Online Code Editor: A Cloud-Based Platform for Real-Time Web Development

Amit Bohra
Assistant Professor, Department of CSE, Global Institute of Technology, Jaipur, Rajasthan, India
amit.bohra@gitjaipur.com

Kritika Paliwal
Assistant Professor, Department of CSE, Global Institute of Technology, Jaipur, Rajasthan, India
kritika.paliwal@gitjaipur.com

Dr. Sangeeta Soni
Associate Professor, Department of CSE, Global Institute of Technology, Jaipur, Rajasthan, India
sangeeta.soni@gitjaipur.com

ABSTRACT: This paper presents the design, development, and deployment of an Online Code Editor, 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 Online Code Editor 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 Online Code Editor contributes to the growing trend of accessible and efficient cloud-based development tools, bridging the gap between local editors and online solutions.

KEYWORDS: Online Code Editor, Web Development, React, Digital Platform.

1. INTRODUCTION

The Online Code Editor project is a browser-based application designed to provide a seamless coding environment for writing, testing, and previewing HTML, CSS, and JavaScript code. Built using modern frontend technologies such as ReactJS, Vite, and TailwindCSS, the editor emphasizes simplicity, responsiveness, and accessibility. The project eliminates the need for local software installations, offering developers an intuitive platform to experiment with code across devices.

Key features include real-time code preview, syntax highlighting, file downloads (individually and as a ZIP archive), and a responsive design that adapts to various screen

sizes. The application caters to a wide audience, from beginner programmers to experienced developers seeking a lightweight tool for prototyping.

2. WEB APPLICATION DEVELOPMENT

2.1 Front-end

Web application development is the combination of the front-end and the back-end development. Front-end web development, also known as client-side development, involves the practice of creating Graphical User Interface (GUI) for clients (users) so that the users can interact with the application. It involves the use of primary web technologies and tools such as HTML, CSS, and JavaScript.

HTML is a mark-up language which provides the structure to a web page. It defines how a web page would look like so it can be considered the skeleton of any web application. CSS, on the other hand, is a style sheet language which provides style and visual enhancements to the documents written in HTML. JavaScript is the most advanced language among these technologies. It performs HTML DOM (Document Object Model) manipulation to provide a dynamic interface to users. Moreover, it provides an interactive interface to the users by creating pop-up messages, validating form inputs, and changing the layout based on events like user-input or mouse clicks. All these technologies are controlled by the browser to provide a front-end web interface.

Back-end web development, also known as server-side development, involves the development of computer programs and databases to serve the client. A web application in its primary days did not need to have a front-end but a functioning server-side application was enough for it to be considered a web application. Several changes have been made in this field since then. Today's sophisticated web applications cannot run without both the front-end and back-end services. Back-end technologies usually consist of the programming languages such as PHP, Ruby, Python, Java, Node.js, and different frameworks.

Web application development encompasses all the web technologies related to the frontend and the back-end. In addition, a web application must ensure communication between the client and the server with the use of different communication protocols. Protocols are a set of rules for exchanging messages in a communication network. Protocols vary with different tasks and layers. HTTP (Hypertext Transfer Protocol), TCP/IP (Transmission Control Protocol/ Internet Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are some common examples of the protocols used in web applications A web application, in its most elementary form, sends an HTTP request to a server to establish connection, and the server sends an HTTP response to the client. A typical example of communication in a web application is illustrated in figure 1.

Figure 1 illustrates the communication among the three layers of a web application model. The first layer is a client-side web browser, the second layer is a server-side dynamic content generator, and the third layer is a database server. A user sends an initial request using the HTTP protocol through the browser over the internet to the server. The web server then processes the request by accessing the database server and retrieving the requested data. The web server then sends the response to the user over the internet through the browser. The response usually contains the data requested by the user.

Building a good Database Management System (DBMS) to store information is a crucial part of web application development. DBMS allows the users to create, save, update, and query (search) data in a web application. There are two types of databases: relational and non-relational. Relational databases, also known as SQL databases, are traditional databases which store data into tables in the form of rows and columns. The tables maintain some kind of relation with each other, thus giving it the name 'relational database'. Oracle database, MySQL, and SQL Server are the popular relational databases

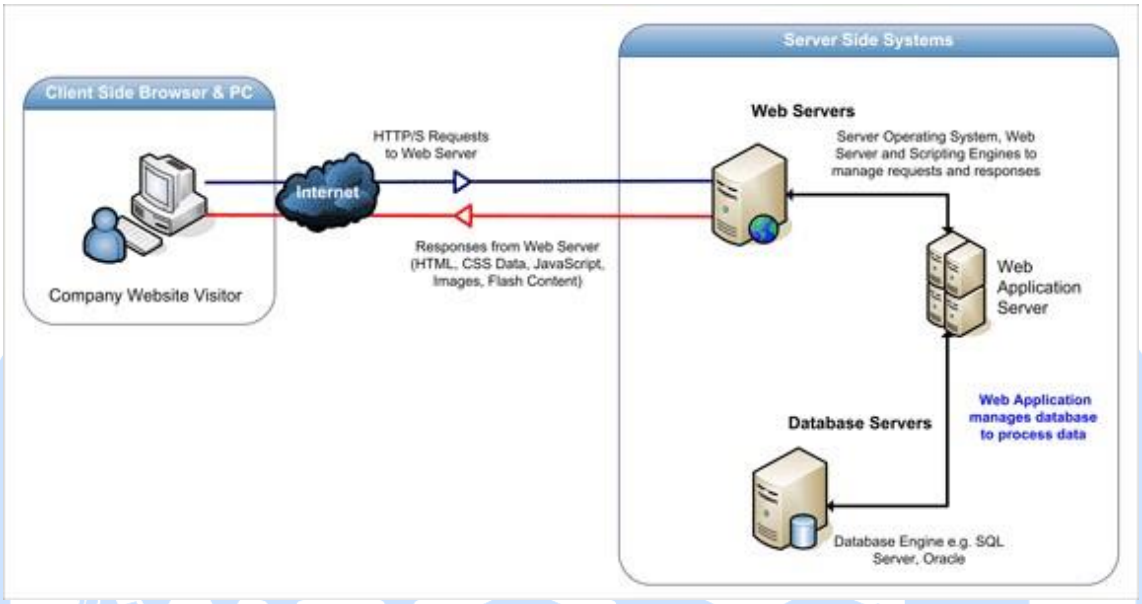


Figure 1: Web Application Model

Non-relational databases, also known as NoSQL databases, store data in forms other than tables. NoSQL databases store data in a key-value pair, graph, document, column, or multi-model depending on the database selected for application development. MongoDB, HBase, Cassandra, Redis, and Riak are some of the examples of popular NoSQL databases. Relational databases are generally used in enterprise applications which can handle large related data, whereas, non-relational databases, which are flexible and scalable in nature, are used in data-driven real-time web applications with rapidly increasing data

In addition to a good database, a good framework for web application development has also become a necessity nowadays. Use of web frameworks in the development of sophisticated web applications is more in practice than the use of native programming languages. Web frameworks, also known as web application frameworks, are designed to ease the task of web application development by providing database-access libraries, templates, session management, and code reuse facility. JavaScript frameworks such as ReactJs, ReactJS, Backbone.js, Ember.js, and Knockout.js are the most popular frameworks in the front-end development. Whereas, PHP frameworks (Laravel, Symfony, CakePHP), JavaScript frameworks (Node.js, Meteor, Express), Rails, Pyramid, ASP.NET, Java EE, Spring, and Django are the most popular frameworks in the backend development.

