node.js.

A database using NoSQL known as MongoDB is made to hold data in a document-oriented fashion. Compared to conventional relational databases, it enables developers to store and manage data in a more flexible and scalable manner.

Node.js, a server-side JavaScript runtime environment, uses the web application framework Express.js to develop web apps and APIs.

React is a JavaScript library that is used to create user interfaces for web applications and is well known for being fast and having reusable code.

Node.js is a flexible technology for creating web applications since it is used for server-side scripting and enables developers to use JavaScript on the server-side.

The MEAN stack first appeared in the middle of the 2000s, as JavaScript gained popularity and single-page apps became more common. The NoSQL database MongoDB, the web application framework Express.js, the front-end JavaScript framework AngularJS, and the server-side JavaScript runtime environment Node.js made up the MEAN stack. Building scalable online applications and real-time applications was where this stack was most helpful. As we can see, Angular.js is used by MEAN's end-to-end JavaScript framework, whereas React and its ecosystem form the foundation of the MERN stack. MERN Stack was created to enable updates as easily as feasible.

- Why should we build mobile and web apps with the MERN Stack?

Because it offers a strong and flexible set of technologies to construct contemporary, scalable applications, the MERN stack is a well-liked option for developing online and mobile applications.

Search Engine Optimized: Search engine optimisation, or SEO, is the act of making a website's pages more friendly to search engines like Google, Yahoo, and others so that the material may be found by users entering relevant search terms. The objective is to increase organic traffic to

the website and the website's position in search engine results pages (SERPs). A website is inherently SEO friendly if it was created with MERN technology.

Better Performance: A website is said to perform better when the reaction time between the backend and the frontend is reduced, which leads to quicker loading times and increased speed. This makes it possible for consumers to access website material more quickly and easily, improving website performance as a whole.

Enhances Security: The security of online applications is significantly improved by MERN technology. Web application security is the use of various procedures, methods, and tools to safeguard web servers and online programmes, such as APIs, from attacks from the internet. Applications built on the MERN stack can simply be linked with secure hosting companies, increasing security. The security precautions of MERN-based apps are also strengthened by the use of security technologies like MongoDB and Node.js.

Fast Delivery: MERN Stack makes it possible for web and mobile applications to be developed and delivered more quickly, offering clients prompt service. The technological stack offers effective tools and resources that let developers create and launch apps efficiently. Businesses benefit from the time and resource savings, which gives them a competitive edge in the market.

Fast Conversion: The MERN stack technology enables online and mobile applications to be quickly converted to meet client needs. The technology stack's effective tools and resources enable developers to easily adapt and change programmes to suit the needs of certain clients. This makes it possible for companies to provide their clients better and more accommodating services.

The MERN stack's four technologies are all open source and publicly usable, giving developers important tools to address problems that can occur while working on open portfolios. In the long run, this is advantageous for developers since it makes it simple for them to acquire resources and support from a community of developers using related technologies. The MERN stack's usage of a single language allows for a seamless change from client to server, making

the development process quick and easy. It is an effective and adaptable technological stack for developing web and mobile applications since switching between client and server is simple.

- What is the structure of MERN and how does it work?

The 3-tier architectural system at MERN is composed of three layers:

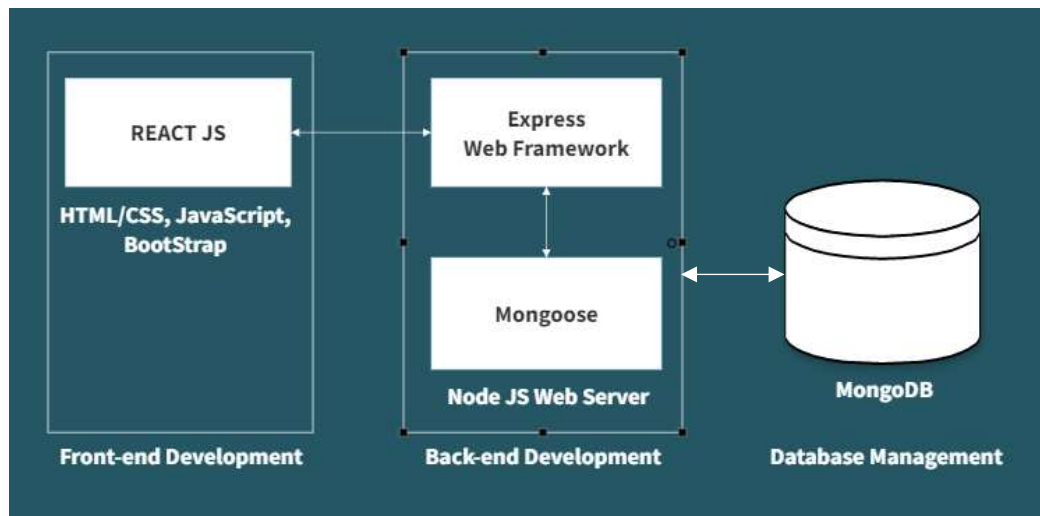


Figure 3: 3 Tier Diagram

The user interface components, such as web pages or mobile applications, are found at the uppermost layer, sometimes referred to as the Presentation layer or Client-side layer.

The middle layer, sometimes referred to as the application layer or server-side layer, is where the programme's business logic and processing elements that control and handle user requests are located.

The database or data storage components of the programme that store and retrieve application data necessary to fulfil user requests are included in the bottom layer, also referred to as the database layer or data access layer.

Let's learn about these three layers.

React.js is a crucial part of the MERN stack for the frontend layer and is frequently used to create the user interface of online applications. It is renowned for making it simple to develop powerful client-side apps. With React, developers can create intricate user interfaces with just one component and connect them to the data on the backend server. Using React Native, React can be used to create both online and mobile applications. Additionally, it encourages code reuse, which has several benefits and helps engineers save a lot of time. One of its standout benefits is the capacity to modify page data without requiring a complete page reload, enabling the development of expansive web applications with improved user interfaces.

The server is the middle tier of the MERN system's three tier design, and it is primarily managed by the MERN stack's Express.js and Node.js libraries. Because Express.js is a component of the Node.js server, these two parts operate together. Express.js is a powerful server framework that is frequently used to improve JavaScript Frameworks and provides developers with simple access to trustworthy APIs and web servers. Node.js, on the other hand, is an open-source server platform that works with JavaScript code without the requirement for a browser. JavaScript is used by Node.js to help users build a variety of network services, online applications, and mobile applications. Additionally, the server-side layer of the MERN stack's HTTP objects can benefit from Node.js' beneficial functionalities.

The database, which houses all the data related to your application, including content, analytics, user profiles, comments, and more, is an essential component of the MERN Stack. The backend of the website is mostly managed by MongoDB, which also guarantees the data's security and integrity. The database keeps an accurate and current record of the data used by the application, which is readily available to users as needed. To guarantee that users may still access the requested information even if the system malfunctions, the website stores data and makes several copies of the data files. However, because MongoDB takes a distinct approach to data storage and retrieval, it cannot be used with a website structure that is built on tables. A well-known NoSQL database, MongoDB doesn't need relational tables or a defined schema to hold data. MongoDB instead stores data in a special format without tables, rows, or columns.

1.5.2 AppSmith

Programmers can quickly create web applications without writing a lot of code using the open-source, low-code AppSmith development platform. Users can design custom UI components and workflows that match their specific needs using the platform's drag-and-drop interface. With AppSmith, users may build programmes that connect to a variety of databases, external services, and APIs.

The JSX programming language, which is built on the React framework, is used by the AppSmith platform. Users have access to a selection of UI components that are already in use and may be included in the user interface of their application. This collection contains widgets that are frequently used in web applications, such as buttons, text input fields, dropdown menus, and other widgets.

Additionally, AppSmith enables users to include a variety of data sources, including SQL databases, NoSQL databases, and APIs, into their apps. The popular databases PostgreSQL, MySQL, MongoDB, and DynamoDB, as well as the APIs offered by services like AWS, Google, and Facebook, are all supported by AppSmith.

Appsmith has several advantages, including the following:

- Improves development speed by providing pre-built UI components and a drag-and-drop interface.
- Allows easy integration with various data sources and third-party services, such as databases and APIs.
- Enhances application security through built-in security features like access control and user authentication.
- Enables collaboration among team members through version control and collaboration features.
- Provides an open-source platform that allows developers to customize the platform and contribute to its development.
- Simplifies the deployment and scaling process, reducing the need to manage infrastructure.
- Facilitates low-code development, which allows developers with little coding knowledge to build web applications quickly.

The server-client design of Appsmith uses Node.js to build the server, which offers backend services like authentication, authorization, and database connectivity. The user interface is delivered by a client that was created using React.

