Design and Development of Skillify: A ReactJS-Based Online Code Editor for Web Development

Mr. Hemant Mittal

Assistant Professor, Department of CSE, Global Institute of Technology, Jaipur

hemant.mittal@gitjaipur.com

Ms. Ayushi Shukla

Assistant Professor, Department of CSE, Global Institute of Technology, Jaipur

ayushi.shukla@gitjaipur.com

Mr. Vikash

Assistant Professor, Department of CSE, Global Institute of Technology, Jaipur

vikash.kumar@gitjaipur.com

Mr. Sudhanshu Vaishitha

Assistant Professor, Department of CSE, Global Institute of Technology, Jaipur

sudhanshu.vaishitha@gitjaipur.com

ABSTRACT: This paper presents the design, development, and deployment of a Skillify, a web-based platform aimed at empowering developers to write, test, and debug HTML, CSS, and JavaScript code in a seamless and portable environment. The editor is implemented using ReactJS, a modern JavaScript library known for its efficiency and component-based architecture, ensuring a fast, responsive, and user-friendly interface. By integrating features such as real-time code rendering, syntax highlighting, error detection, and auto-completion, the platform provides a streamlined workflow for both novice developers and experienced professionals. The Skillify is designed to address the limitations of traditional local development environments, such as lack of portability and dependency on specific hardware or software installations. Hosted on cloud platforms like Netlify and Heroku, the tool ensures accessibility from any modern web browser and across devices, making it ideal for coding on-the-go or collaborative projects. The project also emphasizes scalability and simplicity, allowing developers to prototype and refines their ideas with minimal setup. Through its lightweight and flexible design, the editor supports real-time debugging and user-friendly interaction, showcasing the potential of ReactJS in modern web application development. With an emphasis on educational and professional use cases, the Skillify contributes to the growing trend of accessible and efficient cloud-based development tools, bridging the gap between local editors and online solutions.

KEYWORDS: HTML, CSS, React, Web Development, Code Editor.

1. INTRODUCTION

The rapid growth of web technologies has significantly increased the demand for efficient and accessible development tools. Traditionally, developers rely on local Integrated Development Environments (IDEs) and code editors installed on personal computers to write,

test, and debug web applications. While these tools offer powerful features, they often require specific hardware configurations, software installations, and regular updates. Such dependencies can limit accessibility and portability, especially when developers need to work across multiple devices or collaborate remotely.

With the advancement of cloud computing and modern web frameworks, web-based development environments have emerged as a practical alternative to traditional local editors. Online code editors allow developers to write and execute code directly in a web browser without requiring complex setup processes. These platforms provide flexibility, portability, and ease of use, making them particularly beneficial for students, beginners, and professionals who require quick prototyping and testing environments. In this context, Skillify is proposed as a web-based coding platform designed to simplify front-end web development. The platform enables users to write, test, and debug HTML, CSS, and JavaScript code in a single integrated environment. Developed using ReactJS, a widely adopted JavaScript library known for its component-based architecture and high performance, Skillify offers a responsive and interactive user interface that enhances the coding experience.

The editor incorporates several essential features such as real-time code rendering, syntax highlighting, error detection, and auto-completion, which assist developers in writing efficient and error-free code. By integrating these features within a browser-based environment, Skillify eliminates the need for installing multiple development tools and provides a seamless workflow for users. Another significant advantage of Skillify is its cloud-based deployment, which ensures that the platform is accessible from any device with an internet connection. Hosting services such as Netlify and Heroku enable the application to operate efficiently while maintaining scalability and reliability. This approach supports the growing trend of cloud-based development tools that emphasize accessibility and collaborative development.

The primary objective of this project is to create a lightweight, efficient, and user-friendly web-based code editor that can support both learning and professional development activities. By combining modern web technologies with a simple and intuitive interface, Skillify aims to bridge the gap between traditional desktop editors and fully cloud-based development platforms. The remainder of this paper discusses the system design, implementation, key features, and performance evaluation of the Skillify platform, highlighting its potential contribution to modern web development environments.

2. SYSTEM DESIGN AND METHODOLOGY

The performance of low-power CMOS Operational Transconductance Amplifiers is evaluated using several key parameters that directly influence their suitability for specific applications.

Node.js

Node.js is a software platform which helps to build asynchronous and event-driven network applications. It contains built-in HTTP server libraries which allow developers to create their own web server and build highly scalable web applications on top of it. The V8 JavaScript runtime engine used by Node.js is the same engine used in Google's Chrome browser.

The V8 engine complies the codes directly to the native machine code leaving out the interpreter and byte code execution process, which gives Node.js a huge boost in performance. In addition to the V8 engine, Node.js is composed of several components as shown in figure 1.

