A PROJECT REPORT

On

Joinode - Code Editor

Submitted by

Suraj Kumar (10800220054)
Nikunj Ranjan (10800220044)
Akanksha (10800220002)
Pallavi Kumari (10800220068)

Submitted to Asansol Engineering College in partial fulfilment of the requirements for the degree of

Bachelor of Technology

(Information Technology)

Under the guidance

of

Mr. Santu Mondal
Assistant Professor

Information Technology
Asansol Engineering College
Asansol

AFFILIATED TO

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

June, 2024

ASANSOL ENGINEERING COLLEGE
Kanyapur, Vivekananda Sarani, Asansol, Paschim Bardhaman, West Bengal - 713305
Phone: 225-3057, 225-2108 Telefax: (0341) 225-6334
E-mail: principal.aecwb@gmail.com Website: www.aecwb.edu.in

CERTIFICATE

Certified that this project report on **Joinode** is the bonafide work of **"Suraj Kumar (10800220054), Nikunj Ranjan (10800220044), Akanksha (10800220002), Pallavi Kumari (10800220068)"** who carried out the project work under my supervision.

Mr. Santu Mondal
Assistant Professor
Information Technology

Dr. Anup Kumar Mukhopadhyay
HoD, Information Technology

Department of Information Technology
Asansol Engineering College
Asansol

ACKNOWLEDGEMENT

It is our great privilege to express our profound and sincere gratitude to our Project Supervisor **Mr. Santu Mondal,** Assistant Professor for providing us with very cooperative and precious guidance at every stage of the present project work being carried out under his/her supervision. His valuable advice and instructions in carrying out the present study have been a very rewarding and pleasurable experience that has greatly benefitted us throughout our work.

We would also like to pay our heartiest thanks and gratitude to **Dr. Anup Kumar Mukhopadhyay**, HoD, and all the faculty members of the Information Technology, Asansol Engineering College for various suggestions being provided in attaining success in our work.

Finally, we would like to express our deep sense of gratitude to our parents for their constant motivation and support throughout our work.

……
Suraj Kumar (10800220054)

……
Nikunj Ranjan (10800220044)

……
Akanksha (10800220002)

……
Pallavi Kumari (10800220068)

Date: __/__/____
Place: Asansol

4th Year
Information Technology

CONTENTS

LIST OF FIGURES

1. PROJECT SYNOPSIS

Joinode is a cutting-edge online development platform designed to make writing, compiling, and sharing HTML, CSS, and JavaScript code easy and efficient. Built on the robust MERN stack (MongoDB, Express.js, React, Node.js), Joinode provides a seamless, browser-based coding experience akin to CodePen. Its primary goal is to help developers of all levels quickly prototype frontend projects. With features that support real-time coding and live previews, Joinode streamlines the process of saving, managing, and sharing projects. By enhancing the tools and functionalities available, Joinode aims to improve the overall coding experience for developers.

Here user can see all the results on the output screen present within the platform and save the code manually and can also share the code with others. Joinode comes packed with several key features to create a comprehensive and user-friendly coding environment. The MERN stack forms the backbone of the platform, with React.js and Redux powering the frontend, Node.js and Express.js managing the backend, and MongoDB handling data storage.

One of the standout features of Joinode is its collaborative coding capabilities. Additionally, Joinode supports a wide range of integrations with popular development tools and services, enhancing productivity and workflow.

Joinode also emphasizes learning and growth within the developer community. The platform includes a library of tutorials, code snippets, and example projects to help users learn new skills and best practices. Users can also participate in coding challenges and hackathons, fostering a spirit of competition and innovation.

The project is organized into four phases: Planning and Design (1 month), Development (3 months), Testing, QA(1 month). This structured timeline ensures thorough planning, solid development, and rigorous testing, all aimed at delivering a high-quality product. Joinode is set to transform frontend development by offering a robust, scalable, and user-friendly platform that fosters innovation, collaboration, and continuous learning within the developer community. With its rich feature set and emphasis on community engagement, Joinode aims to become an essential tool for developers looking to create, share web projects.

2. INTRODUCTION

In the ever-evolving landscape of web development, the need for versatile, efficient, and collaborative tools has become paramount. Joinode addresses this demand by offering a comprehensive, browser-based development environment specifically designed for HTML, CSS, and JavaScript projects. Inspired by the functionality and community spirit of platforms like CodePen, Joinode leverages the powerful MERN stack (MongoDB, Express.js, React, Node.js) to provide a robust solution that caters to developers' diverse needs. Joinode aims to streamline the frontend development process, enabling developers to write, compile, and preview and share their code in real-time.

2.1 Problem Formulation

The project is structured to provide an efficient and user-friendly web-based code editor with collaborative features. The frontend is built using React, enabling a dynamic and interactive user interface. Key components include the CodeEditor for writing code, RenderCode for displaying the output in real-time. The backend is developed using Express.js and connects to a MongoDB database using Mongoose. The backend handles API requests for saving and loading code snippets. For testing API endpoints, tools like Insomnia are used to ensure smooth data transactions between the client and server. URL generation and sharing are facilitated within the application, allowing users to share their code with unique URLs.

2.2 Motivation

The motivation behind this project is to create an accessible and interactive coding platform that facilitates learning and collaboration. Many aspiring developers and students face challenges in experimenting with code and seeing real-time results without a cumbersome setup.

2.3 Scope

The scope of this project includes developing a web-based code editor that supports multiple programming languages such as HTML, CSS, and JavaScript. Users will be able to write, execute, and see the real-time output of their code within the browser, eliminating the need for complex local setups. The platform will also offer features for saving and sharing code snippets via unique URLs, enabling easy collaboration and feedback.

3. REQUIREMENT ANALYSIS

The requirement analysis for Joinode focuses on defining the functional and non-functional requirements that will shape the development and implementation of the platform. These requirements are essential for ensuring that Joinode meets the needs of its users effectively and delivers a seamless frontend web development experience. By delineating these functional and non-functional requirements, Joinode aims to deliver a comprehensive online platform that meets the diverse needs of frontend developers. This requirement analysis serves as a foundational step in defining the project scope, guiding development efforts, and ensuring the successful implementation of Joinode as a valuable tool for enhancing productivity, fostering collaboration, and promoting innovation in frontend web development.

3.1 Functional Requirement:

The functional requirement specifies what the system should do, defining a particular function or action it needs to perform to meet user needs.

3.1.1 Syntax Highlighting:

- **Requirement**: The code editor should support syntax highlighting for HTML, CSS, and JavaScript.
- **Description**: Syntax highlighting improves code readability by applying different colors and styles to different language syntax elements.

3.1.2 Code Completion and Suggestions:

- **Requirement**: Offer intelligent code completion and suggestions.
- **Description**: As users type, the editor suggests completions based on context (e.g., variables, functions, CSS classes), enhancing productivity and reducing typing errors.