The platform uses an API to communicate amongst its many microservices, which divide various functions into discrete units. For instance, the database microservice manages database connectivity and query execution while the authentication microservice handles user authentication and authorisation.

In order to facilitate integration with different data sources and services, Appsmith uses a plugin architecture. Node.js-based plugins can be installed and set up by users to integrate with their applications.

The platform can be deployed on a variety of cloud platforms, including AWS, Google Cloud, and Microsoft Azure, thanks to its scalable and fault-tolerant architecture, which uses tools like Kubernetes and Docker.



Figure 4: Appsmith Logo

1.5.3 Ngrok

Using the tool Ngrok, developers may create secure tunnels between a public endpoint and a local web server. It performs the function of a reverse proxy by enabling external access to programmes that are run locally or behind a firewall.

It assigns a distinctive URL that enables visitors to the outside world to access the local application by establishing a connection between the local server and Ngrok's server. The confidentiality and privacy of the sent data are guaranteed by Ngrok's encryption.

By doing away with the requirement to deploy programs to a remote server for testing, it streamlines the development process. Developers may easily create a tunnel and choose the port on which their server is running thanks to its simple command-line interface.

1.5.4 Apache Hop

An open-source platform for data processing and integration called Apache Hop assists users in managing data pipelines for the movement and transformation of data from diverse sources. Users of the platform can drag and drop components to create data pipelines using the platform's graphical user interface. Each element carries out a certain data processing function, such as reading data from a file, filtering the data, and writing the data to a database.

It allows users to develop unique components while also providing pre-built components for readers, writers, transformations, and validators. The data formats, sources, and sinks that Apache Hop supports include databases, file systems, messaging systems, CSV, JSON, and XML.

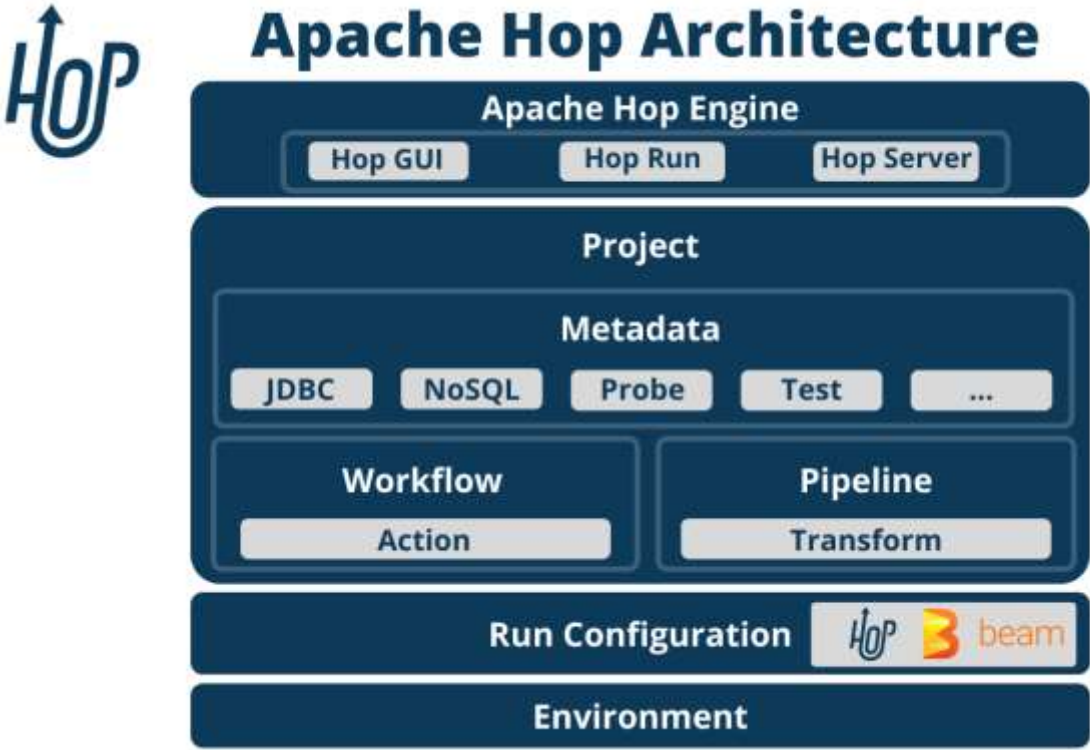


Figure 5: Apache Hop Architecture

Multiple users can work on the same data pipeline at once because to the platform's version control and collaboration features. Additionally, it provides monitoring and logging features that allow customers to keep tabs on how their data pipelines are being used and troubleshoot problems.

Each module in the architecture of Apache Hop has a particular function in the processing of the data flow. The platform is made to be scalable, effective, and extensible, and it is created using Java.

- The core module of Apache Hop manages pipeline execution, which includes pipeline scheduling, component execution, and error handling. The metadata module manages pipeline component metadata, including input and output fields, and provides an API for accessing and modifying the metadata.
- The GUI module of Apache Hop provides a graphical user interface that allows users to build and manage data pipelines through drag-and-drop components on the canvas, configure properties, and connect components to form a pipeline.
- The plugins module of Apache Hop enables users to extend the platform's functionality with custom components. This allows users to create their components using Java or other programming languages and integrate them into their pipelines.
- The hop-server module of Apache Hop provides a REST API for remotely executing and monitoring data pipelines through a web-based interface. This allows users to schedule pipeline execution, monitor pipeline progress, and retrieve pipeline results.

CHAPTER 2

LITERATURE SURVEY

2.1 React Documentation

The React documentation is a thorough manual offered by the React team that provides developers using the React library with in-depth explanations, examples, and directions. It provides code snippets and best practices to help with building effective React code, covering all crucial topics like components, state management, hooks, and more.

The documentation is frequently updated and contains details on the most recent releases as well as migration instructions. It is a priceless tool that gives both novice and seasoned developers the information and direction they need to create effective React apps.

2.2 Javascript Documentation

Developers working with the JavaScript programming language can find thorough instructions, explanations, and examples in the JavaScript documentation. It covers a variety of JavaScript features, including syntax, data types, functions, objects, and built-in APIs. As a reference guide, this documentation provides thorough explanations of JavaScript features, functions, and properties.

Additionally, it provides information on browser compatibility, allowing programmers to confirm that their code runs properly on various web browsers. New language features and standards are continually incorporated into the JavaScript documentation. Developers of all skill levels can use it as a useful tool to help them understand and utilize JavaScript's full capabilities in their projects.

2.3 MongoDB Documentation

MongoDB uses its library - mongoose to make the code more readable, concise and efficient. Mongoose is used widely to enhance the code readability.

2.5 Express Js Documentation

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Routing, middleware, request handling, template engines, and other subjects are covered.

2.4 Appsmith Documentation

Companies create in-house applications, referred to as admin tools or internal tools, to aid their teams in carrying out specific tasks such as managing dashboards, handling databases, and facilitating customer support. Appsmith, an open-source developer tool, streamlines the swift development of such internal tools. Its drag-and-drop feature enables the construction of user interfaces on a grid-style canvas, while facilitating seamless integration between the UI and data sources to enhance application development. Additionally, Appsmith supports JavaScript within widgets, queries, and various components, allowing for the inclusion of logic, data transformation, and the creation of intricate workflows.

2.5 Apache Hop Docs

The documentation for the Apache Hop data integration platform is a thorough and in-depth source that provides advice and instructions on utilizing it. It addresses a number of Hop-related topics, including as installation, configuration, transformation development, workflow construction, and deployment, and it gives developers the knowledge they need to properly utilize the platform's capabilities and functionalities.