2.2 RESTful API

REST is an architectural style used in web development in order to create web services. REST only defines the principles on which a web service is developed for the clientserver communication. It is not a set of rules (protocols) for creating web services. Any web services or APIs (Application Programming Interfaces) that are designed with the REST

architecture are called RESTful APIs, or just REST APIs. REST provides good performance, scalability, and reliability in a distributed computing system. There are several constraints for an application to become a REST application. However, a concrete implementation of REST APIs must follow at least four basic design principles:

- Use of HTTP methods: REST APIs must follow the HTTP methods explicitly. They must use GET to retrieve a resource from the server, POST to create a resource, PUT to modify or update a resource, and DELETE to delete a resource.
- Stateless communication: Communication between the client and the server must be stateless, MERNing that every request from the client must contain all the information required for the server to process them. The server should not require any stored data to process the request.
- Use of directory-structure like URIs: REST APIs must use the URIs that are straightforward, properly structured, and easily understood, such as http://www.myservice.org/discussion/topic/{topic}
- Data transfer in XML or JSON: The data transferred between the client and the service-exchange must be in XML or JSON format so the data are properly structured and readable.

In addition, REST web services (APIs) must have a clear separation of client-side logic and server-side logic. A uniform interface separates clients and servers, which allows developers to work on the individual part of web application and improve one without affecting another. Clients and intermediaries should be able to cache server responses to avoid reuse of stale data in response to future requests. Clients also cannot assume a direct connection to the server. In most cases, intermediaries between the client and the server serve the request-response cycle. Furthermore, in REST web services, server may temporarily extend the client by transferring logic to the client.

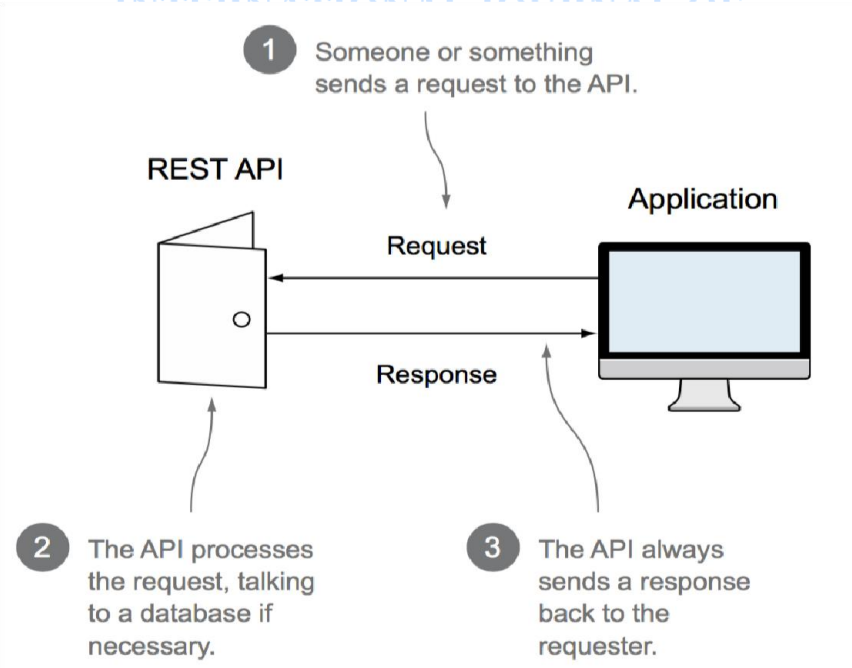


Figure 2: REST API

Figure 2 illustrates the communication between a client-side application and a REST API. The client does not know the implementation of the server and does not directly communicate with the server. The REST API takes all the incoming HTTP requests from the client, processes them, and sends HTTP responses.

2.3 Single Page Application

Single Page Application (SPA) is a web application which fits into a single web page. In contrast to the traditional full page loads, an SPA loads all the resources required to navigate throughout the web application on the first page load. It then dynamically changes the contents as the user interacts with the application, so no full page request will ever be made again. However, URLs are updated in the address bar of the browser with a hash tag following the name of the resources accessed.

