Menofia University

Faculty of Electronic Engineering

Department of Computer Science & Engineering

Graduation Project

Market Navigator

Your guide to smart shopping

By

Maged Magdy Zaghloul

Kareem Mohammed Etaam

Omnia Hosam Selim

Mohammed Mofreh Saad El-din

Mohsen Sabry Abdelmohsen

Supervisor	Head of the department	Dean
Prof. Ahmed Samy	Prof. Mohamed Berbar	Prof.Ayman El-Sayed

Table Of Contents

Chapter 4: Cloud and DevOps work

Chapter 5: Web Application Security Test

Project Acknowledgement Page

We would like to express our sincerest gratitude and appreciation to everyone who
has contributed to the success of this project.

First and foremost, we would like to thank our supervisor for his guidance, support, and valuable feedback throughout the project

Special thanks go to the members of the project team who dedicated their time and effort to make this project a reality. Their hard work and commitment to excellence were instrumental in the success of this project.

Thank you all for your contributions, support, and encouragement throughout this project.

Abstract

In our graduation project, we intend to develop a platform with a focus on e-commerce endeavors. From the perspective of potential customers, envision a scenario where one desires to purchase a specific product, such as a smartwatch. Through our website, users will have the capability to input product details, including specifications, initiating a search. Subsequently, we will furnish a comprehensive list of retail establishments situated within the preferred vicinity, facilitating price comparison and perusal of consumer feedback. This affords users the opportunity to make informed decisions based on pricing and product reviews, ensuring an optimal shopping experience.

Furthermore, we will curate an extensive inventory encompassing various e-commerce platforms such as Amazon, Noon, and Jumia. This enables aficionados of online shopping to conduct comparative analyses across different platforms, thereby empowering them to select the most suitable option based on their preferences and requirements. In essence, our objective is to meticulously survey the market landscape to offer users access to competitive pricing and superior product quality.

On the other hand, from the perspective of retail store proprietors, our platform serves as a conduit for effective product marketing. We facilitate their engagement by enabling them to establish accounts and showcase their merchandise. Consequently, when a prospective buyer conducts a search, our system seamlessly recommends these products to their target demographic, complete with comprehensive details to facilitate direct engagement and eventual purchase.

Web Design
And
Implementation

1.Web Design: -

the process of creating and designing websites. It encompasses various aspects, including the layout, content, graphics, and user interface. The primary goal of web design is to create a website that is visually appealing, easy to navigate, and provides a good user experience.

1.1 Key Components: -

1.Layout:

This refers to how the content is structured and arranged on the webpage. It includes the placement of headers, footers, navigation menus, and content sections.

2. Graphics:

This includes images, icons, and other visual elements that are used to enhance the look of the website. Graphics should be used thoughtfully to avoid clutter and maintain fast loading times.

3. Colour Scheme:

Choosing the right colours is crucial for aesthetics and usability. Colors should align with the brand and be consistent throughout the site.

4.Typography:

This involves selecting appropriate fonts, sizes, and spacing for text. Good typography enhances readability and overall user experience.

5. Content:

The text, images, videos, and other multimedia elements that provide information and engage users. Content should be relevant, informative, and well-organized.

6.Navigation:

This refers to how users move through the website. Effective navigation ensures users can find the information they need quickly and easily. This includes menus, links, and buttons.

7.User Experience (UX):

UX design focuses on making the website intuitive and easy to use. It involves understanding the needs of the users and designing a site that meets those needs.

8.Responsive Design*:*

Ensuring the website looks and functions well on different devices, such as desktops, tablets, and smartphones. This often involves using flexible layouts and media queries.

2. What is UI Design?

UI (User Interface) design is the process of designing the visual and interactive elements of a digital product or interface. It focuses on the look and feel of the product, ensuring that users have an intuitive and aesthetically pleasing experience. UI design involves creating the layout, visual hierarchy, and interactive elements that users engage with.

2.1 Key aspects of UI design: -

Visual Design:

Crafting the overall look and feel of the interface, including colors, typography, and spacing.

Interactive Design:

Designing interactive elements such as buttons, sliders, and forms, ensuring they respond appropriately to user actions.

Information Architecture:

Organizing content and functionality in a way that is logical and user-friendly.

Prototyping:

Creating interactive models of the design to test functionality and user flow before final development.

Usability:

Ensuring the interface is easy to use and understand, reducing the learning curve for users.

2.2 Figma UI Design: -

- A popular design tool used by UI/UX designers.
- Enables designers to create interactive prototypes, simulating how the final product will work.
- Help in understanding the overview of each function of project before any real implementation.
- Prototypes can be shared with stakeholders and users for feedback before development begins.
- Has an intuitive interface that is easy to learn, making it accessible to both beginners and experienced designers.

2.2.1 Usage in Project: -

- Figma has been playing a great role in our project , it gives us a full structure of website and its functionality in each individual part , the interaction among different components with so simple use.
- Figma Provided us with a good design view and accessibility to it so easily.
- Figma Provided us with prototyping show the order of pages and the action happened when moving from on component to the next.
- Figma is a powerful tool in designing the project.

3.Website Implementation

3.1 Tools used in implementing the web pages.

3.1.1 HTML Hyper Text Markup Language

is the standard language used to create and design webpages. It structures the content on the web and describes the semantic meaning of the content.

Usage of HTML

- HTML provides the basic structure of websites
- HTML allows embedding images, videos, audio, and other multimedia.
- HTML enables linking to other documents and resources, both within the same website and externally.
- HTML forms are used for collecting user input and interacting with web servers.

Versions of HTML & Features

HTML 1.0

Basic structure for web documents; very limited set of tags.

HTML 2.0

Added form elements, tables, and other fundamental tags.

HTML 3.2

Support for applets, text flow around images, and more styling options.

HTML 4.01

Introduced three variants (Strict, Transitional, Frameset), enhanced multimedia support, and deprecated older tags.

XHTML 1.0

Reformulated HTML 4.01 in XML; enforced stricter syntax rules.

HTML5

New semantic elements (e.g., <article>, <section>), native audio and video support, improved form controls, canvas element for drawing, and better handling of web applications.

3.1.2 CSS Cascading Styling Sheet

is a stylesheet language used to describe the presentation of a document written in HTML or XML. It controls the layout, colors, fonts, and overall visual appearance of web pages.

Usage of CSS

Styling HTML: CSS is used to style HTML elements, such as setting colors, fonts, spacing, and layout.

Responsive Design: CSS is crucial for creating responsive designs that adapt to different screen sizes and devices.

Animations: CSS can create animations and transitions, enhancing user experience.

Versions of CSS & Features

CSS1:

Basic styling capabilities like fonts, colors, and text alignment.

CSS2:

Added support for positioning, z-index, media types, and more complex selectors.

CSS2.1:

Refined and fixed issues in CSS2, introduced box model, float, and clear properties.

CSS3:

Modularized approach, new features like rounded corners, shadows, gradients, transitions, animations, and new layout models like Flexbox and Grid.

3.1.3 JS Java Script

is a versatile, high-level programming language primarily used for adding interactivity and dynamic behaviour to web pages. It can be used on both the client-side (in the browser) and server-side (with Node.js).

Usage of JavaScript

Client-Side Scripting:

Enhances user interfaces, handles events, and updates content dynamically.

Server-Side Programming:

With environments like Node.js, JavaScript can be used to build server-side applications.

Web APIs:

Interacts with various web APIs to handle tasks like fetching data, manipulating the DOM, and more.

- As you see, for HTML, CSS versions , also JS has multiple versions and many updates, but the most popular usage of JS nowadays is by frameworks.

JS Frameworks

JavaScript frameworks significantly enhance the development process by providing tools and structures to build robust and scalable web applications. Understanding the features and use cases of each framework helps developers choose the right tool for their specific needs, ensuring efficient and maintainable code.

Choosing the Right Framework

Choosing the right JavaScript framework depends on several factors, including the project requirements, team expertise, and long-term maintenance considerations.

Usability of some of JS frameworks

React: Best for building highly interactive user interfaces and SPAs.

Angular: Suitable for large-scale, enterprise-level applications with complex features.

Vue.js: Great for both small and large projects, known for its ease of use and flexibility.

Node.js + Express.js: Ideal for building scalable server-side applications and RESTful APIs.

Svelte: Perfect for high-performance applications with a focus on simplicity and modern features.

3.2 Set Up the Coding Environment

3.2.1 Getting Node JS

What is Node JS?

Node JS is free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

Key Features of Node.js

Event-Driven and Non-Blocking I/O:

Node.js uses non-blocking, event-driven architecture, which makes it efficient and suitable for I/O-heavy operations like reading/writing to disk, network connections, or accessing databases.

Although Node.js runs in a single-threaded environment, it can handle multiple connections concurrently thanks to its event loop.

Fast and Scalable:

Node.js leverages the V8 JavaScript engine, which compiles JavaScript to machine code, making it very fast.

Its non-blocking nature and ability to handle many connections simultaneously make it ideal for scalable applications.

Rich Ecosystem:

Node.js has a robust package ecosystem with npm, the world's largest software registry, which provides thousands of libraries and modules for various functionalities.

Cross-Platform:

Node.js runs on various operating systems like Windows, Linux, macOS, and others, making it highly versatile.

Unified Language:

Developers can use JavaScript for both client-side and server-side development, promoting code reusability and consistency.

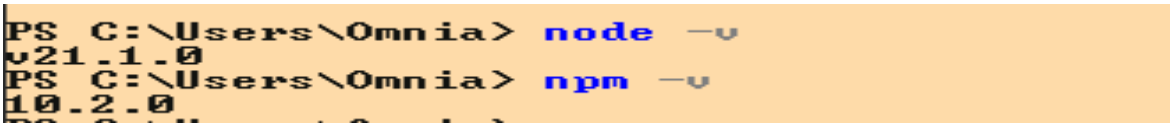
What is npm?

npm (Node Package Manager) is the default package manager for Node.js, which is an environment for executing JavaScript code outside of a web browser. npm allows developers to install, share, and manage JavaScript libraries and dependencies in their projects.

You can install it by terminal, but make sure you have node js already

```
npm install -g npm
```

To check if you have both node , npm already, you can use this:



3.2.2 Creating React Application

- Choose the name of your application and run the command, wait form some minutes to install all default application packages to be able to run it

```
PS C:\Users\Omnia> npx create-react-app --AppName--
```

- Once the folder of the application created, you can open it using any editor to work on it
- Also to run it you need to get into the directory of the application and run the command

```
PS C:\Users\Omnia> npm start
```

- Open http://localhost:3000 to view it in the browser, if it hasn't opened already in your browser.

  The page will automatically reload if you make changes to the code. You will see the build errors and lint warnings in the console.

Additional Information:

npx (Node Package Executor) is a tool that comes with npm (version 5.2.0 and higher). It allows developers to execute Node.js packages without installing them globally, making it easier to run one-off commands or use CLI tools that are not required to be installed in the system permanently.

Web Folder Structure

After creation, your project should look like this: assume your app name is my-app

```
my-app/
  README.md
  node_modules/
  package.json
  public/
    index.html
    favicon.ico
  src/
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
```

Before starting, it's important to know some information about these files

node modules/ Contains all the npm packages and dependencies for the project.

public/

index.html: The main HTML file; contains the root `<div id="root"></div>` where the React app is mounted.

src/

App.js: Main component that serves as the root of your React application.

App.test.js: Test file for the App component using Jest.

index.js: Entry point for the React application; renders the App component into the DOM.

reportWebVitals.js: Measures and reports on the performance of your app (optional).

setupTests.js: Sets up the testing environment with Jest (optional).

Root-Level Files

package.json: Lists dependencies, scripts, and project metadata.

package-lock.json: Lock files to ensure consistent dependency installations.

3.3 Website Structure

3.3.1 Overview

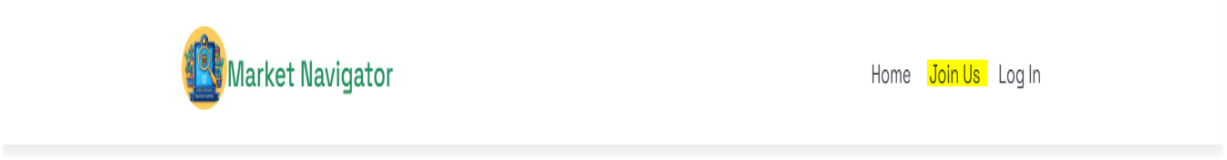
The website is built on many stages

1. **Shared components that show on multiple pages of it.**
2. **Login and register are considered the main function to get your account and interact freely**
3. **Buyer or Seller functionality after login**
   As example, buyer can see all the products, search in it , find the shops near, change his/her profile information, but in the other side , the seller can add products, edit or delete it freely , search in his/her products freely also can change the profile information

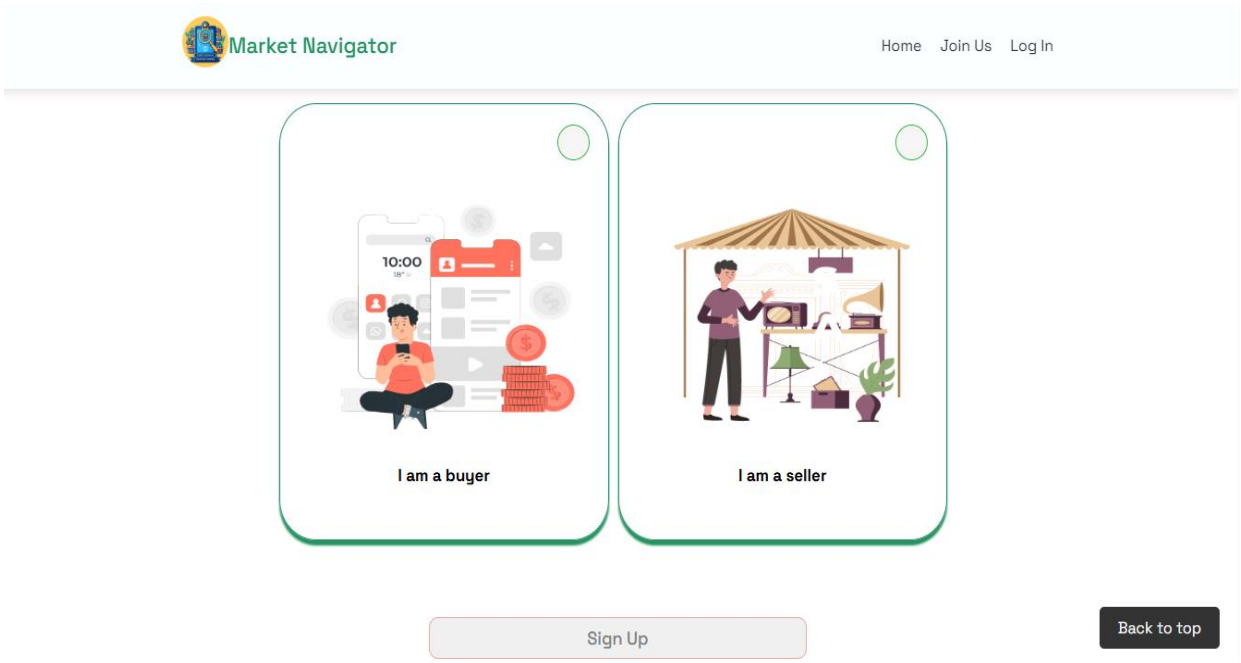
3.2.2 Create an account

There are two choices if you are buyer or seller.

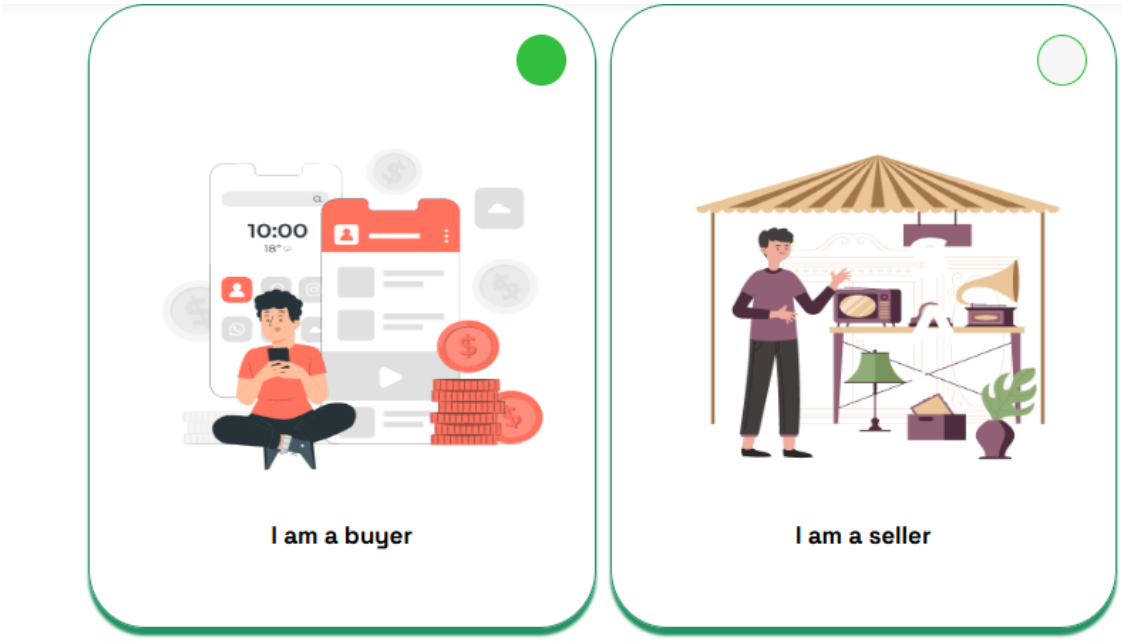
You can easily start make an account by clicking on Join us in the navbar



You directly is being routed to user type page , which give you the free choice if you are buyer or seller

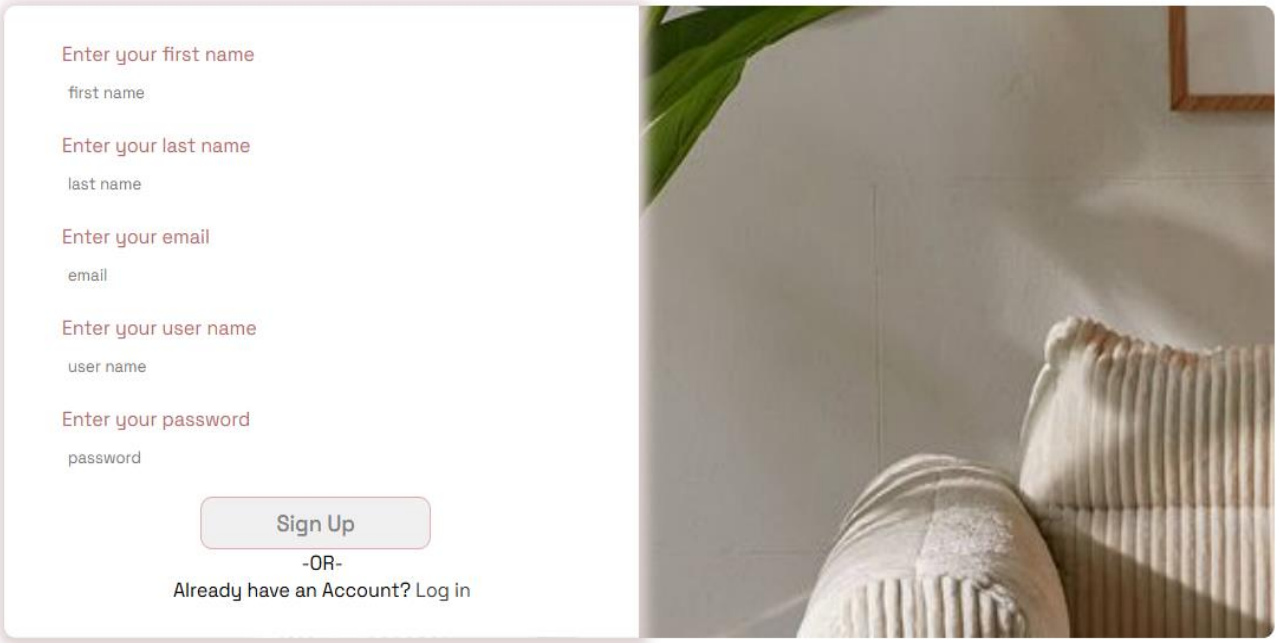


Once you have chosen, click sign up and then automatically going to register page.



The register page for each of them is different in data needed to enter

Buyer Sign-up page



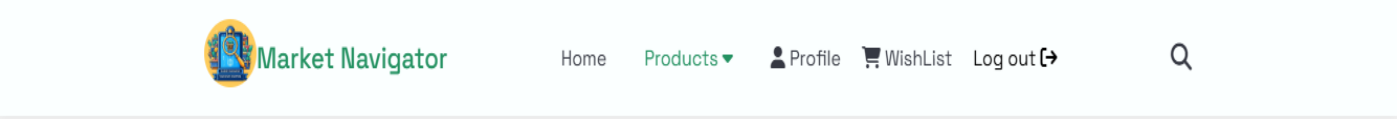
Seller Sign-up Page

You need as a seller to enter some additional details for your shop, location of it and others, it is with the same style but more fields.

Once your validation for Sign Up is correct for password valid ,email valid and username isn't found before , you can successfully got into the home page

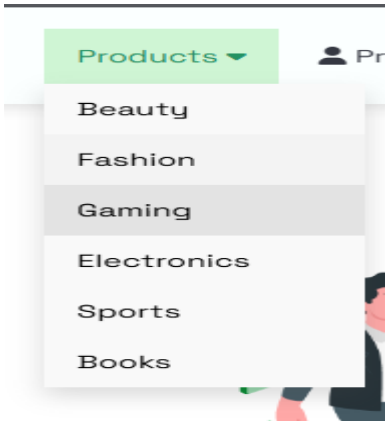
Buyer Account

1. Specific navbar

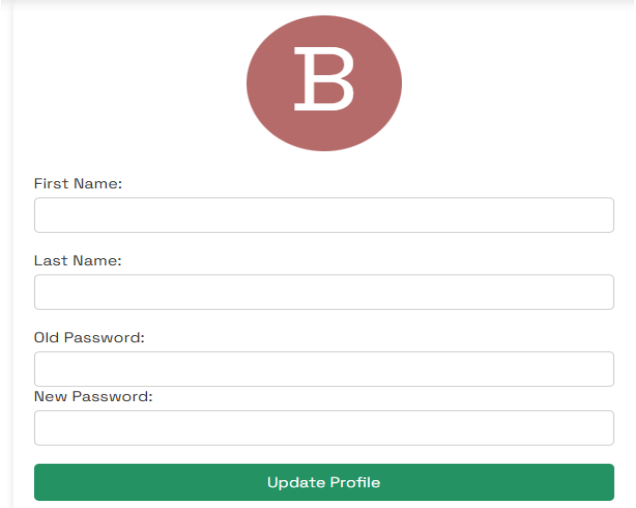


Specific functionality

- show specific products category



- Update the account profile



- Add to the Wishlist or remove from it freely

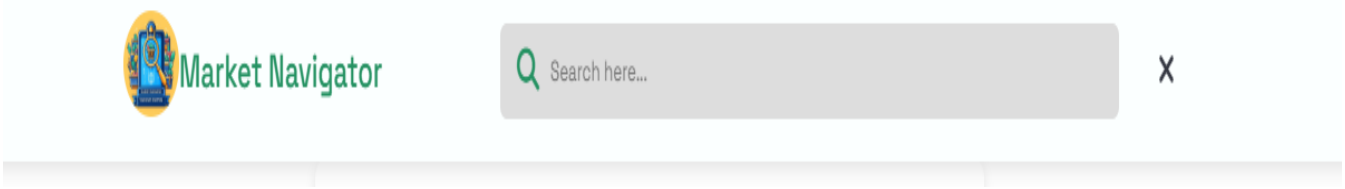
Only click on Wishlist icon in the navbar

- Logout from the website

If you are click logout, you will leave your account for now , but you can return back and find your account as it is , no loss in your wishlist products.

Finally for buyer, he/she can search for products by multiple ways

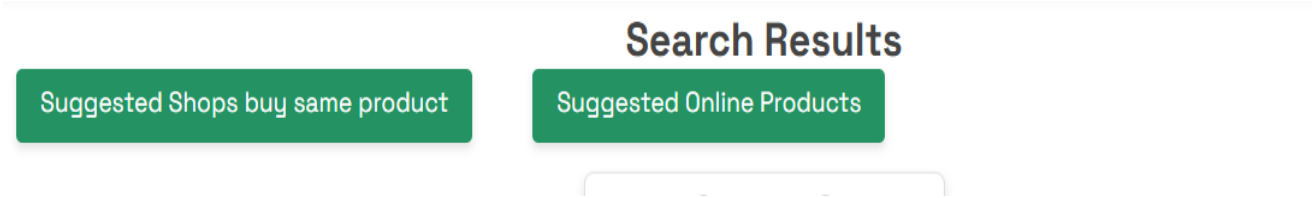
Search by word globally in the website



Put the word you want to search for and click on the search icon, it will route you to search results page with all products related to what you searched.

In it you can have two choices

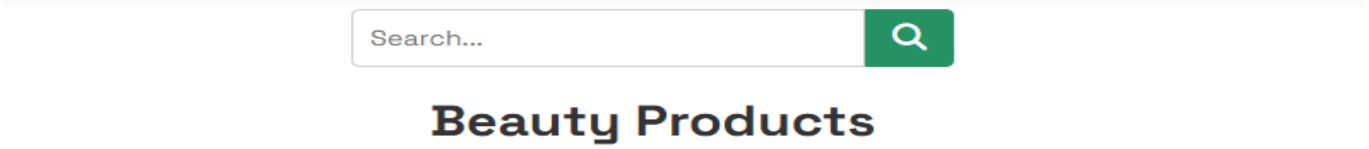
- Getting the offline suggestions for the shops buy the product and their distance far away from your current location.
- Getting products from online website, and you can go to the product page and see it with more details



Search inside each category of products

You can easily here search for any beauty product you need, the search here responsive to each character you write

This custom search bar provided for all products categories

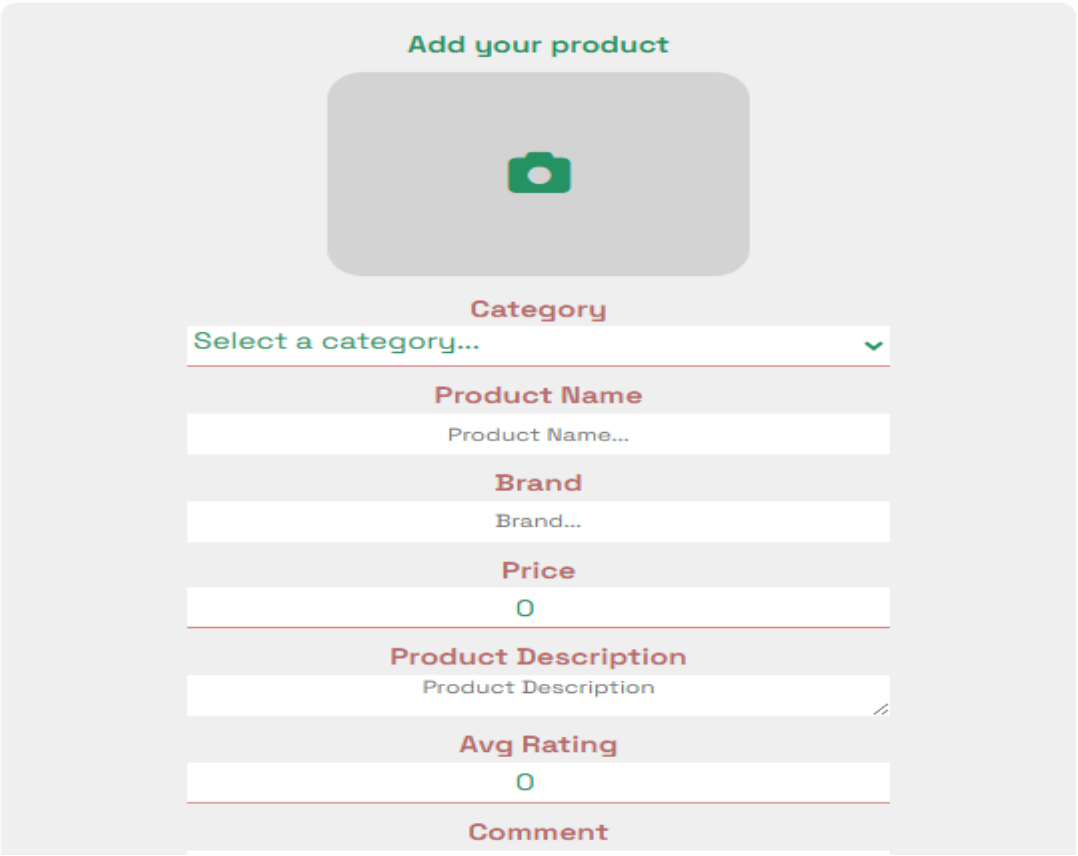


Seller Account

- Specific navbar and header

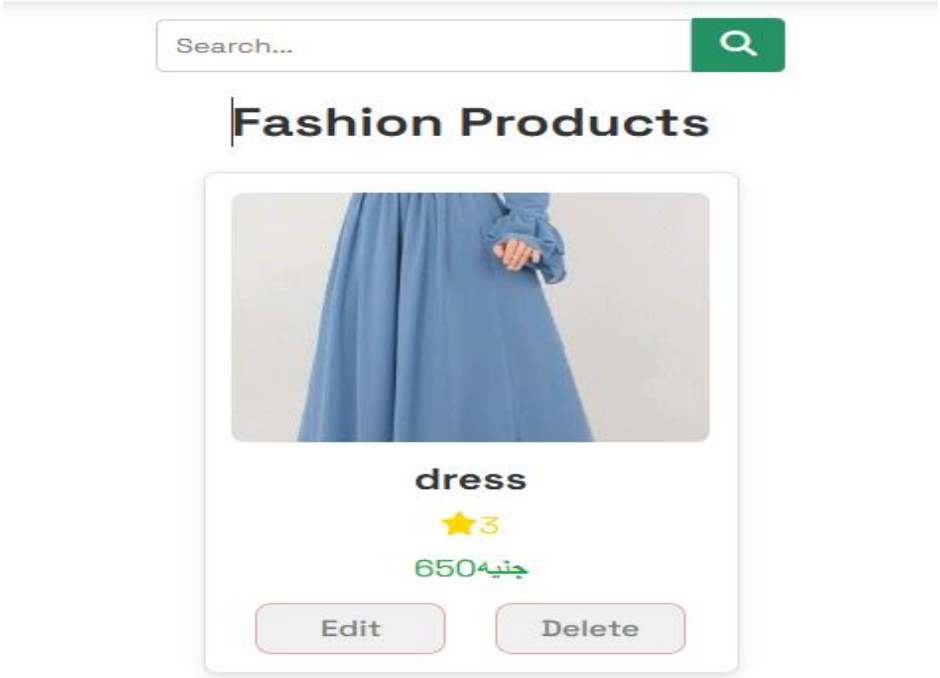
- Create a post (Adding new product)