CHAPTER 3

SYSTEM DESIGN AND DEVELOPMENT

3.1 System Design Diagram

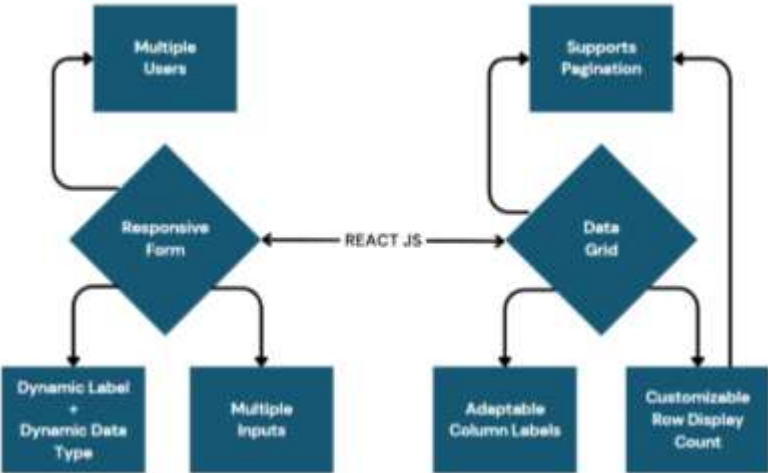


Figure 6: Frontend Design

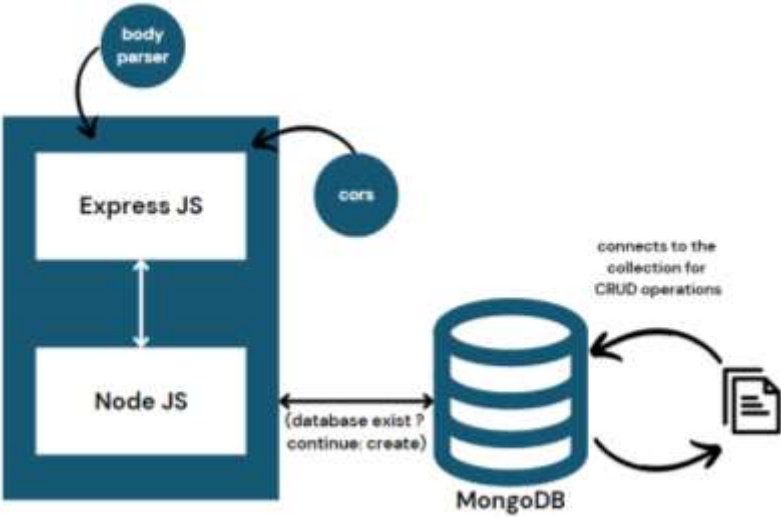


Figure 7: Backend Design

3.2 System Design Implementation

Identification of features:

- Creation of an adaptive form incorporating versatile fields capable of accommodating dynamic data types.
- A single form can encompass numerous dynamic fields.
- Simultaneous inclusion of multiple forms is feasible.
- Displaying a dynamic Data Grid upon request.
- The Data Grid incorporates flexible column labels and the ability to customize the number of rows to be displayed.
- The Data Grid page seamlessly implements pagination based on the specified row display count.
- Creation of Database in the event of nonexistence, followed by seamless retrieval of the collection to enable comprehensive CRUD operations.
- Creation of API end points to interact between frontend and database of our application.

Hardware configuration:

Device name	DESKTOP-JIQ7JB8
Processor	Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
Installed RAM	8.00 GB (7.88 GB usable)

Figure 8: Hardware Requirements

Implementation using traditional coding (MERN Stack):

Folder hierarchy and workspace:

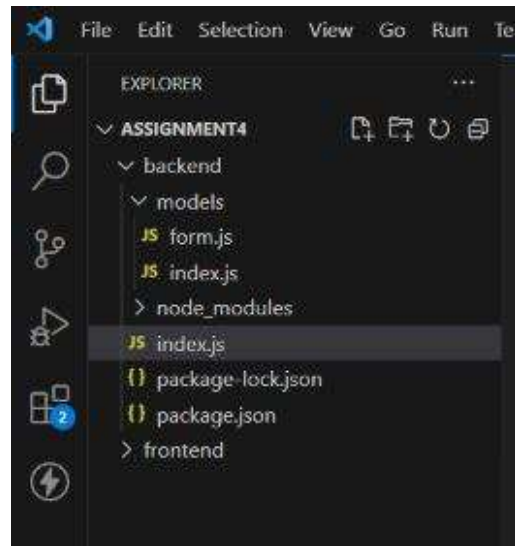


Figure 9: Backend Folder Structure

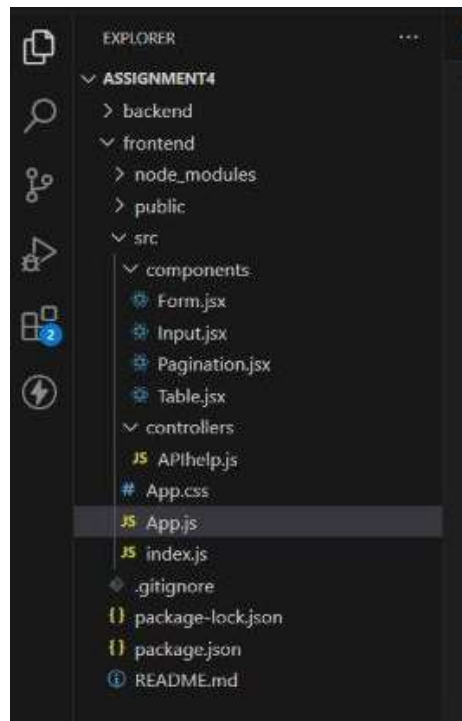


Figure 10: Frontend Folder Structure

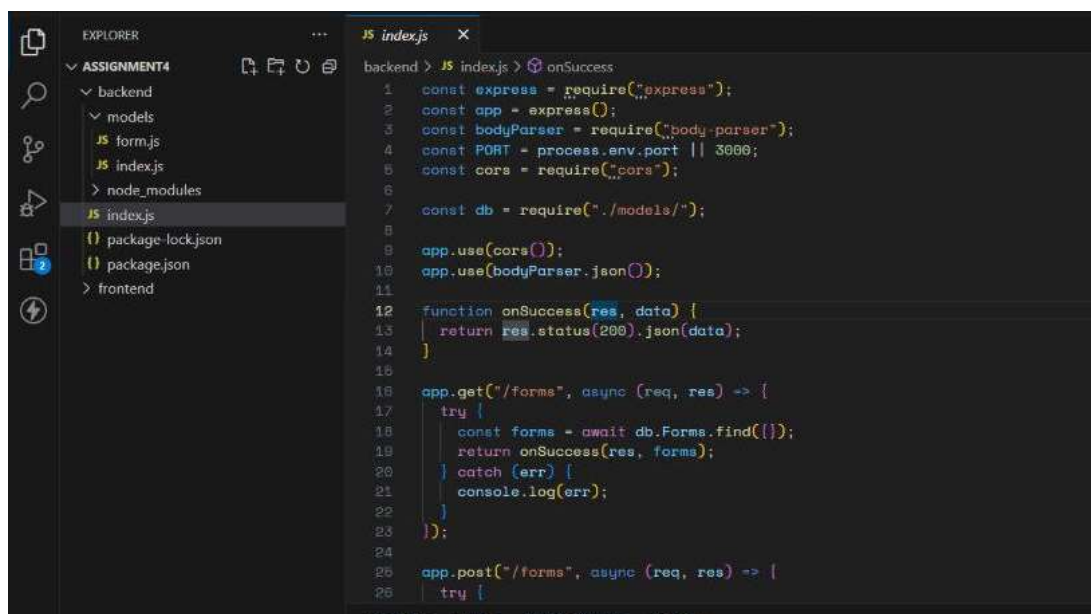
Backend Implementation:

Libraries and Tools:

- Express.js: Web application framework for handling routing, middleware, and server logic.
- Mongoose: MongoDB object modeling tool for schema creation and database interaction.
- Body-parser: Middleware for parsing request bodies.
- CORS: Middleware for handling Cross-Origin Resource Sharing.

Initialization of primary index file:

The project's primary file, `index.js`, serves as the starting point for a Node.js application. It uses essential elements for handling HTTP requests, including Express.js, body-parser, and CORS middleware. The programme is set up to listen on port 3000 so that it can take inbound queries.



```
backend > JS index.js > onSuccess
1  const express = require("express");
2  const app = express();
3  const bodyParser = require("body-parser");
4  const PORT = process.env.port || 3000;
5  const cors = require("cors");
6
7  const db = require("../models/");
8
9  app.use(cors());
10 app.use(bodyParser.json());
11
12 function onSuccess(res, data) {
13   return res.status(200).json(data);
14 }
15
16 app.get("/forms", async (req, res) => {
17   try {
18     const forms = await db.Forms.find({});
19     return onSuccess(res, forms);
20   } catch (err) {
21     console.log(err);
22   }
23 });
24
25 app.post("/forms", async (req, res) => {
26   try {
```