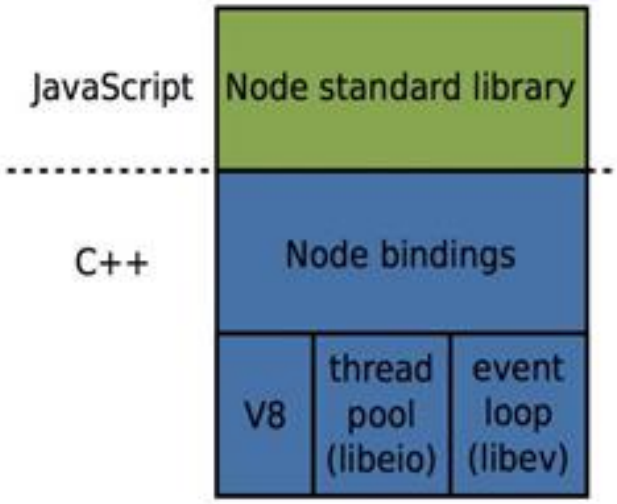

Figure 1: Node.js Architecture

Figure 1 illustrates the architecture of Node.js. Node.js is composed of Node standard library at the top, thin C++ node bindings in the middle, and V8 engine, libeio and libev at the bottom. Node standard JavaScript library exposes operating system features to the application, while the C++ bindings expose the core APIs of the underlying elements to JavaScript. The V8 engine provides the run-time environment for the application, and libeio handles the thread pool to make asynchronous (non-blocking) I/O calls to label, the event loop.

The asynchronous, non-blocking I/O feature of Node.js plays an important role in resource management and performance enhancement of Node.js applications. Unlike other mainstream servers like Apache and IIS, Node.js uses single-threaded, non-blocking I/O operation. What this MERNs is that instead of running each session (request) in a separate thread and providing an associated amount of RAM for each session, Node.js uses a single thread to execute all requests and implements an event loop to avoid blocking of I/O. In a multi-threaded server, as the number of threads increase, the server overhead (scheduling and memory footprints) increases resulting into a slow performance of the overall system. Node.js uses an event-driven and non-blocking I/O approach to handle all the requests to the server.

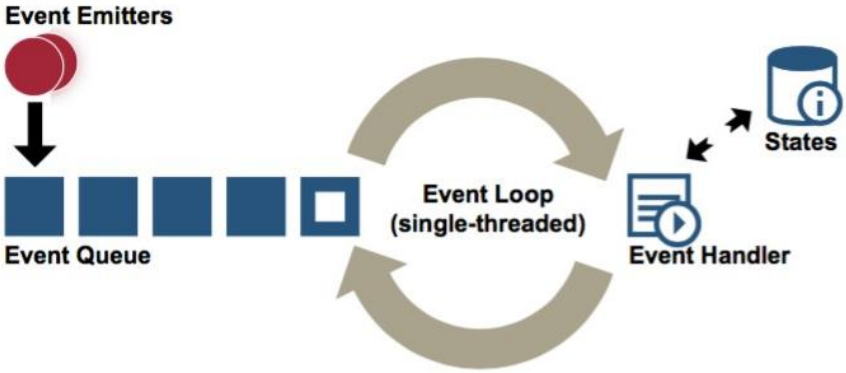

Figure 2: Node.js processing model

In figure 2, Node.js creates an event-loop with event handlers for all requests. When an I/O operation occurs, the associate handler is queued up for execution and a callback function emits an event after the I/O operation is completed. In the MERN-time, other I/O operations keep running outside the event loop of the server. Thus, Node.js performs the I/O operations asynchronously and does not block any script execution, allowing the event loop to respond to other requests. One common use of a callback function displaying the non-blocking I/O operation of Node.js is shown in below code.

```
var fs = require("fs");

fs.readFile('input.txt', function (err, data) { if (err) return console.error(err);
console.log(data.toString());

});

console.log("Reading file");
```

Above code illustrates the non-blocking code example of Node.js. The program is trying to perform three operations: read a file 'input.txt', print the file's data in console, and print message "Reading file" in console. The function readFile() executes first in the order of appearance but the program does not wait for file-reading function to complete. Once the function starts to read. Hence, there is no blocking of I/O operations and all the operations execute asynchronously.

Node.js is actually the foundation of the MERN stack. All the other technologies work on its foundation. The event loop is used by all the available modules and libraries in the Node.js which makes the I/O operations efficient. This Node.js feature maintains the speed and efficiency of the overall system.

Table 1: Performance Benchmark of Node.js, Python-Web, and PHP

Web Development Technology	Value of Fibonacci	Mean Request	Mean Time Per Request
Node.js	Fib(10/20/30)	2491.77/1529/58.85	0.401/0.654/16.993
Python-Web	Fib(10/20/30)	633.68/209.89/2.9	1.578/4.764/345.307
PHP	Fib(10/20/30)	2051.168.8/1.78	0.488/5.942/560.533

Table 1 illustrates the performance results of the Node.js in comparison with Python Web and PHP when calculating Fibonacci (10/20/30). The study was conducted by the researchers in Peking University, China and results were published on the technical report by IEEE. It is clearly seen that the Node.js is better in handling requests and performing calculations than the other two technologies.