You can add image of you project in extension png,jpg only



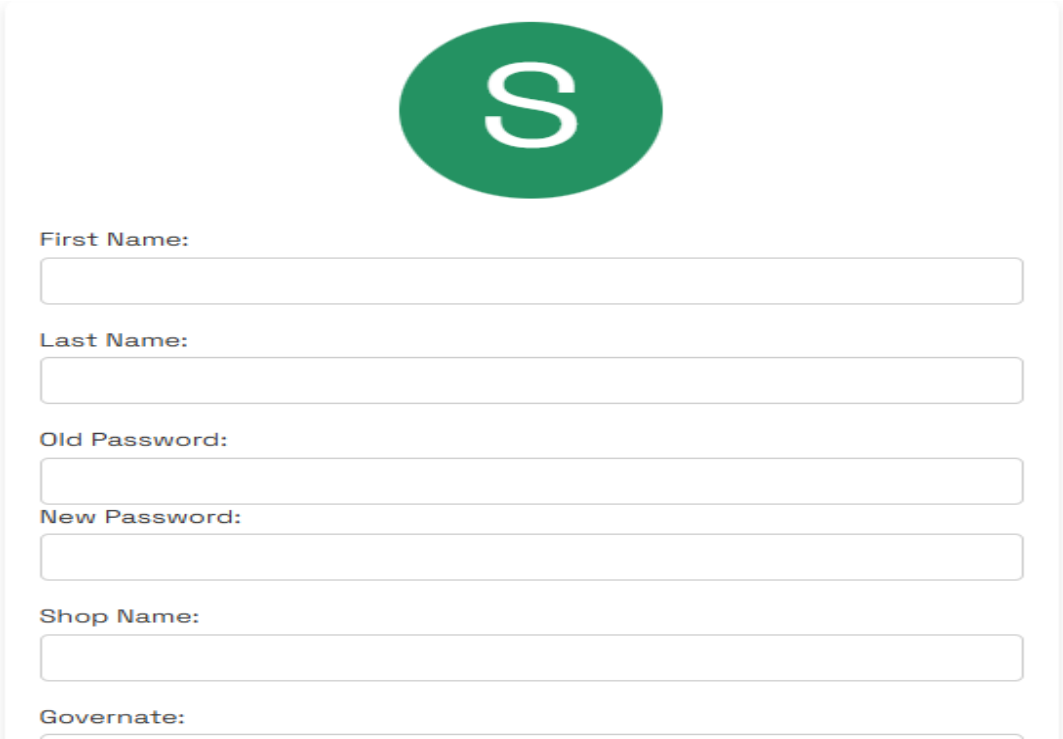
- See the products you added

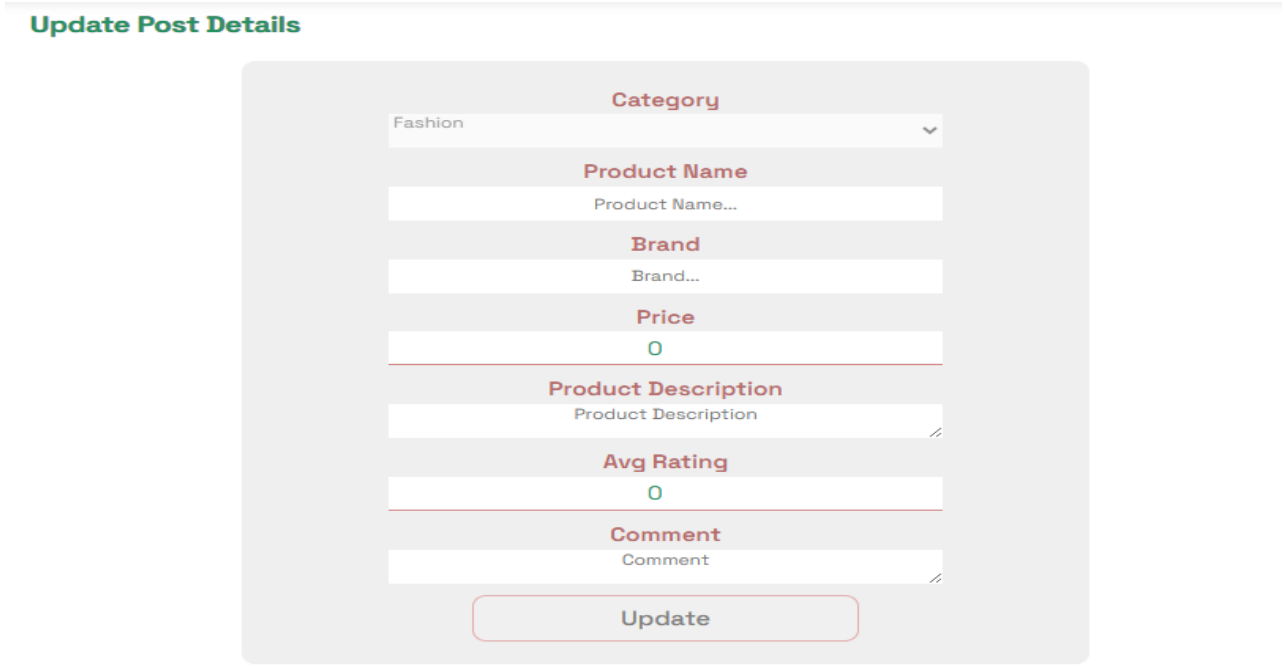
You can edit the product details or delete it



- For update

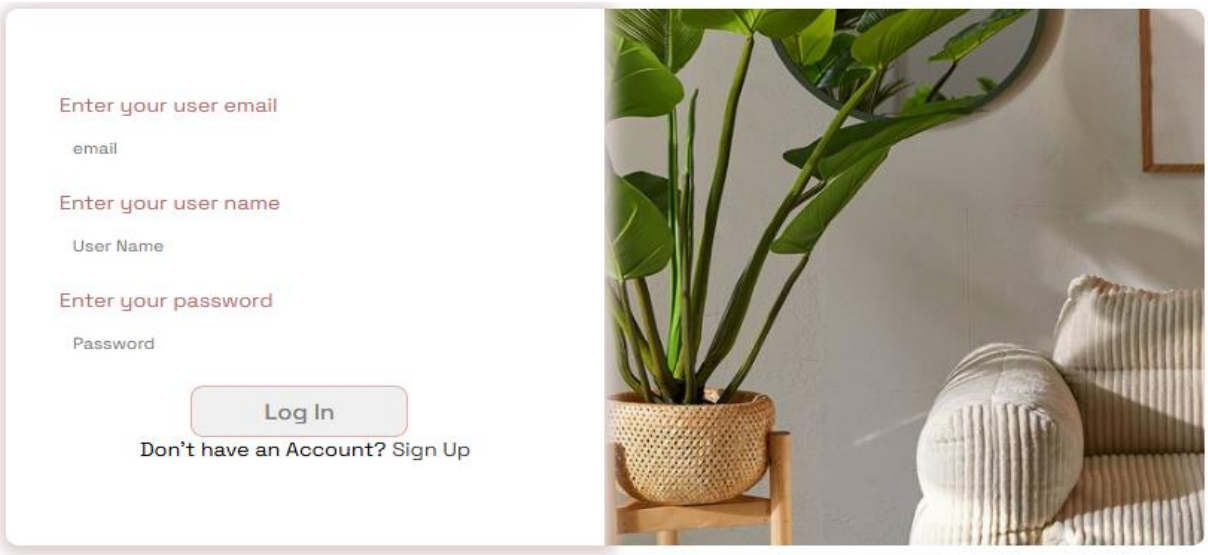
- Change profile details





As you see, the profile page and products are common for both buyers and sellers, only in profile the icon in the first changes based on the user type

Log In your account



3.3.3 Main components used in build the structure

1. **Product Component**
    - Built to be as the parent of all products categories
    - Handle fetching the products available for each category
    - Handle adding product to wishlist
    - Handle remove product from wishlist

- Handle the delete and update the product for seller
- Handle filtering products in search

Using to build it a props to give the component from each category page the api used to fetch their APIs successfully

```
export default function Product({ prodApi, wishApi, imageApi, name, category }) {
```

prodApi is the category api to get its products

wishApi is the category api to get its wishlist products

imageApi is the category api to get its products images

name is the name shown as a title of page of category

category is the name of category itself

by creating a component for each category and passing the parameters needed, you can show the category page so easily

- The product component is rendered in beauty product page and by passing the parameters to it , it will show the product successfully

- The same handling for the rest categories

```
import { beautyAPI } from "../../Constants";
import Product from "./Product";

export default function BeautyProduct(){
return(
    <div>
        <Product
        prodApi={beautyAPI.product}
        wishApi={beautyAPI.wishlist}
        imageApi={beautyAPI.image}
        name="Beauty Products"
        category="beauty"
        />

    </div>
)
}
```

2. **Update Post**

   With the same way of using props , making update post is the component that has a logic , but the children who uses it only passes parameters to it

```
import React, { useState } from 'react';
import Button from '../../Components/Button';
import axios from 'axios';
import { beautyAPI, sportsAPI, bookAPI, electronicAPI, gamingAPI, fashionAPI } from '../../Constants';
import NavBar from '../../Components/NavBar';
import Foot from '../../Components/Foot';
import Modal from '../../Components/Modal'; // Import your existing modal component

export default function UpdatePost({ category }) {
```

Here there a small difference that handling the APIs from the file containing them directly only take the category of product to justify the updated logic.

```
import UpdatePost from "./UpdatePost";
export default function SportsUpdatePost(){
return(
    <div>

        <UpdatePost
        category="sports"
        />

    </div>
)
}
```

3.4 Bottlenecks faced while creating

1. Dealing with the conversion of image stored in database

This problem solved by adding function of conversion binary image format to png or jpg.

```
// Function to convert base64 to data URL
const base64ToDataURL = (base64String) => {
  let mimeType = 'image/png'; // Default MIME type
if(!base64String) return''
  // Check the first few characters to determine the MIME type
  if (base64String.charAt(0) === '/') {
    mimeType = 'image/jpeg';
  } else if (base64String.charAt(0) === 'i') {
    mimeType = 'image/png';
  }

  return `data:${mimeType};base64,${base64String}`;
};
```

It is based on taking the image stored in database and check it of type png or jpeg and based on the type , then use the format in return of the function

2. Deleting the product of seller

The problem was in database dependency, to delete product,

Check if it is in wishlist or no and if so, delete it.

Delete the product image

Delete the product

```
function DeleteProduct(prod,index) {
  const deleteImage = async () => {···
  };

  const deleteFromWishlist = async () => {···
  };

  const deleteProduct = async () => {···
  };

  const isProductInWishlist = async () => {···
  };

  const handleDelete = async () => {···
  };

  handleDelete();
}
```

3. Routing among different pages

To add routing in React, you need to install react-router-dom library

by command in the project directory terminal

```
npx i  react-router-dom
```

After that you can render the components and determine the paths.

Browser router used to obtain routing on the level of App component.

```
<BrowserRouter>
<App />
</BrowserRouter>
```

In the App you can defines the paths to show

First import the paths

Then determine them in Route

```
import { Route, Routes } from "react-router-dom";
import UserType from "./Pages/SignUp/UserType";
import SignUpSeller from "./Pages/SignUp/SignUpSeller";
import SignUpBuyer from "./Pages/SignUp/SignUpBuyer";
import Login from "./Pages/SignIn/Login";
import LoginType from "./Pages/SignIn/LoginType";
```

```
<Route path='/usertype' element={<UserType/>}/>
<Route path='/signupseller' element={<SignUpSeller/>}/>
<Route path='/signupbuyer' element={<SignUpBuyer/>}/>
<Route path='/login' element={<Login/>}/>

<Route path='/logintype' element={<LoginType/>}/>
```

Additional Point that you can pass from the pages itself to other by a tag like anchor tag in html called link, it needs to be imported from the same library 'react-router-dom'

```
<Link  to='/' className="more-link">Return to home page</Link>
```

E-Commerce System
Using ASP.NETCore Web API
(Backend Part)

1. Introduction

Overview of E-commerce Systems

E-commerce systems have transformed the way businesses operate, allowing for online transactions and broadening market reach. Key components of e-commerce systems includeuser management, product management, online suggestion, and offline suggestion. These systems offer convenience and accessibility, making them integral to modern business operations.

Objectives of This Guide

The primary objective of this guide is to develop a secure and efficient e-commerce Web Application using ASP.NET Core Web API. The guide covers the implementation of JWT authentication for secure user access, web scraping for enhanced product suggestions, and defining distinct roles for Buyers and Sellers. The aim is to create a robust system that catersto the needs of both buyers and sellers while ensuring data security and relevance.

Technology Stack

- ASP.NET **Core**
    - High performance, cross-platform framework for building modern, cloud-basedapplications
- Entity **Framework Core**
    - Object-Relational Mapper (ORM) for database interactions
- SQL **Server**
    - Database management system for storing application data

- Swagger
    - API documentation and testing tool

Introducing HTTP

Before we dive into ASP.NET Core 7 and API development, let's first go back to the basics of any web application. Whether a website is run from a browser or a web service (web API), it's always thesame principle: a client and a server will communicate together; a client will send a request to a server, which will then respond to the client. This is all possible with the magic of the HTTP communication protocol. Under this protocol, data can be transported using different formats and constraints, The Hypertext Transfer Protocol (HTTP) is a network protocol for exchanging data between clients and servers. This protocol was invented in 1990 by British computer scientist Tim Berners-Lee to access the World Wide Web (WWW).

a browser using Hypertext Markup Language (HTML) through Uniform Resource Identifier (URI) web addresses. At the beginning of HTTP's history, HTML was the language used to create pages, but since then, HTTP has evolved into a web server that can process data formats other than HTML. For example, a web server can serve (but also accept as input) Extensible Markup Language (XML), which is a structured language, or JavaScript Object Notation (JSON). Of course, a web server can serve other types of data formats, categorized as Multipurpose Internet Mail Extensions (MIME) type,and I will come back to this later.

HTTP follows a technical specification called Request from Comment (RFC), developed by the Internet Engineering Task Force (IETF). There are a ton of RFC specifications identified by numbers.The common point among them is that they define the specifications of the Internet and only the Internet. HTTP is defined by **RFC 7231**. RFC 7231 can be found at this address: www.rfc- editor.org/rfc/rfc7231.

HTTP has three essential characteristics:

1. It is **stateless**: This means that after sending a request to the server and receiving the response, neither the client nor the server retains any information on the exchanged request.

2. It is **connectionless**: An HTTP connection is open between the client and the server. Once the client has received the response from the server, the connection is closed, and the connection between the two systems is not permanent.

3. It is **independent** of the media, that is, the server can transmit any media as long as the client and the server "agree" on exchanging the content.

An HTTP connection works as follows:

a request will receive a response unless the connection is broken. Every request and every response work the same way, and I'll go into more detail in the next section.

An HTTP request works with elements as follows:

- A client (a browser, an application) initiates an HTTP request by invoking a **URI**, the address of the requested resource on the server.

- The URI requires the use of a **verb** that will determine the action to be performed.

- Metadata will be sent in the HTTP request, called **headers**. These headers allow controlling the content negotiated with the server, such as sending authentication information and much more.

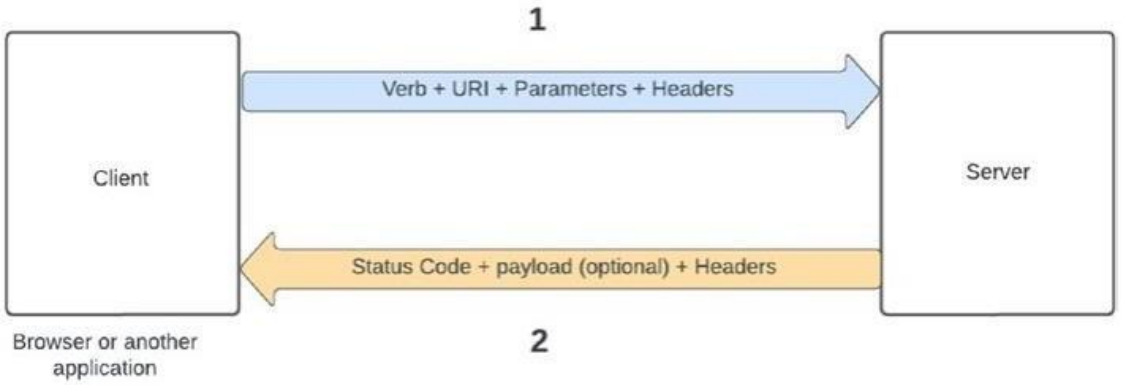
• **Parameters** are necessary to exchange content with the server and obtain the response sought, and the parameters can be in the request's body, the route, or the URI.

An HTTP response works with elements as follows:

• The server returns a response with a simple status code (HTTP status code) to determine how the HTTP request processing took place.

• The server also returns **headers** in response to the client providing with different metadata.
• Finally, the server will return (or not) a **payload** formatted in the MIME type requested by the client.

So far, I have briefly described and simplified how HTTP works



In the following section, I will detail the HTTP **verbs**, the **request headers**, the format of a **URI**, the different **parameters** passed in a request, the HTTP **status codes**, the **response headers**, and the **payload** formats returned to the client.

Headers in the context of HTTP (Hypertext Transfer Protocol):

They are components of both requests and responses that carry additional information about the message being transmitted. They are fundamental to how communication between clients (such as

web browsers or applications) and servers (which host websites or APIs) is structured and managed. Here are key points about headers:

1. **Purpose**: Headers provide essential metadata about the request or response. Theyconvey information necessary for the proper handling and interpretation of the message by both the client and the server.
2. **Location**: Headers are located at the beginning of an HTTP message—either in a request sent by the client to the server or in a response sent by the server to the client.
3. **Types**:
    - **Request Headers**: These headers are included in an HTTP request from theclient to the server. They typically include information about the client itself(like the user agent), the desired action (HTTP method like GET or POST), and details about the content being sent (such as Content-Type or Accept headers).
    - **Response Headers**: These headers are included in an HTTP response from theserver to the client. They provide information about the server's response

status (like HTTP status codes), the content being returned (Content-Type), and other additional details that help the client interpret and process the response.

4. **Common Headers**: Some commonly used headers include:
    - **Content-Type**: Specifies the media type of the resource (e.g., text/html, application/Json).
    - **Content-Length**: Indicates the size of the entity-body in bytes.
    - **Cache-Control**: Directives for caching mechanisms in both requests and responses.
    - **Authorization**: Contains credentials for authenticating the client with the server.
    - **User-Agent**: Identifies the client software (e.g., browser) initiating the request.
    - **Location**: Used in redirects to specify the new location.
5. **Custom Headers**: Besides standard headers defined by HTTP specifications, applications can define custom headers to convey specific application-related information between client and server.
6. **Format**: Headers consist of a name-value pair, where the name is followed by a colon (:) and then the value. Multiple headers can be included in a single message, each on a new line.

In summary, headers in HTTP are crucial for controlling and facilitating communication between clients and servers by providing metadata that ensures requests and responses are processed correctly and efficiently.

HTTP payload (Hypertext Transfer Protocol payload):

a payload refers to the actual data being transmitted in either an HTTP request or response. It is the content that is carried within the message body and is distinct from the headers, which provide metadata about the payload.

Here are key points about HTTP payloads:

1. **Location**: The payload resides within the message body of an HTTP request or response. In requests, it contains data sent from the client to the server (e.g., form datain a POST request). In responses, it contains data returned from the server to the client(e.g., HTML content of a webpage or JSON data).
2. **Format**: HTTP payloads can be formatted in various media types, such as:
    - **Text-based formats**: Like HTML, XML, JSON, plain text.
    - **Binary formats**: Such as images, videos, or any other binary data.
3. **Content Negotiation**: The Content-Type header in both requests and responsesspecifies the media type (MIME type) of the payload being sent or received. Thisallows clients and servers to understand how to interpret the data.
4. **Handling**: Clients and servers process the payload according to the Content-Type header and other headers related to content encoding (Content-Encoding) or contentlength (Content-Length).
5. **Manipulation**: The payload can be manipulated through various HTTP methods likePOST, PUT, and DELETE. For example:
    - **POST**: Used to send data to the server to create a new resource, with thepayload typically in the request body.

- **PUT**: Often used to update or replace an existing resource on the server.
- **GET**: Retrieves a resource from the server without a payload in the requestbody (though it can have one in some cases, like for queries).

6. **Size and Efficiency**: Payload size and efficiency are critical considerations in webperformance and network bandwidth usage. Headers like Content-Lengthhelp indetermining the size of the payload, and compression techniques (like gzip) can beused to optimize payload transmission.

In essence, the HTTP payload refers to the actual data being transmitted in an HTTP message, encapsulated within the message body. Its format and content are defined by the Content-Type header, and it plays a crucial role in how information is exchanged betweenclients and servers over the web.

HTTP Implementation:

Let's dive into more detail to see what HTTP verbs, request headers, response headers, and HTTPstatus codes are and how the client passes its parameters in HTTP requests combined with the invocation of a URI.

HTTP Verbs:

RFC 7231 defines the following verbs:

• **GET**: This is the most well-known. It allows you to request a resource from the server and receive aresponse in the desired format. The response is **cacheable** (retain information in memory.

• **HEAD**: This verb is similar to the **GET** verb but does not return any payload; a payload is used to request metadata at the requested address. Since developers barely use it most of the time, Won't use

it in this project, but it's good to know what it is used for. Like GET, the server response is also **cacheable**.

- **POST**: This verb is interesting because it serves multiple purposes. This verb allows the creation of new resources, and its payload is attached to the request's body. Another way to send data is to use the *form-data* technique, which will be described later in this book. The **POST** verb also allows modifying data by adding content to the data (appending data to the resource representation according to the RFC). **The server response is not cacheable unless freshness information** is added to the response headers.

- **PUT**: This verb is confusing because the RFC states that it replaces a resource on the server. Very often, developers confuse **PUT** and **POST** (replace a resource vs. append data to a resource). The server response here is **not cacheable.** However, if a resource doesn't exist, **PUT** should behave as **POST** by creating the resource.

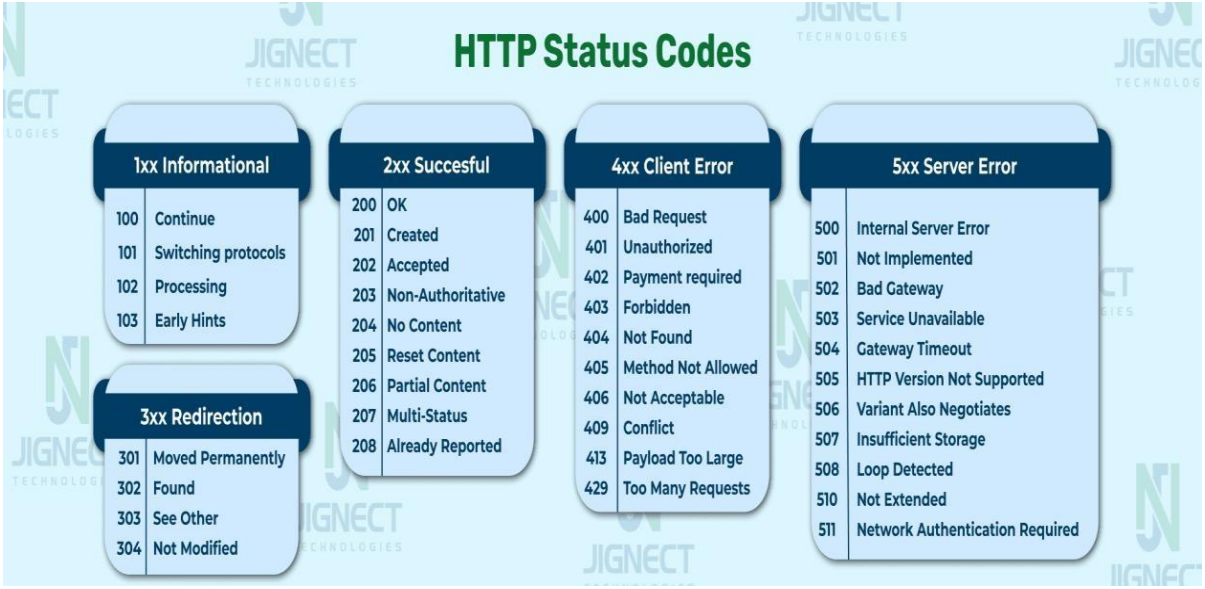
- **DELETE**: This verb is used to delete a resource. The server response is **not cacheable**.

- **CONNECT**: The verb establishes a tunnel to the server through a proxy (a server to which the HTTP request will be delegated and access the server for the requested request). This verb is used for secure requests with *Transport Layer Security (TLS).*

HTTP Status code:

HTTP status codes are used to indicate the result of an HTTP request. They are divided intoseveral categories:

1. **1xx (Informational):**
   - The request was received, continuing process.
2. **2xx (Successful):**
   - The request was successfully received, understood, and accepted (e.g., 200OK, 201 Created).
3. **3xx (Redirection):**
   - Further action needs to be taken to complete the request (e.g., 301 MovedPermanently, 302 Found).
4. **4xx (Client Error):**
   - The request contains bad syntax or cannot be fulfilled (e.g., 400 Bad Request,404 Not Found).
5. **5xx (Server Error):**
   - The server failed to fulfill a valid request (e.g., 500 Internal Server Error, 503Service Unavailable).



2. Project Setup

Creating the Project

Begin by creating a new ASP.NET Core Web API project. Use your preferred IDE or command line tools. Install necessary NuGet packages such as Microsoft.AspNetCore.Authentication.JwtBearer for JWT authentication, Microsoft.EntityFrameworkCore for Entity Framework Core, an ORM framework for database access. It helps in managing database connections and performing CRUD operations, Microsoft.EntityFrameworkCore.SqlServer for Entity Framework Core and HtmlAgilityPack for webscraping. This setup lays the foundation for the project, ensuring all required tools and libraries are available.

Project Structure (Backend)

Understand the project structure of an ASP.NET Core Web API. Key folders include Controllers, Models, DTOs, and Services. Controllers handle incoming requests, Models represent the data, DTOs and Services to handle JWT configuration, more about DTOs and Services we will talk about it next. This clear separation of concerns helps in maintaining andscaling the application.

Before talking about the configuration, we will take some notes about **DTOs** and theimportant of using this concept:

DTOs (Data Transfer Objects) are classes used to transfer data between different layers of anapplication, often between the presentation layer (like controllers in ASP.NET Core) and thedata access layer (such as repositories or services). They serve several purposes in software development:

Purpose of DTOs:

1- Encapsulation of Data: DTOs encapsulate the data that needs to be transferred betweenlayers.They typically contain properties

that represent the data fields but may exclude methods or behavior.

2- Reducing Coupling: DTOs help in reducing the coupling between different parts of the application by providing a standardized way to transfer data. This separation allows changesin one part of the application (e.g., database schema changes) without affecting other parts (e.g., API contracts).

3- Efficient Data Transfer: DTOs can include only the necessary data fields required for a specific operation or response. This optimization reduces the amount of data transferred overthe network, improving performance.

4- Mapping Between Layers: DTOs facilitate mapping between domain/entity models andpresentation models. They help transform complex domain models into simpler representations suitable for specific use cases (e.g., displaying data in a UI).

5- Security and Validation: DTOs can include validation attributes (like those from System.ComponentModel.DataAnnotations) to enforce data integrity and security constraints.They ensure that data transferred between layers meets specified criteria before being processed. Example Scenario:

In an ASP.NET Core Web API application with JWT authentication and web scrapingfeatures: DTOs for User Management,

UserDTO: Contains user-related data fields (e.g., username, email) transferred between APIcontrollers and services.

UserRegistrationDTO: Includes fields for registering new users, validating input data beforepersisting to the database.

DTOs for Product Management:

ProductDTO: Represents product data (e.g., name, price) transferred between controllers andservices.

ProductCreationDTO: Includes fields required for creating new products, ensuring onlynecessary data is transferred and validated.

Benefits:

Clear Data Contracts: DTOs define clear data contracts between different parts of theapplication, improving maintainability and readability.

Enhanced Testability: By decoupling layers and using DTOs, unit tests become easier towrite and maintain.

Adaptability: DTOs can adapt to changes in data structures or requirements without affectingthe entire application.

In summary, DTOs play a crucial role in ASP.NET Core applications by facilitating efficient,secure, and decoupled data transfer between layers, contributing to overall application robustness and scalability.

Serialization and Deserialization in ASP.NET Core Web API

Serialization

Serialization is the process of converting an object into a format that can be easily transmitted over a network or stored. ASP.NET Core typically uses System.Text.Json or Newtonsoft.Json for this purpose.

Deserialization

Deserialization is the reverse process, where data is converted back into an object. This can introduce security risks if not handled correctly, particularly if the data comes from untrusted sources.

Configuration

Set up the development environment, ensuring the dependencies are installed. Configure the appsettings.json file with initial settings, including JWT configuration and database connections. Proper configuration is crucial for the smooth functioning of the application.

Example Configurations

Database Connection Configuration:

- Use ConnectionStrings section to configure database connections, specifying provider (SqlServer, PostgreSQL, etc.), server address, database name, credentials, and additional options.

```
"ConnectionStrin
 gs": {
 "DefaultConnec
 tion":
"Server=YourServer;Database=YourDatabase;User=YourUser;Pass
word=YourPassword;
```

"

}

Security Configuration using middleware:

Middleware Details

Clickjacking Solutions

1. **X-Frame-Options Header**:
    - Adds X-Frame-Options header with value SAMEORIGIN to prevent clickjacking by only allowing the page to be displayed in a frame on the sameorigin as the page itself.
2. **Content-Security-Policy Header for Frames**:
    - Alternatively, adds Content-Security-Policy header with frame- ancestors 'none'to prevent the page from being displayed in a frame.

x-**Content-Type-Options Header**

- Adds X-Content-Type-Optionsheader with value nosniffto prevent MIME typesniffing, enhancing security by ensuring that browsers only interpret files as the specified MIME type.

x-**Powered-By Header**

- Removes the X-Powered-Byheader to obscure server information and reduces thepotential attack surface by not revealing the server technology used.

Content-Security-Policy Header

- Adds Content-Security-Policyheader with several directives:
    - object-src 'none': Disallows loading any plugins like Flash, Silverlight,etc.
    - base-uri 'self': Restricts the document base URL to the same origin.
    - script-src 'nonce-{nonce}' 'strict-dynamic' 'report-sample' https:: Allows scripts with a dynamically generated nonce and from securesources.

- report-uri https://csp.withgoogle.com/csp/gws/fff: Specifies theendpoint for CSP violation reports.

3. Implementing Buyer and Seller Roles

Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) restricts system access to authorized users. RBAC is crucial for e-commerce systems to ensure that users can only perform actions permitted bytheir roles. Define roles clearly to prevent unauthorized access and actions.

Buyer Role Functions

Detail the functions available to buyers, such as browsing products, adding items to the Wishlist, updating it password. Provide a user journey from product discovery to purchase completion.

Additional Buyer Features

Discuss additional features for buyers, such as providing reviews and ratings for purchased products. These features enhance user interaction and trust in the platform. Use visual aids toshow how these features are integrated into the user interface.

Seller Role Functions

Outline the functions available to sellers, including adding new products, updating productinformation, and managing inventory.

4. Configuring JWT Authentication

Understanding JWT

JWT (JSON Web Token) is a compact, URL-safe means of representing claims to be transferred between two parties. JWTs are used for securely transmitting information between parties as a JSON object. They are commonly used in authentication and authorization mechanisms in web applications.

JWT Configuration

Configure JWT in ASP.NET Core by setting up token parameters such as issuer, audience, and secret key. These parameters are essential for generating and validating tokens. Proper configuration ensures only authenticated users can access specific API endpoints.

Securing Endpoints

Apply the [Authorize] attribute to secure API endpoints, ensuring only authenticated users can access them. Provide examples of how to secure endpoints and explain the importance of endpoint security in protecting sensitive data and functionalities.

Token Generation and Validation

1. **Authentication Process**:
    - When a user successfully logs in to the application, typically using credentials (username/password), the server verifies the credentials.
    - Upon successful verification, the server creates a JWT token containing claims (key-value pairs) that represent the user's identity and other information.
2. **Token Payload (Claims)**:
    - JWT tokens consist of three parts: Header, Payload (Claims), and Signature.

- **Claims**: These are pieces of information about the user (e.g., username, roles,permissions) and metadata (e.g., expiration time) encoded into the token.
- Example of a JWT payload:

```
Json
{
 "sub": "Mohamed123",
 "name": "Mohammed",
 "role":
 "Buyer",
 "expDay"
 : 10
}
```

3. **Token Signing**:
   - After creating the payload, the server signs the JWT token using a secret keyor private key (for RS256 algorithm).
   - This signature ensures that the token has not been tampered with. Only serverswith access to the secret key can create valid signatures.
4. **Token Issuance**:
   - The signed JWT token is then returned to the client (typically in the responsebody or as a cookie).

- The client stores this token (e.g., in local storage or a cookie) and includes it insubsequent requests to the server.

Validating JWT Tokens

1. **Token Reception**:
   - When a client makes a request to a protected endpoint, it includes the JWTtoken in the request headers (usually in the Authorization header).
2. **Token Parsing**:
   - The server extracts the JWT token from the request header.
   - It then verifies the token's signature using the secret key (or public key forRS256) to ensure it hasn't been altered since issuance.
3. **Token Decoding**:
   - The server decodes the token's payload to access the claims and expirationtime.
4. **Token Expiry Check**:
   - The server checks the exp claim in the payload to ensure the token is still valid(i.e., it hasn't expired).
   - If the token is expired, the server denies access and may prompt the client torefresh the token.
5. **Authorization Check**:
   - Additional checks can be performed based on the token's claims (e.g., role-based access control) to determine if the user has permission to access the requested resource.