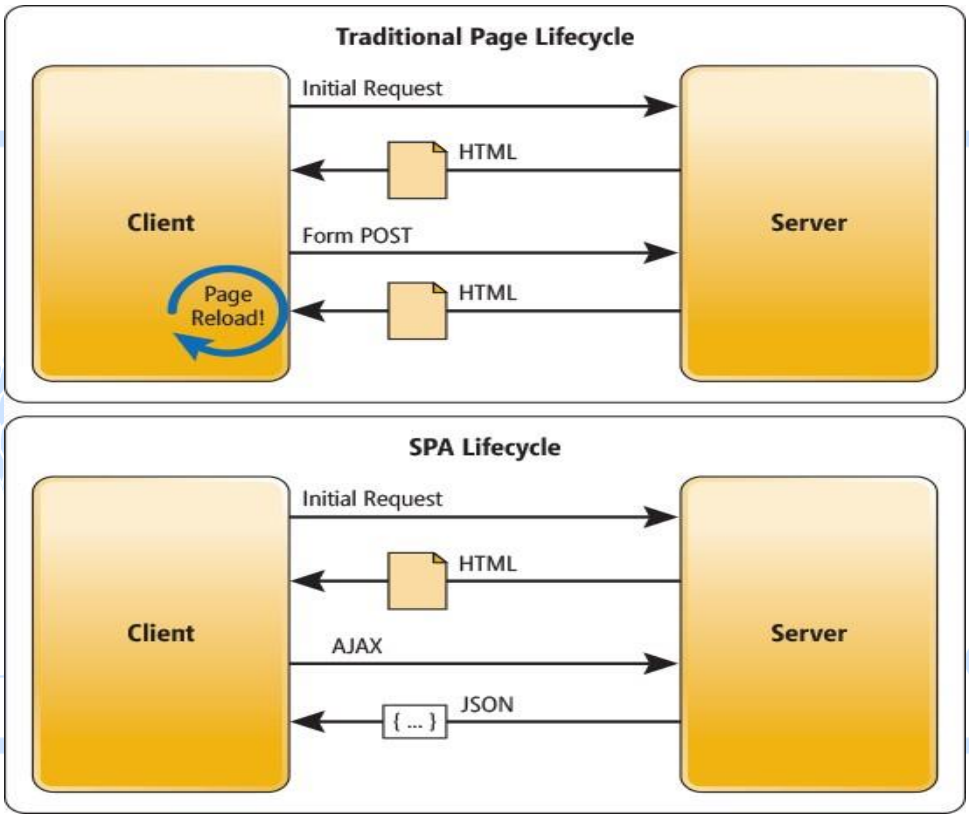


Figure 3: Comparison of traditional page lifecycle and SPA lifecycle

Figure 3 illustrates the distinction between the lifecycle of a traditional web page and an SPA web page. In traditional web applications, every time when a client sends a request to a server, the server will respond by rendering a new HTML page. All the subsequent requests to the server are also processed in a similar fashion and every time a new page is loaded on the client's browser. In an SPA application, after the first page loads, all interactions between the client and the server happen with the AJAX (Asynchronous JavaScript and XML) calls, which in return send data in JSON (or XML) form. The browser then uses the JSON data to update the page dynamically, without any page reloads.

SPAs use AJAX (to interact with the server), HTML templates, MVC frameworks, and JavaScript to perform most of the navigation works on the front-end [23,10]. Modern front-end JavaScript frameworks such as ReactJs, Ember.js, React, and Meteor.js have simplified the tasks of creating SPAs by providing rich DOM manipulation and two-way data binding

features. SPAs provide a rich interface and fluid user experience. Moreover, SPAs make users feel like they were interacting with a desktop application.

2.4 JavaScript

JavaScript is a cross-platform, light-weight, object-oriented scripting language designed primarily for adding interactivity to modern web pages and web applications. It is one of the three main components of the front-end technologies besides HTML (providing structure to the web page) and CSS (providing style to the web page). It was developed by an American programmer Brendan Eich at Netscape in 1995. Originally called by the names 'Mocha' and 'Live Script', JavaScript got its current name in December 1995 with the release of the third beta version.

JavaScript was submitted to ECMA (European Computer Manufacturer Association) for standardization so that other browsers could implement it on the basis of the standardization. ECMA international then published the standard JavaScript language specifications under the name ECMAScript. The current JavaScript specification is based on the ECMAScript 2024, which was released in October 2024.

JavaScript provides rich features to create dynamic and interactive interfaces in client side applications. It can detect the user's browser and OS (Operating System), and performs platform-specific operations. Originally termed as an interpreted language, JavaScript can be executed without preliminary compilation by the browser so it can perform simple calculations on the client-side. It is often used to validate the user's input in forms and send the data asynchronously with the AJAX. It also performs HTML DOM manipulation to provide dynamic interface. It is an object-oriented programming language and supports several built-in objects with inheritance. Thus, JavaScript enables developers to add dynamic features to static HTML pages, control multimedia and add animations. Moreover, JavaScript provides several third-party APIs and libraries to facilitate the task of building dynamic web pages. One of the most popular JavaScript library is jQuery.

JQuery is a library based on JavaScript, developed by an American programmer John Resig. All the functions that can be programmed in traditional JavaScript can be performed in JQuery, but with much simpler methods. Therefore, a function that needs several lines of codes to be written in JavaScript can be easily written in few lines using jQuery. It is almost supported by all modern browsers and does not require extra plugins or extensions to run. One of the main features of jQuery is DOM manipulation. DOM is a tree-like structural representation of the elements in an HTML page. jQuery syntax makes it easy to navigate throughout these elements, find and manipulate them. For example, it can be used to find all elements with a certain property in a web page, change the elements' properties or make them respond to certain events like mouse clicks. This task is difficult to perform in pure JavaScript but jQuery's syntax makes it easier.

For example, in order to change the background colour of the body in an HTML page, one writes the following function in JavaScript:

```
function changeBackground(color) {
document.body.style.background = color;
}
onload="changeBackground('red');"
```

whereas, the same task can be performed in one line of code using the following jQuery function:

```
$('body').css('background', '#ccc');
```

Moreover, jQuery provides simpler functions to create animations, handle different events, and develop AJAX applications.

AJAX (Asynchronous JavaScript and XML) is a web development technique to create asynchronous web applications. With the use of AJAX, asynchronous communication can be made between the client and the server. AJAX uses XMLHttpRequest object to communicate with the server, and sends and receives information in XML, JSON, HTML, or even in text file format. XMLHttpRequest is an API that allows asynchronous communication between the client and the server without the need of a full page reload. Hence, AJAX allows to update a certain portion of a web page without page refresh (reload).

2.5. Rise of Full Stack JavaScript

JavaScript in its primary days earned a bad reputation because of the lack of performance and its incompatibilities with the prominent browsers of that time. However, things started to change after large browser vendors invested time and money in improving the language. JavaScript, finally, became the de facto standard of the client-side scripting. While JavaScript remained prominent in the client-side, several new technologies were introduced in the server-side such as PHP, JAVA, .NET, Ruby, and Python. Developers started to realize that use of two separate languages in the development of the client and the server was complicating the tasks of web programming. Several attempts were made to unify the two sides by creating client components on the server and compiling them into JavaScript (for example: ASP.NET web forms and GWT), but they failed. The only solution to this problem was the implementation of JavaScript on the server-side, and Node.js was introduced.

Node.js is actually the backbone of Full Stack JavaScript web development. It finally put the power of JavaScript on the server with the idea of non-blocking programming paradigm. Node.js became popular in a short time due to its easy-to-use components. It allowed the developers to quickly set up a server and start building applications on top of it. Several frameworks started to emerge to facilitate Node.js implementation such as Express and Connect.js. Express became the most prominent one. Node.js ecosystem continued to expand and a package manager like ‘npm’ was introduced.

While Node.js was invading the server, NoSQL database started to gain popularity in the field of database. MongoDB, a NoSQL database, was introduced with the concept of storing data in Binary JSON. Due to the use of JSON, which is a JavaScript way of storing data, MongoDB became the preferred database for Node.js applications. JavaScript also started to evolve on the client-side and new frameworks like ReactJs, Backbone.js, and ReactJS started to take over the use of traditional JavaScript, JQuery, and AJAX for building Single Page Applications. When all the prominent tools to build a Full Stack JavaScript application were ready, additional tools based on JavaScript were introduced to improve the development process such as Mocha.js and Chai.js for application testing, Gulp.js and Grunt.js for automation of build tasks.