Figure 11: Index File of Server

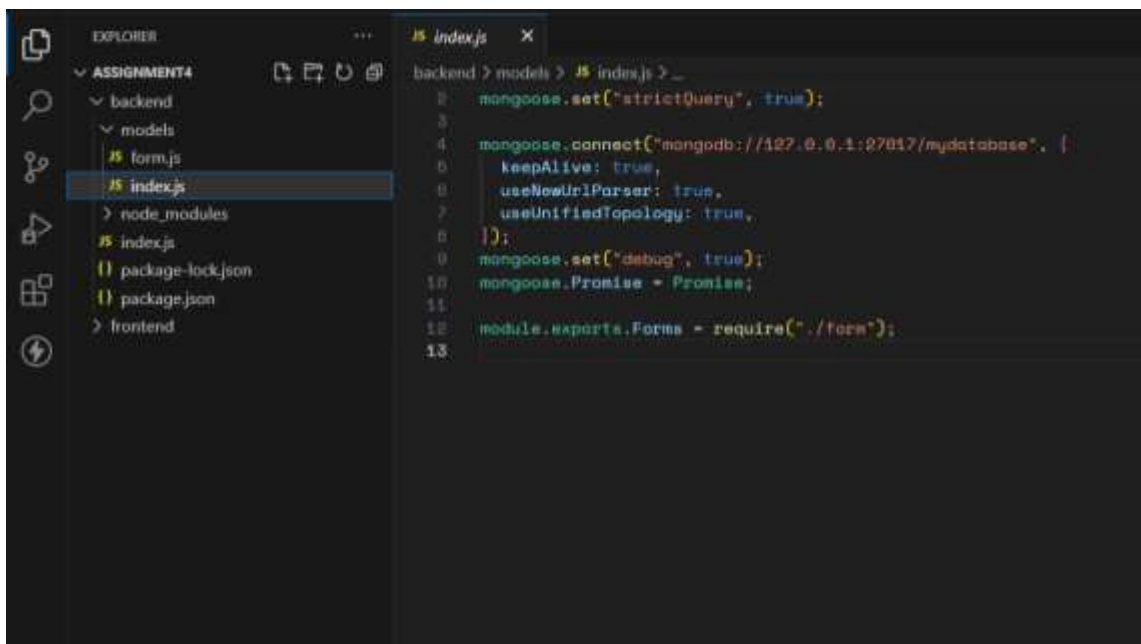
Three important API endpoints—GET, POST—have been defined in the file. When utilising the imported Mongoose model as the interface for dealing with the MongoDB database, these endpoints are in charge of carrying out specified activities.

- The **GET** endpoint utilizes a route handler to retrieve data from the database using the Mongoose model and sends it back as a response to the client.

- An instance of the Mongoose model is created with the supplied data in the case of a **POST** request, and it is saved in the database by the route handler after being extracted from the request body.
- Users can delete a particular resource by its ID using the **POST** endpoint itself. The route handler uses the Mongoose model to locate and remove the appropriate resource, returning a response to show that the operation was successful.

Creation of Model Repository:

The data models are kept in a single location, folder named models. It is made up of several files, including an index file for utilising the Mongoose library to connect to the MongoDB database. The schemaless design of the form using Mongoose is defined in another file located in the models folder. The structure and validation guidelines for the form data are contained in this file.



```
backend > models > .js index.js > ...
1  mongoose.set("strictQuery", true);
2
3
4  mongoose.connect("mongodb://127.0.0.1:27017/mydatabase", {
5    keepAlive: true,
6    useNewUrlParser: true,
7    useUnifiedTopology: true,
8  });
9  mongoose.set("debug", true);
10 mongoose.Promise = Promise;
11
12 module.exports.Forms = require("../form");
13
```

Figure 12: Index File of MongoDB

This particular model is accessible for use in the project's server component due to the main index file of the models folder, which exports it.

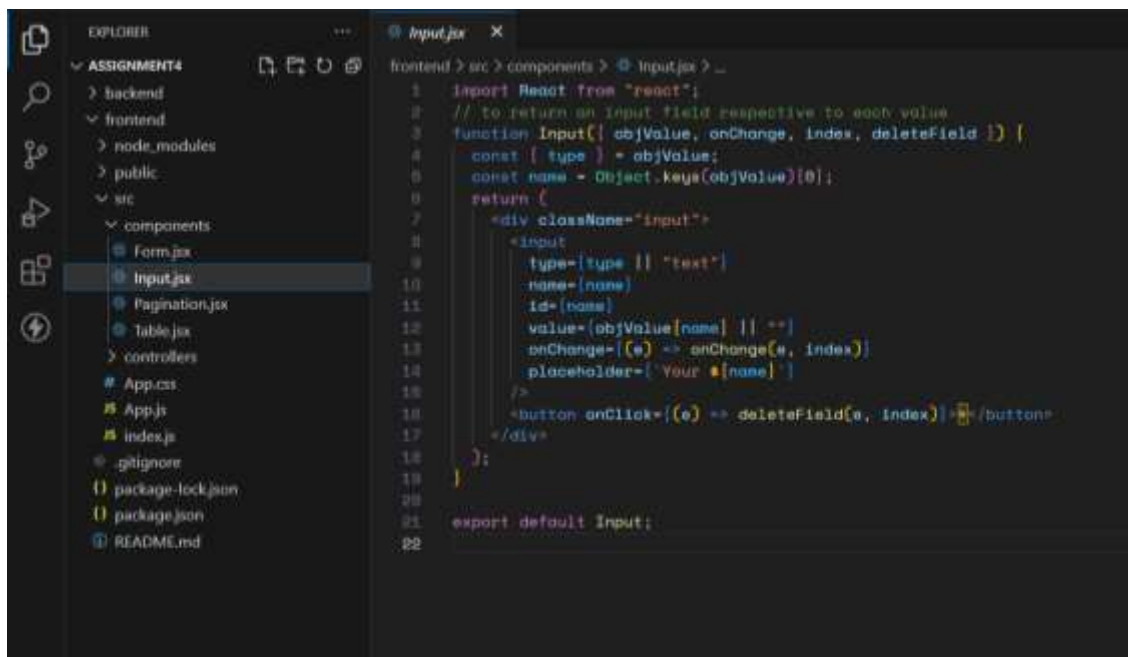
Frontend Implementation:

Creation of Source Folder:

The src folder, serving as the core directory of a React application, encompasses a diverse range of subfolders and files essential to its overall functionality. Within this folder, the components subfolder is of importance, containing crucial reusable components pivotal to the application's operations.

Components Folder:

- The Form component emerges as a pivotal entity responsible for rendering and managing user input within an intelligently structured form. It not only establishes an aesthetically pleasing layout for capturing data but also incorporates robust validation mechanisms to ensure data integrity.
- The Input component plays a pivotal role in the user interface by enabling the creation of input fields where users can conveniently furnish their data. Armed with a multitude of features, including placeholder text, input type validation, and meticulous error handling, it provides a seamless user experience and augments data accuracy.



```
EXPLORER
  ASSIGNMENT4
    backend
    frontend
      node_modules
      public
      src
        components
          Form.jsx
          Input.jsx
          Pagination.jsx
          Table.jsx
        controllers
      App.css
      App.js
      index.js
      .gitignore
      package-lock.json
      package.json
      README.md

input.jsx
  1  import React from "react";
  2  // to return an input field respective to each value
  3  function Input({ objValue, onChange, index, deleteField }) {
  4    const [ type ] = objValue;
  5    const name = Object.keys(objValue)[0];
  6    return (
  7      <div className="input">
  8        <input
  9          type={type || "text"}
 10          name={name}
 11          id={name}
 12          value={objValue[name] || ""}
 13          onChange={e => onChange(e, index)}
 14          placeholder={`Your ${name} `}
 15        />
 16      <button onClick={e => deleteField(e, index)}>✖</button>
 17    </div>
 18  );
 19  }
 20
 21  export default Input;
 22
```

Figure 13: Input Component

- The Table component, a formidable asset within the system, possesses the ability to showcase tabular data in a highly structured and visually pleasing manner. Offering an array of advanced functionalities such it empowers users to efficiently analyze and interact with data, thereby elevating the overall data visualization experience.
- The Table component, a fundamental building block of the application, showcases data in a tabular format with exceptional flexibility. It not only allows for the customization of column labels, providing a user-friendly interface that adapts to varying data sets, but also grants users the ability to personalize the number of rows displayed at a time. By implementing pagination functionality based on the specified row display count, the Table component ensures efficient navigation through large datasets.
- The Pagination component, an integral part of the system, serves the purpose of presenting data in a segmented and organized fashion. By dividing large data sets into manageable pages, it empowers users to effortlessly navigate through different sections.

