***INDUSTRIAL TRAINING REPORT***

***On***

***ChoiceDekho: Best Ed-Tech Solutions***

***At***

***UDEA SOLUTIONS LLP***

*Submitted in partial fulfillment of the requirements*

*for the award of degree of*

Bachelor of technology

Computer Science and Engineering

A circular logo with text and a shield

Description automatically generated

# Submitted by:

# Sushil Verma

# 03013302721

# CSE-7B

CSE 7 B

# Submitted to:

Mr. Karmbir Khatri

Assistant Professor

CSE Department

# HMR INSITUTE OF TECHNOLOGY AND MANAGEMENT HAMIDPUR, DELHI -110036

**Affiliated to**

# GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

**Sector-16C Dwarka, Delhi-110075, India 2021-2025**

# DECLARATION

I, **SUSHIL VERMA,** Student of B. Tech (CSE) declare that the project titled “**CHOICE DEKHO: BEST ED-TECH SOLUTIONS”** which is submitted by me to Department of computer science and engineering, HMR INSTITUTE OF TECHNOLOGY AND MANAGEMENT, HAMIDPUR, NEW DELHI affiliated to Guru Gobind Singh Indraprastha University comprises only my original work and due acknowledgement has been made in the text to all other materials used.

### Date:

**SUSHIL VERMA**

# ACKNOWLEDGEMENT

The successful completion of the **ChoiceDekho** project marks the beginning of a continuous learning journey, transforming ideas and concepts into practical, real-world systems. Working on this project has been an enriching experience, helping me gain valuable skills and confidence to work in a professional environment. The knowledge and hands-on experience gained during this project will undoubtedly open up bright opportunities for me in the future.

I would like to extend my heartfelt gratitude to our Director, **Mr. V. C. Pandey**, and our esteemed faculty for imparting the knowledge that empowered me to complete my internship and prepare this project during that period. I am also sincerely thankful to the mentors at **UDEA SOLUTIONS LLP** for taking a keen interest in my training, internship, providing valuable suggestions, and assisting me throughout the development of **ChoiceDekho**, both directly and indirectly.

SUSHIL VERMA 03013302721

B. Tech (CSE 7B)

# CERTIFICATES

# 

**ABSTRACT**

This report outlines the experience and skills gained during my internship at UDEA Solutions LLP. My training involved working on the **ChoiceDekho** project, focusing on data-driven solutions and leveraging technologies such as **React**, **Node.js**, **Express, MongoDB, Redis**, **data scraping**, and **Azure** for deployment.

The **ChoiceDekho** project is a platform that assists users in discovering the best course options across various categories. The courses displayed on the platform are sourced through **data scraping** from multiple websites as well as directly from **Impact**. The scraped data undergoes preprocessing to ensure accuracy, relevance, and optimal platform performance.

We integrated **Redis** to enhance data caching, which significantly improves response times and the overall scalability of the system. The platform is deployed on **Azure**, leveraging its cloud infrastructure to ensure reliable, secure, and scalable performance.

This project gave me hands-on experience with web development, data management, cloud deployment, and performance optimization, demonstrating how modern technologies such as **Redis**, **data scraping**, and **Azure** are used to build efficient, scalable, and user-friendly platforms.

In summary, this project not only showcases the implementation of cutting-edge technologies but also highlights practical challenges and solutions involved in creating a robust, scalable system in a real-world application.

# ABOUT UDEA SOLUTIONS LLP

UDEA Solutions LLP is a technology and **software development company** based in Delhi, India. Founded with the vision of providing innovative digital solutions, UDEA Solutions specializes in building tailored software products and platforms for businesses across various industries. The company offers a wide range of services, including web and mobile application development, cloud solutions, and data-driven technologies. Known for its customer-centric approach, UDEA Solutions has worked with numerous clients to help them achieve their business goals through custom software development and digital transformation. At the core of UDEA’s mission is a commitment to delivering high-quality, scalable, and innovative solutions that drive success for its clients in the digital age.

# ABOUT INTERNSHIP

UDEA Solutions LLP’s **6-month** Software Development Internship provided an in-depth learning experience in full-stack development, covering areas such as web development, data scraping, Redis integration, and cloud deployment on Azure. Throughout the program, I worked on a diverse range of tasks, from developing APIs to optimizing databases, while also contributing to the deployment of live projects. The hands-on approach involved real-world challenges, and regular feedback, which helped me improve both my technical and problem-solving skills. Initially a 6-month internship, it was extended for an additional **3 months** to ensure the successful completion of a large-scale project, further enhancing my experience in managing complex software systems and meeting project deadlines.

# List of Figures

|  |  |  |
| --- | --- | --- |
| **S.no** | **Figures** | **Page No.** |
| 1. | Flow of MERN | 4 |
| 2. | Architecture | 6 |
| 3. | API Integration | 16 |
| 4. | State vs Prop | 19 |
| 5. | Data Flow Diagram | 38 |
| 6. | Code | 39 |
| 7. | Output | 47 |

### DECLARATION ACKNOWLEDGEMENT CERTIFICATE ABSTRACT

**ABOUT UDEA SOLUTIONS**

**ABOUT INTERNSHIP**

# CONTENT

### i ii iii iv v vi

1. **INTRODUCTION TO MERN STACK**

### MERN Stack Overview……………………………………………………………………...1

### History………………………………………………………………………………………..2

### Frontend in the MERN Stack……………………………………………………………….2

### Backend in the MERN Stack………………………………………………………………..2

### IMPORTANCE & RELEVANCE IN MODEERN WEB DEV

### Importance 3

### Usage of MERN 3

### MERN Methodologies 5

### Phases of MERN Stack 6

### UNDERSTANDING THE COMPONENT OF THE MERN INDIVIDUALLY

### Introduction to MongoDB 8

### History of MongoDB 8

### Features of MongoDB 9

### Data Modeling and schema design 10

* 1. ExpressJS 12
     1. Introduction to ExpressJS 12
     2. Basics of Express.js Framework 13
     3. Parts of ExpressJS 13
     4. Restful API development with ExpressJS 15
  2. React 17
     1. Introduction to React.js 17
     2. History of React.js 17
     3. Features of React 18
     4. Virtual DOM & React rendering 20
  3. Introduction to Node.js 21
     1. Why Node.js 21
     2. Asynchronous programing & server-side scripting 21
     3. Event Driven Architecture 24

1. INTEGRATION OF MERN STACK
   1. MongoDB Installation 25
   2. Node.js Installation 25
   3. Express.js Installation 26
   4. React.js Installation 26
   5. Connecting MongoDB with Node.js/Express.js 27
   6. Building MongoDB with Node.js/Express.js 27
2. [DEVELOPMENT ENVIRONMENT](#_TOC_250015)
   1. Introduction to Visual Studio Code 28
      1. What can Visual Studio Code can do? 28
      2. Advantages of VSS 29
   2. Introduction to GitHub 30
      1. What is Github? 30
      2. Key Features 30
   3. Introduction to Microsoft Azure 31
      1. What is Azure? 31
      2. Key Features 31
      3. Getting Started 32
3. [SYSTEM DESIGN](#_TOC_250005)
   1. Overview 33
   2. Goals & Objectives 33
   3. High-Level Architecture 34
   4. Data Flow 34
   5. Detailed Design 35
      1. Backend 35
      2. Comparison Service 35
      3. User Service 36
      4. Caching Layer 36
      5. Fronted 36

6.6 Database Design 37

6.7 External Integration 37

6.7.1 Impact API Integration 37

6.8 Non-Functional Requirements 38

1. SOURCE CODE 39
2. OUTPUT 47
3. CONCLUSION 50
4. BIBLIOGRAPHY …………………………………………………………………………….51

# CHAPTER 1 INTRODUCTION TO MERN STACK

## MERN STACK Overview

MERN stack is a collection of technologies that enables faster application development. It is used by developers worldwide. The main purpose of using MERN stack is to develop apps using JavaScript only. This is because the four technologies that make up the technology stack are all JS-based. The MERN stack, composed of MongoDB, Express.js, React, and Node.js, revolutionizes modern web development. MongoDB's NoSQL approach fosters flexible data integration, while Express.js simplifies server-side tasks. React's component-based approach optimizes user interfaces, and Node.js enables scalable server-side execution. Together, these technologies form a potent synergy, leveraging JavaScript's versatility across the development stack. This report explores each component's nuances and their collaborative strength, showcasing their practical application through a project developed during the training. The MERN stack's collective power lies in its ability to create robust, scalable, and efficient web applications, making it a cornerstone in contemporary web development landscapes.