Node.js eco-system, with addition of ReactJs on the front-end, Express as the backend framework, and MongoDB as database raised a new term in web application development called MERN (MongoDB, Express.js, ReactJs, Node.js) stack. The word MERN stack is used

to refer Full Stack JavaScript but it is still a subset of the term Full Stack JavaScript. Node.js, Express, and MongoDB are the prominent members of the Full Stack JavaScript. ReactJs, however, can be replaced with Backbone.js, ReactJS, or Ember.js to meet the developers' requirements.

2.6 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 4.

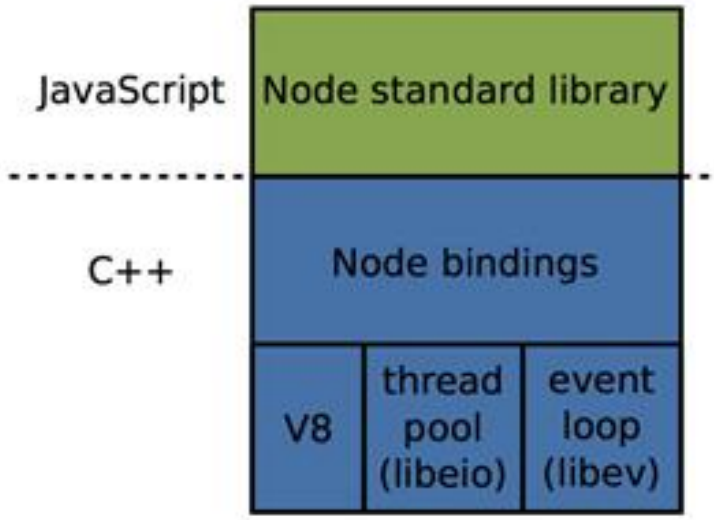


Figure 4: Node.js architecture

Figure 4 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.

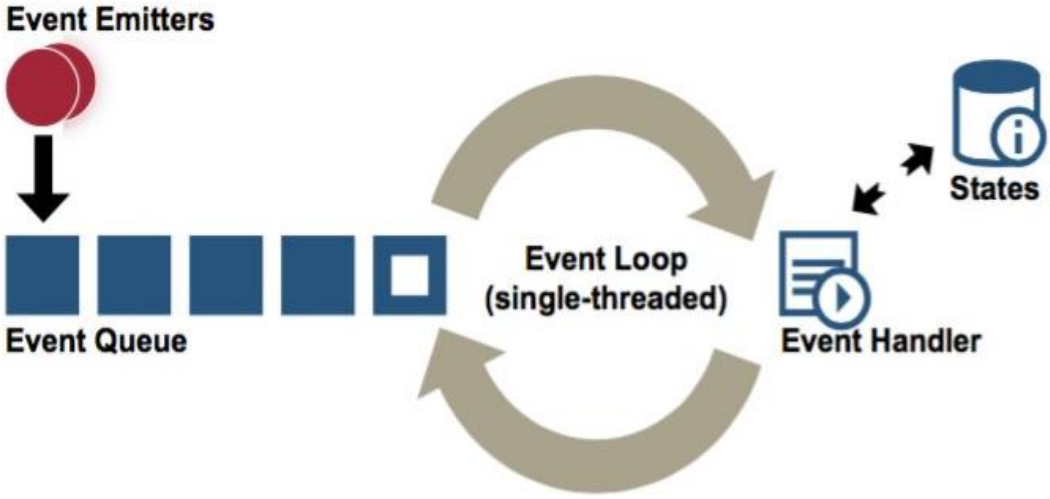


Figure 5: Node.js Processing Model

In figure 5, 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 the file, it passes control to execute the next instruction immediately so the program prints "Reading file" before printing the file's data. Once the file I/O is complete without any error, it will call the callback function function(err, data) and returns the file data as parameters, then prints the data below the 'Reading file' message in the console. Hence, there is no blocking of I/O operations and all the operations execute asynchronously.

Node.js has a rich development eco-system with several compatible libraries and package managers. One of the main package managers that comes pre-bundled with Node.js during installation is npm. Npm is run from the Command Line Interpreter (CLI) and it manages all the dependencies for the Node.js applications. Instead of manually downloading and configuring the JavaScript modules for use in the application, npm provides a simpler alternative. The names of the modules and dependencies can be included with their version numbers in 'package.json' file inside the folder structure, and the modules are downloaded issuing a simple 'npm install' command from the CLI.

The npm automatically handles all the downloading process and save all the named modules in the 'node-modules' folder. The modules can be updated by changing their version number in the 'package.json', and can be easily distributed by uploading a single 'package.json' file to the server instead of uploading the large 'node-modules' folder. After the update or upload, issuing the 'npm install' command installs the specific modules and dependencies. Therefore, Node.js applications are light-weight, flexible and easily sharable.

Node.js is actually the foundation of the MERN stack. All the other technologies work on its foundation. Node.js introduces the power of JavaScript on the server side allowing the developers to create both server side and client-side logic in one single language 'JavaScript'. One of the major advantages of Node.js over other server-side platforms is the built-in event loop. 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.

Web development technology	Calculate value of Fibonacci	Mean requests per second [#/sec]	Mean time per request [ms]
Node.js	Fib (10/ 20/ 30)	2491.77/ 1529.4/ 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)	2051.22/ 168.8/ 1.78	0.488/ 5.942/ 560.553

Figure 6: Performance benchmark of Node.js, Python-Web, and PHP

Figure 6 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. When calculating the small Fibonacci of 10, Node.js and PHP, both perform well, whereas Python-Web lags behind. Node.js handles about 2500 requests per second taking 0.401ms time to handle each request. PHP falls short by only few numbers but the gap in their performances increases drastically when the calculation tends to become complex. When the task reaches to the calculation of Fibonacci of 30, both PHP and Python-Web perform poorly, almost processing 2 or 3 requests per second, and taking 345ms and 560ms to process each request respectively. In contrast to them, Node.js still maintains the quality by processing requests 20 times higher than PHP and taking a relatively short time, about 17ms.

Although Node.js is an ideal choice for a scalable, data-driven, I/O intensive applications, it is not a perfect solution for all applications. It uses JavaScript so it is more efficient when used with other JavaScript-based technologies. The use of single-thread to handle all requests is ideal for some cases but it is not a good feature for intensive data computing applications. Moreover, Node.js uses tight coupling between the web server and the web application so all the classes are dependent on each other. Such tightly coupled system is hard to maintain since a problem in one area causes the whole system to fail. Node.js also does not work well with the relational databases. All these factors must be considered before choosing Node.js as a development platform for any application. Studies have shown that Node.js is ideal for real-time data-driven web applications in collaboration with push technology and web sockets.