Express

Express is a node module which provides a minimal and flexible framework for Node.js web applications. It works on top of the core node modules without hiding any of the features of Node.js. The use of Express framework on top of Node.js helps to maintain clarity of the code. It also makes module integration easy to handle, and provides a solution structure for applications. Express is

app.METHOD (PATH, HANDLER) where:

- app is an instance of express
- METHOD is an HTTP request method
- PATH is a route path (URI)
- HANDLER is the function which executes when the route is matched.

Middleware in Express are the functions that have the pattern function (req, res, next). The req is the incoming request from the client, res is the response from the server, and next is the callback function.

Therefore, middleware functions execute any code inside it, handle request and response objects, end request-response cycle, and call the next middleware function.

A current middleware function must always call the next middleware function, even in the case of incomplete request-response cycle to avoid request hanging.

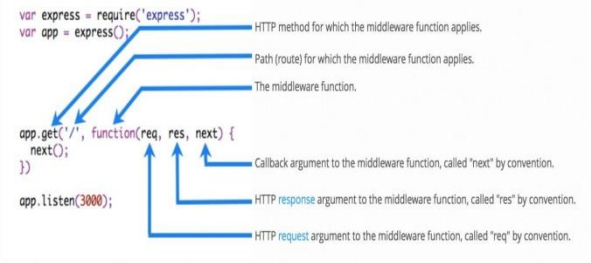

Figure 3: Express web server

Figure 3 illustrates the three building blocks of an Express application: router, route, and middleware. The first two lines uses Node.js require() method to load express module in the application by creating the app object. The third line is a simple router with a route to '/' location and a middleware function. The application listens to port 3000 for any request.

Mongo DB

MongoDB is an open-source, non-relational, document database. It deviates from the need of creating Object Relational Mapping (ORM), for rapid application development. Unlike the

relational databases, it does not contain columns and rows. However, the concept of rows still exists in MongoDB but it is called a document. The document defines both itself and the data it contains. The format in which the MongoDB stores the data is called BSON, which stands for binary JSON. Since JSON is the JavaScript way of storing data, MongoDB works perfectly with the applications built with JavaScript stack. A basic example of a MongoDB document is illustrated below.

```
{
 "challenge": "Two Sum",
 "challengeId": "env52279effc62ca8b0c1000007",
 _id: ObjectId("52279effc62ca8b0c1000007")
 }
```

This illustrates a code snippet from a MongoDB collection. The document stores the first and last names of a customer. Unlike traditional relational databases, MongoDB does not hold a data set corresponding to a set of columns, instead it uses the concept of name-value pair to store data. The _id is the unique identifier (primary key) for that set of data (document). There are some variations in the naming terms of relational database and MongoDB.

Table 2: Comparison of MySQL and MongoDB terms

My SQL	Mongo DB
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Join	Embedded Documents and Linking
Primary Key	Primary Key
Group Key	Aggression

As illustrated in table 2, in MongoDB, some MySQL terms like table is called collection, and row is called BSON document. Instead of Join operation, MongoDB embeds sub documents inside the main document and provides links to the sub documents. MongoDB, like MySQL uses unique identifier (primary key) for each document so that it is easy to query and find the data. MongoDB supports insert, query, update, and delete operations like any other databases. Other important features of MongoDB are auto-sharing and replication. Since the data are stored in a document, they can be stored across multiple locations. As the size of databases increase, Likewise, the data can also be replicated to another data server so that the data are always available in case one server fails. This allows to build highly scalable and efficient data servers in comparison to relational databases such as MySQL and Oracle databases.

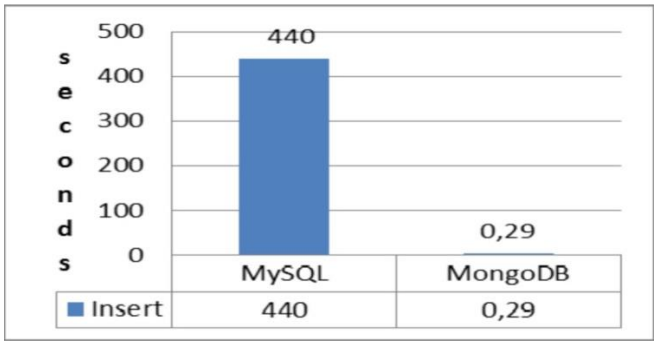

Figure 4: MySQL vs MongoDB insert

Figure 4 illustrates the time taken by MySQL and MongoDB when running a script to insert 10,000 users' data. MySQL which uses traditional database approach took significantly longer time than MongoDB. It shows that MySQL took 440 seconds to insert 10,000 users, while MongoDB only took 0.29 seconds to perform the same task. It proves that the MongoDB is excellent in performing large read-write operations.