### HISTORY

The history of the MERN stack revolves around the evolution and integration of its core technologies—MongoDB, Express.js, React, and Node.js—each with its own distinct timeline.

* + - **Node.js (2009):** Node.js was introduced by Ryan Dahl in 2009. It marked a significant shift by allowing JavaScript to be used for server-side programming. Its non-blocking, event-driven architecture appealed to developers seeking more efficient ways to build scalable applications.
    - **Express.js (2010):** Express.js emerged in 2010 as a minimalist web application framework for Node.js. It simplified the process of building web applications by

  providing robust features for routing, middleware creation, and handling HTTP requests.

* + - **React.js (2013):** Developed by Facebook, React.js was released in 2013. It revolutionized frontend development by introducing a component-based architecture and the concept of a virtual DOM, enhancing performance and reusability in UI development.
    - **MongoDB (2009):** MongoDB, released in 2009, offered a different approach to database management. It departed from traditional relational databases by employing a flexible, document-based data model, making it easier to work with data in the JSON like format.

### Frontend in the MERN Stack

The front-end layer of the MERN stack revolves around React.js, a powerful JavaScript library for building dynamic user interfaces. React's hallmark is its component-based architecture, enabling developers to create reusable UI elements that efficiently manage their own states. Utilizing a virtual DOM, React optimizes rendering by updating only the necessary components when data changes, thereby enhancing performance and user experience.

React seamlessly integrates with backend APIs, enabling smooth data flow between the frontend and backend layers. Its modular approach fosters code reusability, scalability, and maintainability, making it an ideal choice for crafting complex and responsive interfaces.

Additionally, React's extensive ecosystem of tools and libraries facilitates the development of Single Page Applications (SPAs) and rich web applications, ensuring a streamlined development process.

### Backend in the MERN Stack

Node.js powers the backend of the MERN stack, executing JavaScript server-side. Its nonblocking, event-driven architecture allows for handling concurrent requests efficiently, ensuring scalability. Express.js, a lghtweight framework built on top of Node.js, simplifies server-side development by providing robust features for routing, middleware management, and API creation.

# CHAPTER 2

# IMPORTANCE & RELEVANCE IN MODERN

# WEB DEVELOPMENT

## IMPORTANCE

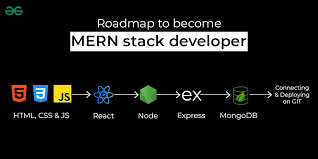
The MERN stack's relevance in modern web development stems from its holistic approach, uniting MongoDB's flexibility, Express.js's streamlined server-side operations, React's dynamic user interfaces, and Node.js's server-side JavaScript execution. Its core advantage lies in leveraging JavaScript throughout the development cycle, promoting code coherence and versatility. This unity simplifies data transfer between frontend and backend, facilitating agile and responsive application development.

The stack's adaptability to handle complex, real-time data and its scalability make it a go-to choice for developers aiming to create robust, high-performance web applications. Its unified ecosystem minimizes compatibility issues and offers a wide array of libraries and tools, fostering rapid development cycles. Embraced by a thriving community, the MERN stack remains pivotal in modern development paradigms, empowering developers to craft sophisticated, data-centric applications while ensuring flexibility, efficiency, and maintainability across the entire development spectrum.

## Usage of MERN

The MERN stack finds extensive use across various sectors within the IT industry due to its versatility, scalability, and efficiency. Here are some key uses of the MERN stack in IT industries:

* Web Application Development: MERN is widely employed for developing web applications across diverse domains, including e-commerce, social media, content management systems, and more. Its robustness in handling real-time data, scalability, and responsiveness makes it ideal for creating dynamic, data-driven web applications.
* Startups and Small Businesses: Due to its open-source nature, ease of learning, and comprehensive ecosystem, the MERN stack is popular among startups and small businesses. It enables rapid prototyping, quicker development cycles, and cost-effective solutions for establishing online platforms.
* Enterprise Solutions: Large-scale enterprises leverage the MERN stack to build scalable, enterprise-grade applications. Its ability to handle complex functionalities, coupled with its performance and scalability features, makes it suitable for mission-critical applications within organizations.
* Real-time Applications: MERN's event-driven architecture, particularly with Node.js, is well-suited for building real-time applications like chat applications, live streaming
* platforms, collaborative tools, and IoT applications. Its ability to handle multiple concurrent connections efficiently is advantageous for such applications.
* Cross-platform Development: React Native, which utilizes React.js, allows developers to
* build cross-platform mobile applications using the same codebase as the web application. This reusability significantly reduces development time and effort for both web and mobile platforms.
* Data-intensive Applications: MongoDB's document-based structure in the MERN stack makes it ideal for handling large volumes of unstructured or semi-structured data. It's used in applications requiring flexible data models, such as content management systems or applications dealing with user-generated content.
* Innovation and Rapid Development: The MERN stack's unified JavaScript ecosystem encourages innovation and experimentation. It facilitates quick iteration, enabling developers to bring new features to market faster and stay agile in an ever-evolving digital landscape.

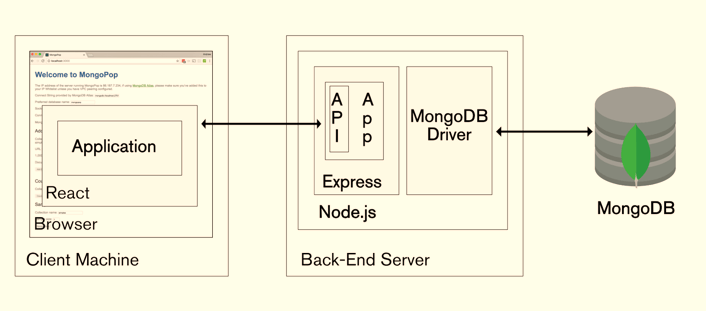


**Fig 1**: Flow of MERN

## MERN Stack Methodologies

The MERN stack isn't tied to a specific methodology but can be adapted to various software development approaches. Here are methodologies commonly applied when working with the MERN stack:

* **Agile Development:** MERN is well-suited for Agile methodologies like Scrum or Kanban. Its modular structure allows teams to work in sprints, iteratively building and testing components, ensuring flexibility to adapt to changing requirements.
* **Iterative Development:** With its modular nature, the MERN stack supports iterative development. Developers can continuously enhance features, adding or modifying components in an incremental manner while ensuring the application remains functional and maintainable.
* **Test-Driven Development (TDD)**: Using testing frameworks like Jest for React and Mocha/Chai for Node.js, developers can adopt a test-driven approach. They write tests first, then write code to pass those tests, ensuring code reliability and facilitating future modifications.
* **Component-Based Development:** Given React's component-based architecture, a methodology centered around building and reusing modular UI components is common. This promotes code reusability, maintainability, and easier collaboration among developers.
* **Scalability-Oriented Design**: Node.js' non-blocking I/O and MongoDB's scalability allow for a methodology focused on scalability. Developers design applications to handle increasing loads efficiently, emphasizing scalability from the initial stages.
* **Microservices Architecture**: While not strictly a methodology, MERN can be used within a microservices architecture. Each component (MongoDB, Express.js, React, and Node.js) can be encapsulated into microservices, promoting flexibility and scalability in complex systems.
* It's important to note that the choice of methodology often depends on project requirements, team preferences, and the specific needs of the application being developed. Teams might adopt a hybrid approach or customize methodologies to best fit the MERN stack's capabilities and the project's objectives.



