304CEM - Web API Development - 2021SEPJAN

URL link to Back end repository - https://github.coventry.ac.uk/304CEM-2021SEPJAN/304cem_back_end.git

URL link to fron end repository - https://github.coventry.ac.uk/304CEM-2021SEPJAN/304cem_front_end.git

Video to Link - https://web.microsoftstream.com/video/b5bbd712-9e6b-42d0-bb39-2b48c091184e

The Benefits Of Full Stack For Application Development

In application development, we refer to full stack as an entire system that comprises of different components communicating with each other to make a website or system work. These components are front end, back end, API and Databases.

Front end: - also known as client-side, where the developer uses front end languages to develop a web application for the users to interact with the system.
There are various frameworks or libraries which render down flat Html tags and CSS styles if a developer does not wish to write code from scratch or use controls that provide more functionality e.g., React, bootstrap, angular, Ant design, and jQuery. Whichever framework we use the result will be in Html CSS and JavaScript which the browser understands.

Backend: - also known as server-side. the code the user does not see; it runs on the server or API. It is the place where the business logic of the web application is written. The back-end system receives the request from the front-end system and processes it and sends the appropriate responses back to the user. frameworks and languages that can be used are Nodejs, ASP.net.

Databases: - Performing read and write operations on the data stored in the database. e.g., MySQL, SQLite, or MongoDB.

A developer who develops all of these components back end, front end and database is called a full stack developer. JavaScript is one of the most popular and powerful languages being used for web development today 67.8% (Stack Overflow, 2019).
Popular stacks like mean, Django, lamp ,lemp all use JavaScript for all their components.

Benefits of full stack for application development using JavaScript.

- *Uniform Language across all components* : - when developing a web application, we could use the same language across different components e.g. Using JavaScript for React (front end) and NodeJS (back end).Switching between them becomes much simpler and easier for the developer. Which make development process much faster and efficient (Altexsoft, 2018).

- *Code-Reusability* : - When we use JavaScript across the front and back end systems, we can save a lot of our time by re-using the code across them where we feel the logic or implementation required are the same. This also helps us during refactoring and maintaining Our code later during the process.

- *Open Source* : - When using JavaScript full stack development most of the frameworks, libraries or packages we use are open source. Most of these tools are developed and maintained by tech giants and contributed from all over the world e.g., React by Facebook. Using an open-source tool is much cheaper than a commercial which many start-ups or business considers when working on a project which has a limited budget (Altexsoft, 2018).

- *Greater speed and Performance* : - Server-side frameworks like Nodejs using JavaScript use an event driven non-blocking IO model which provide greater speed and efficiency when compared to other language's like java. PayPal has stated they have shifted from java to full stack JavaScript after which their development process has become 2 times faster, and response time has decreased by 35% for the same pages (PayPal Engineering, 2013).

- *Huge Community of developers* : - Using a full stack JavaScript architecture for development process would also be beneficial as there is a huge community of developer's who are using it, which indicates there is a lot of information present all over the web and help is more easily available.

Benefits for a full stack developer.

- *Multitasking* - having an in-depth knowledge of different components within an architecture gives them the ability to perform various complex tasks In development processes. This can be from UI designing , prototyping , database normalization , server side and front side development.
- *Adaptability* - more adaptable to work with technologies that aren't in their domain, this is also beneficial for the organization as they do not have to hire developers with a specific skill set.
- *Employability* - trends show that full stack developer is more employed even paid more, as organization are looking for people who can do more than just one role, since multiple roles cost more to the company (Nikita Duggal, 2019).

How The JS Full Stack Ecosystem Is Developing At Its Cutting Edge.

In the earlier days a JavaScript developer meant that you would possess skills to develop only the front-end side. Their primarily function was to add behaviours and functionality to your website. Those days are now gone and today JavaScript is even used on the Server side (Nodejs) and other components.

The JavaScript full stack ecosystem is a whole collection of frameworks and libraries that assist us in the process of development, documentation, and testing etc. These libraries are developed by big tech giants, developers and contributors from all over the world e.g., React by Facebook , Angular By Google (Altexsoft, 2018).

JavaScript ecosystem is cutting edge because of its various tools for the entire development process. These open-source tools which provide several different functionalities to the developers, for their different components(PC, mobile iOS or android, Single web application) all are under one roof. Development tools, testing and debugging, Package managers and deployment tools are a part of this ecosystem.
This is what makes it so famous and loved among developers.

