

# **INDIAN RAILWAY ACCIDENTS DASHBOARD**

## **An Industrial Project Report**

Submitted in fulfilment of the  
Requirement for the award of the degree of

**Master of Computer Applications**  
**(Batch 2021- 2024)**

**To**



**By**

**Riya Garg**  
**(52122107)**

**CRIS Practical Training**

**DEPARTMENT OF COMPUTER APPLICATIONS**  
**NATIONAL INSTITUTE OF TECHNOLOGY, KURUKSHETRA**  
**May/June/July 2024**

## COMPANY CERTIFICATE

क्रिश्न  
CRIS

रेलवे सूचना प्रणाली केन्द्र  
(रेल मंत्रालय भारत सरकार का संगठन)  
**CENTRE FOR RAILWAY INFORMATION SYSTEMS**  
(An Organisation of the Ministry of Railways, Govt. of India)

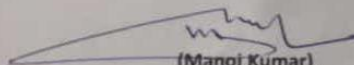


No. 2023/CRIS/NDLS-HQ/ESTAB/TRAINING/0000/4012/PT-I

Dt. 11.06.2024

### TO WHOMSO EVER IT MAY CONCERN

This is to certify that **Ms. Riya Garg** a student of **MCA of National Institute of Technology, Kurukshetra (Haryana)** has undergone practical training in the **CENTRE FOR RAILWAY INFORMATION SYSTEMS, CHANAKYAPURI, NEW DELHI-110021** from 25.01.2024 to 31.05.2024 under the guidance of **Mr. Sharad Kumar Gupta**, Principal Project Engineer of SIMS group.

  
(Manoj Kumar)  
Manager/Personnel

चाणक्यपुरी, नयी दिल्ली-110021  
CHANAKYAPURI, NEW DELHI-110021  
टेलीफोन/TELEPHONE : 24104525, 24106717 फैक्स/FAX : 91-11-26877893

## DECLARATION

### DECLARATION

I hereby declare that the work which is being presented in this project report entitled "*Indian Railway Accidents Dashboard*", in partial fulfilment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS** submitted to the Department of Computer Applications, National Institute of Technology, Kurukshetra is an authentic work done by me since **January 25, 2024** completing 16 weeks under the guidance of **Mr. Sharad Gupta** in **Centre for Railway Information Systems (CRIS)**.

The work presented in this project report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Signature

Name of the Candidate: Riya Garg

Roll no.: 52122107

**This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief. Further, during the above mentioned period he/she worked regularly.**

Date: 8/6/2024

Place: CRIS

Signature

Name of the Company Supervisor(s): Mr. Sharad Gupta  
/Mentor/Team Leader

Designation: PPE/SIMS.

## ACKNOWLEDGMENT

I would like to express my sincere gratitude to my supervisor, Mr. Sharad Gupta, for their continuous guidance and support throughout the development of the "Indian Railway Accidents Dashboard" project. Their invaluable insights and encouragement were instrumental in the successful completion of this project. The opportunity to learn from such an experienced mentor has been truly enriching.

I am also grateful to the Centre for Railway Information Systems (CRIS) for providing this practical training opportunity. The hands-on experience and resources made available by CRIS were crucial in bringing this project to fruition. This training not only allowed me to work on a real-world project but also helped me gain a deep understanding of several key technologies, including React.js, Oracle SQL, Node.js, Express.js, and Chart.js.

The knowledge and skills I acquired through this training have been invaluable. React.js enabled me to create dynamic and responsive user interfaces for the dashboard. Oracle SQL provided the foundation for managing and querying the extensive dataset efficiently. Node.js and Express.js facilitated the development of a robust and scalable server-side application, while Chart.js allowed me to visualize complex data in an intuitive and informative manner.

Looking ahead, I am excited about the prospect of enhancing the features and functionality of the dashboard, making it an even more powerful tool for analyzing and improving railway safety. The potential for further development and innovation in this project is immense, and I am eager to explore new possibilities.

Thank you to everyone who contributed to this project. Your support and collaboration have been deeply appreciated. This experience has been a significant milestone in my professional journey, and I am grateful for the opportunity to learn and grow through this project.

## **ABSTRACT**

The "Indian Railway Accidents Dashboard" project aims to enhance the current system of documenting and analyzing railway accidents. Traditionally, accident data has been maintained in tabular form, which poses significant challenges for visualization and quick comprehension. This project addresses these limitations by leveraging modern web technologies to transform raw data into intuitive and interactive visualizations.

Utilizing tools such as React.js for dynamic user interfaces, Oracle SQL for efficient data management, Node.js and Express.js for a robust server-side application, and Chart.js for powerful data visualization, we have developed a comprehensive dashboard that presents accident data in a more accessible and insightful manner. By converting complex datasets into charts and graphs, stakeholders can now easily identify trends, patterns, and areas of concern.

In summary, the "Indian Railway Accidents Dashboard" project transforms traditional tabular data into visually engaging and analytically useful charts, making railway accident data more comprehensible and actionable.

## **ABOUT COMPANY**

### **OVERVIEW**

The Centre for Railway Information Systems (CRIS) operates under the Ministry of Railways and combines the expertise of IT professionals with the practical knowledge of experienced Railway personnel. This unique synergy allows CRIS to deliver complex IT solutions for core Railway operations. Since its inception, CRIS has been developing and maintaining software for key functional areas within Indian Railways.

### **KEY FUNCTIONAL AREAS OF INDIAN RAILWAYS**

- **Ticketing & Passengers**

CRIS plays a crucial role in addressing the ticketing needs of approximately 20 million passengers daily, with the capability to book over 25,000 tickets per minute. Additionally, it manages over 200 million daily inquiries about train movements and schedules. In a country as vast as India, with one of the world's largest railway networks, managing public services like ticket reservations, seat availability information, and Tatkal bookings would be inconceivable without secure, modern IT solutions. Systems such as the Passenger Reservation System, Unreserved Ticketing System, and Next Generation Ticketing Systems exemplify CRIS's efficiency and performance, positioning it as a leader in providing Government to Citizen (G2C) services.

- **Freight Services**

Managing the movement of 3.5 million tonnes of freight daily, CRIS's systems contribute to over 343 crore INR in freight earnings each day. CRIS facilitates digital payments, achieving 98% cashless revenue collections, and oversees the monitoring of 280,000 wagons across India. The Freight Operation Information System forms the backbone of Indian Railways' freight business, digitizing freight bookings and revenue collection processes. The Rake Management System, deployed at over 8,000 locations, is integral to daily freight management and earnings, generating voluminous data that is a global case study in itself.

- **Operations**

CRIS oversees train traffic control and operations for every single train across the Indian Railways network. It manages coaches, crews, and other operations digitally, and employs satellite-based tracking for real-time train monitoring. Systems such as the Control Office Automation, Integrated Coaching Management System, and Crew Management System are crucial to the management of thousands of passenger and freight trains daily. Real-Time Train Information Systems enhance monitoring by providing precise train locations.

- **Asset Management**

CRIS's software solutions manage the extensive infrastructure of Indian Railways, including stations, workshops, sheds, and yards. They handle the inspection and maintenance of tracks, overhead lines, land, and signals, as well as the management of rolling stock assets like locomotives and rakes. Systems such as the Track Management System, Traction Distribution Management System, and Coaching Maintenance Management System support the lifecycle management of both rolling and non-rolling stock assets.

- **Human Resource & Accounting**

With a workforce of over 1.2 million employees, CRIS's Human Resource Management System and Integrated Payroll Accounting System handle the comprehensive HR management needs of Indian Railways. The digitalization of accounting and finance systems, including online expenditure booking and railway budgeting, is managed through solutions like the Accounting Information Management System and Rail Budget Compilation System.

- **Procurement & Automation**

CRIS has revolutionized procurement processes in Indian Railways through digital solutions like the Indian Railway e-Procurement System and Integrated Material Management & Information System, ensuring transparency and accountability. The adoption of these systems has significantly reduced average procurement times. Additionally, digital monitoring and information sharing with the public are facilitated through dashboard solutions like e-Drishti and RailDrishti.

## CRIS'S DIGITAL ENDEAVOURS

- **Technology And Innovation**

Between 2015 and 2020, CRIS has modernized its IT solutions by integrating emerging technologies such as Blockchain, AI, Machine Learning, Big Data, Business Intelligence, and Cloud Services. These technologies enhance train route optimization, capacity management, predictive maintenance, energy consumption forecasting, and timetable optimization.

- **Digitalization Through Mobile Apps**

CRIS has developed numerous mobile applications to enhance passenger convenience and internal operations. Public-centric apps like UTS, RailConnect, NTES, Rail Madad, and CoachMitra cover aspects from ticket and freight booking to complaint redressal. Internal apps like eDrishti, Aapoorti, and Chalak Dal support digital railway operations and monitoring. These apps are available as native iOS and Android applications, as well as hybrid and progressive web applications, complete with backend design, mobile analytics, and testing capabilities.