Controllers Folder:

The folder serves as a crucial part of the application's backend structure. It contains a JavaScript file that acts as a bridge between the frontend and the backend server. This file utilizes the axios library to facilitate communication and data exchange between the two components.

```

1  import axios from "axios";
2  const URL = "http://localhost:3000/Posts/";
3
4  async function createData(values) {
5    let task = [];
6    // For posting the data, it
7    // iterates over each formData and makes sure a proper format is
8    // passed in the database
9    values.forEach((element) => {
10     const val = Object.keys(element)[0];
11     Object.assign(task, {
12       [val]: element[val],
13     });
14   });
15 }
16
17 const { data: newValues } = await axios.post(URL, {
18   formData: task,
19 });
20 return newValues.formData;
21 }
22
23 async function getAllData() {
24   let result = [];
25   const { data: values } = await axios.get(URL);
26   // on fetching the database, it only returns the formData
27   // from each value
28   for (let idx in values) {
29     result.push(values[idx].formData);
30   }
31   return result;
32 }
33
34 export { createData, getAllData };

```

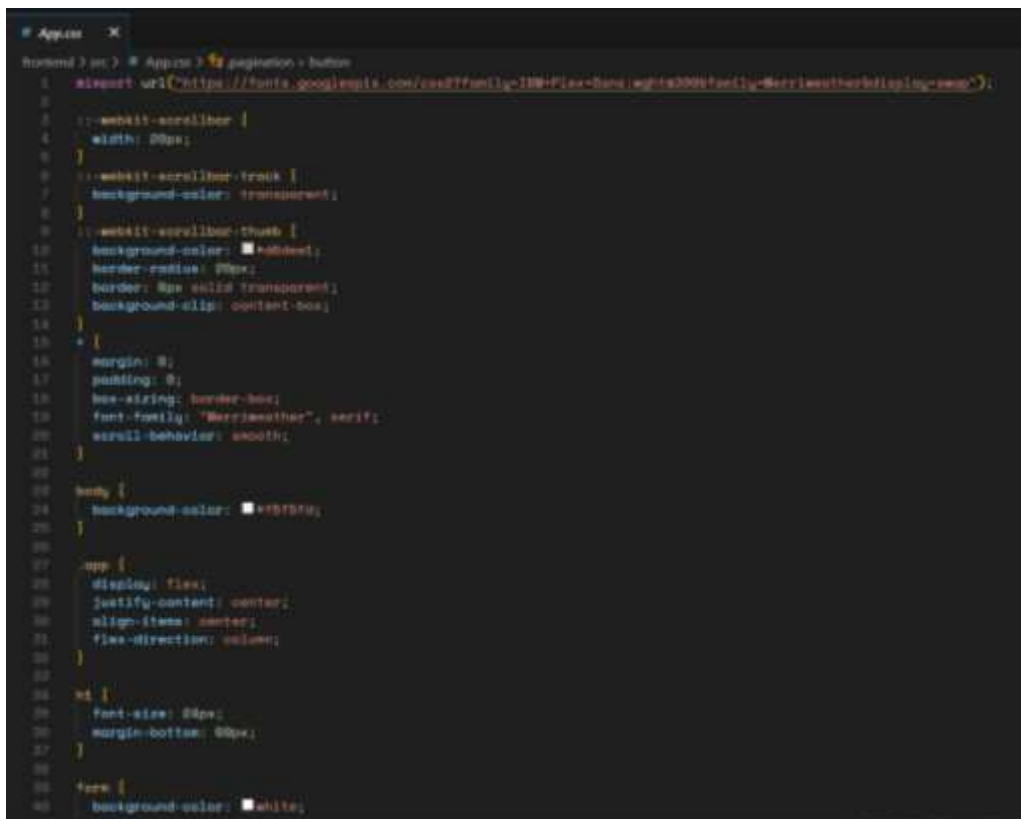

Figure 14: Main Controller File

The App.js file is a central component in the application's frontend structure. It serves as the main container where different components and functionalities are brought together to create a dynamic and interactive user interface. It orchestrates the rendering of various elements, such as forms, inputs, pagination, and tables, allowing users to capture and manipulate data

Styling the Application:

The styling of the application is done using CSS. The use of CSS for styling has a number of benefits. The presentation can be easily separated from the underlying HTML structure, to start. This makes a website more adaptable and maintainable since changes to the visual appearance may be made without affecting its structure or content.

All major web browsers accept CSS, making it a dependable and widely used standard for web styling. This guarantees predictable and consistent style rendering across many devices and browsers.



```
1 <script src="https://fonts.googleapis.com/css?family=IBM+Plex+Sans|Roboto+Mono|Merriweather|Play=web"></script>
2
3 ::-webkit-scrollbar {
4   width: 20px;
5 }
6 ::-webkit-scrollbar-track {
7   background-color: transparent;
8 }
9 ::-webkit-scrollbar-thumb {
10  background-color: #444;
11  border-radius: 10px;
12  border: 3px solid transparent;
13  background-clip: content-box;
14 }
15 * {
16   margin: 0;
17   padding: 0;
18   box-sizing: border-box;
19   font-family: "Merriweather", serif;
20   scroll-behavior: smooth;
21 }
22
23 body {
24   background-color: #f1f3f4;
25 }
26
27 .app {
28   display: flex;
29   justify-content: center;
30   align-items: center;
31   flex-direction: column;
32 }
33
34 h4 {
35   font-size: 24px;
36   margin-bottom: 20px;
37 }
38
39 html {
40   background-color: #white;
```

Figure 15: CSS File

Implementation using No Code/ Low Code Platforms:

Backend Implementation Using Apache Hop:

Creation of Pipelines:

For the project, pipelines have been built utilising Apache Hop to carry out various data processes. These pipelines, which go by the names getData, postData, and updateData, each play a particular part in how data is handled by the application.

- The primary goal of the getData pipeline is to extract data from the MongoDB database. It processes data from the database as input to produce a JSON output. This pipeline plays a key role in extracting data and structuring it so that it can be used or processed further by other application components.

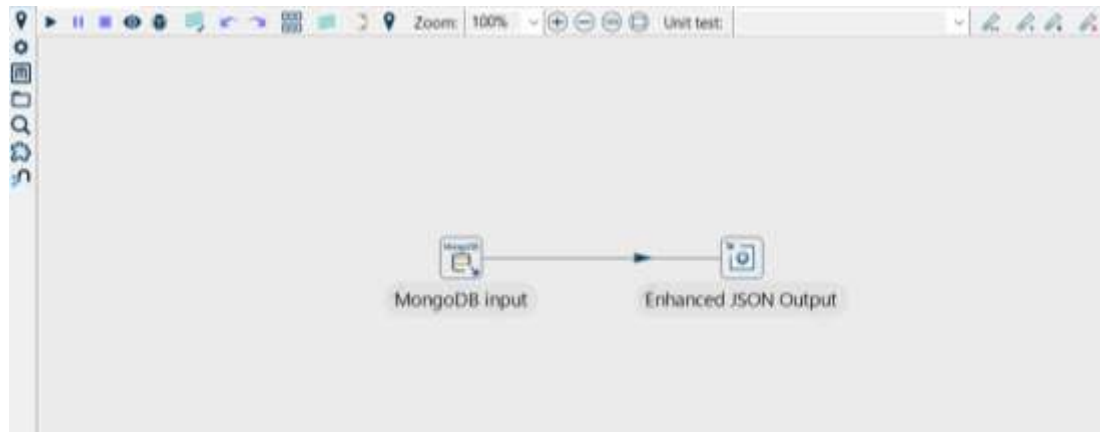


Figure 16: Get Pipeline

Extracting information from a MongoDB collection is the responsibility of the "Mongo Input" component. It creates a connection to the MongoDB server and retrieves the pertinent information based on the parameters you specify, like queries or filters. The subsequent pipeline component receives the obtained data after that.

The "Enhanced JSON output" component, on the other hand, uses the input data from the "Mongo Input" component and adds further processing to create an array called "result." This array represents a modified version of the input data that includes certain improvements in accordance with pipeline specifications.

- HTTP POST requests are handled by the postData pipeline. Data from the request body is taken, properly formatted, and checked to make sure it is in the right format for MongoDB database storage. This pipeline is crucial in maintaining fresh data given via POST requests, which enables the application to successfully store and handle user-generated content.



Figure 17: Post Pipeline

A body variable from the request supplied by the browser is received by the "Get Variables" component. The data in this body variable needs to be handled by the pipeline. This information is retrieved by the component and sent to the pipeline's following stage. The "Enhanced JSON Output" component does the required changes to put the input data that it receives from the "Get Variables" component into the proper format. The data may need to be formatted in accordance with predetermined criteria, certain fields may need to be filtered, or data modifications may be applied.

The "Mongo Output" component, which connects to the MongoDB database, receives the processed data at the end. The component then stores the data by inserting it into the proper MongoDB collection.

- The updateData pipeline is in charge of updating current data in the MongoDB database. Data from an HTTP request is received, formatted in accordance with the necessary update criteria, and compared to the pertinent data in the database. The pipeline, in addition to updating data, also provides the functionality to delete files using the "isDeleted" boolean attribute present in each data entry. When triggered by an HTTP POST request by a discard button, this pipeline identifies the specified data entry, marks it as deleted by setting the

"isDeleted" boolean to true, and updates the corresponding record in the MongoDB database.