3.1.3 Real-Time Preview Integration:

- **Requirement**: Integrate with a real-time preview pane for immediate code visualization.
- **Description**: Changes made in the code editor should update instantly in the adjacent preview pane, allowing users to see the impact of their code modifications in real-time.

3.1.4 Notification:

- **Real-Time Notifications:** Notify users of important events such as url copied through real-time notifications.

3.2. Software Requirement Specification (SRS)

A Software Requirements Specification (SRS) document comprehensively describes the functional and non-functional requirements, use cases, and design constraints for a software application.

3.2.1 Hardware Requirements:

Hardware requirements specify the physical components, such as processors, memory, and storage, necessary to run the software application efficiently.

- **Servers:** Multi-core processors, sufficient RAM (e.g., 8GB+), and SSD storage for optimal performance.
- **Database:** Utilize MongoDB as the database management system (DBMS) for storing user data, project details, and configurations. Consider MongoDB Atlas for managed database services to ensure scalability and high availability.
- **Networking:** Require stable internet connectivity and sufficient bandwidth to support real-time collaborative editing and data transmission.

3.2.2 Software Requirements:

Software requirements detail the necessary software components and dependencies, such as operating systems, frameworks, and libraries, required to develop and run the application.

- **Operating System:** Ubuntu Server for stability, security patches, and compatibility with MERN stack components. Cross-browser compatibility (Chrome, Firefox, Safari) and OS compatibility (Windows) for frontend development.
- **Frontend Development:**
    - **IDEs**: Visual Studio Code for coding, debugging, and integration with Git.
    - **Frameworks/Libraries**: React.js, Redux Toolkit for state management, and Tailwind CSS for UI components.
    - **Version Control**: Git for collaborative development, with GitHub or GitLab for repository hosting and version history management.

- **Backend Development:**
    - **IDEs/Editors**: Visual Studio Code, for Node.js development.
    - **Frameworks/Libraries**: Node.js with Express.js for server-side application logic, Mongoose for MongoDB object modeling.
    - **API Documentation**: Swagger for documenting RESTful APIs.

- **Database Management:**
    - MongoDB for NoSQL database storage, supporting JSON-like document structures and scalability.
    - Use MongoDB Compass for GUI-based database management and monitoring.

4. MODEL ANALYSIS

The project's architecture comprises a three-tier model: a client-side frontend, a server-side backend, and a database for data storage. Visualized in a diagram, the frontend is built with React, communicating with the backend through API endpoints. The server, powered by Node.js and Express, handles requests, incorporates compiler design principles for code analysis.

4.1 Architecture Design:

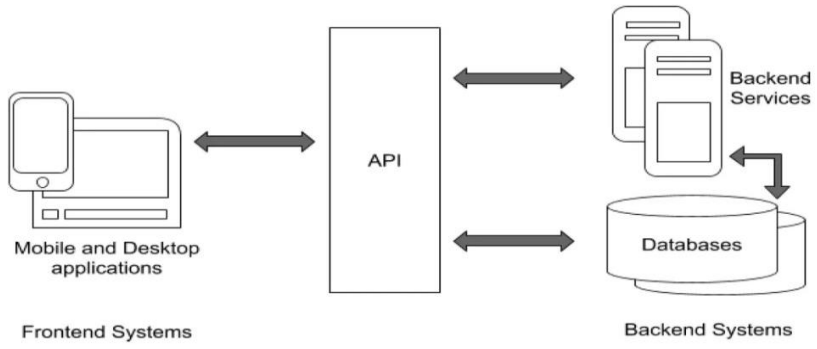
Architecture design refers to the conceptual blueprint that defines the structure, behavior, and interaction of software components within a system, ensuring optimal performance, scalability, and maintainability.

- **MERN Stack with TypeScript**: Joinode adopts the MERN (MongoDB, Express.js, React.js, Node.js) stack enhanced with TypeScript for developing a full-stack web application. TypeScript brings static typing to JavaScript, improving code quality, maintainability, and scalability across the application layers.
- **Database Model**: MongoDB serves as the NoSQL database for storing user profiles, project metadata, and code snippets.

4.2 State Management:

Handling and synchronizing the state of an application, ensuring consistent and predictable data flow and user interface updates.

- **Redux Toolkit with TypeScript**: Implements Redux Toolkit for managing application state in React.js. TypeScript enhances Redux development by providing type-safe action creators, reducers, and store configurations, reducing runtime errors and improving developer productivity.



[Figure 01: Frontend and backend system **]**

The MERN stack provides a powerful and versatile foundation for building modern web applications, combining the strengths of JavaScript across the entire development stack. It supports rapid development, scalability, and flexibility, making it an ideal choice for developing the project efficiently while ensuring a responsive and engaging user experience. Here are the major components of MERN stack:

- **MongoDB: Cross-platform Document-Oriented Database:** MongoDB is a NoSQL database where each record is a document consisting of key-value pairs that are similar to JSON (JavaScript Object Notation) objects. MongoDB is flexible and allows its users to create schema, databases, tables, etc. Documents that are identifiable by a primary key make up the basic unit of MongoDB. Once MongoDB is installed, users can make use of the Mongo shell as well. Mongo shell provides a JavaScript interface through which the users can interact and carry out operations (eg: querying, updating records, deleting records).

- **Express:** Back-End Framework: Express is a Node.js framework. Rather than writing the code using Node.js and creating loads of Node modules, Express makes it simpler and easier to write the back-end code. Express helps in designing great web applications and APIs. Express supports many middlewares which makes the code shorter and easier to write

- **React: Front-End Library:** Created by Facebook, ReactJS is a JavaScript library designed for crafting dynamic and interactive applications that is used for building user interfaces. React is used for the development of single-page applications and mobile applications because of its ability to handle rapidly changing data. Operating as an open-source, component-based front-end library, React allows users to code in JavaScript and create UI components.

- **Node.js:** Node.js has gained immense popularity among developers for its ability to handle server-side operations efficiently and effectively. Built on Chrome's V8 JavaScript engine, Node.js is designed to build scalable and high-performance applications. Node.js provides a JavaScript Environment which allows the user to run their code on the server (outside the browser). Node pack manager i.e. npm allows the user to choose from thousands of free packages (node modules) to download.

- **TypeScript:** It is an open-source, object-oriented programming language developed and maintained by Microsoft Corporation. Its journey began in 2012, and since then, it has gained significant traction in the developer community. It is a Strict Superset of JavaScript, which means anything implemented in JavaScript can be implemented using TypeScript along with adding enhanced features (every existing JavaScript Code is a valid TypeScript Code). It is designed mainly for large-scale project.