Popular Frameworks for the web - Backbone.js, Angular JS, Ember, React JS.
JavaScript Framework for the mobile - React Native, Kendo UI, Ionic.

Full stack Js tool - A combination of different tools is used to create a whole modern web application. 4 of these most popular frameworks is mongo dB, express, Angular, NodeJS also known as the “mean” stack.

- Mongo dB - No sql database. In this we see a database of documents rather than table , rows or columns. Main advantage the data is stored in json format which is perfect when writing in JavaScript.
- Express – is a NodeJS library which is a simple web server framework, which makes hosting your website on a local server much simpler and testing your routes, serving static files and cookie management.
- Angular – an MVC JavaScript framework to make single web page application.
- Nodejs – used to build the server, uses Google v8 engine.

the key advantage of using the MEAN stack, is that we could use JavaScript at all the levels, this allows code reusability across different components, using just one language also saves time as the developer wouldn’t have to search for syntax to use.

Mobile development - As mentioned above, JavaScript was first only used for front end development but now it has progressed to server side and also mobile development (Altexsoft, 2018).

- Meteor – if you want to make apps for PC or mobile(iOS or android) we could use this framework ; it is an isomorphic JavaScript framework which is written using Nodejs.it allows us to build UI.
- React Native – this is regarded as one of the best libraries to use when working for mobile development. It does not use the standard html or CSS but has prebuilt classes for CSS. We can use react native to develop on platforms such as Linux or windows.
- Kendo UI – this is a hybrid framework used to build mobile apps using JavaScript and html. developers can solely focus on the content of the app while Kendo UI will be able to handle the different platforms it has to run such as iOS or android.

Package managers - JavaScript libraires is not too complex or complicated because the way we download it, install, configure and then run these libraries. To make great applications we at times re-use code that is already been written by developers this could be to implement a UI design or implement basic to advanced functionalities for our app or website (Altexsoft, 2018). Just by using a few lines of commands you can use these plugins or dependencies and save hours of works. JavaScript ecosystem has the perfect option for this called Node package manager (NPM). I use this to download libraries to any of our components front end, mobile, or back end. Most JavaScript tools are easily available through the npm commands.

Tools used for testing and debugging - within the JavaScript ecosystem there are several testing tools available to use.

- AVA – lightweight testing framework which can easily be installed using npm. It is a test runner for NodeJS and allows you to test your JavaScript functions concurrently so vastly decreasing the test time.
- Jest – this testing tool is part of the react ecosystem, a powerful testing tool that works with the react library.

Some additional libraries I have used till now

- Passport – used for authentication.
- Role – Acl - used for providing role-based permissions to users.
- Eslint – used for code consistency , syntax correction, enforcing rules during coding.

The JavaScript ecosystem is always growing. A finding suggested that new frameworks, libraries or plugins are being added each year for all components (mobile, front, back end).
The list of tools and interesting packages used in the JavaScript stack are endless, I have mentioned some of the most famous ones above.
For a full stack developer, it is a complete package with all the tools in one place.

Emerging Commercial Best Practices, The Developers Should Be Aware Of.

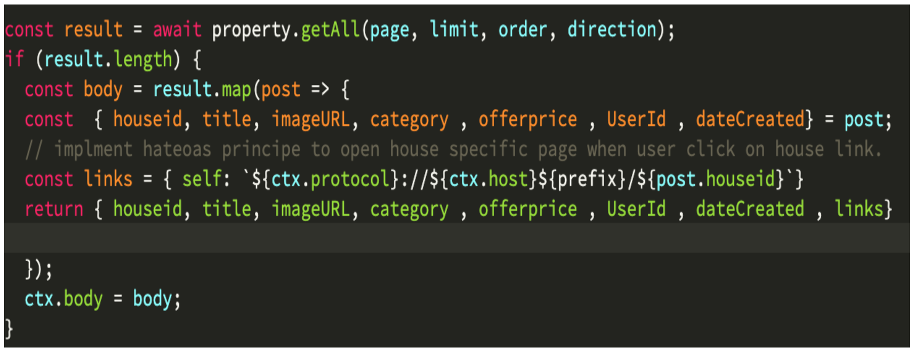
We are allowed today to communicate with data so much more than before, twitter, Instagram, Facebook, git hub, Netflix etc. all the major tech giants Organization provide us services to communicate with their API's endpoints.