2.7 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. In addition, it provides robust and clean functions to add to the node modules so the

development of Node.js application using Express is far easier than using the native node modules. Due to the simplicity of Express, it has been adopted by large companies such as MySpace, Paypal, Apiary.io, Persona, and Ghost. 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 installed using the npm package manager issuing "npm install express" command.

An Express server is made up of three building blocks: router, routes, and middleware. A web server's core functionality depends on its excellent routing methods. In a client-server communication, a client requests some resources from the server, the server locates the resources and responds by sending the resources to the client. This is the core functionality of a web server and it requires excellent routing methods to serve the request. Express makes this tedious job really easy by allowing developers to create routes in simple structure. A route in Express is a combination of a HTTP verb and a path. The HTTP verb is generally one of the four HTTP methods: GET, POST, PUT and DELETE, and the path is the location of the resource (URI). A basic route in Express is created as below:

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. A simple Express web server containing all three building blocks is shown in figure 6.

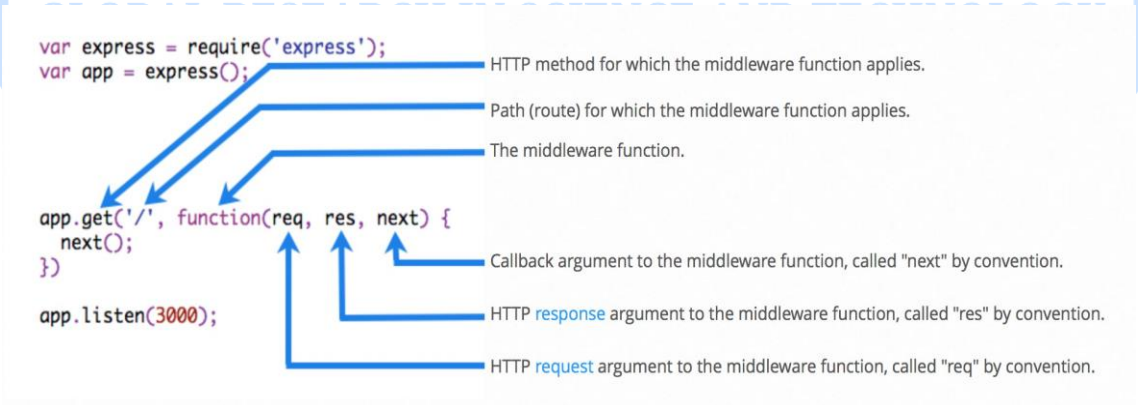


Figure 7: Express web server

Figure 7 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.

Express also provides a simple application generator tool for providing structure to our Node.js application. It can be installed from the CLI by issuing 'npm install express-

generator' command. It also provides the options for creating template engine to write HTML codes. The application generator, by default, creates jade templates for the views, but it also supports Handlebars, EJS, Pug, and Mustache. The application structure created by the Express generator has separate directory for routes, views, and public-rendering files. However, the application structure is just one of many ways to structure an Express application. It can be easily modified during the application development process to meet the requirements of the application.

2.8 MongoDB

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 exits in MongoDB but it is called a document. The document defines both itself and the data it contains. A document is a set of fields and it can contain complex data such as lists, arrays, or an entire document. Each document contains an ID field which can be used as a primary key for a query operation. A set of documents is called a collection and MongoDB holds a set of collections. 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 in listing 2.

```
{
"firstName": "Homer",
"lastName": "Simpsons",
_id: ObjectId("52279effc62ca8b0c1000007")
}
```

Listing 1. Example of a MongoDB document

Listing 1 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.

MySQL	MongoDB
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Join	Embedded documents and linking
Primary key	Primary key
Group by	Aggregation

Figure 8: Comparison of MySQL and MongoDB terms

As illustrated in figure 8, 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.

One of the features that makes MongoDB stand out against the traditional databases is the inclusion of dynamic schema. Collections in MongoDB have different schema and the documents within the same collection can have as many different schema and shapes as required. This feature enables developers to start storing data in the database with any consideration of the database structural design. The documents' keys and values can be changed and updated when required since there is no pre-defined rule governing the data type validation.

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, a single machine may not be able to store data and handle read- write operations. MongoDB solves this problem by allowing horizontal scaling, MERNing that the data are distributed to multiple servers and all servers can work together to support data growth and provide efficient read-write operations. 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.

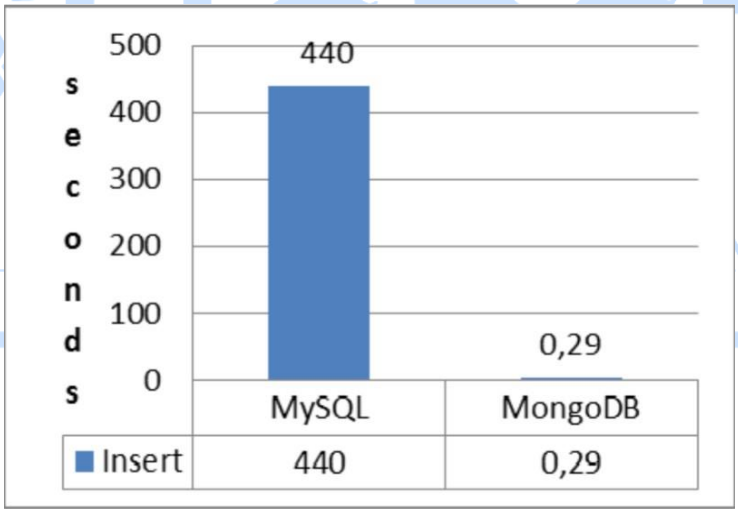


Figure 9: MySQL vs MongoDB insert

Figure 9 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.

Although MongoDB is good at performing majority of tasks, it has its drawbacks. MongoDB cannot take multiple operations as one transaction. If any operation in one transaction fails, it will cause the entire operation to fail. It also can't perform the join operations like MySQL database so it is not a good choice in an application where there are multiple relationships among the data. The data has to be searched and updated in multiple documents at once and

all operations need to be tied in a single transaction to ensure that the data is updated in all collections. Transactional databases work better in such cases than the non-transactional database like MongoDB. Although the flexibility of MongoDB to allow any sort of data is good in some cases, most applications still need certain kind of structure to their data to work properly. To solve this problem, Mongoose was created by the company behind MongoDB.

Mongoose is a MongoDB data modelling for Node.js. It was created to solve the problem of writing complex data validation, casting, and business logic in MongoDB. It provides simple, elegant, data-modelling feature to the application. Using mongoose, one can define what kind of data can be in a document and what data must be in a document. In technical term, it provides data validation rules, query building functions, and business logic to the application data. Moreover, it provides an entire set of new features to use on top of MongoDB. 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.

2.9 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.