**Figure 1. CRIS Logo**



## LIST OF FIGURES

<b>Figure 1. CRIS Logo</b>	<b>VII</b>
<b>Figure 2. Oracle DB Setup</b>	<b>27</b>
<b>Figure 3. Import and Register Bar Chart</b>	<b>28</b>
<b>Figure 4. Pseudocode for Database.js</b>	<b>34</b>
<b>Figure 5. Pseudocode for Server.js</b>	<b>36</b>
<b>Figure 6. Pseudocode for App.jsx</b>	<b>41</b>
<b>Figure 7. Doughnut Chart</b>	<b>45</b>
<b>Figure 8: Pseudocode for Doughnut Chart</b>	<b>46</b>
<b>Figure 9. Line Chart</b>	<b>47</b>
<b>Figure 10: Pseudocode for Line Chart</b>	<b>48</b>
<b>Figure 11. Bar Chart</b>	<b>49</b>
<b>Figure 12: Pseudocode for Bar Chart</b>	<b>50</b>
<b>Figure 13: Pseudocode for AccidentTable.jsx</b>	<b>53</b>
<b>Figure 14: Snapshot of Indian Railway Accidents Dashboard Project</b>	<b>53</b>

## **TABLE OF CONTENTS**

<b>COMPANY CERTIFICATE</b>	<b>I</b>
<b>DECLARATION</b>	<b>II</b>
<b>ACKNOWLEDGEMENT</b>	<b>III</b>
<b>ABSTRACT</b>	<b>IV</b>
<b>ABOUT COMPANY</b>	<b>V</b>
<b>LIST OF FIGURES</b>	<b>VIII</b>
<b>CHAPTER 1 – INTRODUCTION</b>	<b>1-5</b>
<b>1.1 Introduction</b>	<b>1</b>
<b>1.2 Motivation</b>	<b>2</b>
<b>1.3 Organization of the Report</b>	<b>4</b>
<b>CHAPTER 2 – LITERATURE REVIEW</b>	<b>6-9</b>
<b>2.1 Existing System</b>	<b>6</b>
<b>2.2 Drawbacks of Previous System</b>	<b>6</b>
<b>2.3 Tools and Technology used in Previous Method</b>	<b>8</b>
<b>CHAPTER 3 – PROJECT OBJECTIVES</b>	<b>10-11</b>
<b>CHAPTER 4 – THE PROPOSED SOLUTION</b>	<b>12-16</b>
<b>4.1 Proposed System</b>	<b>12</b>
<b>4.2 Proposed System Requirements</b>	<b>14</b>
<b>4.3 Project Planning</b>	<b>15</b>

<b>CHAPTER 5 - IMPLEMENTATION DETAILS &amp; RESULTS</b>	<b>17-53</b>
<b>5.1 Tools and Technologies Used</b>	<b>17</b>
<b>5.2 Installation and Setup</b>	<b>23</b>
<b>5.3 Discussion on Implementation</b>	<b>30</b>
<b>CHAPTER 6 - CONCLUSION AND FUTURE SCOPE</b>	<b>54</b>
<b>6.1 Conclusion</b>	<b>54</b>
<b>6.2 Future Scope</b>	<b>54</b>
<b>REFERENCES</b>	<b>55</b>

# Chapter 1: INTRODUCTION

## 1.1 INTRODUCTION

In the current era of digital transformation, data visualization has become a cornerstone for effective data analysis and decision-making. The traditional approach of displaying data in tabular form, while accurate and detailed, often falls short in conveying complex information quickly and intuitively. This project, the Indian Railway Accidents Dashboard, aims to address these shortcomings by transforming traditional tabular data into interactive and insightful visualizations. This shift from tables to charts is not merely a trend but a necessary evolution to meet the demands of modern data users.

### 1.1.1 Drawbacks of Tables Over Charts

#### 1. *Difficulty in Identifying Trends*

Tables are inherently designed to list data in a structured, row-by-row format. While this is useful for detailed analysis and exact values, it does not lend itself well to the identification of trends or patterns. When dealing with large datasets, it becomes exceedingly difficult to discern overarching trends or anomalies that could be critical for decision-making. Charts, on the other hand, can immediately highlight these trends, making it easier to spot patterns and outliers that may indicate underlying issues or opportunities.

#### 2. *Limited Visual Appeal*

The visual appeal of data presentation is crucial in today's fast-paced information environment. Tables, being text-heavy and often monochromatic, fail to engage users visually. This lack of visual stimulation can lead to decreased user engagement and interest, ultimately affecting the efficacy of the data presented. In contrast, charts utilize colors, shapes, and spatial arrangements to create a more engaging and visually stimulating experience. This not only captures the user's attention but also aids in the retention and comprehension of the information.

#### 3. *Inefficiency in Comparative Analysis*

Comparing multiple data points across a table can be a cumbersome and time-consuming process. Users must scan through rows and columns, mentally calculating differences and similarities. This

manual process is prone to errors and can significantly slow down analysis. Charts, such as bar graphs or line charts, can present comparative data side-by-side or overlay it in a single view, making comparisons instantaneous and much more intuitive. This efficiency is particularly beneficial in time-sensitive decision-making scenarios.

#### 4. *Scalability Issues*

As the volume of data grows, tables become increasingly unwieldy and difficult to navigate. Large tables require extensive scrolling and can overwhelm users with sheer amounts of information. This scalability issue hampers the ability to quickly find and interpret relevant data. Charts, however, can scale more effectively. They can aggregate data, display summaries, and use interactive features like zooming and filtering to manage large datasets without losing clarity or usability.

## 1.2 MOTIVATION

The motivation behind the Indian Railway Accidents Dashboard project is rooted in the need to overcome these inherent limitations of tabular data presentations. By adopting modern visualization techniques, we aim to enhance the usability, engagement, and analytical capabilities of the data.

### **Enhanced Data Interpretation**

Charts transform raw data into visual narratives. For instance, a line chart depicting the number of accidents over time can immediately reveal trends such as increases or decreases in accident frequency. A pie chart can quickly show the proportion of different types of accidents. These visual narratives are not only easier to understand but also more impactful, allowing users to derive insights at a glance. Enhanced data interpretation means better, faster decision-making, which is crucial in critical areas like railway safety.

### **Improved User Experience**

A user-friendly interface is paramount for effective data analysis. Charts provide a visually engaging way to interact with data. Interactive elements like tooltips, clickable legends, and zoomable areas add to the user experience, making data exploration not just easier but also more

enjoyable. This improved user experience encourages more frequent and thorough analysis, leading to more informed and proactive decisions.

### **Efficient Comparative Analysis**

Visualization tools enable users to compare multiple datasets effortlessly. For example, comparing accident data across different regions or time periods becomes straightforward with side-by-side bar charts or layered line charts. This capability is essential for identifying areas that require attention, assessing the effectiveness of safety measures, and planning future initiatives. Efficient comparative analysis drives more strategic and impactful actions.

### **Scalability and Clarity**

Charts handle large datasets by summarizing and aggregating information. They can display high-level summaries while allowing users to drill down into more detailed views as needed. This scalability ensures that as data grows, the dashboard remains manageable and effective. Features like interactive filters and dynamic updates maintain clarity and relevance, making it easier to stay focused on the most critical information.

### 1.3 ORGANISATION OF THE REPORT

**Chapter-1 Introduction:** The introduction section introduces the Indian Railway Accidents Dashboard project, emphasizing the importance of data visualization for effective decision-making. It discusses the drawbacks of traditional tabular data presentations and explains the motivation behind adopting interactive visualizations. The project aims to enhance data interpretation, improve user experience, enable efficient comparative analysis, and maintain clarity and scalability.

**Chapter-2 Literature Review:** The literature review reviews the existing system used for managing Indian railway accident data, which relies on traditional tabular formats and outdated technologies such as the Struts2 framework. The primary drawbacks of this system include a lack of visualization, which hampers the ability to quickly interpret complex datasets, and the use of outdated technologies that limit performance and scalability. The chapter highlights the need for upgrading to modern web technologies like React.js, Node.js, and Express.js to provide a more dynamic, responsive, and interactive user experience. It also details the limitations of the tools and technologies used in the previous method, emphasizing the need for an enhanced system that can better meet the demands of contemporary data analysis and visualization.

**Chapter-3 Project Objectives:** The objectives of the Indian Railway Accidents Dashboard project include replacing tables with charts for better visualization, displaying various data categories for improved decision-making, and creating a user-friendly interface. The project also aims to add search capabilities, ensure up-to-date information with easy CSV imports, integrate modern technologies, provide clear error handling, and plan for future deployment on a scalable hosting platform like AWS, Heroku, or Azure.

**Chapter-4 The Proposed Solution:** This chapter outlines the proposed solution for the Indian Railway Accident Dashboard system. It describes a modern dashboard that uses date pickers and various charts to display railway accident data, replacing traditional tables with interactive visualizations. The system requirements include ReactJS for the front end, Node.js and Express.js for the backend, Oracle DB for data storage, and Visual Studio Code for development. The project planning section details steps such as defining objectives, identifying stakeholders, determining

project scope, selecting technologies, planning deployment, and establishing monitoring mechanisms. This plan ensures the project is well-structured and efficiently managed.

**Chapter-5 Implementation Details and Results:** This chapter covers setting up the backend with Node.js and Express, configuring database connections, and defining endpoints to fetch accident data based on various criteria. On the frontend, React.js is used to visualize the data using different chart components like doughnut, line, bar, and pie charts. Additionally, a table component is introduced to display accident details in a structured format. Overall, Chapter 5 focuses on turning the application concept into a functional reality, facilitating data visualization and analysis for railway accidents.

**Chapter-6 Conclusion and Future Scope:** The conclusion of this project highlights its success in implementing a robust system for visualizing railway accident data, which significantly enhances decision-making processes and user experience through intuitive visualizations. In the future, the project aims to expand its scope by incorporating additional data tables from the database and utilizing SQL joins to extract deeper insights. This expansion will enable more comprehensive analysis and provide stakeholders with enhanced tools for improving railway safety and accident prevention.