Compiler Design Principles in Code Analysis and Execution:

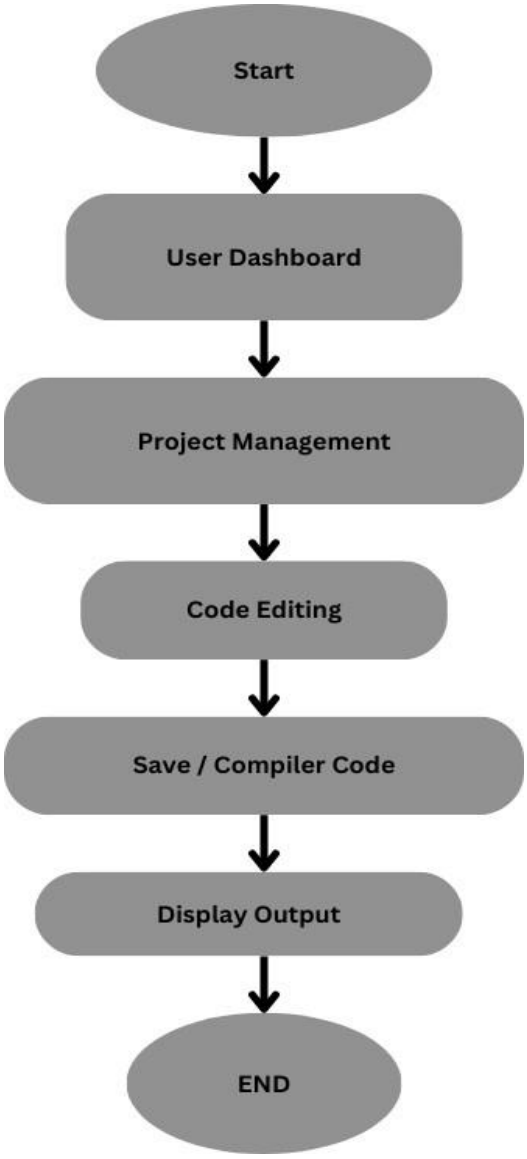
- **Lexical Analysis:** The first stage of compiler design is lexical analysis, also known as scanning. In this stage, the compiler reads the source code character by character and breaks it down into a series of tokens, such as keywords, identifiers, and operators. These tokens are then passed on to the next stage of the compilation process.
- **Syntax Analysis:** The second stage of compiler design is syntax analysis, also known as parsing. In this stage, the compiler checks the syntax of the source code to ensure that it conforms to the rules of the programming language. The compiler builds a parse tree, which is a hierarchical representation of the program's structure, and uses it to check for syntax errors.
- **Semantic Analysis:** The third stage of compiler design is semantic analysis. In this stage, the compiler checks the meaning of the source code to ensure that it makes sense. The compiler performs type checking, which ensures that variables are used correctly and that operations are performed on compatible data types. The compiler also checks for other semantic errors, such as undeclared variables and incorrect function calls.
- **Code Generation:** The fourth stage of compiler design is code generation. In this stage, the compiler translates the parse tree into machine code that can be executed by the computer. The code generated by the compiler must be efficient and optimized for the target platform.
- **Optimization:** The final stage of compiler design is optimization. In this stage, the compiler analyzes the generated code and makes optimizations to improve its performance. The compiler may perform optimizations such as constant folding, loop unrolling, and function inlining.
- **Code Generation**: In this final phase, the compiler translates the intermediate representation into executable machine code or another target language. This process involves mapping high-level constructs to low-level instructions and managing memory allocation.
- **Error Handling**: Throughout the compilation process, compilers must handle errors gracefully by providing meaningful error messages and recovering from recoverable errors to continue processing the code.
- **Data Flow Analysis**: Compilers perform data flow analysis to track the flow of data throughout the program. This analysis helps identify variables' values at different points in the code, enabling optimizations like dead code elimination and constant propagation.

5. DESIGN

Joinode operates as a dynamic and inclusive platform for frontend web development, integrating advanced tools, and robust security measures. By leveraging the MERN stack with TypeScript, Joinode enhances development workflows and empowers users to create, share, and deploy web projects effectively. The platform's intuitive interface, coupled with comprehensive functionality, supports a diverse range of users from individual developers to teams, fostering creativity and accelerating web development initiatives.

5.1 Workflow of the Project:

The workflow involves code input through the editor, state management via Redux, backend interaction through API calls.



[Figure 02: Workflow of Joinode **]**

- **Code Editing:** Users are directed to the code editor interface where they can write HTML, CSS, and JavaScript code.

- **Real-Time Editing:** As users type, the code editor provides real-time feedback, syntax highlighting, and code suggestions, enhancing the coding experience.

- **Code Analysis and Suggestions:** The application analyzes the code for errors, provides suggestions, and highlights potential improvements to enhance code quality.

- **Saving Code:** Users can save their code snippets for future reference or sharing. Saved codes are stored securely in the database.

5.2 Data Flow Diagram:

The data flow diagram illustrates the movement of data through the system, from user inputs to the final output displayed on the web interface.

5.2.1 Level 0 DFD

The diagram illustrates the workflow of an online compiler, which is a system designed to translate and execute code written by a user. The process begins with the user typing their code into an interface provided by the online compiler. Users may also ask questions about their code during this process. Once the code is submitted, the online compiler acts as a translator, converting the human-readable code into a machine-readable format that computers can execute.

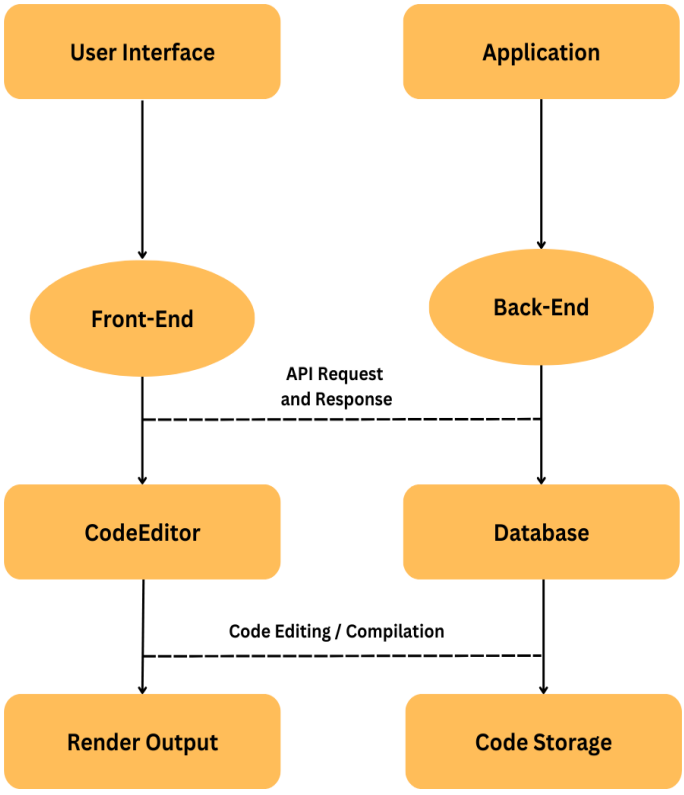
After the translation process, the compiler displays the results to the user. These results include the translated code, error messages if there are any issues with the code, and the execution output of the code.

[Figure 03: 0-Level DFD]

5.2.2 Level 1 DFD

The Level 1 DFD illustrates the flow of data between the User, Code Editor, State Management, and Backend API, facilitating code input, management, and storage within the system.