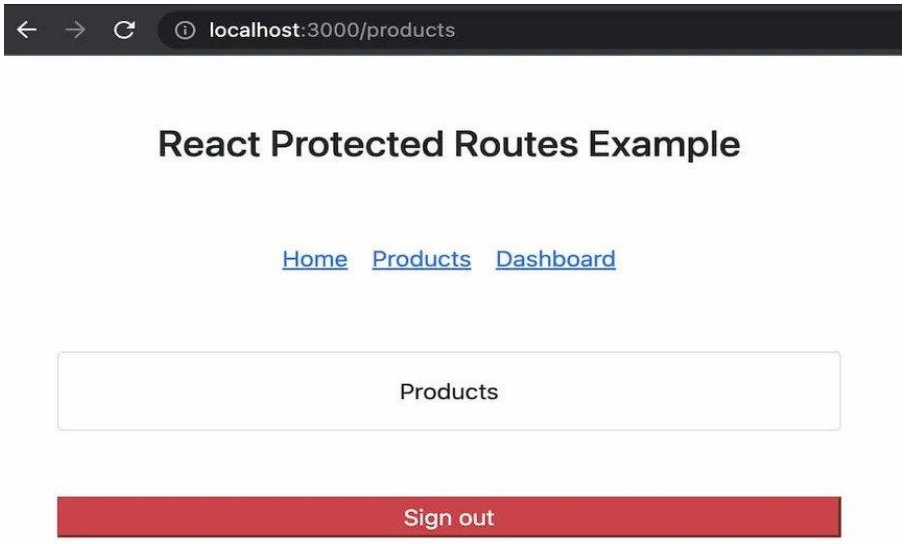


Figure 10: Example of website in ReactJs

- **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 component of the component, while props are passed in from the outside into the component. It should be noted that we should not update the state directly using this. state but always use set State to update. Update the state of the objects. 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.

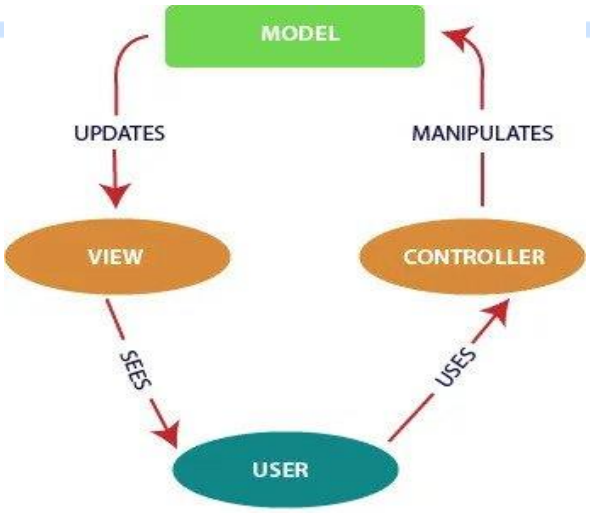
- **Pros and Cons of ReactJS**

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.



MVC Architecture

Figure 11: MVC Architecture of ReactJs

3. IMPLEMENTATION

3.1 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. An admin can create courses, provide course descriptions, embed course videos, provide tutorials and other course materials. A user can simply browse the courses and request for full course videos. 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 courses and tutorials.

After analysing the requirements, the following interfaces were implemented.

- An interface for admin to create courses, add description, embed video URLs, and other course materials.
- An interface for user to browse courses, view courses' teaser videos, and request the full video playlists providing the email address.
- An interface for admin to manage the courses, 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, and the view and the controller are implemented in the client. Moreover, the application does not communicate directly to the server, but uses REST APIs to feed the data. 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 12 and figure 13.

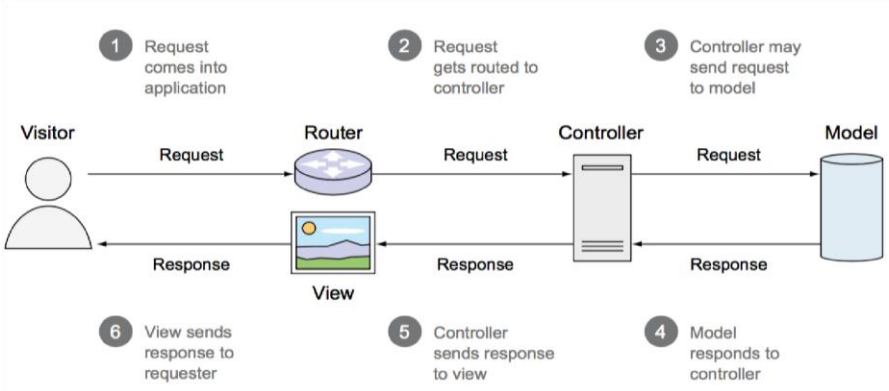


Figure 12: Request-response flow in MVC application

Figure 12 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.

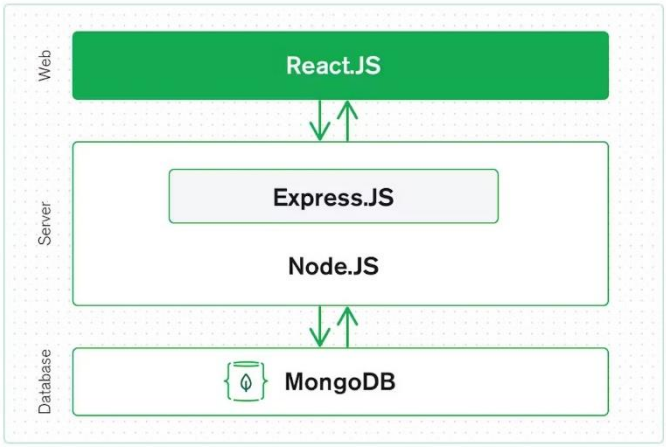


Figure 13: MERN stack architecture of learn.io

Figure 13 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.

3.2 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 as illustrated in figure 14.

```
helios:~ anuj$ node -v
v4.5.0
helios:~ anuj$
```

Figure 14: Node.js version check

Figure 14 illustrates the command to view the Node.js version installed on the computer. The version used in the application is 4.5.0. It confirms that Node.js is properly installed in the development machine.

3.3 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.

3.4 MongoDB and Mongoose

MongoDB is installed from the official web site:

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.

ReactJs, React Routes, JQuery, Bootstrap (front-end packages)

There are several ways to download the front-end packages. In this application, bower was used to download and configure the front-end package. Bower is the package manager for the front-end, whereas npm is the package manager for the back-end. All the modules and dependencies were downloaded using npm and bower.

nodemon

nodemon is a utility that monitors the changes in the source code during development and automatically restarts the server. It is downloaded using the command: npm install –g nodemon, and implemented using the nodemon command.