## **Chapter 2: LITERATURE REVIEW**

### **2.1 EXISTING SYSTEM**

The existing system for managing Indian railway accident data relies heavily on traditional tabular formats and older technologies, such as the Struts2 framework. Struts2, a popular Java-based web application framework, has been the backbone of many enterprise-level applications, including those used by Indian Railways. This framework, along with Java and Oracle SQL, has provided a solid foundation for developing and maintaining the current system. Data is stored in Oracle SQL databases and presented to users in tabular formats through web interfaces built using Struts2.

In this setup, data is typically entered manually or imported into the database and then displayed in tables. Users interact with these tables to view accident reports, analyze data, and generate necessary documentation. While this approach has served its purpose over the years, it comes with significant limitations, especially when dealing with large datasets and the need for quick, insightful analysis.

### **2.2 DRAWBACKS OF THE PREVIOUS SYSTEM**

#### **Lack of Visualization**

One of the primary drawbacks of the previous system is its reliance on tabular data presentation. Tables, while useful for detailed data inspection, are not ideal for visualizing trends, patterns, and anomalies. They require users to manually sift through rows and columns, which can be time-consuming and prone to errors, especially when dealing with large volumes of data. This lack of visualization hampers the ability to quickly understand and interpret complex datasets, making it difficult to make informed decisions.

Tables are particularly inadequate for providing a quick overview of data. For instance, identifying a spike in accident occurrences over a specific period requires scanning multiple rows and columns, a process that can be both tedious and error-prone. Visual tools like graphs and charts can instantly highlight such trends, allowing for faster recognition and response to potential issues. The absence of such visual tools in the existing system means that critical insights are often buried in the data, delaying actionable intelligence.

### **Outdated Technologies**

The use of older technologies like the Struts2 framework further compounds the limitations of the existing system. While Struts2 was once a leading technology for building Java-based web applications, it has been surpassed by newer frameworks that offer improved performance, scalability, and ease of development. Modern web technologies such as React.js, Node.js, and Express.js provide more dynamic, responsive, and interactive user experiences. These new technologies support real-time data updates, seamless integration with various data sources, and enhanced user interfaces, all of which are essential for modern data visualization and analysis.

Moreover, outdated technologies are often more challenging to maintain and extend. As software development practices evolve, newer tools and frameworks offer better ways to structure code, manage dependencies, and ensure security. Clinging to older technologies can result in a system that is not only less efficient but also more vulnerable to security risks and harder to upgrade. The lack of support for modern web standards in older frameworks like Struts2 means that the system cannot easily adopt new features or improvements, further hindering its effectiveness and adaptability.

### **Need for Upgradation**

The fast-paced evolution of technology means that systems built on older frameworks need to be updated to stay relevant and efficient. The current system's reliance on legacy technologies limits its ability to integrate with modern tools and platforms, reducing its effectiveness and scalability. Upgrading to newer technologies can significantly enhance the system's capabilities, providing users with more powerful tools for data analysis and visualization, improving overall efficiency and decision-making processes.

## **2.3 TOOLS AND TECHNOLOGIES USED IN THE PREVIOUS METHOD**

### **Struts2 Framework**

Struts2, an extension of the Apache Struts framework, is designed for building Java EE web applications. It follows the Model-View-Controller (MVC) design pattern, which separates the application's data model, user interface, and control logic. Struts2 has been widely used for its ability to streamline the development process and maintain a clear separation of concerns. However, its static nature and limited support for modern web standards and features have become significant drawbacks in the face of newer, more dynamic frameworks.

### **Java**

Java, a versatile and powerful programming language, has been the cornerstone of many enterprise applications, including the Indian Railways' accident management system. Java's platform independence, robustness, and extensive library support have made it a preferred choice for backend development. Despite its strengths, Java-based web applications, particularly those built using older frameworks like Struts2, often lack the dynamic and responsive features found in modern web technologies.

### **Oracle SQL**

Oracle SQL has been the primary database management system for storing and managing railway accident data. Known for its reliability, scalability, and comprehensive feature set, Oracle SQL provides robust support for complex queries and data manipulation. While it remains a powerful tool for database management, the integration of advanced visualization tools and real-time data processing requires more flexible and interactive database solutions that can seamlessly connect with modern web technologies.

### **Traditional Tabular System**

The traditional tabular system used in the existing setup presents data in rows and columns, a format that is straightforward but limited in its ability to convey complex information effectively. Users must manually analyze the data, which is inefficient and error-prone. This approach lacks the interactive and visual elements needed to quickly identify trends, correlations, and anomalies, making it less effective for comprehensive data analysis and decision-making.

By understanding the limitations of the existing system and the advantages of modern technologies, we can better appreciate the need for upgrading the Indian Railway Accidents Dashboard. This upgrade will not only address the current system's deficiencies but also pave the way for a more efficient, scalable, and user-friendly solution that meets the evolving needs of data analysis and visualization in the railway industry.

## **CHAPTER 3: PROJECT OBJECTIVES**

The Indian Railway Accidents Dashboard project aims to revolutionize the way railway accident data is presented and analyzed. The primary objectives of this project are outlined below, focusing on enhancing user convenience, improving decision-making capabilities, and integrating modern technologies into the existing system. The objectives of the project are:

### **1. Replacing Tables with Graphs and Charts for Visualizations**

One of the key objectives is to replace traditional tabular data presentations with dynamic graphs and charts. This shift aims to enhance user convenience by making complex data easier to understand at a glance. Visualizations can highlight trends, patterns, and anomalies more effectively than rows and columns, providing users with immediate insights and a clearer understanding of the data.

### **2. Displaying Different Categories of Railway Data Through Visualizations**

The project aims to display various categories of railway data through diverse visualizations. By categorizing data and using specific charts and graphs for each category, the dashboard will facilitate better decision-making. Users can quickly compare different data sets, identify critical areas that need attention, and assess the effectiveness of implemented measures.

### **3. Providing a User-Friendly and Comprehensive Application**

To ensure the application is user-friendly and comprehensive, the project will utilize separate components and appropriate styling. Each component will be designed to handle specific functions, making the application modular and easier to maintain. Thoughtful styling will enhance the user experience, making the interface intuitive and engaging.

### **4. Search Capabilities for Dynamic Data Display**

The dashboard will include robust search capabilities, allowing users to search for data within specified date ranges. This feature will dynamically display the relevant data based on the search criteria, enabling users to perform targeted analysis and retrieve specific information quickly.

## **5. Providing Up-to-Date Information**

Keeping the railway data current is a crucial objective. The system will support easy importation of new data from CSV files into the database. Once the import is completed, the dashboard will automatically update to reflect the latest information, ensuring users always have access to the most recent data.

## **6. Integrating New Technologies**

The project aims to integrate modern technologies into the existing system to enhance performance, scalability, and user experience. By leveraging contemporary tools and frameworks, the system will become more efficient, responsive, and capable of handling complex data visualization tasks.

## **7. Error Handling and Maintenance**

Effective error handling and maintenance are critical for the system's reliability. The project will implement clear error messages to assist developers in troubleshooting and resolving issues promptly. This will ensure the system remains robust and operational with minimal downtime.

## **8. Deployment**

The deployment of the system will be on a suitable hosting platform such as AWS, Heroku, or Azure. These platforms will ensure the system is scalable, performs efficiently, and remains highly available. The choice of hosting platform will depend on factors like cost, scalability needs, and specific project requirements.

These objectives collectively aim to transform the Indian Railway Accidents Dashboard into a powerful, user-friendly tool that enhances data analysis and decision-making capabilities for railway safety and management.

## CHAPTER 4: THE PROPOSED SOLUTION

### 4.1 PROPOSED SYSTEM

The proposed Indian Railway Accidents Dashboard is designed to offer a more intuitive and efficient way to analyze railway accident data. By incorporating modern visualization techniques, the dashboard aims to enhance user engagement, improve decision-making processes, and provide a comprehensive tool for railway safety management.

The system consists of three main sections: Date Picker, Charts or Graphs, and Table.

#### 4.1.1 Date Picker:

The dashboard includes two date pickers that allow users to select a specific date range for which they want to view accident data. This feature ensures that users can filter the data according to their requirements, providing a focused and relevant dataset for analysis.

#### 4.1.2 Charts or Graphs:

The dashboard utilizes different types of charts to present data in a visually appealing and comprehensible manner. The visualizations are divided into various categories as detailed below:

##### *1. Railway:*

This section displays the number of accidents in each railway zone, such as Central Railway (CR), Eastern Railway (ER), South Central Railway (SC), etc. The data is represented through bar charts or pie charts, providing a clear comparison of accident frequency across different railway zones.

2. *Division:*

Accidents are displayed division-wise, using station codes like BSP (Bilaspur Junction), ALD (Allahabad Junction), and PGT (Palakkad Junction). This visualization helps in understanding the distribution of accidents across different divisions, making it easier to identify high-risk areas.

3. *Location of Accident:*

This category shows the number of accidents based on their location, such as yard, mid-section, and at stations. Pie charts or bar charts illustrate the accident distribution, aiding in pinpointing specific areas where accidents are more frequent.

4. *Accident Category:*

Accidents are categorized into types like consequential accidents, unusual accidents, yard accidents, etc. This visualization helps in understanding the nature and severity of accidents, providing insights into areas that need more attention.