**Fig 2**: Architecture

## Phases of MERN Stack

While the MERN stack doesn't strictly follow designated phases, its development process generally involves several key stages:

**Planning and Requirement Analysis:**

* Understanding project objectives, user requirements, and functionalities needed in the application.
* Determining the scope, features, and architecture that align with the MERN stack's capabilities.

**Design and Architecture:**

* Defining the application's overall structure, including database schema, frontend UI/UX design, and backend architecture with Node.js and Express.js.

**Development:**

* Creating the application's components using React.js for the frontend and implementing server-side logic with Node.js and Express.js.
* Integrating MongoDB to handle data storage and retrieval, creating APIs for communication between frontend and backend.

**Testing:**

* Conducting unit tests, integration tests, and end-to-end tests to ensure functionality,

performance, and compatibility across the MERN components.

* Employing testing frameworks like Jest, Mocha, or Enzyme for React and Node.js testing.

**Deployment and Optimization:**

* Configuring deployment environments and setting up CI/CD pipelines for automated deployment.
* Performance optimization to enhance application speed and efficiency, ensuring scalability and reliability.

**Monitoring and Maintenance:**

* Implementing monitoring tools to track application performance, identify issues, and make necessary improvements.
* Continuous maintenance, including bug fixes, feature updates, and security patches

# CHAPTER 3

**UNDESTANDING THE COMPONENT OF THE**

**MERN INDIVIDUALLY**

## Introduction to MongoDB

MongoDB, a prominent NoSQL database, diverges from traditional relational databases by employing a schema-less approach. Its document-based structure and scalability make it ideal for managing unstructured data, facilitating high performance and adaptability in modern application development within the MERN stack

This database system boasts high scalability, allowing for distributed architectures and horizontal scaling across clusters. It excels in handling large volumes of data, making it preferred choice for applications requiring flexibility in data modeling and real-time analytics. MongoDB's rich query language and indexing capabilities facilitate efficient data retrieval and manipulation. With its versatility, scalability, and capability to handle complex data models, MongoDB has emerged as a cornerstone in modern application development, playing a pivotal role in the MERN stack's backend infrastructure.

## History to MongoDB

MongoDB, initially developed by MongoDB Inc., emerged in 2007 as a response to the limitations of traditional relational databases. Its alpha version debuted in 2009. Dwight Merriman, Eliot Horowitz, and Kevin Ryan conceptualized MongoDB to address the growing demand for a more scalable, flexible, and easily accessible database solution for modern web applications. By introducing a document-oriented database model and leveraging JSON-like documents, MongoDB aimed to simplify development and scalability, marking a paradigm shift in database management. Since its inception, MongoDB has evolved through various versions, continually enhancing its features and functionalities to meet the evolving needs of contemporary application development.emerged as a cornerstone in modern application development, playing a pivotal role in the MERN stack's backend infrastructure.

## Features to MongoDB

In MongoDB, being a document-oriented database, data is stored in JSON-like documents within collections. Here are some key aspects of MongoDB's document-oriented approach:

1. **Flexible Schema:**

* Documents can have varying structures within the same collection, offering schema flexibility.
* Fields can be added or modified without requiring changes to other documents, facilitating easy adaptation to evolving data models.

1. **BSON Format:**

* BSON (Binary JSON) is MongoDB's binary-encoded serialization of JSON-like documents, providing efficient storage and data retrieval.
* Supports various data types, including strings, numbers, arrays, and nested documents.

1. **Document Hierarchy:**

* BSON Documents can have nested structures, allowing for complex and hierarchical data models.
* This nesting capability reflects real-world relationships more naturally than traditional table-based structures.

1. **Atomic Operations:**

* MongoDB supports atomic operations on single documents, ensuring consistency and data integrity.
* Updates within a single document are atomic, preventing inconsistencies in data modifications.

1. **Scalability and Performance:**

* Document-based storage enhances performance by storing related data in a single document, reducing the need for joins.
* Supports horizontal scaling by distributing collections or shards across multiple servers or clusters.

1. **Schema Evolution:**

* Facilitates dynamic schema changes, allowing applications to evolve without disrupting existing data.
* Enables seamless modifications to adapt to changing business needs or application requirements

1. **Rich Querying:**

* MongoDB's query language (MongoDB Query Language - MQL) supports complex queries, filtering, and aggregations, empowering efficient data retrieval and manipulation.
* The document-oriented nature of MongoDB empowers developers to work with data in a more intuitive and adaptable manner, allowing for agile and flexible application development.

## Data modeling and schema design

Data modeling and schema design in MongoDB revolve around creating structures that best fit the application's needs while taking advantage of MongoDB's document-oriented nature. Here's an overview:

1. **Understanding Data Requirements:**

* Analyze the application's data requirements, considering the nature of data, relationships, and how it will be accessed or modified.

1. **Document Structure:**

* Design documents that represent entities or objects within the application. Each document typically corresponds to a record or an entity.
* Utilize embedded documents and arrays to represent complex relationships and hierarchical data.

1. **Normalization vs. Embedding:**

* Consider whether to normalize data (separate collections and use references) or embed related data within documents.
* Embedding can improve query performance by reducing the need for joins but may impact update complexities and document size.

1. **Data Consistency:**

* Ensure consistency within documents. Atomic operations on single documents maintain data integrity.
* Normalize or separate data if frequent updates to a particular subset of fields might lead to high contention.

1. **Schema Design Best Practices:**

* Design for query patterns, considering how data will be queried and accessed.
* Use appropriate data types, indexing, and compound indexes to optimize performance.

1. **Scalability Considerations:**

* Plan for scalability by considering how data will scale as the application grows.
* Employ sharding strategies, distribute data across shards, and leverage replica sets for redundancy and fault tolerance.

1. **Iterative Approach:**

* Refine the data model iteratively based on real-world usage and feedback, adjusting as the application evolves.
* MongoDB's flexibility allows for agile schema design and data modeling, enabling developers to tailor structures to specific application needs, optimize performance, and accommodate changes over time without compromising efficiency.

## ExpressJS

Express.js, a Node.js web application framework, simplifies backend development with streamlined route handling and middleware management. Its lightweight nature and extensive middleware support enable rapid API and web app creation, making it a popular choice for scalable and efficient server-side architectures.

## Introduction to Express

Express.js, a Node.js web application framework, epitomizes simplicity and flexibility in server-side development. Serving as a robust middleware, Express.js streamlines the creation of web applications and APIs by providing an intuitive and minimalist structure. With its focus on routing, middleware integration, and handling HTTP requests, Express.js empowers developers to construct scalable and dynamic applications effortlessly. Its lightweight nature allows for quick prototyping while offering a rich set of features for production-grade applications.

Express.js embraces JavaScript, fostering a modular and organized approach to backend development. It enables the construction of efficient server-side architectures, leveraging

 Node.js's event-driven paradigm to manage asynchronous operations efficiently. This framework's extensive middleware support extends its functionality, allowing developers to easily integrate additional features like authentication, logging, and error handling.