Importance of Token Integrity

- **Preventing Unauthorized Access**: JWT tokens are signed and validated with a secretkey, ensuring their integrity. If a token is modified ,the server rejects it, preventing unauthorized access to protected resources.
- **Stateless Authentication**: JWT tokens are stateless, meaning

servers don’t need tostore session data. This scalability benefit makes JWT tokens suitable for microservices architectures and APIs.

- **Efficiency**: JWT tokens are compact and can be transmitted efficiently over networks,reducing overhead compared to traditional session-based authentication.

Best Practices

- **Secure Key Management**: Safeguard the secret key used for signing tokens toprevent unauthorized token creation or tampering.
- **Token Expiry**: Set appropriate expiration times for tokens to balance security andusability. Short-lived tokens minimize the risk of token misuse.
- **Use HTTPS**: Always transmit JWT tokens over HTTPS to prevent interception andtampering during transmission.

By following these principles and implementing JWT authentication correctly, developerscan ensure secure and efficient authentication mechanisms in their ASP.NET Core applications, maintaining the integrity and confidentiality of user data.

Testing JWT Authentication

Demonstrate how to test JWT authentication using tools like Postman, or we can use Swagger to handle this. Provide step-by-step instructions for testing token generation, protected endpoints, and troubleshooting common issues. Testing ensures the authentication mechanism is functioning correctly.

5. Adding Web Scraping Features

Introduction to Web Scraping

Web scraping involves extracting data from websites. It is used to gather information such asproduct prices, reviews, and availability. While web scraping is powerful, it comes with ethical and legal considerations. Ensure compliance with website terms of service and privacy policies.Online suggestions are derived from real-time data gathered through web scraping. These suggestions include trending products, latest reviews, and price comparisonsfrom various e-commerce websites. Online suggestions provide users with up-to-date and relevant information.

HtmlAgilityPack Overview

HtmlAgilityPack is a .NET library used for parsing HTML documents. It simplifies the process of extracting data from web pages. Install and set up HtmlAgilityPack in your projectto leverage its capabilities for web scraping tasks.

Implementing Web Scraping

Describe the basics of web scraping, including fetching HTML content and parsing it to extract useful information. Provide examples of scraping data from e-commerce websites, focusing on extracting product information, reviews, and prices. store and process scraped data efficiently. storage options, such as databases, and data processing techniques to cleanand prepare data for use in the application. Emphasize the importance of data accuracy andrelevance.

 Construct **the Amazon search URL**: The GetAmazonSuggestions method constructs a searchURL using the base Amazon Egypt URL

and the provided product name.

 HTTP **Client and HTML Document setup**: It uses HTTP Client to send an HTTP GET request to the constructed URL and retrieves the HTML response. The HTML Document fromHtmlAgilityPack is used to load and parse the HTML content.

 XPath **for product details**: The method uses XPath to select product nodes from theHTML document. For each product node, it extracts:

- **Title**: The product title.
- **Link**: The product link, ensuring it is a full URL.
- **Price**: The product price.
- **Image**: The URL of the product image.

 Error **handling**: If any part of the process fails, the method catches exceptions andrethrows them with a relevant error message.

6. Handling Offline Suggestions

Offline Suggestions: Offline suggestions are generated based on historical data and user behavior within the system. These suggestions include recommending products based on past purchases, suggesting similar products, and highlighting popular products in a specific category.

Examples of Offline Suggestions

Describe common algorithms used for generating offline suggestions, such as collaborative filtering and content-based filtering, controller class in an ASP.NET Core application designed to provide offline product suggestions and seller locations from a local database. It offers endpoints to fetch all shops and to get seller locations based on a product name or description.

Our Implementation:

1. **GetAllShops**: This method retrieves all sellers from the database and returns them inthe response.
2. **GetSellerLocations**:
    - Searches for product names or descriptions matching the provided productName Or Description in multiple product categories (Beauty, Books, Electronics, Fashion, Gaming, Sports).
    - Collects the emails of sellers whose products match the search criteria.
    - Retrieves seller details (first name, last name, email, longitude, latitude) forsellers whose emails were found in the product search.
    - Returns the seller locations in the response. If no sellers are found, it returns amessage "No Products Found".

Helper Classes

GraduationDataBaseContext

- **Purpose**: Represents the database context used to interact with the database. Itincludes DbSet properties for each product category and sellers.

Example Usage

- **GetAllShops**: To fetch all shops, make a GET request to api/OfflineSuggestion.The response will include a list of all sellers in the database.
- **GetSellerLocations**: To fetch seller locations based on a product name or description,make a GET request to api/OfflineSuggestion/GetSellerLocations?productNameOrDescript ion={va lue}, replacing {value} with the desired product name or description. The response will include the locations of sellers whose products match the search criteria. If no matches are found, the response will indicate "No Products Found".

7. Custom search based on category type

The CustomSearchBasedOnCategoryController is a controller class in an ASP.NET Core application that provides custom search functionality for different product categories. It
contains endpoints to search for products within specific categories, allowing for more efficientand targeted searches compared to a global search across all categories.

Benefits of Custom Category Search

- **Improved Performance**: Searching within specific categories reduces the amount ofdata processed, leading to faster response times compared to global searches across the entire database.
- **Relevant Results**: By targeting specific categories, the search results are morerelevant and tailored to the user's needs.
- **Resource Efficiency**: Reduces the load on the database by limiting the scope of thesearch queries, thus improving overall system performance.
- **Scalability**: As the database grows, category-specific searches remain efficient,whereas global searches may become slower and more resource-intensive.

Example Usage

- **Beauty Products Search**: To search for beauty products, make a GET request to api/CustomSearchBasedOnCategory/Beauty?query={value}, replacing {value}with the desired search term. The response will include a list of beauty products that match the search term.
- **Books Products Search**: To search for books, make a GET request to api/CustomSearchBasedOnCategory/Books?query={value},

replacing {value}with the desired search term. The response will include a list of book products that match the search term.

- **Fashion Products Search**: To search for fashion products, make a GET request to api/CustomSearchBasedOnCategory/Fashion?query={value}, replacing {value}with the desired search term. The response will include a list of fashion products that match the search term.
- **Electronics Products Search**: To search for electronics products, make a GET request to api/CustomSearchBasedOnCategory/Electronics?query={val ue},replacing {value}with the desired search term. The response will include a list ofelectronics products that match the search term.
- **Gaming Products Search**: To search for gaming products, make a GET request to api/CustomSearchBasedOnCategory/Gaming?query={value}, replacing {value}with the desired search term. The response will include a list of gaming products thatmatch the search term.
- **Sports Products Search**: To search for sports products, make a GET request to api/CustomSearchBasedOnCategory/Sports?query={value}, replacing {value}with the desired search term. The response will include a list of sports products that match the search term.

8. Project Overview and Summary

This project is an e-commerce system designed to facilitate online and offline transactions between buyers and sellers. It is built using ASP.NET Core Web API and integrates JWT authentication to ensure secure access. The project includes web scraping capabilities using HtmlAgilityPack to gather product information and suggestions.

Key Features

1. **User Roles and Authentication:**
   - **JWT Authentication:** The system uses JSON Web Tokens (JWT) to authenticate users, ensuring secure access to the API.
   - **User Roles:** Two main roles are defined in the system: Buyer and Seller, each with specific functions and permissions.
2. **Seller Features:**
   - **Registration and Login:** Sellers can register and log in to the system.
   - **Product Management:** Sellers can add, update, and delete products in their catalog.
3. **Buyer Features:**
   - **Registration and Login:** Buyers can register and log in to the system.
   - **Product Search and Browsing:** Buyers can search for and browse products available in the catalog.
4. **Web Scraping for Suggestions:**
   - **HtmlAgilityPack Integration:** The system uses HtmlAgilityPack to scrape product information from various sources.
   - **Offline and Online Suggestions:** The scraped data is used to provide suggestions for both online and offline shopping, enhancing the user's purchasing experience.
5. **API Documentation:**
   - **Swagger Integration:** The project includes Swagger for

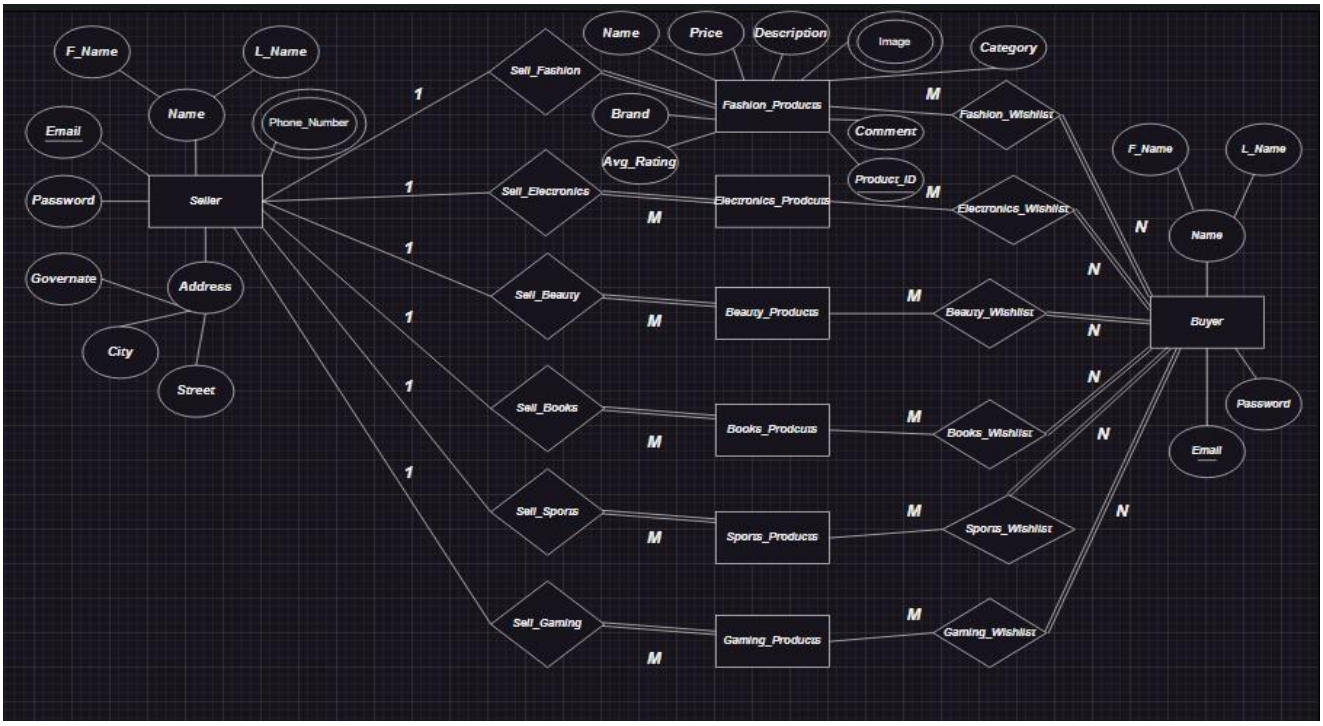
API documentation, making it easy for developers to understand and use the API.
    - **Detailed Endpoints:** Each endpoint is documented with details about the request and response formats, parameters, and expected outcomes.
6. **Security Features:**
    - **Secure Data Transmission:** The system ensures secure data transmission using HTTPS.
    - **Data Validation and Sanitization:** Inputs are validated and sanitized to prevent security vulnerabilities such as SQL injection and cross-site scripting (XSS).
7. **Scalability and Performance:**
    - **Efficient Data Handling:** The system is designed to handle large volumes of data efficiently.
    - **Scalable Architecture:** The architecture is scalable, allowing for easy expansion and integration with additional features or services.
8. **User Experience:**
    - **Responsive Design:** The system is designed to be responsive, ensuring a seamless user experience across different devices.
    - **User-Friendly Interface:** The interfaces for both buyers and sellers are intuitive and easy to navigate.

9. Conclusion

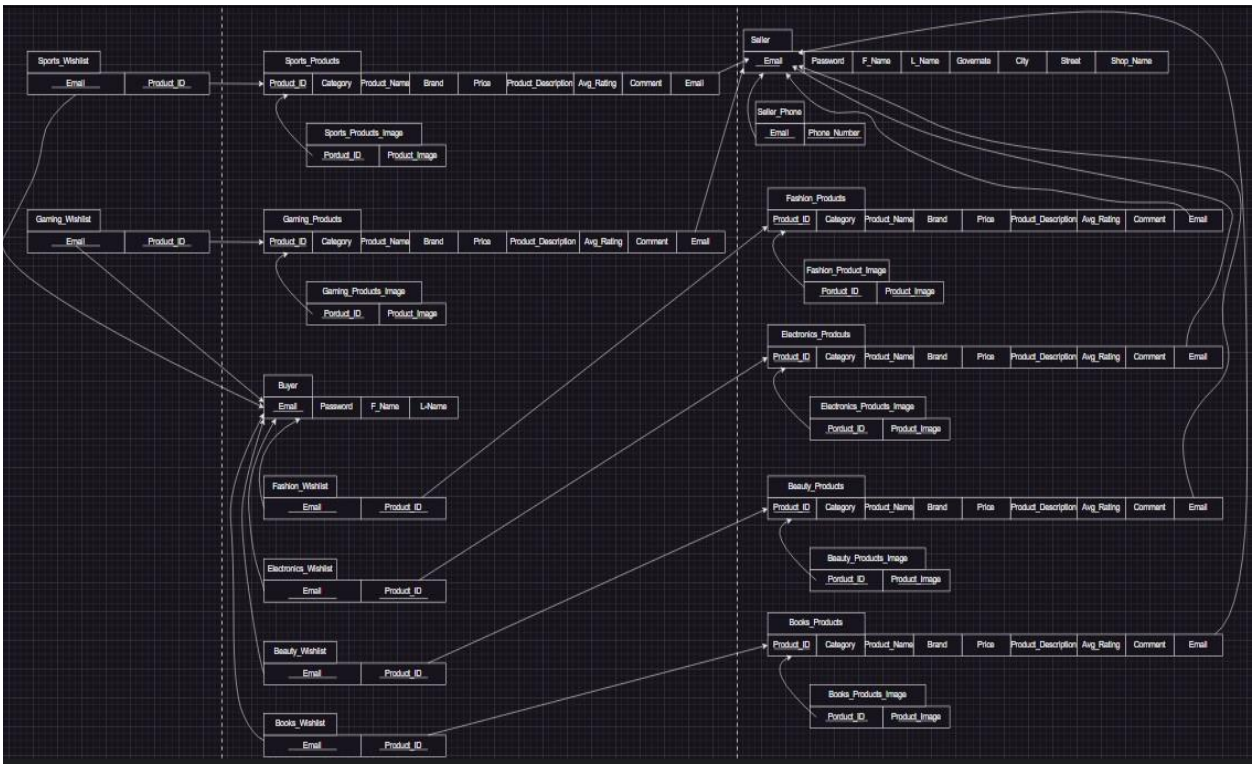
This guide has provided a comprehensive overview of developing a secure and efficient e-commerce system using ASP.NET Core Web API. We covered essential topics such as projectsetup, JWT authentication, role-based access control, web scraping for online suggestions, handling offline suggestions, and implementing custom search based on category type. By following this guide, you should have a solid foundation for building and enhancing an e-commerce application that caters to both buyers and sellers while ensuring data security andrelevance.

Security Rules
In The Project

The ER Digram :



Relational Schema:



1-Database Security Layer :

After Creating the whole database there are some steps to ensure our database are secured and safe from cyber attacks :

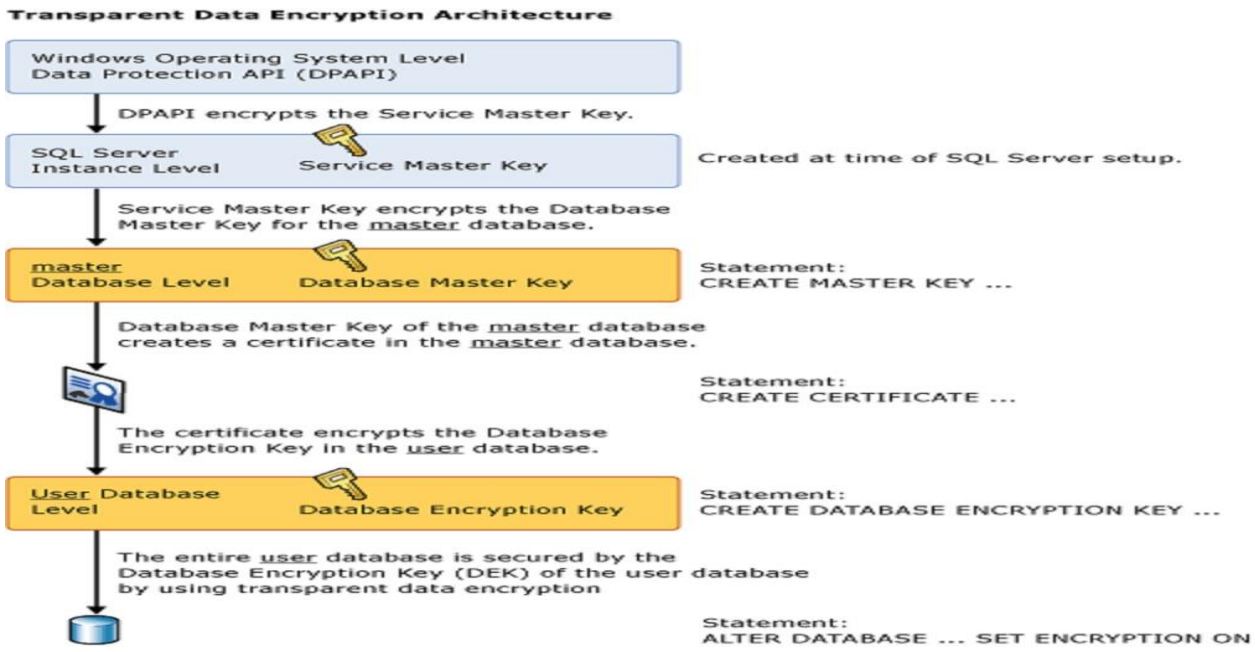
1-need to enable encryption with the whole database while data at not moving or at disk using : TDE

2-need to ensure the encryption the data while transfer using : SSL/TLS encryption (in transport layer)

Transparent Data Encryption (TDE) in SQL Server is a feature that encrypts the entire database at rest. Here's an overview of how TDE works and how to implement it:

How TDE Works:

1. **Encryption at Rest:** TDE encrypts the storage of an entire database (data and log files) at the file level. This means the data is encrypted on disk, which helps protect it from unauthorized access if the physical media (disk files) are stolen.
2. **Transparent Operation:** TDE operates transparently to applications accessing the database. When data is written to disk, it's automatically encrypted. Similarly, when data is read from disk, it's automatically decrypted.
3. **Encryption Hierarchy:** TDE uses a two-tiered encryption hierarchy:
    - The Database Encryption Key (DEK), which is used to encrypt the database.
    - The Service Master Key (SMK), which encrypts the DEK.
4. **Key Management:**
    - The SMK is automatically generated when SQL Server is installed.
    - You can configure TDE to use a certificate or an asymmetric key stored in the database for encrypting the DEK.
    - Backup of the SMK and DEK is essential for recovery scenarios.



5. The image depicts a layered approach to data encryption within a SQL Server database, utilizing multiple keys and certificates. The process starts at the Windows Operating System Level where the Data Protection API (DPAPI) encrypts the Service Master Key. This key is generated at the time of SQL Server setup.

Moving to the SQL Server Instance Level, the Service Master Key encrypts the Database Master Key, which is stored in the master database. The Database Master Key, in turn, creates a certificate within the master database. This certificate is used to encrypt the Database Encryption Key (DEK), which is stored in the user database. The DEK secures the entire user database, ensuring data confidentiality.

The process involves various SQL statements:

- **CREATE MASTER KEY**: Creates the Database Master Key.
- **CREATE CERTIFICATE**: Creates the certificate used for encrypting the DEK.
- **CREATE DATABASE ENCRYPTION KEY**: Generates the DEK for the user database.
- **ALTER DATABASE... SET ENCRYPTION ON**: Enables transparent data encryption for the user database, utilizing the DEK.

This layered approach ensures that data is secured at multiple levels, enhancing the overall security of the database.

Implementing TDE:

*--Create Master Key &&

*--bay attention the password her used for encryption and decryption when we backup the database

*--use strong password = sguz>4DLv3-K"Fb7cXHSNW

use master;

GO

BACKUP DATABASE [project] TO DISK = N'C:\back-up\databak.bak';

--Create Master Key &&

--bay attention the password her used for encryption and decryption when we backup the database

--use strong password = sguz>4DLv3-K"Fb7cXHSNW

use master;

GO

BACKUP DATABASE [project] TO DISK = N'C:\back-up\databak.bak';

After I get my database restore back not start the process :

1-**Create Master Key:**

USE master;

Go

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'U`m6(zx84N#~P{G2;L<Env';

GO

This step creates a master key in the' master' database. It's important to securely manage this password as it's used for encrypting and decrypting the master key itself.

2- **Create Certificate:**

```
--Create Certificate
CREATE CERTIFICATE Project_Cert WITH SUBJECT = 'Project Certificate'
GO
```

This creates a certificate named Project_Cert in the current database (project).

3- **Create Database Encryption Key (DEK):**

```
--Create Encryption Key , Encrypted by Server Certificate.

Use project;
Go
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE Project_Cert;
GO
```

This step creates the DEK for the project database, encrypting it using the Project_Cert certificate.

--after that step =Warning: The certificate used for encrypting the database encryption key has not been backed up. You should immediately back up the certificate and the private key associated with the certificate. If the certificate ever becomes unavailable or if you must restore or attach the database on another server, you must have backups of both the certificate and the private key or you will not be able to open the database.

4- **Backup Certificate and Private Key:**

--Backup Certificate && same password used above

Use master;

Go

BACKUP CERTIFICATE Project_Cert

To File = 'C:\back-up\cert\Project_Cert.cer'

WITH PRIVATE Key (File = 'C:\back-up\cert\Project_Cert_Key.pvk' , ENCRYPTION BY PASSWORD = 'U`m6(zx84N#~P{G2;L<Env')

GO

You correctly back up the certificate Project_Cert along with its private key. This backup is crucial for restoring or attaching the database on another server.

5- **Enable Encryption:**

ALTER DATABASE projectSET ENCRYPTION ON;

GO

This command enables TDE for the project database using the created DEK.

6- **Verify Certificate and Encryption Status:**

USE master

SELECT * FROM sys.certificates WHERE pvt_key_encryption_type <> 'NA';

GO

This query verifies the existence of certificates and their private key encryption status.

use master

SELECT encryptor_type, key_length , key_algorithm ,encryption_state ,create_date

FROM sys.dm_database_encryption_keys;

GO

This query verifies the encryption state and details of database encryption keys.

7- **Take Encrypted Backup:**

--here after all this steps the file that you have take a backup in not encrypted so get another backup

--with new encryption with the certificate

use master;

GO

BACKUP DATABASE [project] TO DISK = N'C:\back-up\databaknew with encrytion\fullbackup.bak' WITH NOFORMAT,NOINIT,NAME = N'FullBackUp',

SKIP,NOREWIND,NOUNLOAD,STATS=10 ;

GO

Finally, you take a new backup of the project database with encryption enabled, ensuring that the backup itself is encrypted.

restore a database backup that was encrypted using Transparent Data Encryption (TDE) on a different SQL Server instance

1- **Restore Database Backup:**

use master

go

restore database [project]

from disk = N'C:\back-up\databaknew with encrytion\fullbackup.bak';

This command attempts to restore the project database from the encrypted backup file (fullbackup.bak).

2- **Create Master Key on New Instance:**

Since you're restoring to a different SQL Server instance, you need to create a new master key specific to that instance.

```
--create Master Key on other Instance because it is different instance

use master

Go

create master key encryption by password = 'L@7_6GQ>[vf<WnAt^8rUa`';

GO
```

This command creates a new master key on the new SQL Server instance, using a **different password** (L@7_6GQ>[vf<WnAt^8rUa``) than the one used on the original instance (Um6(zx84N#~P{G2;L<Env).

3- **Restore Certificate and Private Key:**

To restore the certificate and private key for decryption:

```
create certificate Project_cert2

from file = N'C:\back-up\cert\Project_Cert.cer'

with private key (file='C:\back-up\cert\Project_Cert_Key.pvk', ENCRYPTION BY PASSWORD = 'U`m6(zx84N#~P{G2;L<Env')

go
```

Here, you should reference the original certificate (Project_Cert) and private key (Project_Cert_Key.pvk) used for encryption on the original instance. Note the

DECRYPTION BY PASSWORD clause should match the password used during the backup process on the original instance (Um6(zx84N#~P{G2;L<Env`).

4- **Attempt Database Restore Again:**

```
USE master;

RESTORE DATABASE [project]

FROM DISK = N'C:\back-up\databaknew with encrytion\fullbackup.bak'

WITH RECOVERY;

GO
```

This command should now attempt to restore the project database using the newly restored master key and certificate that can decrypt the backup.

Notes:

- Make sure the certificate and private key files (Project_Cert.cer and Project_Cert_Key.pvk) are accessible to the new SQL Server instance.
- Ensure the password used for **DECRYPTION BY PASSWORD** matches the password used when the private key was originally backed up.

2- Encryption in Transit: Implement SSL/TLS

Implementing SSL/TLS ensures that data transferred between the client and the SQL Server is encrypted.

2-**Vulnerability Management** & **Threat Modelling :**

- Threat Modelling:

Threat modelling is a systematic approach to **identifying, prioritising, and addressing potential security threats** across the organisation. By simulating possible attack scenarios and assessing the existing vulnerabilities of the organisation's interconnected systems and applications, threat modelling enables organisations to develop proactive security measures and make informed decisions about resource allocation.

Threat modelling aims to reduce an organisation's overall risk exposure by identifying vulnerabilities and potential attack vectors, allowing for adequate security controls and strategies. This process is essential for constructing a robust defence strategy against the ever-evolving cyber threat landscape.

As mentioned above, the main goal of threat modelling is to reduce the organisation's risk exposure. So before we deep dive into its application

Threat	Refers to any potential occurrence, event, or actor that may exploit vulnerabilities to compromise information confidentiality, integrity, or availability. It may come in various forms, such as cyber attacks, human error, or natural disasters.
Vulnerability	A weakness or flaw in a system, application, or process that may be exploited by a threat to cause harm. It may arise from software bugs, misconfiguration, or design flaws
Risk	The possibility of being compromised because of a threat taking advantage of a vulnerability. A way to think about how likely an attack might be successful and how much damage it could cause.

High-Level Process of Threat Modelling

Before delving into different threat modelling frameworks, let's briefly run through a simplified, high-level process.

1. **Defining the scope**

Identify the specific systems, applications, and networks in the threat modelling exercise.

2. **Asset Identification**

Develop diagrams of the organisation's architecture and its dependencies. It is also essential to identify the importance of each asset based on the information it handles, such as customer data, intellectual property, and financial information.

3. **Identify Threats**

Identify potential threats that may impact the identified assets, such as cyber attacks, physical attacks, social engineering, and insider threats.

4. **Analyse Vulnerabilities and Prioritise Risks**

Analyse the vulnerabilities based on the potential impact of identified threats in conjunction with assessing the existing security controls. Given the list of vulnerabilities, risks should be prioritised based on their likelihood and impact.

5. **Develop and Implement Countermeasures**

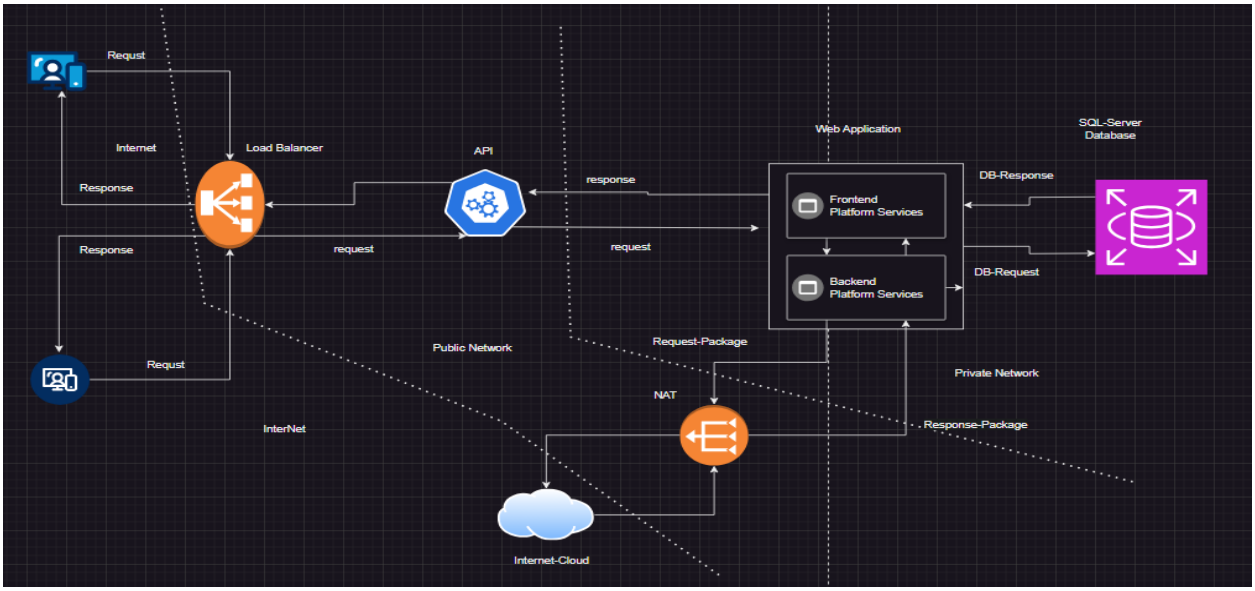
Design and implement security controls to address the identified risks, such as implementing access controls, applying system updates, and performing regular vulnerability assessments.

6. **Monitor and Evaluate**

Continuously test and monitor the effectiveness of the implemented countermeasures and evaluate the success of the threat modelling exercise. An example of a simple measurement of success is tracking the identified risks that have been effectively mitigated or eliminated.

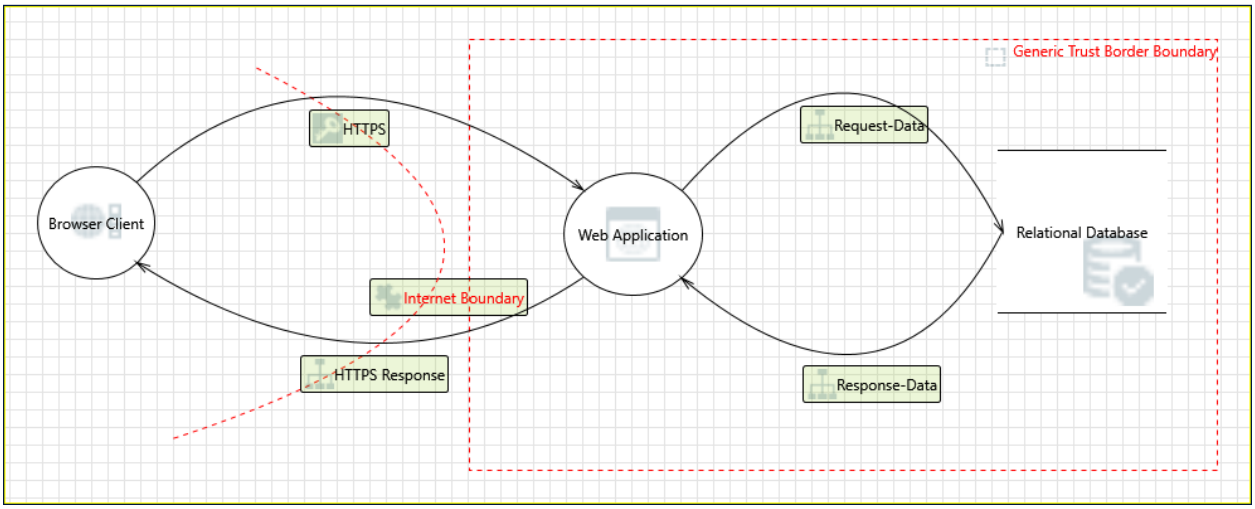
2-Asset Identification :

the network infrastructure



Using MTMT :

Diagram: Diagram 1



Back-end-server

Web-server : 2 containers one for front-end and one for back-end

3-Identify Threats:

Cross-Site Scripting (XSS):

- **Attack**: Injecting malicious scripts into web pages viewed by other users.
- **Mitigation**: Implement input validation and output encoding to sanitize user inputs and prevent script execution.