5. *Accident Code:*

Accidents are displayed according to specific codes like D3, C4, P1, D6, D4, etc. This chart helps in identifying and analyzing different types of accidents based on standardized accident codes.

6. *Type of Accident:*

This section categorizes accidents based on their type, such as derailment, casualty, breach of block rules, fire, collision, etc. The visualization assists in understanding the common causes and types of accidents, facilitating targeted preventive measures.



## 7. District:

Accidents are displayed district-wise, such as in Rohtak, Jaipur, Agra, etc. This geographical visualization helps in identifying districts with higher accident rates, allowing for focused safety measures and resource allocation.

### 4.1.3 Table:

In addition to visualizations, the dashboard also includes a table that lists detailed information about accidents within the selected date range. Major columns in the table include accident ID, accident description, district, type of accident, relief description, and more. This tabular representation complements the visual data, providing a detailed and comprehensive view of each accident.

## 4.2 PROPOSED SYSTEM REQUIREMENTS

1. **Operating System Compatibility:** The system must seamlessly integrate with the preferred operating system, such as Windows.
2. **Web Development Framework:** To construct the user interface, the project utilizes the ReactJS web development framework, given its suitability for web-based applications.
3. **Backend Framework:** Node.js serves as the backend framework, responsible for managing server-side logic and data processing. Additionally, Express.js facilitates routing through HTTP GET methods and URL patterns.
4. **Database Management System:** Our project relies on Oracle DB for the storage and management of accident-related data.
5. **Server Environment:** For future hosting and deployment, a server environment like Apache or Nginx will be necessary to ensure optimal performance.
6. **Development Tools and IDEs:** Visual Studio Code serves as the integrated development environment (IDE) for coding and debugging purposes, enhancing development efficiency and workflow management.

## 4.3 PROJECT PLANNING

The project planning for the Indian Railway Accident Dashboard System revolves around utilizing JavaScript frameworks for both client-side and server-side development. The following steps outline the strategic planning process:

### 1. Define Project Objectives:

The primary focus is to define clear objectives and goals for the system, emphasizing the provision of a user-friendly interface and the effective display of different accident categories.

### 2. Identify Stakeholders:

Key stakeholders, including CRIS administrators and the safety management department, are identified to ensure their involvement and alignment with project objectives.

### 3. Scope Definition:

The scope of the project is defined by outlining the functionalities and features to be included, such as selecting and validating date ranges, displaying category-wise data, and presenting a detailed table view.

### 4. Technology Selection:

ReactJS is chosen as the client-side framework due to its component-based architecture, virtual DOM, and efficient rendering capabilities. Node.js is selected as the server-side runtime environment for its scalability and asynchronous programming features, with Express.js for routing. Additionally, Chart.js is utilized for chart and graph visualization.

### 5. Deployment and Hosting:

Planning for system deployment involves considering hosting requirements for both the ReactJS front-end and Node.js back-end, ensuring compatibility with existing system requirements.

Suitable hosting providers are evaluated, and infrastructure configuration is planned accordingly.

## **6. Project Monitoring and Control:**

Mechanisms for monitoring and controlling project progress are established to track milestones, manage resources, and address any deviations from the project plan. Regular checkpoints and feedback loops ensure project alignment with objectives and stakeholder expectations.

## **CHAPTER 5: IMPLEMENTATION DETAILS & RESULTS**

### **5.1 TOOLS AND TECHNOLOGIES USED**

#### **5.1.1 ReactJS**

ReactJS, a robust JavaScript library, was chosen for developing the client-side of the Indian Railway Accidents Dashboard project. ReactJS is renowned for its component-based architecture, which allows developers to create reusable UI components that can be easily integrated and maintained. This modularity facilitates better code organization and promotes reusability. ReactJS employs a virtual DOM (Document Object Model) that ensures efficient and fast updates to the user interface, making the application highly responsive and dynamic.

React's declarative nature simplifies the process of building interactive UIs. By focusing on what the UI should look like, developers can effectively manage the state and render components based on that state. This one-way data flow ensures that the application's state is predictable and easier to debug. The extensive ecosystem of React, including libraries and tools, significantly accelerates the development process. For instance, react-router-dom is utilized for navigating between different pages within the application, providing a seamless user experience with features like dynamic routing and route parameters.

#### **5.1.2 Material-UI (UI Library)**

Material-UI was integrated into the project to enhance the visual design and user experience of the dashboard. This React-based UI library follows Google's Material Design principles, offering a comprehensive collection of pre-built and customizable components. These components include grids, boxes, buttons, and date pickers, which are essential for building a consistent and appealing user interface.

Material-UI's grid system allows for responsive layout design, ensuring the dashboard is accessible and visually coherent across various devices and screen sizes. The theming

capabilities of Material-UI enable easy customization of the application's appearance, aligning it with the branding guidelines of the Indian Railways. This adaptability not only improves aesthetics but also enhances user engagement and satisfaction by providing a polished and professional interface.

### **5.1.3 Node.js**

Node.js serves as the backbone for the server-side development of the Indian Railway Accidents Dashboard project. As an open-source JavaScript runtime environment, Node.js allows the execution of JavaScript code on the server side. Its event-driven, non-blocking I/O model makes it lightweight and efficient, perfect for handling multiple simultaneous connections with high throughput.

Node.js operates on a single-threaded event loop, enabling it to manage numerous concurrent operations without the overhead associated with traditional multi-threaded server architectures. This model is particularly beneficial for real-time applications, such as the railway dashboard, where rapid data processing and updates are crucial. Furthermore, Node.js boasts a rich ecosystem of libraries and modules through npm (Node Package Manager), which accelerates development by providing reusable code for various functionalities.

### **5.1.4 Express.js**

Express.js, a minimalist web application framework for Node.js, was chosen for developing the backend logic and RESTful APIs of the project. Express simplifies server-side development by providing a flexible and intuitive framework that supports middleware architecture. This architecture allows for efficient handling of HTTP requests and responses, routing, and error handling.

Express's middleware capabilities enable the project to incorporate various functionalities, such as authentication, logging, and validation, in a modular fashion. By abstracting the complexities of server-side operations, Express helps in building scalable and maintainable

backend code. Its extensive community support and comprehensive documentation make it an ideal choice for developing robust web applications.

### **5.1.5 Cors**

Cors, or Cross-Origin Resource Sharing, is a critical middleware used in the project to manage cross-origin HTTP requests. Given that the frontend and backend servers may operate on different domains, Cors ensures secure and seamless communication between them. It defines and enforces access control policies that specify which domains are allowed to access resources on the server, thus preventing unauthorized access and safeguarding sensitive data.

Cors is highly configurable, allowing developers to specify allowed origins, methods, and headers. This flexibility ensures that the dashboard can interact securely with other services and APIs, maintaining the integrity and confidentiality of the data being exchanged.

### **5.1.6 Oracle DB**

Oracle Database is employed as the primary database management system for storing and managing the railway accident data. Oracle DB is known for its reliability, scalability, and robust performance in handling large volumes of data. It uses SQL (Structured Query Language) for querying and managing data, which is a powerful and standardized language supported by many database systems.

The choice of Oracle DB ensures that the data is stored securely and can be accessed and manipulated efficiently. Its advanced features, such as data partitioning, indexing, and transaction management, provide the necessary tools to handle complex queries and maintain data integrity. Oracle DB also offers strong support for backup and recovery, ensuring the safety of critical data.

### 5.1.7 Chart.js

Chart.js is a powerful and flexible JavaScript library used for creating interactive and visually appealing data visualizations on the web. It provides a straightforward API for generating a variety of chart types, making it an excellent choice for the Indian Railway Accidents Dashboard project. By leveraging Chart.js, we can transform raw data into meaningful insights through dynamic and engaging visualizations. Below is an in-depth look at the different types of charts used in the project:

#### *Pie chart*

A pie chart is a circular chart divided into sectors, each representing a proportion of the total. It is particularly useful for showing the relative sizes of parts to a whole. In the context of the railway dashboard, pie charts can be employed to display the distribution of accidents across different categories, such as accident types or accident locations.

- Representation: Each slice of the pie represents a category of data. The size of each slice is proportional to the quantity it represents.
- Use Case: Displaying the proportion of different accident types (e.g., derailments, collisions) or accident locations (e.g., yards, stations).
- Benefits: Provides a clear visual comparison of the parts of a whole, making it easy to see which category is the largest or smallest at a glance.

#### *Bar chart*

A bar chart presents data with rectangular bars with lengths proportional to the values they represent. Bar charts are ideal for comparing quantities across different categories.

- Representation: Each bar represents a category of data, and the height or length of the bar is proportional to the data value.
- Use Case: Displaying the number of accidents per railway division or per accident type.

- Benefits: Makes it easy to compare different categories side-by-side. Vertical bar charts are useful for comparing discrete categories, while horizontal bar charts can be used for larger sets of categories.

### *Line chart*

A line chart connects data points with a continuous line, making it useful for displaying data trends over time. This type of chart is ideal for visualizing changes in data at regular intervals.

- Representation: Data points are plotted and connected by straight lines.
- Use Case: Displaying the trend of accidents over a specific time period, showing increases or decreases in accident occurrences.
- Benefits: Effective for illustrating trends and patterns over time, making it easy to identify rises, falls, and cyclical trends in the data.

### *Doughnut Chart*

A doughnut chart is similar to a pie chart but with a blank center, giving it a "doughnut" shape. This chart type is also useful for showing proportions of a whole but allows for additional design flexibility and can be more visually appealing in some contexts.