As a developer, we are always consuming data from various API, the API is like a graphical user interface for a developer, and if an organization wants more users to interact with its API, the design has to be simple and easy to use.
Being in web development or full a stack developer we would be at some point developing our own Rest API, and so it is crucial that we have consistency in our design and follow some design principles or standards which will allow us to craft API which are simple, straightforward, robust and the most importantly secure.
There are numerous articles, essays on the net on standards, key design principles one should be following while designing an API. Some popular ones are Microsoft Guidelines for an API ,Google design for an API and Pay pal design Guide.

In this part of the report, I would be focusing on some key practice or standards that are emerging in the industry, should be followed, tools or framework's that can be employed with some short example from my own coursework.

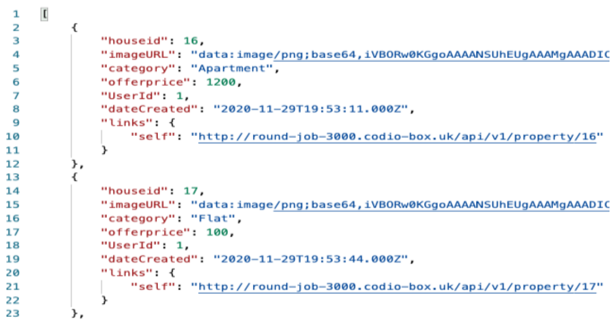
HATEOAS
This principle will appear in many articles and API guidelines. Hypermedia as the engine of application state as daunting this might sound, this is a simple and easy to understand principle that's should be followed when designing an API. it is a constraint of the rest API and is the reason what keeps the restful API architecture different from the other networking architecture.

The idea behind HATEOAS : "A client interacts with a REST API entirely through the responses provided dynamically by the server". Let's see this principle with example now.
Example :- I am building an API for a website that sells houses, when you visit the site home page, you see a list of all houses being displayed. In the background when you visited the website home page, the browser actually made a get request to the server. the server then sends a list of all houses it has in its database. This is in JSON format, let's look at what one of those products might look like



```
const result = await property.getAll(page, limit, order, direction);
if (result.length) {
  const body = result.map(post => {
  const  { houseid, title, imageURL, category , offerprice , UserId , dateCreated} = post;
  // implment hateoas principe to open house specific page when user click on house link.
  const links = { self: `${ctx.protocol}://${ctx.host}${prefix}/${post.houseid}`}
  return { houseid, title, imageURL, category , offerprice , UserId , dateCreated , links}

  });
  ctx.body = body;
}
```

we can add the links to the action we want to perform, this can contain as many links you like, example, link for add to basket, details about the product.



```json
[
    {
        "houseid": 16,
        "imageURL": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAMgAAADIC
        "category": "Apartment",
        "offerprice": 1200,
        "UserId": 1,
        "dateCreated": "2020-11-29T19:53:11.000Z",
        "links": {
            "self": "http://round-job-3000.codio-box.uk/api/v1/property/16"
        }
    },
    {
        "houseid": 17,
        "imageURL": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAMgAAADIC
        "category": "Flat",
        "offerprice": 100,
        "UserId": 1,
        "dateCreated": "2020-11-29T19:53:44.000Z",
        "links": {
            "self": "http://round-job-3000.codio-box.uk/api/v1/property/17"
        }
    },
```

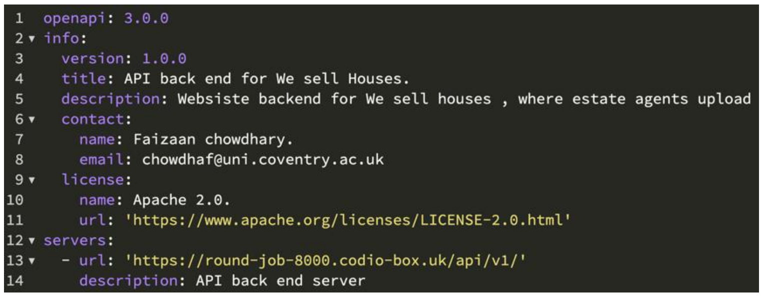
We can see here a JSON object containing list of all the houses, what we are intrested in are the links property which contain a field self. We can use this 'self' link and it to a <a> tag URL which will then open the specific house detail page.