- **User Interface:** This is where the user interacts with the application.
- **Frontend:** Handles the presentation layer and user interaction components.
- **Backend:** Manages the application logic, data processing, and communication with the database.
- **Code Editor:** Allows users to input, edit, and compile code snippets in various programming languages.
- **Rendered Output:** Displays the output of the compiled code, such as rendered web pages or executed scripts.
- **Database:** Stores saved code snippets and associated metadata.
- **API Requests/Responses:** Facilitates communication between the frontend and backend through API requests and responses.
- **Code Storage:** Responsible for storing and retrieving saved code snippets from the database.



[Figure 04: 1-Level DFD **]**

6. IMPLEMENTATION

The "Joinode" application is an online code editor designed to empower users with a seamless coding experience. Built using React on the frontend and Express.js on the backend, it offers a dynamic and interactive platform for writing, compiling, and sharing HTML, CSS, and JavaScript code snippets. Tailwind CSS and Lucid Icons have been integrated to ensure a visually appealing and intuitive user interface. Users can write code in the editor and instantly visualize the output. The integration of Sonner allows for the display of toast notifications, providing instant feedback on actions such as saving or copying code snippets. Each code snippet is assigned a unique URL, enabling users to easily share their creations. Utilizing MongoDB as the backend database ensures efficient storage and retrieval of code snippets.

6.1 Library Used:

We have used React and Express to create a robust and dynamic online code editor application. React is a popular JavaScript library for building user interfaces, known for its component-based architecture and declarative approach to building UIs. Express.js is a minimal and flexible web application framework for Node.js, designed for building web applications and APIs.

Here are some key aspects of the React and Express:

- **Component-Based Architecture**: React organizes UIs into reusable components, making it easy to maintain and scale complex applications.

- **Virtual DOM**: React uses a virtual representation of the DOM to efficiently update only the components that have changed, leading to improved performance.

- **Declarative Syntax**: React's declarative syntax allows developers to describe the desired UI state, and React takes care of updating the DOM to match that state.

- **One-Way Data Binding**: React follows a unidirectional data flow, where data flows downward from parent to child components, making it easier to reason about data changes.

- **JSX**: React's JSX syntax enables developers to write HTML-like code within JavaScript, enhancing code readability and maintainability.

- **Middleware**: Express uses middleware functions to handle requests, enabling developers to modularize the application logic and add additional functionalities like authentication, logging, etc.

- **Routing**: Express provides a robust routing mechanism that allows developers to define endpoints for handling different HTTP methods and routes.

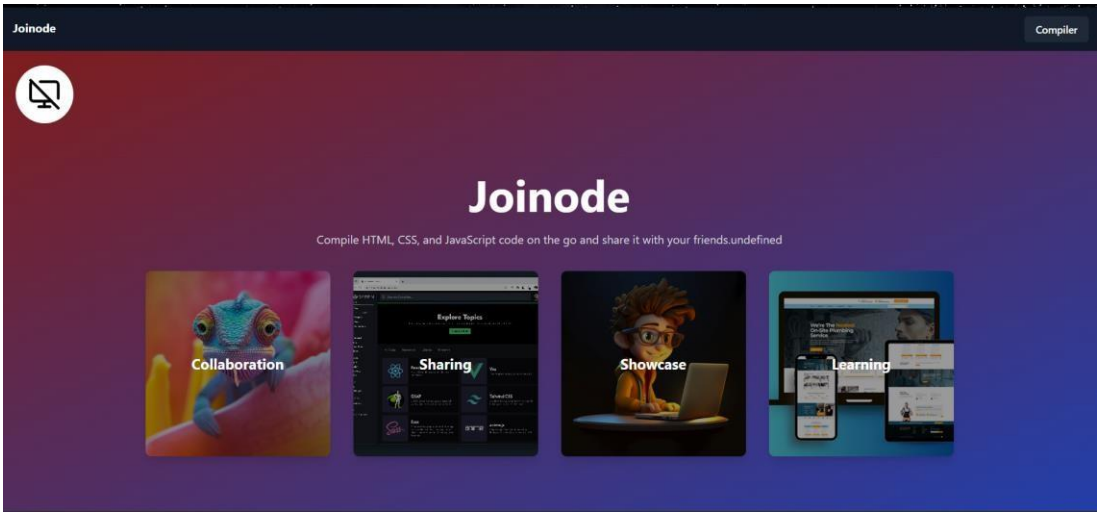
- **HTTP Utility Methods**: Express simplifies working with HTTP requests and responses by providing utility methods for common tasks like setting headers, sending files, etc.

- **Template Engines**: Express supports various template engines like EJS, Pug, etc., allowing developers to generate dynamic HTML content on the server.

- **Error Handling**: Express provides built-in error handling middleware and allows developers to define custom error handling logic, improving application robustness.

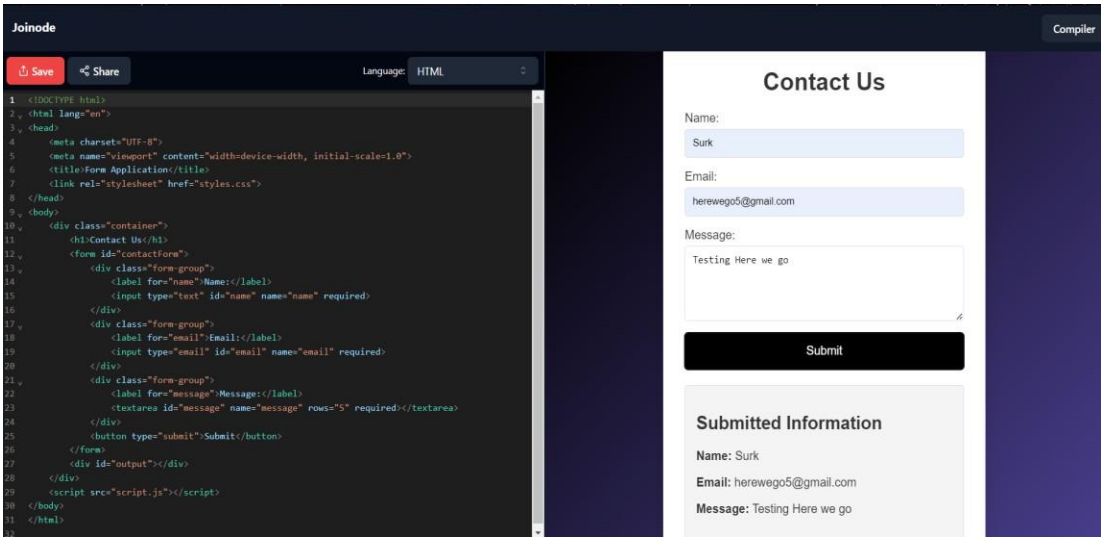
6.2 Frontend:

The frontend interface of "Joinode" application is designed for simplicity and ease of use. It has a home page where it contains a header. From there we can navigate to the compiler page which is the code editor. The code editor provides syntax highlighting and auto-completion features, enhancing code readability and productivity. Here users can toggle between the languages. Additionally, it also offers real-time output preview, allowing users to visualize the result of their code changes instantly. The frontend also includes features for code sharing and responsiveness across various devices and screen sizes. Users can save the codes that will be stored in the database.