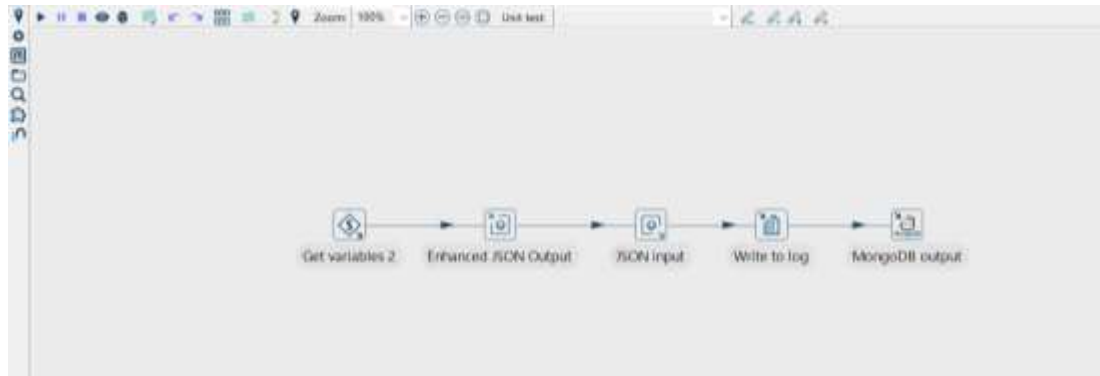


Figure 18: Update Data Pipeline

The body variable from the request sent by the browser must be obtained via the "Get .Variables" component. The data in this body variable needs to be handled by the pipeline. The component gathers the necessary variables and sends them on to the pipeline's following step.

The "JSON Input" component does the necessary transformations to guarantee that the input data is in the proper JSON format after receiving it from the "Get Variables" component. Depending on the demands, it can entail parsing and reformatting the JSON structure.

To the "Mongo Output" component is where the processed data is transferred. In this instance, the component is set up to update the data in the MongoDB database because the "update data" checkbox is selected. The component locates the corresponding data in the database by using a matching key that was chosen using a set of criteria. The appropriate modifications are subsequently applied to the MongoDB collection's matching records.

Creating API Endpoints:

A part of the Apache Hop framework called Apache Hop Webservice Metadata is used to specify and set up web services within a project. It offers a methodical way to handle the particulars and preferences pertaining to the creation, deployment, and use of web services. Users and outside

systems can communicate with the application's data features through these endpoints. It includes various options like:

- Name - The name of the web service. This is the name that is passed into the webService URL.
- Enabled - Enables or disabled the web service
- Filename on the server - This is the filename on the server. Make sure that the pipeline you want to execute is available on the server.
- Output transform - The name of the transform from which this service will take the output row(s).
- Output field - The output field from which this service will take data from, convert it to a String and output it
- Content type - The content type which will get reported by the webService servlet
- List status on server - Enable this option if you want the executions of the web service pipeline to be listed in the status of the server.
- Request body content variable - This is the name of the variable that, when it is being used in a programme, will hold the request body content. When making a POST request to the webservice, this is helpful.

Following API endpoints are created using the Webservice Metadata

- The fetch pipeline serves as an API endpoint for retrieving data from the MongoDB database. It responds to HTTP GET requests and returns the data in JSON format.



Figure 19: Fetch Webservice

- The updateData pipelines function as API endpoints for submitting new data and updating existing data in the MongoDB database. They handle HTTP POST requests, and process the incoming data accordingly.



Figure 20: Update Webservice

- The postData pipeline is responsible for processing incoming data sent through an HTTP POST request. This pipeline takes the data from the request's body, formats it according to the desired structure, and then inserts it into the MongoDB database.



Figure 21: Save data Webservice

By integrating these pipelines with Apache Hop Server, the project can easily create API endpoints for data retrieval, submission, and updates.

Frontend Implementation using AppSmith:

Creation of Landing Page:

First, we create a new application from our dashboard. Then we create the main page of our application. There is a container that contains a table on this page.

The table's columns have headers like Name, Phone, Address, and Age to give the data a structured look. A fetch API call is conducted to get the required information that will be displayed on the website before rendering the table. By doing this, it is ensured that the table has the pertinent data that was retrieved from the backend of our application.

The table includes particular features, such as the "Update" section. An "Update Data" button is included in this area, and clicking it causes the associated update API to run. The most recent data is then refreshed and obtained through a fetch API request, guaranteeing that the updated information is appropriately shown.

Using the "Discard" button on the table, values can be softly deleted from the database. The application changes the appropriate data entry's "isDeleted" field to true and marks it for deletion when the discard button is pressed. After that update API is called and the above process is repeated to ensure refreshed data is shown.

The user is directed to the form page via a button on the website. This button offers a simple way to move around the application's components.

Form Page:

A container component that acts as a form for data entry is present on the page. The container originally has a default field with the label "Name." On the other hand, the user has the freedom to add more dynamic input fields in accordance with their needs. There is a delete option next to each of these fields, allowing the user to eliminate any blank fields.

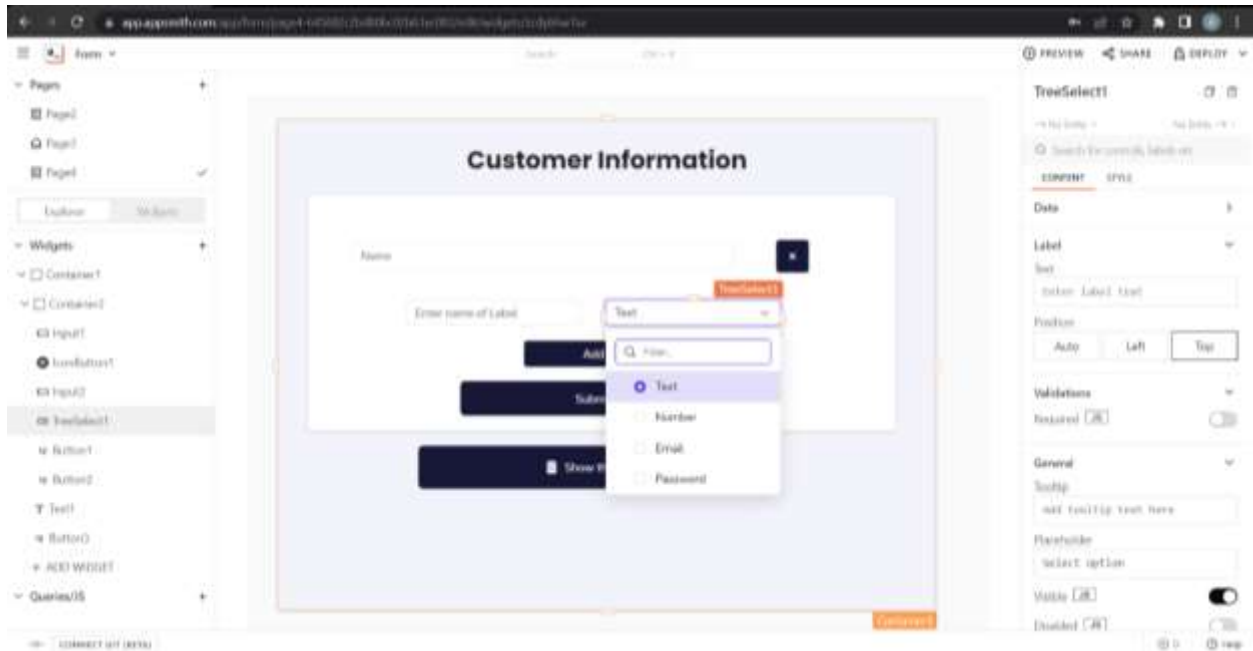


Figure 22: Appsmith Dashboard

The page has a submit button, which is inactive until all the input fields are filled in, to protect the integrity of the data. The user can submit the form after filling it out with the necessary data. A post API call is started when the button is clicked, sending the form's data to the database for storage. The form is then reset, enabling the user to enter fresh data without the influence of earlier inputs.

Integrating Backend with Frontend:

Data pipeline integration into the Appsmith application is made simple by integrating Apache Hop with Ngrok. To enable external access to the locally hosted Apache Hop server, Ngrok functions as a tunnelling service.

Ngrok is set up to expose the Apache Hop server to the internet in order to establish the integration. Ngrok creates a public URL that routes incoming requests to the local server by providing the local port of the Apache Hop server.

