Phuong Nguyen

Applying HTTP Protocol of TCP communication layer to build login system by modern web technology

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology Degree Programme

Bachelor’s Thesis

25 October 2022

Abstract

Author:	Phuong Nguyen
Title:	Building Login System by Modern Web Technology
Number of Pages:	53 pages + 3 appendices
Date:	25 October 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology of the degree programme
Professional Major:	Name of the professional major
Supervisors:	Janne Salonen, Title (Head Degree)

This thesis introduces an overview of modern technology for website, and web application to dive in deeper the knowledge of TCP network communication through application layer, is called HTTP mechanism. A login system for web application is presented as well. In this technology age, Cloud Computing is blooming, making serverless database become more and more popular. MongoDB is one of the most outstanding NoSQL databases of storing the persistent data. Node.js is used for environment runtime from the server side, is combined with Express.js as a web server builder as well as React acts as a simulated sever from client. It creates a perfect full stack for developing web application in a fitted way with previous JavaScript on client-side rendering, in reason to guarantee the web application can become smoother and higher effective in NodeJS ecosystem.

MVC models is applied to keep the architecture of source code easily follow. The MVC framework is combined with React to separate the Model, View, Controller in a clean structure. As a result, MVC approach can separate concern of a software product into three main logic tier architecture for scalability, availability, reusability, confidentiality with strictly principles is applied.

Cloud Computing conveniently bring better performance and low cost-budget for every firm from small to huge size. It is easily scalable, available without the physical harms. In theory, the data is safe and convenient management by the cloud hosting company. But API from the 3rd party, confidential data could be arisen security issues for the customers who care the privacy. OAuth 2.0 protocol represents a delegation procedure of using token to avoid universal password based on security issues, totally perfect for 3rd party APIs.

To sum up, this thesis is an analysis of instruction of building a web application by JavaScript technology with its framework combination, and network package examination of login system as an exemplar for TCP communication through HTTP layer.

Keywords: login, modern web technology, OAuth 2.0, client-side rendering, server-side programming language, JavaScript, MERN stack.

Tiivistelmä

Tekijä:	Phuong Nguyen
Otsikko:	Insinöörityön otsikko
Sivumäärä:	53 sivua + 3 liitettä
Aika:	25.10.2022

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Login systeemi
Ammatillinen pääaine:	Phuong Nguyen
Ohjaajat:	Janne Salonen

Tiivistelmän tekstiosuus kirjoitetaan niin, että se mahtuu sivulla käytössä olevaan tilaan. Tekstiosuudessa käytetään Leipäteksti ilman välistystä -tyyliä.

Avainsanat:	MERN, TCP, HTTP

Contents

List of Abbreviations

MERN:	Mongodb, Express, React, Node.js.
MVC:	Model View Controller Architecture.
API:	Application Programming Interface.
TCP/IP:	Transmission Control Protocol/Internet Protocol
TCP:	Transmission Control Protocol.
HTTP:	Hypertext Transfer Protocol.
URL:	Uniform Resource Location
CORS:	Cross-Origin Resource Sharing
DOM:	Document Object Model.
W3C:	World Wide Web Consortium
AJAX:	Asynchronous JavaScript and XML
CI:	continuous integration
CD:	continuous delivery
ACID:	Atomicity, Consistency, Isolation, Durability
BASE:	Basic Availability, Soft-state, Eventual consistency
DBMS:	Database management system. Software for maintaining, querying, and updating data and metadata in a database.
RAM:	Random Access Memory
ODM:	Object-Document Mapping
JSON:	JavaScript Object Notation
BSON:	Binary JSON
I/O:	Input/Output
UI:	User Interface
HTML:	Hypertext Markup Language
XML:	Extensible Markup Language
JWT:	JSON Web Token
WebAuthn:	Web Authentication API
DSL:	Domain Specific Language
GUI:	Graphical User Interface
WOFF:	Web Open Font Format
CDN:	Content Delivery Network

List of Figures

List of Tables

1 Introduction

In Cloud Computing age, the development of web application is boosted by different types of database and technology stacks. The organization from the small size to huge size can select an advantage, beneficial approach to build a web application easily based on specific needs and purpose. Client-side rendering becomes powerful when JavaScript develops technology for web development, by enormous and diverse frameworks.

The purpose of this thesis is focus on the grasp of how to build a website, or web application by understand the necessities of web technologies via a real-life login system. A login module is use widely, conveniently, frequently in any web application. This idea is totally convenient for creating a reusable login module by applying MERN (MongoDB, Express.js, React.js, Node.js) stack. Especially, MVC (Model, View, Controller) concept is combined to keep the structure of node modules more convenient manageably. APIs (Application Programming Interface) from the 3rd parties have increased tremendously by giant technology companies such as Google, Microsoft, Facebook, Twitter, et cetera. It provides a favourable way to integrate interesting, time-saving functionality and data from other sources to a specific source with various features.

Modern web technology is growing faster and faster everyday thanks to open source for contributors all over the world. Many frameworks of JavaScript are created, is overwhelmed for those who cannot have a clear vision of the usage of JavaScript stack. Therefore, this thesis is aiming to explain how to be integrated mix and match technology with frameworks of JavaScript, is called MERN stacks.

2 Technology Concepts

Firstly, a TCP/IP (Transmission Control Protocol/Internet Protocol) model can be useful to show how a network system works on a web application through

HTTP (Hypertext Transfer Protocol) protocol of communication layer. Secondly, a MERN stacks represents layers from identify stage to output process stage of the outcome. Simply put, MERN is an engaged technology in modern web era of connecting a NoSQL database, making routes of calling APIs data from the user interface modules to the back-end server modules.

2.1 TCP Communication

The TCP (Transmission Control Protocol) is a communication standard to start the process between initiator and receiver aka client and server. This protocol helps master devices connect with end devices to transfer information package, or data message.

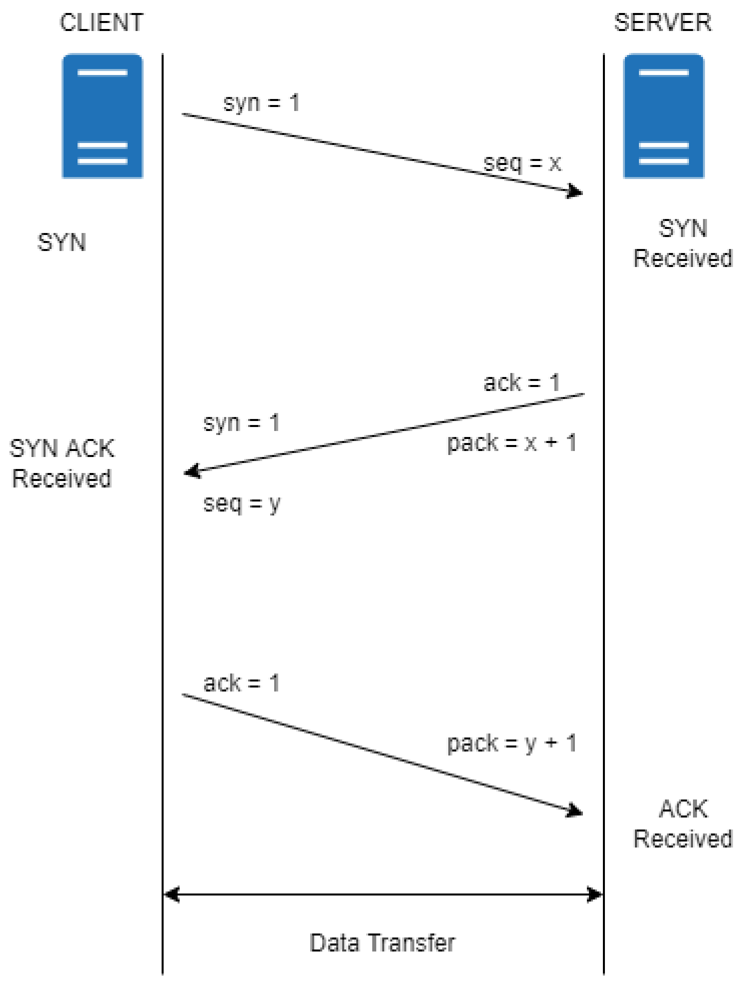


Figure 1. A conceptual model of TCP communication.

In Figure 1, three-ways handshakes [1] within a connection time among terminals to ensure the transferrable data.

To fathom the concept, TCP socket is considered as an operation of TCP/IP application. The sockets from the client and server enables remote ports to receive full data package. At the first stage, an establish connection from active client to send SYN with seq = x to passive server. After SYN received seq = x, the data transfer stage begins at server sends SYN ACK back to client with SYN = y, ACK = x + 1 as well as starting at seq = y. At the close connection stage, server receive a message of ACK = y + 1 means acknowledgement full data package and stop the process with closed session.

2.2 HTTP Mechanism

As good definition of software architecture from Roy Fielding [2] about REST architecture [3], stateless is a clarity, effective way of communicating.

The HTTP [4] works as client-server mechanism [5] of transferring data by request and response process through pipeline of applications' layers. To deep dive the knowledge, we must understand clearly REST and RESTful API aka REST API [6].

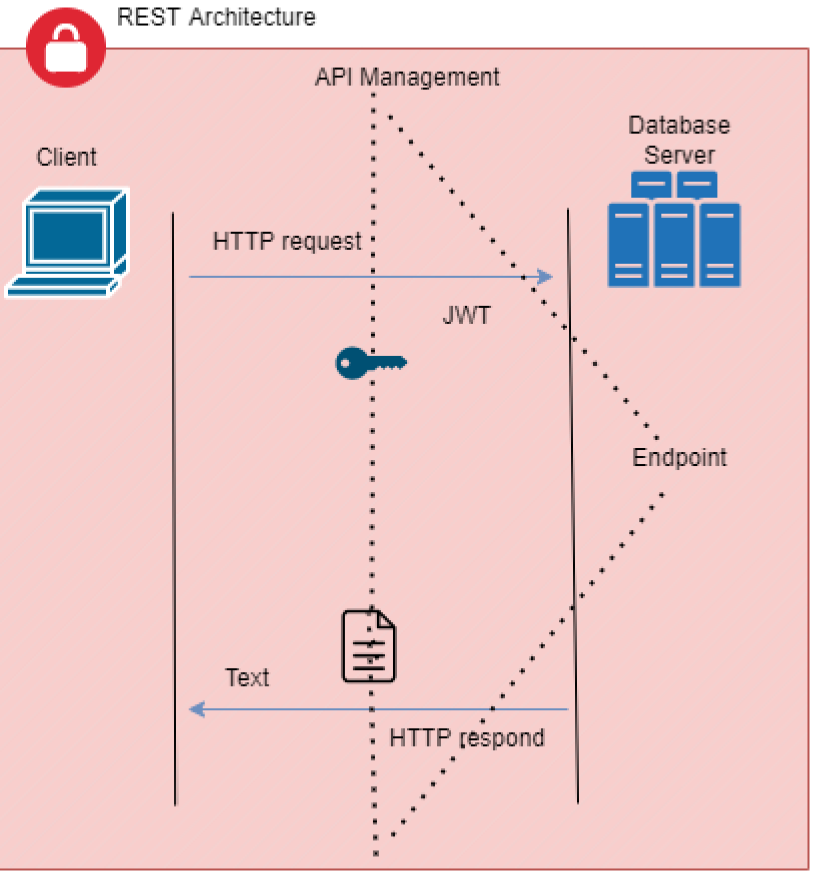


Figure 2. REST architecture and its application.

In figure 2, API is under control of REST architecture which enables transferred data from server to client by HTTP mechanism. API can work as a key to access the right resource which calls API endpoint, then send back the data package in form of JSON files. Therefore, user can access data via APIs communication. Thanks for APIs, data also can perform, interact, and integrated information from database and API management system [7].

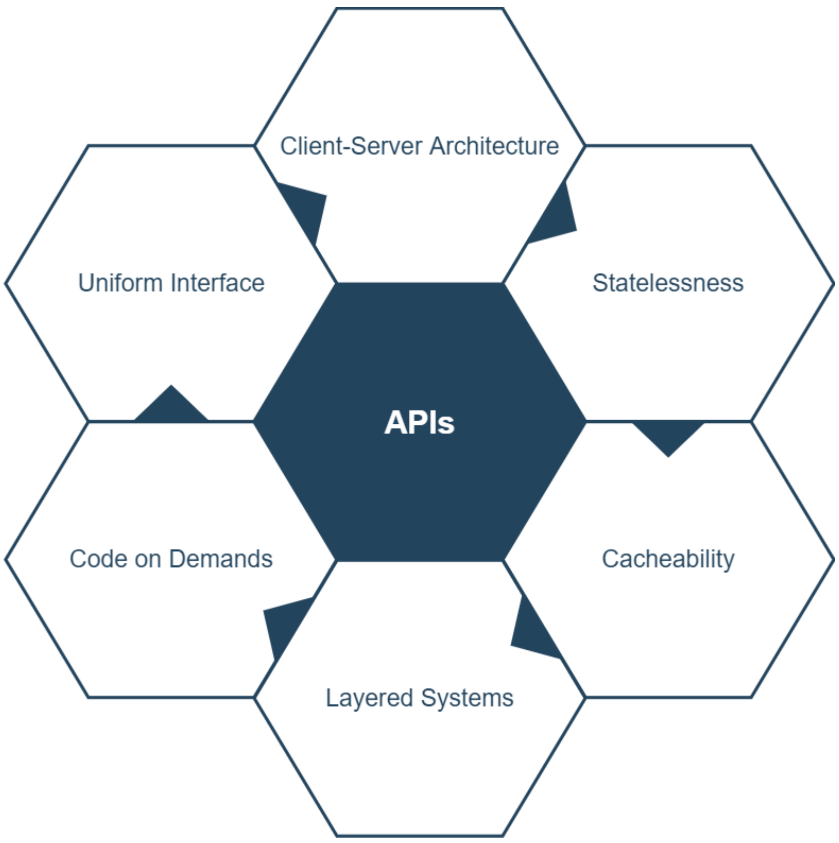


Figure 3. Six Constraints of RESTful system.

Figure 3 represents 6 constraints of REST architecture style [8]. And APIs is applied RESTful system to generate a web service between client and server. This architecture almost innovates an effective way of modern web by following the loosely couple rule of software architecture. The terminals always be the same, but the interfaces of terminals can change the past to create new future by separating parts and apply rules of creativity: break, bend, blend. It is the vital part of understanding software architecture styles and principles.

APIs is an application of REST architecture constraint. Therefore, stateless request and response is applied to generate APIs endpoint [9].

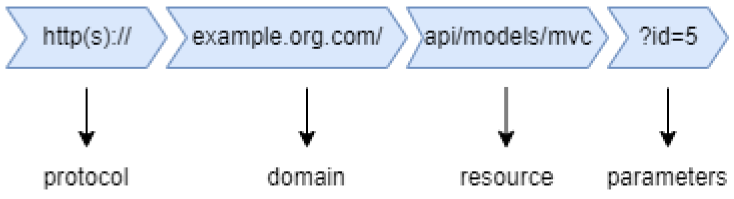


Figure 4. Uniform Resource Locator parts.

To form a HTTP request to a server, client need URL (Uniform Resource Location), method, list of headers, body.

Below are tables of HTTP methods, HTTP status code to form a URL for APIs endpoint after user enables a request.

HTTP methods	Description	Method Properties
HEAD	To send header data-package.	Safe, idempotent, cacheable.
TRACE	To check-up connection of server.	Safe, idempotent.
PUT	To upload files into server.	Idempotent.
PATCH	To modify a part of resource	No properties.
DELETE	To eliminate files from server.	Idempotent.
OPTION	To describe selection of method support and return suitable error.	Safe, idempotent.
CONNECT	To build a tunnel to the server.	No properties.
POST	To create or update resource.	Only works for update information.

Table 1. HTTP methods.

Status Code Number	HTTP response status	Description
100-199	Information response.	Indication from server.
200-299	Successful responses.	Request succeeded.

300-399	Redirection messages.	URI changed.
400-499	Client error responses.	Problems happen from client connection.
500-599	Server error responses.	Problems happen from server connection.

Table 2. HTTP status code.

HTTP mechanism contains a set of HTTP request methods which are defined as cacheable, idempotent, safe, and HTTP response status code.

To upgrade the security level [10], most CORS (cross-origin resource sharing) becomes a crucial browser security feature to consider whether enable CORS support. Because most of non-simple cross-origin HTTP request need permission to access the resource from the different domains.

Simply put, starting with real-usage concept of fetching [11] can demonstrate CORS [12] and HTTP origin header [13] vividly. Fetch API follows the rule of same origin [14], meaning that different host, protocol, port brings failure access of receiving data from different resources. Therefore, CORS is used to allow cross-origin access. CORS could play a role in HTTP process which qualify other outside hosts so that browser can have power of permission to load web content without blocking policy of same origin.

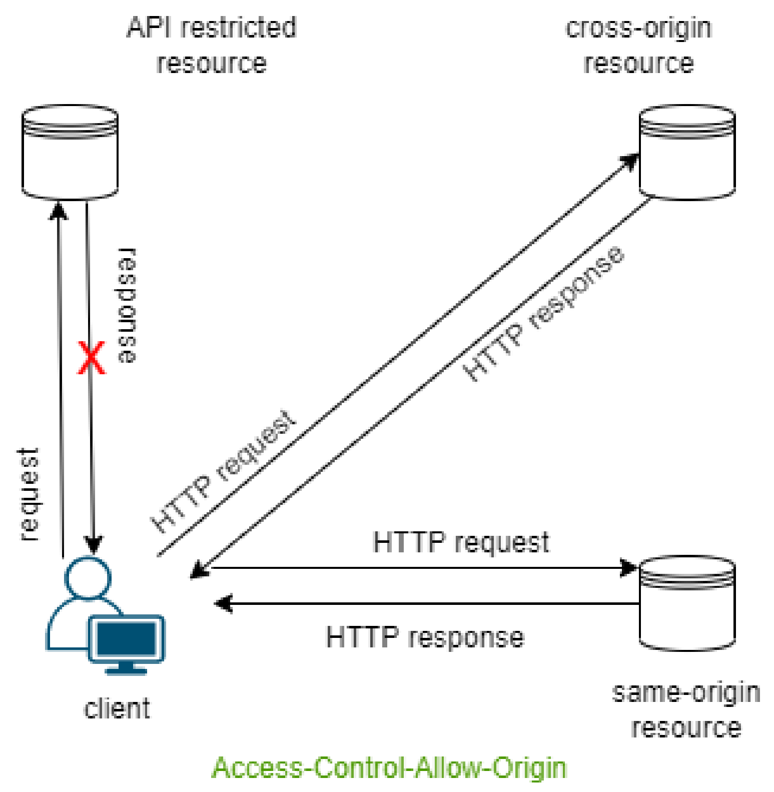


Figure 5. CORS support outside resource by HTTP header request.

The process of how to detect security issues of CORS configuration base on HTTP origin header. The figure 5 shows how a CORS work securely effectively with header data values from HTTP header request. Understanding this concept, we can work easily with the resources from the origin resource as well as cross-origin resource sharing. Besides, the server guardian who has responsibility of security can make a good configuration to protect API restricted resources from client-side who can make a harmful access.

2.3 JavaScript Innovation

JavaScript is invented [15] as a concept of a lightweight, simple, easy-to-learn, fast, powerful, popular, widely used [16] programming language which can work productively from small to bigger web application by merging features of functional programming and object-oriented programming. Moreover, the support from most of browsers bring a big plus advantage for JavaScript. Hence, when

Nodejs was appeared, was changed the capability of the JavaScript technology by expanding the ecosystem of JavaScript.

On very first day, web application is very simple start with HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JavaScript. DOM (Document Object Oriented) has established by W3C (World Wide Web Consortium) since 1998 [17], has changed the way of industrial web development. DOM provides an API of permission to access and update content, structure, style of web documents. Due to the performance, DOM becomes hard to manipulate data by cross-browser, web application era comes up with jQuery, AJAX (Asynchronous JavaScript and XML). AJAX is a combination of XMLHttpRequest object and DOM to render data on client-side [18].

Disadvantage of AJAX is implementation through different browsers. Therefore, jQuery is a useful library of JavaScript can be a helpful support for AJAX. As a result, JavaScript is more and more improvement, to become a beautiful programming language by many supporting libraries and frameworks. Nowadays, modern web technologies can work compatible with most of cross-platform.

JavaScript is still enhancing the performance by technology stack driven. MERN stack [19] is one of the best selections of mix and match of modern web technology. Moreover, machine learning is a wonderful concept [20] of this age. Node.js has an API which called child process module [21], can streaming data of python process with Node.js and displaying the output on the interface of end user [22]. Digging deeper knowledge of MERN stack combination, the mingle of infrastructure, and cloud computing technology is mind-blowing. Azure pipeline [23] is an operation when CI (continuous integration) and CD (continuous delivery) is mixed implementation. More amusement, Azure machine learning [24] is one of the most astonishing tools, which can predictive analysis [25], in the market.

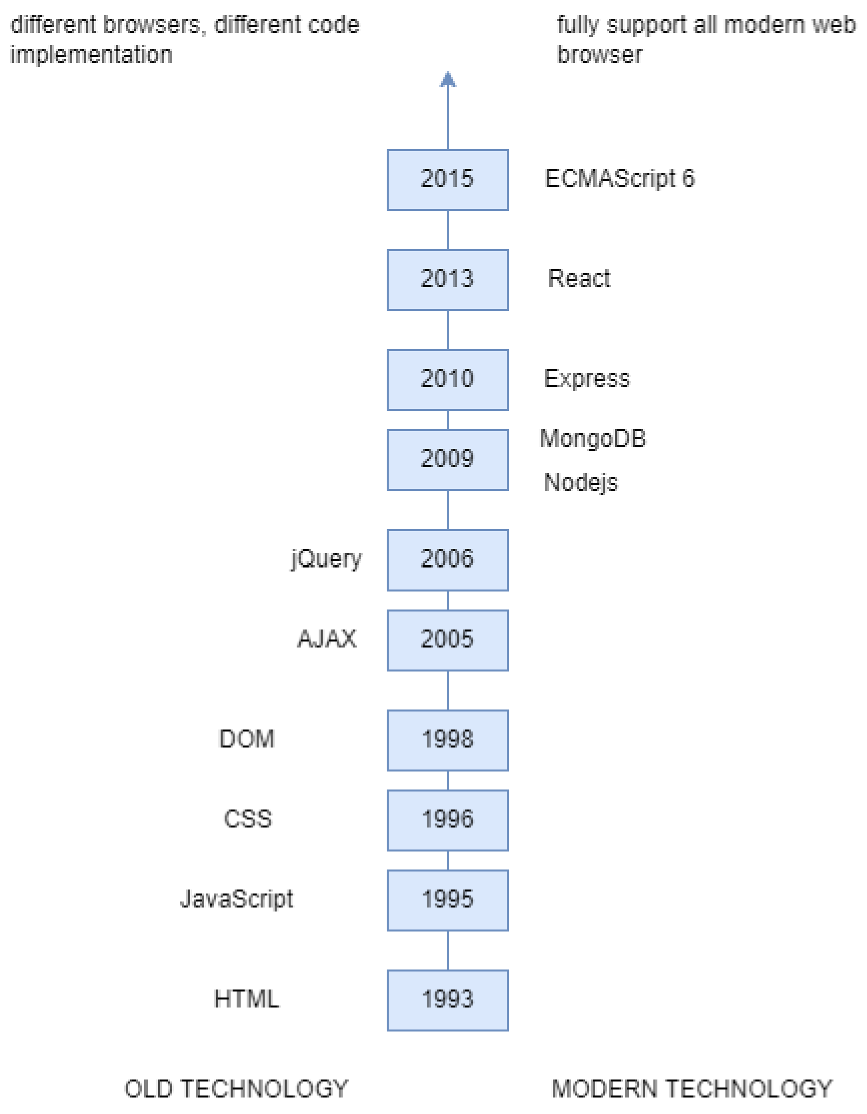


Figure 6. The history from old to modern of web technology.

Figure 6 demonstrate a short introduction of the first day of web application. It starts with HTML as a markup language [26] for website works with browser. It could be boring because of plain text, simple images. CSS appears as a good solution to sharpen, enhance the style [27] of web content [28] includes paragraphs, video, image. JavaScript brings dynamic effects [29] to a website. Since DOM has created, JavaScript is still a painful [30] programming language because the different default setting of each browser. Until 2005, Ajax appeared as a new style [31] for web application. From that time, asynchronous [32] management and event handling are more popular with JavaScript language. With the application of Ajax engine, scripts and style sheet just call once as well as asynchronous request can be received merely necessity content. This advantage is useful for time performance and server bandwidth. It sounds helpful on performance perspective. On contradicting viewpoint, security issues [33] within same origin policy, bring hardship for websites which work with Ajax. So far, the perfection mingle between Ajax and jQuery has been used to load data from the server as the real-time strategy of coding.