[Figure 05: Home Page **]**

Users can click on the compiler button to jump into the code editor.

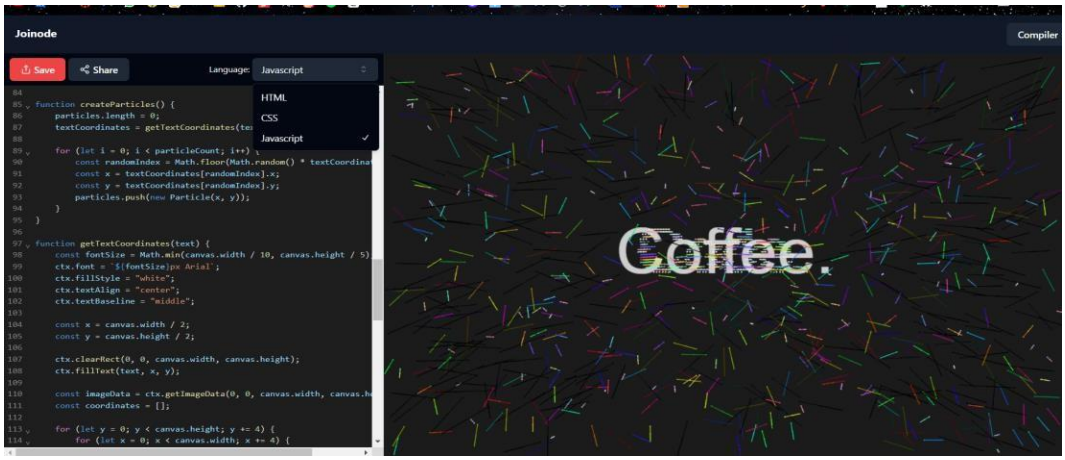


[Figure 06: Preview of the interface **]**

Compiler Picture - Preview of the interface of code editor. This is a flexible code editor. We can change the size of it according to our uses. Users can toggle between the languages in the top right corner of the editor. Users can view the results displayed in the Output screen. Code editor provides syntax highlighting and auto-completion features.

6.2.1 Results Display Area:

- Utilizing iframes, "Joinode" renders the output of HTML, CSS, and JavaScript code snippets in a separate section.
- It enables rapid iteration and experimentation,empowering users to refine their code and debug issues efficiently.



[Figure 07: Compiler **]**

6.2.2 Code rendering functionality:

The code rendering functionality converts the user-input code into a visually presented format, ensuring proper display and execution within the application.

- **State Selection**: It utilizes the useSelector hook to select the fullCode state from the Redux store. This state contains the HTML, CSS, and JavaScript code snippets entered by the user.

- **Combining Code**: The component combines the HTML, CSS, and JavaScript code snippets obtained from the Redux store into a single string named combinedCode. It encapsulates the HTML within <html> and <body> tags, embeds the CSS within <style> tags, and includes the JavaScript code within <script> tags.

- **Redux State Selection**: It utilizes the useSelector hook from React Redux to select the fullCode object from the Redux store's compilerSlice. This object contains the user's code for HTML, CSS, and JavaScript.

- **Iframe Source Generation**: It generates the source URL for the iframe by encoding the combinedCode string as a data URL with the MIME type text/html;charset=utf-8. This URL is assigned to the src attribute of the iframe, effectively rendering the combined code as an HTML document within the iframe.

```
import { RootState } from "@/redux/store"
import { useSelector } from "react-redux"

export default function RenderCode() {
    const fullCode = useSelector((state:RootState) => state.compilerSlice.fullCode);

    const combinedCode = `<html><style>${fullCode.css}</style><body>${fullCode.html}</body><script>${fullCode.javascript}
    </script></html>`

    const iframeCode = `data:text/html;charset=utf-8,${encodeURIComponent(
    combinedCode
  )}`;
  return (
    <div className="bg-white h-[calc(100dvh-60px)]">
        <iframe className="w-full h-full" src={iframeCode}/>

    </div>
  )
}
```

[Figure 08: Rendering Code Using IFrame **]**

6.2.3 Code sharing Feature:

The code sharing feature enables users to generate a unique URL for their code snippets, facilitating easy sharing and collaboration with others.

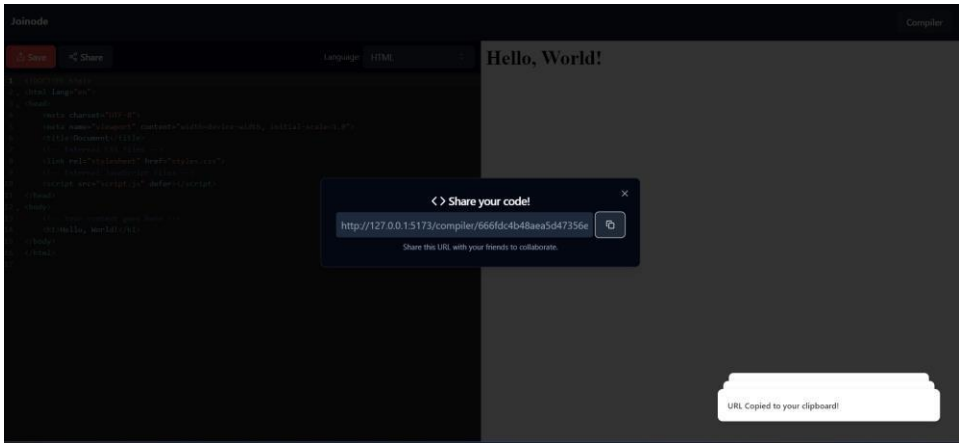
- **DialogTitle**: Within the DialogHeader, a DialogTitle component is used to display the title "Share your code!". It is styled using Tailwind CSS classes and centered horizontally with flexbox utilities.

- **URL Input**: Below the title, an input field is rendered to display the URL of the current page. The input is disabled to prevent user input and styled with Tailwind CSS classes to match the application's theme.

```
<DialogTitle className="flex gap-1 justify-center items-center"><Code />Share your code!</DialogTitle>
<div className="__url flex justify-center items-center gap-1">
  <input
    type="text"
    disabled
    className="w-full p-2 rounded bg-slate-800 text-slate-400 select-none"
    value={window.location.href}
  />
   <Button
    variant="outline"
    className="h-full"
    onClick={() => {
      window.navigator.clipboard.writeText(
        window.location.href
      );
      toast("URL Copied to your clipboard!");
    }}
  >
    <Copy size={14} />
  </Button>
  </div>
```

[Figure 09: Share Code **]**

- **Copy Button**: Adjacent to the URL input, a Button component is rendered with an icon indicating the copy action. Upon clicking this button, the URL of the current page is copied to the clipboard using the navigator.clipboard.writeText API.