- Representation: Similar to a pie chart, but with a hole in the center. This center space can be used to display additional information or emphasize the proportions.
- Use Case: Showing the distribution of accident types or locations, similar to a pie chart, but with a different visual style.
- Benefits: Provides a variation on the pie chart that can be aesthetically pleasing and can offer space for additional information or labels in the center.

### *Detailed Benefits and Features of Chart.js*

- Customization and Styling:  
Chart.js provides extensive customization options for all chart types. Colors, fonts, and



labels can be easily adjusted to match the branding and visual guidelines of the Indian Railway Accidents Dashboard. This ensures that the charts are not only informative but also visually cohesive with the overall design of the dashboard.

- Interactivity:

Charts created with Chart.js are interactive by default. Users can hover over data points to see detailed tooltips, click on segments to isolate data, and even interact with legends to toggle data visibility. This interactivity enhances user engagement and makes it easier for users to explore and understand the data.

- Responsive Design:

Chart.js supports responsive design, meaning charts automatically resize and adjust to fit different screen sizes and resolutions. This is crucial for ensuring that the dashboard is accessible and functional on various devices, from desktop computers to tablets and smartphones.

- Animations:

Chart.js includes built-in animations that make the transition and presentation of data smooth and visually appealing. These animations help draw users' attention to important changes and trends in the data, making the dashboard more engaging and dynamic.

- Ease of Integration:

Chart.js integrates seamlessly with ReactJS, making it straightforward to incorporate charts into the React components of the dashboard. This integration allows for the efficient rendering of charts and dynamic updates based on the application's state and user interactions.

- Performance:

Despite the rich features and interactivity, Chart.js is optimized for performance. It is designed to handle large datasets efficiently, ensuring that the charts load quickly and perform smoothly even with complex data.

## 5.2 INSTALLATION AND SETUP

This section outlines the installation and setup process for the various tools and technologies used in the Indian Railway Accidents Dashboard project.

### 5.2.1 ReactJS Using Vite

Vite is a modern frontend build tool that offers fast development and build speed. It is particularly useful for React projects due to its performance advantages.

- Install Node.js: Ensure that Node.js is installed on your system. You can download it from Node.js.
- Create a Vite Project:

```
npm create vite@latest indian-railway-dashboard
```

- Navigate to the Project Directory:

```
cd indian-railway-dashboard
```

- Install Dependencies:

```
npm install
```

- Start the Development Server:

```
npm run dev
```

- Install react-router-dom: Run the following command in your terminal at the root of your React project:

```
npm install react-router-dom
```

- Set up routes using BrowserRouter and Route components from react-router-dom.

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
```

### 5.2.2 Material-UI

Material-UI is used for implementing the UI components of the dashboard. It provides pre-built components like Grid, Table, Box, and Date picker that follow Material Design guidelines.

- Install Material-UI:

```
npm install @mui/material @emotion/react @emotion/styled
```

- Import and use the Grid component to create responsive layouts:

```
import Grid from '@mui/material/Grid';
```

- Import and use the Box component:

```
import Box from '@mui/material/Box';
```

- Import and use the Table component to display tabular data:

```
import Table from '@mui/material/Table';  
import TableBody from '@mui/material/TableBody';  
import TableCell from '@mui/material/TableCell';  
import TableContainer from '@mui/material/TableContainer';  
import TableHead from '@mui/material/TableHead';  
import TableRow from '@mui/material/TableRow';  
import Paper from '@mui/material/Paper';
```

- Install date picker and dayjs adapter:

```
npm i @mui/x-date-pickers dayjs
```

- Import and use date picker component:

```
import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';  
import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs';  
import { DatePicker } from '@mui/x-date-pickers/DatePicker';  
import dayjs from 'dayjs';
```

### 5.2.3 Node.js

Node.js is used for server-side development, providing a runtime environment for executing JavaScript on the server.

- Install Node.js: Download and install Node.js from [Node.js](https://nodejs.org/).
- Initialize a Node.js Project:

```
npm init -y
```

### 5.2.4 Express.js

Express.js is a minimal and flexible web application framework for Node.js, providing a robust set of features for web and mobile applications.

- Install Express.js:

```
npm install express
```

- Create a file server.js and set up a basic Express server:

```
import express from "express";  
const app = express();
```

### 5.2.5 Cors

Cors (Cross-Origin Resource Sharing) is middleware used to enable communication between different domains.

- Install Cors:

```
npm install cors
```

- Import and use Cors in your server.js:

```
import cors from 'cors';  
app.use(cors());
```

### 5.2.6 Oracle DB

Oracle Database is used for storing and managing data related to railway accidents.

- Install Oracle Client: Download and install the Oracle Database client from the [Oracle website](#).
- Install Node.js Oracle DB Package:

```
npm install oracledb
```

- Import the oracledb module, set the output format to OUT\_FORMAT\_OBJECT, and define an asynchronous function initPool that creates a connection pool with the specified configuration:

```

import oracledb from "oracledb";
oracledb.outFormat = oracledb.OUT_FORMAT_OBJECT;

let pool;
async function initPool() {
  const config = {
    user: "myuser",
    password: "password",
    connectString: "localhost:1521/XEPDB1",
    poolIncrement: 0,
    poolMax: 4,
    poolMin: 4,
  };
  try {
    pool = await oracledb.createPool(config);
    console.log("Connection pool created successfully");
  } catch (err) {
    console.error("Error creating connection pool", err.message);
  }
}

```

Figure 2. Oracle DB Setup

### 5.2.7 Chart.js

Chart.js is used for creating interactive data visualizations.

- Install Chart.js:

```
npm install react-chartjs-2 chart.js
```

- Import and register bar chart:

```
import { Bar } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  BarElement,
  Title,
  Tooltip,
  Legend
} from 'chart.js';

ChartJS.register(
  CategoryScale,
  LinearScale,
  BarElement,
  Title,
  Tooltip,
  Legend
);
```

**Figure 3. Import and Register Bar Chart**

- Import and register line chart:

```
import { Line } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  LineElement,
  Title,
  Tooltip,
  Legend
} from 'chart.js';

ChartJS.register(
  CategoryScale,
  LinearScale,
  LineElement,
  Title,
  Tooltip,
  Legend
);
```

- Import and register doughnut chart:

```
import { Doughnut } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  Title,
  Tooltip,
  Legend,
  ArcElement
} from 'chart.js';

ChartJS.register(
  Title,
  Tooltip,
  Legend,
  ArcElement
);
```

- Import and register pie chart:

```
import { Pie } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  Title,
  Tooltip,
  Legend,
  ArcElement
} from 'chart.js';

ChartJS.register(
  Title,
  Tooltip,
  Legend,
  ArcElement
);
```



## 5.3 DISCUSSION ON IMPLEMENTATION

In this section, we delve into the detailed implementation of the Indian Railway Accidents Dashboard system. The implementation discussion is divided into two key sections: Backend Server and Client-Side. Through these sections, we outline the intricacies of both the server-side and client-side aspects of the system, shedding light on the technologies used, the structure of the codebase, and the functionalities they enable. Let's explore how each component contributes to the overall functionality and effectiveness of the dashboard.

### 5.3.1 Backend Implementation

The backend of the Indian Railway Accidents Dashboard system is structured using Node.js and Oracle Database. Two main files to handle the backend are: `database.js` and `server.js`.

#### *Database.js*

`database.js` is designed to handle all interactions with an Oracle database using the `oracledb` library. It performs the following primary functions:

1. Initialization of Connection Pool: It sets up a connection pool to efficiently manage multiple database connections.
2. Connection Creation: It provides a mechanism to create new database connections from the pool.
3. Data Retrieval Functions: It includes several functions to execute specific SQL queries and return the results, such as fetching accident counts based on various criteria like railway, division, location, and type.

- Pseudocode for database.js

Here is the pseudocode that outlines the key operations performed in `database.js`:

```
// Import the oracledb module
import oracledb from "oracledb";

// Set the output format to object
oracledb.outFormat = oracledb.OUT_FORMAT_OBJECT;
```

```
// Initialize the connection pool
let pool
async function initPool(){
  // Database configuration
  const config = {
    user: "myuser",
    password: "password",
    connectString: "localhost:1521/XEPDB1",
    poolIncrement: 0,
    poolMax: 4,
    poolMin: 4,
  };
  try{
    // Create connection pool
    pool = await oracledb.createPool(config);
    // Log success message
    console.log("Connection pool created successfully");
  }catch(err){
    // Log error message if pool creation fails
    console.error("Error creating connection pool", err.message);
  }
}
```

```

// Function to create connection
async function createConnection() {
  // If pool is not initialized, initialize it
  if (!pool) {
    await initPool();
  }
  try {
    // Get connection from pool
    let connection = await oracledb.getConnection();
    // Log success message
    console.log("Connection successful");
    return connection;
  } catch (error) {
    // Log error message if connection fails
    console.error("Connection failed:", error.message);
  }
}

```

```

// Function to fetch data from the database
export async function getData(){
  let conn;
  try{
    conn = await createConnection()
    // Execute query to fetch all rows from RAILWAY_ACCIDENTS table
    const data = await conn.execute(
      `SELECT * FROM RAILWAY_ACCIDENTS`
    )
    return data.rows;
  }catch(err){
    console.error(err)
  }finally{
    await conn.close();
    console.log("Connection closed")
  }
}

```