- **SQL Injection**:
  - **Attack**: Injecting SQL commands through input fields to manipulate the database.
  - **Mitigation**: Use parameterized queries or prepared statements to prevent dynamic SQL construction using user inputs.
- **Cross-Site Request Forgery (CSRF)**:
  - **Attack**: Forcing users to execute unwanted actions on a web application where they are authenticated.
  - **Mitigation**: Use CSRF tokens and validate them on the server side for state-changing requests to ensure requests originate from the expected user.
- **Session Hijacking**:
  - **Attack**: Stealing a user's session ID and impersonating the user.
  - **Mitigation**: Use HTTPS to encrypt session data in transit, employ secure cookies with HttpOnly and Secure flags, and regenerate session IDs upon authentication.
- **Sensitive Data Exposure**:
  - **Attack**: Revealing sensitive data like passwords, credit card details, or personal information due to insecure storage or transmission.
  - **Mitigation**: Encrypt sensitive data both at rest and in transit using strong encryption algorithms and protocols.
- **Remote Code Execution (RCE)**:
  - **Attack**: Running arbitrary code on a server to gain control or disrupt its operations.
  - **Mitigation**: Keep software and libraries updated, sanitize inputs to prevent injection attacks, and apply least privilege principles to limit access.
- **Brute Force Attacks**:
  - **Attack**: Attempting to gain unauthorized access to user accounts by systematically trying different passwords or keys.
  - **Mitigation**: Implement account lockout mechanisms, enforce strong password policies, and use CAPTCHA or rate limiting to thwart automated attacks.

 Clickjacking:

- **Attack**: Tricking a user into clicking on a disguised or invisible link that performs actions unknowingly.
- **Mitigation**: Implement X-Frame-Options or Content Security Policy (CSP) headers to prevent your site from being embedded in an iframe without permission.

 Denial of Service (DoS):

- **Attack**: Overloading a web server or application with excessive requests to disrupt its availability.
- **Mitigation**: Implement rate limiting, use load balancers to distribute traffic, and employ CAPTCHA or token-based authentication to protect against automated bot attacks.

Malicious File Uploads:

- Attackers might upload files containing scripts or malware disguised as photos.
- They could exploit file parsing vulnerabilities to execute malicious code.
- **Mitigation :** Implement strict validation to ensure uploaded files are of expected types (e.g., JPEG, PNG), also Verify file headers and content to prevent uploading of malicious scripts disguised as images .

Data Flow Sniffing [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	Data flowing across HTTP may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. Consider encrypting the data flow.

Mitigation : Implement TLS (Transport Layer Security): Encrypt all sensitive data transmissions using HTTPS (HTTP over SSL/TLS).

3-Analyse Vulnerabilities and Prioritise Risks :

Vulnerability Management

Vulnerability management is an ongoing, proactive, and frequently automated activity that protects computer systems, networks, and enterprise solutions from cyberattacks and data breaches. Consequently, it is a vital component of an overall security program. By discovering, evaluating, and correcting potential security flaws, businesses can help avoid attacks and mitigate their effects if they occur.

 Common Vulnerability Scoring System (CVSS): CVSS is a scoring system that rates the severity of vulnerabilities and identifies their characteristics. It assigns severity scores to all defined vulnerabilities, which is used to prioritise mitigation efforts and the required resources based on the severity. The range of possible scores is 0 to 10, with 10 representing the most severe.

CVSS(3) Score	Severity Rating
0	None
0.1 to 3.9	Low
4.0 to 6.9	Medium
7.0 to 8.9	High
9.0 to 10	Critical

In Vulnerability Management we will interest in Vulnerability Assessment:

Vulnerability Assessment :

a vulnerability assessment is a scan of the vulnerabilities found on networks and applications, it is also faster and has a lighter load on the infrastructure. As opposed to a penetration test, during a vulnerability assessment, you do not proceed to the exploitation phase.

**** Type of attack we concerned with : Application Scaning**

Vulnerability Assessment Lifecycle

1. **Planning** – Define scope, goals, and objectives.
2. **Scanning** – Use tools to discover potential vulnerabilities.
3. **Analysis** – Evaluate results for false positives and significance.
4. **Reporting** – Deliver actionable information to stakeholders.
5. **Remediation** – Address and patch vulnerabilities.
6. **Verification** – Ensure vulnerabilities have been properly addressed.

In this life cycle we test the application in two manner :

1-SAST (Static Application Security Testing)

2-DAST (Dynamic Application Security Testing)

Static Application Security Testing

Static Application Security Testing (SAST) refers to using automated tools for code analysis. The idea is not to replace manual code reviews but to provide a simple method to automate simple code checks to quickly find vulnerabilities during the development process without requiring a specialised individual.

Semgrep is a powerful static analysis tool used primarily for writing and running code patterns to find security vulnerabilities, bugs, and anti-patterns in various programming languages. Here's an abstract overview of Semgrep:

Purpose:

- **Static Analysis**: Semgrep performs static analysis on codebases to detect potential issues without executing the code.
- **Security and Code Quality**: It is commonly used to identify security vulnerabilities, code smells, and adherence to coding standards.

Advantages:

- **Fast Analysis**: Performs quick analysis even on large codebases.

- **Community Support**: Large community contributing rules and providing support.
- **Open Source**: Available under an open-source license, allowing customization and community collaboration

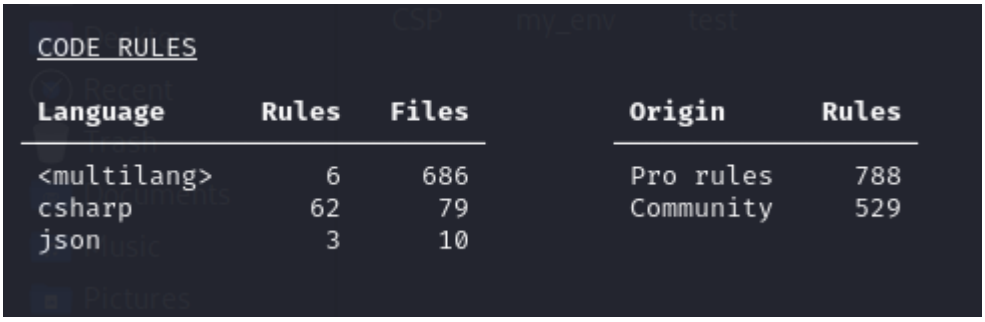
Command used by Semgrep tool :

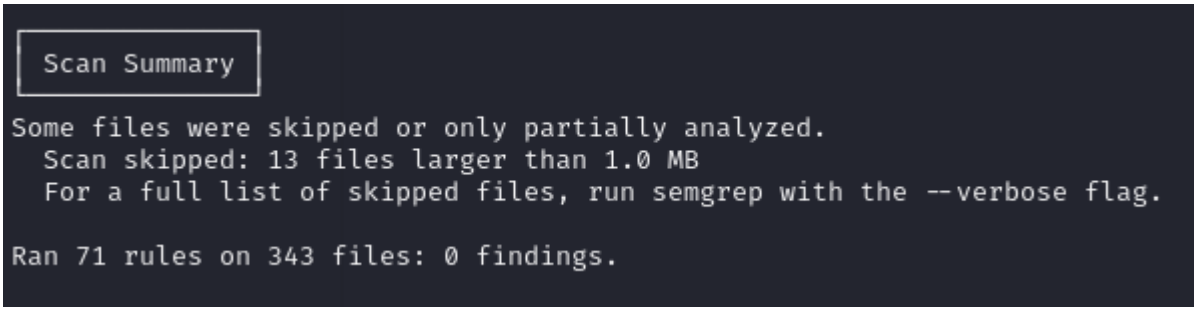
In Back-End Code

owasp-top-ten

The OWASP Top 10 is an industry-recognized report of top web application security risks. Use this ruleset to scan for OWASP Top 10 vulnerabilities.

semgrep --config "p/owasp-top-ten" back-end_folder or .

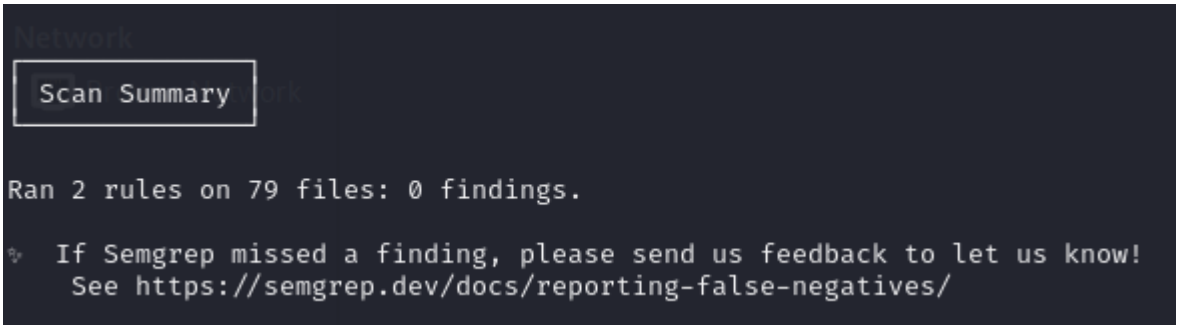




the code is good and all functions has it security measures like sanitizing and filtering the input

Find Command Injection vulnerabilities in your code base.

semgrep --config "p/command-injection"



r2c-security-audit

Scan code for potential security issues that require additional review. Recommended for teams looking to set up guardrails or to flag troublesome spots for further review.

security

audit

xxe

injection

deserialization

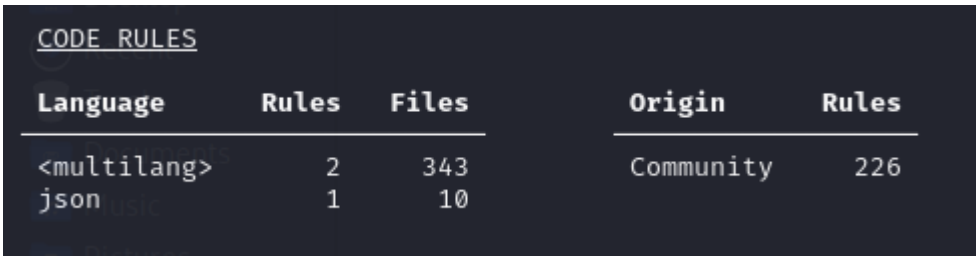
xss

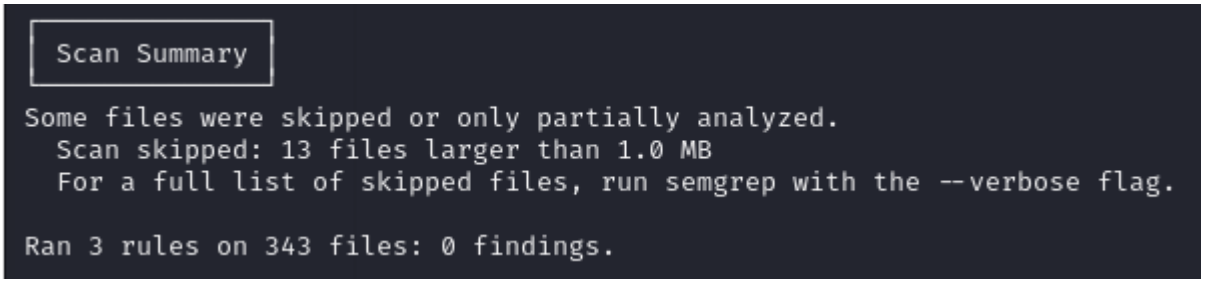
jwt

csrf

crypto

semgrep --config "p/r2c-security-audit"





Run `semgrep ci` to find dependency

vulnerabilities and advanced cross-file findings.

semgrep ci

In front-End code :

1 -package-lock.json file :

ejs - CVE-2024-33883

- **Severity:** MODERATE
- **Description:** ejs < 3.1.10 is vulnerable to Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution').

Fix: **Update ejs to version 3.1.10 :**

npm install ejs@3.1.10

express - CVE-2024-29041

- **Severity:** MODERATE
- **Description:** express < 4.19.2, < 5.0.0-beta.3 are vulnerable to Improper Validation of Syntactic Correctness of Input.
- **Fix:** Update express to version 4.19.2 or 5.0.0-beta.3

npm install express@4.19.2

npm install express@5.0.0-beta.3

follow-redirects - CVE-2024-28849

- **Severity:** MODERATE

- **Description:** follow-redirects <= 1.15.5 is vulnerable to Exposure of Sensitive Information to an Unauthorized Actor.
- **Fix:** Update follow-redirects to version 1.15.6

npm install follow-redirects@1.15.6

ws - CVE-2024-37890

- **Severity:** HIGH
- **Description:** ws < 5.2.4, < 6.2.3, < 7.5.10, < 8.17.1 are vulnerable to NULL Pointer Dereference.
- **Fix:** Update ws to version 5.2.4, 6.2.3, 7.5.10, or 8.17.1

npm install ws@5.2.4

npm install ws@6.2.3

npm install ws@7.5.10

npm install ws@8.17.1

braces - CVE-2024-4068

- **Severity:** HIGH
- **Description:** Affected versions of braces are vulnerable to Excessive Platform Resource Consumption within a Loop.
- **Fix:** Update braces to version 3.0.3

npm install braces@3.0.3

postcss - CVE-2023-44270

- **Severity:** MODERATE
- **Description:** Affected versions of postcss are vulnerable to Improper Neutralization Of Special Elements In Output Used By A Downstream Component ('Injection') / Improper Neutralization Of Line Delimiters.
- **Fix:** Update postcss to version 8.4.31

npm install postcss@8.4.31

nth-check - CVE-2021-3803

- **Severity:** HIGH
- **Description:** nth-check versions before 2.0.1 are vulnerable to Inefficient Regular Expression Complexity when parsing crafted invalid CSS nth-checks.
- **Fix:** Update nth-check to version 2.0.1

npm install nth-check@2.0.1

webpack-dev-middleware - CVE-2024-29180

- **Severity:** HIGH
- **Description:** Affected versions of webpack-dev-middleware are vulnerable to Improper Limitation Of A Pathname To A Restricted Directory ('Path Traversal').
- **Fix:** Update webpack-dev-middleware to version 5.3.4, 6.1.2, or 7.1.0

npm install webpack-dev-middleware@5.3.4

npm install webpack-dev-middleware@6.1.2

npm install webpack-dev-middleware@7.1.0

or to update them all in one time :

npm update -g

SAST findings:

- **ejs**: CVE-2024-33883
- **express**: CVE-2024-29041
- **follow-redirects**: CVE-2024-28849
- **ws**: CVE-2024-37890
- **braces**: CVE-2024-4068
- **postcss**: CVE-2023-44270

- **nth-check**: CVE-2021-3803
- **webpack-dev-middleware**: CVE-2024-29180

after the update in front-end with the same and additional rule for react ;

react

React security rules.

semgrep --config "p/react"

all good

After updated all is clear and good

Dynamic Application Security Testing :

Dynamic Application Security Testing (DAST) is the process of testing a running instance of a web application for weaknesses and vulnerabilities. It focuses on a black-box testing approach where vulnerabilities are found just like a regular attacker would find them. Simply put, DAST identifies vulnerabilities by trying to exploit them, either manually or through automated tools.

By testing the application from an outside perspective, we can abstract ourselves from its inner workings and focus on identifying vulnerabilities that an attacker would likely find. This means the results obtained from DAST will often point to vulnerabilities requiring prioritised attention as they are expected to be found without prior knowledge of the application.

- **Manual DAST:** A security engineer will manually perform tests against an application to check for vulnerabilities.
- **Automatic DAST:** An automated tool will scan the web application for vulnerabilities.

a DAST tool will perform at least the two following tasks against the target website:

- **Spidering/Crawling:** The tool will navigate through the web app, trying to map the application and identify a list of pages and parameters that can be attacked.

- **Vulnerability Scanning:** The tool will try to launch attack payloads against the identified pages and parameters. The user can typically customise the type of attacks to include only the ones relevant to the target application.

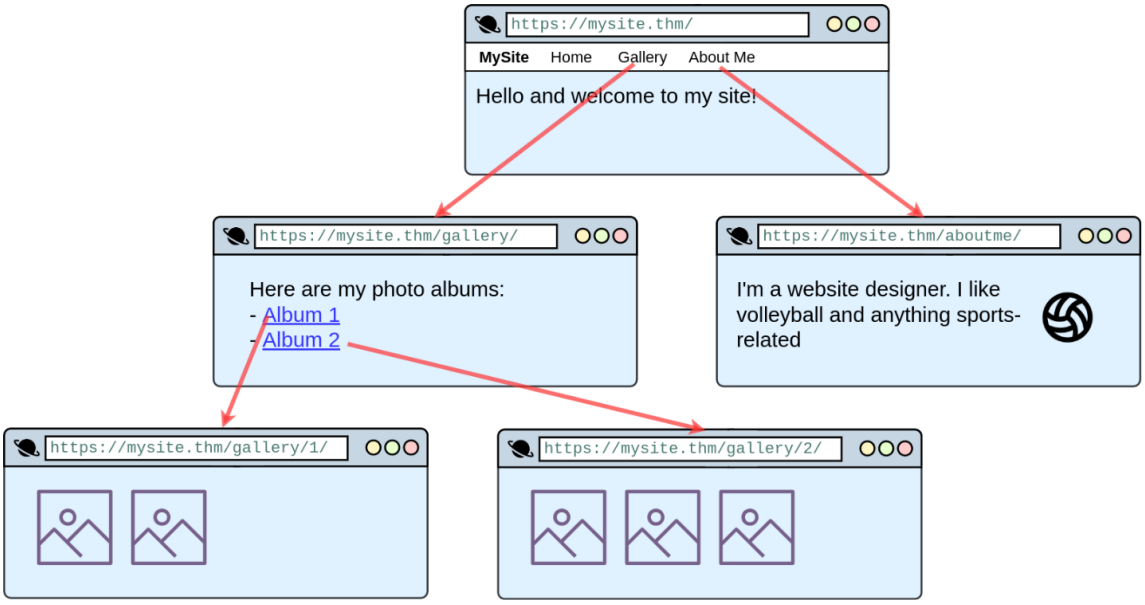
We will use Zap-proxy tool :

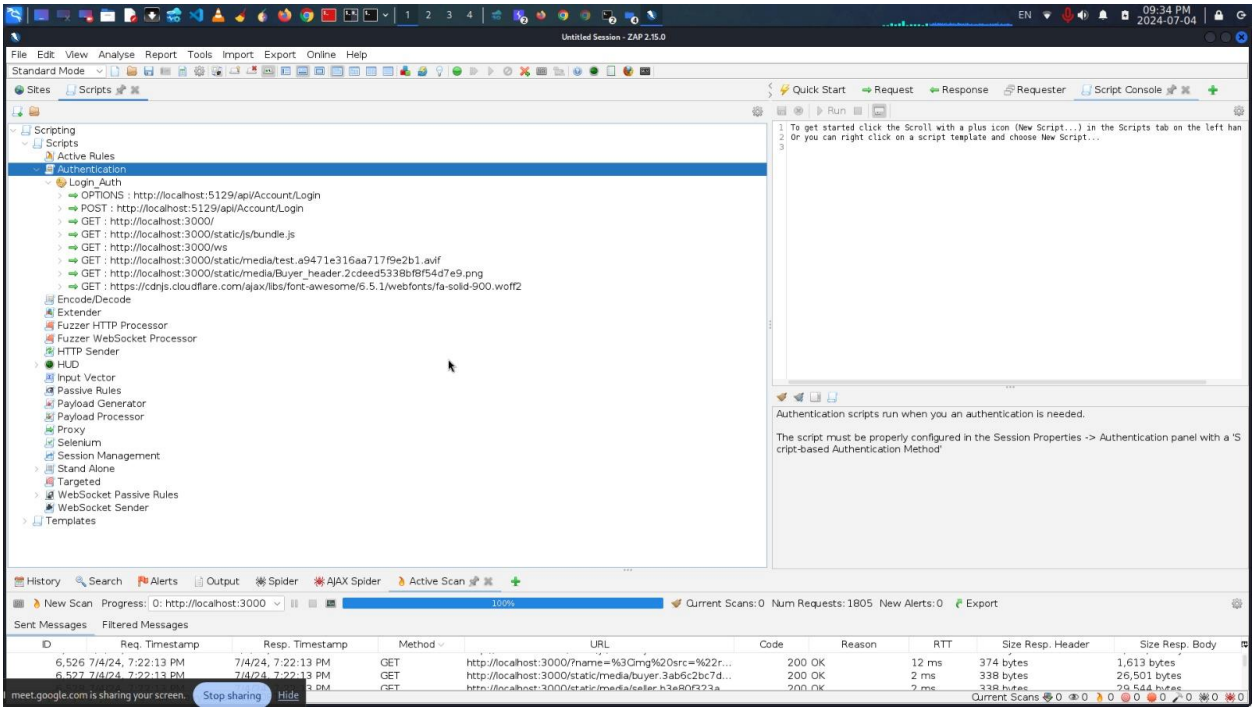
ZAP Proxy, or OWASP ZAP (Zed Attack Proxy), is an open-source security tool used for finding vulnerabilities in web applications during the development and testing phases. Here's a brief overview of ZAP Proxy

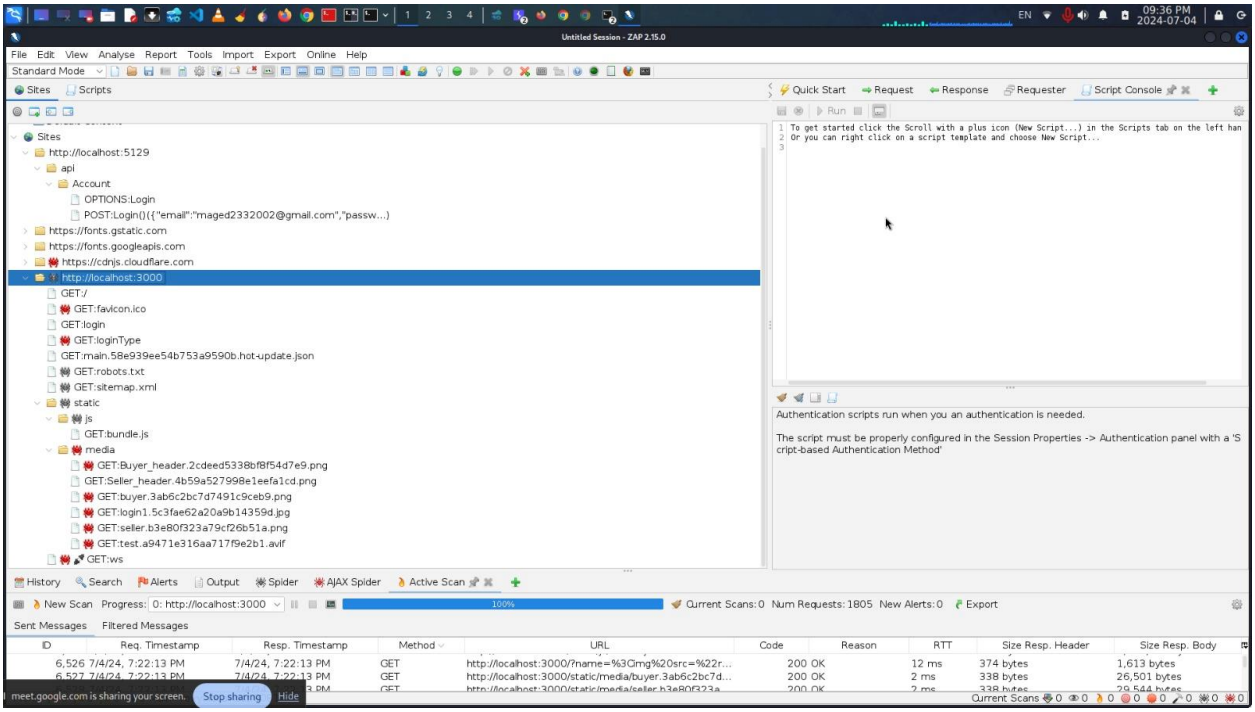
 Vulnerability Assessment: ZAP helps identify security vulnerabilities in web applications through automated scans and manual testing.

 Security Testing: **It supports various testing techniques like automated scanning, fuzzing, and manual testing to uncover security weaknesses.**

Spider will navigate to a starting page you provide and fundamentally explore the website by following all the links it can detect.







After first scan :

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
CSP: Wildcard Directive	Medium	1 (9.1%)
Content Security Policy (CSP) Header Not Set	Medium	11 (100.0%)
Cross-Domain Misconfiguration	Medium	22 (200.0%)
Missing Anti-clickjacking Header	Medium	11 (100.0%)
Private IP Disclosure	Low	1 (9.1%)
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	21 (190.9%)
Timestamp Disclosure - Unix	Low	1 (9.1%)
X-Content-Type-Options Header Missing	Low	20 (181.8%)
Information Disclosure - Suspicious Comments	Informational	12 (109.1%)
Modern Web Application	Informational	11 (100.0%)
Retrieved from Cache	Informational	4 (36.4%)
Total		11

We handle Some of Vulnerability and misconfiguration error :

1-CORS

```
builder.Services.AddCors(options =>
        {
            options.AddPolicy("AllowSpecificOrigin",
                builder => builder
                    .WithOrigins("www.poject.com") // Add the allowed origins here
                    .AllowAnyMethod()
                    .AllowAnyHeader()
                    .AllowCredentials());
        });
```

Security Implications:

- **Preventing Unauthorized Access**: By restricting WithOrigins to a specific trusted origin

Protection Against CSRF: Cross-Origin Resource Sharing helps protect against Cross-Site Request Forgery (CSRF) attacks by requiring a same-origin policy. This prevents malicious scripts from making requests on behalf of the user from other origins.

Data Privacy: Restricting access helps protect sensitive data in your API responses from being read by unauthorized parties, including malicious scripts that might attempt to fetch data via cross-origin requests**.**

2-**Content Security Policy (CSP)** is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

The Configuration in Front-end and back-end

Front-end :

```
<meta http-equiv="Content-Security-Policy" content="http:192.168.1.8:9000 frame-ancestors 'self'; script-src 'self' 'unsafe-eval' fonts.googleapis.com cdnjs.cloudflare.com; ../src/index.css  'self' 'unsafe-inline' fonts.googleapis.com;">
 <meta charset="utf-8" />
 <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"
 />
```

Back-end:

```
  app.Use(async (context, next) =>
        {
           context.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
           await next();
        });
```

2-Missing Anti-clickjacking Header :

The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.

Clickjacking is **an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element**.

this is parameter is missing :x-frame-options : SAMEORIGIN

the correct in the back-end :

```
app.Use(async (context, next) =>
{
   context.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
   await next();
});
```

3-X-Content-Type-Options Header Missing :

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

Parameter **x-content-type-options** is missing

```
#region X-Content-Type-Options header
app.Use(async (context, next) =>
   {
      context.Response.Headers.Add("X-Content-Type-Options", "nosniff");
      await next();
   });
#endregion
```

x-content-type-options : nosniff

we repeat the life cycle : we found :

Alert type	Risk	Count
Absence of Anti-CSRF Tokens	Medium	1 (3.6%)
CSP: Wildcard Directive	Medium	8 (28.6%)
CSP: script-src unsafe-eval	Medium	1 (3.6%)
CSP: style-src unsafe-inline	Medium	1 (3.6%)
Content Security Policy (CSP) Header Not Set	Medium	4 (14.3%)
Cross-Domain Misconfiguration	Medium	44 (157.1%)
Format String Error	Medium	4 (14.3%)
Missing Anti-clickjacking Header	Medium	4 (14.3%)
Application Error Disclosure	Low	1 (3.6%)
CSP: Notices	Low	1 (3.6%)
Cookie with SameSite Attribute None	Low	1 (3.6%)
Cross Site Scripting Weakness (Persistent in JSON Response)	Low	5 (17.9%)
Cross-Domain JavaScript Source File Inclusion	Low	1 (3.6%)
		1

4-Server Leaks Information via "X-Powered-By" HTTP Response Header Field

The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.

Evidence : X-Powered-By: Express

```
// Remove the X-Powered-By header
        #region X-Powered-By header
        app.Use(async (context, next) =>
          {
```

```
        context.Response.Headers.Remove("X-Powered-By");
        await next();
    });
#endregion
```

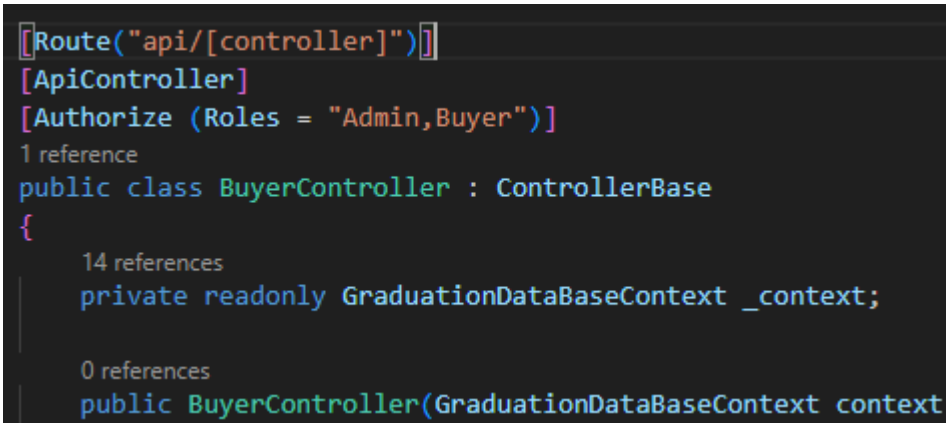
Why Remove "X-Powered-By"?

1. **Information Disclosure**: Revealing specific software versions or technology stacks can aid attackers in identifying vulnerabilities or exploits that target those versions.
2. **Reducing Attack Surface**: Limiting the amount of information disclosed helps to reduce the attack surface and make it harder for attackers to gather intelligence about your infrastructure.

5- CSP: script-src unsafe-eval :

Alert description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

The API before solution



After solution

```
[Route("api/[controller]")]
```

```
[ApiController]
[ValidateAntiForgeryToken]
[Authorize (Roles = "Admin,Buyer")]
public class BuyerController : ControllerBase
{
    private readonly GraduationDataBaseContext _context;

    public BuyerController(GraduationDataBaseContext context)
    {
        _context = context;
    }

    // GET: api/buyer/example@g.com
    [HttpGet("{email}")]
    public async Task<ActionResult<Buyer>> GetBuyer(string email)
    {
        var buyer = await _context.Buyers.FindAsync(email);

        if (buyer == null)
        {
            return NotFound();
        }

        return buyer;
    }
```

At last :

We build our threat-model based on STRIDE model :

What is the STRIDE Framework?

The STRIDE framework is a threat modelling methodology also developed by Microsoft, which helps identify and categorise potential security threats in software development and system design. The acronym STRIDE is based on six categories of threats, namely:

Category	Definition	Policy Violated
Spoofing	Unauthorised access or impersonation of a user or system.	Authentication
Tampering	Unauthorised modification or manipulation of data or code.	Integrity
Repudiation	Ability to deny having acted, typically due to insufficient auditing or logging.	Non-repudiation
Information Disclosure	Unauthorised access to sensitive information, such as personal or financial data.	Confidentiality
Denial of Service	Disruption of the system's availability, preventing legitimate users from accessing it.	Availability
Elevation of Privilege	Unauthorised elevation of access privileges, allowing threat actors to perform unintended actions.	Authorisation

1. Asset Identification

- **Database Server:** Contains critical user data.
- **Back-end Container:** Manages business logic and database interactions.
- **Front-end Container:** User interface and client-side logic.

2. Threat Identification

Spoofing

- **Examples:**
    - Sending an email as another user.
    - Creating a phishing website mimicking a legitimate one to harvest user credentials.
- **Mitigation:** Strong authentication mechanisms implemented.

Tampering

- **Examples:**
    - Updating the password of another user.
    - Installing system-wide backdoors using elevated access.
- **Mitigation:** Data integrity checks implemented to detect and prevent unauthorized modifications.

Repudiation

- **Examples:**
    - Denying unauthorized money-transfer transactions without proper audit trails.
    - Denying sending an offensive message without proof of reception.
- **Mitigation:** Comprehensive logging and monitoring to track access and changes.