Why do we need to follow HATEOAS why it so important?.
Loose coupling – if s client side needs a restful API service which hard codes all the resources URL , it would then mean it is tightly coupled. With HATEOAS your client side would no longer have any knowledge of the steps involved in the workflow, it does not require the hardcoded URL structure and so it is loosely coupled.
Every restful framework provides their own way of creating HATEOAS links and associated structure.

Documentation
An important process during API design, which developers always prefer to avoid or keep it for the last. Most of the time when developers write code they want to get things done easily. I personally do not like to explain stuff when I know how it works, but when it comes to API, you would have to document everything in your API – routes or endpoints, functions, responses , request body parameters etc. Your documentation will be first point of entry the developer will come to read if they want to consume your data from your API. Documentation will be a great factor to determine the success of your API. All of the information has to be laid out in a simple way, easy to read, understand and efficient manner.

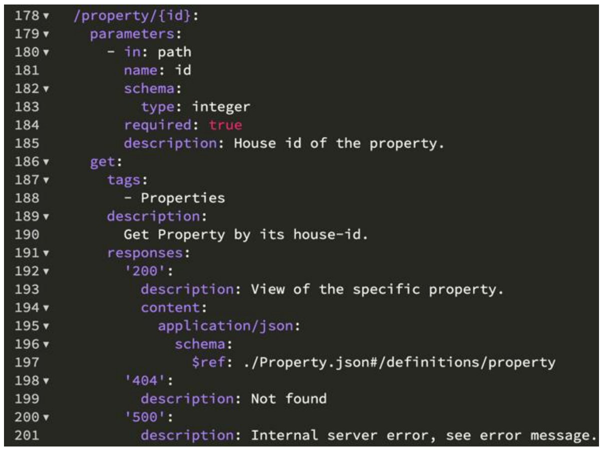
I would be discussing a Documentation tool I recently used called Open Api with an example which is a "broadly adopted industry standard for modern API" (Open API organization, n.d.).

It assists in creating a HTML type document where are all our information describing the API is defined. We can then host this html document like any other web page for users to view.



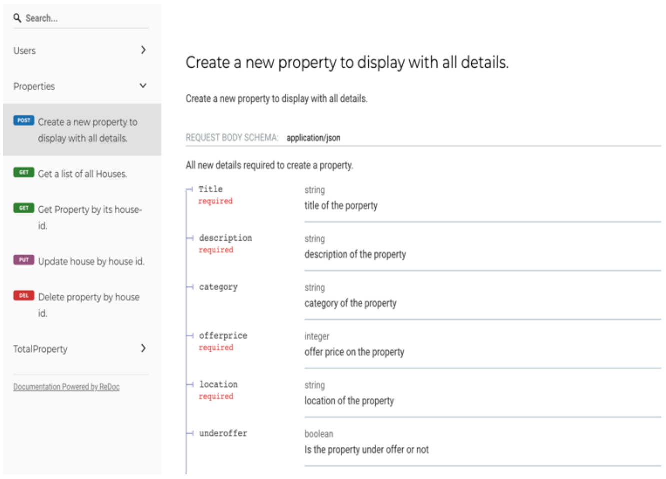
```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: API back end for We sell Houses.
  description: Websiste backend for We sell houses , where estate agents upload
  contact:
    name: Faizaan chowdhary.
    email: chowdhaf@uni.coventry.ac.uk
  license:
    name: Apache 2.0.
    url: 'https://www.apache.org/licenses/LICENSE-2.0.html'
servers:
  - url: 'https://round-job-8000.codio-box.uk/api/v1/'
    description: API back end server
```

We first create a .yaml file in which we define all the information from version, title, description, author, license, servers etc. relating to the API.



```yaml
/property/{id}:
  parameters:
    - in: path
      name: id
      schema:
        type: integer
      required: true
      description: House id of the property.
  get:
    tags:
      - Properties
    description:
      Get Property by its house-id.
    responses:
      '200':
        description: View of the specific property.
        content:
          application/json:
            schema:
              $ref: ./Property.json#/definitions/property
      '404':
        description: Not found
      '500':
        description: Internal server error, see error message.
```

Over here we can see how each path or endpoints has been documented, if it contains parameters, types, description
What is the type of request body data types it's accepting and different responses (status code and description)the client would be receiving
Request body information can be added type JSON schema.