```

// Functions to fetch data based on different criteria
export async function railway(date1,date2){
    // Pseudocode for fetching railway data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function division(date1,date2){
    // Pseudocode for fetching division data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function location(date1,date2){
    // Pseudocode for fetching location data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

```

```

export async function category(date1, date2){
    // Pseudocode for fetching category data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function code(date1, date2){
    // Pseudocode for fetching code data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function type(date1, date2){
    // Pseudocode for fetching type data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

```

```

export async function rk(date1, date2){
    // Pseudocode for fetching rk data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function rg(date1, date2){
    // Pseudocode for fetching rg data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function rs(date1, date2){
    // Pseudocode for fetching rs data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

export async function district(date1, date2){
    // Pseudocode for fetching district data
    // 1. Establish connection
    // 2. Execute SQL query with parameters date1 and date2
    // 3. Return query result
}

```

**Figure 4. Pseudocode for Database.js**

- Detailed Explanation

- 1. Importing Modules and Setting Configuration:**

The oracledb module is imported and the output format is set to OUT\_FORMAT\_OBJECT, ensuring results are returned as JavaScript objects.

- 2. Connection Pool Initialization:**

The initPool function initializes the connection pool with specific configurations such as user, password, connectString, and pool settings.

The pool is created and success or error messages are logged accordingly.

### **3. Creating Database Connections:**

The `getConnection` function retrieves a database connection from the pool.

If the pool is not already initialized, it calls `initPool` to initialize it first.

A connection is obtained and a success message is logged; if it fails, an error message is logged.

### **4. Fetching Data from the Database:**

The `getData` function establishes a connection, executes a SQL query to fetch all rows from `RAILWAY_ACCIDENTS`, returns the data, and ensures the connection is closed.

### **5. Additional Functions for Fetching Data Based on Criteria:**

Functions like `byRailway`, `byDivision`, `byLocation`, etc., follow a similar structure:

- 1) Establish a connection.
- 2) Execute a SQL query with provided parameters.
- 3) Return the query result.
- 4) Close the connection.

### *Server.js*

The `server.js` file sets up an Express server to provide a RESTful API for querying railway accident data from an Oracle database. It uses various routes to handle different types of queries based on date ranges and other criteria. This server also includes middleware for handling Cross-Origin Resource Sharing (CORS) to allow requests from different origins.

- Pseudocode for `server.js`

1. Import necessary modules:
  - express
  - cors
  - functions from database.js
2. Create an instance of the Express application.
3. Set the server port to 8800.
4. Use CORS middleware to handle cross-origin requests.
5. Define routes to handle different API endpoints:
  - GET /: Fetch all data
  - GET /railway: Fetch data by railway
  - GET /division: Fetch data by division
  - GET /location: Fetch data by location
  - GET /category: Fetch data by category
  - GET /code: Fetch data by code
  - GET /type: Fetch data by type
  - GET /rk: Fetch data by RK
  - GET /rg: Fetch data by RG
  - GET /rs: Fetch data by RS
  - GET /district: Fetch data by district
6. For each route:
  - Extract query parameters (date1, date2).
  - Call the corresponding function from database.js.
  - Send the result as a JSON response.
  - Handle any errors by logging and sending an error response.
7. Start the server and listen on the specified port.

Figure 5. Pseudocode for Server.js

- Detailed Explanation

## 1. Module Imports

- 1) `express`: A web application framework for Node.js to build APIs.
- 2) `cors`: Middleware to enable Cross-Origin Resource Sharing.
- 3) Functions (`division`, `getData`, `location`, `railway`, `category`, `code`, `type`, `rs`, `rk`, `rg`, `district`) from `database.js` to handle different database queries.

## **2. Express Application Setup**

`app` is created as an instance of the Express application and `PORT` is set to 8800(say).

## **3. CORS Middleware**

`app.use(cors())` is used to enable CORS for all routes.

## **4. Routes Definition**

Each route is defined using the `app.get()` method to handle GET requests.

## **5. Root Route (/)**

- 1) Fetches all data by calling `getData()`.
- 2) Returns the data as a JSON response.
- 3) Handles errors by logging and sending a 500 status code.

## **6. Railway Route (/railway)**

- 1) Extracts `date1` and `date2` from query parameters.
- 2) Calls `railway(date1, date2)` to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

## **7. Division Route (/division)**

- 1) Extracts `date1` and `date2` from query parameters.
- 2) Calls `division(date1, date2)` to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.



## **8. Location Route (/location)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls location(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

## **9. Category Route (/category)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls category(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

## **10. Code Route (/code)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls code(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

## **11. Type Route (/type)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls type(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

## **12. RK Route (/rk)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls rk(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

## **13. RG Route (/rg)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls rg(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

#### **14. RS Route (/rs)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls rs(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

#### **15. District Route (/district)**

- 1) Extracts date1 and date2 from query parameters.
- 2) Calls district(date1, date2) to fetch data.
- 3) Returns the data as a JSON response.
- 4) Handles errors by logging and sending a 500 status code.

#### **16. Starting the Server**

```
app.listen(PORT, () => { console.log(Server running at port ${PORT}); });
```

It starts the server and listens on the specified port like 8800, and logs a message indicating the server is running.

### **5.3.2 Client-Side Implementation**

The client-side implementation of the railway accident data visualization application is built using React. The main component, App.jsx, serves as the entry point and coordinates the rendering of various chart modules. Each module is responsible for displaying specific types of data, such as accidents by division, location, or category. Material UI is utilized to provide a rich set of UI components, including the date picker for selecting date ranges, which

enhances the user experience by allowing them to filter data based on specific time periods.

### *App.jsx*

The App.jsx file is the main entry point of our React-based client application. This component orchestrates the fetching and displaying of railway accident data in various chart formats. It acts as the central hub for managing the state of the application, making asynchronous requests to the server, and passing the retrieved data to different chart components for visualization. Additionally, App.jsx integrates a date picker to allow users to select a date range for which they want to view the railway accident data, enhancing the interactivity and user experience of the application.

- The primary objectives of App.jsx are:
  1. State Management: Maintaining the state for different categories of railway accident data.
  2. Data Fetching: Making HTTP requests to the server to fetch data for each category.
  3. Rendering: Displaying the data through various chart components to provide a comprehensive view of the railway accident statistics.
  
- Overview of Functionality
  1. State Initialization: The component initializes state variables for each data category using React's useState hook.
  2. Data Fetching: Utilizes the axios library within the useEffect hook to fetch data from the server when the component mounts.
  3. Chart Rendering: Renders specific chart components for each category of data, ensuring the application provides a detailed and interactive visualization of the railway accident statistics.

By modularizing the code in this manner, App.jsx maintains a clean separation of concerns, making the codebase more manageable, scalable, and easier to understand.

- Pseudocode of App.jsx

```
1. Import necessary modules:
  - React
  - useState, useEffect from React
  - axios for making HTTP requests
  - Chart components for different categories

2. Define the App component:
  - Initialize state to store fetched data.
  - Use useEffect to fetch data from the server when the component mounts.
  - Define functions to fetch data for different categories.
  - Render the chart components, passing the appropriate data as props.