[Figure 10: Code Sharing**]**

6.2.4 Managing Routes in the application:

Managing routes in the application involves defining URL paths and associating them with specific components or functionality to ensure proper navigation

- **Routes**: The <Routes> component from the React Router library is used to define the routing configuration of the application. It includes multiple <Route> components, each mapping a specific URL path to a corresponding React component.

- **Compiler Routes**: Both "/compiler" and "/compiler/" paths are mapped to the <Compiler> component. This component is responsible for handling code compilation and execution functionalities, including saving and loading code snippets.

- **NotFound Route**: The "*" path is a wildcard route, rendering the <NotFound> component when none of the specified routes match the URL path. This component typically displays a 404 error page or a message indicating that the requested page could not be found.

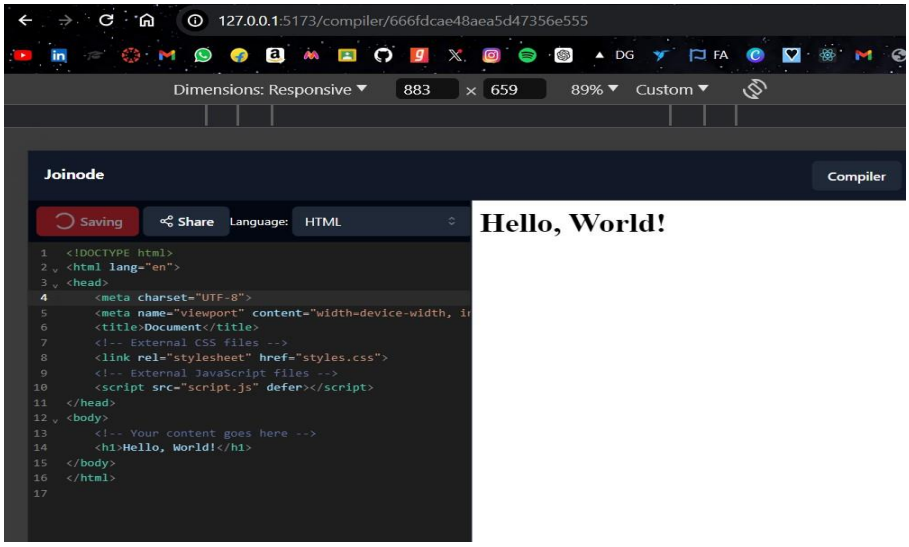
```
function App() {

  return (
    <>
    <Toaster position="bottom-right"/>
        <ThemeProvider defaultTheme="dark" storageKey="vite-ui-theme">
      <Header />
      <Routes>
        <Route path="/" element={<Home />}/>
        <Route path="/compiler" element={<Compiler />}/>
        <Route path="/compiler/:urlid" element={<Compiler />}/>
        <Route path="*" element={<NotFound />}/>
      </Routes>
      </ThemeProvider>

    </>
  )
}
```

[Figure 11: Managing Route Using ReactRouterDOM **]**

6.2.5 Saving codes and receiving code suggestions

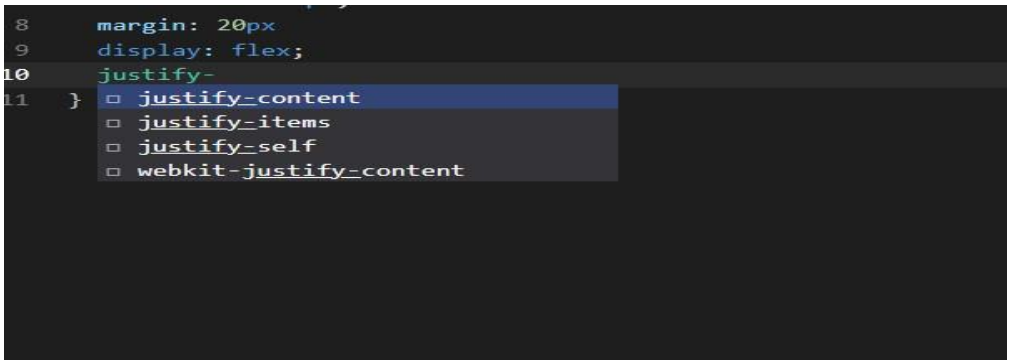
Saving codes and receiving code suggestions enhance user experience by providing the ability to store code snippets for later use and offering suggestions for optimizing or improving code quality.

- **Save Codes:** Users have the ability to save their code snippets within the application. Upon saving, each code snippet is assigned a unique identifier or URL, allowing users to access and retrieve their saved codes at any time. This feature enables users to store and organize their code projects, facilitating easy retrieval and reuse of code snippets for future reference or collaboration.



[Figure 12: Saving Code & URL Generation **]**

- **Code Suggestion:** The application incorporates features for code suggestion, providing users with recommendations and insights to improve their coding experience. This feature may include autocomplete suggestions, syntax highlighting, and error detection, helping users write cleaner, more efficient code. Code suggestion enhances productivity by reducing the time and effort required for manual coding tasks and promoting best coding practices.



[Figure 13: Code Suggestion**]**

6.3 Backend

The backend of the application is built using Node.js and Express.js, providing a robust server-side infrastructure to handle HTTP requests and responses. It follows a RESTful architecture, defining routes and controllers to manage the application's functionality. MongoDB is utilized as the database to store user-generated code snippets and associated metadata. Mongoose, an ODM (Object Data Modeling) library for MongoDB and Node.js, simplifies interactions with the database by providing a schema-based solution and facilitating data validation. The backend offers endpoints for saving and loading code snippets, allowing users to persist their work and retrieve it later for further editing or sharing. CORS (Cross-Origin Resource Sharing) middleware is implemented to enable cross-origin requests, facilitating communication between the frontend and backend components. Error handling is implemented throughout the backend to ensure robustness and reliability, with appropriate status codes and error messages returned to the client in case of unexpected issues.

6.3.1 Initializes the backend server:

Sets up the necessary environment and resources for handling incoming requests and executing server-side logic.

- **Middleware Configuration**: Express middleware is configured using app.use(). The express.json() middleware is used to parse incoming JSON requests, while the cors() middleware enables Cross-Origin Resource Sharing, allowing the frontend to make requests to the backend from a different origin.

- **Route Configuration**: The application uses the compilerRouter module to handle routes related to code compilation and execution. The app.use() function is used to specify that requests starting with "/compiler" should be handled by the compilerRouter.

- **Database Connection**: The dbConnect() function is called to establish a connection to the MongoDB database.