The vibrant community and an expansive ecosystem of third-party middleware and plugins further enhance Express’s appeal. Its adaptability and customizable nature make it a preferred choice for developers seeking agility and performance in building APIs and web applications within the Node.js environment. Express.js stands as a cornerstone for building scalable, efficient, and feature-rich server-side components within the MERN stack and beyond.

## Basics of Express.js Framework

Express.js, built on Node.js, offers a powerful framework for developing web applications and APIs. At its core, Express simplifies backend development by providing an intuitive, minimalist structure that streamlines server-side processes. It introduces a straightforward routing system, allowing developers to define various endpoints and handle HTTP requests efficiently. Express's middleware capabilities enable the incorporation of additional functionalities into the application, such as authentication, logging, and error handling. This framework's flexibility allows for customization, enabling developers to adapt the server logic to specific project needs. Express.js' lightweight nature, extensive community support, and vast ecosystem of plugins make it an indispensable tool for building scalable and robust web applications, providing a solid foundation for crafting diverse and efficient backend architectures

## Parts of Express.js

1. **Routing in Express.js:**

* Express.js routing involves defining endpoints for handling incoming HTTP requests. It allows developers to map specific URLs (or patterns) to corresponding code that processes those requests.
* For instance, a route might handle requests to "/products" by fetching product data from a database and rendering it on a webpage. Routes can also incorporate parameters, enabling dynamic URL segments (e.g., "/products/:id") to access specific resources.
* Express's robust routing system enables middleware to be attached to specific routes,allowing for authentication checks, logging, or data validation before reaching the route handler. This modular approach aids in code organization, making applications easier to maintain and scale.

1. **Middleware in Express.js:**

* Express.js middleware functions intercept incoming HTTP requests and responses. They sit between the client request and the server's final response, performing specific tasks such as authentication, error handling, or logging. Middleware functions are executed in sequence, and they have access to the request and response objects, enabling manipulation of data or headers. This powerful feature enables cross-cutting concerns to be addressed uniformly across routes without duplicating code, enhancing code reusability, maintainability, and ensuring consistent behavior across the application.

1. **HTTP Request Handling in Express.js:**

* Express.js simplifies handling HTTP requests and responses by providing a clear and intuitive interface. It parses incoming data (e.g., form submissions or JSON payloads), making it accessible to route handlers. Express enables setting response status codes, headers, and sending back responses to clients. Through middleware, developers canpreprocess incoming data, sanitize it, or validate it before it reaches the route handlers, ensuring cleaner and more manageable code within the routes themselves. This streamlined approach aids in efficiently managing the request-response lifecycle, facilitating robust and scalable applications.

## 3.2.4 RESTful API development with ExpressJS

Developing RESTful APIs with Express.js involves leveraging its routing and middleware capabilities to create endpoints that adhere to RESTful principles. Here's a breakdown of the key steps and concepts:

1. **Define Routes:**

* Express.js to define routes that correspond to different HTTP methods (GET, POST, PUT, and DELETE) and resource endpoints.
* For example, /api/users might handle GET requests to retrieve user data, POST requests to create new users, etc.

1. **HTTP Methods and Endpoints:**

* Map HTTP methods to CRUD operations (Create, Read, Update, Delete) on specific endpoints.
* Use app.get(), app.post(), app.put(), app.delete() methods in Express to handle different HTTP methods for each route.

1. **Route Handling and Middleware:**

* Implement middleware functions to perform tasks like authentication, data validation, or error handling.
* Middleware can be used globally or applied to specific routes to maintain consistency and security across API endpoints.

1. **Use of HTTP Status Codes:**

* Utilize appropriate HTTP status codes (e.g., 200 for successful GET requests, 201 for successful POST requests, 404 for not found) to convey the status of API responses accurately.

1. **Resourceful Routing:**

* Structure routes and endpoints to represent resources logically, following RESTful conventions.
* For instance, /api/books for managing books, /api/users for managing users, adhering to consistent naming conventions.

1. **Data Handling:**

* Implement logic within route handlers to interact with data sources (such as databases) to perform CRUD operations on resources.

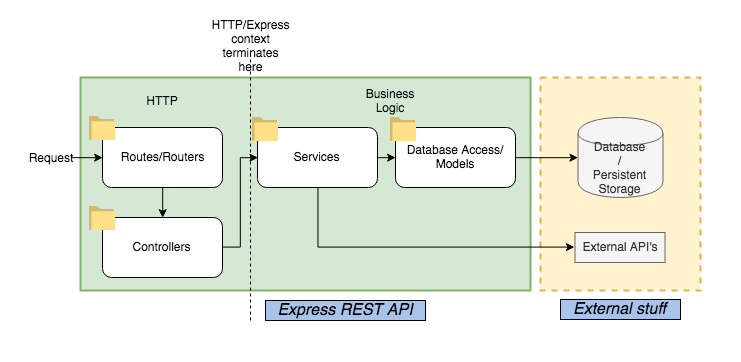
1. **Response Formatting:**

* Format responses in a consistent and standardized way, often as JSON, to provide clear and predictable data to clients.

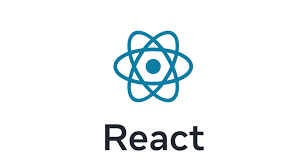
1. **Testing and Documentation:**

* Thoroughly test API endpoints using tools like Postman or automated tests to ensure proper functionality **.**
* Document API endpoints, their usage, and expected responses using tools like Swagger or OpenAI for better developer understanding and integration.

**Fig 3**: API Integration



## React

React.js is a widely used JavaScript library that simplifies front-end development by enabling the creation of reusable UI components. Its component-based architecture, efficient rendering, and virtual DOM facilitate building interactive, high-performance web applications.

## Introduction to React.js

React.js, a JavaScript library for building user interfaces, revolutionizes front-end development by offering a modular and declarative approach. It simplifies UI creation through reusable components, breaking down complex interfaces into manageable parts. React's virtual DOM and one-way data flow ensure efficient rendering, enabling selective updates without rerendering the entire interface. This optimizes performance and boosts responsiveness, vital for modern web applications. Widely adopted for its flexibility and scalability, React's ecosystem, including tools like Redux for state management and JSX for easy syntax, empowers developers to create dynamic, interactive interfaces. Its popularity stems from its ability to handle complex UI logic, making it a go-to choice for crafting engaging and high-performance applications across various industries and platforms.

## History to React.js

React.js emerged from a need to streamline front-end development at Facebook. Initially developed by Jordan Walke, React started as an internal project in 2011 and was first deployed on Facebook's newsfeed in 2012. It aimed to solve the challenges of efficiently updating complex UIs. React was open sourced in 2013, marking its official public release. Its innovative Virtual DOM concept and component-based architecture revolutionized UI development, enabling faster rendering by selectively updating components.

Over the years, React has seen significant growth and adoption beyond Facebook. It’s community-driven development, regular updates.

React's influence expanded beyond web development into mobile with ReactNative, allowing developers to build cross-platform mobile applications using the same Reactprinciples. As a result, React.js remains a pivotal force in modern front-end development, continually evolving to meet the demands of scalable and dynamic user interfaces.

## Features of React

1. **Components:**

* In React, components are reusable, self-contained building blocks that encapsulate a piece of the user interface along with its behavior. They can be as simple as a button or as complex as an entire page. Components can be functional or class-based. Functional components are simpler, relying solely on functions to define the UI, while class components have additional features like lifecycle methods and state management.