While the rapidly growing of client-rendering-framework such as React, Angular, Svelte, Vue, etc; jQuery is still living with Ajax technology from client-side application. That’s why favourable code implementation doesn’t mean it is the best way to keep. To provide a clear vision of the reason, we can think frameworks which is a craving of integrating functional compatibilities between node modules. NodeJS bring packages together in a union application. The problem happens at the compatibility levels, if not, the other problem is the impact. In consequence, logic business must separate concerns in each tier architecture, become modular design, to handle data in a pipeline which calls it a name of microservices. As a hope of popular programming languages, JavaScript is a non-stop improvement language through multi-purpose frameworks [34] invented.

3 MERN Stack

The diversity of different technologies of JavaScript brings the dream of a full stack development can become true. It is an impeccable practice, but it is quite complicated to apply with novice. Hence, MERN stack is a good technology shelf for implementation from front-end to back-end.

3.1 MongoDB

Distributed management is a good design [35] concept. Especially, data management due to the scalability, flexibility, availability, and durability of a workload. MongoDB is a practical application of distributed data management. Compare with traditional relational database, MongoDB is called NoSQL [36] database management system.

Dig deeper of the approach of DBMS (database management system), is the concepts of a database which operates with the ACID (Atomicity, Consistency, Isolation, Durability) properties, or with the BASE (Basic Availability, Soft-state, Eventual consistency) properties.

ACID properties represent vulnerable transaction [37], which relates to stable and consistent data. BASE data model known as aggregate stores [38], which is real-time analytic, unstructured data with saving cost of Random Access Memory (RAM) [39] due to the process of dynamically allocates and de-allocates RAM.

Based on different purposes, DBMS can be flexible usage of selection and matching. MongoDB is a popular tool [40] of document database of NoSQL database, is applied BASE properties.

To provide a direct solution of reusable code of validation, casting, business logic [41] of MongoDB, Mongoose is an object modelling in NodeJS packages, is created to work with data in asynchronous environment [42] by schema-based solution. A NoSQL database like MongoDB can apply ODM (object document mapper) as a tool to manipulate JavaScript objects. ODM use a JSON (JavaScript Object Notation), BSON (Binary JSON) API to convert object notation to document notation. Therefore, Mongoose is a good tool of ODM, has functions to map [43] between object models and document databases, straight forward method to perform operations with objects, and translate information into the proper data query, schema types.

3.2 Express.js

ExpressJS [44] is one of a module of Node.js system, contains a huge library of useful middleware, ranging from template engine to high performance-based of HTTP mechanism.

Express can interact [45] with MongoDB through Mongoose schema. To work with Mongoose schema, creating a model is a necessary task which is helpful for generate the database collection. After model classes are defined, queries of create, read, update, delete records can work with saved or retrieved data.

Moreover, there are more popular built-in middleware function [46] in Express library such as express.json, express.Router, express.urlencoded, etc. Like JavaScript, Express.js also use the diverse objects with different methods combines with various properties to complete the process of handling events.

For example, the parameters [47] of callback function in Express application are request object and response object. Request object blends with body, app, cookies, path, query, xhr properties to performance specific task related in HTTP request. And request object also provides many methods to execute operation on the request body and header of HTTP.

Routes and controllers are main parts of Express middleware.

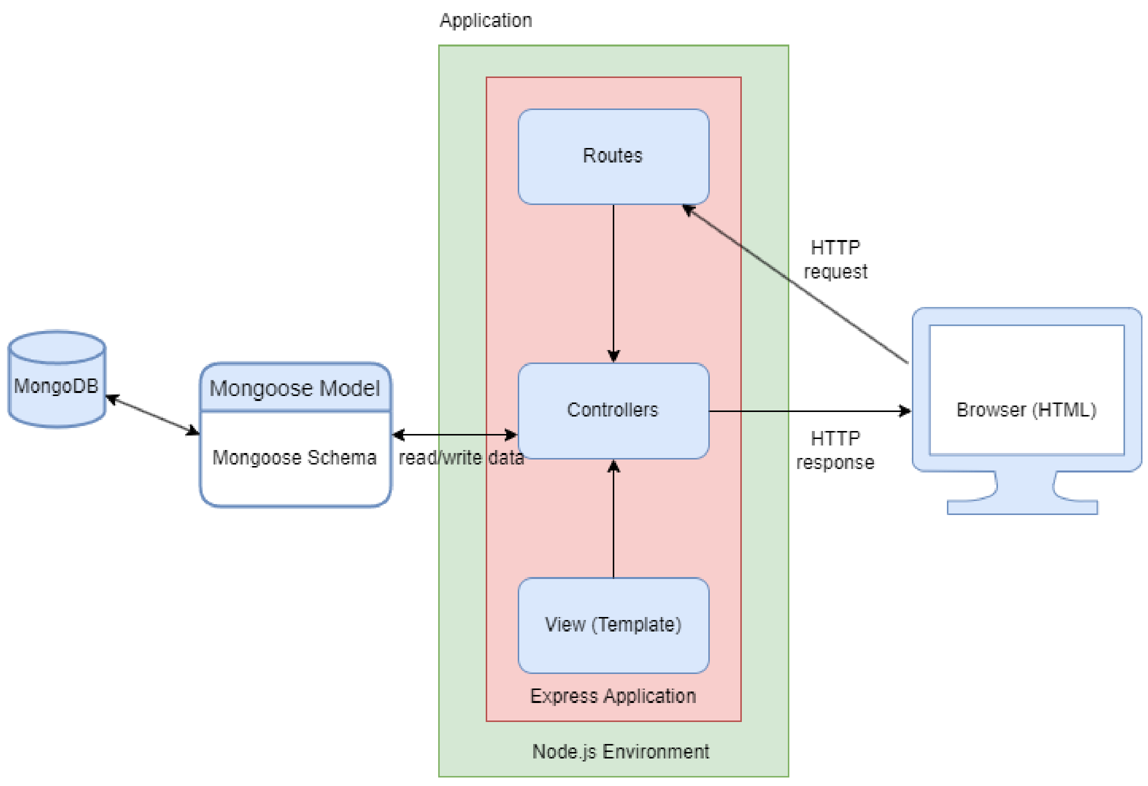


Figure 7. The diagram shows the handlers of HTTP request and response.

3.3 React.js

React is a modern technology of web application interaction. Although React is client-side framework, it can work as a pseudo-server-side. React uses state, props to render components when data is updated.

Thanks to the business logic of component-based [48], DOM of web application aka API for HTML (Hyper Text Markup Language) or XML (Extensible Markup Language) documents with a logical tree is replaced by a virtual DOM [49]. React can enclose a private component in a capsule which own states to create complex UIs (User Interfaces) by re-render technique [50].

Short introduction, React can load from a <script> tag to make the scope globally, and import React from ‘react’ to start using the package in Node.js package system. Below is the list of React top-level API [51] to render React applications.

Name	Usage	Description
Components	React.Component React.PureComponent	Modular design.
Elements	React.createElement()	Without JSX.
Fragments	React.Fragment	Without a wrapper.
Refs	React.createRef React.forwardRef	Access DOM nodes.
Suspense	React.lazy React.Suspense	Dynamic loading.
Transitions	React.startTransition React.useTransition	To mark up update as changing shift.
Hooks	Basic hooks, Additional hooks, Library hooks.	Update state and React features without create class.

Table 3. List of React top-level API.

3.4 Node.js

NodeJS is an open source of cross platform of JavaScript run time environment, runs on V8 engine. The purpose of NodeJS is a lightweight of non-blocking I/O (Input/Output) model which focus on event driven of I/O web application. Hence, outstanding features of NodeJS is fast execution, asynchronous, no buffering, multitask approach based on supported library of single thread concurrency with multiple components.

Node.js is a high effective performance tools of real time application for scalability, faster, responsive. Due to the caching in Node.js application, the performance boost to an impressive speed.

According to the experiment [52], from developer community of dev.to, node memory-cache [53] helps request reduce time performance. With the explosion of Cloud computing in modern era, Nodejs is a good mix and match high-tech in implementation of web application as well as fetching data [54].

3.5 Security Issues

Focus on the data sending and receiving, HTML forms on a web application is used to accumulate privacy and non-privacy information from users. Therefore, many techniques of hacking can be exploited in this process with various purposes. Hackers can use data package from input user to inject malicious codes which is harmful for server, or attackers can utilize the secret data of users for damage reasons. The input elements [55] is varying types of collected data, with various attributes [56]. The art of working with form includes validation, sanitization input data, event handlers, respond the state results.

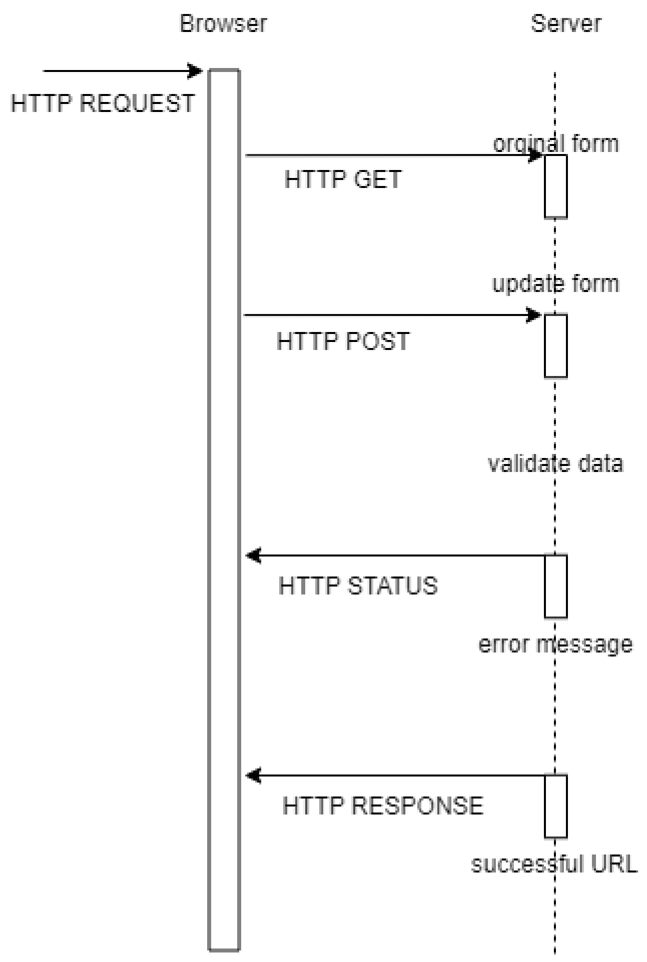


Figure 8. Sequential flow of processing data with form.

Figure 8 shows how data process when a form is submitted. Normally, GET and POST methods of HTTP create routes for validation, processing data in form. Therefore, express's middleware such as express-validator [57] module to provide many functions with parameters related to the valid and sanity of data before the data is sending to server.

To dig deeper, the security solution for most of safety issues are authentication and authorization.

3.5.1 Authentication

The concept of authentication for identity is start with the web authentication API [58] to enhance the vulnerabilities.

To keep the data is safe from the attack, authentication must strong enough for users to avoid becoming targets of attackers. There are various styles develop login and logout module. Most of the approaches is the concept of encrypting user's credentials [59] or using hash function to elude stored plain text passwords.

For instance, JSON Web Token [60] aka JWT applied cryptographic algorithm to guard package content and divided into 3 parts with a period, in a syntax: header.payload.signature.

Another example is Web Authentication API. There are 2 methods create () and get () to allow users sign up or sign in on web application with the public key [61] option. Usually, most of passwords is not strong enough, although with the validator input, malicious data, and sensitive data of organization such as bank, social media, email, mobile applications still can be harmful to any user who have to use the sceptical network. Therefore, Web Authentication API aka WebAuthn [62] is a new way for alternating a credential confirmation instead of weak information as encrypted password to avoid potential harmful on the internet.

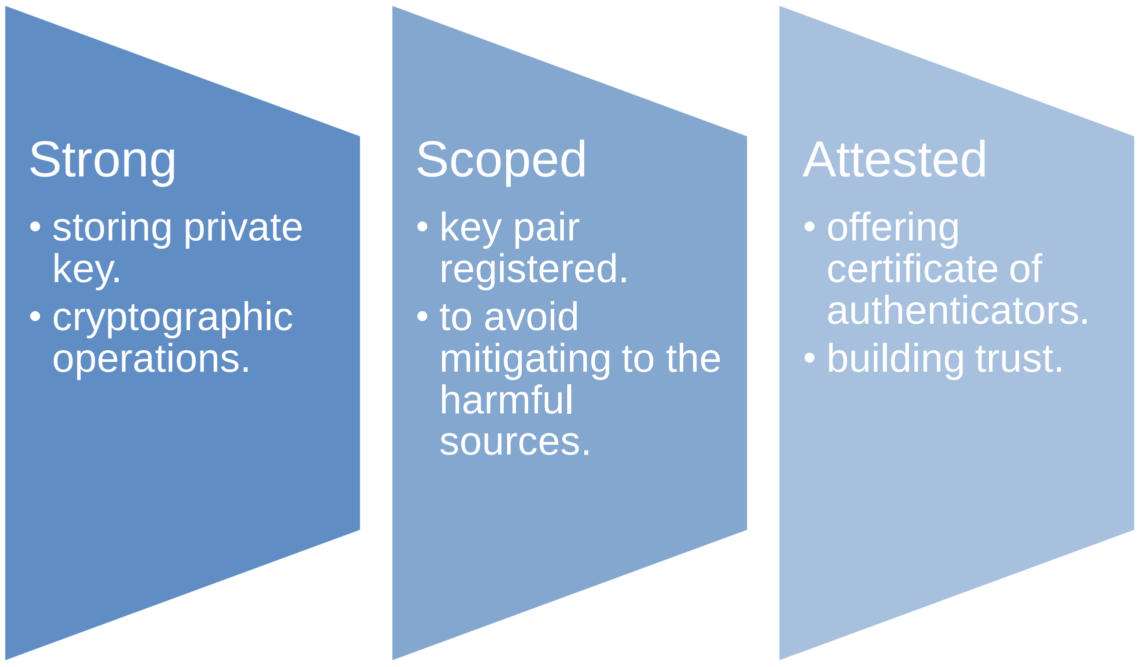

Strong
• storing private key.
• cryptographic operations.
Scoped
• key pair registered.
• to avoid mitigating to the harmful sources.
Attested
• offering certificate of authenticators.
• building trust.

Figure 9. Three main properties of Web authentication.

Based on the technique of asymmetric cryptography, there are a public and a private key for encrypting and decrypting process of messages, to secure sensitive data from unauthorized access. Only the public key appears on the server, that’s why hackers cannot decrypt the data with the private key is on the user’s gadget. It is called multi-factors authentication to limit guessing password of the brute force attack [63] which can become DDoS attack [64].

3.5.2 Authorization

Authorization is a closely associated process after the authenticated action. After the verification of identity of an account, granting permission of token access can restrict which data can be made operation by which users. To explore how authorization implement and manage the data access, role-based access control [65] is a common indispensable approach for most API developers who choose MongoDB as a database stack. In other words, when a user makes a request, Authentication business area make a confirm of identity for Authorization server dispatch a token with the scope [66], to guide the actions’ accounts which permissions’ levels of users’ roles have privilege.

Besides, there is another popular approach of authorization, is called attribute-based access control [67], also used for NoSQL database such as MongoDB, but not supported by MongoDB.

4 Building Login System

As a full stack project, every single module should be well-organized. It is a useful approach to implement a fully web application from infrastructure, database, front-end to back-end interfaces, rely on separation of concern principle [68]. Accordingly, MVC architectural pattern [69] provide a good technique for arranging similar codes by functional based.

In a short introduction of web technology, back to the simplest approach of login form in the past, plain JavaScript (aka vanilla JavaScript) combines with ajax technique to read and write data through HTTP mechanism.

In the Node.js ecosystem, template engine is a wide range of handling fast performance in both client side and server side to manage the static files and dynamic content of views and layouts. Because there is no safe user input, handling forms and validating data is a must-do approach. Middleware [70] works as a pipeline to operate the streaming of every part of the application into every single edge of the flow as the most suitable design purpose.

An anatomy of MERN stacks' operation is similar of building a sandcastle. MERN applied 3-tier architecture, is named MVC model. Firstly, a strong base is constructed by dependencies of node modules in package.json file. Next, a stack gathers related components together. In that case, Express plays a role of pipeline for transferring data between middleware [71], to handle data between the client-side and server-side.

For that reason, MERN stacks contains hundreds of boilerplates [72] to implement web application, groove on design patterns to keep implemented codes in neat organization, easily maintainable application. Subsequently, there are real lines of codes is applied to replace the existed code in boilerplate. It means occurred code must be removed by suitable codes to fit the expected output.

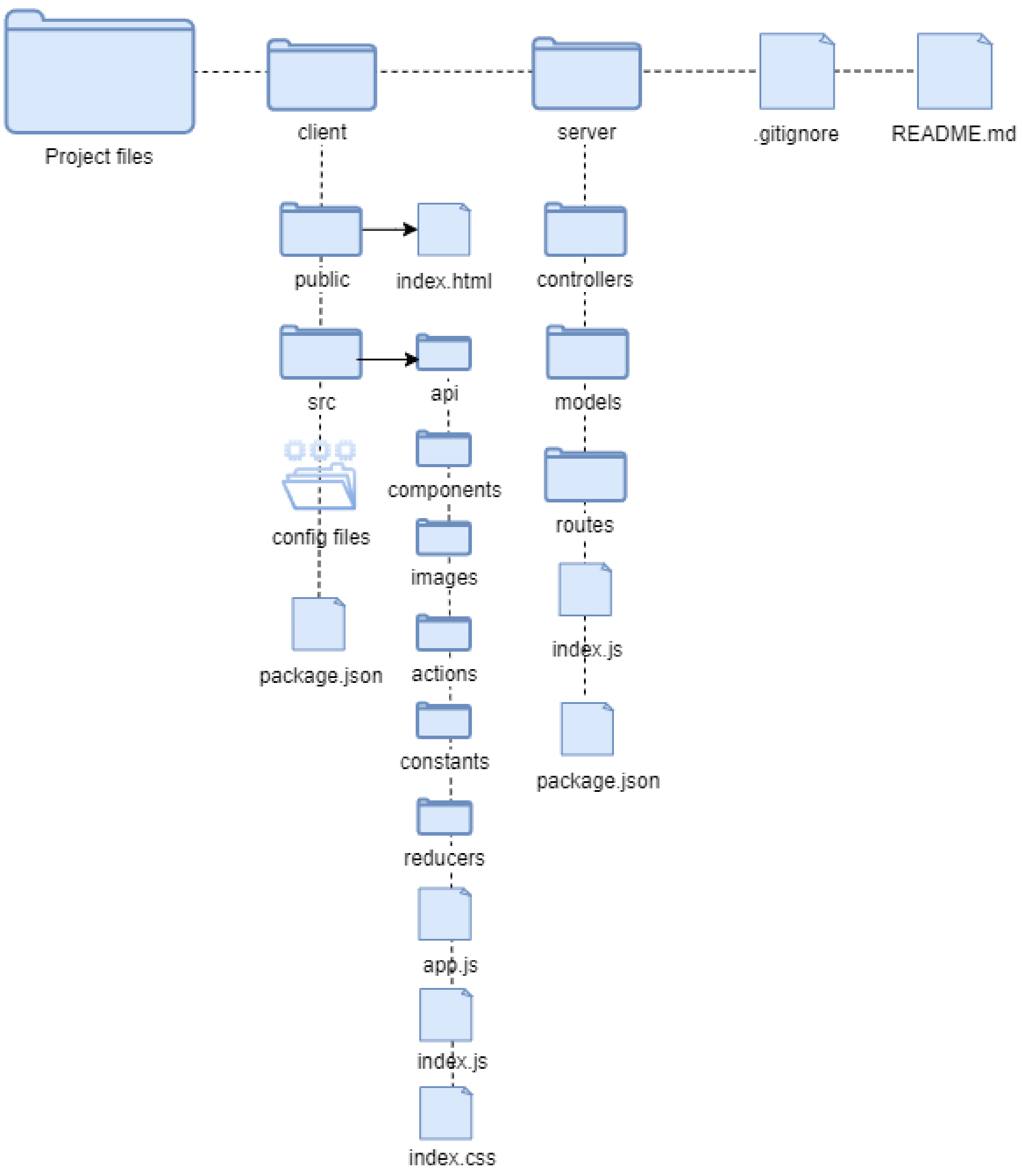


Figure 10. Diagram displays a folder structure of MVC architecture style on Github.

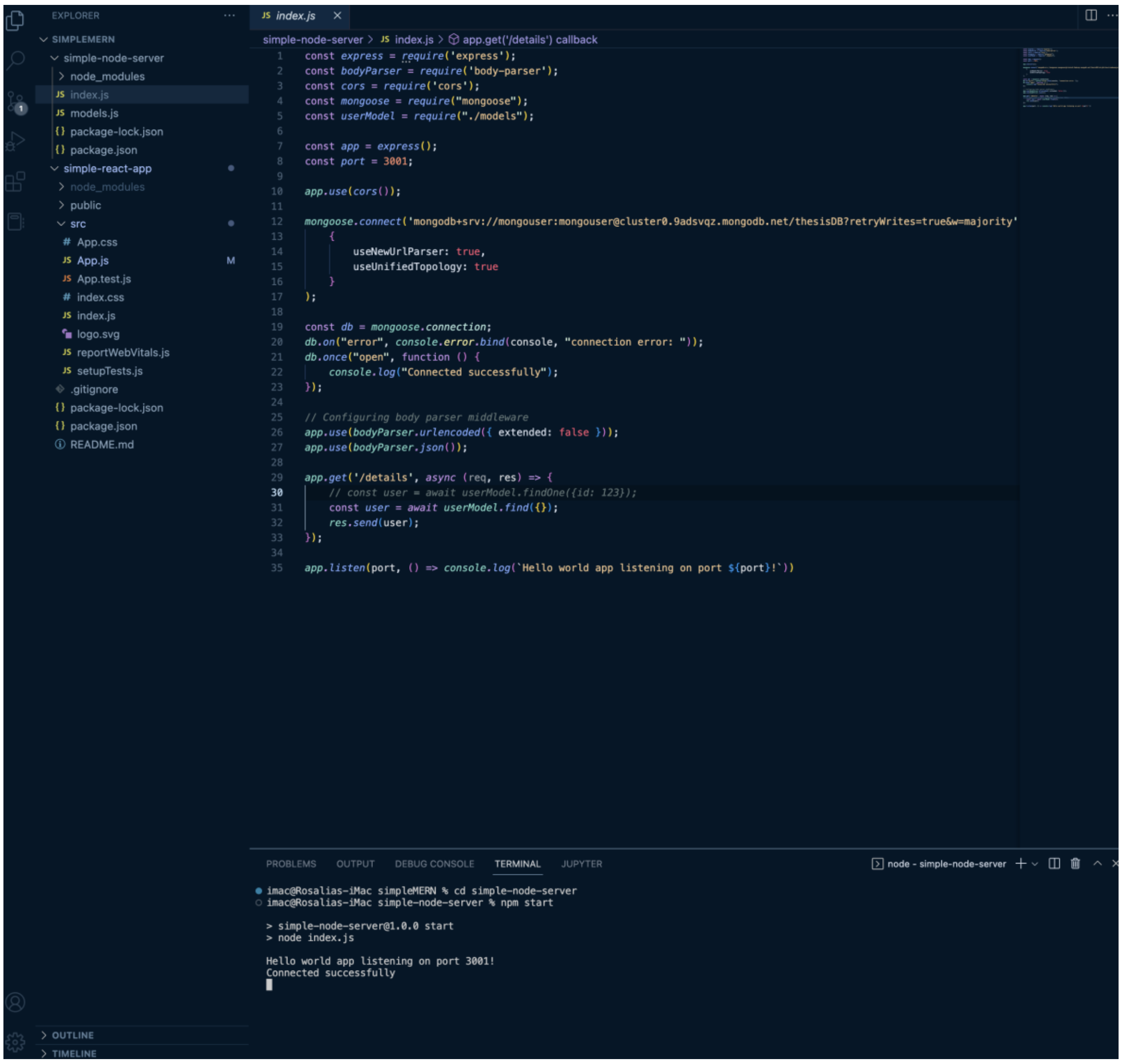
To push code on Git, developers need to have an account on Github. MVC architecture style on Github had better followed by client and server folder. README.md is a file for documenting the introduction of a software product. Moreover, .gitignore file helps eliminate unnecessary files such as node modules, dotenv module, which developers want to hide in production launch process.

While server folder is responsible for controller, model; client folder is designed for view.

Because MVC architecture style chases up the approach of separation of concern style for scalability, reusability code. Model, view, controller is implemented in real-life is not easy to catch up. For instance, client folder contains what browser renders such as package.json, configuration files such as babelrc.js, eslintrc.js, public folder with index.html file, src folder includes self-sufficient files, or folders such as specific name of module, reducers, actions, api, images, app.js, index.js, index.css, et cetera. And server folder consists of package.json, index.js, controllers folder, models folder, routes folder.

In MVC structure, view is involved in rendering layout, server acts the part of controlling the routes based on database model. In Github, a distinct of real-life application, MVC pattern is divided into client and server for code management. On that account, client should contain components of view layers such as layout, template. Furthermore, the view from client folder can be optimized by other linked components, named reducers, constants, action, api, images for independent tasks. Similarly, the controller from server folder defines properly routes from the model of database's part. That being the case of making connection for database and web server, then focus on layers of rendering view of UIs, establishing favourable data from request process.

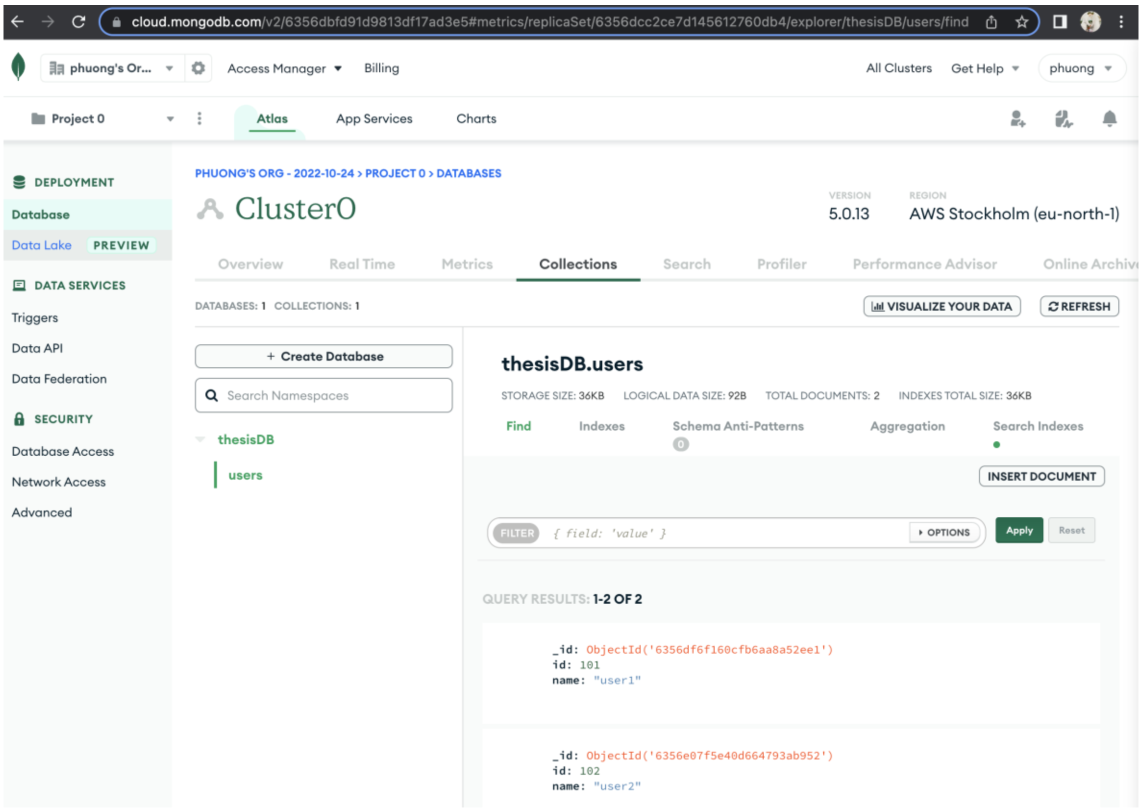
In this below example, a simple MERN stack app is deployment to practice the theory of client and server in MVC architecture. The tutorial [73] starts with 2 folders: one is for simple-node-server, the other one is for simple-react-app. To start a React project, which is called single-page-application in React, npx create-react-app [74] helps developers set up a ready-to-start environment with latest JavaScript features. For a view, the application is rendered API, is data which is controlled from the routes which made from models of database. For dependencies set up in server, we must install necessity packages such as cors, express, mongoose, body-parser.



```javascript
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const mongoose = require("mongoose");
const userModel = require("./models");

const app = express();
const port = 3001;

app.use(cors());

mongoose.connect('mongodb+srv://mongouser:mongouser@cluster0.9adsvqz.mongodb.net/thesisDB?retryWrites=true&w=majority'
    {
        useNewUrlParser: true,
        useUnifiedTopology: true
    }
);

const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error: "));
db.once("open", function () {
    console.log("Connected successfully");
});

// Configuring body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.get('/details', async (req, res) => {
    // const user = await userModel.findOne({id: 123});
    const user = await userModel.find({});
    res.send(user);
});

app.listen(port, () => console.log(`Hello world app listening on port ${port}!`))
```

```
imac@Rosalias-iMac simpleMERN % cd simple-node-server
imac@Rosalias-iMac simple-node-server % npm start

> simple-node-server@1.0.0 start
> node index.js

Hello world app listening on port 3001!
Connected successfully
```

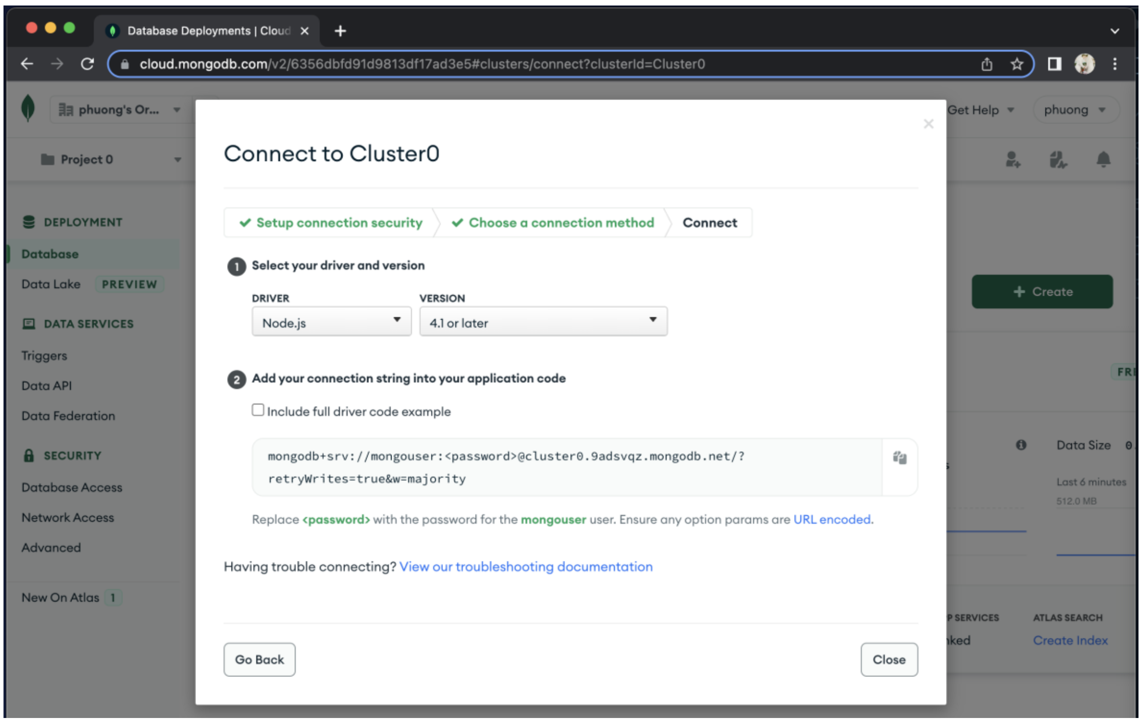
Figure 11. Picture shows a block of codes to connect front end side and back-end side applied MERN stack.



```
_id: ObjectId('6356df6f160cfb6aa8a52ee1')
id: 101
name: "user1"
```

```
_id: ObjectId('6356e07f5e40d664793ab952')
id: 102
name: "user2"
```

Figure 12. Database is created from CDN (Content Delivery Network) of MongoDB server.


Database Deployments | Cloud
cloud.mongodb.com/v2/6356dbfd91d9813df17ad3e5#clusters/connect?clusterId=Cluster0
phuong's Or...
Project 0
DEPLOYMENT
Database
Data Lake PREVIEW
DATA SERVICES
Triggers
Data API
Data Federation
SECURITY
Database Access
Network Access
Advanced
New On Atlas 1
Get Help
phuong
Create
Connect to Cluster0
Setup connection security
Choose a connection method
Connect
1 Select your driver and version
DRIVER Node.js
VERSION 4.1 or later
2 Add your connection string into your application code
Include full driver code example
Go Back
Close

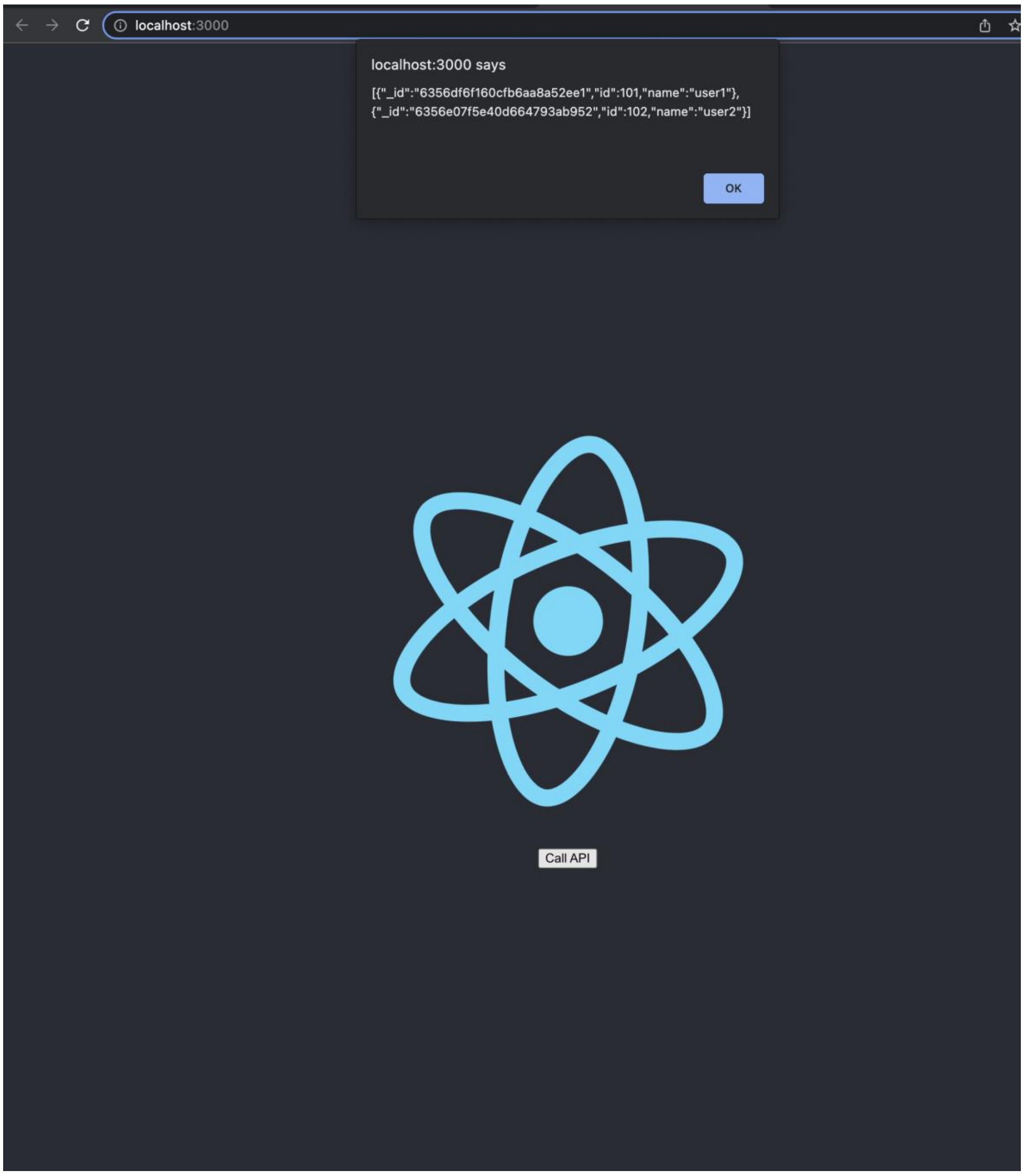
```
mongodb+srv://mongouser:<password>@cluster0.9adsvqz.mongodb.net/?
retryWrites=true&w=majority
```

Replace <password> with the password for the mongouser user. Ensure any option params are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Figure 13. The database connection is made from database deployment.

From the figure 12, data in thesisDB.users database is connected to the web application by the connection is made from figure 13.



```
[{"_id":"6356df6f160cfb6aa8a52ee1","id":101,"name":"user1"},
{"_id":"6356e07f5e40d664793ab952","id":102,"name":"user2"}]
```

Figure 14. Browser on localhost displays connection from database.

In figure 14, a final connection between the client and server which is connected from cloud database of mongoDB. illustrates how user interface should appear.

4.1 Client-Side Rendering

To build a website or a web application, client-side frameworks is not the only way to do; server-side rendering, a static site generator, a content management system can do as well. In this part of the thesis, client-side techniques should be focus on.

Because JavaScript is supported by most of browsers. Therefore, it is a big advantage for diving into the client-side frameworks of JavaScript.

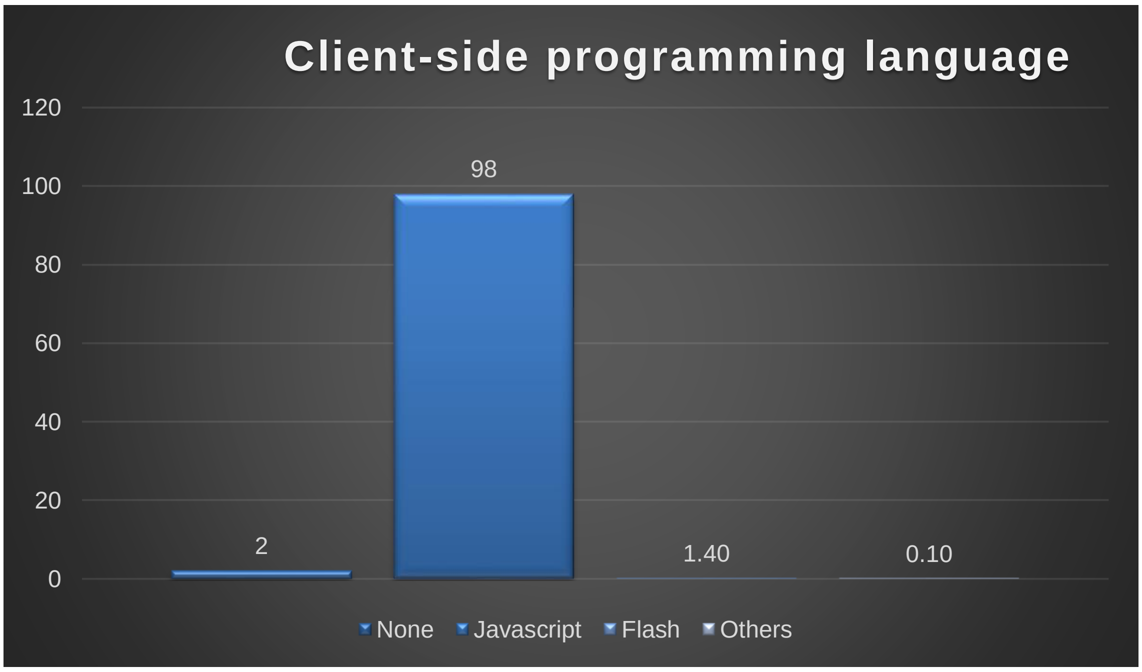


Figure 15. The chart shows the popularity of client-side programming language.

According to the statistic of the usage of client-side programming languages for websites on W3Tech.com [75], on 15 October 2022, JavaScript owns a significant number usage on client-side rendering. Consequently, many frameworks are created, followed on the rules of DSL (Domain Specific Language) [76]. Accordingly, numerous features [77] of frameworks are utilized by the power of JavaScript to streamline the data processing which builds UIs.

Name	Description	DSL types	Framework Application

JSX [https://jsx.github.io/]	JavaScript and XML. HTML combined syntax.	Embedded DSL	Vue.js, React.js
handlerbars	Reusable website template, resembles HTML.	Builder DSL	handlebars
Typescript	Strictly syntax to catch errors.	Type-safe DSL for filtering syntax	typescript
Component model	Writing module.	Functional DSL	React.js, Ember.js, Angular.js, Vue.js
Properties/props	External data of a rendered component.	Monadic DSL	React.js
State	State handling mechanism.	Monadic DSL	React.js
Events	Respond.	internal DSL	N/A
Styling components	Style module.	Embedded DSL	Sass, Less, PostCSS
Handling dependencies	JavaScript module system.	Builder DSL	Node.js, babel, webpack, Require.js, Common.js
Components in components	Component composes together.	internal DSL	React.js
Dependency injection	Nested levels in a component to	Functional DSL	Vue.js, React.js, Ember.js

	pass data through layers.		
Lifecycle	Stage collection	Builder DSL	React.js
Rendering elements	Virtual DOM	Embedded DSL	Vue.js, React.js
Routing	Client-side routing, navigation of view/layout.	Monadic DSL	React.js, Angular.js
Testing	Tools for JavaScript environments.	Testing DSL	Testing library

Table 4. List of main features of JavaScript frameworks.

The problems can easily happen due to the mingles of features. The priorities to make a complex application which satisfy users' expectations seems to be fast, convenient, and simple by the combination of JavaScript modules, or JavaScript technology stacks. However, the ignorance of HTML-based can make applications become inaccessible, or applications lose the coherence. For this reason, modern application makers should prioritize the purpose with far sight to arrange the performance of capacity of tech-stacks to make sure the adaptability, then the user experience takes.

Thanks to innovation of V8 engine, NodeJS become a competitive ecosystem which have multipurpose frameworks of JavaScript can support render tasks, update data from client-side as well as server-side.

In other words, server renders JSON object through RESTful API by React components with MVC frameworks. Technology applications from client side must handle sensitive data well. It means data must well organized in protected state, not stored.

4.2 Server-Side Rendering

Building a dynamic website/web application to deliver data from the interaction between users' devices and database server is complicated. On top of the security issues, the management and control sensitive data grow into the art of handling data. Being on the case, performance is another hard challenge need to be skip at first. Many server-side frameworks of JavaScript offer countless methods to make data is secured.

Every website or web application always need a web server to host the application. And developers need GUIs (Graphical User Interface) as tools to have eyesight what is happening in implementation codes, such as APIs, middleware, uploading files to server, querying database.

To start understanding what technology can integrated with HTML, CSS, JavaScript on browsers, which means client-side rendering, we have a brief introduction of server-side frameworks. A website may use more than one programming language on server-side rendering.

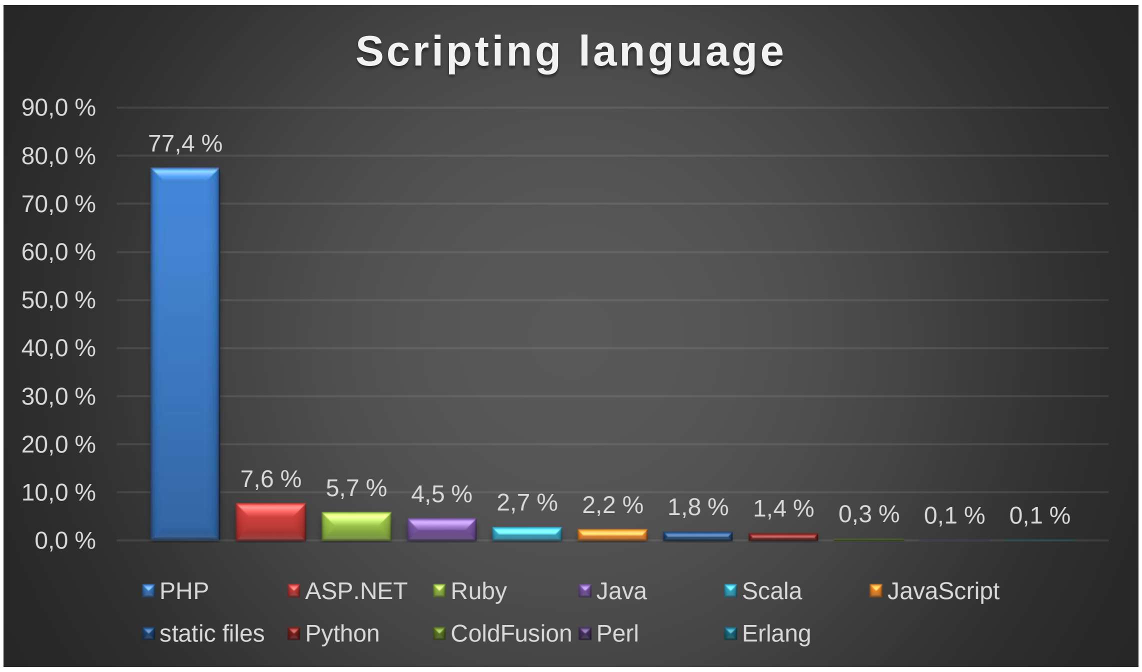


Figure 16. Lists of server-side programming languages for website on W3Tech [78].

In figure 16, PHP is the most popular programming language. On the other hand, JavaScript is the most popular client-side programming. Having insight thought on business, we could inspect the report of market position of server-side programming, on W3Tech to have a big picture before having ideas of doing software products.

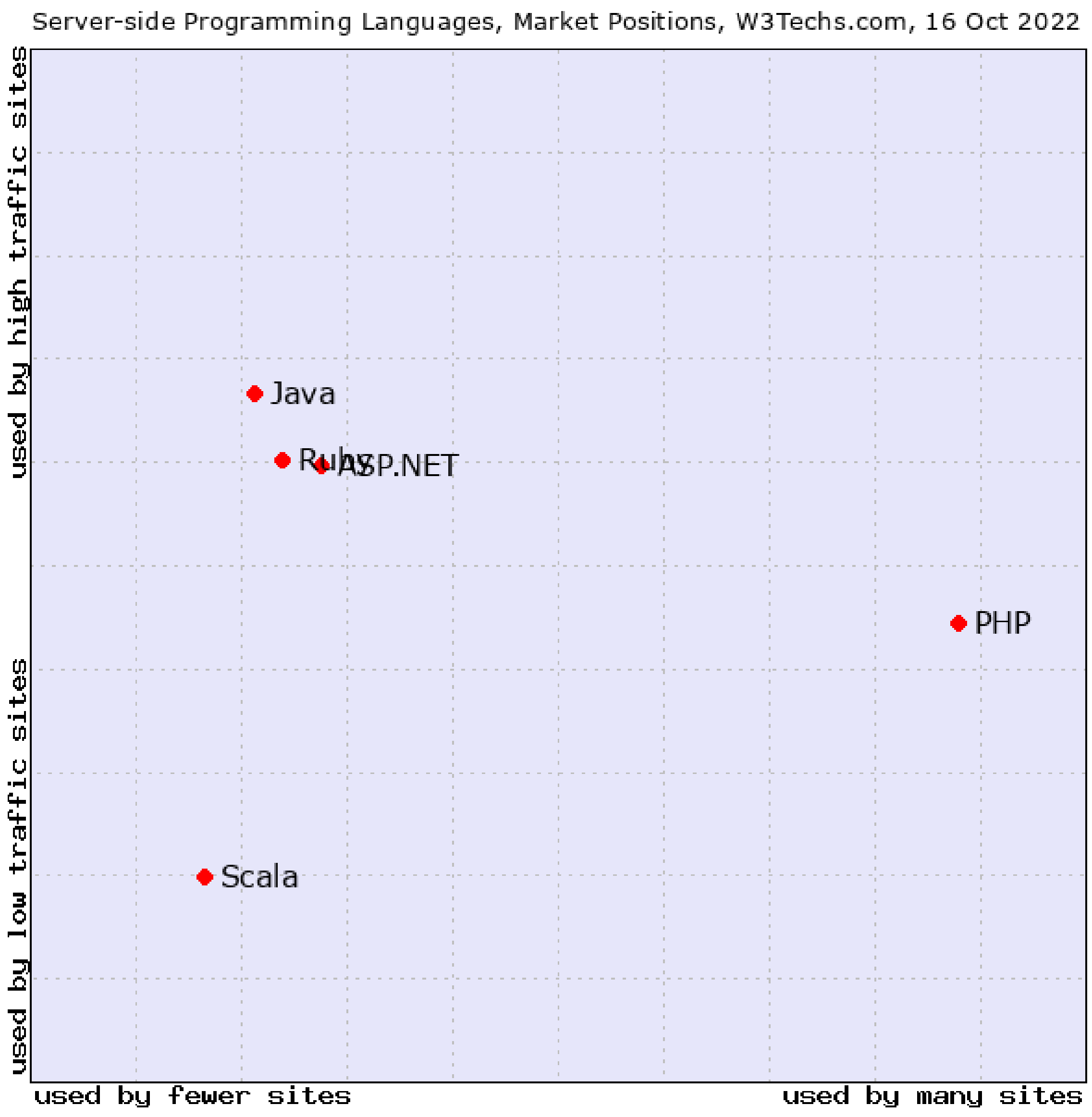


Figure 17. Market position of server-side programming language in reference to popularity and traffic of the 5 most popular server-side scripting.
Source: https://w3techs.com/technologies/market/programming_language

The diagram illustrates technology of Java, Ruby, ASP.NET, is deployed by fewer sites, but highly traffic, and PHP is used by many sites but lower traffic sites, in reverse. This data shows us an acknowledge of the popularity is not on the performance, neither on security issues.

For instance, Scala [79] is a programming language with object-oriented features mixed seamlessly with functional programming features. Scala is exploited by top technology companies [80] such as Linkedin, Twitter, Netflix, AirBnB, Tumblrb, etc. In this part of the thesis, we can find deep analysis of different scripting languages on server-side technology.

Programming language	Advantage	Disadvantage
PHP	Platform-independent, flexibility, easy management, stability.	Poor quality of secure data, error-handling, debugging tools.
Java	Secure language, object-oriented, platform-independent, stable, multi-threaded.	Slow performance, require memory space, verbose and complex codes, automatic garbage collection.
Ruby	Ruby on Rails web application framework.	Slow processing, lower flexibility.
Python	Interpreted language [81], clear and readable syntax, diverse libraries, multi-purpose programming language, asynchronous development, high compatible with multiple programming languages.	Low speed, memory consumption, database access, runtime error.

ASP.NET	Modular support, web UI and web APIs, hosting, fast and open-source, cross platform, dependency injection support, minimize version conflict of software development.	Lack of 3rd party library support, workflow service, WCF services implementation.
Scala	Scalability, highly functional paradigm, inherently immutable objects, IDE support.	Hybrid paradigm, small community.
NodeJS (JavaScript)	Real time application, scalability, caching, cross platform, large community.	Unstable API, asynchronous programming model, poorly supported libraries.

Table 5. List of the for and against of diverse programming languages.

Server-side technology is still developing, despite the drawbacks of compatibility, support, security, and community as well as overwhelming of emerging frameworks, libraries, dependencies. In that event, we have a look at stackoverflow community to understand which popular programming languages, popular frameworks of JavaScript, developer choose to work with.

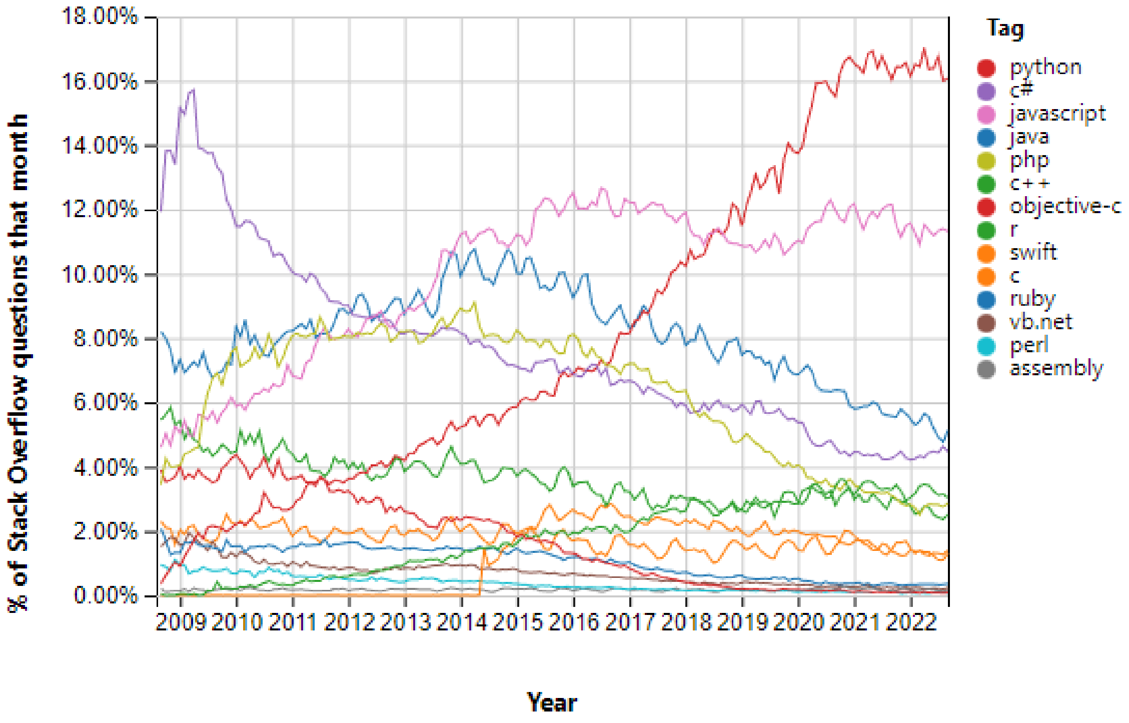
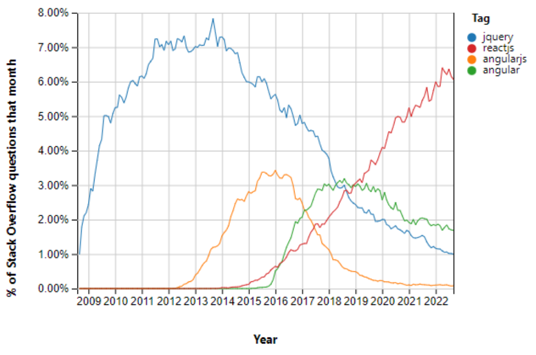


Figure 18. Statistic based on the questions of programming language from 2009 to 2022.

Source: https://insights.stackoverflow.com/trends



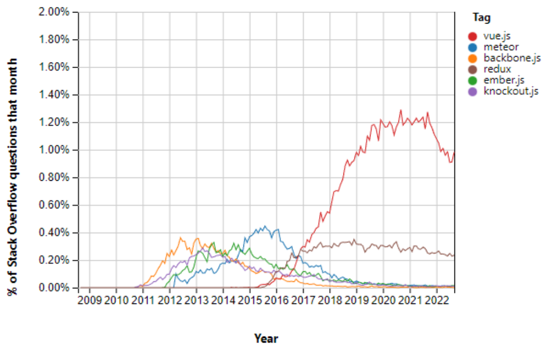


Figure 19. Statistic based on the questions of JavaScript frameworks from 2009 to 2022.

Source: https://insights.stackoverflow.com/trends

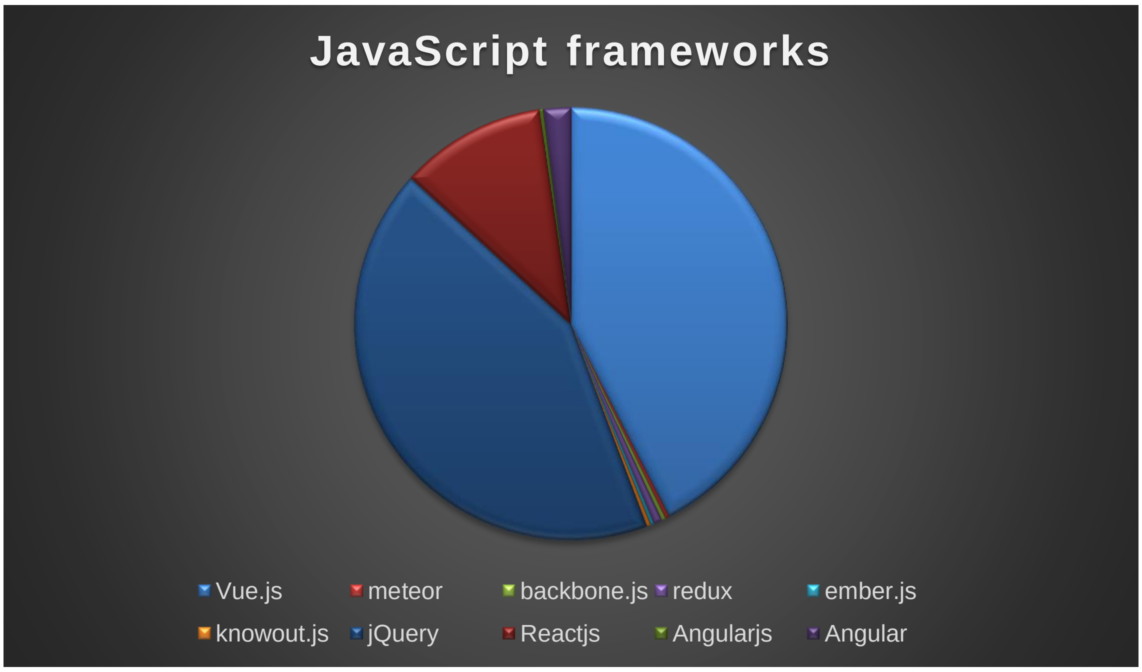


Figure 20. Percentage of questions received on stackoverflow community, according to different types of frameworks of JavaScript, in 2022.

Regardless of using frameworks, pure [82] Node.js can build static files server by it own. To configure server properly, media type of content must be described by MIME types [83] to avoid violating the HTTP process.

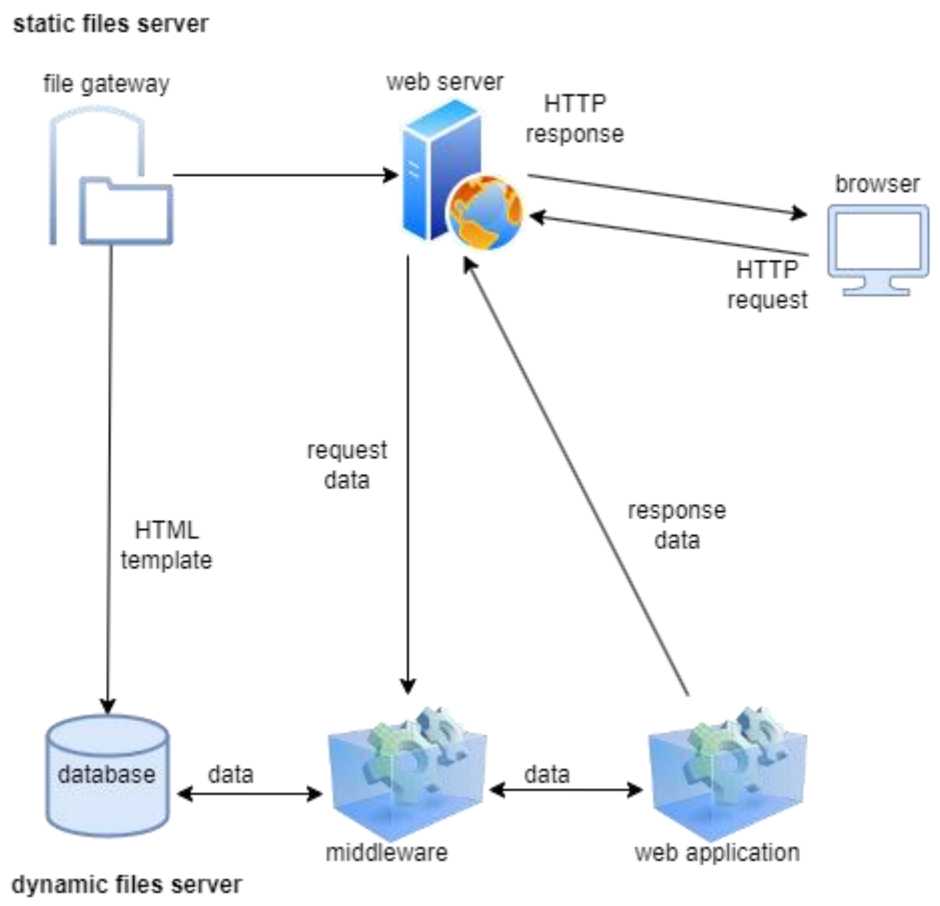


Figure 21. HTTP process in static file server and dynamic file server.

In figure 21, to boost the performance of data handling, static file server is created to generate HTML template, and combine its templates with requested data contains URL encoding, GET/POST data, cookies in middleware's web application. After that operation, HTTP response will send executed codes in HTML files to browser.

4.3 OAuth 2.0 protocol

OAuth 2.0 is a protocol for authentication and authorization of Google APIs [84]. Application developers retrieve OAuth 2.0 credentials from the Google API platform, with an access token from the Google Authorization server. Then, level of permission will be granted to individual user by inspecting scope of access. And there are distinct role models based on the access token which direct request to an API which storing suitable data.

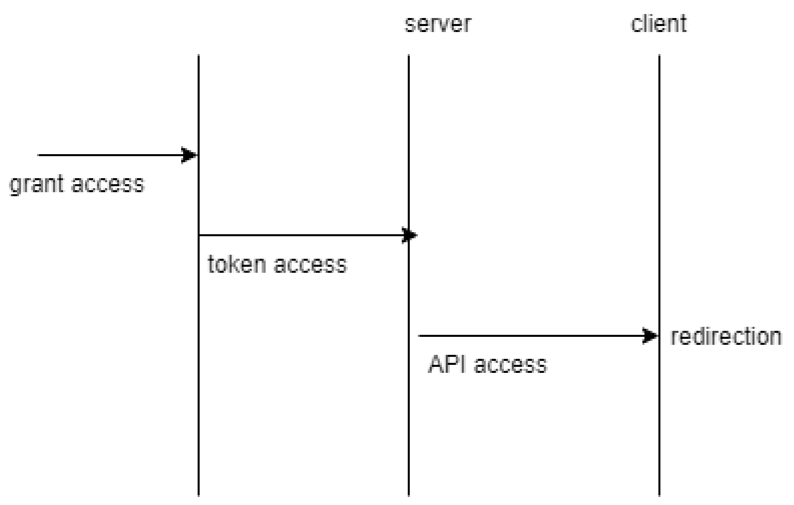


Figure 22. Workflow of three endpoints of OAuth 2.0 protocol processing.

Figure 22 shows the business flow of authorization process of OAuth 2.0 protocol. At the end, client can receive a redirection which return data. After the authorized server taking grant access, it sends the token access to support OAuth client get the requested data from resource owner of resource server. Therefore, API represents URL string of HTTP with the specific address.

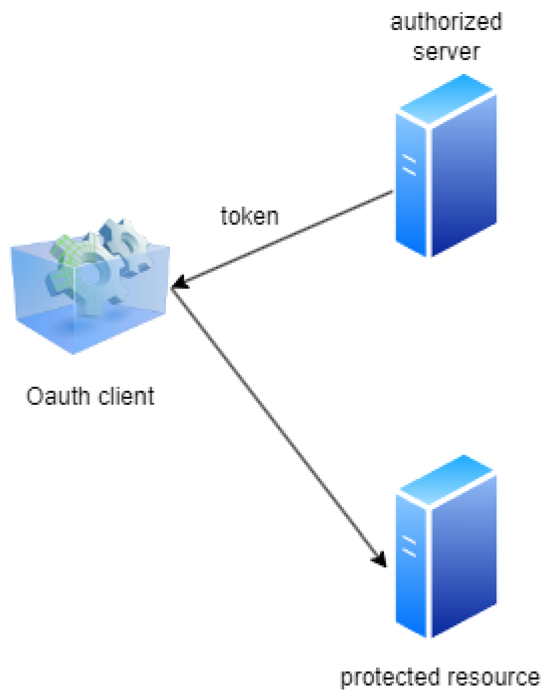


Figure 23. OAuth client is allowed to permitted resource by token.

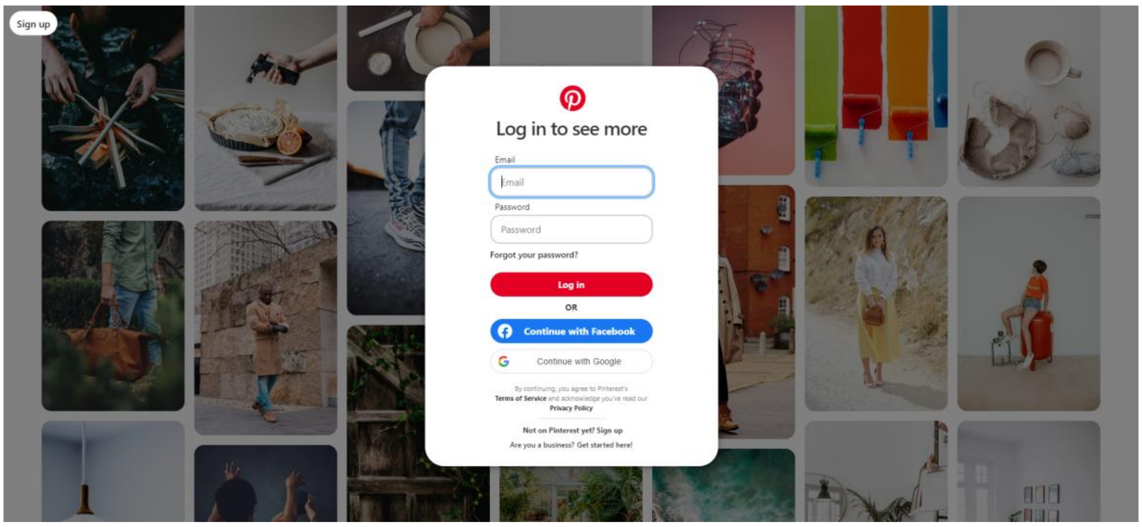
In figure 23, after building OAuth client, a token was sent from authorized server for OAuth client usage at protected resource of resource server. There are many options of parameters for designing APIs. Different scope brings divergent results to provide an API service in validating scope selections of the role for authorization procedure.

OAuth 2.0 have main 2 endpoints [85] oauth/token and /authorize with disparate request parameters [86]. OAuth is only a delegation mechanism, a provision tool for creating the access token after authorizing users' roles of each login account.

4.4 Case study of a login module

To understand how OAuth 2.0 works, the login module of entertainment application, is named Pinterest, is used for a case study.

Pinterest has 3 methods to login, including login with Facebook, login with Google, login with Pinterest account. After the account is verified, user can be redirected to the existed account's workspace, or leads to admin page based on admin role of authorization system to modify which fields user wants to follow, setting account preferences, and protect account with two-factor authentication.



By continuing, you agree to Pinterest's Terms of Service and acknowledge you've read our Privacy Policy

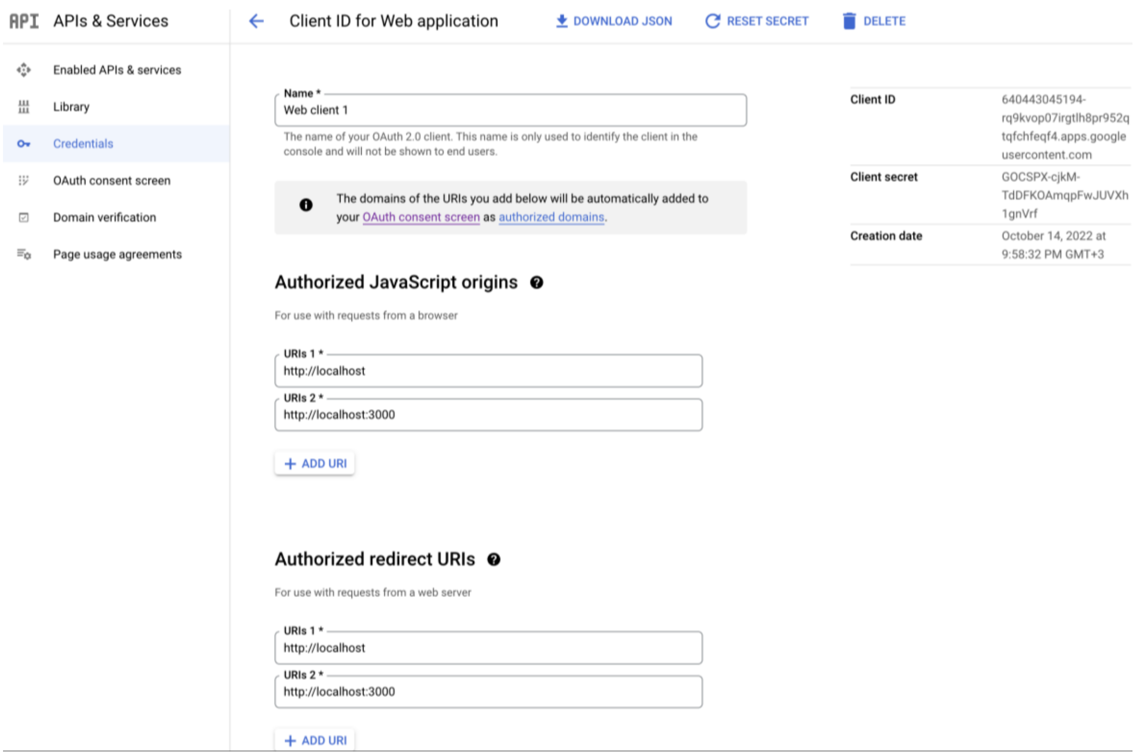
Not on Pinterest yet? Sign up

Are you a business? Get started here!

Figure 24. Form login of Pinterest application.

4.4.1 Implement a Google sign-in button

To deployment code [87] for 3rd party login from Google, we need to create credentials in Google API platform, to take Client ID from Google.



Name *
Web client 1

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.