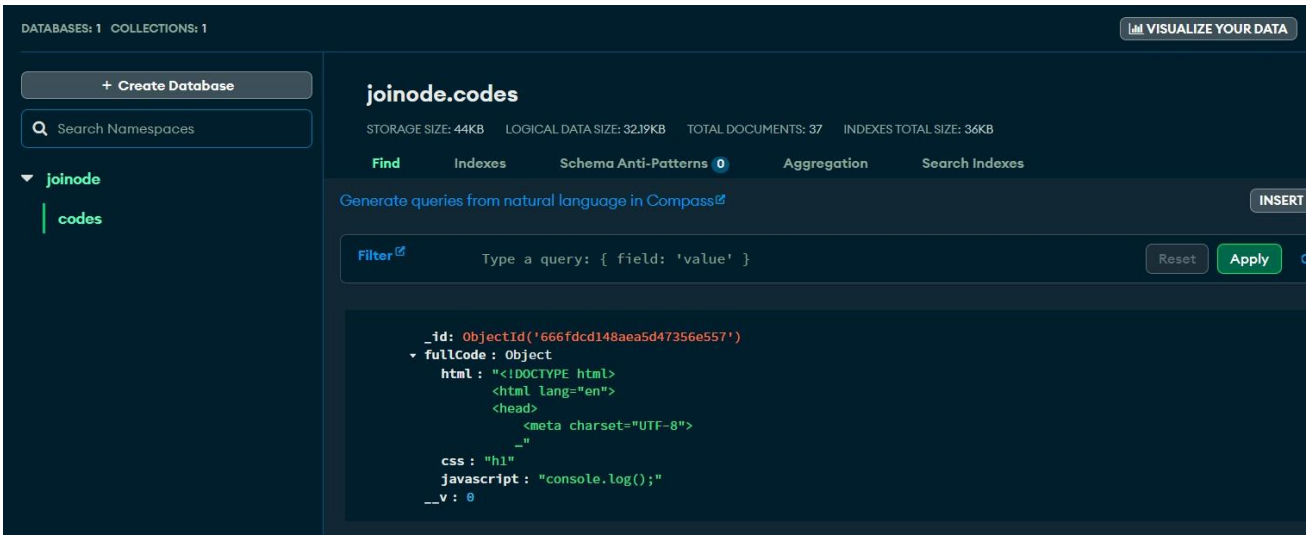
- **Server Listening**: The Express app listens on port 4000 for incoming HTTP requests. Once the server is successfully started, a message is logged to the console indicating the server's URL (http://localhost:4000).

```
import  express  from "express";
import cors from "cors";
import { config } from "dotenv";
import { dbConnect } from "./lib/dbConnect";
import { compilerRouter } from "./routes/compilerRouter";

const app = express();

app.use(express.json());
app.use(cors());
config();

app.use("/compiler", compilerRouter);

dbConnect();

app.listen(4000, () => {
    console.log("http://localhost:4000");
});
```

[Figure 14: Connecting Backend To Frontend**]**

6.3.2 Database Connection:

The database connection establishes a link between the application and the database, enabling data storage and retrieval operations.



[Figure 15: Database Connection**]**

- **Connection Parameters**: The mongoose.connect() method is called with two arguments: the MongoDB connection URI specified in the environment variable process.env.MONGO_URI and an options object. The dbName option is set to "joinode", specifying the name of the MongoDB database to connect to.

- **Connection Status Logging**: If the connection is successfully established, a message "Database Connected" is logged to the console using console.log(). If an error occurs during the connection attempt, the error message is caught by the catch block, and "Error connecting to Database" is logged to the console.

```
import mongoose from "mongoose";

export const dbConnect = async () => {
    try{
    await mongoose.connect(process.env.MONGO_URI!,{
        dbName:"joinode",
    });
    console.log("Database Connected");
    }catch(error){
        console.log("Error connecting to Database")
    }
};
```

[Figure 16: Database Connection code**]**

Creating Schema for Database:

It defines the structure and organization of the data stored in the database,specifying the fields, data types, and relationships between different entities.

- **Schema Definition**: It creates a Mongoose schema CodeSchema using the mongoose. Schema constructor. The schema defines the structure of documents in the "codes" collection. It specifies that each document should have a fullCode field with subfields html, css, and javascript, each of type String.

- **Model Creation**: It creates a Mongoose model named Code using the mongoose.model method. This model represents the "codes" collection in the MongoDB database and enforces the schema defined by CodeSchema.

```
import mongoose from "mongoose";

interface ICodeSchema {
  fullCode: {
    html: string;
    css: string;
    javascript: string;
  };
}

const CodeSchema = new mongoose.Schema<ICodeSchema>({
  fullCode: {
    html: String,
    css: String,
    javascript: String,
  },
});

export const Code = mongoose.model("Code", CodeSchema);
```

[Figure 17: Creating Schema For Database**]**

6.3.3 Handling HTTP requests:

Handling HTTP requests involves processing incoming requests from clients, routing them to the appropriate endpoints.

- **SaveCode**:

  This asynchronous function serves as a handler for HTTP POST requests directed to the "/compiler/save" endpoint. It anticipates the request body to include a property named "fullCode," presumably indicating the code snippet intended for preservation. Enclosed within a try-catch block, it employs the Code.create() method to generate a new entry in the MongoDB database collection linked with the Code model.

- **LoadCode:**

  It anticipates the request body to include a property named "urlId," presumably indicating the identifier of the code snippet intended for retrieval. Enclosed within a try-catch block, it utilizes the Code.findById() method to fetch the document from the MongoDB database collection associated with the Code model, based on the provided urlId.

```
import { Request, Response } from "express";
import { Code } from "../models/Code";

export const saveCode = async (req: Request, res: Response) => {
  const {fullCode} = req.body;
  try {
    const newCode = await Code.create({
      fullCode: fullCode
    });
    return res.status(201).send({ url: newCode._id, status: "saved!"});
  } catch (error) {
    return res.status(500).send({ message: "Error saving code", error });
  }
};

export const loadCode = async (req: Request, res: Response) => {
  const { urlId } = req.body;
  try {
    const existingCode = await Code.findById(urlId);
    if (!existingCode) {
      return res.status(404).send({ message: "Code not found" });
    }
    return res.status(200).send({ fullCode:existingCode.fullCode });
  } catch (error) {
    return res.status(500).send({ message: "Error loading code", error });
  }
};
```