3. Fetch data function:
  - Use axios to make a GET request to the server.
  - Store the fetched data in the state.

4. Render the App component:
  - Render chart components for different categories, passing the fetched data as props.
```

**Figure 6. Pseudocode for App.jsx**

- Detailed explanation

### **1. Module Imports**

- 1) React and Hooks: Core modules from React are imported to build the component and manage its state and lifecycle.
- 2) Material-UI Components: Grid and Box components from Material-UI are used for layout. The LocalizationProvider, AdapterDayjs, and DatePicker from Material-UI/Pickers are used for date selection.

- 3) dayjs: A library for date manipulation, used to handle date formatting and restrictions.
- 4) axios: A promise-based HTTP client for making requests to the server to fetch data.
- 5) Chart Components: Custom components such as RailwayChart, DivisionChart, LocationChart, CategoryChart, CodeChart, TypeChart, and DistrictChart are imported to render different charts based on the data fetched from the server.

## **2. App Component Definition**

- 1) State Variables: The App component uses the useState hook to define state variables from, to, and toKey. These variables are used to manage the date range for filtering data.
- 2) Event Handlers: handleFromChange and handleToChange functions are defined to update the state when the user selects new dates in the DatePicker components.
- 3) Data Fetching with useEffect: The useEffect hook is used to fetch data from the server whenever the date range changes. The data fetching functions are called with the selected date range and the corresponding state setter functions are used to update the state with the fetched data.

## **3. JSX Structure**

- 1) Grid Container: The main layout is managed using a Grid container, providing a responsive layout for the components.
- 2) LocalizationProvider: Wraps the DatePicker components to provide localization and date handling using AdapterDayjs.
- 3) DatePicker Components: Two DatePicker components are included for selecting the "From" and "To" dates. These components are configured with properties

such as label, format, value, onChange, orientation, minDate, and maxDate to control their behavior and appearance.

- 4) Title Display: A Grid item is used to display the title "Indian Railway - Accident Cases" in the center of the layout.
- 5) Category Charts: Additional Grid items are used to render the category charts. Each chart component is passed the date range as props to fetch and display data accordingly.

#### **4. Category Chart Components**

- 1) Data Fetching: Each chart component fetches data from the server based on the provided date range. This is typically done using an useEffect hook within the chart component.
- 2) Rendering Charts: The chart components render visual representations of the fetched data, allowing users to see trends and patterns in railway accident cases based on different criteria.

By following this structure, the client-side application efficiently fetches and displays data from the server, allowing users to visualize railway accident data interactively. The integration of Material-UI's date picker enhances user experience by providing an intuitive way to select and manipulate date ranges.

#### *Charts in App.jsx*

In our client-side application, we utilize various modules to display different categories of railway accident data through charts. Each module is designed to fetch specific data from the server and render it using a charting library. This modular approach allows for easy maintenance, scalability, and a clear separation of concerns within the codebase.

- Chart Modules

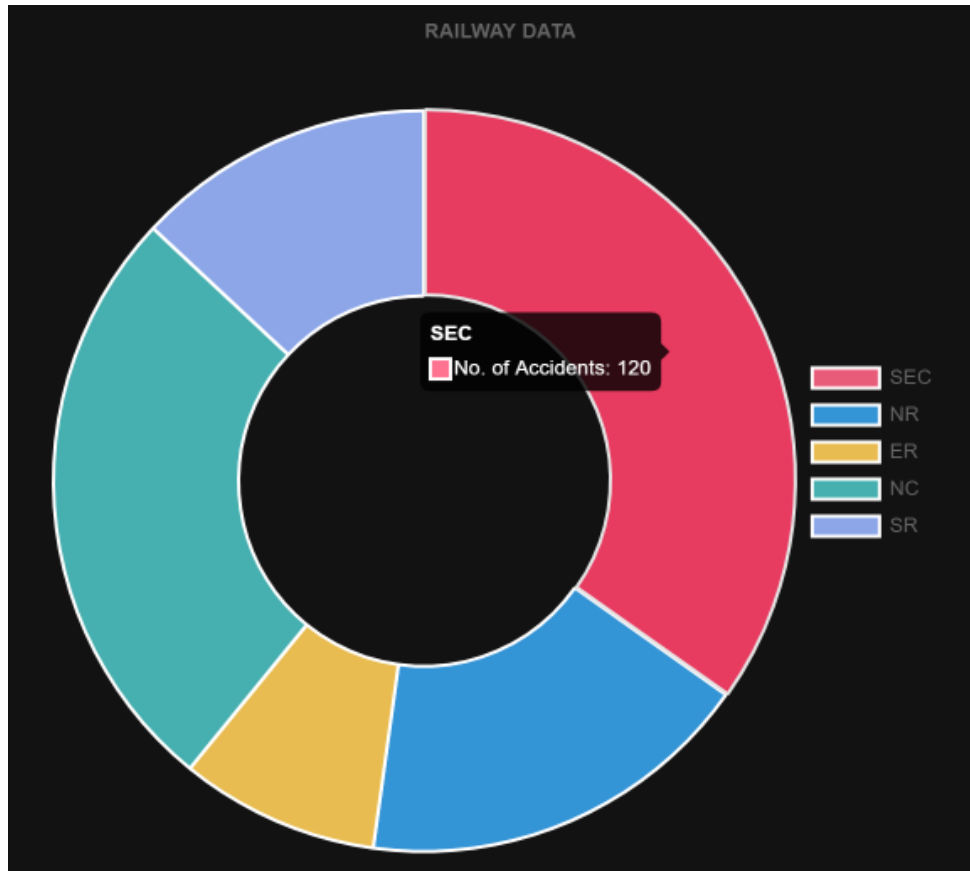
1. `RailwayAccidentsByRailway.jsx`
2. `RailwayAccidentsByDivision.jsx`
3. `RailwayAccidentsByCode.jsx`
4. `RailwayAccidentsByCategory.jsx`
5. `RailwayAccidentsByLocation.jsx`
6. `RailwayAccidentsByType.jsx`
7. `RailwayAccidentsConsequential.jsx`
8. `RailwayAccidentsByDistrict.jsx`

Each of these modules will be responsible for:

- 1) Fetching data from the server.
- 2) Storing the fetched data in state.
- 3) Rendering a chart to display the data.

- Pseudocode for each module:

- 1. `RailwayAccidentsByRailway.jsx`**



**Figure 7: Doughnut Chart**

This module displays railway accidents categorized by different railway lines. It utilizes a doughnut chart to visualize the distribution of accidents across various railway networks.



```

import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Doughnut } from 'react-chartjs-2';

export default function DoughnutChart({ from, to }) {
  const [data, setData] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        // Fetch data from the server
        // Update state with fetched data
      } catch (error) {
        // Handle errors
      }
    };
    fetchData();
  }, [from, to]);

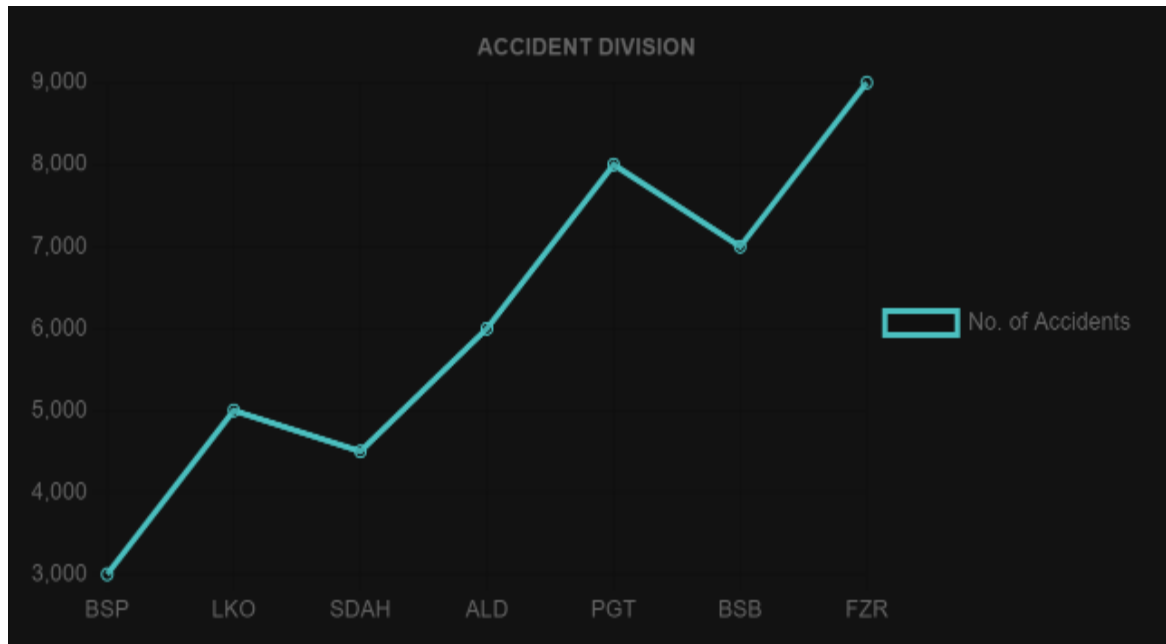
  const chartData = {
    labels: data.map(item => item.Label),
    datasets: [
      {
        data: data.map(item => item.Value),
        backgroundColor: [ /* Define colors */ ],
        hoverOffset: 4
      },
    ],
  };

  return (
    <div>
      <h2>Railway Wise Accidents</h2>
      <Doughnut data={chartData} />
    </div>
  );
}

```

Figure 8: Pseudocode for Doughnut Chart

## 2. RailwayAccidentsByDivision.jsx



**Figure 9: Line Chart**

This module illustrates railway accidents based on divisions or administrative regions. It employs a line chart to depict trends in accident occurrences over time within each division.

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Line } from 'react-chartjs-2';

export default function LineChart({ from, to }) {
  const [data, setData] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        // Fetch data from the server
        // Update state with fetched data
      } catch (error) {
        // Handle errors
      }
    };
    fetchData();
  }, [from, to]);
```

```

const chartData = {
  labels: data.map(item => item.Label),
  datasets: [
    {
      label: 'No. of Accidents',
      data: data.map(item => item.Value),
      borderColor: "rgb(75,192,192)",
      fill: false,
      tension: 0.1,
      pointBackgroundColor: "yellow"
    },
  ],
};

```

```

const options = {
  responsive: true,
  plugins: {
    legend: {
      labels: {
        font: {
          size: 14
        }
      }
    },
    title: {
      display: true,
      text: "DIVISION WISE ACCIDENTS",
      font: {
        size: 20
      }
    }
  }
};