Mongoose is a MongoDB data modelling for Node.js. It was created to solve th It can manage the connections to MongoDB database, as well as read, write, and save data. It also allows only valid data to be saved in MongoDB by providing validation rules at the schema level.

ReactJS

Virtual-DOM: Virtual-DOM is a JavaScript object; each object contains all the information needed to create a DOM, when the data changes it will calculate the change between the object and the real tree, which will help optimize re-render DOM tree. It can be assumed that is a virtual model can handle client data.

Component : React is built around components, not templates like other frameworks. A component can be created by the create Class function of the React object, the starting point when accessing this library.

ReactJS creates HTML tags unlike we normally write but uses Component to wrap HTML tags into stratified objects to render. Among React Components, render function is the most important. It is a function that handles the generation of HTML tags as well as a demonstration of the ability to process via Virtual-DOM. Any changes of data at any time will be processed and updated immediately by Virtual-DOM.

Figure 4: Example of website in ReactJs

Props and State

Props: are not controlled by Component, actually stands for Properties. The title = “Title” line creates a name attribute with the value "Title". It looks like a function call. It is true that props are passed to the component in the same way that an argument is passed to a function.

A component may also have a default props, so if the component does not pass any props, it will still be set.

State: private data is controlled by Component. Like props, sate also holds information about the component. However, the type of information and how to handle it varies. State works differently than Props. The state is a comp. Use set State to re-renders one component and all its children. This is great, because you do not have to worry about writing event handlers like any other language.

Pros and Cons of ReactJS

Pros of ReactJS:

• Update data changes quickly.

• React is not a framework so it offloads the constraints of libraries together.

• Easy access to who understands JS.

Cons of ReactJS:

• ReactJS only serves the View tier, but the library size is on par with Angular while Angular is a complete framework.

• Incorporating ReactJS within common MVC frameworks demands reconfiguration.

• Hard to reach for beginners on website developing.

Understanding MVC (Model-View-Controller)

MVC is a software architectural pattern commonly used for developing user interfaces that separates an application into three interconnected components:

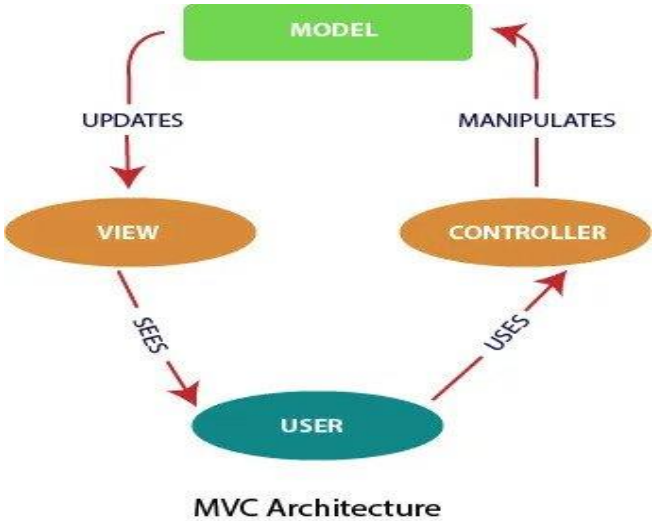

Figure 5: MVC Architecture of ReactJs

Model

- Represents the data and the business logic of the application.
- Directly manages the data, logic, and rules of the application.
- It responds to requests for information and responds to instructions to update itself (e.g., a database record).

View

- Represents the UI (User Interface).
- Displays data from the Model to the user.
- Sends user interactions (like button clicks) to the Controller.

Controller

- Acts as an intermediary between Model and View.

Receives input from the user via the View, processes it (e.g., with business logic from the Model), and updates both the Model and the View accordingly..

3. IMPLEMENTATION

Project Description

Learn.io is the name of the prototype web application that was developed to illustrate the implementation of the MERN stack in a real-life project. The web application provides minimalistic features of an online teaching platform. The application provides a basic foundation of an online teaching platform which can be extended later as a full e-commerce website to provide paid challenges and tutorials.

After analyzing the requirements, the following interfaces were implemented.

- An interface for admin to create challenges, add description, embed video URLs, and other course materials.
- An interface for user to browse challenges, view challenges' teaser videos, and request the full video playlists providing the email address.
- An interface for admin to manage the challenges, users and requests.

In addition to the above requirements, the application should also meet the following best practices set by the standard MERN application:

- The application should be built with the MVC architecture.
- The application should be a Single Page Application (SPA).
- The application should consist of REST APIs for data communication.
- The application should follow the responsive mobile-first design principles.