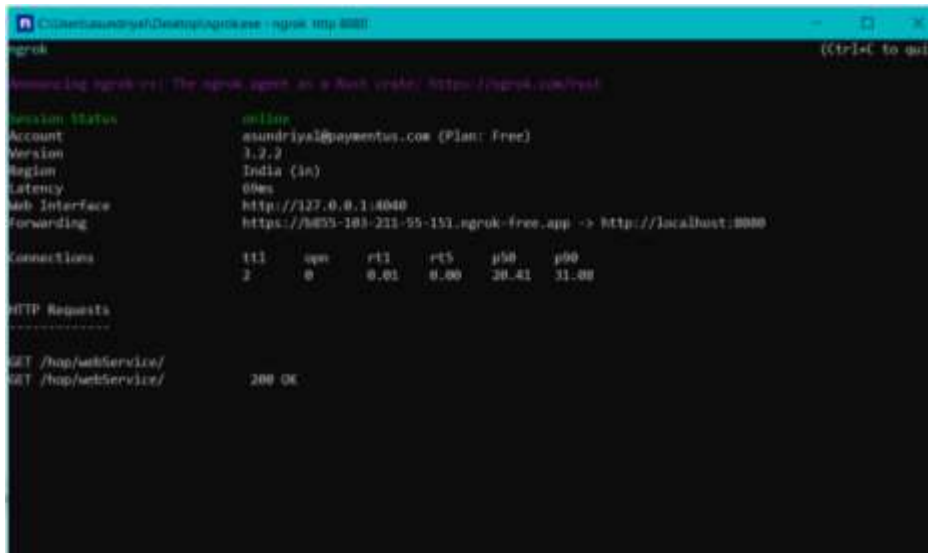


Figure 23: Ngrok Setup

The configuration allows for the integration of Apache Hop with Appsmith.

Creation of various APIs in AppSmith:



Figure 24: Update API



Figure 25: Save API

CHAPTER 4

PERFORMANCE ANALYSIS & RESULTS

4.1 Output Using Traditional Mern Stack:

A highly flexible and interactive online application was produced by using the MERN (MongoDB, Express.js, React, Node.js) stack to create dynamic data capture platform with data presentation features.

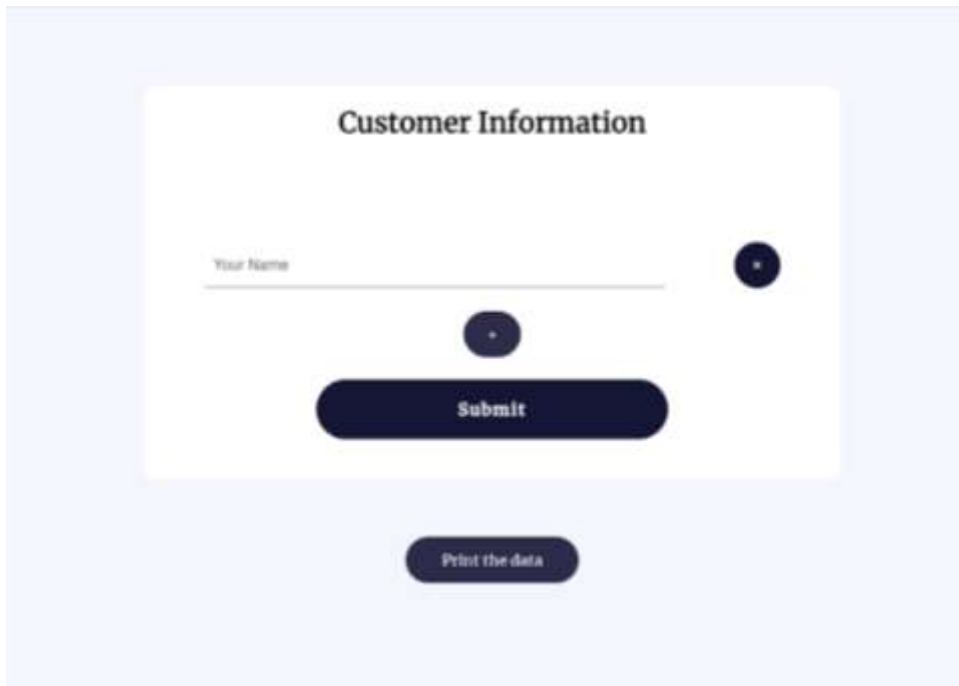


Figure 26: MERN Form

React, a flexible JavaScript toolkit renowned for its component-based architecture, was used to construct the frontend. The virtual DOM and effective rendering of React allowed for the creation of reusable and modular components for the dynamic form. The building of logical input fields, dropdown menus, and buttons was possible thanks to the ease with which the frontend development route could be navigated. The form validation, data manipulation, and submission processes were made easier using React's component lifecycle functions, enabling a responsive and engaging interface.

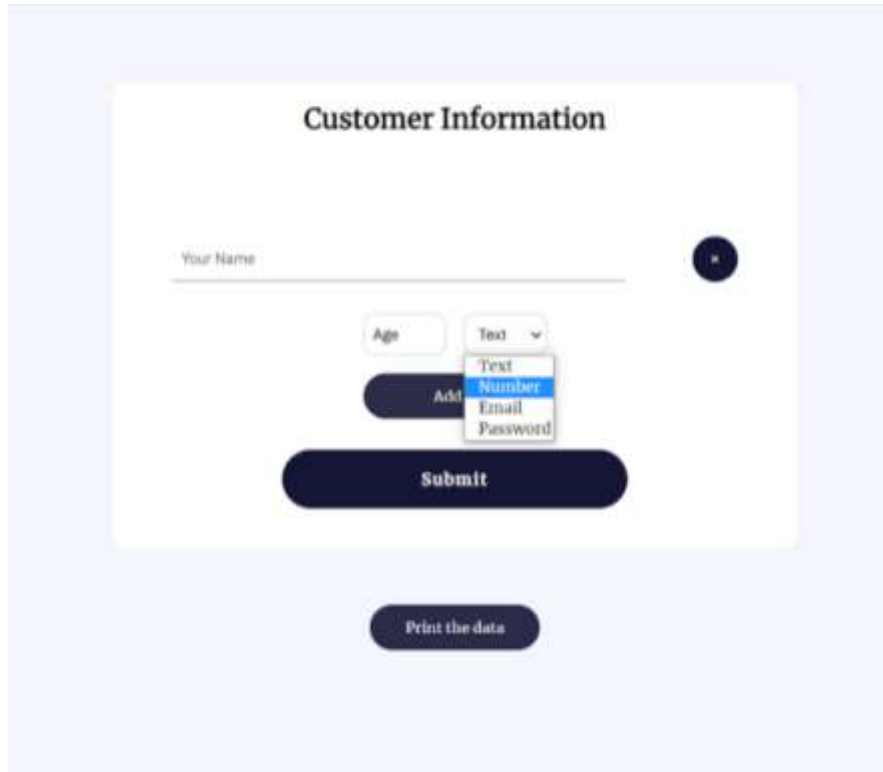


Figure 27: Dynamic Inputs

Express.js was the backend framework used to manage HTTP requests and create API endpoints. Due to its flexible and basic design, routing and middleware configuration were made easier while the frontend and MongoDB database were smoothly integrated. A scalable environment for processing form data and assuring high-performance execution was given by the backend, which was driven by Node.js.

The effective storing and retrieval of form data was made possible by the integration with MongoDB, a NoSQL database. The dynamic nature of the form combined with MongoDB's flexibility in its schema and document structure allowed for simple adaptation to meet changing data needs.

Select Number of Rows Per Page

15

Index	Name	Age	Phone	Address
1	Navya Yadav	19	9977114488	LKO
2	Navya Yadav	19	9977114488	LKO
3	Navya Yadav	19	9977114488	LKO
4	Navya Yadav	19	9977114488	LKO
5	Navya Yadav	19	9977114488	LKO
6	Akshat Gopal	19	7722549315	New Delhi
7	Akshat Gopal	19	7722549315	New Delhi
8	Akshat Gopal	19	7722549315	New Delhi
9	Akshat Gopal	19	7722549315	New Delhi
10	Akshat Gopal	19	7722549315	New Delhi
11	Akshat Gopal	19	7722549315	New Delhi
12	Akshat Gopal	19	7722549315	New Delhi

Prev 1 Next

Figure 28: Table with Pagination

Overview of MERN Stack:

- **Complete Customization and Control:** The MERN stack gives developers total flexibility and control over the whole development process. Developers have the freedom to plan and construct any component of the application to meet their unique needs, from the frontend using React through the backend with Express.js and Node.js. High-end, one-of-a-kind solutions can be developed with this degree of control.
- **Huge Community and Ecosystem:** The MERN stack has a sizable and active developer community, which means there are a tonne of tools, guides, and open-source libraries accessible. Rapid development and problem-solving are made possible by MERN's broad ecosystem since developers can use already-developed solutions and draw on community knowledge. This network of support speeds up development and increases productivity.
- **Development Time and work:** Compared to low-code platforms like Appsmith and Apache Hop, building a project from scratch using the MERN stack takes more time and work. Although the MERN stack allows for customization and flexibility, it also necessitates manual coding and configuration, which can lengthen the development cycle.