[Figure 18: RESTFul API Design**]**

- **Two POST routes:**

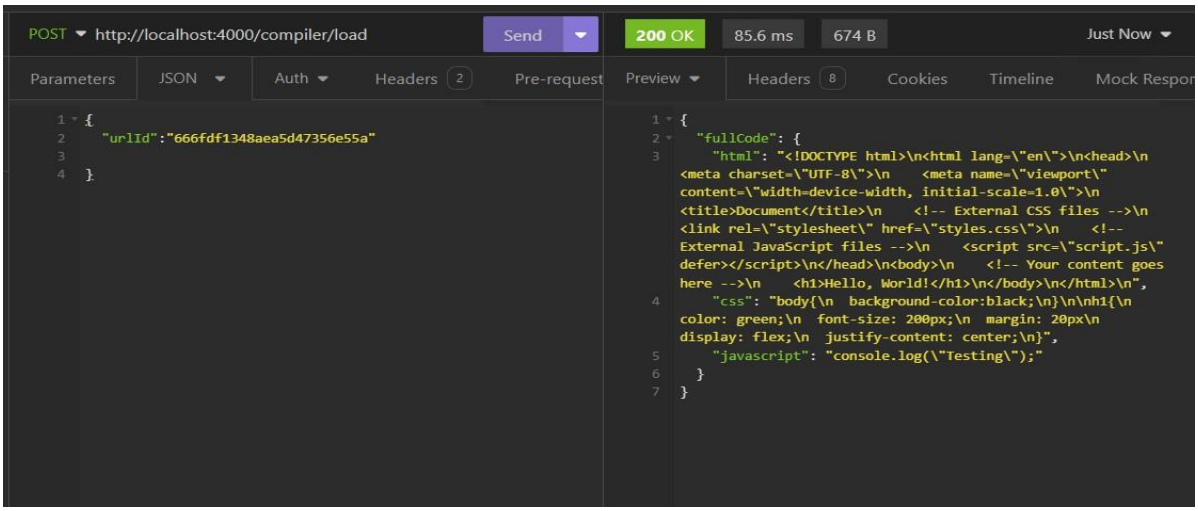
  The "/save" route is linked with the saveCode function imported from the "../controllers/compilerController" module. This route is tasked with saving code snippets. On the other hand, the "/load" route is associated with the loadCode function imported from the "../controllers/compilerController" module. Its responsibility lies in loading code snippets.

```
import express from 'express'
import { loadCode, saveCode } from '../controllers/compilerController'

export const compilerRouter = express.Router()

compilerRouter.post("/save", saveCode);
compilerRouter.post("/load", loadCode);
```

[Figure 19: API Requests**]**

6.3.4 Testing APIs

We have used Insomnia for testing APIs. Insomnia is a powerful tool that we utilized for testing the APIs in the "Joinode" application. It serves as a feature-rich REST client, enabling us to send various types of HTTP requests and observe the responses from our backend server.



[Figure 20: API Testing in Insomnia**]**

6.3.5 URL Generation:

URL generation involves creating dynamic URLs based on specific parameters or routes within the application.

- **URL Parameters Handling**: It utilizes the useParams hook from React Router to extract the urlId parameter from the URL, which represents the unique identifier of the code snippet being accessed.

- **Code Loading**: Upon component mount or when the urlId parameter changes, it triggers a function loadCode() to fetch the code snippet.

- **Resizable Panel Group**: It renders a resizable panel group using components from the ui/resizable module. This group contains two resizable panels, one for the code editor (CodeEditor) and the other for rendering the code output (RenderCode).

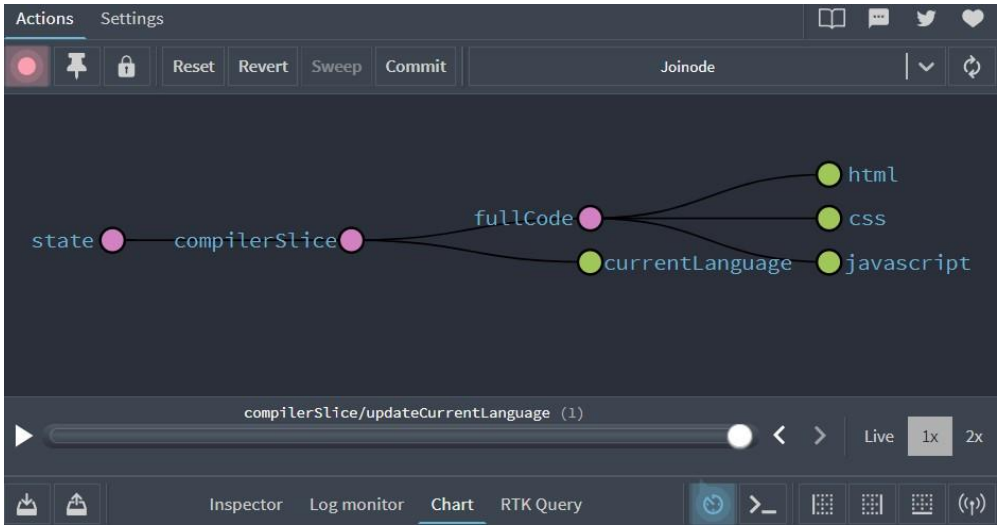
```
export default function Compiler() {
  const {urlId} = useParams();
  const dispatch = useDispatch();
  const loadCode = async () => {
    try{
      const response = await axios.post("http://localhost:4000/compiler/load", {
        urlId: urlId
      });
      dispatch(updateFullCode(response.data.fullCode));
      // console.log(response.data)
    }catch(error){
      handleError(error)
    }
  }
  useEffect(() => {
    if(urlId){
      loadCode();
    }
  },[urlId]);
```

[Figure 21: URLID Generation**]**

Here we have imported necessary dependencies such as use Effect and use Dispatch from React, as well as axios for making HTTP requests. Any errors that occur during the HTTP request are caught by the catch block and handled using the handleError() function, which is assumed to be defined elsewhere in the codebase.

6.3.6 Redux Devtools Chart

This is the data flow of the application created using Redux DevTool Extension.



[Figure 22: Visualization of dataflow in Redux **]**

7. CONCLUSION AND FUTURE SCOPE

The Joinode project is an innovative online code editor developed with the MERN stack (MongoDB, Express.js, React.js, Node.js) and TypeScript. It is designed to streamline and enhance the web development process with features such as real-time code preview, syntax highlighting, and error checking, catering to both individual developers and teams. These functionalities make it easier to create, edit, and manage projects within a user-friendly interface supported by robust backend services. Future developments for Joinode aim to include advanced code analysis tools and expanded language support, allowing developers to work with a broader range of programming languages. Enhanced code sharing functionality, generating shorter URLs, and providing options for sharing via email or social media platforms will also improve user experience.

We may add some features in future to make it better for user experience by implementing real-time collaboration features that would allow multiple users to edit and view code simultaneously, facilitating collaborative coding sessions. We can also introduce version control functionality that would enable users to track changes to their code over time, revert to previous versions, and collaborate more effectively on larger projects. User authentication and authorization would enhance security by allowing users to create accounts, log in securely, and manage their saved code snippets privately. We can enhance the code sharing functionality by generating shorter, more user-friendly URLs for shared code snippets and providing options for sharing via email or social media platforms. Apart from these things, we may integrate code analysis tools to provide real-time feedback and suggestions to users as they type, helping them identify errors, optimize code, and learn best practices. Users can also customize the appearance of the code editor and the overall UI by selecting different themes, font sizes, and color schemes to suit their preferences.

8. REFERENCE

1. https://blog.logrocket.com/complete-guide-react-refs/
2. https://medium.com/@AbbasPlusPlus/how-to-create-references-between-different-mongodb-documents-456e395c6500
3. https://developers.mews.com/how-to-get-your-cross-references-in-redux-straight/
4. https://codemirror.net/docs/ref/