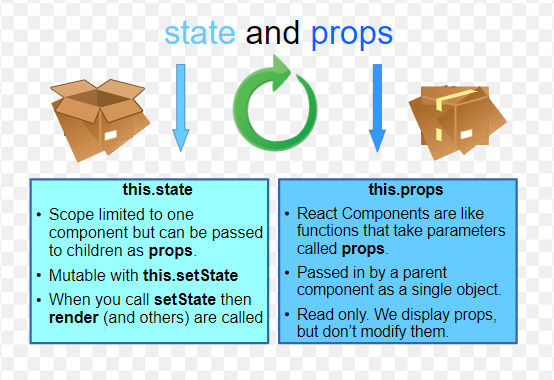
1. **Props (Properties):**

* Props allow data to be passed from parent to child components, enabling the customization and dynamic rendering of components. They are read-only and help maintain the immutability of data in React. Components receive props as function parameters (in functional components) or as properties of the this.props object (in class components). Props enable components to be reusable and flexible by providing different data or configurations based on their use within the application.

1. **State Management:**

* State represents the internal data of a component that can change over time. It allows components to manage their data independently. State is initialized within the component using this.state (in class components) or useState hook (in functional components). When a state changes, react re-renders the component, updating the UI to reflect the new state. Stateful components manage their own data, making them dynamic and interactive.

1. **State vs. Props:**

* State is internal to a component and can change within that component, triggering re-renders when modified. Props are passed down from parent components and are immutable within the receiving component, ensuring a unidirectional flow of data. Understanding the distinction between these concepts is crucial for managing data flow and reactivity in React applications. **Fig 4**: State vs Prop

1. **State Lifting and Management:**

* In larger applications, managing state becomes more complex. State lifting involves moving the state to higher-level components to share data among multiple child components. React also offers state management solutions like the Context API for global state management across components or Redux for centralized state management in more extensive applications.
* Mastering components, effectively utilizing props for data flow, and managing state efficiently are foundational skills in React development. They enable developers to build scalable, maintainable, and interactive user interfaces

## Virtual DOM and React rendering

## Virtual DOM

React's Virtual DOM stands as a pivotal performance optimization tool, acting as a lightweight, in-memory representation of the actual browser DOM. It serves as a reconciliation layer between the application's components and the browser, optimizing the process of updating the user interface. When changes occur within a React application, instead of directly manipulating the real DOM, React creates a new Virtual DOM tree, reflecting the updated UI based on the component changes.The key efficiency of the Virtual DOM lies in React's diffing algorithm.

This algorithm compares the previous Virtual DOM with the newly generated one, identifying the specific parts that have changed. By computing the difference between the two trees, React determines the minimal set of modifications required to bring the actual DOM up to date with the application's current state. Consequently, it selectively updates only the elements that have changed, significantly reducing the number of costly and potentially slow manipulations to the real DOM.

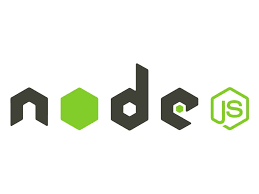
By leveraging the Virtual DOM, React minimizes direct interactions with the browser's DOM, which is relatively expensive in terms of performance. This optimization technique leads to a more responsive and efficient rendering process, particularly in applications with dynamic content or frequent updates, enhancing overall user experience and ensuring smoother interactions.

## React Rendering Process

React's rendering process involves the creation and management of a Virtual DOM, which serves as an intermediary step before affecting the actual browser DOM. When a component's state or props change, React triggers a re-rendering process. It constructs a new Virtual DOM representation based on the updated state, allowing it to efficiently determine the differences between the previous and updated Virtual DOMs.

React's diffing algorithm then performs a comparison between these two Virtual DOM representations, analyzing the changes and computing the minimal set of alterations required to synchronize the actual browser DOM with the updated state of the application manipulations.

## Introduction to Node.js

 Node.js is an open-source, server-side JavaScript runtime environment that allows developers to execute JavaScript code on the server. It operates outside the browser and enables the creation of highly scalable and efficient network applications. Node.js, introduced in 2009, is an open-source, server-side JavaScript runtime environment built on Chrome's V8 JavaScript engine. It revolutionized back-end development by enabling JavaScript to execute outside the browser, allowing developers to build scalable and high-performance applications. Node.js employs an event-driven, non-blocking I/O model, facilitating efficient handling of concurrent requests. Its lightweight nature, extensive ecosystem through npm (Node Package Manager), and cross-platform compatibility make it a preferred choice for diverse applications, from web servers and APIs to real-time applications and IoT devices. By leveraging JavaScript on both the client and server sides, Node.js simplifies code sharing and accelerates development workflows. Its asynchronous architecture allows for optimized handling of I/O-bound tasks, making it well-suited for building dataintensive and real-time applications. Node.js continues to evolve, remaining a cornerstone in modern server-side development, empowering developers to create robust, scalable, and agile applications.

## 3.4.1 Why Node.js?

Node.js stands out as a pivotal choice in server-side development owing to its multifaceted advantages. Its most compelling feature lies in enabling JavaScript utilization not only for client-side but also server-side programming, unifying the language across the entire application stack. This facilitates seamless code sharing, fostering a coherent development environment and enhancing productivity. Node.js's event-driven, non-blocking I/O architecture empowers applications to handle concurrent operations efficiently, ensuring scalability without compromising performance. Its extensive npm ecosystem simplifies development by providing an array of readily available modules, accelerating the creation of robust applications.

The platform's lightweight nature and optimization for speed ensure swift responses and reduced resource usage. Such attributes collectively position Node.js as an ideal framework for building high-performance, real-time applications, micro services, APIs, and more, offering developers a versatile tool to craft responsive, scalable, and resource-efficient applications across diverse domains.

## 3.4.2 Asynchronous programming & server-side scripting

## Asynchronous programming:

* Asynchronous programming is a fundamental aspect of Node.js, enabling applications toperform multiple operations simultaneously without waiting for each one to complete before starting the next. This approach significantly enhances efficiency, especially in I/O-bound tasks.
* Node.js leverages an event-driven, non-blocking I/O model, allowing it to handle numerous operations concurrently. When an asynchronous function is called, Node.js doesn't halt the execution of subsequent code. Instead, it continues processing other tasks and registers a callback to be executed upon completion of the asynchronous operation. This callback-driven approach ensures that Node.js remains responsive and can manage high throughput without getting blocked by time-consuming tasks.
* Callbacks, the traditional approach, involve passing functions as arguments to handle asynchronous tasks' results. While effective, they often lead to callback hell, a nesting of multiple callbacks, making code readability and maintenance challenging.
* Promises to introduce a more structured way of dealing with asynchronous operations. They represent a future value or an eventual result of an asynchronous task, allowing chaining and handling of success or failure through then () and catch() methods. Promises alleviate callback nesting issues and simplify error handling, enhancing code readability.
* Async/await is a newer addition, providing a more synchronous-like syntax to asynchronous code. It operates on top of Promises, allowing developers to write asynchronous code that appears synchronous. By using the async keyword before a function and await within it to pause execution until a Promise settles, this paradigm significantly enhances code clarity, making it easier to understand and maintain.
* These paradigms collectively enable developers to manage asynchronous tasks efficiently in Node.js. While callbacks are foundational, Promises and async/await have evolved to address callback complexities, providing more elegant solutions for handling asynchronous operations, enhancing code readability, and simplifying error handling. Developers often choose between these paradigms based on code structure, readability, and their specific application's needs.