4.2 Output Using Nocode/Low Code Platform:

Using AppSmith and Apache Hop, a dynamic data capture platform that can display data was successfully created. The project's goal was to use the low-code platforms' capabilities to build an extremely flexible and interactive form while minimize the manual coding work and complexity generally associated with conventional development methods.

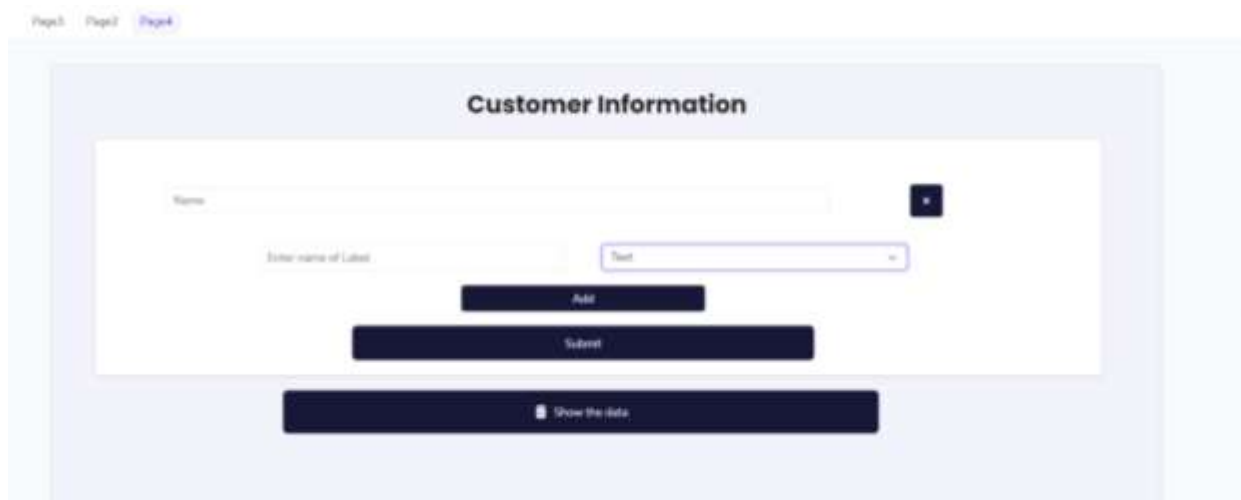
The image shows a web form titled "Customer Information" within a browser window. The form has a white background and is set against a light blue border. At the top, there are three tabs labeled "Page 1", "Page 2", and "Page 4". The form contains a text input field for "Name" with a small blue square icon to its right. Below this is a label "Enter name of Label" followed by a dropdown menu currently showing "Text". Underneath the dropdown are three dark blue buttons: "Add", "Submit", and "Show the data".

Figure 29: Appsmith Form

The frontend components of the dynamic form were designed and implemented using an easy-to-use visual interface offered by AppSmith, a potent low-code platform. It made it simple and quick to create form elements like input fields, dropdown menus, and checkboxes because to its drag-and-drop capabilities and pre-built UI widgets. The visual editor in AppSmith made it simple to set up the form's layout, design, and validation guidelines, removing the need for manual code and shortening the development cycle.

The backend implementation of the dynamic form relied heavily on Apache Hop, a no-code data integration platform. It supported smooth integration with the form and allowed for the extraction, transformation, and loading (ETL) of data from diverse sources. The visual workflow designer in Apache Hop made it possible to build data pipelines for the structured transformation, aggregation, and storage of form submission data.

Data displayed in a Tabular format

Name	Phone	Address	Age	Update	Save / Discard
Manav	8865238754	agra	78	Make Changes	Discard
ranj	8788998212	delhi	25	Make Changes	Discard
Nave	5678456588	delhi	22	Make Changes	Discard
ran	1245784512	del	21	Make Changes	Discard
Aks	6588784512	del	22	Make Changes	Discard
Ashut	7845128754	Delhi	22	Make Changes	Discard

[Go to Form](#)

Figure 30: Appsmith Table

Thanks to Apache Hop’s ETL capabilities, the data that was submitted through the form was quickly processed and saved in a backend database. The user-friendly data components in AppSmith were then used to dynamically display the retrieved information.

The simplicity and quickness of development made using AppSmith with Apache Hop advantageous. Due to the low-code nature of these platforms, substantial manual coding was not necessary, allowing the project to go quickly. Without extensive knowledge of programming languages or intricate backend infrastructure, developers were able to create and configure the form and the accompanying data processing workflows thanks to the visual interfaces offered by both platforms.

Overview of Apache Hop and AppSmith:

- **Rapid Development:** By reducing the need for human coding, AppSmith and Apache Hop offer a low-code/no-code environment that speeds up development. To design and create the dynamic form rapidly, developers can make use of pre-built components, drag-and-drop capability, and visual interfaces. This greatly cuts down on development time and speeds up prototyping and iteration.
- **Simplified Integration:** Apache Hop and AppSmith provide simple connectivity with a range of data sources and APIs. Processes like data extraction, processing, and loading are made easier by the built-in connectors and adapters they offer. This makes it easier to

integrate the dynamic form with databases, backend systems, and outside services. The simplified integration possibilities get rid of manual configuration and minimize mistakes.

- **Low Resources:** When compared to well-known frameworks and stacks like the MERN stack, low-code platforms like AppSmith and Apache Hop have less documentation and community support, which could be a drawback. These platforms frequently have a smaller user base, which results in less resources, tutorials, and community forums available to handle complicated problems or offer thorough help. Because of the lack of thorough documentation, troubleshooting and problem-solving procedures may take longer, which could affect project completion dates and overall development effectiveness. Furthermore, there may be fewer community-driven plugins, extensions, or integrations available, necessitating more bespoke development or workarounds.
- Low-code platforms may offer a vast choice of pre-built components and functionality but may be limited in their ability to be customised. It's possible that low-code platforms won't be able to give the MERN stack with the same level of freedom and control as traditional coding.

CHAPTER 5

CONCLUSION

5.1 Conclusion

This project report has examined the comparability of the MERN stack and no-code/low-code platforms for creating online and mobile applications. The results imply that no-code/low-code platforms can provide the MERN stack with considerable advantages when given better resources and customisation choices.

Platforms that need little or no coding, like AppSmith and Apache Hop, make it easy to create applications quickly. Because these systems support drag-and-drop capabilities, visual interfaces, and pre-built components, they eliminate the need for manual coding and shorten development cycles. Because no-code/low-code platforms are simple to use and offer streamlined integration possibilities, they can be quickly prototyped and iterated upon, making them ideal for projects with condensed development cycles.

However, it's critical to recognise that the availability of improved resources, in-depth documentation, and a supportive community are crucial for no-code/low-code platforms to be effective. The lack of development in these areas can be problematic for complicated projects or cutting-edge functionality. Before deciding to use a particular no-code/low-code platform, developers and organisations must carefully evaluate the resources and assistance offered by that platform.

Overall, compared to the MERN stack, no-code/low-code platforms have the ability to provide a more streamlined and effective development experience when given more resources and customization possibilities. They are an appealing option for projects with shorter schedules or those looking for a higher level of flexibility and simplicity of use because they enable rapid prototyping, simplified integration, and significant customization. To make informed judgements and increase development efficiency, it is essential to properly assess the capabilities and limitations of no-code/low-code platforms in relation to project needs.

5.2 Future Scope

- Development of a fully functional application with additional features and functionalities.
- Deployment of the developed applications on Kubernetes and Docker containers for enhanced scalability and portability.
- Exploration of the integration possibilities between no-code/low-code platforms and artificial intelligence (AI) technologies.

REFERENCES

- [1] React. (n.d.). React Documentation. Retrieved from <https://reactjs.org/docs>
- [2] JavaScript. (n.d.). JavaScript Documentation. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
- [3] MongoDB. (n.d.). MongoDB Documentation. Retrieved from <https://docs.mongodb.com>
- [4] Express.js. (n.d.). Express.js Documentation. Retrieved from <https://expressjs.com/en/api.html>
- [5] Node.js. (n.d.). Node.js Documentation. Retrieved from <https://nodejs.org/en/docs>
- [6] Apache Hop. (n.d.). Apache Hop Documentation. Retrieved from <https://hop.apache.org/documentation.html>
- [7] AppSmith. (n.d.). AppSmith Documentation. Retrieved from <https://docs.appsmith.com>
- [8] Ngrok. (n.d.). Ngrok Documentation. Retrieved from <https://ngrok.com/docs>
- [9] MERN Stack. (n.d.). MERN Stack Tutorial - Build a MERN App From Scratch. Retrieved from <https://www.mongodb.com/mern-stack>