Based on the requirements, the application was built in the MVC architecture. There is a complete separation between the server-side and the client-side logic. The model is

implemented in the server. The application uses the routes created in ReactJs to navigate throughout the web pages but all data are served by updating a single page. This SPAREST API approach and the MVC architecture of the application are illustrated in figure 7 and figure 8.

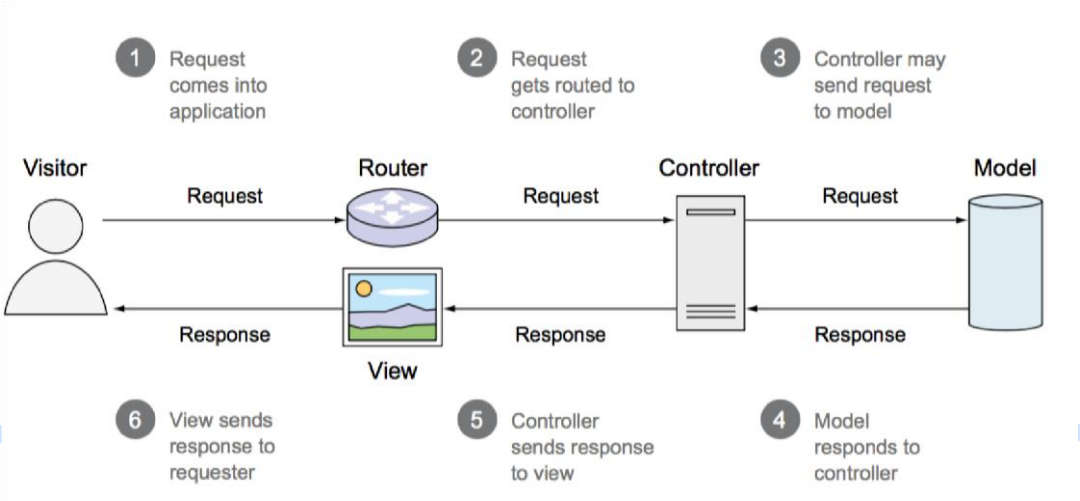

Figure 7: Request-response flow in MVC application

Figure 7 illustrates the MVC architecture of the application. It also illustrates the routing mechanism the application uses to access specific web pages in the application. A request from a visitor is routed to the controller. The controller, if needed, makes a request to the model and the model responds to the controller. The controller then sends a response to the view and the view renders the page on the visitor's browser.

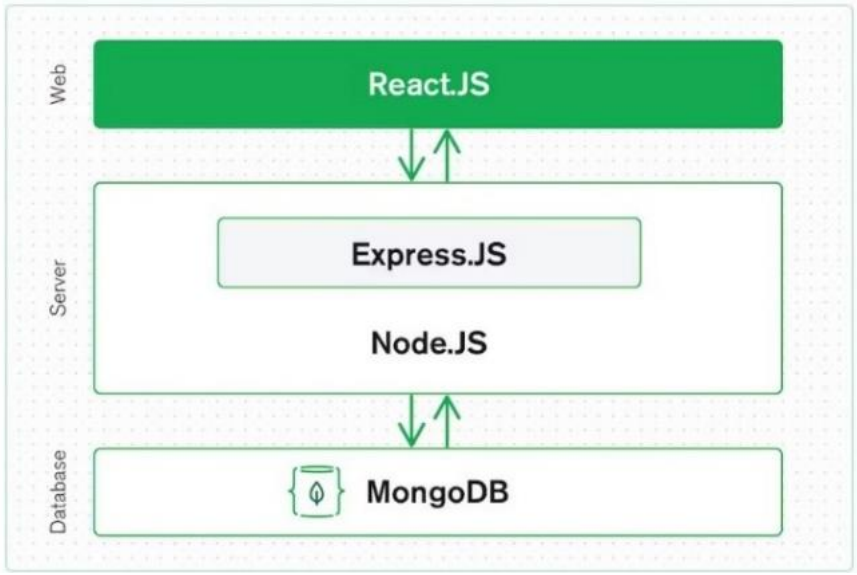

Figure 8: MERN stack architecture of Skillify

Figure 8 illustrates a common MERN stack architecture that the application follows. A REST API is built using MongoDB, Express and Node.js, and a Single Page Application (SPA) is created using ReactJs. The REST API then feeds JSON data to the ReactJs SPA on the browser.

Development Environment Setup

The application was developed in Microsoft running the operating system OS X El Capitan 10.11.6. Besides the hardware, several programs and tools were downloaded and configured to set up an environment for the MERN application.

WebStorm IDE (Integrated Development Environment)

WebStorm IDE is a lightweight and powerful IDE which facilitates the development of a MERN application. It has built-in git support and embedded CLI. It has also built-in support for the latest technologies such as Node.js and ReactJs. It can be downloaded from the official website: https://www.jetbrains.com/webstorm.

Node.js

Node.js is the first program to be downloaded since its provides the platform on which a

MERN application is built. It can be downloaded from the website: https://nodejs.org/en/download.