## Server-Side Scripting:

* Server-side scripting involves executing scripts on the server to generate dynamic content or respond to client requests. In the context of Node.js, server-side scripting typically involves writing JavaScript code to handle incoming requests, process data, interact with databases, and generate dynamic responses.
* With Node.js, JavaScript can be used for both client-side and server-side scripting, unifying the programming language across the entire application stack. This unification streamlines development workflows, allowing developers to use the same language and coding patterns on both ends.
* In server-side scripting, Node.js leverages its event-driven, non-blocking I/O model. When a client sends a request, Node.js handles it asynchronously, allowing the server to continue processing other tasks without waiting for the request to complete. This approach ensures scalability and responsiveness, making Node.js well-suited for applications that require handling multiple simultaneous requests, real-time data processing, or I/O-intensive operations.
* Node.js facilitates server-side scripting by providing modules and APIs that enable developers to create robust, high-performance applications. It allows for the creation of web servers, APIs, micro services, and various backend functionalities, leveraging JavaScript's flexibility and Node.js's event-driven architecture to build efficient server-side applications.

## 3.4.3 Event Driven Architecture

Event-driven architecture (EDA) is a design paradigm where the flow of the system is determined by events such as user actions, system notifications, or messages from other components. In an event-driven system, components communicate asynchronously by generating, capturing, and responding to events.

In the context of Node.js, EDA plays a fundamental role due to its event-driven, non-blocking I/O model. Node.js employs an event loop that continuously listens for events. When an event occurs, such as an incoming HTTP request or the completion of a database query, Node.js triggers the associated event handler or callback function.

This approach allows Node.js applications to be highly responsive and scalable. Instead of waiting for each task to finish before moving to the next, Node.js can efficiently handle multiple events concurrently, making it particularly well-suited for I/O-bound tasks where operations like reading files, making network requests, or interacting with databases occur.

Event-driven architecture enables the development of highly reactive and efficient systems. It promotes loose coupling between components, as they interact through events, making applications more modular and easier to maintain. Node.js's event-driven nature is a core factor in its ability to handle numerous concurrent connections and perform efficiently in real-time applications, chat systems, streaming services, and more.

# CHAPTER 4

**INTEGRATION OF MERN STACK**

## MongoDB Installation:

## Download MongoDB:

* Vist the MongoDB website and navigate to the download section.
* Choose the Community Server edition compatible with your Windows version.
* Download the MSI installer.

## Install MongoDB:

* Run the downloaded MSI installer.
* Follow the installation wizard's instructions.
* Choose the Complete installation type for a standard setup or select Custom to customize the installation path.
* During the setup, select "Install MongoDB as a Service" and check "Run service as Network Service user."
* Complete the installation process.

## Verify MongoDB Installation:

* Open Command Prompt and navigate to the MongoDB installation directory's
* bin folder.
* Run MongoDB --version to verify the installation and display the MongoDB version.
* To start the MongoDB service, run MongoDB.

## Node.js Installation:

## Download Node.js:

* Go to the Node.js website and download the Windows installer.
* Choose the LTS (Long-Term Support) version for stable releases.

## Install Node.js:

* Run the downloaded Node.js installer.
* Follow the installation wizard's instructions.
* Accept the license agreement and choose the installation location.
* Click "Next" through the remaining steps and complete the installation.

## Verify Node.js: Installation:

* Run the downloaded Node.js installer.Open Command Prompt and type node -v to check the Node.js version.
* Run **npm -v** to verify that **npm** (Node Package Manager) is installed along with Node.js.

## Express.js Installation:

## Create an Express.js Project:

* Open Command Prompt and create a directory for your Node.js/Express.js
* project: mkdir my-express-app.
* Move into the newly created directory: cd my-express-app.
* Initialize your Node.js project with npm: npm init -y. This creates a package.json file with default setting.

## Install Express.js:

* Run npm install express to install the Express.js framework as a dependency for your project.

## React.js Installation:

## Create an React.js Project:

* Open Command Prompt and install create-react-app globally if not already installed: npm install -g create-react-app.
* Create a new React app: npx create-react-app my-react-app. Replace my- reactapp with your desired project name.

## Start the React Development Server:

* Run **npm** start inside the React project directory to start the development server.
* Access the React app in a web browser by visiting ***http://localhost:3000***.

## Connecting MongoDB with Node.js/Express.js:

## Install Mongoose:

* Inside your Express.js project directory, install Mongoose, a MongoDB object modeling tool for Node.js: npm install mongoose.

1. **Set Up Database Connection**:
   * + Use Mongoose to establish a connection to MongoDB by configuring your Express.js application to connect to the MongoDB database.

## Building MongoDB with Node.js/Express.js:

## Backend Development with Express.js:

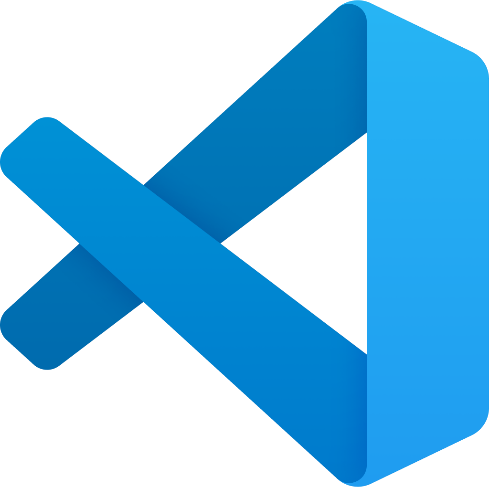
* Define routes and endpoints using Express.js to handle HTTP requests from the frontend or external clients.
* Implement middleware functions for request processing, authentication, data validation, etc.
* Create RESTful APIs for CRUD operations (Create, Read, Update, Delete) to interact with MongoDB.

## Frontend Development with Express.js:

* Develop the user interface and client-side components using React.js.
* Design and create React components that interact with the RESTful APIs built with Express.js to fetch and manipulate data from the backend.
* Implement state management, user interactions, and dynamic rendering using React's component-based architecture.

# CHAPTER 5 DEVELOPEMENT ENVIRNOMENT

## Introduction to Visual Studio Code

Visual Studio Code is a code editor in layman’s terms. Visual Studio Code is “a free-editor that helps the programmer write code, helps in debugging and corrects the code using the intelli-sense method”. In normal terms, it facilitates users to write the code in an easy manner. Many people say that it is half of an IDE and an editor, but the decision is up to the coders. Any program/software that we see, or use works on the code that runs in the background. Traditionally coding was used to do in the traditional editors or even in the basic editors like notepad! These editors used to provide basic support to the coders.

## What can Visual Studio Code can do?

Visual Studio Code has some incredibly unique features. They are listed as below: Support for multiple programming languages: Supports multiple programming languages. So earlier, programmers needed Web-Support: a different editor for different languages, but it has built-in multi-language support. This also means it easily detects if there’s any fault or cross-language reference, it’ll be able to detect it easily.

* + - * **Intelli-Sense:** It can detect if any snippet of code is left incomplete. Also, common variable syntaxes and variable declarations are made automatically.
      * **Cross-Platform Support**: Traditionally, editors used to support either Windows or Linux or Mac Systems. But Visual Studio Code is cross-platform. So, it can work on all three platforms.
      * **Extensions and Support:** Usually supports all the programming languages but, if the user/programmer wants to use the programming language which is not supported then, he can download the extension and use it.
      * **Repository:** With the ever-increasing demand for the code, secure and timely storage is equally important. It is connected with Git or can be connected with any other repository for pulling or saving the instances.
      * **Web-Support:** Comes with built-in support for Web applications. So web applications can be built and supported in VSC.
      * **Hierarchy Structure:** The code files are located in files and folders. The required code files also have some files, which may be required for other complex projects.
      * **Improving Code:** Some code snippets can be declared a bit differently, which might help the user in the code.
      * **Multi-Projects:** Multiple projects containing multiple files/folders can be opened simultaneously. These projects/folders might or might not be related to each other.