Search...
Users
Properties
POST Create a new property to display with all details.
GET Get a list of all Houses.
GET Get Property by its house-id.
PUT Update house by house id.
DEL Delete property by house id.
TotalProperty
Documentation Powered by ReDoc

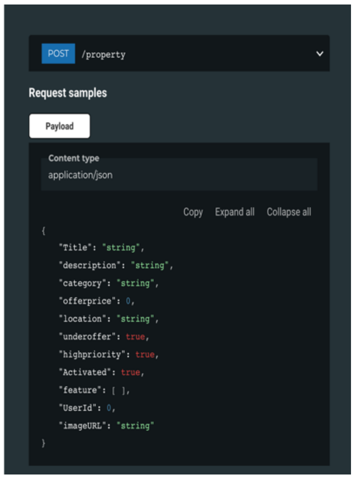
Create a new property to display with all details.

Create a new property to display with all details.

REQUEST BODY SCHEMA: application/json

All new details required to create a property.

Title required	string title of the porperty
description required	string description of the property
category	string category of the property
offerprice required	integer offer price on the property
location required	string location of the property
underoffer	boolean Is the property under offer or not



```json
{
   "Title": "string",
   "description": "string",
   "category": "string",
   "offerprice": 0,
   "location": "string",
   "underoffer": true,
   "highpriority": true,
   "Activated": true,
   "feature": [ ],
   "UserId": 0,
   "imageURL": "string"
}
```

This is the end result with all information beautifully laid out for developers to understand.

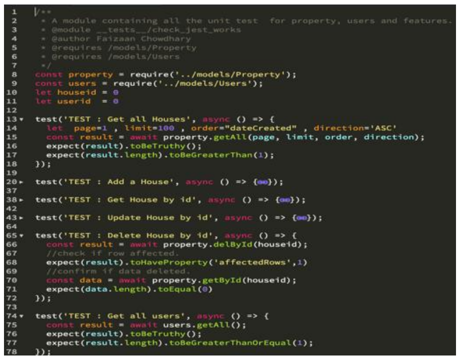
Testing

Testing API is an important part of the development process, testing different aspects of our API tell us if it meets expectation in terms of functionality, security and performance.

continuous early testing provides us :-

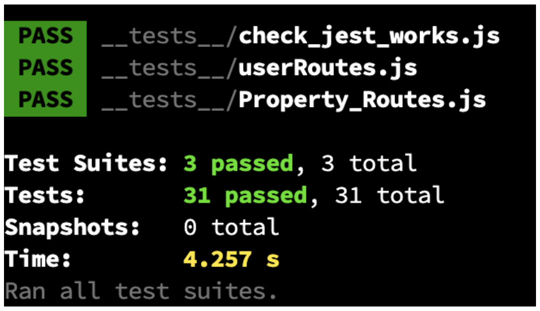
- To identify and bugs or error in the application in the earlier phases.
- Refactor our code is becoming much simpler.
- The process of testing can act as self-documentation.

There are different types of tests and testing frameworks but In this part I would be focusing on unit testing function and routes with a tool called Jest and super test I recently used.



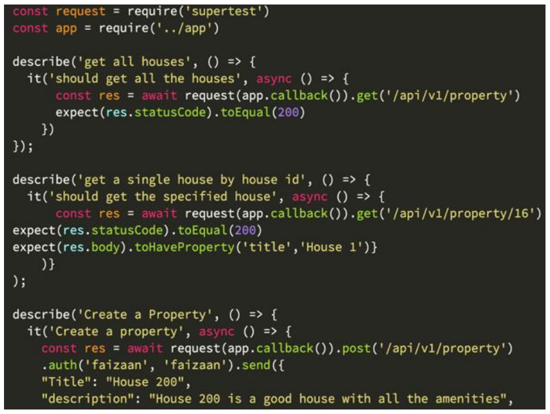
```javascript
/**
 * A module containing all the unit test  for property, users and features.
 * @module __tests__/check_jest_works
 * @author Faizaan Chowdhary
 * @requires /models/Property
 * @requires /models/Users
 */
const property = require('../models/Property');
const users = require('../models/Users');
let houseid = 0
let userid  = 0

test('TEST : Get all Houses', async () => {
  let  page=1 , limit=100 , order="dateCreated" , direction='ASC'
  const result = await property.getAll(page, limit, order, direction);
  expect(result).toBeTruthy();
  expect(result.length).toBeGreaterThan(1);
});

test('TEST : Add a House', async () => {…});

test('TEST : Get House by id', async () => {…});

test('TEST : Update House by id', async () => {…});

test('TEST : Delete House by id', async () => {
  const result = await property.delById(houseid);
  //check if row affected.
  expect(result).toHaveProperty('affectedRows',1)
  //confirm if data deleted.
  const data = await property.getById(houseid);
  expect(data.length).toEqual(0)
});

test('TEST : Get all users', async () => {
  const result = await users.getAll();
  expect(result).toBeTruthy();
  expect(result.length).toBeGreaterThanOrEqual(1);
});
```