After the download and installation, it can be checked by issuing node –v command from the terminal or CLI .

Express

After the installation of Node.js, a root directory is made and inside the directory, express is installed issuing command from the terminal npm install –g express. An express generator can also be installed to provide simple structure to the application. All the node modules and dependencies were installed using npm install command.

MongoDB and Mongoose

MongoDB is installed from the official website available at this URL address that is, https://www.mongodb.com/downloadcenter#community.

After locating the downloaded folder, MongoDB server can be started with simple command from the terminal. Mongoose can be installed using the command: npm install mongoose.

Implementation

After the installation and configuration of all the required tools and programs, the development process was started.

Figure 9 illustrates an overview of the application's folder structure as seen in WebStorm IDE. The 'package.json' file contains all the dependencies required to run the application. The 'index.js' file (~/MERNapp/index.js) contains all the Express middleware, and API routes of the application. There is a clear division between the server-side logic and the client-side logic. All the client-side scripts are placed under the 'public' directory. The data models are placed in the server-side, whereas views and controllers are placed on the client-side. All the server-side dependencies are installed in 'node_modules' folder and all the client-side dependencies including ReactJs are installed in 'bower_components' folder.

package.json

The ‘package.json’ file contains all the important information about the application and the dependencies required to build and run the application.

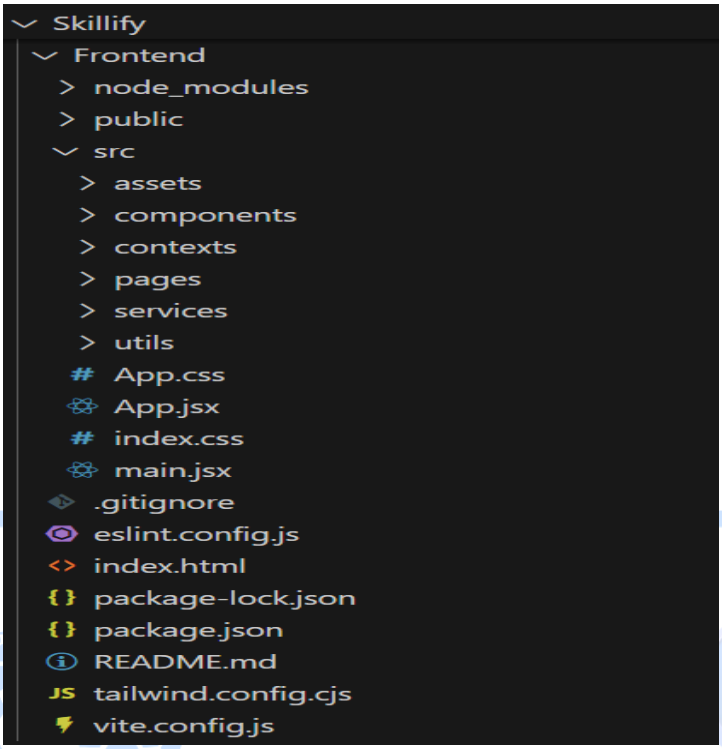

Figure 9: Web Application folder frontend Structure

Figure 10: package.json file

Figure 10 illustrates the 'package.json' file's contents which provides overall information about the application.

It contains the application name, version, and all the dependencies. On line 6, the code shows that the application needs the scripts on /bin/www file to start the server and it can be started with the node ./bin/www command.

index.js (Server-side)

Figure 11 illustrates the routing mechanism in ReactJs. The routes refer to the pages to perform CRUD operations. The first line of code adds ngRoute as a dependency in the application module and then uses $routeProvider to configure the routes. The contents provided by the routes are put inside a ng-view directive in index.html. The ng-view directive then acts as a place holder for all the available views, thus creating a Single Page Application. When the user provides one of the routes, the related controller is invoked and the view is rendered inside the ng-view directive of the index.html without page reloads.

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/login" element={<Login />} />
  <Route path="/register" element={<Register />} />
  <Route path="/challenges" element={<ChallengeList />} />
  <Route path="/challenges/:id" element={<ChallengeDetail />} />
  <Route
    path="/coding/:id"
    element={
      <ProtectedRoute>
        <CodingEnvironment />
      </ProtectedRoute>
    }
  />
  <Route
    path="/dashboard"
    element={
      <ProtectedRoute>
        <Dashboard />
      </ProtectedRoute>
    }
  />
  <Route path="/leaderboard" element={<Leaderboard />} />
  <Route
    path="/profile"
    element={
      <ProtectedRoute>
        <Profile />
      </ProtectedRoute>
    }
  />
  <Route path="*" element={<NotFound />} />
</Routes>
```

Figure 11: Routes in ReactJS

Controllers in ReactJs

The ReactJs controller module is used to consume the REST APIs when the user provides the appropriate routes (URIs) and shows the results to the user.

```
const express = require('express');
const Challenge = require('../models/Challenge');
const Submission = require('../models/Submission'); // Add this for user challenge queries
const User = require('../models/User');
const { auth, admin } = require('../middleware/auth');
const { validateChallenge } = require('../middleware/validation');

const router = express.Router();
```

Figure 12: challenge.js controller

Figure 12 illustrates a code snippet from the Course.js controller class. The code snippet illustrates the GET request method for all challenges. The controller module uses the $HTTP. Get method to consume the REST service at the given URI (/Api/v1/challenges). It assigns the JSON returned back from the API to the $scope. Challenges, which sets a model object named challenges. ReactJs then binds challenges object to the application's DOM rendering it on the user's browser.

4. RESULTS AND DISUSSION

The Skillify was successfully developed using modern frontend technologies like ReactJS, Vite, and TailwindCSS. This application demonstrates the power of a frontend-only architecture, leveraging the flexibility of React for building dynamic user interfaces and real-time code previews without requiring a backend server.

The application allows users to write and test HTML, CSS, and JavaScript directly in their browser, with changes reflected instantly in a live preview. Additionally, users can download their code as individual files (HTML, CSS, JS) or save them together in a ZIP file, offering seamless interactions without the need for a server-side setup.

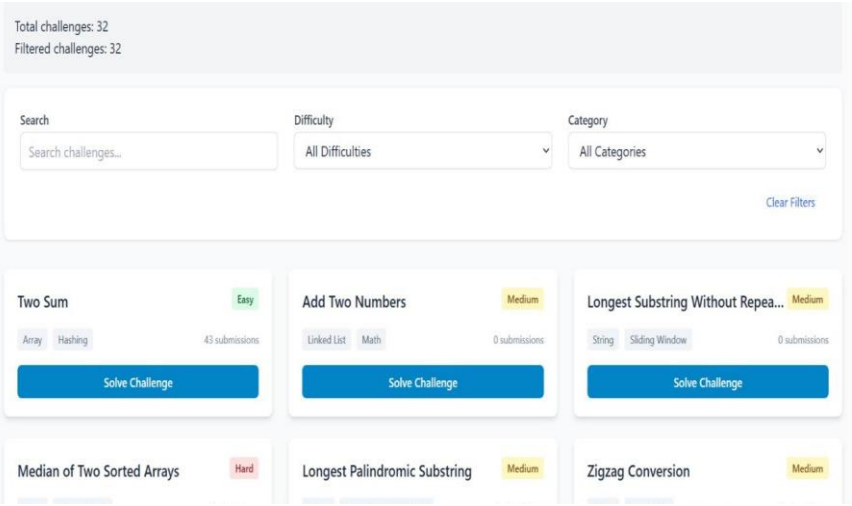

Figure 13: Skillify Interface

Figure 13 illustrates the code editor interface where users can enter HTML, CSS, and JavaScript code in separate panels. The editor uses ReactJS to handle the dynamic rendering of code changes and a live preview of the output. As users modify the code, the Preview component updates the display without reloading the page, providing a fast and fluid experience.

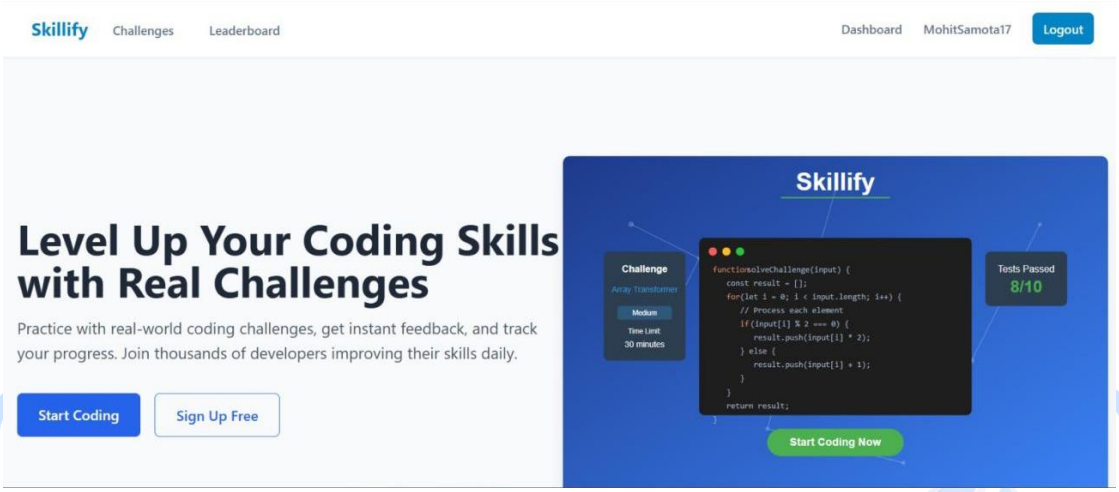

Figure 14: Responsive Design

Figure 14 demonstrates the responsive design of the Skillify. Using TailwindCSS, the application is fully responsive, adjusting its layout to fit different screen sizes, from desktop monitors to mobile phones. This ensures that users have a seamless experience, whether they are using the editor on a desktop or a smaller device.

5. CONCLUSION

The primary goal of this project was to study the feasibility and effectiveness of building a fully browser-based Skillify using modern frontend technologies like ReactJS, Vite, and TailwindCSS. The project focused on implementing a lightweight, responsive, and interactive web application that eliminates the need for server-side support while providing key features like real-time code preview, syntax highlighting, and file download capabilities.

However, it is important to acknowledge the limitations of this architecture. The lack of backend support means that user data cannot be stored or shared persistently without additional mechanisms like browser-based storage or cloud integrations. Furthermore, features like collaboration or user authentication would require backend services or third-party integrations to extend the application's capabilities.

Despite these limitations, the Skillify excels in providing a simple and effective tool for quick prototyping, personal development, and teaching environments. It highlights the potential of frontend technologies to create applications that are not only functional but also easy to use and accessible across devices. With further improvements, such as offline support, cloud synchronization, and advanced editor features, the project can evolve into a robust development platform.