## Advantages of VSS

There are many advantages over any other IDE; they are as follow:

1. **Cross-platform support**: Windows ,Linux, Mac

### Lightweight 3.Robust Architecture

1. **Intelli-Sense**
2. **Freeware:** Free of Cost- probably the best feature of all for all the programmers out there, even more for the organizations.

## Introduction to GitHub

In the modern development landscape, collaboration and version control are crucial to building software efficiently. GitHub, a platform for hosting and collaborating on code, has become the go-to tool for developers worldwide. It provides a seamless way to manage code, track changes, and collaborate with other developers. With GitHub, both individuals and teams can work on projects, manage versions, and contribute to open-source software effortlessly.

## What is GitHub?

GitHub is a web-based platform that uses Git, a version control system, to manage and store code repositories. It allows developers to collaborate on projects from anywhere, track changes, and manage contributions in a structured and organized manner. GitHub is also home to millions of open-source projects and enables developers to contribute, fork, and learn from others' codebases. Beyond code hosting, GitHub provides additional features such as project management, CI/CD pipelines, and automated workflows.

## Key Features

### Version Control:

GitHub provides powerful version control using Git. This enables developers to track every change made to the code, revert to previous versions, and manage multiple versions or branches of a project simultaneously.

### Collaboration:

GitHub fosters collaboration through pull requests, forking, and issue tracking. Teams and contributors can propose changes, discuss issues, and manage project features efficiently.

### Code Hosting:

GitHub hosts repositories in the cloud, making them accessible from anywhere. Repositories can be public, allowing anyone to view or contribute to the code, or private, limiting access to specific collaborators.

1. **Project Management:**

Notebooks can be easily shared with others, promoting collaboration and reproducibility. They can be exported in various formats, including HTML, PDF, and slides.

1. **GitHub Pages:** GitHub Pages is a feature that allows developers to host static websites directly from a GitHub repository.

## Introduction to Microsoft Azure

In today’s cloud-first world, businesses and developers are constantly seeking platforms that offer scalability, reliability, and flexibility to host applications and manage data. Microsoft Azure is one such powerful cloud computing platform that provides a wide range of services, including computing, analytics, storage, and networking. It is widely used for building, deploying, and managing applications through Microsoft's global data center network.

## What is Azure?

Azure is a cloud computing platform and service created by Microsoft. It enables users to build, manage, and deploy applications on a massive global network. Azure supports various programming languages, frameworks, and tools, making it an ideal solution for both small startups and large enterprises. It offers an array of cloud services including virtual machines, databases, AI, and storage, which can be utilized to meet diverse business needs

## Key Features

### Scalability and Flexibility:

Azure is built to scale resources based on your application's needs. Whether you’re running a simple web app or a complex machine learning model, Azure can easily adjust resources dynamically, offering both vertical and horizontal scaling capabilities.

### Hybrid Cloud Support:

Azure stands out with its support for hybrid cloud environments, allowing businesses to manage resources across on-premises data centers and the cloud. This hybrid approach enables seamless integration and flexibility for organizations looking to maintain a mixture of cloud and on-premises infrastructure.

### AI and Machine Learning:

Azure offers a suite of AI and machine learning services like Azure Cognitive Services and Azure Machine Learning. These tools provide advanced capabilities like language processing, image recognition, and predictive analytics that can be integrated into applications easily.

### DevOps Integration:

Azure integrates seamlessly with DevOps tools, enabling continuous integration.

### Security and Compliance:

Azure places a strong emphasis on security, providing features like multi-factor authentication, encryption, and advanced threat detection. Additionally, it offers compliance certifications for a broad set of industry standards, making it a trusted platform for businesses handling sensitive data.

## Getting Started

To begin using Azure, you need to create an account on the Azure portal, where you can access a wide range of cloud services. Once you have an account, you can create resources, configure services, and deploy your applications.

### Create an Azure Account

Visit [Azure's official website](https://azure.microsoft.com) and sign up for a free account. You may get free credits to explore Azure services.:

1. **Install Azure CLI::**

To interact with Azure from the command line, install the Azure Command-Line Interface (CLI) with the following command:

**curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash**

### Login to Azure: After installation, log in to your Azure account by running:

This will open a browser where you can authenticate using your Azure credentials.

### az login

1. **Create Your First Resource:**

Use the Azure portal or the CLI to create a new resource, such as a virtual machine:

### az vm create --resource-group myResourceGroup --name myVM --image UbuntuLTS --generate-ssh-keys

1. **Deploy Applications:**

Azure offers several services to deploy applications

### Deploy via CI/CD:

For automated deployments, Azure integrates with **Azure DevOps** or third-party tools like **GitHub Actions**.

**az pipelines create --name myPipeline --repository-url <repo-url>**

# CHAPTER 6 SYSTEM DESGIN

## 6.1 Overview

## The ChoiceDekho Application aggregates and normalizes course offerings from multiple educational platforms (e.g., Coursera, Udemy, edX) and provides users with an easy-to-use interface. Users can search for courses and compare them based on essential factors such as price, duration, ratings, course content, helping them make informed decisions on the best learning options.

## 6.2 Goals & Objectives

## Data Aggregation:

## This involves scraping data from various educational platforms like Coursera, Udemy, and edX. The scraping process collects course information such as titles, prices, ratings, reviews, mode, descriptions, etc. This data is then normalized to ensure consistency, despite differences in how platforms present information.

## Comparison Engine:

## The comparison engine allows users to evaluate courses across multiple platforms based on a variety of factors such as pricing, course duration, content quality, ratings. The engine enables side-by-side comparisons, making it easier for users to identify which course best suits their learning needs, budget, or preferences.

## Filtering and Sorting:

## ChoiceDekho offers extensive filtering and sorting options to help users narrow down their choices. Filters may include course price range, duration, merchant, mode, ratings, affiliation and category. Users can also sort courses by price. This feature ensures that users can quickly find the most appropriate courses tailored to their specific criteria.

## User Experience:

## A seamless user interface is essential for comparison applications like ChoiceDekho. The platform focuses on intuitive navigation, clear presentation of data, and responsiveness across devices. By prioritizing a user-friendly design, the platform ensures that users can easily search for, compare and enhance their overall experience.

## 6.3. High-Level Architecture

## Frontend: Built with React.js for dynamic, responsive user interfaces using a modular component-based architecture.

## Backend: Uses Node.js with Express.js for the API layer, enabling scalable, non-blocking operations and efficient handling of user requests.

## Database: MongoDB stores flexible, varied course data, supporting efficient storage and retrieval for large datasets.

## External Services:

## Scraping: Directly scrapes course data from platforms like Coursera and Udemy for comprehensive listings.

## Proxy Service: Manages requests for efficient routing during scraping.

## Impact API: Pulls specific data to enhance course details.

## Data Mapping: Queue processes handle scraped raw data asynchronously, converting it into structured formats as the system scales.

## Caching Layer: Redis caches frequently accessed data, reducing database load and speeding up response times.

## 6.4 Data Flow

## User Request:

## User initiates a search for courses by keyword, subject, or merchant.

## Data Aggregation:

## The Course Aggregation Service queries multiple external platform APIs.

## The service normalizes course data (ensuring consistent formats for price, duration, ratings, etc.).

## Aggregated data is stored in a database.

## Comparison Logic:

## The user can apply filters such as price range, ratings, duration, and description.

## The Course Comparison Service fetches data from the database and performs the comparison logic.

## Response to User:

## The comparison results are displayed in the frontend, showing key details like price, ratings, and a link to the course on the respective platform.

## 6.5. Detailed Design

## 6.5.1 Backend

## Course Aggregation Service

## Technology: Python for web scraping, Node.js with Axios for API calls.

## Purpose: Responsible for scraping course data from third-party educational platforms and fetching impact data via APIs.