Client ID	640443045194-rq9kvop07irgtlh8pr952qtqfchfeqf4.apps.googleusercontent.com
Client secret	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
Creation date	October 14, 2022 at 9:58:32 PM GMT+3

Authorized JavaScript origins

For use with requests from a browser

URIs 1 *
http://localhost

URIs 2 *
http://localhost:3000

+ ADD URI

Authorized redirect URIs

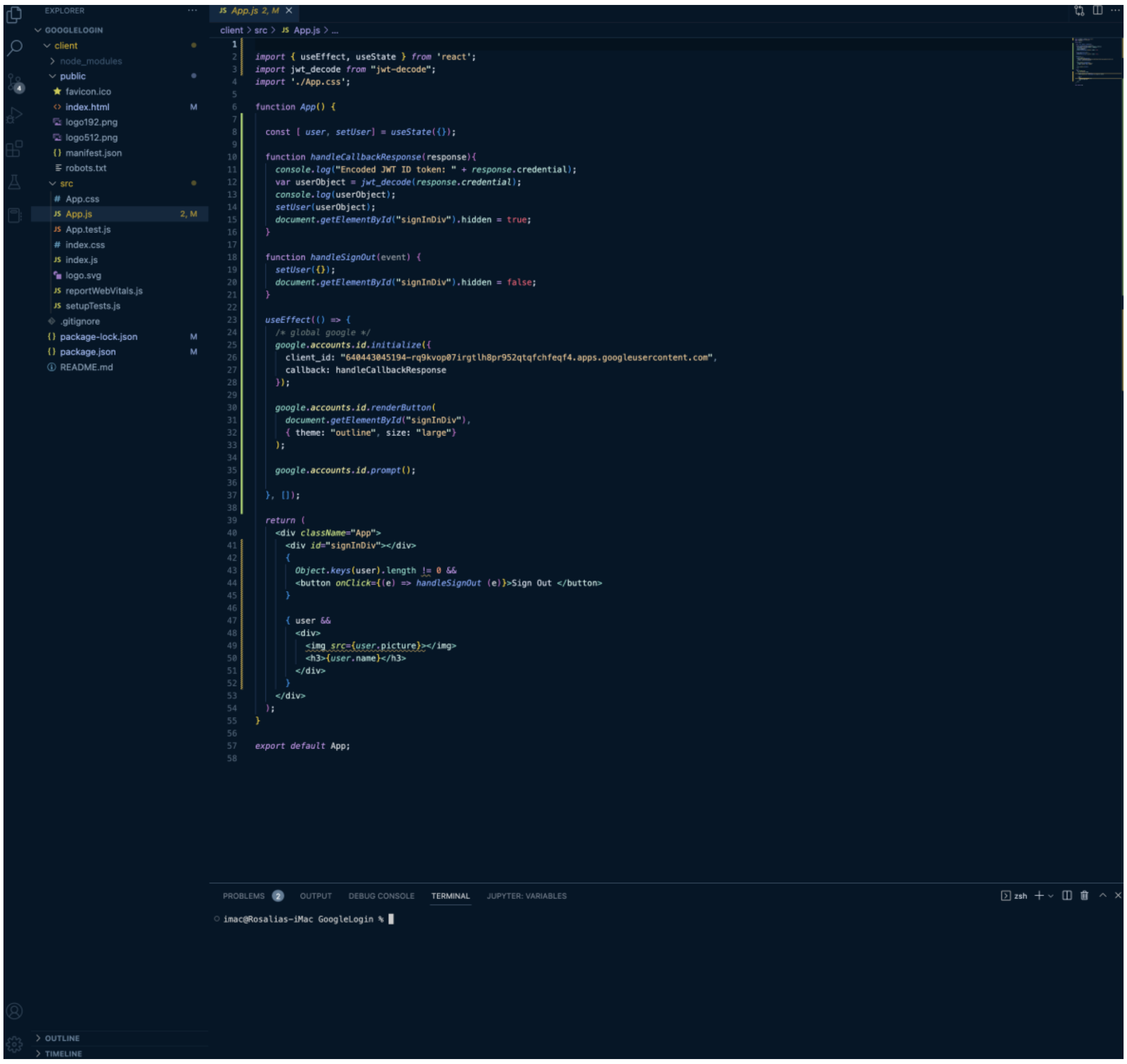
For use with requests from a web server

URIs 1 *
http://localhost

URIs 2 *
http://localhost:3000

+ ADD URI

Figure 25. Client ID and Client secret from Google.



```javascript

import { useEffect, useState } from 'react';
import jwt_decode from "jwt-decode";
import './App.css';

function App() {

  const [ user, setUser] = useState({});

  function handleCallbackResponse(response){
    console.log("Encoded JWT ID token: " + response.credential);
    var userObject = jwt_decode(response.credential);
    console.log(userObject);
    setUser(userObject);
    document.getElementById("signInDiv").hidden = true;
  }

  function handleSignOut(event) {
    setUser({});
    document.getElementById("signInDiv").hidden = false;
  }

  useEffect(() => {
    /* global google */
    google.accounts.id.initialize({
      client_id: "640443045194-rq9kvop07irgtlh8pr952qtqfchfeqf4.apps.googleusercontent.com",
      callback: handleCallbackResponse
    });

    google.accounts.id.renderButton(
      document.getElementById("signInDiv"),
      { theme: "outline", size: "large"}
    );

    google.accounts.id.prompt();

  }, []);

  return (
    <div className="App">
      <div id="signInDiv"></div>
      {
        Object.keys(user).length != 0 &&
        <button onClick={(e) => handleSignOut (e)}>Sign Out </button>
      }

      { user &&
        <div>
          <img src={user.picture}></img>
          <h3>{user.name}</h3>
        </div>
      }
    </div>
  );
}

export default App;

```

```
imac@Rosalias-iMac GoogleLogin %
```

Figure 26. Picture shows code folder structure for Google login/logout.

In index.js, DOM of React can render the UI with the modify from .css file, code of hook like useEffect in App.js is using client ID from Google as the main responsibility for making authorization with Google account by token.

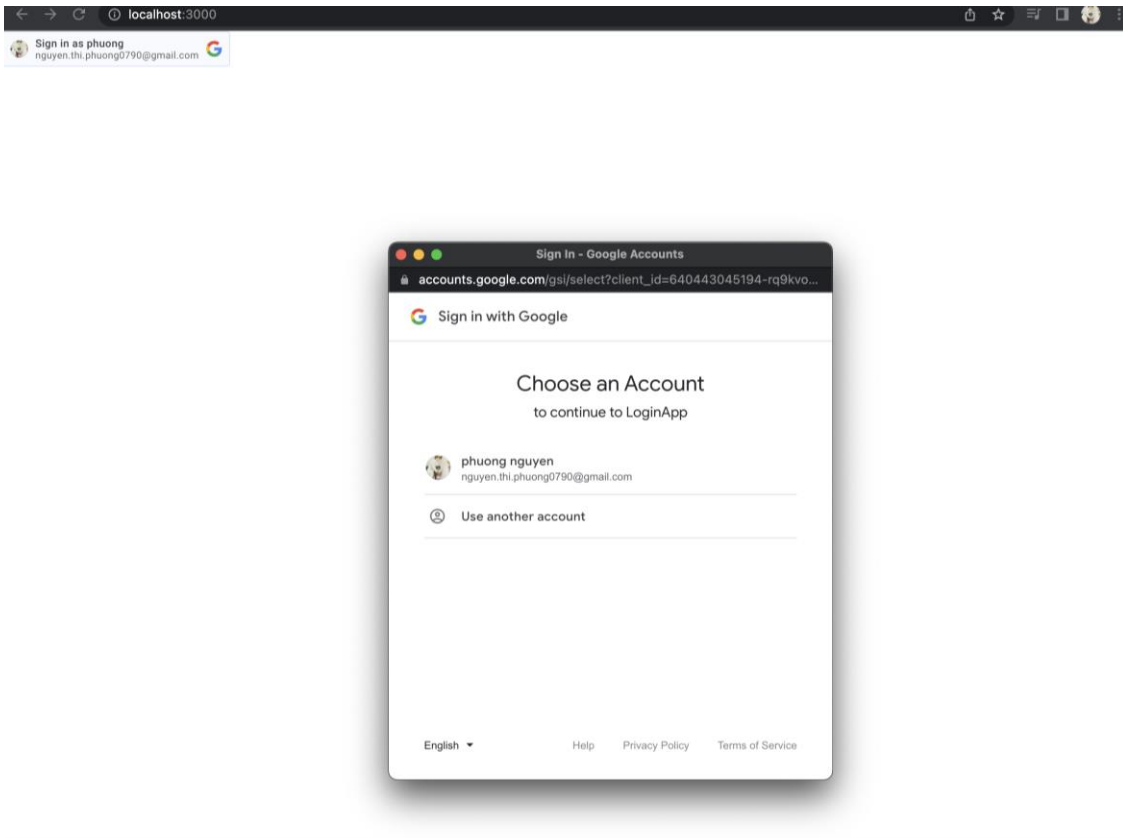


Figure 27. Sign in button from Google.

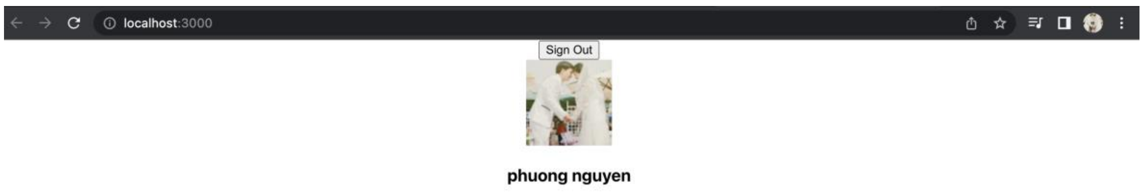


Figure 28. Sign out button from Google.

After sign in successful by Google account, user is sent into a link, with the log out function inside the page.

4.4.2 Network transferring in a login system

For educational purposes, in this thesis, the network traffic analysis [88] is only applied in a small area to understand the performance of 3rd party API through TCP communication.

After signing in as username of Google or Facebook or Pinterest account, Hypertext transfer protocol is used to access the data already have on the world wide web (WWW). The network package is transferred online through the TCP protocol [89] with unordered data, error control, flow control, data segments retransmitted. Then, the URL redirects to a precise address on HTTP link.

For example, after login, Pinterest access to the https://fi.pinterest.com/ link, with the status code is 200. If there are any problems with the authorization, the package network will try to notice with the status code 401 [90].

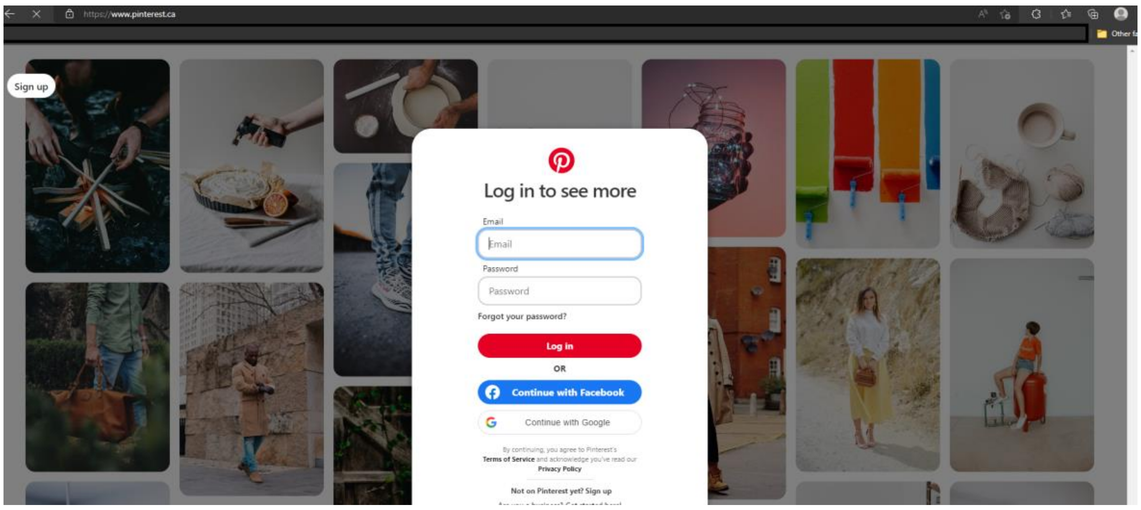
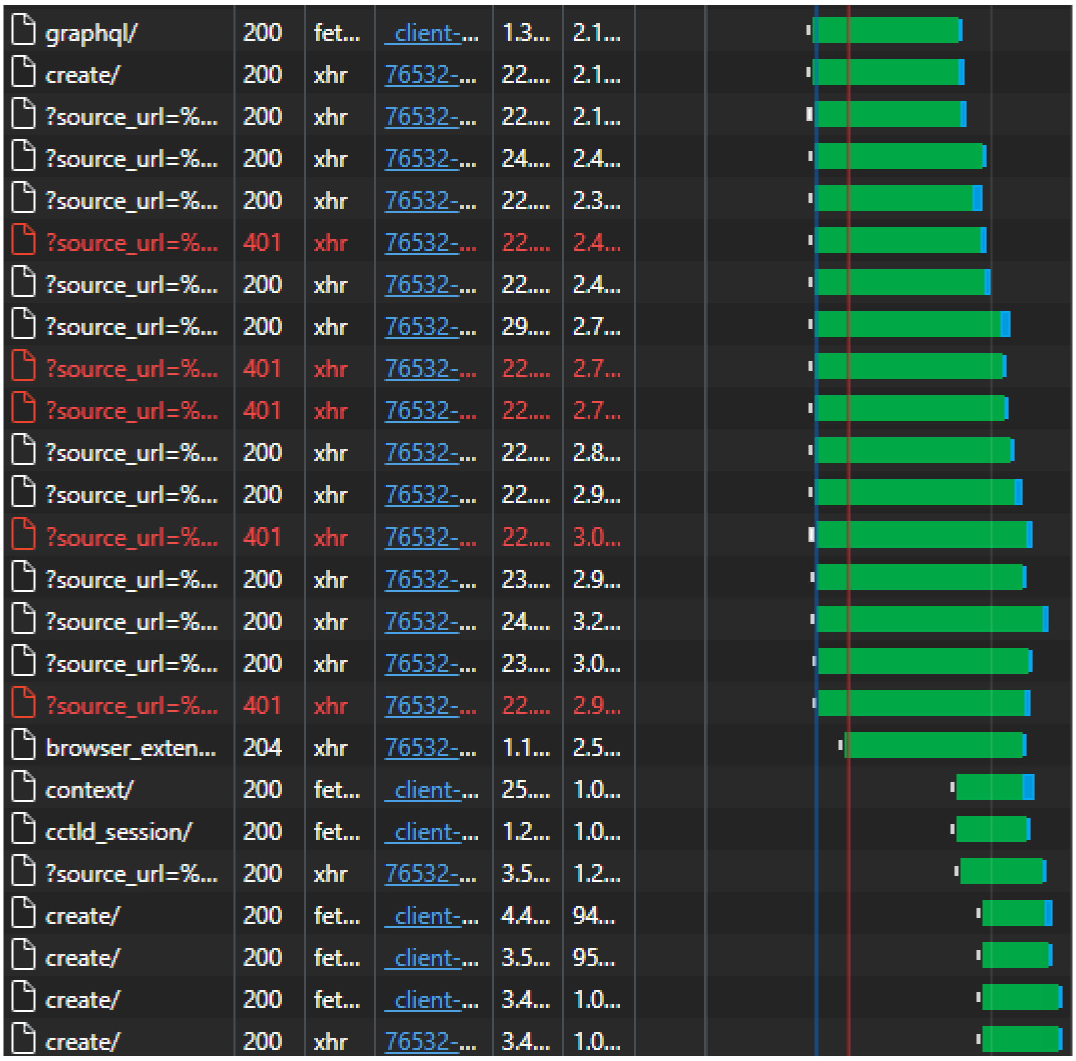


Figure 29. Login page of Pinterest in Canada.

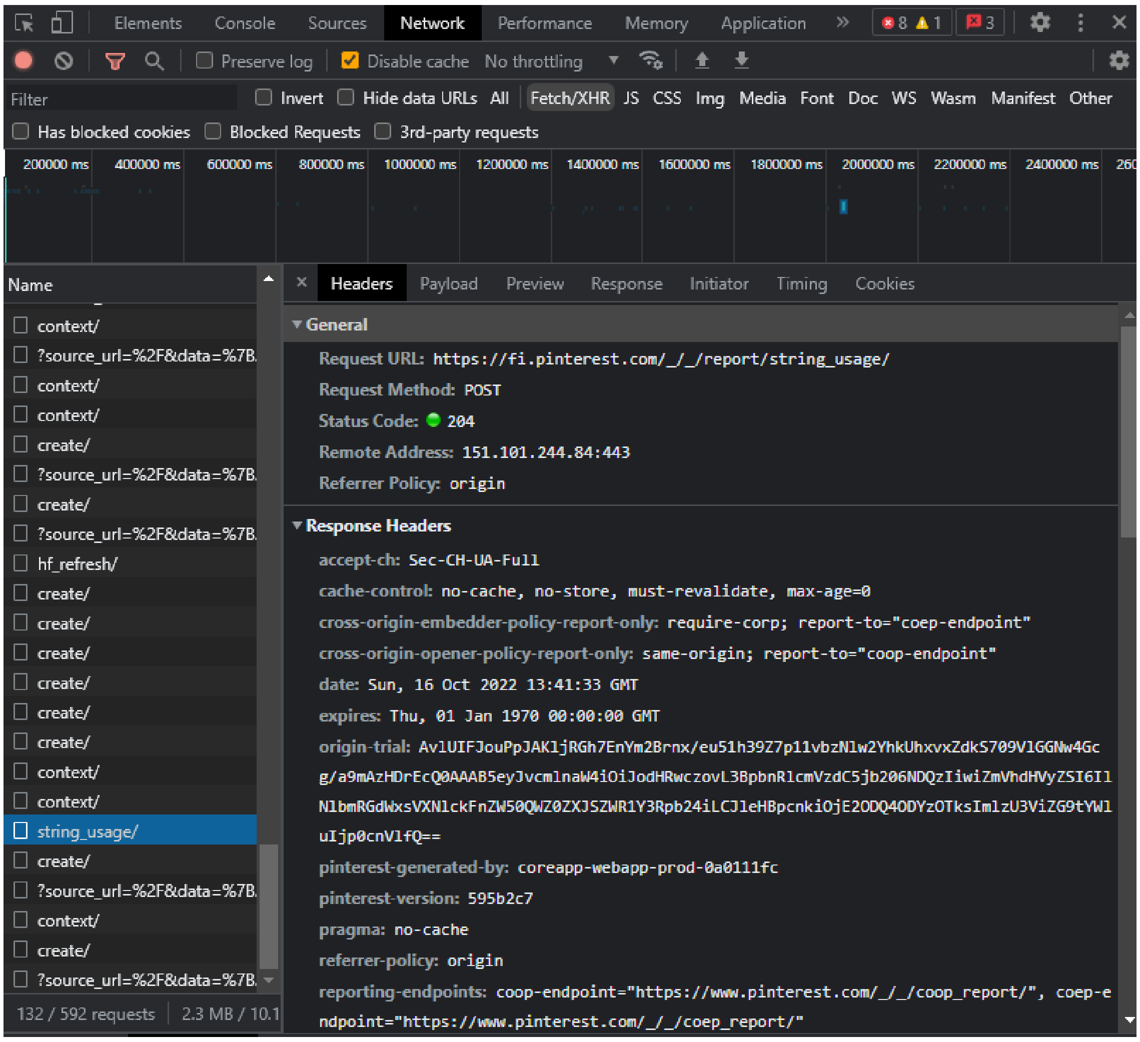


graphql/	200	fet...	client-...	1.3...	2.1...
create/	200	xhr	76532-...	22....	2.1...
?source_url=%...	200	xhr	76532-...	22....	2.1...
?source_url=%...	200	xhr	76532-...	24....	2.4...
?source_url=%...	200	xhr	76532-...	22....	2.3...
?source_url=%...	401	xhr	76532-...	22....	2.4...
?source_url=%...	200	xhr	76532-...	22....	2.4...
?source_url=%...	200	xhr	76532-...	29....	2.7...
?source_url=%...	401	xhr	76532-...	22....	2.7...
?source_url=%...	401	xhr	76532-...	22....	2.7...
?source_url=%...	200	xhr	76532-...	22....	2.8...
?source_url=%...	200	xhr	76532-...	22....	2.9...
?source_url=%...	401	xhr	76532-...	22....	3.0...
?source_url=%...	200	xhr	76532-...	23....	2.9...
?source_url=%...	200	xhr	76532-...	24....	3.2...
?source_url=%...	200	xhr	76532-...	23....	3.0...
?source_url=%...	401	xhr	76532-...	22....	2.9...
browser_exten...	204	xhr	76532-...	1.1....	2.5...
context/	200	fet...	client-...	25....	1.0...
cctld_session/	200	fet...	client-...	1.2...	1.0...
?source_url=%...	200	xhr	76532-...	3.5...	1.2...
create/	200	fet...	client-...	4.4...	94....
create/	200	fet...	client-...	3.5...	95....
create/	200	fet...	client-...	3.4...	1.0...
create/	200	xhr	76532-...	3.4...	1.0...

Figure 30. XMLHttpRequest object which interact with the server have error.

In figure 17, URL has been replaced by https://www.pinterest.ca, because the authorization cannot be completed, the server continues looping request and response to the server until it can be verified. If the browser starts non-stop iterate the process, end-user can clear his/her cookies, caching in history of the browser.

When authorization process is finished, we can examine the network package send and receive data in Pinterest web application. HTTP request and response objects made of body and header. When headers [91] play a role of a bridge to helps client and server pass authenticated mandatory information, body holds the data package [92] of HTML and JSON.



```
Name
context/
?source_url=%2F&data=%7B.
context/
context/
create/
?source_url=%2F&data=%7B.
create/
?source_url=%2F&data=%7B.
hf_refresh/
create/
create/
create/
create/
create/
create/
context/
context/
string_usage/
create/
?source_url=%2F&data=%7B.
context/
create/
?source_url=%2F&data=%7B.
132 / 592 requests | 2.3 MB / 10.1
```

```
▼ General
    Request URL: https://fi.pinterest.com/_/_/report/string_usage/
    Request Method: POST
    Status Code: 204
    Remote Address: 151.101.244.84:443
    Referrer Policy: origin
▼ Response Headers
    accept-ch: Sec-CH-UA-Full
    cache-control: no-cache, no-store, must-revalidate, max-age=0
    cross-origin-embedder-policy-report-only: require-corp; report-to="coep-endpoint"
    cross-origin-opener-policy-report-only: same-origin; report-to="coop-endpoint"
    date: Sun, 16 Oct 2022 13:41:33 GMT
    expires: Thu, 01 Jan 1970 00:00:00 GMT
    origin-trial: Av1UIFJouPpJAK1jRGh7EnYm2Brnx/eu51h39Z7p11vbzNlw2YhkUhxvxZdkS709V1GGNw4Gc
    g/a9mAzHDrEcQ0AAAB5eyJvcmlnaW4iOiJodHRwczovL3BpbnRlcmVzdC5jb206NDQzIiwiZmVhdHVyZSI6Il
    NlbmRGdWxsVXNlckFnZW50QWZ0ZXJSZWR1Y3Rpb24iLCJleHBpcnkiOjE2ODQ4ODYzOTksImlzU3ViZG9tYWl
    uIjp0cnVlfQ==
    pinterest-generated-by: coreapp-webapp-prod-0a0111fc
    pinterest-version: 595b2c7
    pragma: no-cache
    referrer-policy: origin
    reporting-endpoints: coop-endpoint="https://www.pinterest.com/_/_/coop_report/", coep-e
    ndpoint="https://www.pinterest.com/_/_/coep_report/"
```

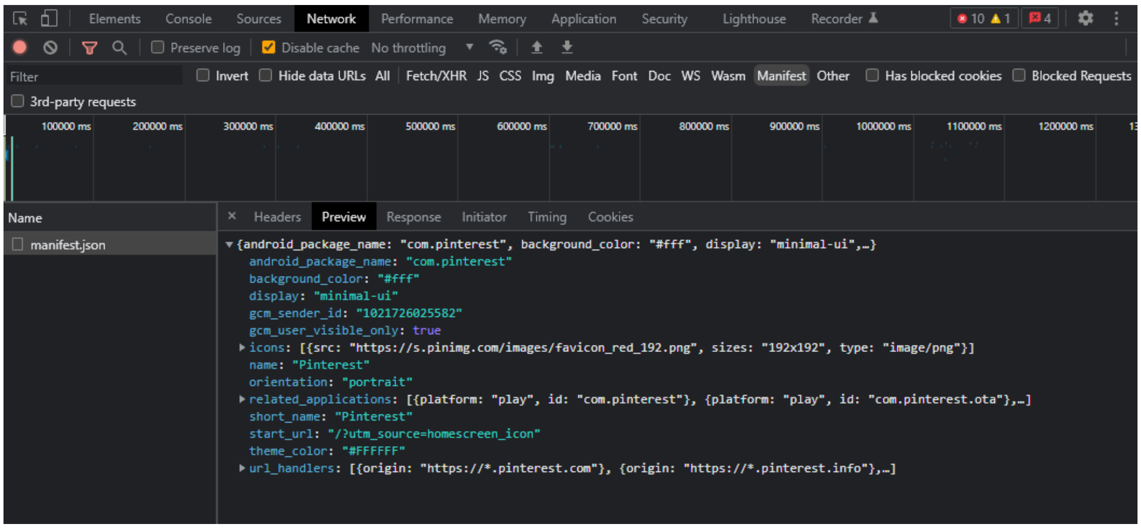
Figure 31. Fetch/XHR data examines on Network tab of Developer tool.

On Network tab of developer tool, Fetch/XHR events shows 2 types of data: xhr [93] and fetch [94] with status code is 200. Especially, string_usage/, hf_refresh/, browser_extension/ has status code 204 [95], it means the request has succeeded with POST/PUT [96] request payload, and there are no content at the current page.

With Initiator tab, there are full stack trace information of asynchronous callback process. Timing tab show TCP connection open in browser queue request with detail time. Cookies tab contains value of _routing_id, csrftoken, _auth, pinterest_sess, _b, cm_sub.

Getting more attention on JS files, Pinterest use radar.js as origin policy, recapcha_en.js as strict-origin-when-cross-origin as well as origin. Pinterest also have scripts for some events such as QuickSave button, SendShareLink button, and mjs [97] for embedded API; stylesheet for elements' styles such as icon, buttons, background, fonts, text, container, even special font language including Cyrillic, Greek, Hebrew, Latin, Vietnamese. Special font language [98] is supported external font files by WOFF (Web Open Font Format), and Pinterest implements WOFF 2.0 for byte-level compression by the recommendation of W3C.

Because Pinterest is famous for picture sharing platform, the challenge is handling image data, <div> tag with class need to store images in jpeg form to keep the neat style. The last but not the least, is manifest.json [99] file, is used to state metadata of the application.



```
▼{android_package_name: "com.pinterest", background_color: "#fff", display: "minimal-ui",…}
   android_package_name: "com.pinterest"
   background_color: "#fff"
   display: "minimal-ui"
   gcm_sender_id: "1021726025582"
   gcm_user_visible_only: true
 ▶ icons: [{src: "https://s.pinimg.com/images/favicon_red_192.png", sizes: "192x192", type: "image/png"}]
   name: "Pinterest"
   orientation: "portrait"
 ▶ related_applications: [{platform: "play", id: "com.pinterest"}, {platform: "play", id: "com.pinterest.ota"},…]
   short_name: "Pinterest"
   start_url: "/?utm_source=homescreen_icon"
   theme_color: "#FFFFFF"
 ▶ url_handlers: [{origin: "https://*.pinterest.com"}, {origin: "https://*.pinterest.info"},…]
```

Figure 32. JSON formatted file for defining metadata of Pinterest.

In figure 20, Pinterest's manifest file is in JSON format. With the name of android package means Pinterest started the application with Android [100] first, with the background colour [101] is white to enhance the general colour contrast. Next, display mode is minimal-ui [102] means the application's elements display in differ by browser. Default orientation is on [103] vertically fit with the devices, with the icon as in figure 21.

Figure 33. The Pinterest icon appears as application's logo on home screen of users' devices.
Source: https://s.pinimg.com/images/favicon_red_192.png

And the last one is url_handlers [104], using the prefix and suffix of the root link, being on https://*.pinterest.com, then its convert into 21 origin links: .com, .info, .at, .com.au, .ca, .ch, .cl, .de, .dk, .es, .fr, .co.uk, .ie, .it, .jp, .co.kr, .com.mx, .nz, .se, .ph, .pt, .ru and so much more combination generates from the root url with * symbol such as https://no.pinterest.com, https://fi.pinterest.com, et cetera.

That's all for the basic surfing network package to acknowledge what a web application needs to take care of in real-life. The gauge of making a web application should focus on merging the UI modules, data handling, embedded DSL, 3rd party API, encryption in particular HTTPS [105], supported libraries and the mingle of multi programming languages for optimizing the performance of time speed, memory leaks and memory consumption in web applications.

5 Conclusion

If an adequate preparation in knowledge is made, the logic approach of a good understanding of information technology knowledge in interaction of web applications directly influences the way of programming mindset. Since newbie of developers' world, can hook what is fashionable modern technology, based on knowledge of software design, they utterly can create brand-new frameworks, or programming language in the future. Via case study of Pinterest application, a login module mingles local login with 3rd party login from Google strategy, Facebook strategy, network packages are analysed to supply an awareness of the inner style of how HTTP mechanism works. The topic also strives to decontaminate redundant pieces of ambiguous parts of programming language which leads to an excessively deep, overwhelming side of enormous size of JavaScript frameworks.

In this thesis, there is a merely guide to start a passion in expanding experience of self-contained a website, web application from a novice level, especially in JavaScript. Therefore, the knowledge is conveyed in a simplest, shortest way which enough to understand how an interface from user can interact with database's connection from a web server. It called modern technology of this era where people are not only read information but also work with data. Consequently, security issues are a vast sector which this thesis cannot mention deeply.

In future, a login feature can be used as an integrated module to combine with others multi-purposed application such as e-commerce, education, entrepreneur companies, organizations. JavaScript is still developing days after days thanks to open sources and SOLID principles [106] of software design [107]. The security levels soon will be better and better by middleware package addition in every day of development. JavaScript at first day is only a client-side rendering, and nowadays, JavaScript is evolved into a server-side rendering, the JavaScript security frameworks will be invented likewise.

References

1 TCP 3-Way Handshake Process. Web document accessed 25.06.2022. https://www.geeksforgeeks.org/tcp-3-way-handshake-process/

2 Roy T. Fielding: Understanding the REST Style. Web document accessed 25.06.2022. https://www.computer.org/csdl/magazine/co/2015/06/mco2015060007/13rRUx0xPA5

3 REST Architecture. Web document accessed 25.06.2022. https://www.redhat.com/en/blog/rest-architecture

4 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Web document accessed 25.06.2022.
https://datatracker.ietf.org/doc/html/rfc7231

5 Steps toward the glory of REST. Web document accessed 25.06.2022.
https://martinfowler.com/articles/richardsonMaturityModel.html

6 What is a REST API? Web document accessed 25.06.2022.
https://www.redhat.com/en/topics/api/what-is-a-rest-api

7 What is an API? Web document accessed 25.06.2022.
https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

8 REST Architectural Constraints. Web document accessed 25.06.2022.
https://restfulapi.net/rest-architectural-constraints/

9 What Is an API Endpoint? Web document accessed 26.06.2022.
https://blog.hubspot.com/website/api-endpoint

10 Top 50 Cybersecurity Statistics, Figures and Facts. Web document accessed 26.06.2022.
https://connect.comptia.org/blog/cyber-security-stats-facts

11 Fetch API. Web document accessed 26.06.2022.
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

12 Cross-Origin Resource Sharing (CORS). Web document accessed 26.06.2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

13 Origin. Web document accessed 26.06.2022.
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin

14 Basics of CORS (Cross-Origin Resource Sharing). Web document accessed 26.06.2022.
https://dzone.com/articles/basics-of-cors

15 JavaScript: The Good Parts, Douglas Crockford, 2008.

16 Usage statistics of client-side programming languages for websites. Web document accessed 27.06.2022.
https://w3techs.com/technologies/overview/client_side_language

17 Document Object Model Specification. Web document accessed 03.07.2022.
https://www.w3.org/TR/1998/WD-DOM-19980416/

18 AJAX Introduction. Web document accessed 03.07.2022.
https://www.w3schools.com/xml/ajax_intro.asp

19 Performance Optimization using MERN stack on Web Application. Web document accessed 04.07.2022.

https://www.ijert.org/research/performance-optimization-using-mern-stack-on-web-application-IJERTV10IS060239.pdf

20 Quora Question Pairing system. Web document accessed 04.07.2022. https://github.com/kashaudhan/questionPairing

21 Node.js v18.4.0 documentation. Web document accessed 04.07.2022. https://nodejs.org/api/child_process.html

22 The Elements of User Experience, Second Edition: User-Centered Design for the Web and Beyond, Jesse James Garrett, 2010.

23 What is Azure Pipelines? Web document accessed 04.07.2022. https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines

24 What is Azure Machine Learning? Web document accessed 04.07.2022. https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning

25 Full Stack Machine Learning on Azure. Web document accessed 04.07.2022. https://towardsdatascience.com/full-stack-machine-learning-on-azure-f0f6b77be07e

26 HTML Tutorial. Web document accessed 04.07.2022. https://www.w3schools.com/html

27 CSS Tutorial. Web document accessed 04.07.2022. https://www.w3schools.com/css

28 Web Content. Web document accessed 04.07.2022. https://www.techopedia.com/definition/23885/web-content

29 Using dynamic styling information. Web document accessed 04.07.2022. https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model/Using_dynamic_styling_information

30 Painful JavaScript syntax features. Web document accessed 04.07.2022. https://ofstack.com/javascript/4507/painful-javascript-syntax-features.html

31 Asynchronous. Web document accessed 04.07.2022. https://developer.mozilla.org/en-US/docs/Glossary/Asynchronous

32 Ajax: A New Approach to Web Applications. Web document accessed 04.07.2022. https://immagic.com/eLibrary/ARCHIVES/GENERAL/ADTVPATH/A050218G.pdf

33 Ajax Security Issues. Web document accessed 04.07.2022. https://resources.infosecinstitute.com/topic/ajax-security-issues

34 The Most Popular JavaScript Frameworks – 2011/2021. Web document accessed 04.07.2022.
https://statisticsanddata.org/data/the-most-popular-javascript-frameworks-2011-2021

35 Distributed Systems Concepts and Design Fifth Edition, George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, 2012.

36 NoSQL Databases: An Overview. Web document accessed 31.07.2022.
https://www.thoughtworks.com/insights/blog/nosql-databases-overview

37 ACIDRain: concurrency-related attacks on database backed web applications. Web document accessed 31.07.2022.
https://blog.acolyer.org/2017/08/07/acidrain-concurrency-related-attacks-on-database-backed-web-applications/

38 NoSQL Databases: Aggregated DBs. Web document accessed 31.07.2022.
http://www.diag.uniroma1.it//~rosati/dmds-1617/aggregated-databases.pdf

39 In-Memory Databases Explained. Web document accessed 31.07.2022.
https://www.mongodb.com/databases/in-memory-database

40 Introduction to MongoDB. Web document accessed 31.07.2022.
https://www.mongodb.com/docs/manual/introduction/

41 Elegant mongodb object modeling for node.js. Web document accessed 31.07.2022.
https://mongoosejs.com/

42 Express Tutorial Part 3: Using a Database (with Mongoose). Web document accessed 31.07.2022.
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

43 Getting Started with MongoDB & Mongoose. Web document accessed 31.07.2022.
https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/

44 Express. Web document accessed 25.10.2022.
https://www.npmjs.com/package/express

45 Express Tutorial Part 3: Using a Database (with Mongoose). Web document accessed 25.10.2022.
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose

46 Express API reference. Web document accessed 25.10.2022.
https://expressjs.com/en/4x/api.html

47 ExpressJS Request & Response. Web document accessed 25.10.2022. https://www.pabbly.com/tutorials/express-js-request-response/

48 React JS. Web document accessed 25.10.2022. https://reactjs.org/

49 Managing DOM components with ReactDOM. Web document accessed 25.10.2022. https://blog.logrocket.com/managing-dom-components-reactdom/

50 How and when to force a React component to re-render. Web document accessed 25.10.2022. https://blog.logrocket.com/how-when-to-force-react-component-re-render/

51 React Top-Level API. Web document accessed 25.10.2022. https://reactjs.org/docs/react-api.html

52 Simple in-memory cache in Node.js. Web document accessed 25.10.2022. https://dev.to/franciscomendes10866/simple-in-memory-cache-in-node-js-gl4

53 Caching In Node.js Applications. Web document accessed 25.10.2022. https://www.honeybadger.io/blog/nodejs-caching/

54 Fasten your Node JS Application with a Powerful Caching Mechanism using Redis. Web document accessed 25.10.2022. https://medium.com/geekculture/fasten-your-node-js-application-with-a-powerful-caching-mechanism-using-redis-fd76b8aa482f

55 The Input element from Form Input. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

56 HTML Living Standard. Web document accessed 25.10.2022. https://html.spec.whatwg.org/multipage/input.html

57 Validator and sanitizer functions of express.js middlewares. Web document accessed 25.10.2022. https://express-validator.github.io/docs/

58 Web Authentication API. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API

59 Credential Management. Web document accessed 25.10.2022. https://www.w3.org/TR/credential-management-1/

60 Introduction to JSON Web Tokens. Web document accessed 25.10.2022. https://jwt.io/introduction

61 Web Authentication: An API for accessing Public Key Credentials. Web document accessed 25.10.2022.
https://www.w3.org/TR/webauthn-2/

62 About WebAuthn. Web document accessed 25.10.2022.
https://webauthn.guide/

63 Blocking Brute Force Attacks. Web document accessed 25.10.2022.
https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

64 Understanding Denial of Service and Brute Force Attacks. Web document accessed 25.10.2022.
https://blog.sucuri.net/2014/03/understanding-denial-of-service-and-brute-force-attacks-wordpress-joomla-drupal-vbulletin.html

65 Role-Based Access Control. Web document accessed 25.10.2022.
https://www.mongodb.com/docs/manual/core/authorization/

66 Access Token Scope of OAuth 2.0 protocol. Web document accessed 25.10.2022.
https://www.rfc-editor.org/rfc/rfc6749#section-3.3

67 Eeshan Gupta, Jaideep Vaidya, Shamik Sura, Vijayalakshmi Atluri, 2021. Attribute-Based Access Control for NoSQL Databases. Web document accessed 25.10.2022.
https://dl.acm.org/doi/pdf/10.1145/3422337.3450323

68 David Farley, 2021. Modern Software Engineering: Doing What Works to Build Better Software Faster.

69 The Model View Controller Pattern – MVC Architecture and Frameworks Explained. Web document accessed 25.10.2022.
https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

70 Middleware Definition. Web document accessed 25.10.2022.
https://www.techtarget.com/searchapparchitecture/definition/middleware

71 Using middleware. Web document accessed 25.10.2022.
https://expressjs.com/en/guide/using-middleware.html

72 A highly scalable, professional boilerplate for building fast, robust, and adaptable mern web apps. Web document accessed 25.10.2022.
https://github.com/getspooky/cookiescript

73 Creating a Simple MERN Fullstack Application. Web document accessed 25.10.2022.
https://niruhan.medium.com/creating-a-simple-mern-fullstack-application-2cbcfbdf3940

74 Popular React toolchains: Create a New React App. Web document accessed 25.10.2022. https://reactjs.org/docs/create-a-new-react-app.html

75 Usage statistics of client-side programming languages for websites. Web document accessed 25.10.2022. https://w3techs.com/technologies/overview/client_side_language

76 Gabor Karsa, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, Steven Völkel, 2009. Design Guidelines for Domain Specific Languages. Web document accessed 25.10.2022. https://www.se-rwth.de/publications/Design-Guidelines-for-Domain-Specific-Languages.pdf

77 Framework main features. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Main_features

78 Usage statistics of server-side programming languages for websites. Web document accessed 25.10.2022. https://w3techs.com/technologies/overview/programming_language

79 The Scala Programming Language. Web document accessed 25.10.2022. https://www.scala-lang.org/

80 How Tech Giants Use Scala. Web document accessed 25.10.2022. https://sysgears.com/articles/how-tech-giants-use-scala/

81 Interpreted vs Compiled Programming Languages: What's the Difference?. Web document accessed 25.10.2022. https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/

82 Node.js server without a framework. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Node_server_without_framework

83 Properly configuring server MIME types. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Configuring_server_MIME_types

84 Using OAuth 2.0 to Access Google APIs. Web document accessed 25.10.2022. https://developers.google.com/identity/protocols/oauth2

85 OAuth 2.0 Authorization Framework. Web document accessed 25.10.2022. https://auth0.com/docs/authenticate/protocols/oauth

86 Antonio Sanso and Justin Richer, 2017. OAuth 2 in Action.

87 Thecoopercodes, 2022. Google Identity Services Login with React (2022 React Google Login) https://www.youtube.com/watch?v=roxC8SMs7HU

88 Network traffic analysis for IR: Data collection and monitoring. Web document accessed 25.10.2022. https://resources.infosecinstitute.com/topic/network-traffic-analysis-for-ir-data-collection-and-monitoring/

89 Examples of TCP and UDP in Real Life. Web document accessed 25.10.2022. https://www.geeksforgeeks.org/examples-of-tcp-and-udp-in-real-life/

90 HTTP status of Unauthorized. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/401

91 HTTP headers. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

92 HTTP Messages. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

93 XHR (XMLHttpRequest). Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Glossary/XHR_(XMLHttpRequest)

94 Introduction to fetch data. Web document accessed 25.10.2022. https://web.dev/introduction-to-fetch/

95 The HTTP 204 No Content. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/204

96 The HTTP PUT request method. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT

97 A new approach to embedded scripting. Web document accessed 25.10.2022. https://mongoose-os.com/blog/mjs-a-new-approach-to-embedded-scripting/

98 The Web Open Font Format (WOFF). Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/Guide/WOFF

99 Using manifest.json. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json

100 Pinterest releases new iPad and Android apps. Web document accessed 25.10.2022. https://edition.cnn.com/2012/08/15/tech/mobile/pinterest-mobile-apps-android-ipad/index.html

101 Why You Should Never Use Pure Black for Text or Backgrounds. Web document accessed 25.10.2022. https://uxmovement.com/content/why-you-should-never-use-pure-black-for-text-or-backgrounds/

102 The display mode of media feature. Web document accessed 25.10.2022. https://developer.mozilla.org/en-US/docs/Web/Manifest/display

103 ScreenOrientation in Android Application. Web document accessed 25.10.2022. https://developer.android.com/reference/androidx/browser/trusted/ScreenOrientation

104 URL Handler usage. Web document accessed 25.10.2022. https://wcm.io/handler/url/usage.html

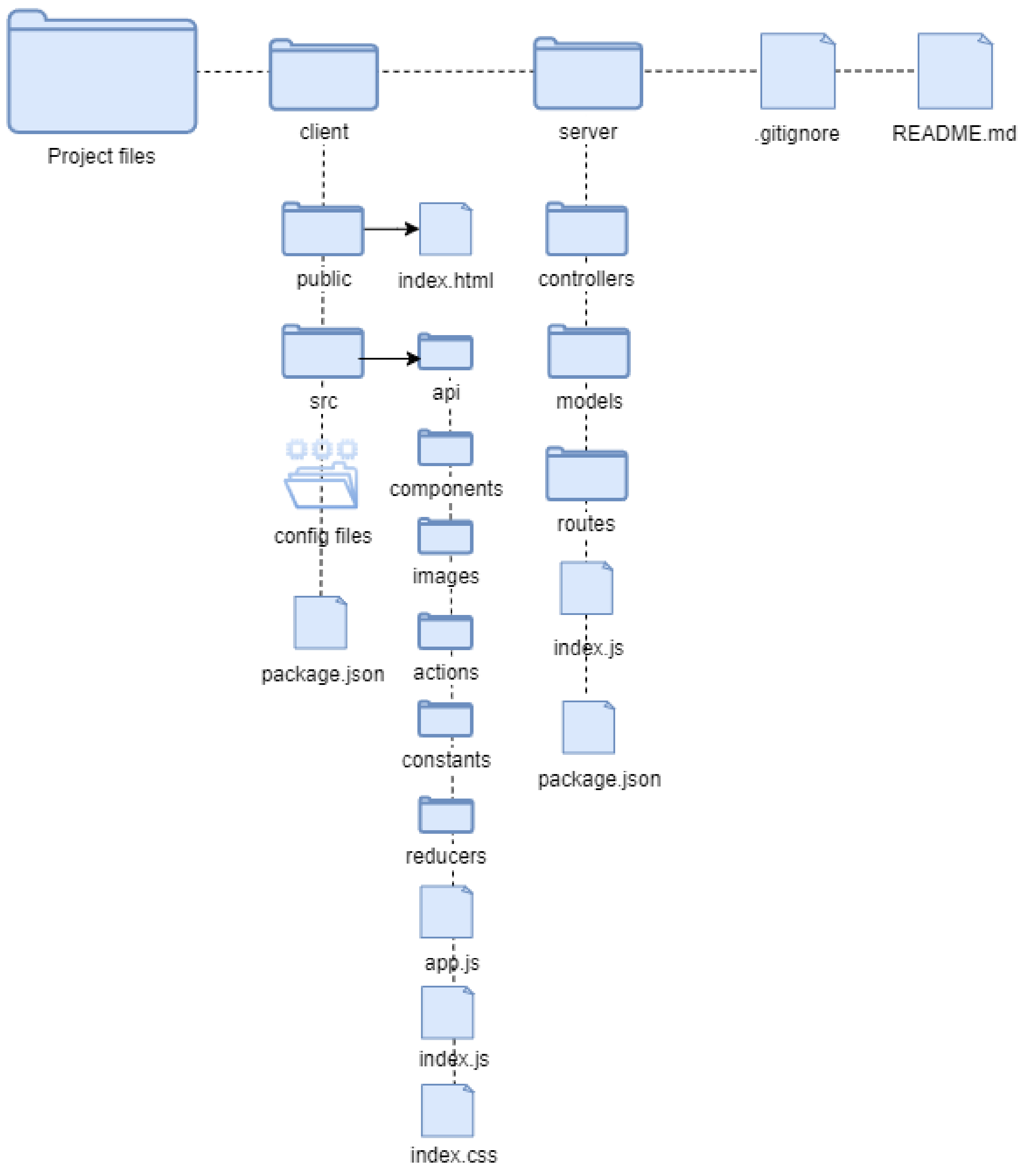
105 Introduction to HTTPS. Web document accessed 25.10.2022. https://https.cio.gov/faq/

106 Gary McLean Hall, 2017. Adaptive code: Agile coding with design patterns and SOLID principle Second Edition.

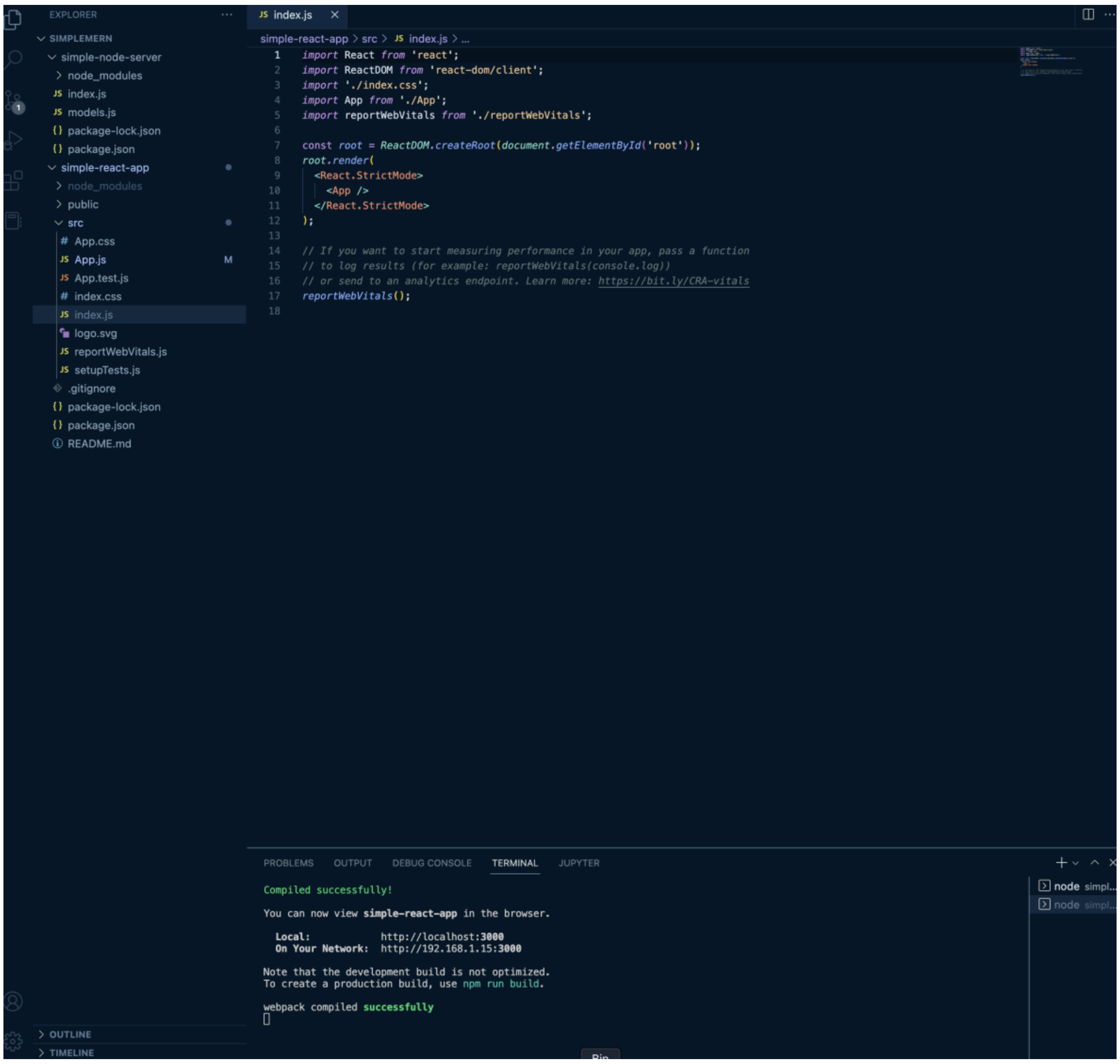
107 Richard F.Schmidt, 2013. Software Engineering: Architecture-Driven Software Development.

Appendices

Appendix 1: MVC folder structure

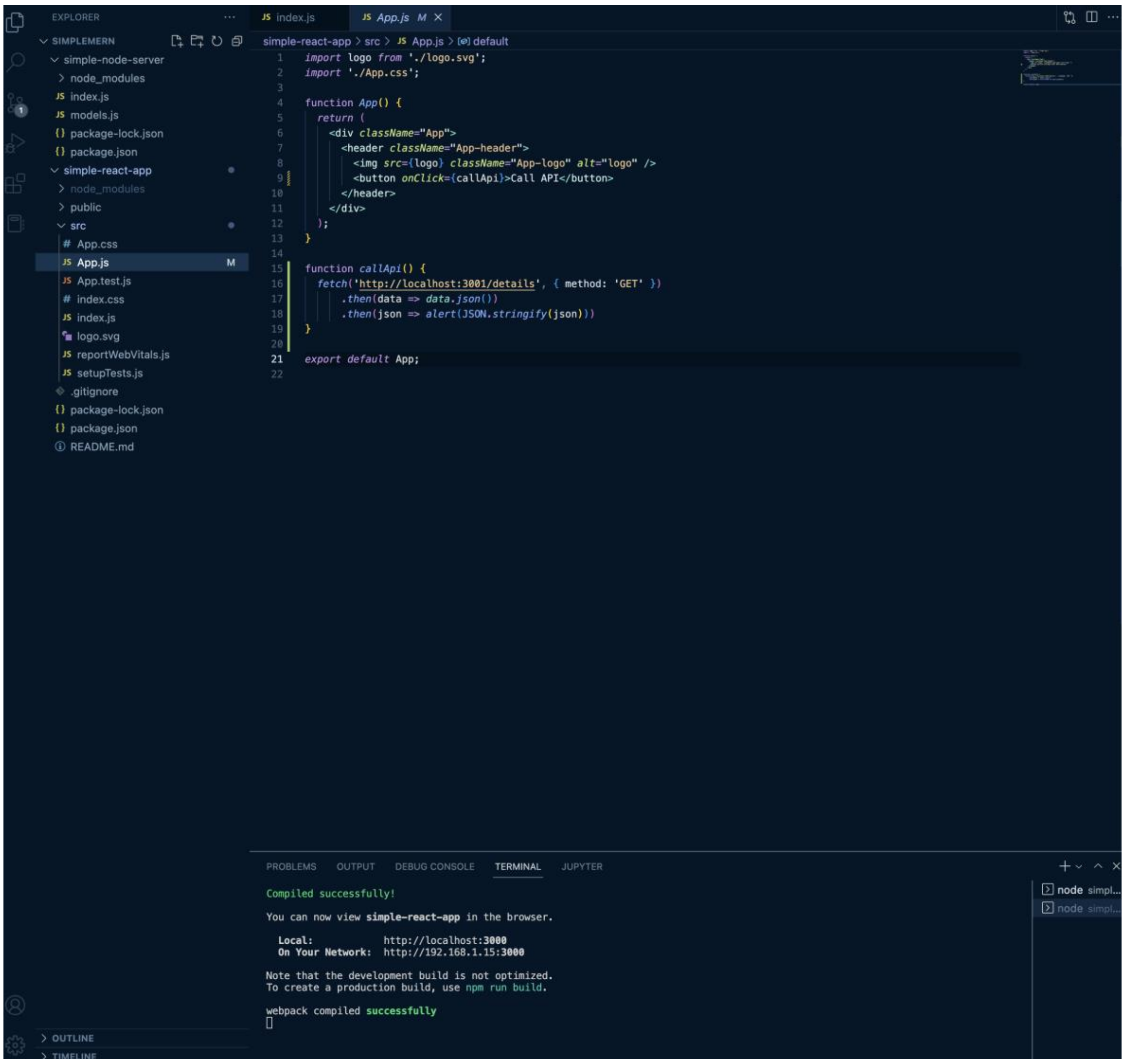


Appendix 2: MERN stacks Boilerplate Template



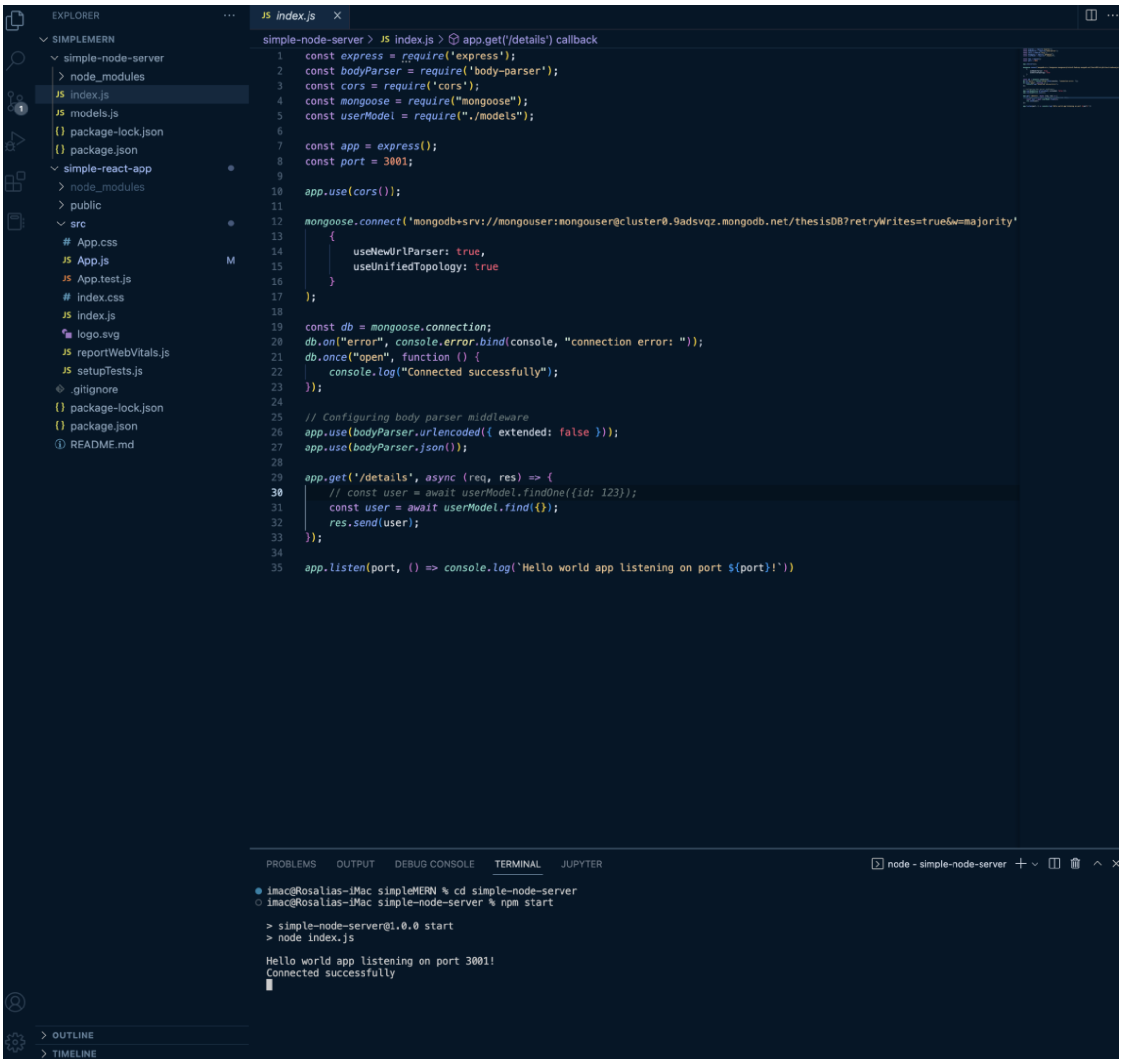
```javascript
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

```
Compiled successfully!

You can now view simple-react-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.15:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



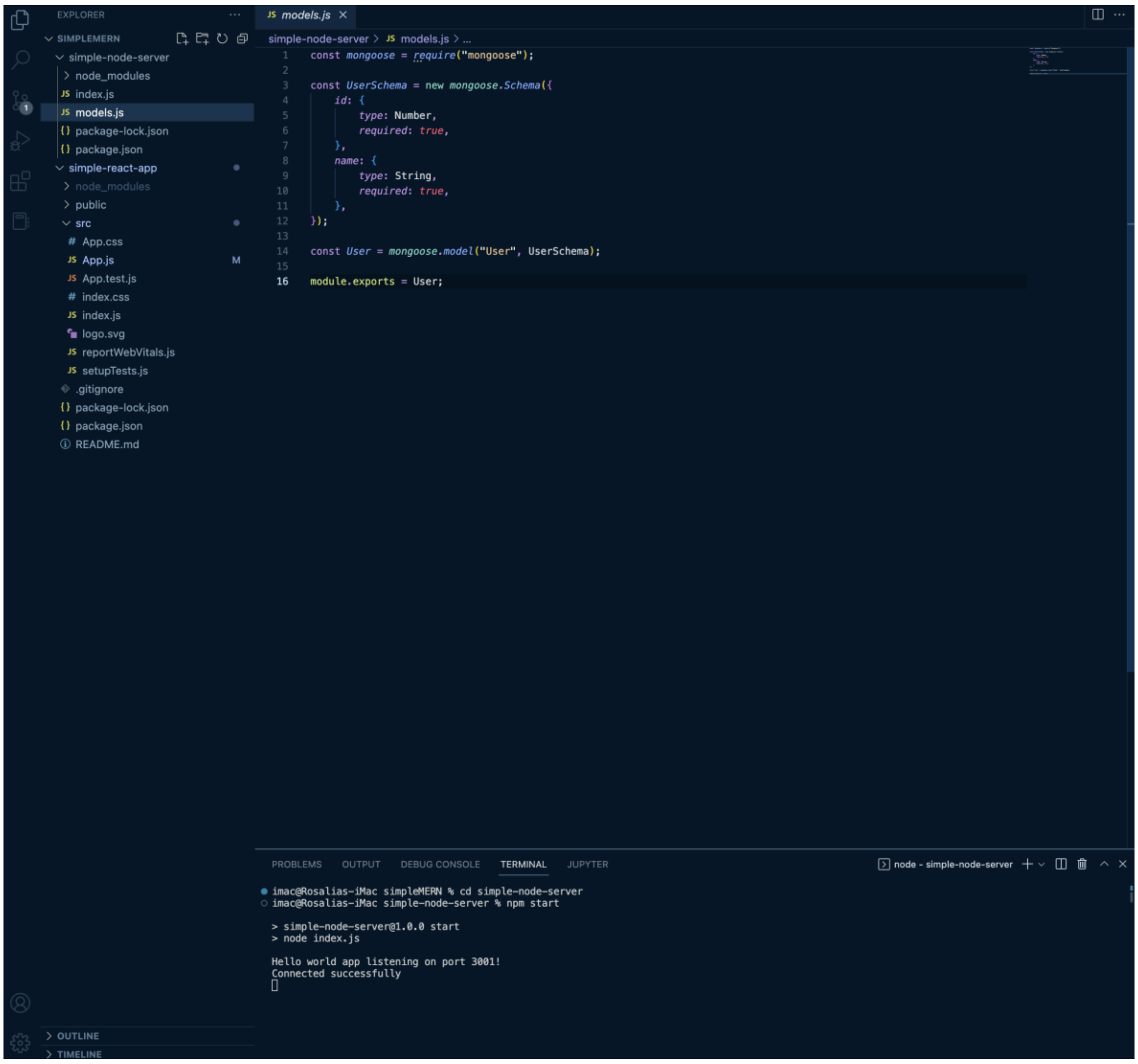
```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <button onClick={callApi}>Call API</button>
      </header>
    </div>
  );
}

function callApi() {
  fetch('http://localhost:3001/details', { method: 'GET' })
      .then(data => data.json())
      .then(json => alert(JSON.stringify(json)))
}

export default App;
```

```
Compiled successfully!

You can now view simple-react-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.15:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



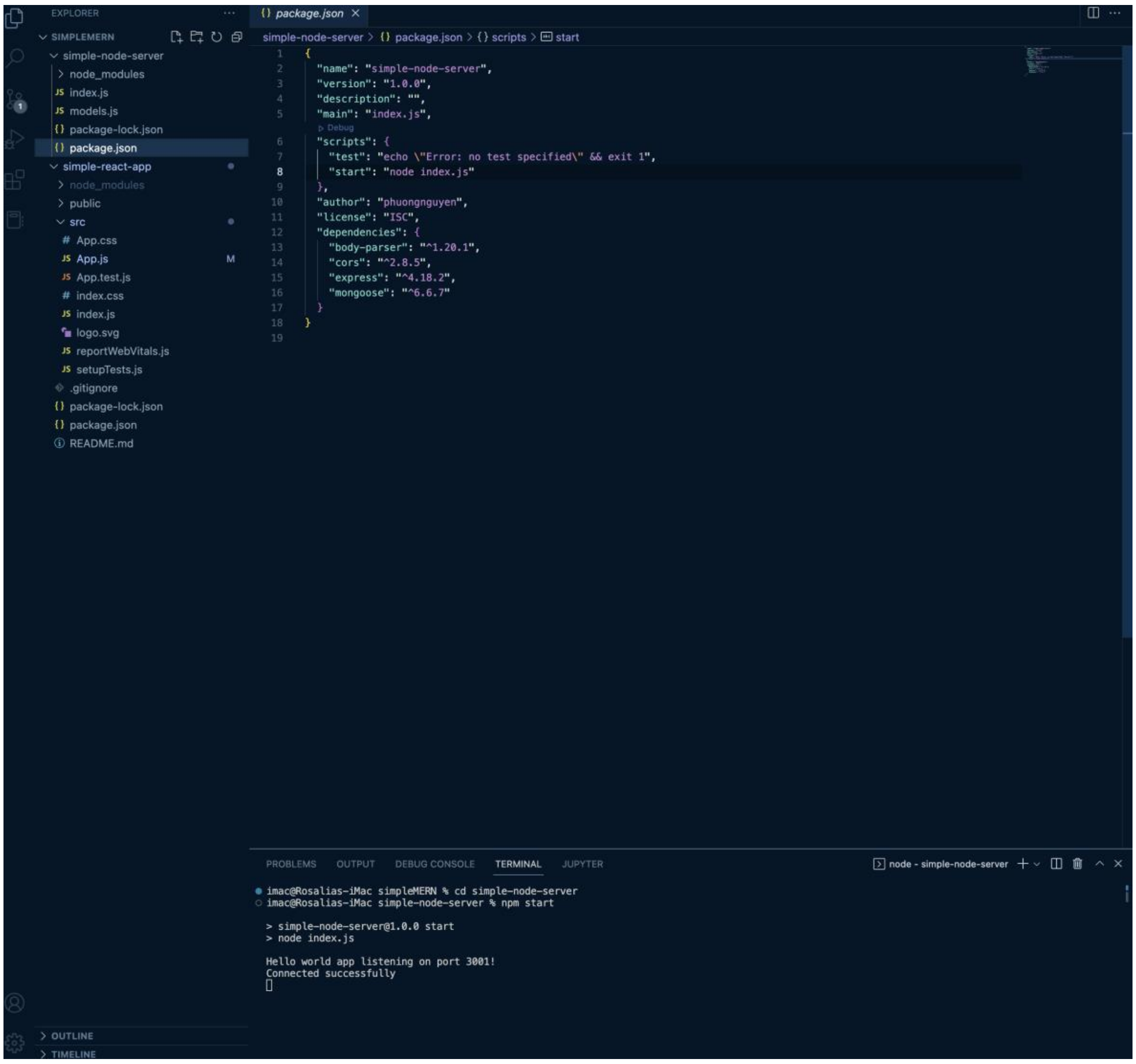
```javascript
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const mongoose = require("mongoose");
const userModel = require("./models");

const app = express();
const port = 3001;

app.use(cors());

mongoose.connect('mongodb+srv://mongouser:mongouser@cluster0.9adsvqz.mongodb.net/thesisDB?retryWrites=true&w=majority'
    {
        useNewUrlParser: true,
        useUnifiedTopology: true
    }
);

const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error: "));
db.once("open", function () {
    console.log("Connected successfully");
});

// Configuring body parser middleware
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.get('/details', async (req, res) => {
    // const user = await userModel.findOne({id: 123});
    const user = await userModel.find({});
    res.send(user);
});

app.listen(port, () => console.log(`Hello world app listening on port ${port}!`))
```

```
imac@Rosalias-iMac simpleMERN % cd simple-node-server
imac@Rosalias-iMac simple-node-server % npm start

> simple-node-server@1.0.0 start
> node index.js

Hello world app listening on port 3001!
Connected successfully
```



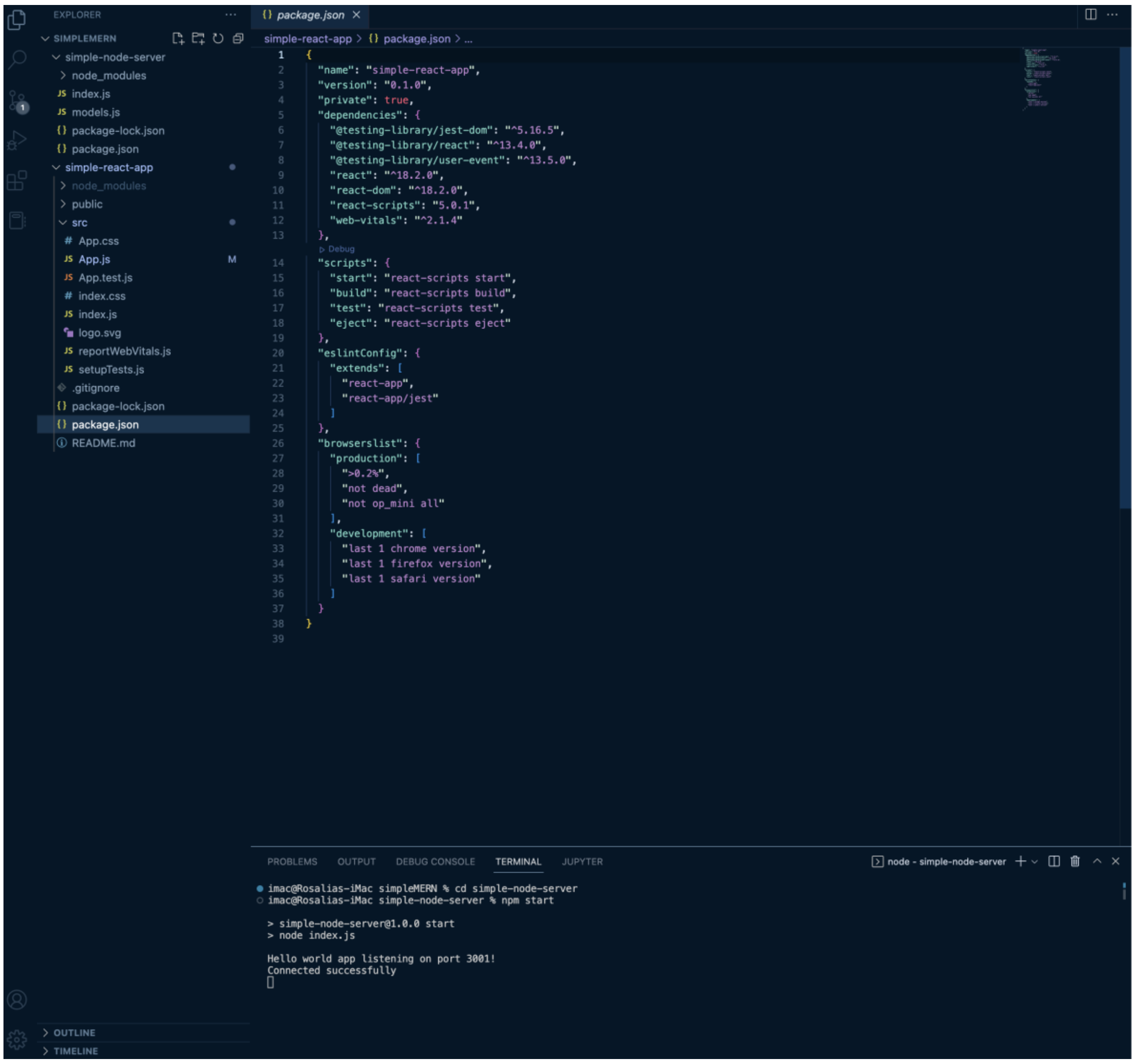
```
const mongoose = require("mongoose");

const UserSchema = new mongoose.Schema({
    id: {
        type: Number,
        required: true,
    },
    name: {
        type: String,
        required: true,
    },
});

const User = mongoose.model("User", UserSchema);

module.exports = User;
```

```
imac@Rosalias-iMac simpleMERN % cd simple-node-server
imac@Rosalias-iMac simple-node-server % npm start

> simple-node-server@1.0.0 start
> node index.js

Hello world app listening on port 3001!
Connected successfully
```



```json
{
  "name": "simple-node-server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
  "author": "phuongnguyen",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.1",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "mongoose": "^6.6.7"
  }
}
```

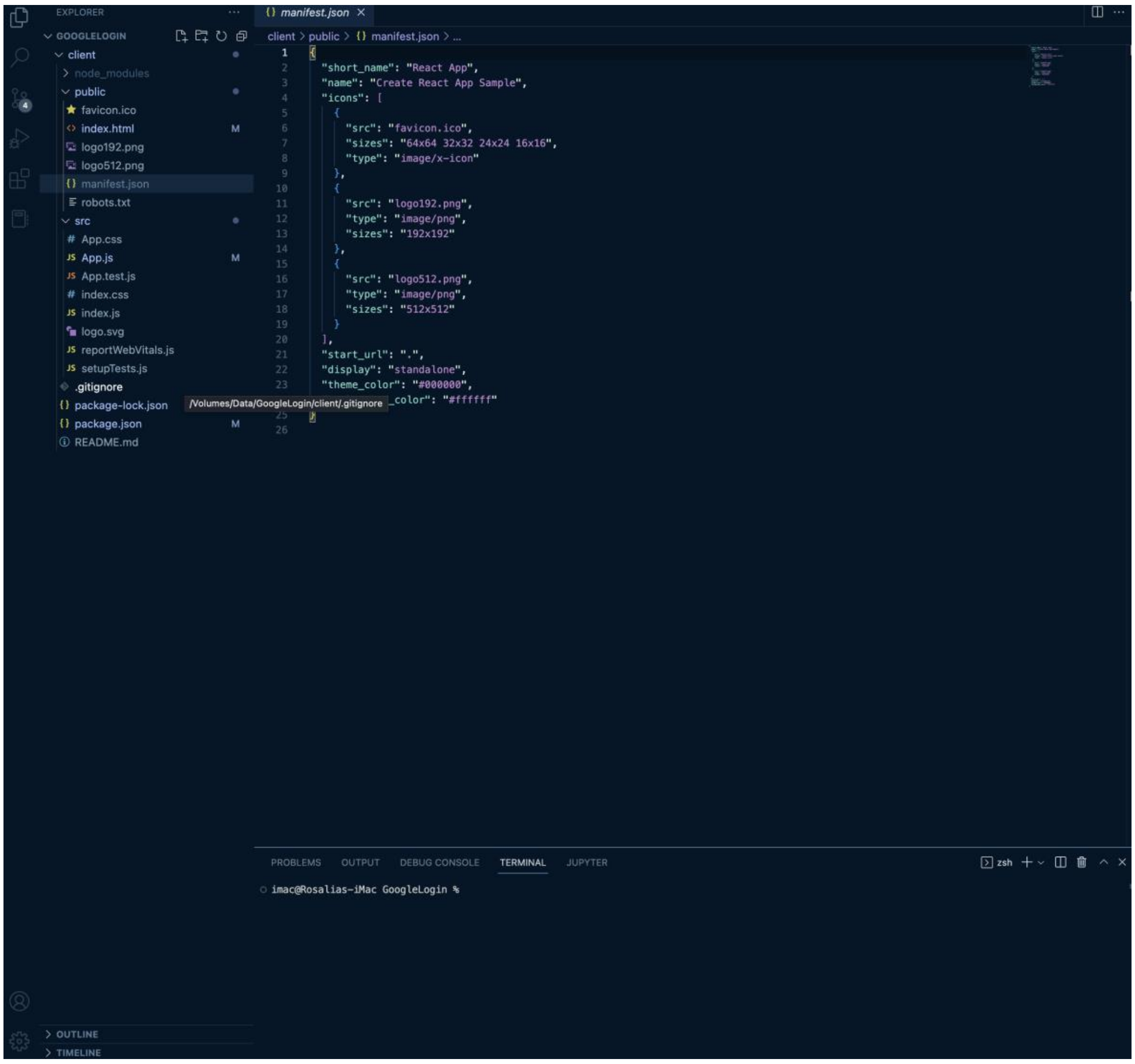
```
imac@Rosalias-iMac simpleMERN % cd simple-node-server
imac@Rosalias-iMac simple-node-server % npm start

> simple-node-server@1.0.0 start
> node index.js

Hello world app listening on port 3001!
Connected successfully
```



```json
{
  "name": "simple-react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

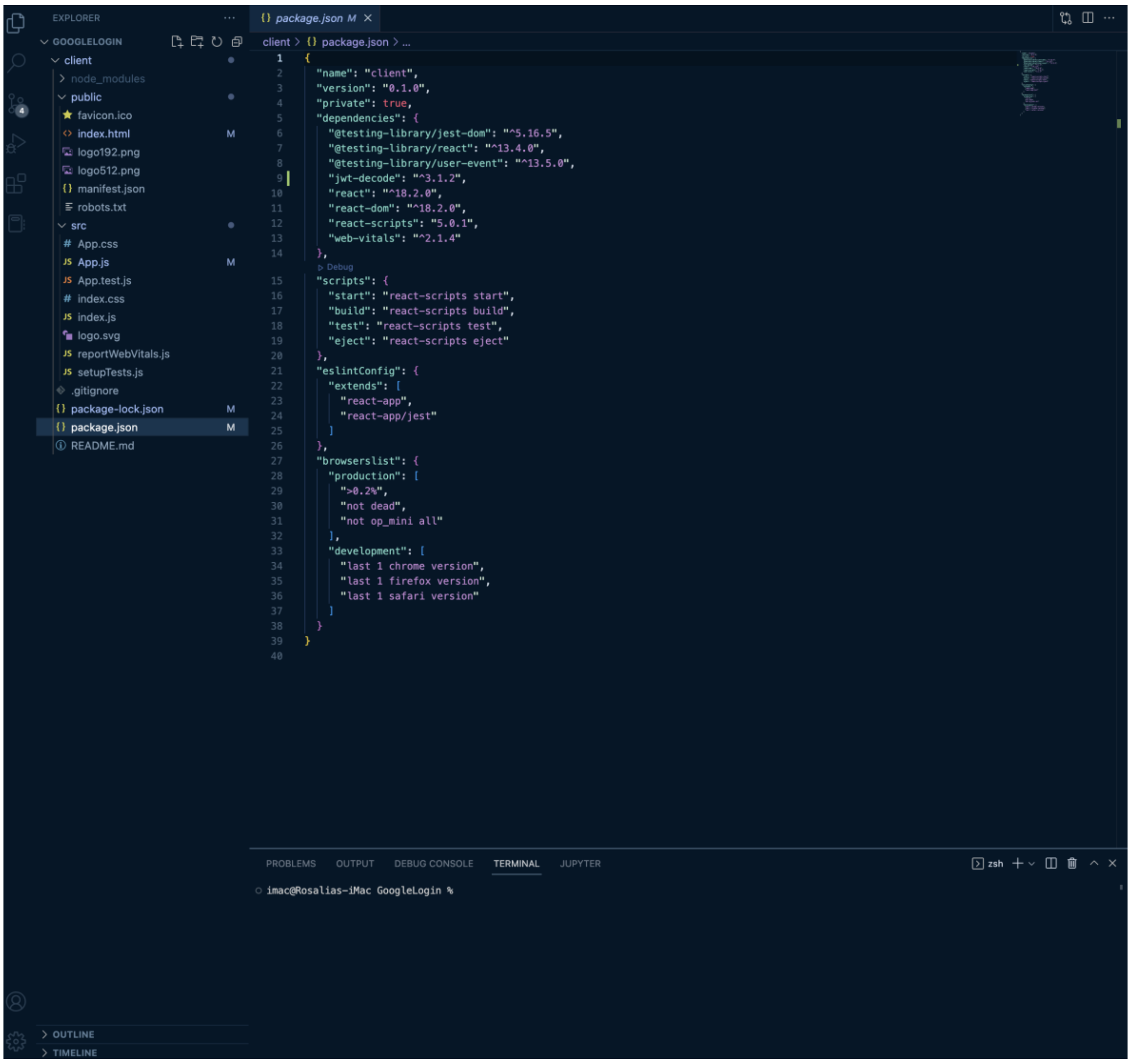
```
imac@Rosalias-iMac simpleMERN % cd simple-node-server
imac@Rosalias-iMac simple-node-server % npm start

> simple-node-server@1.0.0 start
> node index.js

Hello world app listening on port 3001!
Connected successfully
```

Appendix 3: Google Sign in/Sign out Button



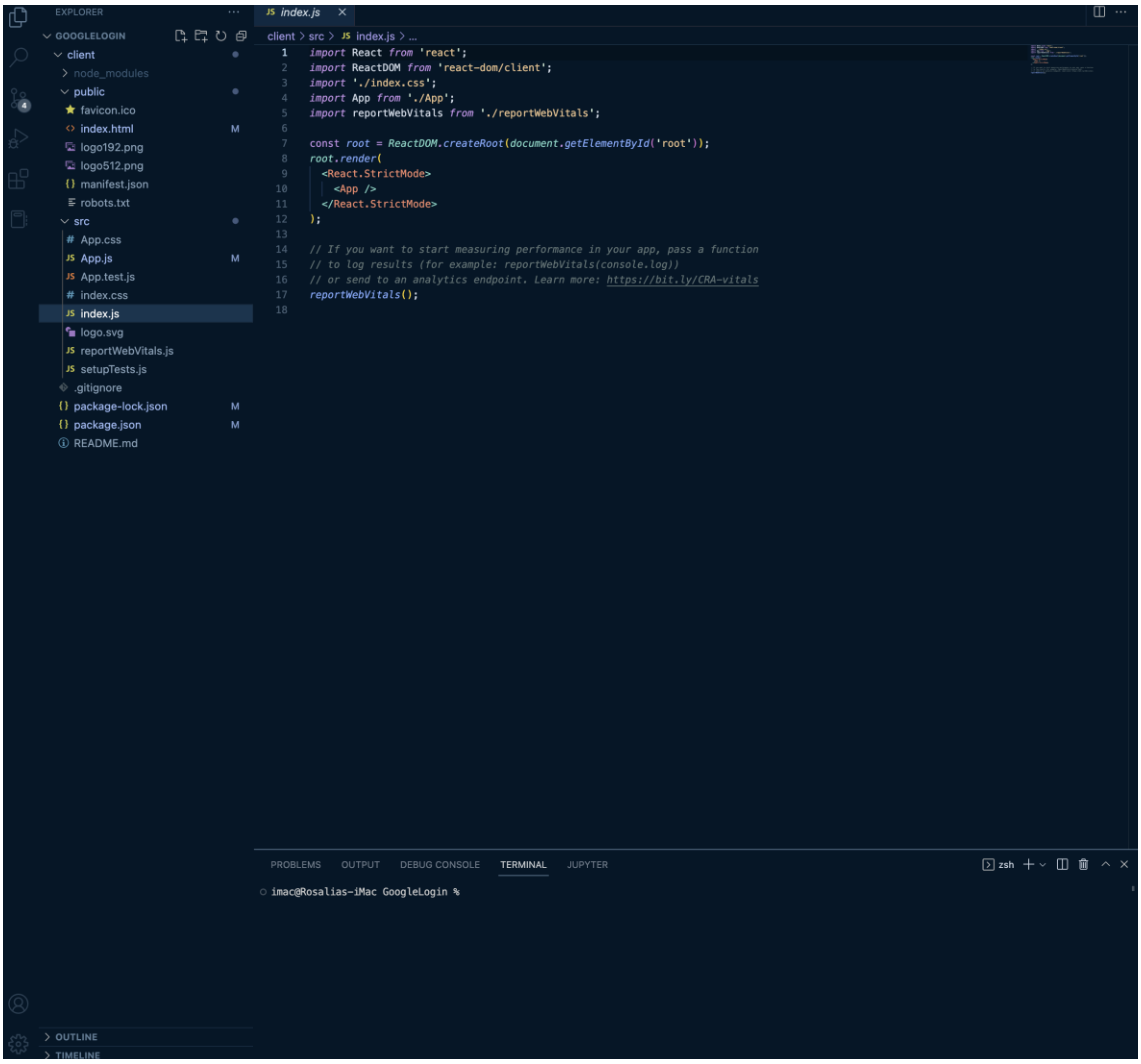
```json
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
            _color": "#ffffff"
}
```

```
imac@Rosalias-iMac GoogleLogin %
```



```json
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "jwt-decode": "^3.1.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

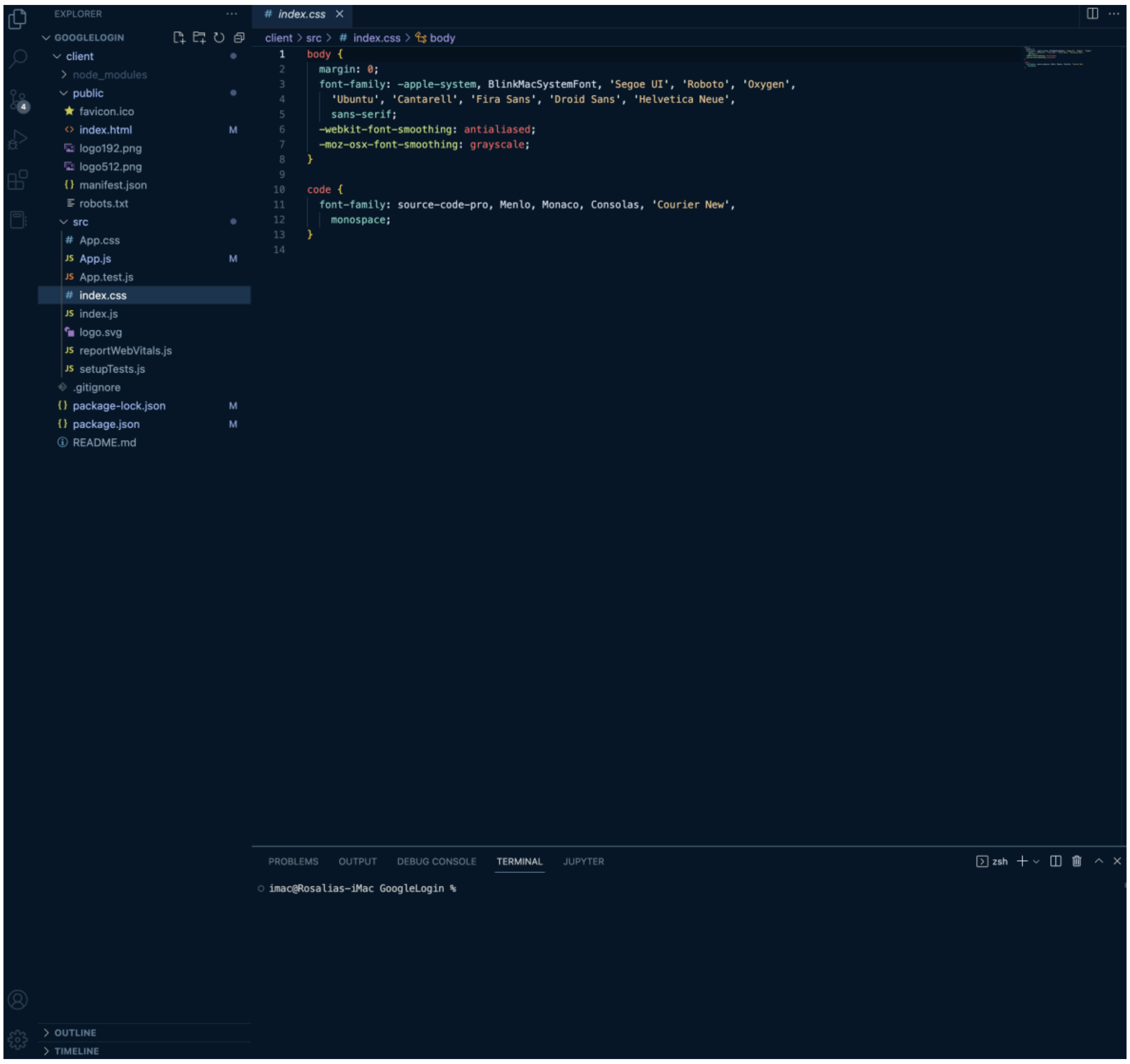
```
imac@Rosalias-iMac GoogleLogin %
```



```
client
  node_modules
  public
    favicon.ico
    index.html        M
    logo192.png
    logo512.png
    manifest.json
    robots.txt
  src
    App.css
    App.js            M
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json   M
  package.json        M
  README.md
```

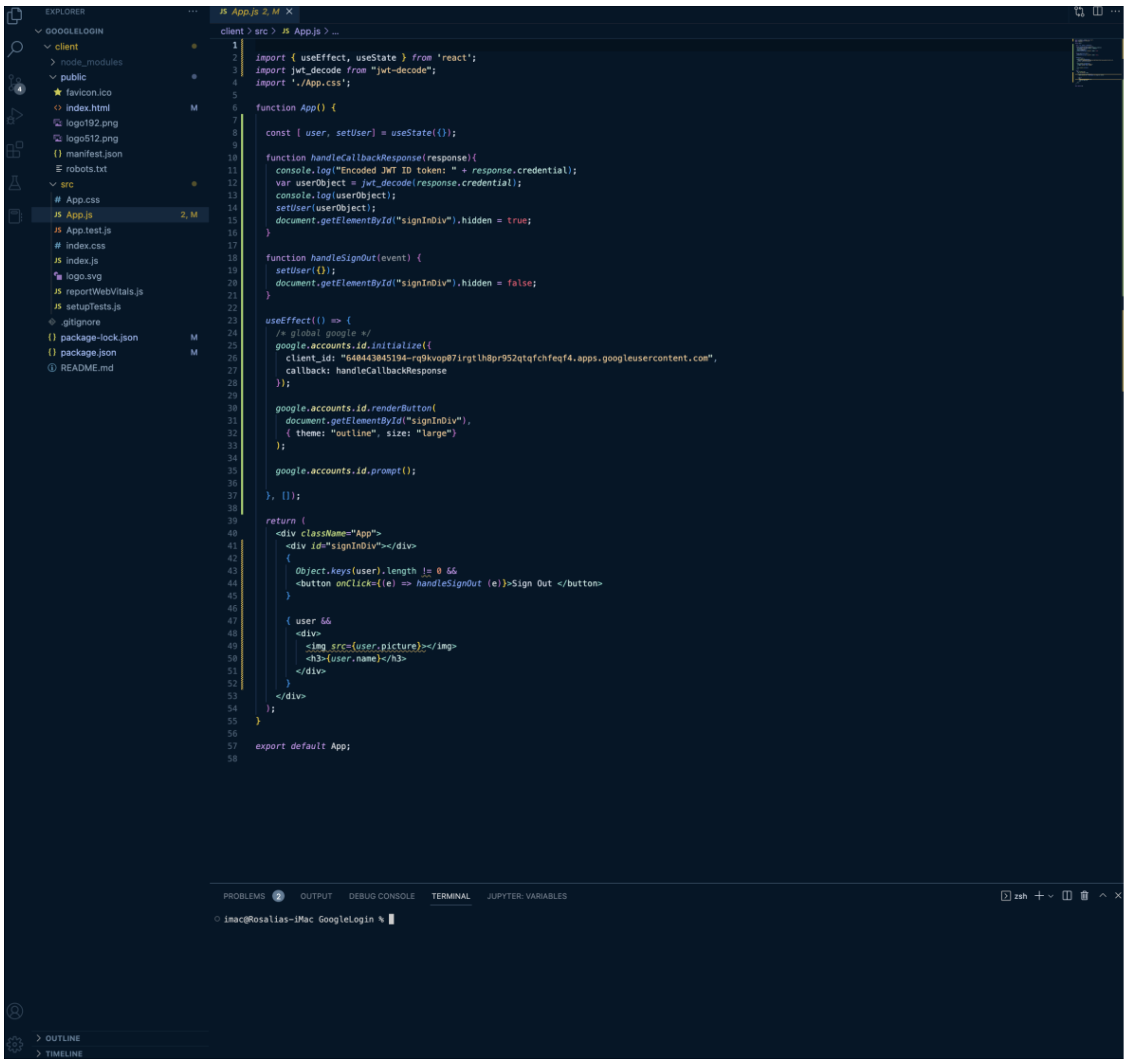
```js
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

```
imac@Rosalias-iMac GoogleLogin %
```



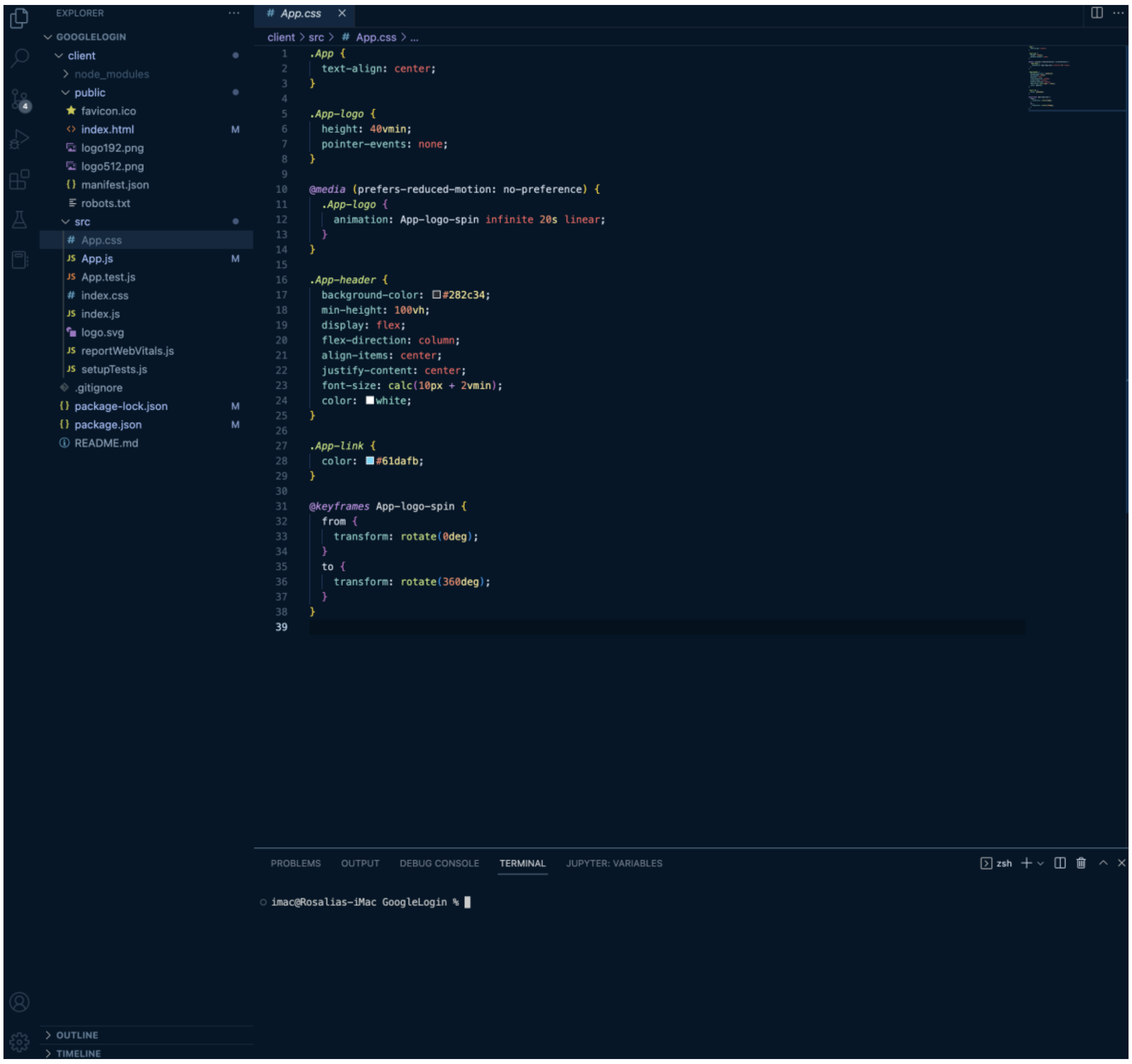
```css
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

```
imac@Rosalias-iMac GoogleLogin %
```



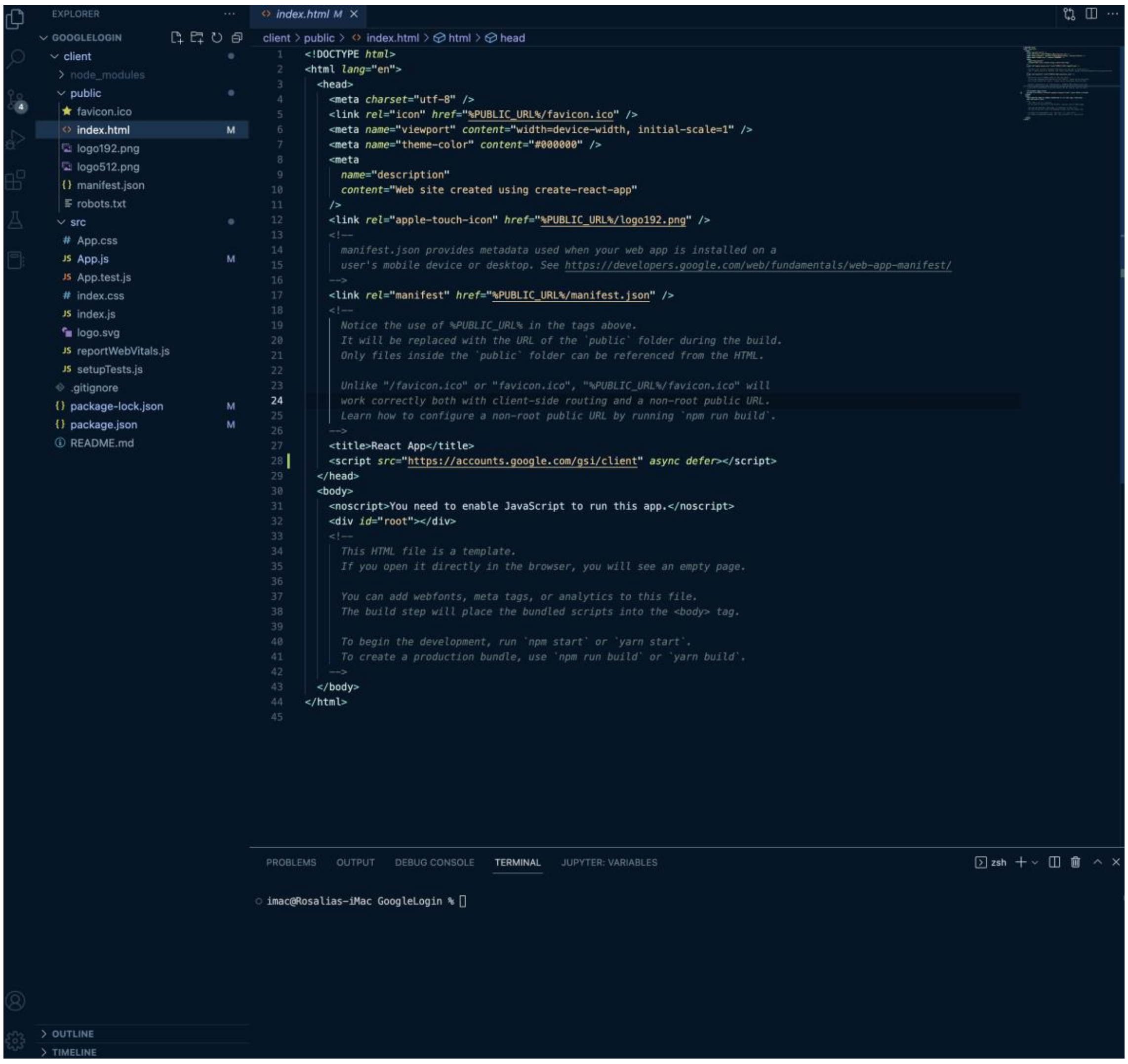
```

import { useEffect, useState } from 'react';
import jwt_decode from "jwt-decode";
import './App.css';

function App() {

  const [ user, setUser] = useState({});

  function handleCallbackResponse(response){
    console.log("Encoded JWT ID token: " + response.credential);
    var userObject = jwt_decode(response.credential);
    console.log(userObject);
    setUser(userObject);
    document.getElementById("signInDiv").hidden = true;
  }

  function handleSignOut(event) {
    setUser({});
    document.getElementById("signInDiv").hidden = false;
  }

  useEffect(() => {
    /* global google */
    google.accounts.id.initialize({
      client_id: "640443045194-rq9kvop07irgtlh8pr952qtqfchfeqf4.apps.googleusercontent.com",
      callback: handleCallbackResponse
    });

    google.accounts.id.renderButton(
      document.getElementById("signInDiv"),
      { theme: "outline", size: "large"}
    );

    google.accounts.id.prompt();

  }, []);

  return (
    <div className="App">
      <div id="signInDiv"></div>
      {
        Object.keys(user).length != 0 &&
        <button onClick={(e) => handleSignOut (e)}>Sign Out </button>
      }

      { user &&
        <div>
          <img src={user.picture}></img>
          <h3>{user.name}</h3>
        </div>
      }
    </div>
  );
}

export default App;

```

```
imac@Rosalias-iMac GoogleLogin % 
```



```css
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

```
imac@Rosalias-iMac GoogleLogin %
```



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
    <script src="https://accounts.google.com/gsi/client" async defer></script>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

```
imac@Rosalias-iMac GoogleLogin % 
```