In conclusion, the "Skillify" successfully showcases how modern frontend technologies can empower developers to build scalable and responsive applications while eliminating the complexities of backend management. This approach is particularly useful for educational tools, prototyping applications, and lightweight development environments, making it a valuable addition to the ecosystem of web-based tools.

In conclusion, Skillify successfully showcases how modern frontend technologies can empower developers to build scalable and responsive applications while eliminating the complexities of backend management. This approach is particularly useful for educational tools, prototyping applications, and lightweight development environments, making it a valuable addition to the ecosystem of web-based tools. With continued improvements, Skillify can evolve into a fully functional, collaborative, and cloud-powered coding platform.

REFERENCES

[1] "Dana Nourie. Java technologies for Web Applications [online]". Oracle official website http://www.oracle.com/technetwork/articles/java/webapps-1-138794.html.

[2] "Web Applications: What are They? What of Them [online]". Acunetix official web-site URL: http://www.acunetix.com/websitesecurity/web-applications/

[3] "Mimi Gentz. NoSQL vs SQL [online]". Microsoft Azure official website; URL: https://azure.microsoft.com/en-us/documentation/articles/documentdbnosql-vs-sql/

[4] "Rodriguez A. RESTful Web services: The basics [online]". IBM official website; 9 February 2015 URL: https://www.ibm.com/developerworks/library/ws-restful/

[5] "MongoDB official website [online]." URL: https://docs.mongodb.com/manual/

[6] "Express official website [online]." URL: https://expressjs.com

[7] "MongoDB documentation official website [online]." Available at this online website URL: https://docs.mongodb.com/getting-started/shell/introduction/

[8] A. Chauhan, R. Misra, "Outline of Web Development Life cycle in Software Engineering", International National Conference on Recent Trends in Engineering & Technology, 2023.

[9] A. Bohra, K. Paliwal, S. Soni, "Online code editor: A cloud-based platform for real-time web development," International Journal of Global Research in Science and Technology, vol. 9, pp. 52–76, 2024.

[10] P. Upadhyay, K. K. Sharma, R. Dwivedi and P. Jha, "A Statistical Machine Learning Approach to Optimize Workload in Cloud Data Centre," 2023 7th International Conference on Computing Methodologies and Communication (ICCMC), pp. 276-280, 2023.

[11] A. Kalwar, R. Ajmera, and C. S. Lamba, "An empirical study in small firms for web application development and proposed new parameters," International Journal of Innovative Technology and Exploring Engineering, vol. 8, no. 4, Feb. 2019.

[12] R. Ajmera, A. Kalwar, and C. S. Lamba, "A modern study on progressions and issues of web application development in small firms," International Journal of Scientific Research in Science and Technology, vol. 3, no. 8, pp. 1–6, Nov.–Dec. 2017.

[13] A. Maheshwari, R. Ajmera and D. K. Dharamdasani, "Unmasking Embedded Text: A Deep Dive into Scene Image Analysis," 2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT), pp. 1403-1408, 2023.

[14] R. Ajmera and D. Dharamdasani, "Comparative study of existing food delivery applications," Global Research Journal, pp. 454–463, 2022.

[15] M. K. Sain and N. Sharma, "A study of research issues and challenges of big data analytics," Journal of Advances and Scholarly Researches in Allied Education, vol. 16, no. 5, pp. 1699–1707, 2019.

[16] P. Jha, M. Mathur, A. Purohit, A. Joshi, and A. Johari, "LibUno: A React-based digital platform for smart library management," International Journal of Global Research in Science and Technology, vol. 9, pp. 38–51, 2024.