Information Disclosure

- **Examples:**
    - Unauthenticated access to a misconfigured database containing sensitive customer information.
    - Accessing public cloud storage handling sensitive documents.
- **Mitigation:** Transparent Data Encryption (TDE), strict access controls, and secure configurations.

Denial of Service

- **Examples:**

    - Flooding a web server with requests to overwhelm resources and make it unavailable.
    - Deploying ransomware that encrypts system data, preventing access to resources.
- **Mitigation:** Implementing rate limiting, load balancing, and robust DDoS protection mechanisms.

Elevation of Privilege

- **Examples:**
    - Creating a regular user account and gaining access to the administrator console.
    - Exploiting unpatched systems to gain local administrator privileges.
- **Mitigation:** Implementing least privilege access controls and regularly updating systems with security patches.

3. Mitigation Strategies

Database Server

- Transparent Data Encryption (TDE), Hashing User Passwords, Backup Encryption.

Back-end and Front-end Containers

- Content Security Policy (CSP), X-Content-Type-Options, X-Frame-Options, Remove X-Powered-By header.
- [ValidateAntiForgeryToken] added to every API endpoint.
- Updated vulnerable packages to mitigate supply chain risks.

4. Conclusion

- Summary of key findings, actions taken, and recommendations for ongoing security improvements

After remediation and address the vulnerabilities :

We need to : Verification ensure vulnerabilities have been properly addressed

Contents

About this report

Report parameters

Contexts

No contexts were selected, so all contexts were included by default.

Sites

The following sites were included:

- https://cdnjs.cloudflare.com
- http://localhost:3000

(If no sites were selected, all sites were included by default.)

An included site must also be within one of the included contexts for its data to be included in the report.

Risk levels

Included: High, Medium, Low, Informational

Excluded: None

Confidence levels

Included: User Confirmed, High, Medium, Low

Excluded: User Confirmed, High, Medium, Low, False Positive

No alerts were found within the report parameters.

It is the last scan it is all clear and the Vulnerability Assessment lifecycle done it operation

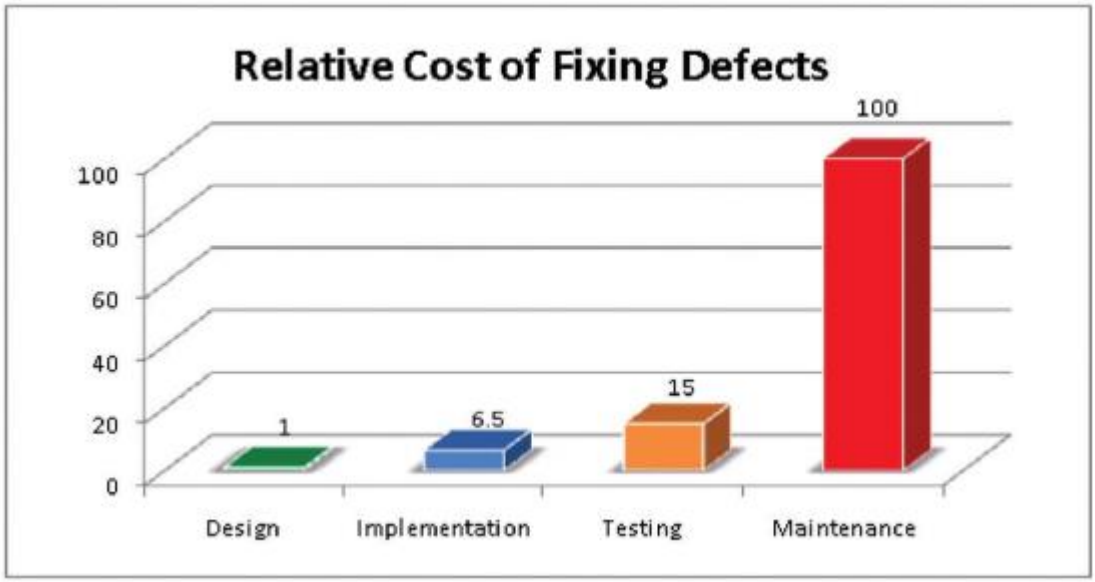
Now we are done our work on putting the security layers in each phase in the SDLC to be SSDLC

Secure Software Development Lifecycle (SSDLC)

During SDLC, security testing was introduced very late in the lifecycle. Bugs, flaws, and other vulnerabilities were identified late, making them far more expensive and time-consuming to fix. In most cases, security testing was not considered during the testing phase, so end-users reported bugs after deployment. Secure SDLC models aim to introduce security at every stage of the SDLC.

Why is Secure SDLC important?

A study conducted by the Systems and Sciences institute at IBM discovered that it costs six times more to fix a bug found during implementation than one identified as early as during the design phase. It also reported that it costs 15 times more if flaws are identified during testing and up to 100 times more costly if identified during the maintenance and operation phases. See the figure below:


Relative Cost of Fixing Defects
100
80
60
40
20
0
1
6.5
15
100
Design
Implementation
Testing
Maintenance

Cloud
And
DevOps work

Technologies Used

- **Infrastructure:**
    - ***AWS (Amazon Web Services)***: Our infrastructure is fully hosted on AWS, a comprehensive cloud computing platform offering a wide range of services, including computing power, storage, and networking capabilities and here is a list of some of the services we used:
        - ***VPC***: Amazon Virtual Private Cloud (AWS VPC) is a service that allows you to create a logically isolated network within the AWS cloud, enabling you to define and control a virtual networking environment. With AWS VPC, you can launch AWS resources, such as EC2 instances, into a virtual network that you've defined, providing complete control over your network configuration, including IP address ranges, subnets, route tables, and network gateways.
        - ***AWS EC2***: Amazon Elastic Compute Cloud (AWS EC2) is a web service that provides resizable compute capacity in the cloud, allowing you to quickly scale up or down based on your computing needs.
        - ***AWS Application Load Balancer (ALB)***: The AWS Application Load Balancer (ALB) is a fully managed service designed to automatically distribute incoming application traffic across multiple targets, such as EC2 instances, containers, and IP addresses, within one or more Availability Zones. ALB operates at the application layer (Layer 7) of the OSI model, providing advanced routing features such as host-based and path-based routing, SSL termination, and WebSocket support.
        - ***RDS:*** Amazon Relational Database Service (AWS RDS) is a managed database service that simplifies the setup, operation, and scaling of relational databases in the cloud. Supporting multiple database engines, including MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server, AWS RDS automates routine tasks such as hardware provisioning, database setup, patching, and backups.

        - ***ECR:*** Amazon Elastic Container Registry (AWS ECR) is a fully managed container image registry that simplifies the process of storing, managing, and deploying Docker container images.

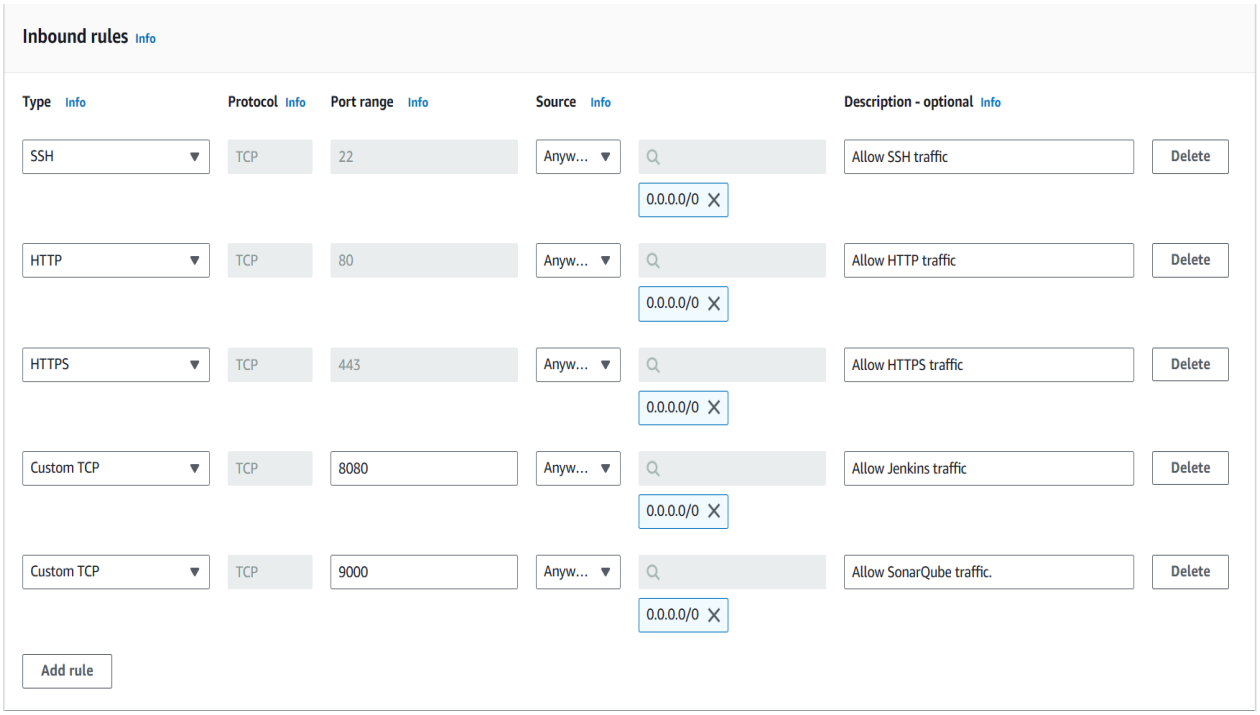
- **DevOps Tools:**
    - ***GitHub***: GitHub is indispensable for source code management, continuous integration, and collaboration, providing a centralized platform for version control, automated testing, and seamless deployment workflows.
    - ***Jenkins***: We employ Jenkins, an open-source automation server, for continuous integration and continuous delivery (CI/CD) pipelines.
    - ***Docker***: Our application is containerized using Docker, enabling efficient deployment and scalability across different environments.
    - ***Trivy***: We utilize Trivy, a vulnerability scanner for containers and other artifacts, to ensure the security of our application.
    - ***SonarQube***: We leverage SonarQube, an open-source platform for continuous inspection of code quality, to maintain code health and identify potential issues.
    - ***OWASP Dependency Check***: OWASP Dependency Check is crucial for automating the detection of vulnerable dependencies, ensuring the security and integrity of the software by identifying and mitigating risks early in the development cycle.
    - ***Slack***: Slack is essential for facilitating real-time communication, collaboration, and automated notifications, enabling teams to swiftly respond to build statuses, deployment updates, and incident alerts.
    - ***Terraform***: Terraform is crucial for infrastructure as code, enabling the automated provisioning, scaling, and management of cloud resources through declarative configuration files.

Basically, I will be creating two pipelines one for the ASP.NET app (backend) and the other is for the React app (frontend). Below we will be demonstrating the steps needed to create these pipelines.

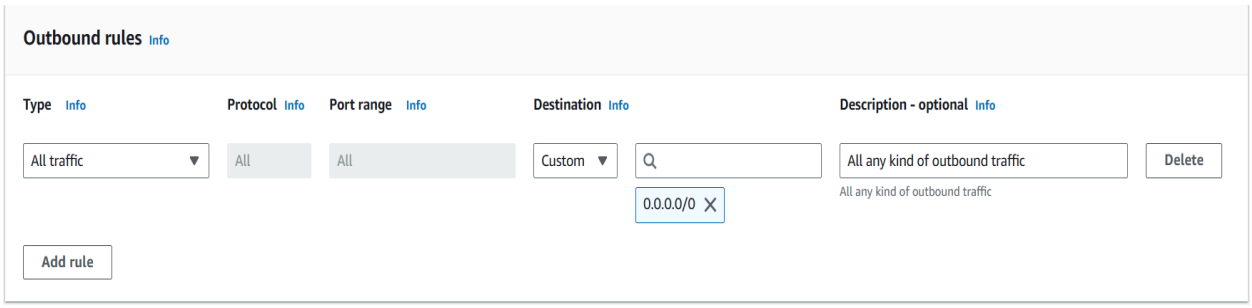
Initial Setup:

1. Create a security group for the EC2 instance which will be used for installing the pipeline required tools:

The security group will allow ***SSH*** traffic on port 22, ***HTTP*** traffic on port 80, ***HTTPS*** traffic on port 443, ***SonarQube*** traffic on port 9000 and ***Jenkins's*** traffic on port 8080 so the inbound traffic rules will be as following:



The outbound traffic will be as following:



2. Create an Ubuntu EC2 instance of type t2.large:

Using the AWS management console create an EC2 instance of type t2.large. The following images shows the details of the creation of that server:

Name and tags Info

Name

DEVOPS-FEE

Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents | Quick Start

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Li

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type — Free tier eligible

ami-0e001c9271cf7f3b9 (64-bit (x86)) / ami-058b428b3b45defec (64-bit (Arm))

Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2024-04-11

Architecture

64-bit (x86)

AMI ID

ami-0e001c9271cf7f3b9

Verified provider

Instance type Info | Get advice

Instance type

t2.large
Family: t2 2 vCPU 8 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.1208 USD per Hour
On-Demand RHEL base pricing: 0.1528 USD per Hour
On-Demand SUSE base pricing: 0.1928 USD per Hour
On-Demand Linux base pricing: 0.0928 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

DEVOPS-FEE-KEY-PAIR

Create new key pair

Network settings Info

Edit

Network Info
vpc-0c31cfe61d382b3a8

Subnet Info
No preference (Default subnet in any availability zone)

Auto-assign public IP Info
Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group | Select existing security group

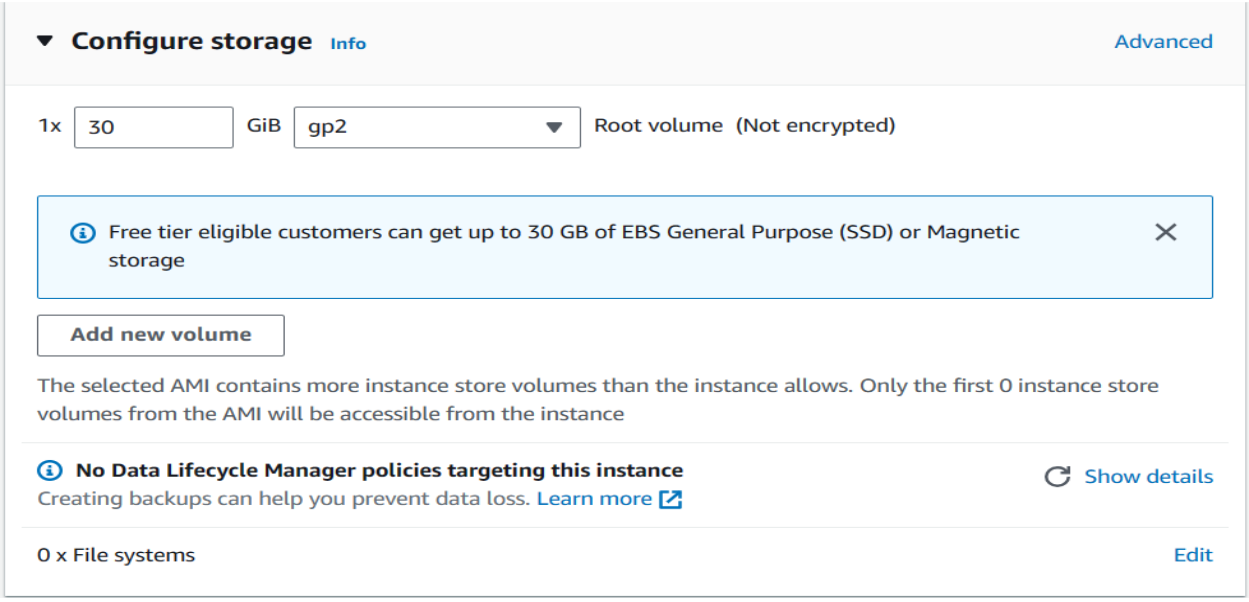
Common security groups Info

Select security groups

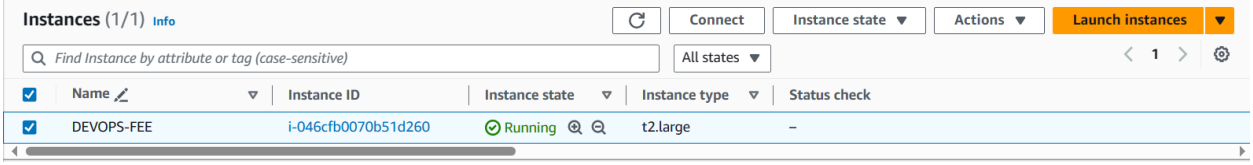
DEVOPS-FEE-SG sg-07a83cdfb1225b244
VPC: vpc-0c31cfe61d382b3a8

Compare security group rules

Security groups that you add or remove here will be added to or removed from all your network interfaces.



and now the EC2 instance is running and accessible from the internet.



3. Install Jenkins, Docker and Trivy. Create a Sonarqube Container using Docker.

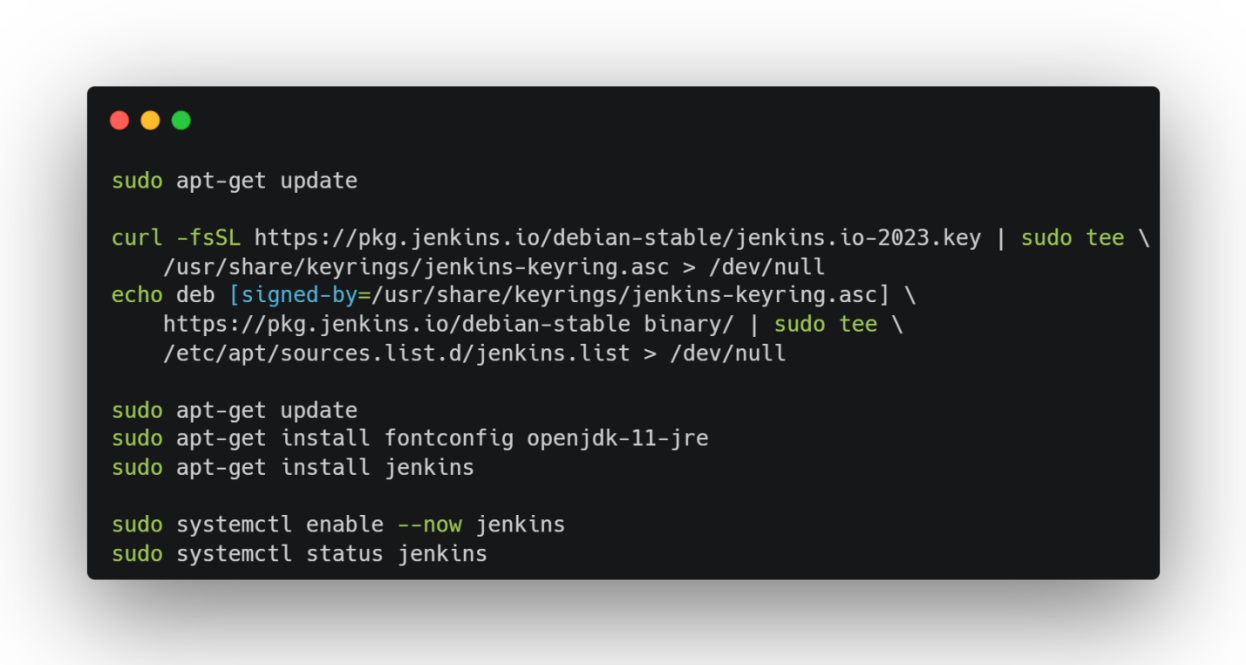
- *Get into the directory in which the Login key file is in and use the following command:*

```
ssh -i "DEVOPS-FEE-KEY-PAIR.pem" ubuntu@instance_IP_address
```

- *First, we will install Jenkins:*

Jenkins is an open-source automation server widely used for continuous integration and continuous delivery (CI/CD) pipelines. It acts as a hub for automating the building, testing, and deployment of software projects, allowing developers to focus more on coding and less on repetitive tasks. With its extensive plugin ecosystem, Jenkins offers flexibility to integrate with various tools and technologies, making it a favorite among development teams for its versatility and scalability. From small

startups to large enterprises, Jenkins serves as a reliable backbone for streamlining the software development lifecycle, fostering collaboration, and ensuring the rapid delivery of high-quality software products.

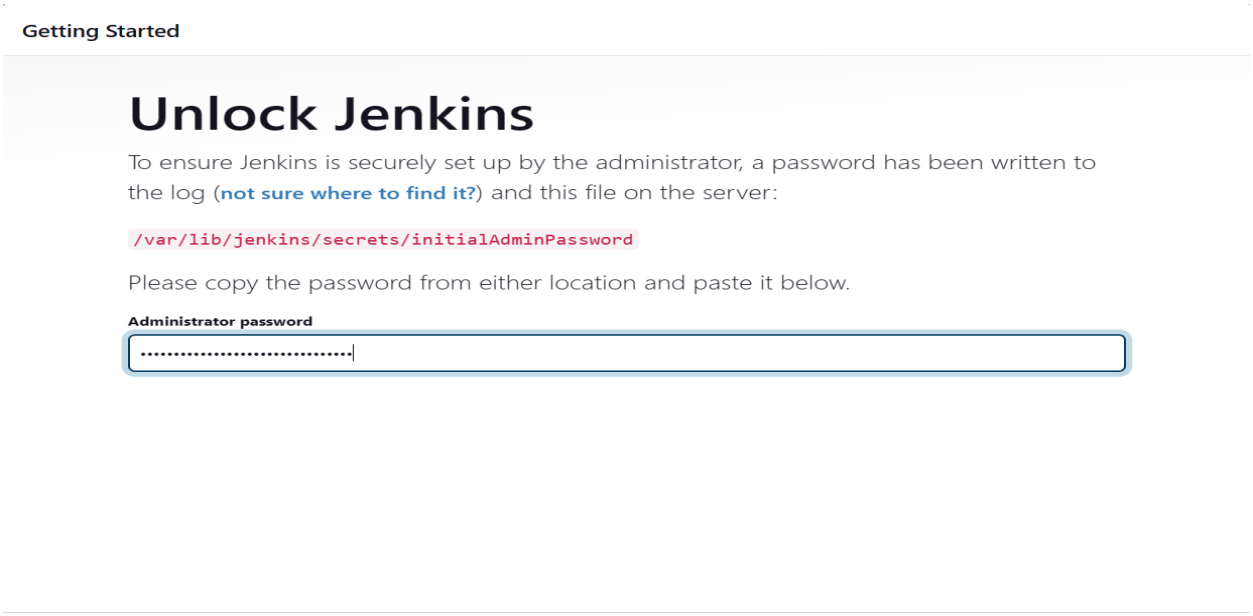


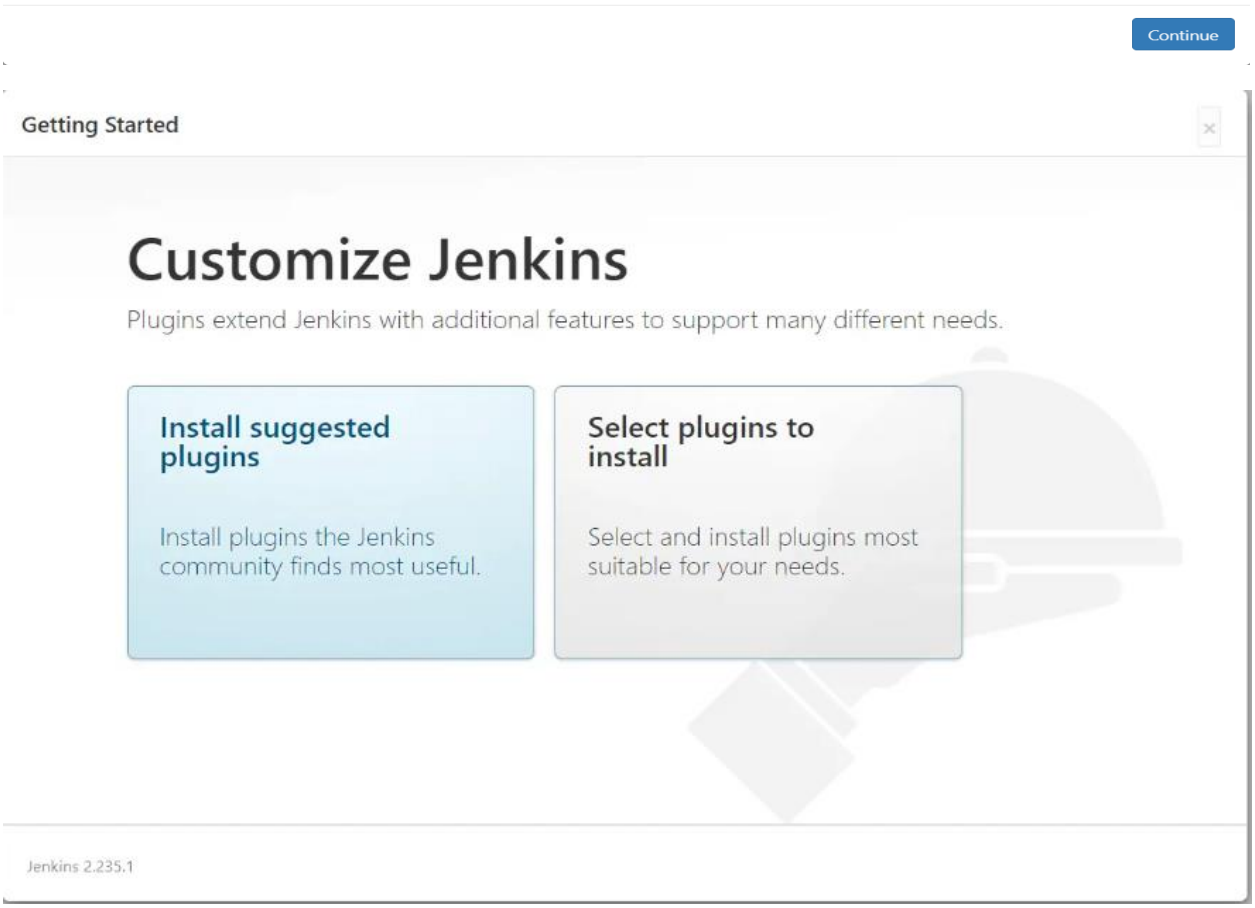
```
sudo apt-get update

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
    /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
    https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
    /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update
sudo apt-get install fontconfig openjdk-11-jre
sudo apt-get install jenkins

sudo systemctl enable --now jenkins
sudo systemctl status jenkins
```

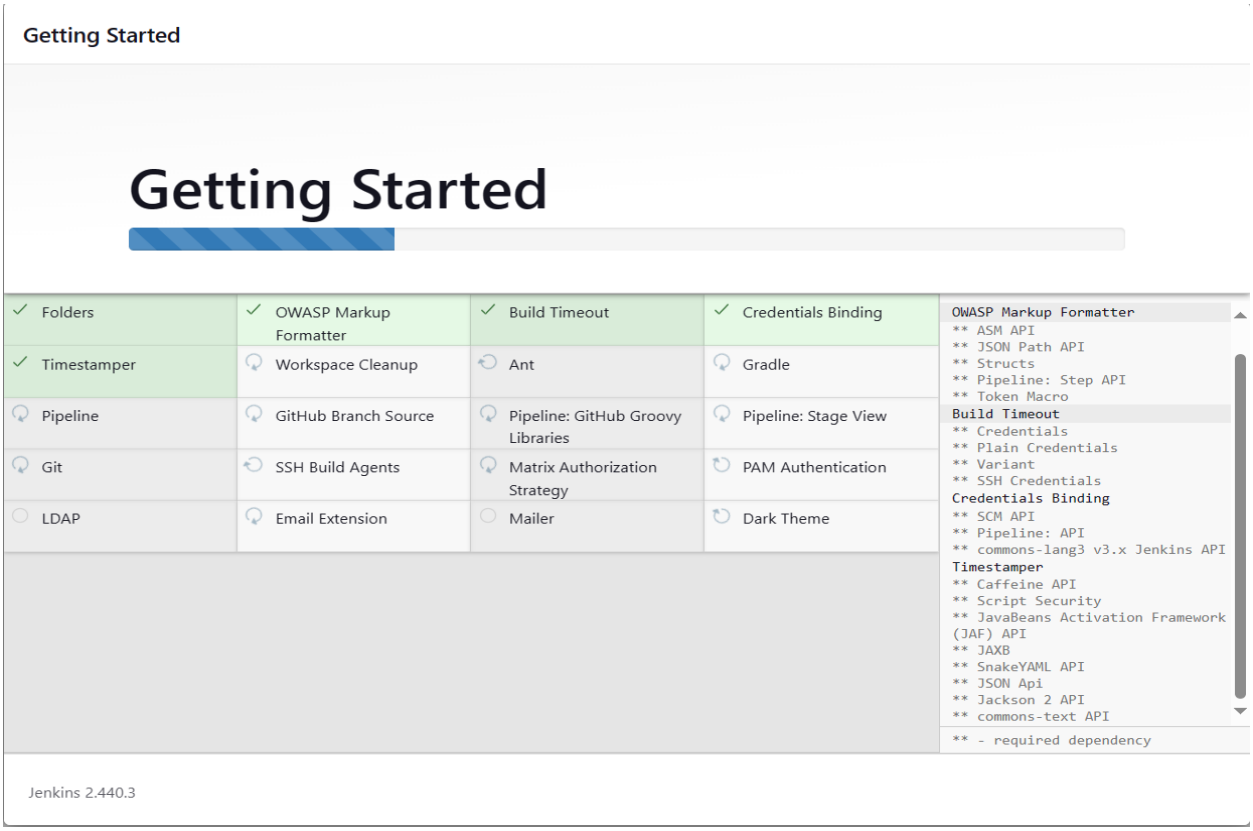
After the installation is successful, we can access the Jenkins UI by using the following URL:

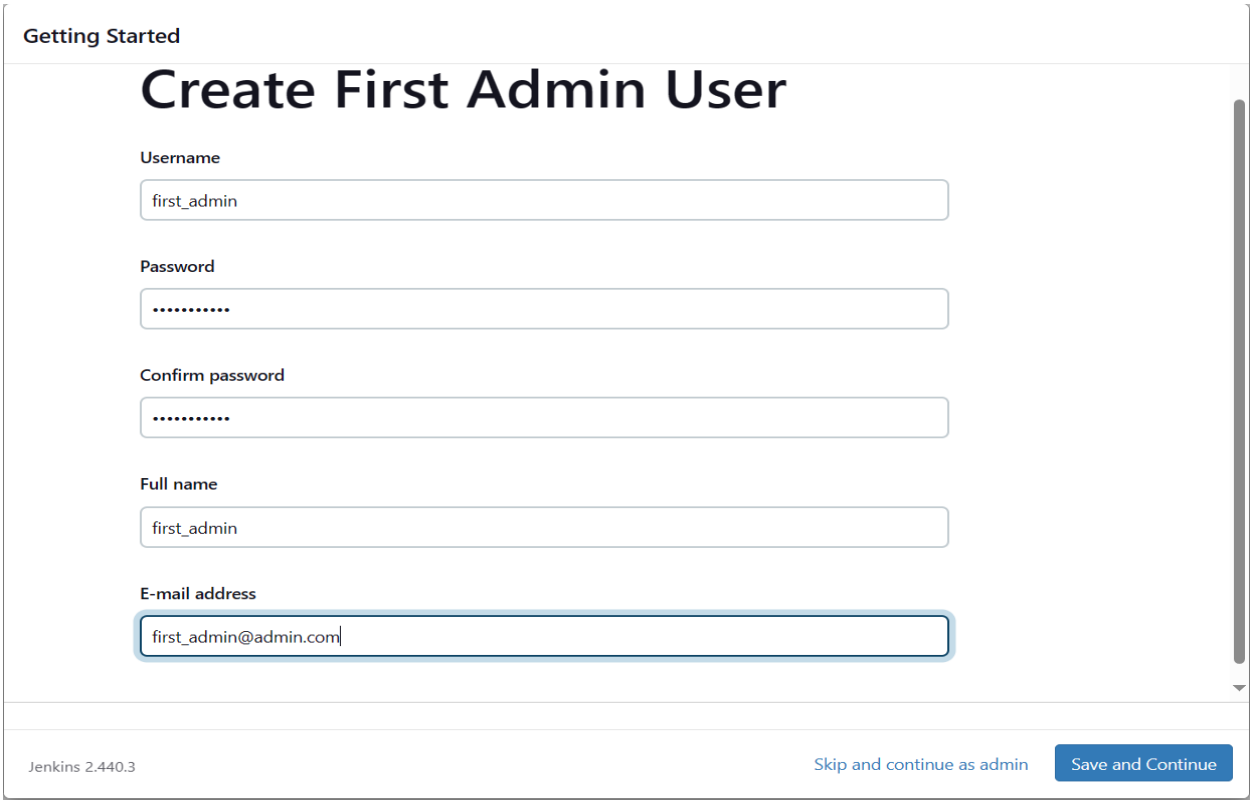
http://<public-ip-address>:8080

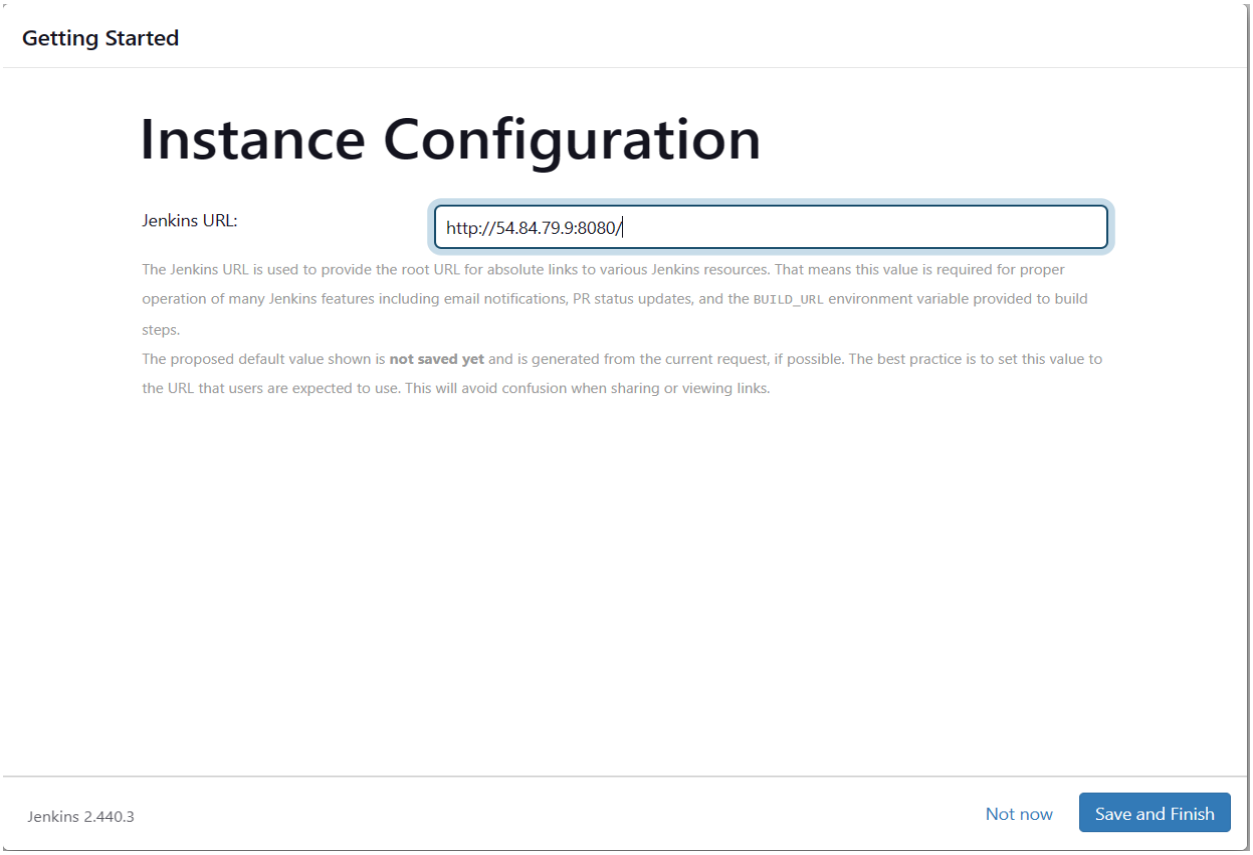
Now you will need to go through the following steps which will be shown using the images below to complete the installation:

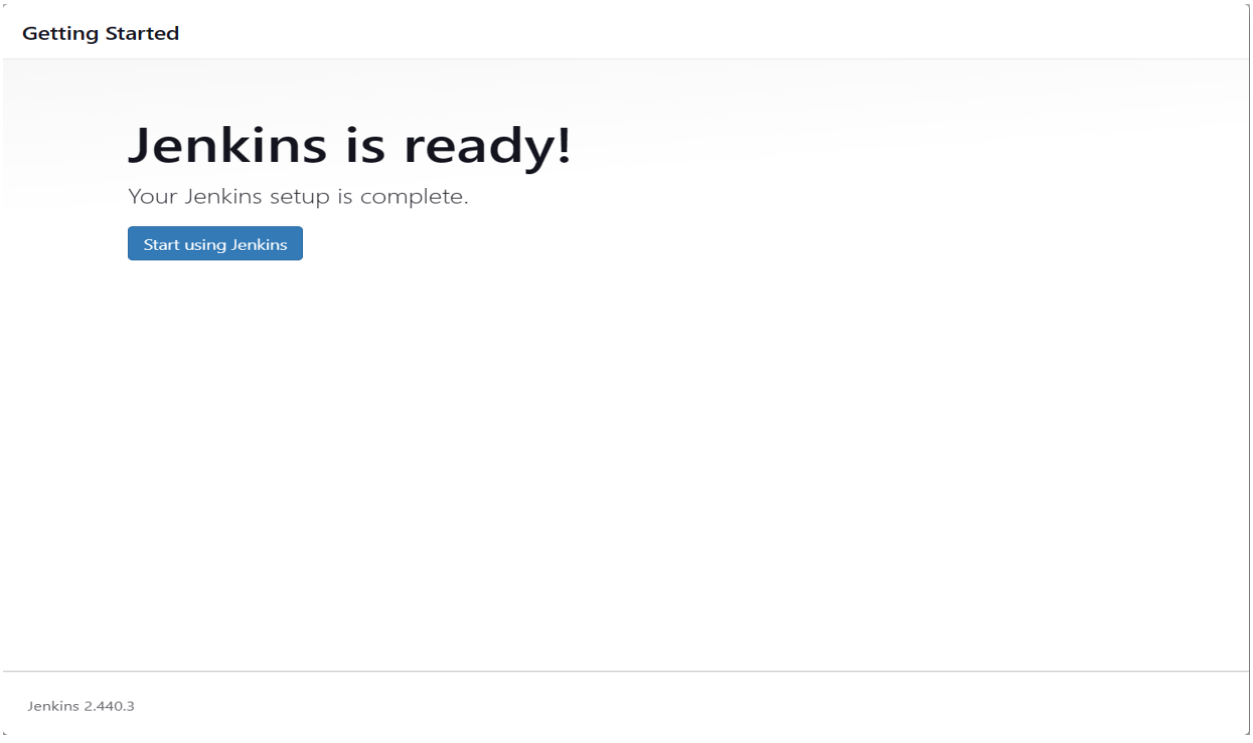

Getting Started
Unlock Jenkins
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:
/var/lib/jenkins/secrets/initialAdminPassword
Please copy the password from either location and paste it below.
Administrator password


Continue
Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.235.1

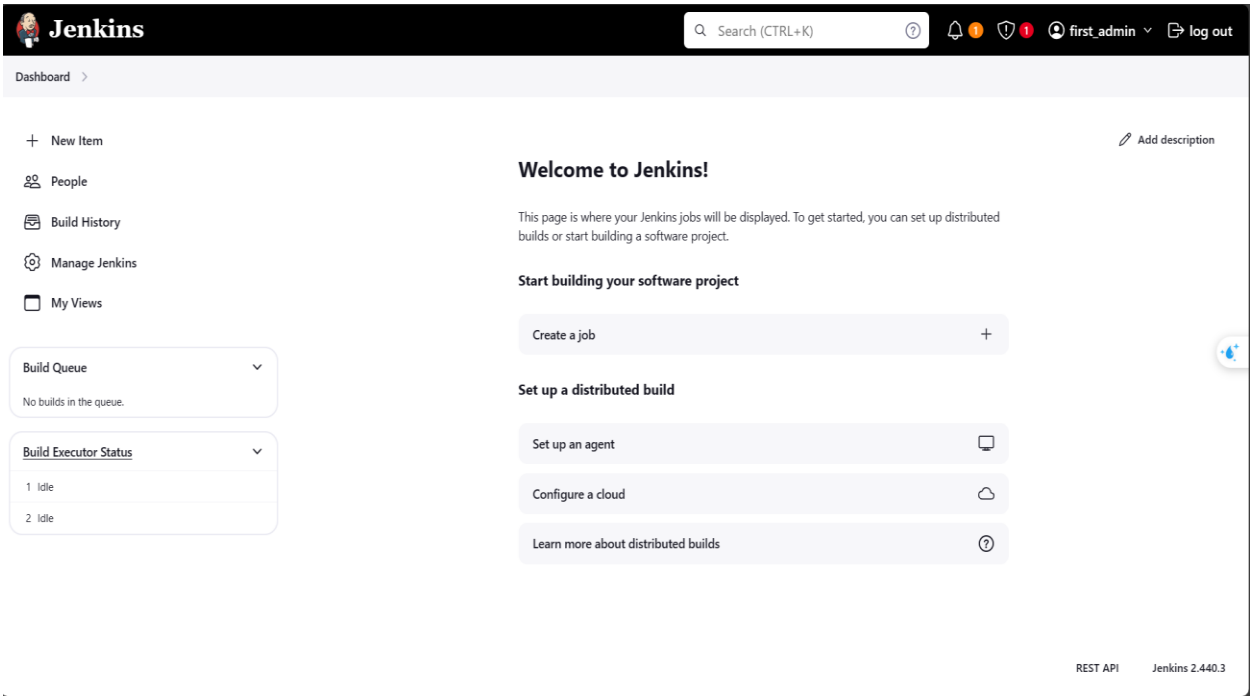

Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline: Stage View
Git
SSH Build Agents
Matrix Authorization Strategy
PAM Authentication
LDAP
Email Extension
Mailer
Dark Theme
OWASP Markup Formatter
** ASM API
** JSON Path API
** Structs
** Pipeline: Step API
** Token Macro
Build Timeout
** Credentials
** Plain Credentials
** Variant
** SSH Credentials
Credentials Binding
** SCM API
** Pipeline: API
** commons-lang3 v3.x Jenkins API
Timestamper
** Caffeine API
** Script Security
** JavaBeans Activation Framework (JAF) API
** JAXB
** SnakeYAML API
** JSON Api
** Jackson 2 API
** commons-text API
** - required dependency
Jenkins 2.440.3


Getting Started
Create First Admin User
Username
first_admin
Password
Confirm password
Full name
first_admin
E-mail address
first_admin@admin.com
Jenkins 2.440.3
Skip and continue as admin
Save and Continue





After following these steps, you will be able to access the UI:

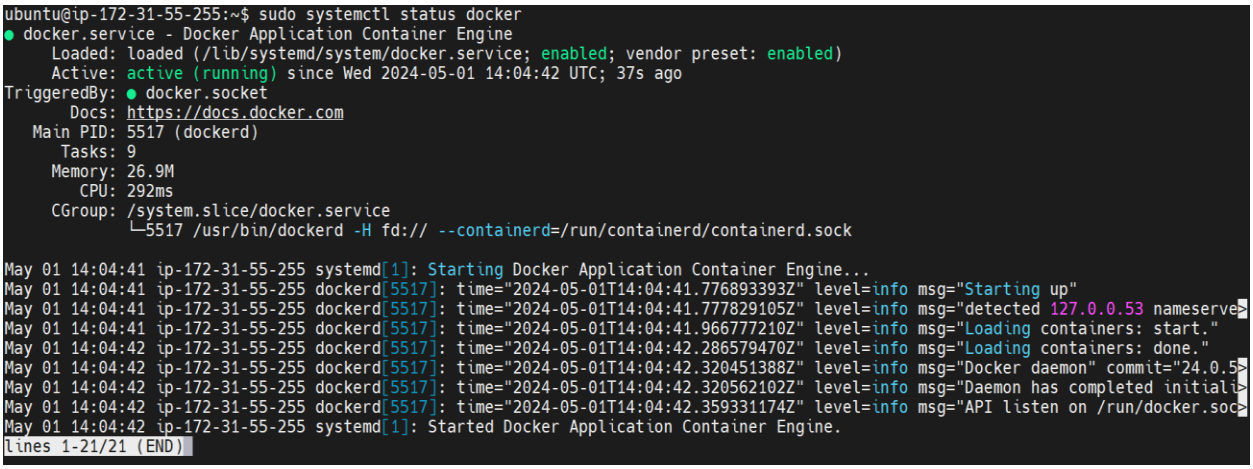


- *Second, we will install docker:*

Docker revolutionizes software development and deployment by providing a platform for containerization. With Docker, developers can package their applications and all their dependencies into a standardized container, ensuring consistency across different environments, from development to production. These containers are lightweight, portable, and isolated, enabling easy deployment on any infrastructure, whether it's on-premises, in the cloud, or hybrid environments. Docker's simplicity and efficiency have made it immensely popular among developers, DevOps teams, and enterprises, streamlining the development process, enhancing scalability, and facilitating the adoption of microservices architectures. Its ecosystem of tools and services, combined with robust community support, cements Docker's position as a cornerstone technology in modern software development and deployment workflows.

```
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER
sudo chmod 777 /var/run/docker.sock
sudo docker ps #Check if it's running as intended
```

and now if we checked the status of docker we will see that it's running:



- *Third, we will install Trivy:*

Trivy is a popular open-source vulnerability scanner designed specifically for containerized environments. It offers developers and DevOps teams a comprehensive solution for identifying security vulnerabilities within container images and their dependencies. Trivy employs an extensive vulnerability database and a fast-scanning engine to detect potential security threats, including outdated packages, misconfigurations, and known vulnerabilities. Its seamless integration with CI/CD pipelines and container orchestration platforms makes it an invaluable tool for ensuring the security posture of containerized applications throughout the software development lifecycle. With Trivy, organizations can proactively address security risks, strengthen their defenses against potential attacks, and maintain the integrity of their containerized environments with ease and efficiency.

```
sudo apt-get install wget apt-transport-https gnupg lsb-release

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee
/usr/share/keyrings/trivy.gpg > /dev/null

echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb
$(lsb_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list

sudo apt-get update

sudo apt-get install trivy -y
```

```
ubuntu@ip-172-31-55-255:~$ trivy -v
Version: 0.50.2
```

- *Fourth, we will create a container for SonarQube:*

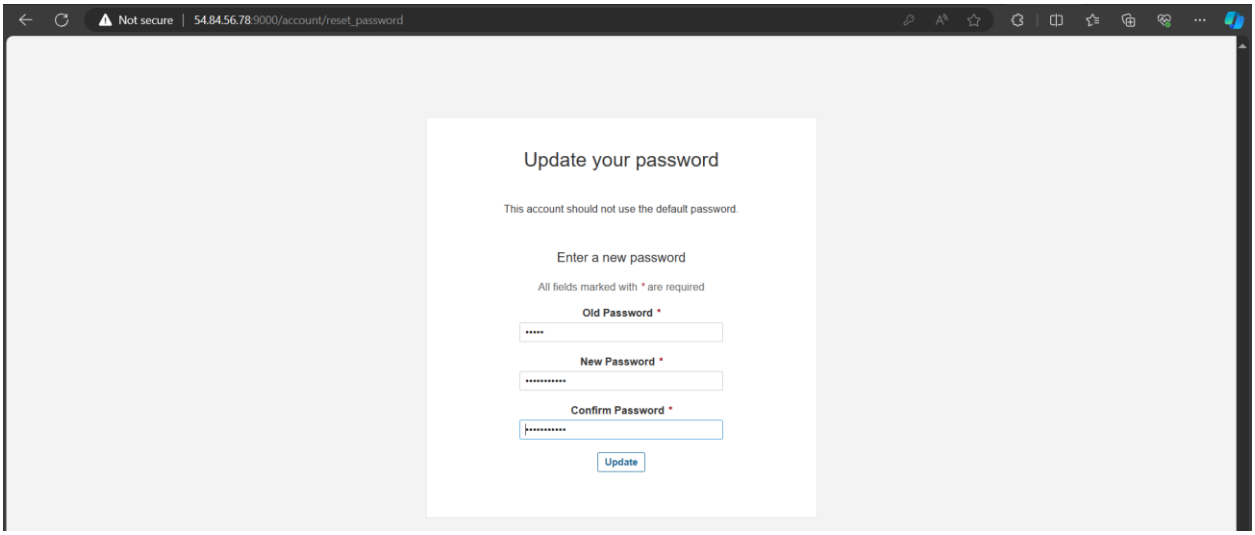
SonarQube is a leading open-source static code analysis tool that helps developers and organizations enhance the quality and maintainability of their codebase. It offers a comprehensive suite of features for identifying code smells, bugs, vulnerabilities, and security vulnerabilities across multiple programming languages. SonarQube provides detailed reports and actionable insights, empowering development teams to prioritize and address issues early in the development process. With its integration capabilities with popular CI/CD pipelines and IDEs, SonarQube seamlessly fits into existing workflows, enabling continuous code quality improvement. Its user-friendly interface and customizable rule sets make it a versatile tool for teams of all sizes, from small startups to large enterprises, striving to deliver high-quality, secure, and maintainable software products.

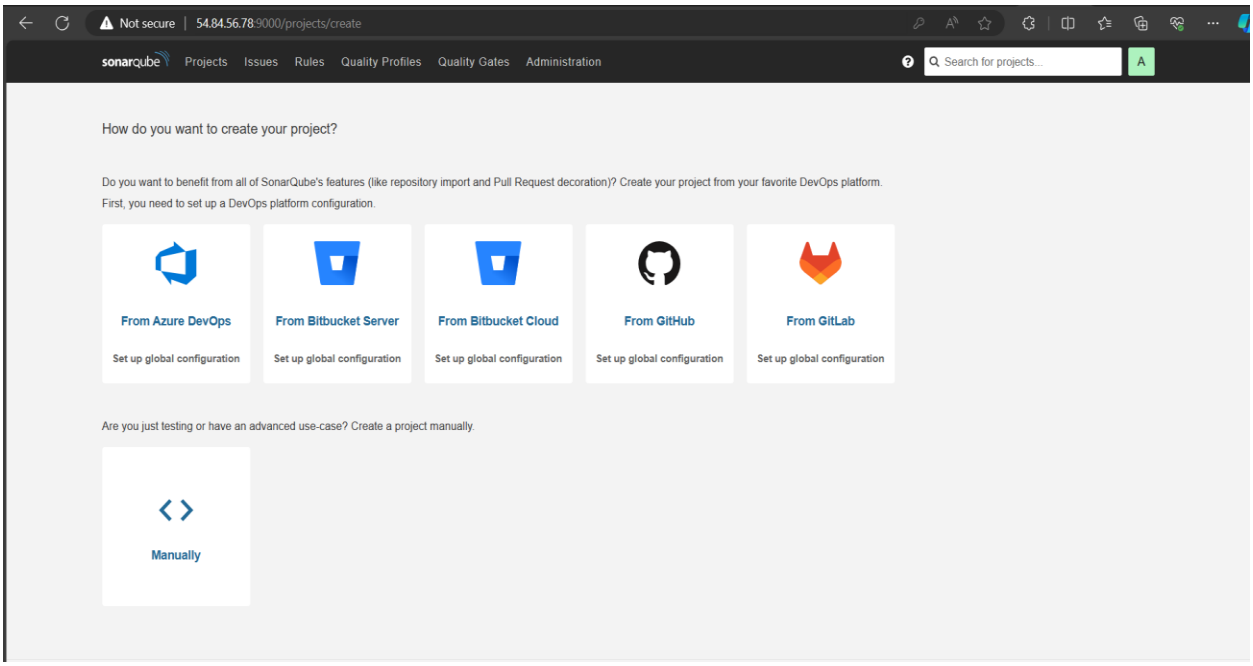
```
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

After the installation is successful, we can access the SonarQube UI by using the following URL:

http://<public-ip-address>:9000

You will go through the following steps to finish the installation of SonarQube:

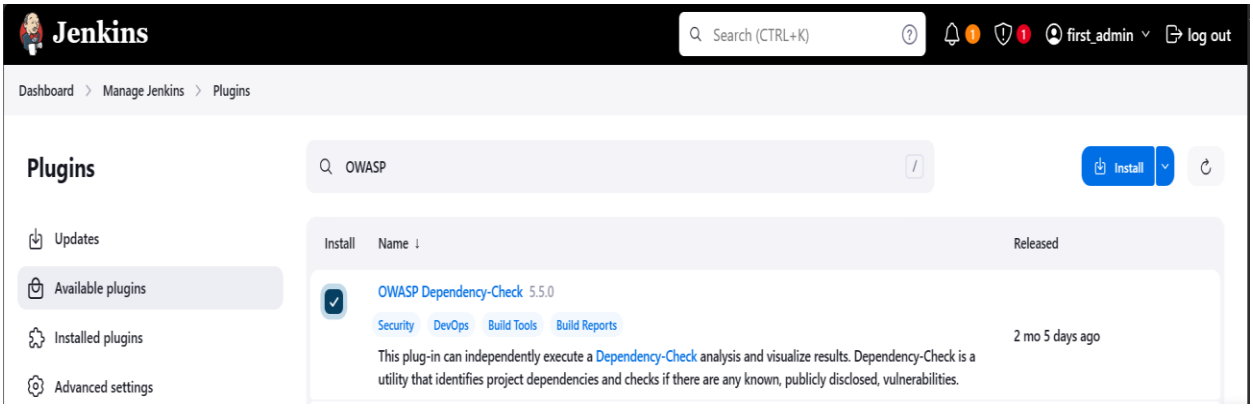




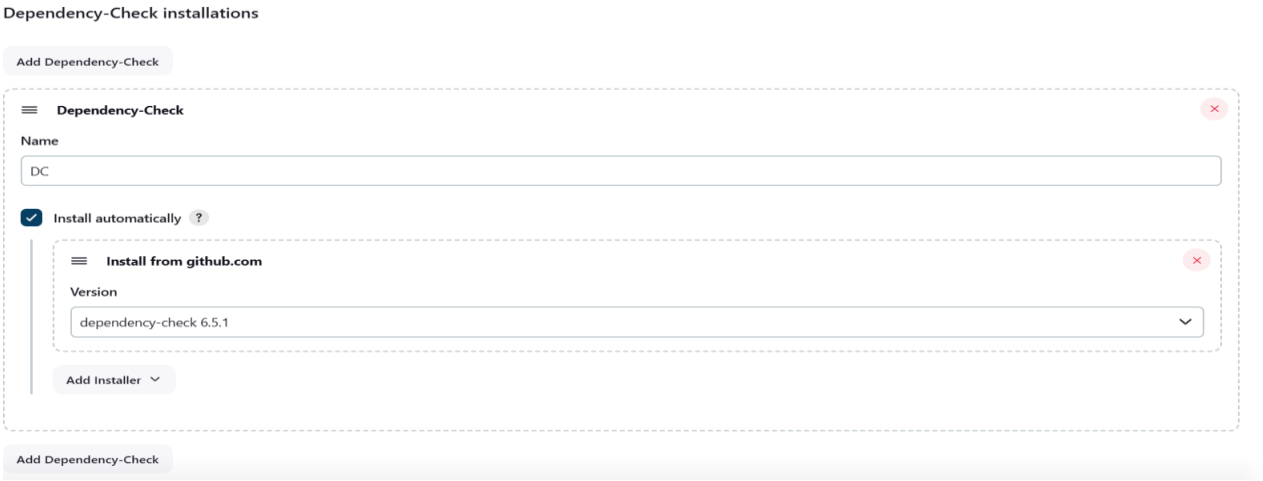
4. Install Plugins like JDK, Sonarqube Scanner, OWASP Dependency Check:

- *Go to Manage Jenkins → Plugins → Available Plugins:*

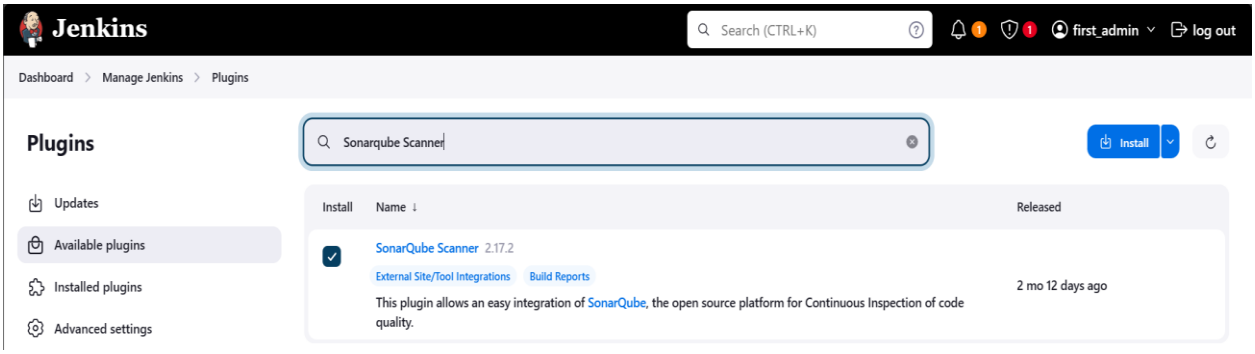
- *First Let's install and configure the OWASP Dependency Check plugin:*



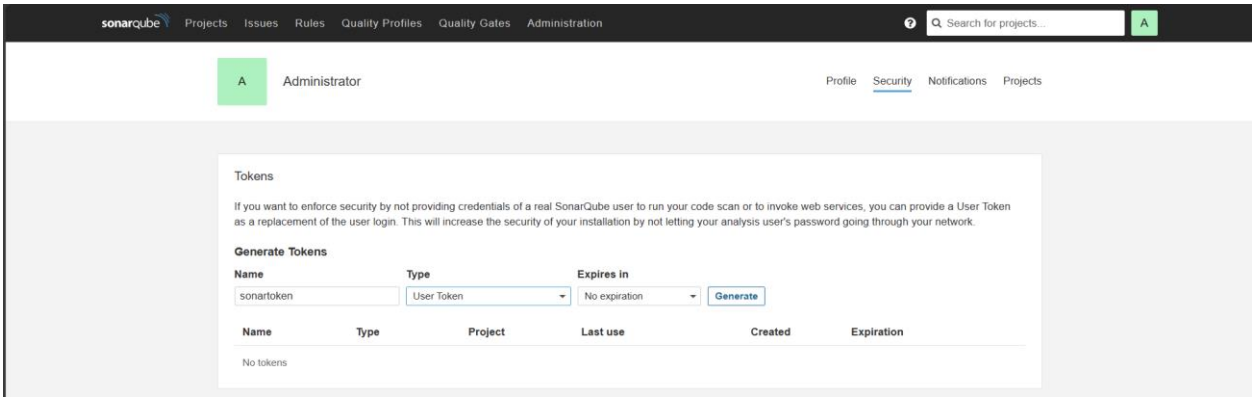
and from Manage Jenkins → Tools we will configure the plugin:

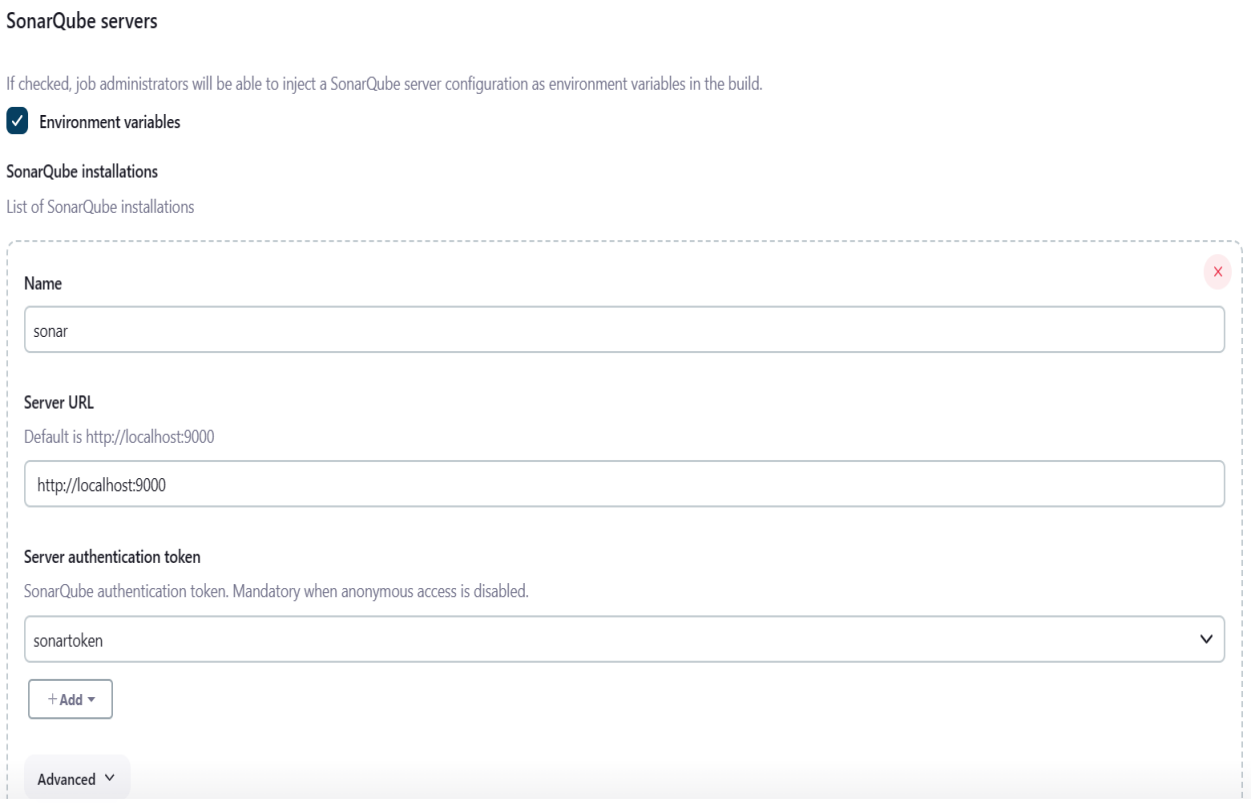


- *Second, Let's install and configure the SonarQube Scanner plugin*



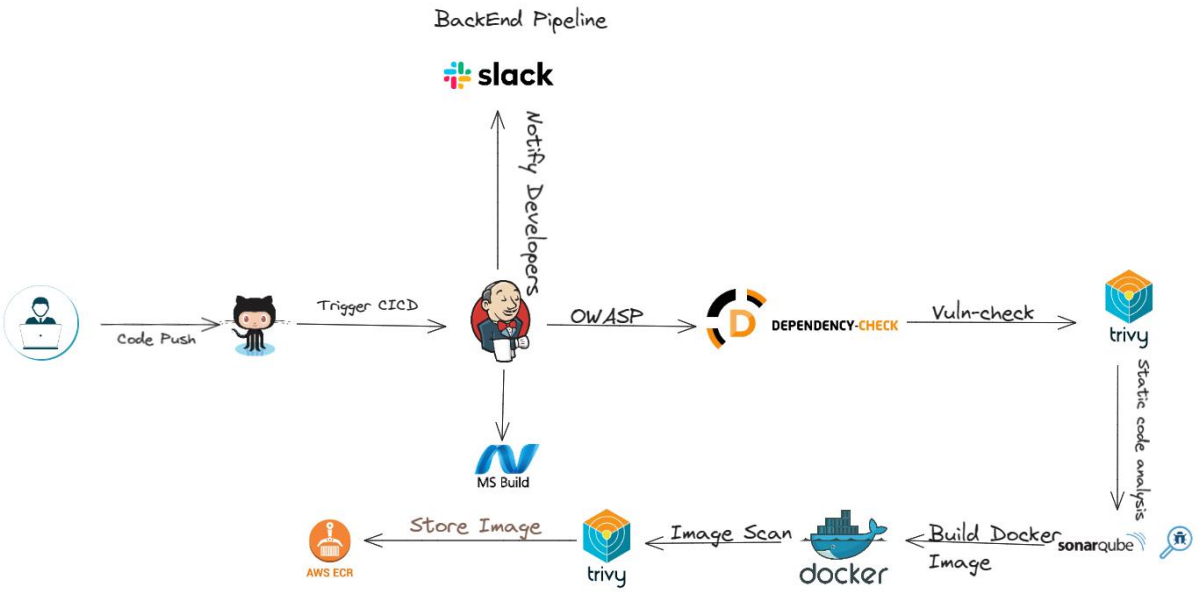
and now we will need to do some steps to integrate SonarQube with Jenkins and we will start by generating a token through the SonarQube UI and use it to configure the SonarQube Server configuration at Jenkins: From the SonarQube UI go to Administration → Security and generate the token then go to Jenkins and configure the SonarQube Server by going to Manage Jenkins → System and go to the SonarQube server section:





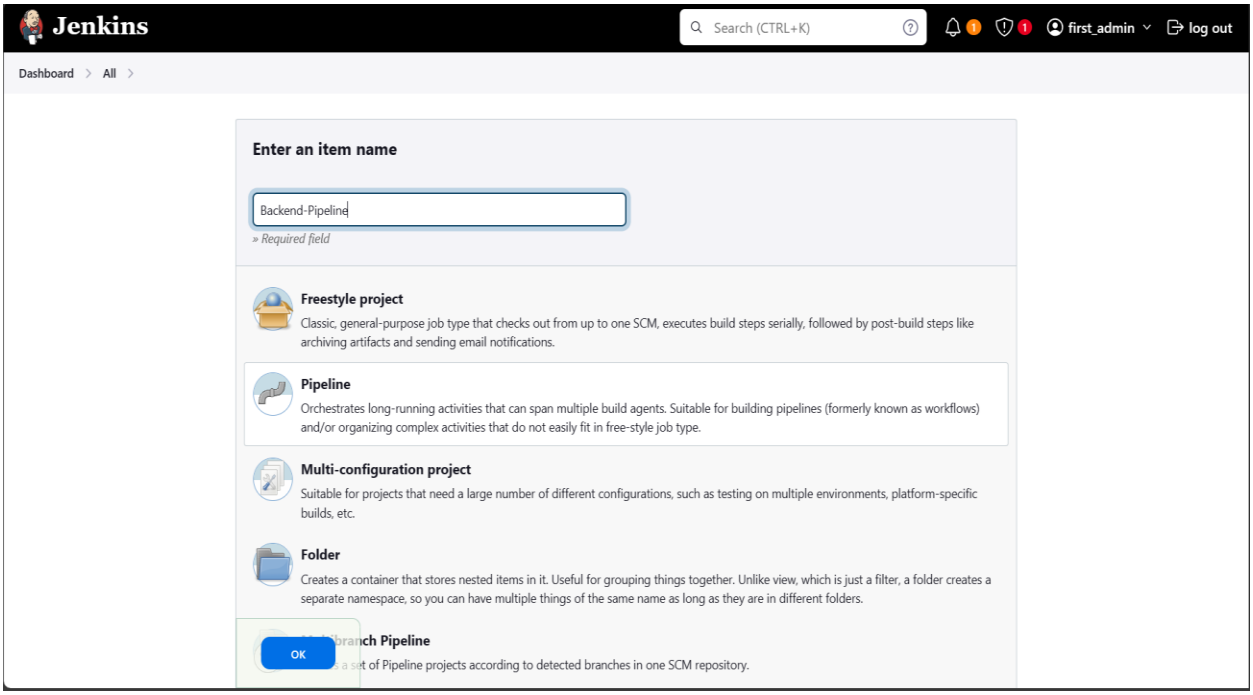
and from Manage Jenkins → Tools:

The Backend Pipeline:



Let's walk through the steps of the pipeline step by step and implement each step:

- *From the Jenkins UI we will create a job of type Pipeline:*



- *As shown in the image above representing the pipeline, it will start by fetching the code from the repository, this can be done using the syntax below (In the Jenkinsfile):*

```
pipeline {
    agent any

    stages {
        stage('Fetching Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
    }
}
```

- *The next step in the pipeline is to perform the OWASP dependency check of the fetched code(8.0.2):*

*The **AssemblyAnalyzer** requires the installation of the .net 6 runtime to work.*

```
pipeline {
    agent any

    stages {
        stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                //For HTML reports
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
    }
}
```

- *The next step in the pipeline is to use trivy to scan the file system of the directory structure of the project:*

```
pipeline {
    agent any

    stages {
        stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
        stage('Trivy FS vulnerability scanning') {
            steps {
                //sh "trivy fs -f json -o result.json ."  //Output as json
                sh "trivy fs ."
            }
        }
    }
}
```

- *The next step in the pipeline is to use SonarQube for the static code analysis task, which include tasks like scanning the app code for potential bugs, code coverage (checking what percentage of the code is covered by the written code for testing), and also scanning for potential security vulnerabilities.*

```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
        stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
        stage('Trivy FS vulnerability scanning') {
            steps {
                //sh "trivy fs -f json -o result.json ."  //Output as json
                sh "trivy fs ."
            }
        }
        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Backend \
                    -Dsonar.projectKey=Backend '''
                }
            }
        }
    }
}
```

- *In this step we will be building the docker image, we want to run our app in an isolated environment that packages the app and all of its dependencies in a single package which is called an image in the world of docker:*

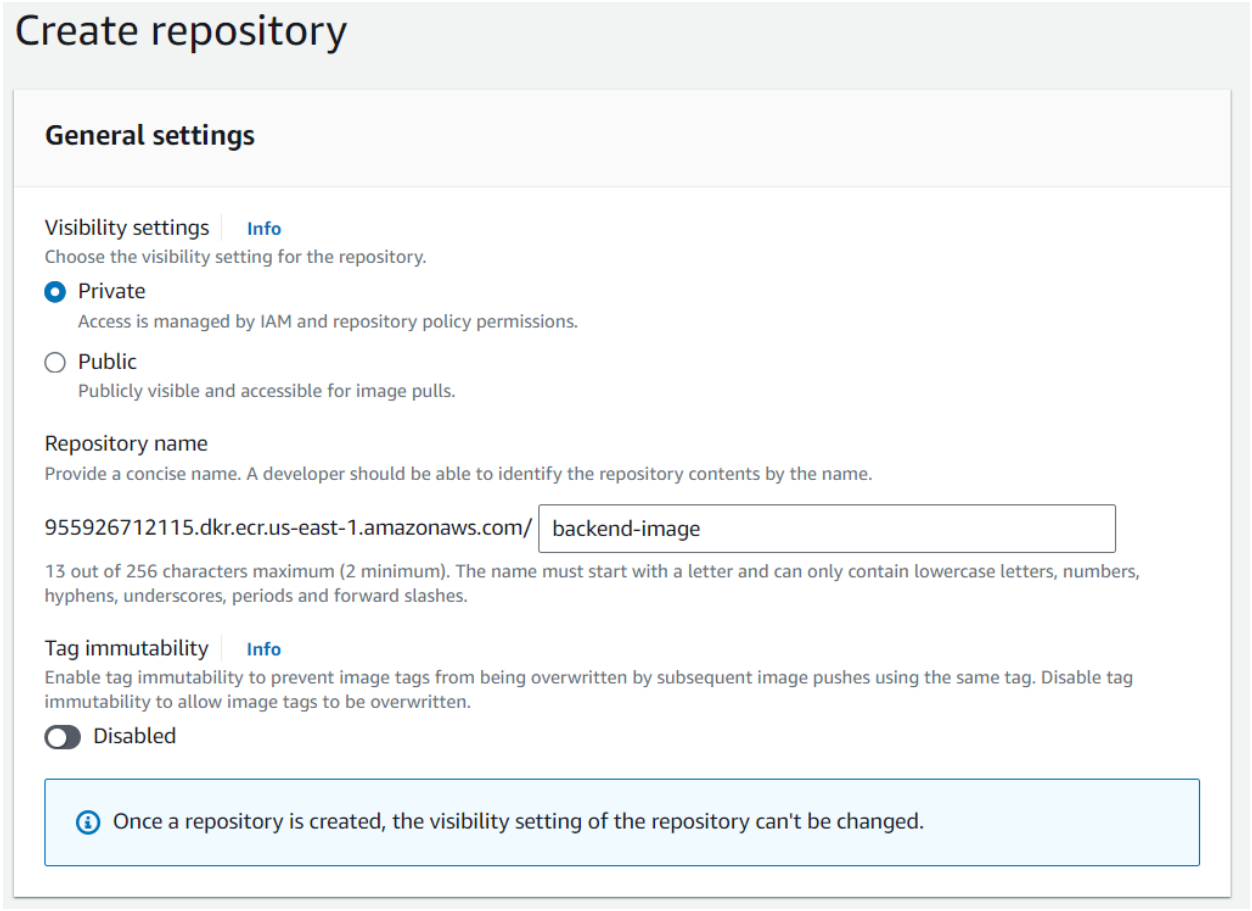
```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
        stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
        stage('Trivy FS vulnerability scanning') {
            steps {
                //sh "trivy fs -f json -o result.json ."  //Output as json
                sh "trivy fs ."
            }
        }
        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Backend \
                    -Dsonar.projectKey=Backend '''
                }
            }
        }
        stage ("Building Docker Image"){
            steps {
                sh "echo ********************Building the backend docker
image******************************"
                sh "docker build -t backend-image -f ./APIs/APIs/Dockerfile ./APIs/APIs"
                sh "echo ********************Done Building the
image*************************************"
            }
        }
    }
}
```

- *Now we need to scan the docker image that we created in the previous step before we can push it to the AWS ECR registry, and this is a very important step for ensuring that the image is error and vulnerability free:*

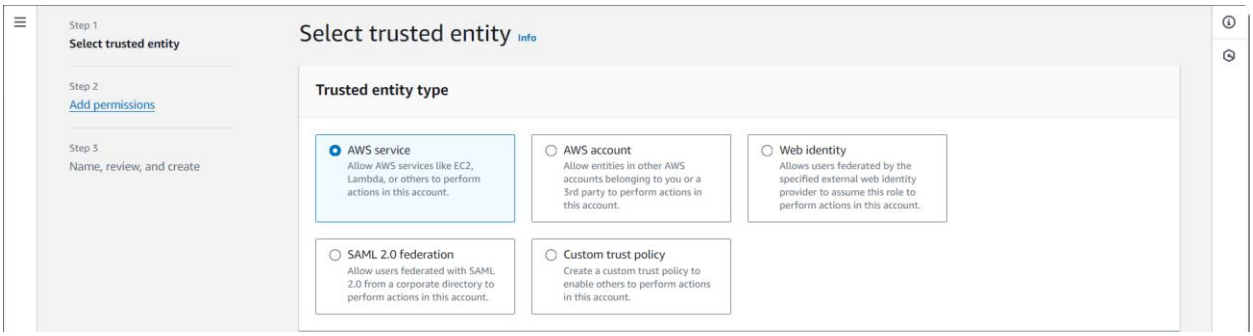
```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
        stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
        stage('Trivy FS vulnerability scanning') {
            steps {
                //sh "trivy fs -f json -o result.json ."  //Output as json
                sh "trivy fs ."
            }
        }
        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Backend \
                    -Dsonar.projectKey=Backend '''
                }
            }
        }
        stage ("Building Docker Image"){
            steps {
                sh "echo ********************Building the backend docker
image*******************************"
                sh "docker build -t backend-image -f ./APIs/APIs/Dockerfile ./APIs/APIs"
                sh "echo ********************Done Building the
image***************************************"
            }
        }
        stage ("Scanning the docker image") {
            steps {
                sh "echo ************************* Scanning the docker image*****************"
                sh "trivy image backend-image"
                sh "************************* Done Scanning the image****************
            }
        }
    }
}
```

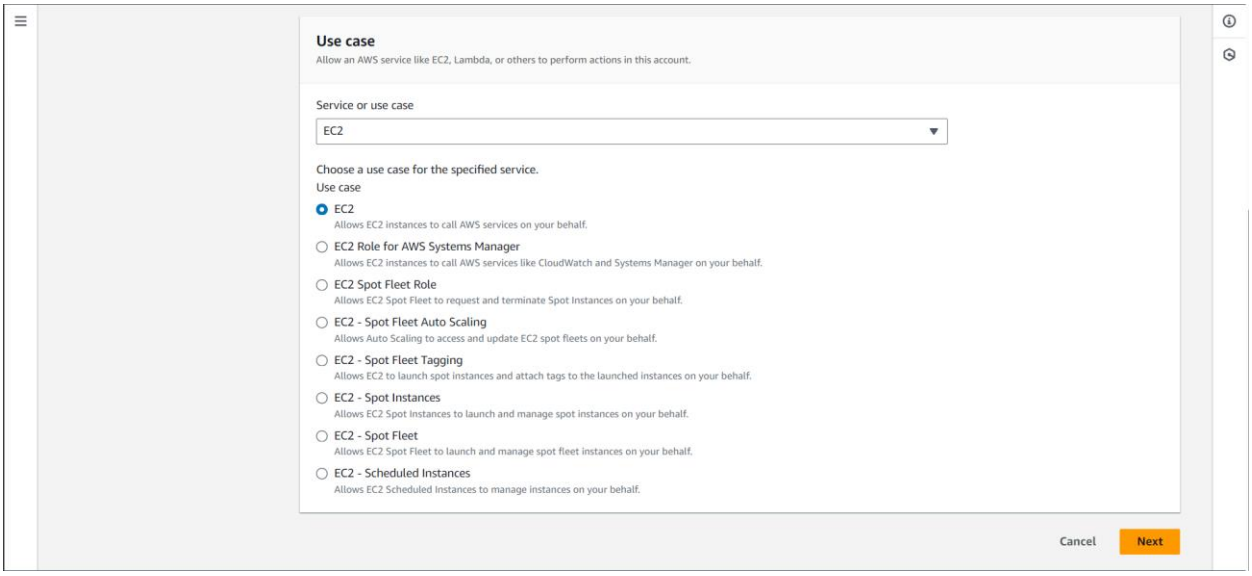
- *The final step of this CI pipeline pushes the docker image to AWS ECR registry:*

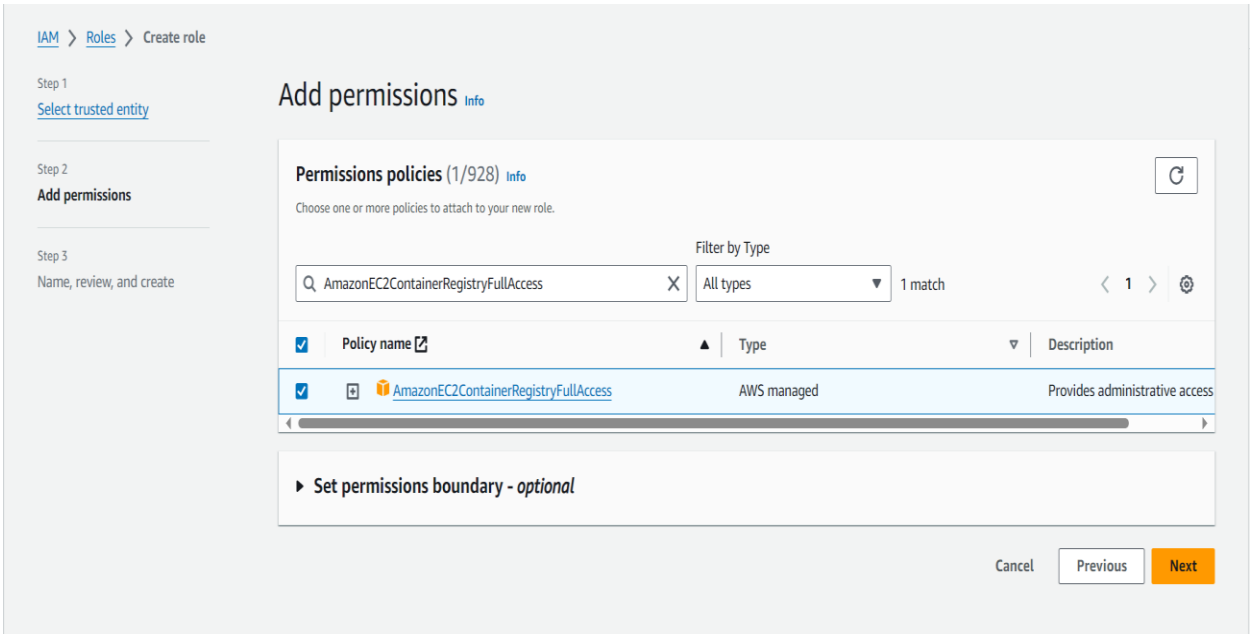
- First go the AWS management console and create a new repository in ECR.

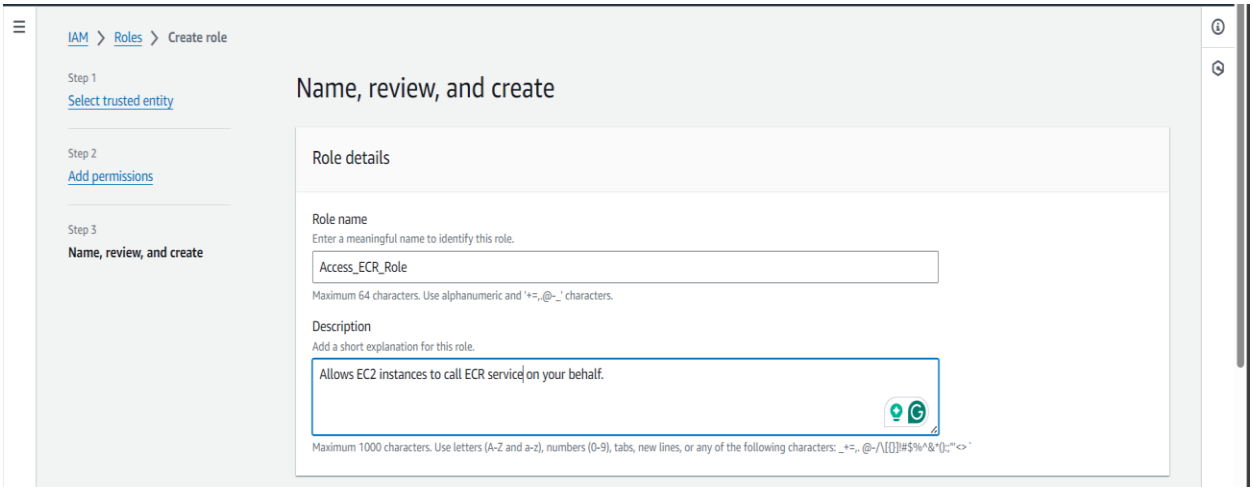


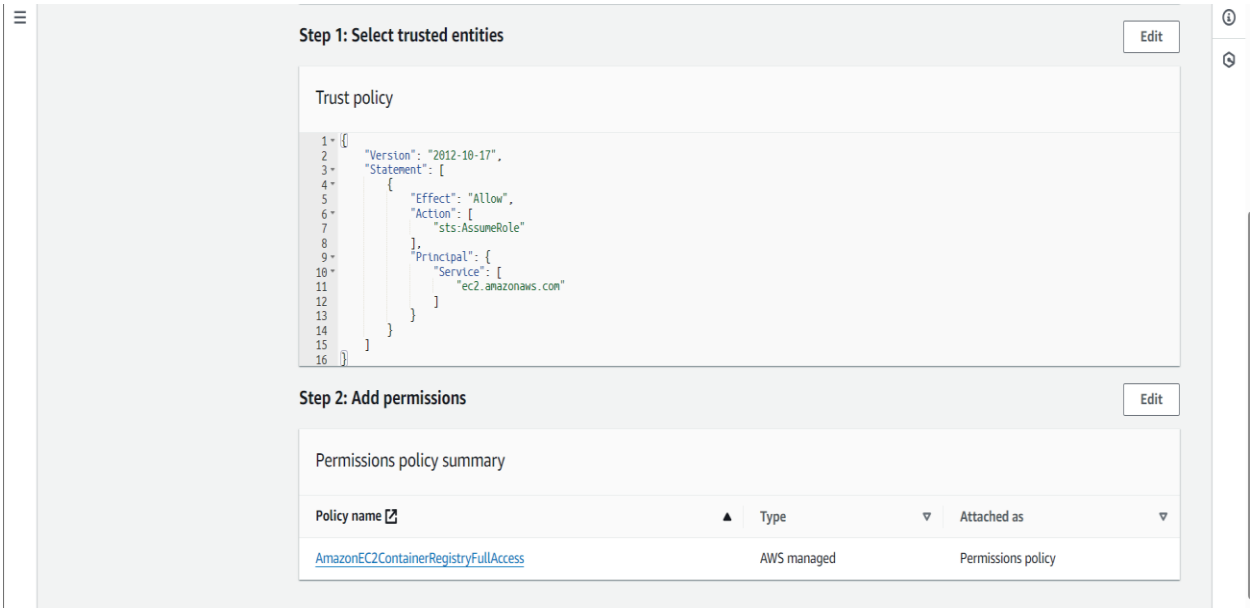
- Then create a new IAM role with the permissions ***AmazonEC2ContainerRegistryFullAccess***:



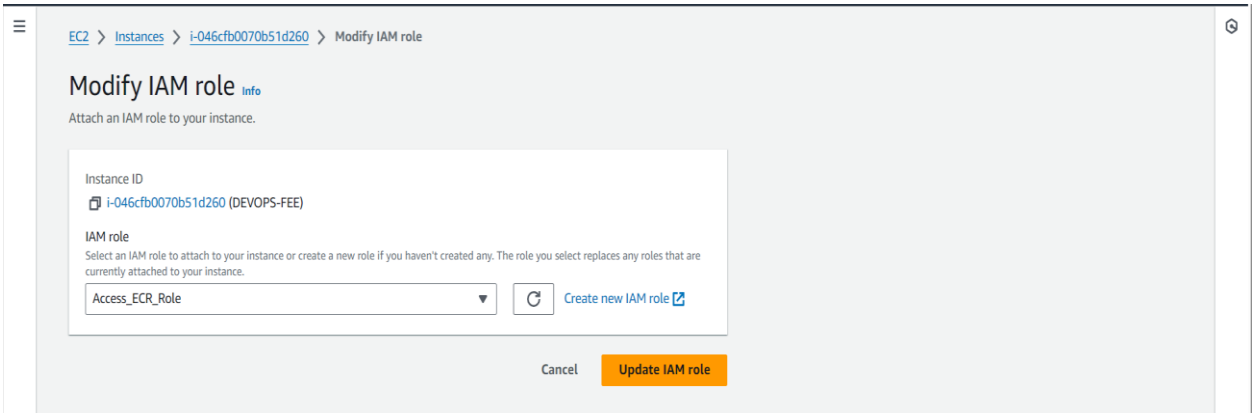

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.
Service or use case
EC2
Choose a use case for the specified service.
Use case
EC2
Allows EC2 instances to call AWS services on your behalf.
EC2 Role for AWS Systems Manager
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
EC2 Spot Fleet Role
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
EC2 - Spot Fleet Auto Scaling
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
EC2 - Spot Fleet Tagging
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
EC2 - Spot Instances
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.
EC2 - Spot Fleet
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
EC2 - Scheduled Instances
Allows EC2 Scheduled Instances to manage instances on your behalf.
Cancel
Next


IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Add permissions Info
Permissions policies (1/928) Info
Choose one or more policies to attach to your new role.
Filter by Type
AmazonEC2ContainerRegistryFullAccess
All types
1 match
Policy name
Type
Description
AmazonEC2ContainerRegistryFullAccess
AWS managed
Provides administrative access
Set permissions boundary - optional
Cancel
Previous
Next


IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
Name, review, and create
Role details
Role name
Enter a meaningful name to identify this role.
Access_ECR_Role
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.
Description
Add a short explanation for this role.
Allows EC2 instances to call ECR service on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^&*();:'"<> `



and now we need to attach this Role to the EC2 instance that is running Jenkins:



Now, after implementing all the steps of this CI pipeline we now have a docker image of the .NET app stored in an ECR registry which will be used to create a container from and deploy the app.

The following shows the final content of the Jenkinsfile after implementing all the steps of the pipeline:

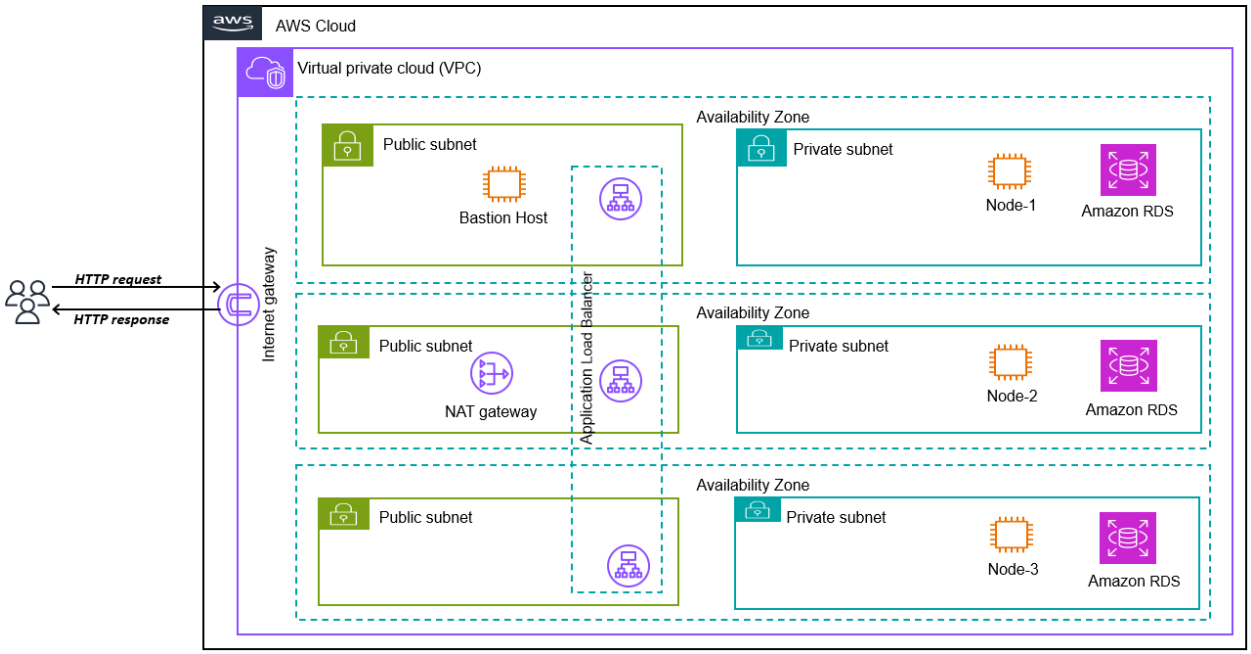
```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
        stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
        stage('Trivy FS vulnerability scanning') {
            steps {
                //sh "trivy fs -f json -o result.json ."  //Output as json
                sh "trivy fs ."
            }
        }
        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Backend \
                    -Dsonar.projectKey=Backend '''
                }
            }
        }
        stage ("Building Docker Image"){
            steps {
                sh "echo ******************Building the backend docker
image*******************************"
                sh "docker build -t backend-image -f ./APIs/APIs/Dockerfile ./APIs/APIs"
                sh "echo ********************Done Building the
image*************************************"
            }
        }
        stage ("Scanning the docker image") {
            steps {
                sh "echo ************************ Scanning the docker image****************"
                sh "trivy image backend-image"
                sh "************************ Done Scanning the image***************
            }
        }
        stage ("Pushing the docker image to ECR repo") {
            steps {
                sh "echo ************************Pushing the docker image****************"
                sh "aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 955926712115.dkr.ecr.us-east-1.amazonaws.com"
                sh "docker tag backend-image:latest 955926712115.dkr.ecr.us-east-1.amazonaws.com/backend-
image:latest"
                sh "docker push 955926712115.dkr.ecr.us-east-1.amazonaws.com/backend-image:latest"
                sh "echo ************************The image has been pushed successfully****************
            }
        }
    }
}
```

The Frontend Pipeline:

The frontend pipeline will have the same steps as the backend pipeline only the build tools will change so the final content of the ***Jenkinsfile*** of the frontend pipeline will be as follows:

```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
       stage('Fetch Code') {
            steps {
                git branch: 'main', credentialsId: 'Graduation_Project_Access_Token', url:
'https://github.com/Kareem-Ataam/Graduation-Project'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: '--scan ./Client ', odcInstallation: 'DC'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                // dependencyCheck additionalArguments: '--scan ./ --format HTML', odcInstallation: 'DC'
                // dependencyCheckPublisher pattern: '**/dependency-check-report.html'
            }
        }
        stage('Trivy FS vulnerability scanning') {
            steps {
                //sh "trivy fs -f json -o result.json ."  //Output as json
                sh "trivy fs ./Client"
            }
        }
        stage('Sonarqube Analysis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Frontend \
                    -Dsonar.projectKey=Frontend '''
                }
            }
        }
        stage ("Building Docker Image"){
            steps {
                sh "echo ********************Building the frontend docker
image*******************************"
                sh "docker build -t frontend-image -f ./Client/Dockerfile ./Client"
                sh "echo ********************Done Building the
image*************************************"
            }
        }
        stage ("Scanning the docker image") {
            steps {
                sh "echo ************************* Scanning the docker image****************"
                sh "trivy image frontend-image"
                sh "************************* Done Scanning the image****************"
            }
        }
        stage ("Pushing the docker image to ECR repo") {
            steps {
                sh "echo *************************Pushing the docker image****************"
                sh "aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 955926712115.dkr.ecr.us-east-1.amazonaws.com"
                sh "docker tag frontend-image:latest 955926712115.dkr.ecr.us-east-
1.amazonaws.com/frontend-image:latest"
                sh "docker push 955926712115.dkr.ecr.us-east-1.amazonaws.com/frontend-image:latest"
                sh "echo *************************The image has been pushed successfully****************"
            }
        }
    }
}
```

Infrastructure:



Our infrastructure is fully hosted on AWS, Designing the infrastructure for a cloud-hosted application involves several considerations to ensure the application is scalable, reliable, secure, and cost-effective. Here are the key factors to consider:

- ***Scalability:***
    - We ensured that the application is scalable either down or up by depending on AWS ASG which helps in this task so when this high traffic volume it will create additional servers to handle the load and it will also remove any unneeded servers for cost effectiveness.
- ***Reliability and Availability:***
    - Redundancy: Having multiple instances of critical components to avoid single points of failure.
    - High Availability (HA): Deploying applications across multiple availability zones or regions.
- ***Monitoring and Logging:***
    - Monitoring Tools: Using tools like CloudWatch to track application performance and health.

- Logging: Setting up centralized logging for easy access and analysis of logs.

- ***Security:***
  - Identity and Access Management (IAM): Using roles and policies to control access to resources.
  - Encryption: Encrypting data both in transit and at rest.
  - Network Security: Configuring firewalls, security groups, and VPCs to protect against unauthorized access.
  - Using private subnets for the servers that will be used for hosting the app and all of the servers will behind an application load balancer.
- ***Cost Management:***
  - Resource Optimization: Right-sizing instances and using reserved or spot instances to save costs. We have chosen the minimum resources that can do the task.
- ***Networking:***
  - Network Design: Designing a network that meets the application's needs, including VPCs, subnets, and peering connections.
  - Latency and Throughput: Ensuring network design supports required latency and throughput levels.

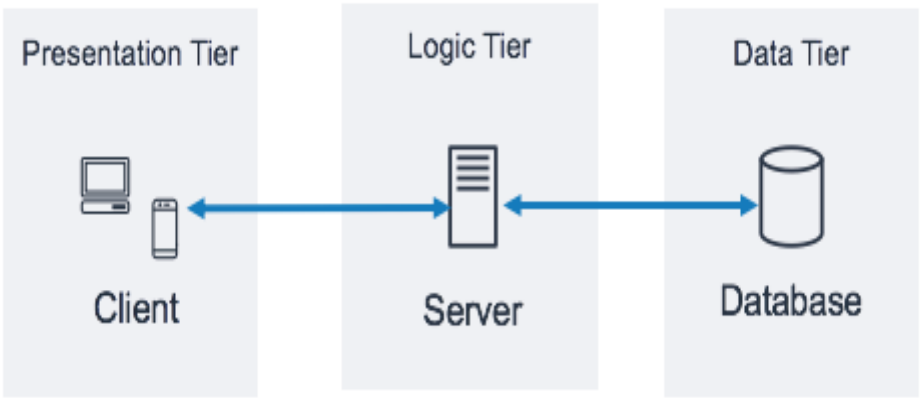
The web application architecture:

In today's cloud-native world, containerization has become a standard practice for deploying and managing applications. As a DevOps engineer, I've had the opportunity to work on a 3-tier application, where we encapsulate each part of the application (frontend, backend, and database) in its own Docker container. This approach provides numerous benefits, including isolation, scalability, and ease of deployment.

Understanding the 3-Tier Architecture:

A 3-tier architecture typically consists of three layers:

1. ***Presentation Layer (Frontend):*** This is the user interface of the application, usually a web application or mobile app that interacts with the user. In our case this layer is written in ReactJS.
2. ***Logic Layer (Backend):*** This layer handles the business logic of the application. It processes user requests, interacts with the database, and returns the appropriate responses. In our case this layer is an ASP .NET application.
3. ***Data Layer (Database):*** This is where the application data is stored, managed, and retrieved. We have used a SQL server database for persisting the data.



Benefits of Using Docker Containers:

Deploying each tier in its own Docker container provides several advantages:

- ***Isolation***: Each component runs in its own isolated environment, ensuring that dependencies and configurations do not interfere with each other.
- ***Scalability***: Containers can be easily scaled up or down based on demand, allowing for flexible resource management.
- ***Consistency***: Containers ensure that the application runs consistently across different environments, from development to production.
- ***Portability***: Docker containers can run on any system that supports Docker, making it easier to deploy across different platforms.

Web Application Security Test

1. Introduction

This document details the security testing of a web application, focusing on various vulnerabilities including Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Server-Side Request Forgery (SSRF), vulnerabilities identified using web scanners, and Remote Code Execution (RCE). The objective of this test is to identify potential security weaknesses that could be exploited by attackers to compromise the application and its users.

2. Overview of XSS Vulnerability

Cross-Site Scripting (XSS) is a type of security vulnerability that allows an attacker to inject malicious scripts into content from otherwise trusted websites. These scripts can be used to steal cookies, session tokens, or other sensitive information.

Types of XSS:

1. **Stored XSS:**

- Malicious scripts are permanently stored on the target server (e.g., in a database).
- Users retrieve and execute the scripts when they request the stored data.
- Example: An attacker inputs a malicious script into a comment section of a website. When other users view the comment, the script runs.

2. **Reflected XSS:**

    - Malicious scripts are reflected off a web server, usually via a URL or form submission.
    - The script is then executed as part of the response.
    - Example: An attacker creates a URL that includes a malicious script and tricks a user into clicking it. The script executes in the user's browser.

3. **DOM-Based XSS:**

- The vulnerability is in the client-side code rather than the server-side code.
- Malicious scripts manipulate the Document Object Model (DOM) of the web page.
- Example: A script dynamically modifies the page's content based on user input without proper validation, allowing for script injection.

How XSS Works:

1. Injection: An attacker identifies a vulnerable input field or URL and injects a script.
2. Execution: The web application processes the input without proper validation or sanitization, delivering the malicious script to users.
3. Impact: The malicious script executes in the user's browser with the same permissions as the web application, allowing attackers to:
    - Steal cookies, session tokens, or other sensitive information.
    - Perform actions on behalf of the user.
    - Redirect users to malicious websites.
    - Deface websites or deliver phishing content.

Consequences:

- Data Theft: Attackers can capture sensitive user data.
- Session Hijacking: Attackers can hijack user sessions to impersonate them.
- Malware Distribution: Users can be redirected to sites that distribute malware.
- Loss of Trust: Users lose trust in the affected web application or website.

Prevention:

1. Input Validation: Ensure all input is validated, sanitized, and escaped.
2. Output Encoding: Encode output data to prevent the browser from interpreting it as code.
3. Content Security Policy (CSP): Implement CSP to restrict sources of executable scripts.
4. Use Secure Libraries and Frameworks: Leverage libraries and frameworks that automatically handle input validation and output encoding.

Example:

<script>alert('XSS Attack!');</script>

Testing Methodology

Manual Scanning

To identify XSS vulnerabilities, a manual scanning process was conducted. This involved:

Reviewing Source Code: Inspecting the source code for common XSS attack vectors.

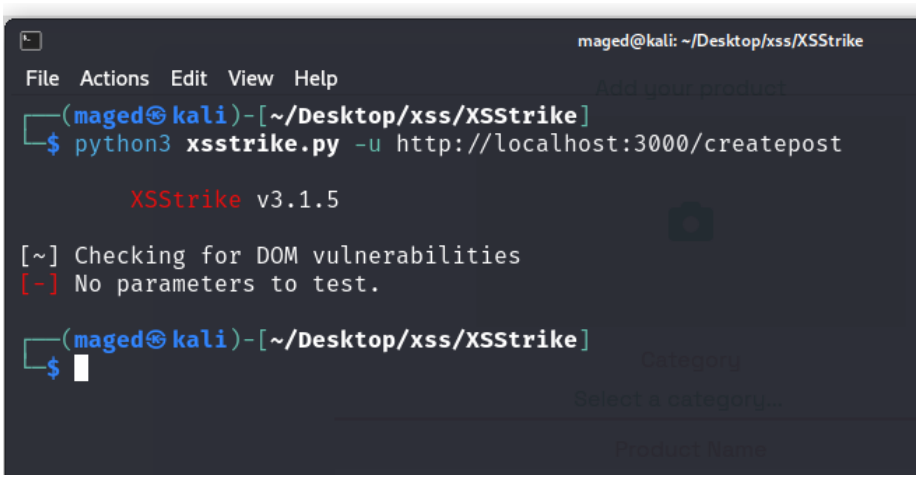
Form and Input Field Testing: Entering various XSS payloads into input fields to observe application behavior.

Reviewing Error Messages: Checking for any error messages or unexpected behaviors that could indicate a vulnerability.

Automated Tools

In addition to manual scanning, automated tools were employed to ensure comprehensive testing. The tools used include:

XSStrike: An advanced XSS detection suite that performs payload fuzzing, context analysis, and intelligent payload generation.



XSSVibes: A tool designed to detect XSS vulnerabilities using a variety of payloads and attack vectors.

Findings

Both manual and automated scanning did not reveal any XSS vulnerabilities in the application. The input fields and parameters appear to be properly sanitized and validated.

Conclusion

The web application does not currently exhibit any XSS vulnerabilities based on the tests conducted.

Recommendations

- **Continuous Monitoring**: Regularly monitor and test the application for XSS vulnerabilities, especially after making changes to the codebase.
- **Input Validation**: Continue to ensure that all user inputs are properly validated and sanitized.
- **Security Headers**: Implement security headers such as Content-Security-Policy (CSP) to mitigate XSS attacks.

3. Overview of CSRF Vulnerability

Overview of CSRF Vulnerability

Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious site tricks a user's browser into performing an unwanted action on a different site where the user is authenticated. This can result in unauthorized actions being performed on behalf of the user.

Testing Methodology

I examined the 'reset your password' feature to identify potential CSRF vulnerabilities. The form allowed users to write the email address, new password, and confirm the new password without token validation.

Findings:

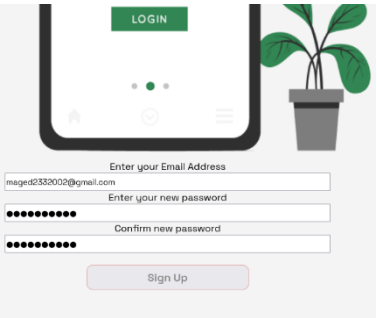
During testing, a CSRF vulnerability was discovered in the "reset your password" feature of the application. The issue arises because the developer made an error, allowing users to reset their passwords by entering their email address, new password, and confirming the new password without validating a CSRF token.

Steps to Reproduce:

1. Navigate to the "Reset Password" page.
2. Enter an email address, new password, and confirm the new password.
3. Submit the form.

4. Observe that the password reset occurs without any CSRF token validation.

This lack of token validation allows an attacker to craft a malicious link or form that can be used to reset the passwords of users without their knowledge.



Conclusion

The application has a CSRF vulnerability that could be exploited to reset users' passwords without their consent.

Recommendations

- **Implement CSRF Tokens**: Use anti-CSRF tokens to validate all form submissions and ensure they are from authenticated users.
- **SameSite Cookies**: Set cookies with the SameSite attribute to prevent CSRF attacks.

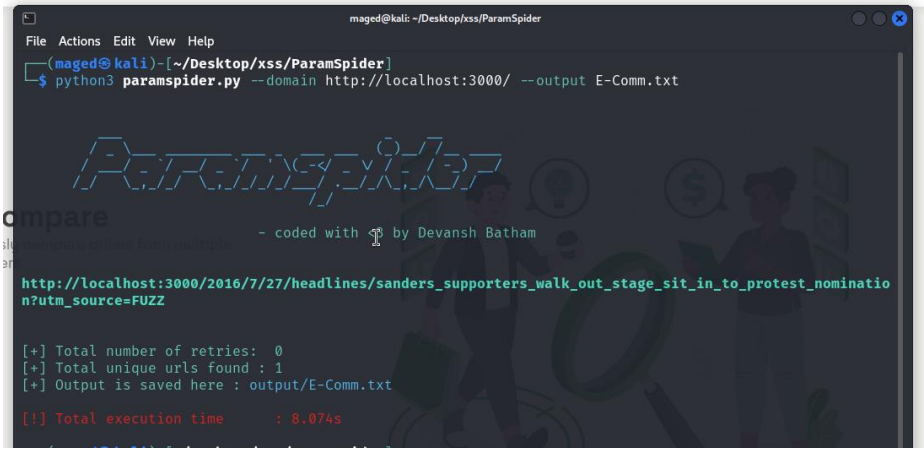
4. Overview of SSRF Vulnerability

Server-Side Request Forgery (SSRF) is a security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an unintended location. This can lead to unauthorized access to internal resources and potentially sensitive information.

Testing Methodology

Automated Tools

To identify SSRF vulnerabilities, the tool **ParamSpider** was used. ParamSpider is designed to gather domains with URL parameters, which can then be tested for SSRF. The tool automates the discovery process, making it easier to identify potential SSRF entry points.



Manual Testing

In addition to automated tools, manual testing was conducted. This involved:

Reviewing URL Patterns: Analyzing the URL patterns used in the application.

Modifying URL Parameters: Manually modifying URL parameters to attempt accessing internal resources.

Findings

No URL parameters were found using ParamSpider, and manual attempts to modify the URL to access internal resources were not allowed.

Conclusion

The web application does not currently exhibit any SSRF vulnerabilities based on the tests conducted.

Recommendations

- **Parameter Validation**: Ensure all URL parameters are properly validated and sanitized.
- **Firewall Rules**: Implement strict firewall rules to control internal network access.

5. Web Scanner Vulnerabilities

During the manual security scan of the web application, web scanners such as Burp Suite were run in the background. This approach was used to understand the site's working mechanism and to uncover potential vulnerabilities that may not be immediately visible through manual testing alone.

Testing Methodology:

Web Scanners

1. **Burp Suite:** A comprehensive web vulnerability scanner used to identify security issues within web applications. It automates the process of crawling the application and testing for common vulnerabilities.
2. **Acunetix**: Another powerful web vulnerability scanner that automates the detection of various security issues. It provides detailed reports on potential vulnerabilities and misconfigurations.

Findings

Both Burp Suite and Acunetix identified two warnings that may lead to potential vulnerabilities:

1. Cross-Origin Resource Sharing (CORS)

Description: Cross-Origin Resource Sharing (CORS) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the resource originated. Improperly configured CORS policies can lead to security vulnerabilities by allowing unauthorized domains to access sensitive information.

Risk: If the CORS policy is too permissive, it may allow malicious sites to read sensitive information, leading to data breaches or unauthorized access.

Recommendation:

- Review and tighten the CORS policy to ensure only trusted domains are allowed.

- Implement proper validation and security measures to protect sensitive data from unauthorized access.

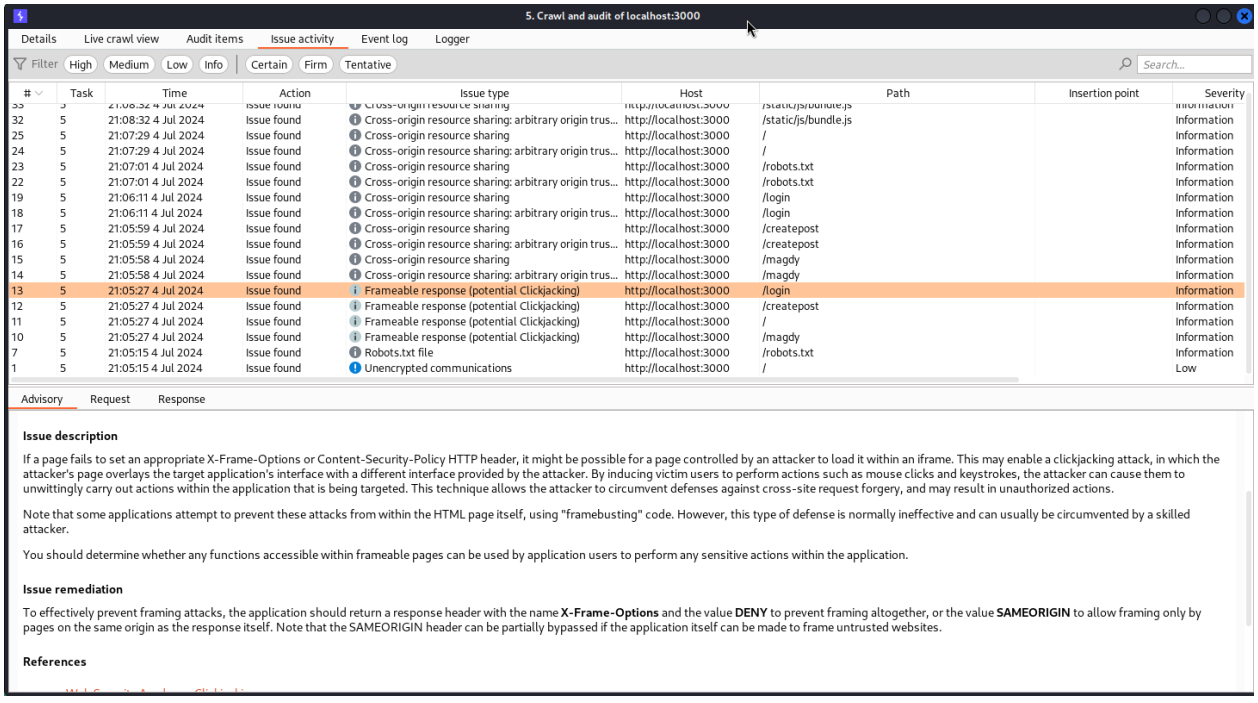
2. Frameable Response (Potential Clickjacking)

Description: Clickjacking is a type of attack where a malicious site tricks a user into clicking on something different from what the user perceives, potentially revealing confidential information or taking control of their computer while clicking on seemingly harmless web pages.

Risk: Allowing the application to be framed by other sites can expose users to clickjacking attacks, leading to unauthorized actions being performed without the user's consent.

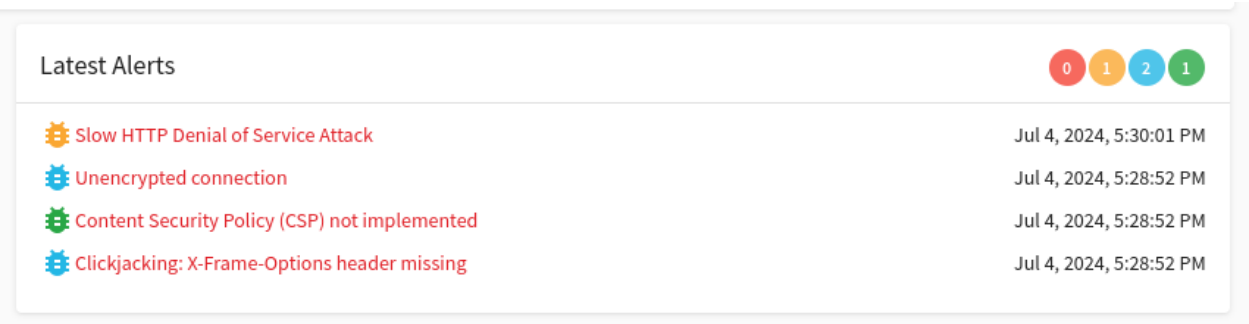
Recommendation:

- Implement the X-Frame-Options header to prevent the application from being framed.
- Use the Content-Security-Policy header with the frame-ancestors directive to control which sites can frame the application.



Additional Findings from Acunetix

In addition to the warnings identified by Burp Suite, Acunetix also reported the following:



3. Firewall

Description: The scan identified issues related to the firewall configuration. This is typically expected when testing on localhost and may not represent an actual vulnerability in a production environment.

Recommendation:

- Review firewall settings to ensure they are appropriately configured for the production environment.

4. HTTPS

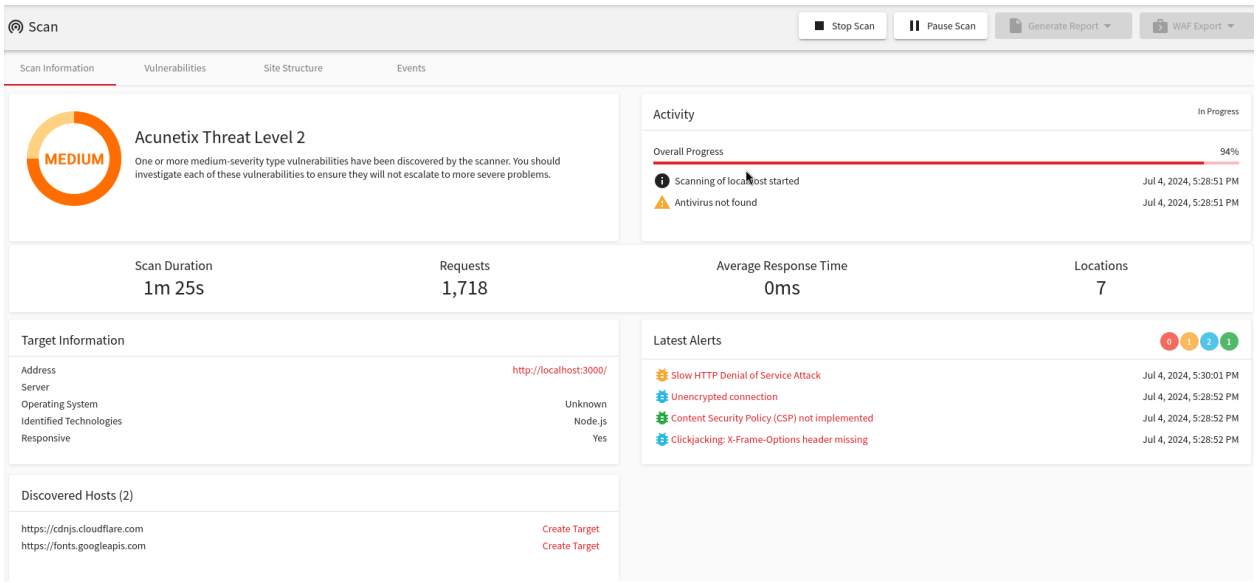
Description: The scan flagged the absence of HTTPS, which is common when testing on localhost. However, this is critical in a production environment to ensure data is transmitted securely.

Recommendation:

- Ensure HTTPS is enabled and properly configured in the production environment to protect data in transit.

Conclusion

The use of web scanners identified several potential vulnerabilities and misconfigurations. While some findings are specific to the testing environment (localhost), others warrant further investigation and remediation to ensure the security of the application.



wrote a security report to the back-end Developer **Eng: Mohamed Mofreh** to Mitigate these warning in the code By Adding Security Headers to any Request.

- **Security Report: Web Scanner Warnings**

Introduction

During the security testing of our web application, web scanners were run in the background to analyze the site's working mechanism and uncover potential vulnerabilities. Two warnings were identified that may lead to security vulnerabilities: Cross-Origin Resource Sharing (CORS) and Frameable Response (Potential Clickjacking).

Warnings

1. Cross-Origin Resource Sharing (CORS)

Description: CORS is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the resource originated. Improperly configured CORS policies can lead to security vulnerabilities by allowing unauthorized domains to access sensitive information.

Risk: If the CORS policy is too permissive, it may allow malicious sites to read sensitive information, leading to data breaches or unauthorized access.

Recommendation:

- Review and tighten the CORS policy to ensure only trusted domains are allowed.
- Implement proper validation and security measures to protect sensitive data from unauthorized access.

2. Frameable Response (Potential Clickjacking)

Description: Clickjacking is a type of attack where a malicious site tricks a user into clicking on something different from what the user perceives, potentially revealing confidential information or taking control of their computer while clicking on seemingly harmless web pages.

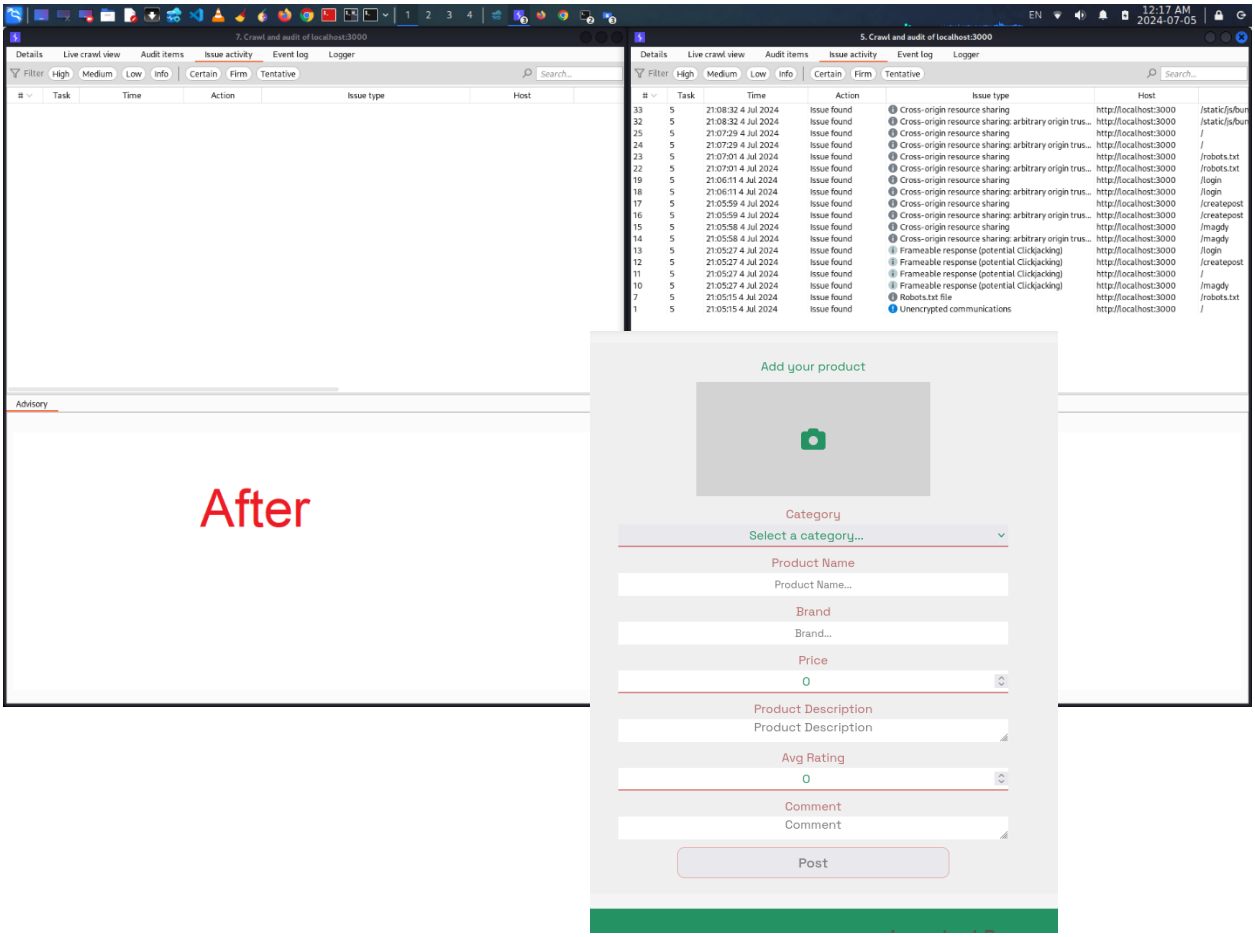
Risk: Allowing the application to be framed by other sites can expose users to clickjacking attacks, leading to unauthorized actions being performed without the user's consent.

Recommendation:

- Implement the X-Frame-Options header to prevent the application from being framed.
- Use the Content-Security-Policy header with the frame-ancestors directive to control which sites can frame the application.

The Back-End Engineer replied to me that he had Fixed the warnings

I started the Burp Suite again to make sure there were no warnings



6. RCE Vulnerability

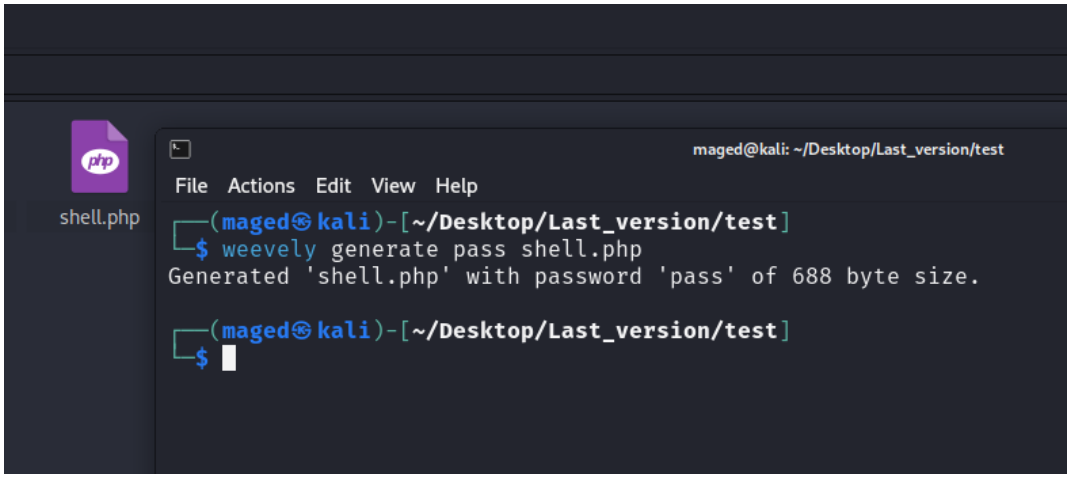
Remote Code Execution (RCE) is a critical security vulnerability that allows an attacker to execute arbitrary code on the server hosting the web application. This can lead to full system compromise and data breaches.

Testing Methodology

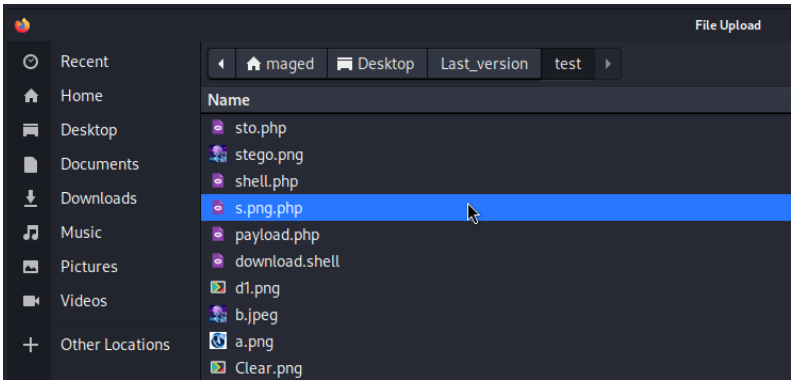
To test for RCE vulnerabilities, the validation mechanisms of the project upload form were examined. The form was supposed to accept only image files with specific extensions (png, jpeg, jpg).

Steps Taken:

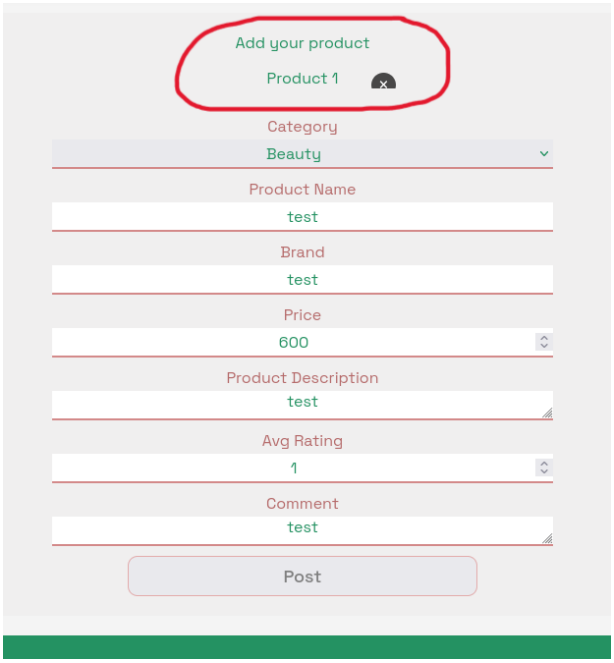
1. **Initial Validation Check**: Verified that the form only accepted files with png, jpeg, and jpg extensions.
2. **Generating a Malicious Shell**: Used the Weevely tool to generate a PHP shell. The generated shell was named shell.php.

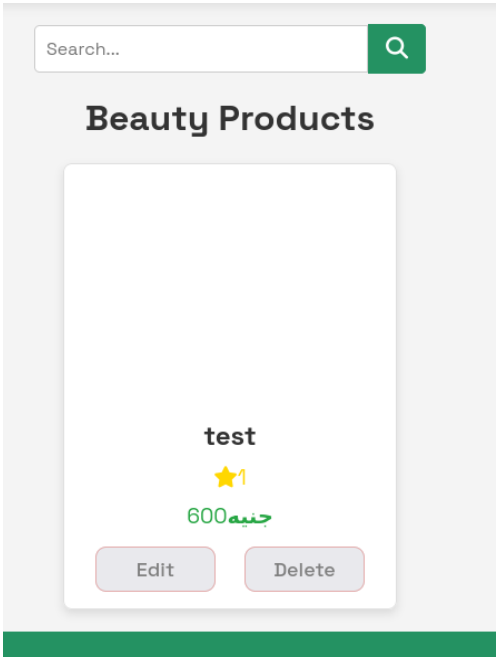


3. **Renaming the Shell**: Renamed the generated shell to image.png.shell.



4. **Uploading the Shell**: Attempted to upload the renamed shell to the form.





Findings

The upload form accepted the file with the .shell extension as an image, despite the additional extension. This indicates that the validation mechanism is flawed, and the application does not properly validate the file type, leading to a potential RCE vulnerability.

Steps to Reproduce:

1. Generate a shell using Weevely: weevely generate password shell.php.
2. Rename the generated shell to image.png.shell.
3. Navigate to the upload form.
4. Select the image.png.shell file and upload it.
5. The form accepts the file without any errors, treating it as a valid image.

Conclusion

The presence of an RCE vulnerability in the upload form is a significant security risk. It can lead to unauthorized code execution, system compromise, and data breaches.

Recommendations

To mitigate the risk of RCE attacks, it is recommended to:

1. Improve File Validation: Implement stricter file validation checks to ensure that only valid image files are accepted. This should include validating the file's MIME type and extension.
2. Sanitize File Uploads: Sanitize and validate all file uploads on the server side to prevent malicious files from being uploaded.
3. Use Secure Libraries: Utilize secure libraries and functions for handling file uploads to prevent common vulnerabilities.
4. Regular Security Audits: Conduct regular security audits to identify and address potential vulnerabilities in the file upload functionality.

References: -

1- W3Schools: HTML

2-W3Schools: CSS

3-W3Schools: JavaScript

4-MDN Web Docs: Node.js

5-https://www.crowdstrike.com/cybersecurity-101/secops/vulnerability-assessment/

6-https://owasp.org/www-community/Threat_Modeling

7-https://circleci.com/blog/sast-vs-dast-when-to-use-them/

8-https://www.hackerone.com/knowledge-center/what-ssdlc-secure-software-development-life-cycle

9-https://datadome.co/learning-center/what-is-ssl-tls-encryption/

10-https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver16

11- Simple and Efficient Programming with C# Skills to Build Applications with Visual Studio and .NET - Second Edition - Vaskaran Sarcar.

12- Coding Clean, Reliable, and Safe REST APIs with ASP.NET Core 7 Develop Robust Minimal APIs with .NET 7 -Anthony Giretti.

13-https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&tabs=visual-studio

14- https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli

15- https://learn.microsoft.com/en-us/dotnet/csharp/linq/

16-OWASP. (2024). Remote Code Execution. Retrieved from OWASP.

17-OWASP. (2024). File Upload Security. Retrieved from OWASP.

18-Weevely. (2024). Weevely Documentation. Retrieved from Weevely.