## Key Responsibilities:

## Scrape course data from platforms (e.g., Coursera, Udemy, edX) using Python and store it in the database.

## Normalize data formats (e.g., price in USD, duration in hours) for consistency.

## Fetch course impact data (e.g., ratings, reviews) using the impact API via Axios in Node.js.

## 6.5.2 Comparison Service

## Technology: Node.js

## Purpose: Handles all course comparison logic by fetching data from the database and performing comparisons based on the user’s selected courses.

## Key Responsibilities:

## Fetch aggregated data from the database.

## Compare courses based on factors such as:

## Price

## Ratings

## Duration

## Language

## Description

## Sort and filter courses according to user preferences (e.g., highest-rated, cheapest, shortest duration).

## 6.5.3 User Service

## Technology: Node.js, MongoDB

## Purpose: Manages user accounts, preferences, and saved comparisons.

## Key Responsibilities:

## User authentication and authorization

## Manage user profile (email, password, saved preferences)

## Allow users to save and retrieve comparisons

## 6.5.4 Caching Layer

## Technology: Redis

## Purpose: Reduces the load on the database and external APIs by caching frequently accessed data.

## Key Responsibilities:

## Store frequently accessed course data for fast retrieval.

## Cache search results for commonly queried courses and comparison results.

## Expire data at set intervals to ensure freshness

## 6.5.5 Frontend:

## Built with React.js, leveraging its modular, reusable components to ensure scalability and flexibility. This framework enhances the overall user experience with faster rendering and dynamic UI updates, delivering seamless interactions across devices.

## Search and Filters:

## Powerful search functionality with input fields for keywords, categories, and merchants, allowing users to easily find relevant courses.

## Advanced filtering options like price range, ratings, course duration, and specific merchants to tailor searches based on personal preferences.

## Sorting options enable users to organize results by lowest price, and high price, making the discovery process efficient.

## Comparison Interface:

## Clean, user-friendly interface that allows users to compare courses side-by-side, displaying crucial information like price, rating, duration, and key features.

## The comparison table ensures that users can make informed choices by highlighting differences and similarities between courses briefly, making the decision-making process faster and more efficient.

## Database Design

## Courses Table

## Stores normalized course data from all platforms.

## Fields: \_id, platform, title, price, duration, rating, review\_count, headlines, course\_type, duration\_per\_week, price, modules, mode, level, affiliation, affiliated, review\_count, website\_url, image\_url, prerequisite, tags, description, out\_come, instructor,status ,created\_at, updated\_at.

## Users Table

## Stores user data using a unified schema, applied across both the admin panel and client side of ChoiceDekho, ensuring consistency in user management and access control

## Fields: \_id, email, password, otp, googleId, role, verified, image, contactNo, address, employeeId, provider, dateOfJoin, dateOfBirth, gender, status, is\_deleted, created\_at, updated\_at.

## Favorites Table

## Stores results of favorite courses, allowing users to save and revisit them.

## Fields: userId, courseId, statu.s.

## 6.7. External Integrations

## 6.7.1Impact API Integration:

## The Impact API is integrated into the system to fetch additional data related to courses from multiple educational platforms. This API provides structured, verified data that supplements scraped information, improving accuracy and comprehensiveness.

## Key features of the integration include:

## Data Enrichment: The API provides additional metadata for courses like course ratings, reviews, or completion rates, enriching the data scraped from platforms.

## Resilience: If the Impact API fails temporarily, the system implements retry logic, ensuring a consistent data flow without significant interruptions.

## This integration enhances the quality and reliability of course data for better user experience.

## 6.8. Non-Functional Requirements

## Scalability:

## Efficient database indexing to handle large datasets of courses.

## Reliability:

## Use of retries for failed API calls to ensure data consistency.

## Performance:

## Caching frequently requested data using Redis to reduce database load and improve response times.

## A screenshot of a computer Description automatically generatedFig 5: Data Flow Diagram

# CHAPTER 7 SOURCE CODE

**7.1 Fronted code overview**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**7.2 Backend code overview**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A computer screen shot of a code

Description automatically generated**A screenshot of a computer program

Description automatically generated

**A screenshot of a computer code

Description automatically generated**

A computer screen shot of a computer code

Description automatically generated

**A screenshot of a computer program

Description automatically generated**

A close-up of a computer screen

Description automatically generatedA screenshot of a computer program

Description automatically generated

# CHAPTER 8 OUTPUT

A computer screen shot of a person sitting at a desk

Description automatically generated

A screenshot of a computer

Description automatically generated

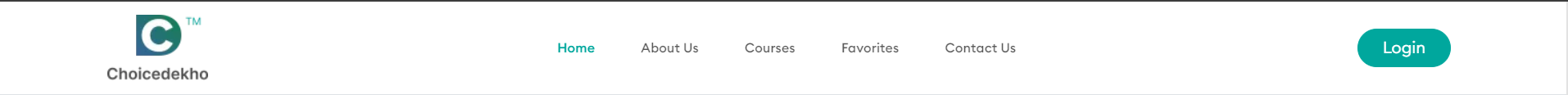
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated**A screenshot of a phone

Description automatically generated**

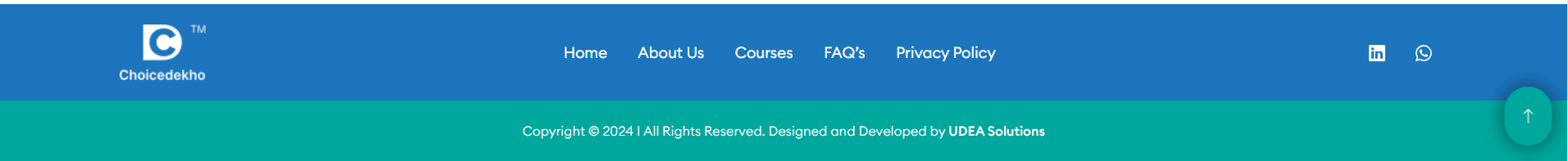


**A screenshot of a computer

Description automatically generated** A screenshot of a login form

Description automatically generated A screenshot of a person smiling

Description automatically generated



**Fig. 15 Output**

# CONCLUSION

ChoiceDekho serves as a groundbreaking solution for students and professionals seeking to enhance their educational journey by simplifying the course selection process. By intelligently scraping, normalizing, and categorizing data from diverse educational platforms, it eliminates the confusion and time-consuming task of manually comparing different offerings. The platform’s robust filtering and comparison capabilities empower users to customize their searches based on various criteria, such as course type, provider, pricing, duration, and more. Additionally, the system continuously updates its data to ensure users have access to the most current and accurate information.

Through its intuitive user interface, ChoiceDekho fosters an environment of informed decision-making, making it easier for individuals to choose courses that best fit their personal, academic, or professional goals. The platform not only saves time but also adds significant value by providing insights into user reviews, course content, and market demand. This comprehensive approach positions ChoiceDekho as a one-stop platform that transforms how people navigate the vast world of online education, driving better outcomes for learners by offering convenience, clarity, and confidence in their choices.

# BIBLIOGRPAHY

<https://www.mongodb.com/resources/languages/mern-stack>

<https://www.geeksforgeeks.org/mern-stack/>

<https://nodejs.org/docs/latest/api/>

<https://expressjs.com/en/4x/api.html>

<https://react.dev/reference/react>

<https://code.visualstudio.com/api/get-started/your-first-extension>

<https://docs.github.com/en/get-started>

<https://docs.npmjs.com/creating-a-new-npm-user-account>

**Live Project link**

<https://www.choicedekho.com/>