```

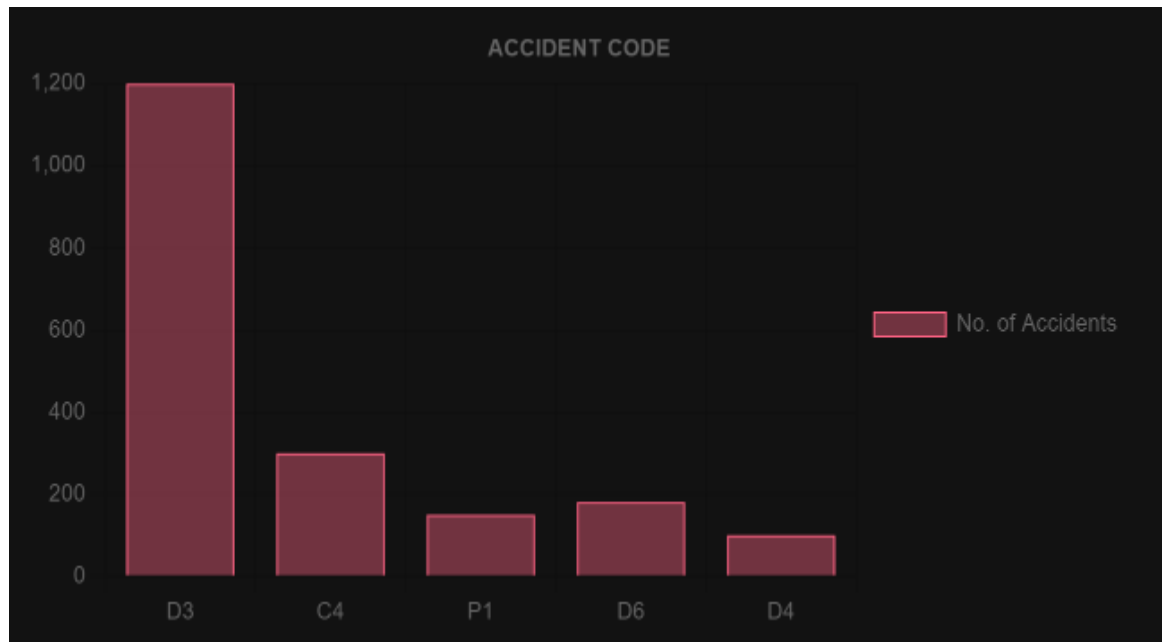
```

return (
  <div>
    <h2>Division Wise Accidents</h2>
    <Line data={chartData} options={options} />
  </div>
);
}

```

Figure 10: Pseudocode for Line Chart

### 3. RailwayAccidentsByCode.jsx



**Figure 11: Bar Chart**

This module presents railway accidents categorized by accident codes. It utilizes a bar chart to show the frequency of different types of accidents represented by their respective codes.

```

import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { Bar } from 'react-chartjs-2';

export default function BarChart({ from, to }) {
  const [data, setData] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        // Fetch data from the server
        // Update state with fetched data
      } catch (error) {
        // Handle errors
      }
    };
    fetchData();
  }, [from, to]);

```

```

const chartData = {
  labels: data.map(item => item.Label),
  datasets: [
    {
      label: 'No. of Accidents',
      data: data.map(item => item.Value),
      backgroundColor: "rgba(87, 22, 126, 0.4)",
      borderColor: "rgba(87, 22, 126, 1)",
      borderWidth: 1
    }
  ]
};

return (
  <div className="chart-container">
    <Bar height={400} data={chartData} />
  </div>
);
}

```

Figure 12: Pseudocode for Bar Chart

**4. RailwayAccidentsByCategory.jsx:**

This module showcases railway accidents classified by accident categories. It utilizes a pie chart to provide an overview of the distribution of accidents across various categories.

**5. RailwayAccidentsByLocation.jsx:**

This module displays railway accidents based on their geographical locations. It employs a doughnut chart to visualize the distribution of accidents across different locations.

**6. RailwayAccidentsByType.jsx:**

This module presents railway accidents categorized by the type of accident. It utilizes a doughnut chart to provide insights into the distribution of accident types.

**7. RailwayAccidentsByConsequential.jsx:**

This module combines the consequential accident categories, including RK, RG, and RS, into a single visualization. These categories represent different types of consequential accidents.

When the user interacts with the consequential section of the category pie chart, it triggers a new tab to open. This new page, routed using react-dom-router, displays a bar chart. This bar chart provides counts for RK, RG, and RS, offering detailed insights into the distribution and frequency of consequential accidents within the railway accident dataset.

**8. RailwayAccidentsByDistrict.jsx:** This module illustrates railway accidents based on districts or geographical subdivisions. It employs a bar chart to depict the frequency of accidents occurring within different districts.

### *AccidentTable.jsx*

AccidentTable.jsx is a React component designed to display accident data fetched from an API endpoint. This component renders a table that presents various attributes of each accident, such as the accident ID, description, division, and district. By utilizing axios for HTTP requests, it fetches the data asynchronously upon component mount using useEffect hook. The fetched data is then mapped and displayed in the table rows. This component serves as a user-friendly interface to view and analyze accident data efficiently.

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

const DataTable = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const result = await axios.get('http://localhost:8800/getData');
        setData(result.data);
      } catch (error) {
        console.error('Error fetching data:', error.message);
      }
    };
    fetchData();
  }, []);
```

```

return (
  <div>
    <h2>Accident Data Table</h2>
    <table>
      <thead>
        <tr>
          <th>Accident ID</th>
          <th>Accident Description</th>
          <th>Accident Division</th>
          <th>Accident District</th>
        </tr>
      </thead>
      <tbody>
        {data.map((item, index) => (
          <tr key={index}>
            <td>{item.accidentId}</td>
            <td>{item.accidentDescription}</td>
            <td>{item.accidentDivision}</td>
            <td>{item.accidentDistrict}</td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
);

```

Figure 13: Pseudocode for AccidentTable.jsx

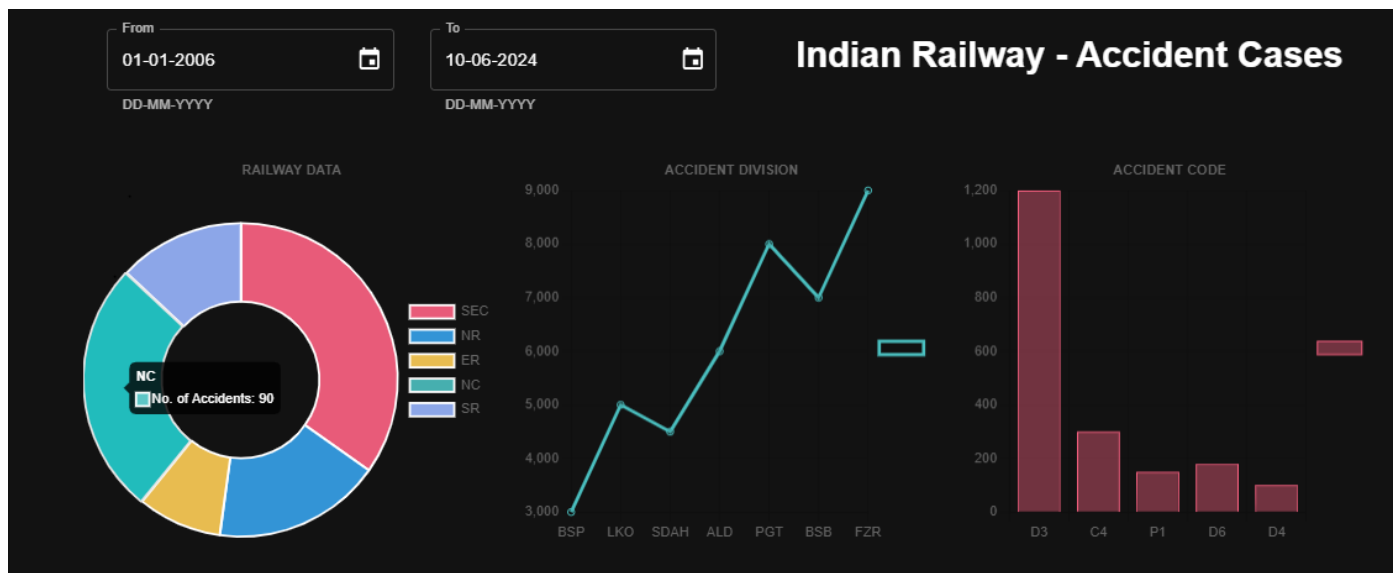


Figure 14: Snapshot of Indian Railway Accidents Dashboard Project



## **CHAPTER 6: CONCLUSIONS AND FUTURE SCOPE**

### **6.1 CONCLUSIONS**

In conclusion, this project has successfully implemented a system for visualizing railway accident data, providing stakeholders with valuable insights to enhance decision-making processes. By leveraging various visualization techniques such as charts and tables, users can easily interpret complex data and identify patterns, trends, and areas for improvement. This enhancement in data presentation significantly improves the overall user experience, making the application more intuitive and efficient.

### **6.2 FUTURE SCOPE**

While the current implementation focuses on a single table of accident data, there is ample opportunity for expansion and refinement in the future. One potential avenue for growth is to incorporate additional tables from the database and utilize SQL joins to extract more comprehensive insights from the data. By integrating multiple datasets and applying advanced analytical techniques, such as correlation analysis and predictive modeling, the system can offer even deeper and more nuanced perspectives on railway safety and accident prevention. This expansion will further enhance the value proposition of the application, making it an indispensable tool for railway authorities and safety regulators.

## REFERENCES

- [1] React, "Learn React," [Online]. Available: <https://react.dev/learn>.
- [2] Material-UI, "Getting Started - Material-UI," [Online]. Available: <https://mui.com/material-ui/getting-started/>.
- [3] Material-UI, "React Grid - Material-UI," [Online]. Available: <https://mui.com/material-ui/react-grid/>.
- [4] "Node-oracledb Documentation," Node-oracledb.readthedocs.io. [Online]. Available: [https://node-oracledb.readthedocs.io/en/latest/user\\_guide/introduction.html#getting-started-with-node-oracledb](https://node-oracledb.readthedocs.io/en/latest/user_guide/introduction.html#getting-started-with-node-oracledb).
- [5] "Installing - Express.js," Expressjs.com. [Online]. Available: <https://expressjs.com/en/starter/installing.html>.
- [6] Chart.js, "Documentation - Chart.js," [Online]. Available: <https://www.chartjs.org/docs/latest/>.