3.5 Implementation

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

Figure 15 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 'app.js' file (~/MERNapp/app.js) contains all the Express middleware, and API routes of the application. The 'www' file contains the command to start server at port 3000. 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 models, views and controllers are separated as per the MVC architecture. 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 16 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.

app.js (Server-side)

The app.js file is the main configuration file of the Express application. It configures the server and imports all the modules to run the application.

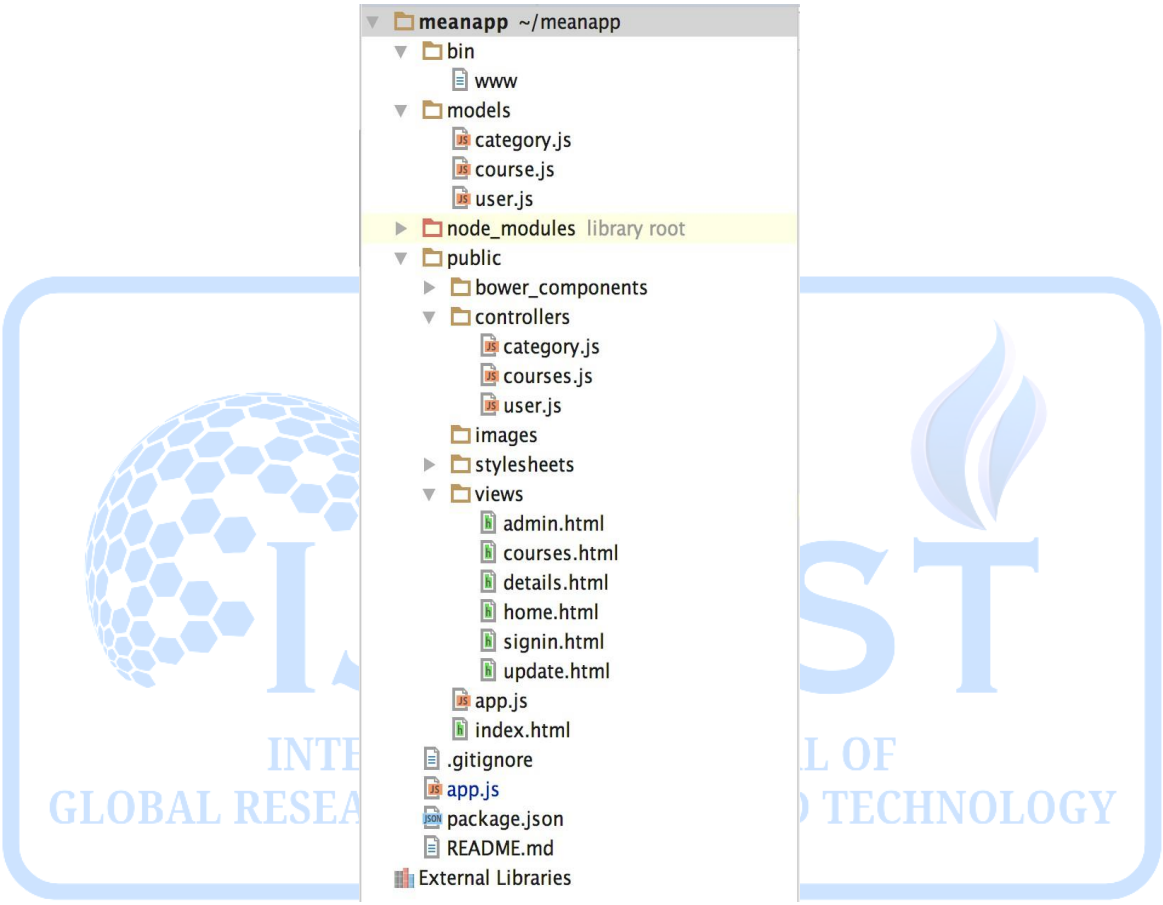


Figure 15: Application folder structure

```
1  {
2    "name": "meanapp",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "body-parser": "~1.15.1",
10     "cookie-parser": "~1.4.3",
11     "debug": "~2.2.0",
12     "express": "~4.13.4",
13     "jade": "~1.11.0",
14     "morgan": "~1.7.0",
15     "serve-favicon": "~2.3.0",
16     "mongoose": "*"
17   }
```

Figure 16: package.json file

3.6 REST APIs

The routing mechanism in Express helps to create REST APIs for the application. Generally, in a MERN application, REST APIs are built to feed the data to the client using Node.js, Express, and MongoDB.

```
app.get('/api/v1/courses', function(req, res){     Course.getCourses(function(err, courses){
if(err){            throw err;
        }
        res.json(courses);
    });
});  app.get('/api/v1/courses/:_id', function(req, res){
Course.getCourseById(req.params._id, function(err, courses){         if(err){
            throw err;
        }
        res.json(courses);
    });
});

app.post('/api/v1/courses', function(req, res){     var courses=req.body;
    Course.addCourses(courses,function(err, courses){         if(err){             throw err;
        }
        res.json(courses);
    });
});

app.put('/api/v1/courses/:_id', function(req, res){     var id=req.params._id;     var
courses=req.body;
    Course.updateCourses(id,courses,{},function(err, courses){         if(err){             throw err;
        }
        res.json(courses);
    });
});
```

```
app.delete('/api/v1/courses/:_id', function(req, res){     var id=req.params._id;
   Course.removeCourses(id,function(err, courses){         if(err){              throw err;
      }
      res.json(courses);
   }); });
```

Listing 2. REST APIs for course

Listing 2 illustrates the code snippet for creating REST APIs for the courses. The HTTP methods GET, POST, PUT, DELETE were used to perform the various operations at the endpoints. The URI follows the path root-api/api/v1/ representing the version one of the REST APIs. When a user provides the route (URI) such as http://localhost:3000/api/v1/courses , the getCourses() function is invoked which will return all the courses in JSON if there is no error.

Public directory (Client-side scripting)

The public directory contains all the files and scripts needed to provide the routes (URIs) and render HTML pages. The ngRoute module routes the Angular application to different pages without page reloads to make a Single Page Application. The routes are defined in the public/app.js file.

```
var app = angular.module('myApp',['ngRoute']);

app.config(function($routeProvider,$locationProvider){
    $routeProvider.when('/', {
        controller:'CoursesController',
        templateUrl: '/views/home.html'

    })
        .when('/courses', {
            controller:'CoursesController',
            templateUrl: 'views/courses.html'
        })
        .when('/courses/details/:id',{
            controller:'CoursesController',
            templateUrl: 'views/details.html'
        })
        .when('/courses/add',{
            controller:'CoursesController',
            templateUrl: 'views/admin.html'
        })
        .when('/courses/edit/:id',{
            controller:'CoursesController',
            templateUrl: 'views/update.html'
        })
```

Figure 17: Routes in ReactJs

Figure 17 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.