Jest - Is a lightweight testing framework for Nodejs. Minimal, fast and runs test concurrently with a simple syntax I have used it recently to perform test on my functions.
In jest syntax is simple and uses promises, we can use plans to assert that a test passes only after certain number assertions have been met.
We can pass and fail our test depending upon what we type of response we are trying to test.



```
PASS  __tests__/check_jest_works.js
PASS  __tests__/userRoutes.js
PASS  __tests__/Property_Routes.js

Test Suites: 3 passed, 3 total
Tests:       31 passed, 31 total
Snapshots:   0 total
Time:        4.257 s
Ran all test suites.
```

Running tests is simple and straightforward to check if they pass or fail and check for errors by failing your tests on it.
We can test all our different files at once. Testing our modules as well our routes at one time.



```
const request = require('supertest')
const app = require('../app')

describe('get all houses', () => {
  it('should get all the houses', async () => {
      const res = await request(app.callback()).get('/api/v1/property')
      expect(res.statusCode).toEqual(200)
    })
});

describe('get a single house by house id', () => {
  it('should get the specified house', async () => {
      const res = await request(app.callback()).get('/api/v1/property/16')
expect(res.statusCode).toEqual(200)
expect(res.body).toHaveProperty('title','House 1')}
    )}
);

describe('Create a Property', () => {
  it('Create a property', async () => {
    const res = await request(app.callback()).post('/api/v1/property')
    .auth('faizaan', 'faizaan').send({
    "Title": "House 200",
    "description": "House 200 is a good house with all the amenities",
```

SuperTest – using supertest to test routes, and check if they return the right responses the ones we are expecting, we can use this to pass or fail the test.
supertest can make the get , post , put or delete request to the routes or endpoints, we export all our routes for supertest to access them then test those routes and check for expected responses and properties depending upon which the test would pass or fail.

Conclusion

This Assignment provided me the knowledge and understanding of different components involved while developing websites. I now have a clear picture what each component is (back end , front end , databases), how they are developed, what role each one of them plays, how they communicate with each other. Using different packages for different functionalities made me more aware of how they are used , how we read their documentation and deploying them. The key of this module is in the variety of tools we use which is the best part (Postman , datagrip , Open API , jsdocs , eslint , role-acl , passport-http) and the different process involved (documentation , testing , continuous integration) and most importantly the JavaScript ecosystem which was amazing to work with.

Bibliography

Altexsoft. (2018, January 18). *Altexsoft*. Retrieved from The Good and the Bad of JavaScript Full Stack Development: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-javascript-full-stack-development/

Altexsoft. (2018, November 27). *Altexsoft softare and engineereing*. Retrieved from JavaScript Ecosystem: 38 Tools for Front- and Back-End Development: https://www.altexsoft.com/blog/engineering/javascript-ecosystem-38-tools-for-front-and-back-end-development/

Jedrzejewski, B. (2018, february 16). *HATEOAS – a simple explanation*. Retrieved from https://www.e4developer.com/: https://www.e4developer.com/2018/02/16/hateoas-simple-explanation/

Nikita Duggal. (2019, December 24). *Who is a Full Stack Developer, and Advantages of Becoming one*. Retrieved from Simpli Learn: https://www.simplilearn.com/full-stack-developer-skills-and-advantages-article

Open API organization. (n.d.). *Open Api - Home page*. Retrieved from Open Api : https://www.openapis.org/

PayPal Engineering. (2013, November 22). *Medium - PayPal Engineering*. Retrieved from Medium: https://medium.com/paypal-engineering/node-js-at-paypal-4e2d1d08c

Stack Overflow. (2019). *https://insights.stackoverflow.com/survey/20*. Retrieved from Stack Overflow: https://insights.stackoverflow.com/survey/20