3.7 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.

```
myApp.controller('CoursesController', ['$scope', '$http', '$location', '$routeParams',
    function($scope, $http, $location, $routeParams){
    console.log('CoursesController loaded...');

    $scope.getCourses = function(){
        $http.get('/api/v1/courses').success(function(response){
            $scope.courses = response;
        });
    }
```

Figure 18: Course.js controller

Figure 18 illustrates a code snippet from the Course.js controller class. The code snippet illustrates the GET request method for all courses. The controller module uses the $http.get method to consume the REST service at the given URI (/api/v1/courses). It assigns the JSON returned back from the API to the $scope.courses, which sets a model object named courses. ReactJs then binds courses object to the application's DOM rendering it on the user's browser.

3.8 Results and Discussion

The Online Code Editor 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.

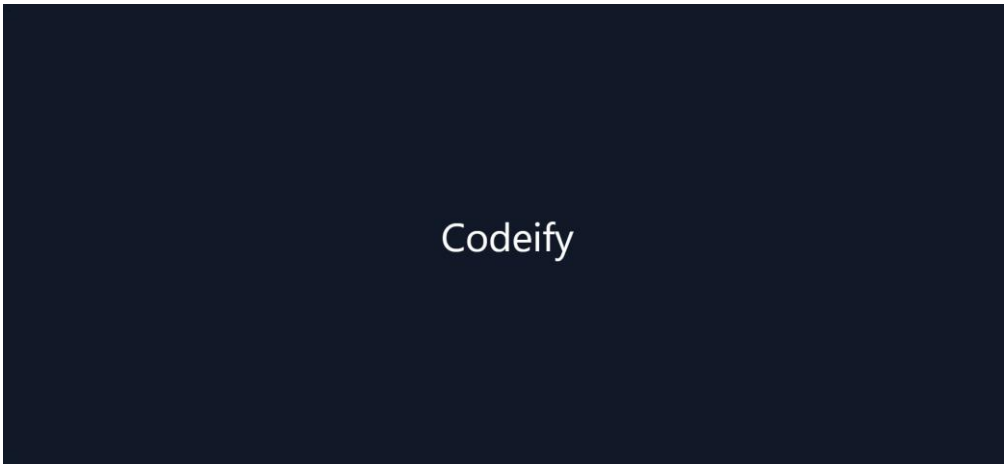


Figure 19: Code Editor Interface

Figure 19 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 20: Preview of Code Output

Figure 20 shows the real-time preview of the code. The application uses an iframe to display the output of the HTML, CSS, and JavaScript entered by the user. This live preview is updated instantly as users write or modify their code, without needing to reload the page.

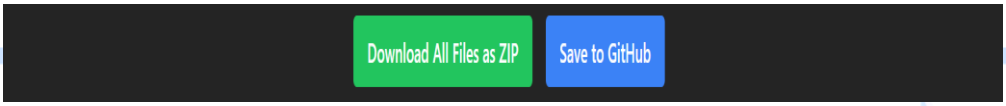


Figure 21: Download Capabilities

Figure 21 illustrates the download capabilities added to the application. Users can download the HTML, CSS, and JavaScript code they have written as individual files. Additionally, the Download All button allows users to download all files in a ZIP archive using JSZip and FileSaver.js.

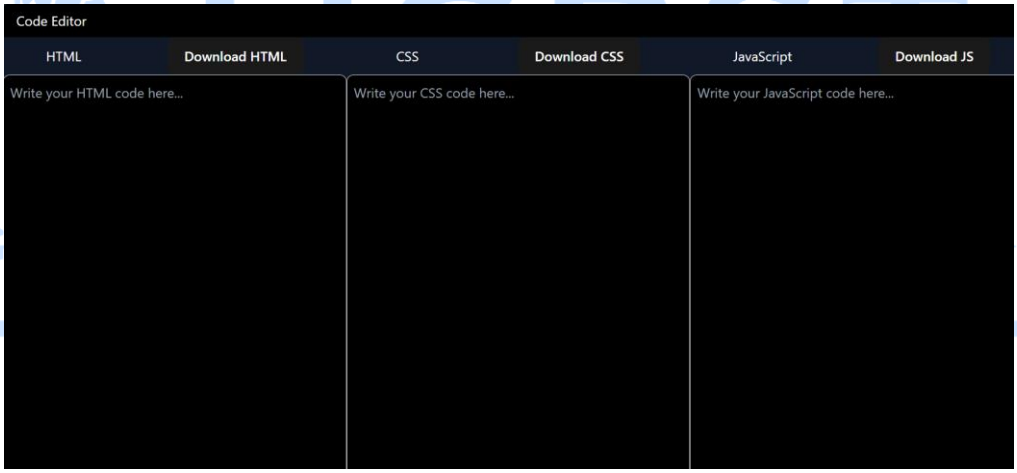


Figure 22: Responsive Design

Figure 22 demonstrates the responsive design of the Online Code Editor. 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.

4. CONCLUSION

The primary goal of this project was to study the feasibility and effectiveness of building a fully browser-based Online Code Editor 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.

A significant portion of the development process was spent researching the technologies used and applying best practices for frontend-only applications. The theoretical understanding of modern web technologies was combined with practical implementation, resulting in a functional prototype capable of meeting the needs of web developers. The code editor provides minimalistic yet essential features for writing and testing HTML, CSS, and JavaScript code in a user-friendly environment.

Based on the research and implementation, it can be concluded that a frontend-only architecture can effectively address the requirements of lightweight and portable development tools. By leveraging ReactJS for dynamic state management, Vite for optimized builds, and TailwindCSS for responsive design, the editor demonstrates the potential of these technologies in creating interactive and efficient web applications.

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 Online Code Editor 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 Codeify: "Online Code Editor" 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.

REFERENCES

[1] Hernandez A. Init.js: A Guide to the Why and How of Full-Stack JavaScript [online]. Toptal LLC developers' official website.
URL: https://www.toptal.com/javascript/guide-to-full-stack-javascript-initjs

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

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

[4] R. Ajmera, A. Kalwar, and C. S. Lamba, "A modern study on progressions & issues of web applications development in small firms," Int. J. of Scientific Research in Science and Technology, vol. 3, no. 8, Nov.–Dec. 2017.

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

[6] A. Kalwar and R. Ajmera, "ARQI: Model for developing web application," Int. J. on Technical and Physical Problems of Engineering (IJTPE), vol. 13, no. 47, pp. 7–13, Jun. 2021.

[7] Mozilla developers' official website [online].

URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

[8] Single Page Application Tracking [online]. Google developers' official website URL: https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications