2808ICT/7623ICT Assignment 2

Automate and Build Full-stack Project

Team: XXXTeam

Members:

Yip Hing Wa , Tom (s5252235)

Arnaud TACHNAKIAN (s5307225)

Table of Contents

[Dockerfiles 3](#_Toc116751443)

[frontend/Dockerfile: 3](#_Toc116751444)

[mongo-express/Dockerfile: 3](#_Toc116751445)

[mongodb/Dockerfile: 3](#_Toc116751446)

[proxy/Dockerfile: 3](#_Toc116751447)

[Content of docker-compose.yml 8](#_Toc116751448)

[Automate Security Testing of the Project 10](#_Toc116751449)

[Automate Test 10](#_Toc116751450)

[Manual Scan: 11](#_Toc116751451)

[Final Results: 12](#_Toc116751452)

[Code issue 1: 12](#_Toc116751453)

[Code issue 2: 14](#_Toc116751454)

# Dockerfiles

## frontend/Dockerfile:

#Each instruction in this file creates a new layer

#Here we are getting our node as Base image

FROM node:latest

#Creating a new directory for app files and setting path in the container

RUN mkdir -p /usr/src/app

#setting working directory in the container

WORKDIR /usr/src/app

#copying the package.json file(contains dependencies) from project source dir to container dir

COPY package.json /usr/src/app

# installing the dependencies into the container

RUN npm install

#copying the source code of Application into the container dir

#note you must be in the source code directory and remember COPY is recursive

COPY . /usr/src/app

#container exposed network port number

EXPOSE 3000

#command to run within the container

CMD ["node", "server.js"]

## mongo-express/Dockerfile:

#Here we are getting mongo-express as Base image

FROM mongo-express

#container exposed network port number

EXPOSE 8081

#command to run within the container

CMD ["mongo-express"]

## mongodb/Dockerfile:

#Here we are getting mongo as Base image

FROM mongo

#container exposed network port number

EXPOSE 27017

## proxy/Dockerfile:

#Here we are getting nginx as Base image

FROM nginx

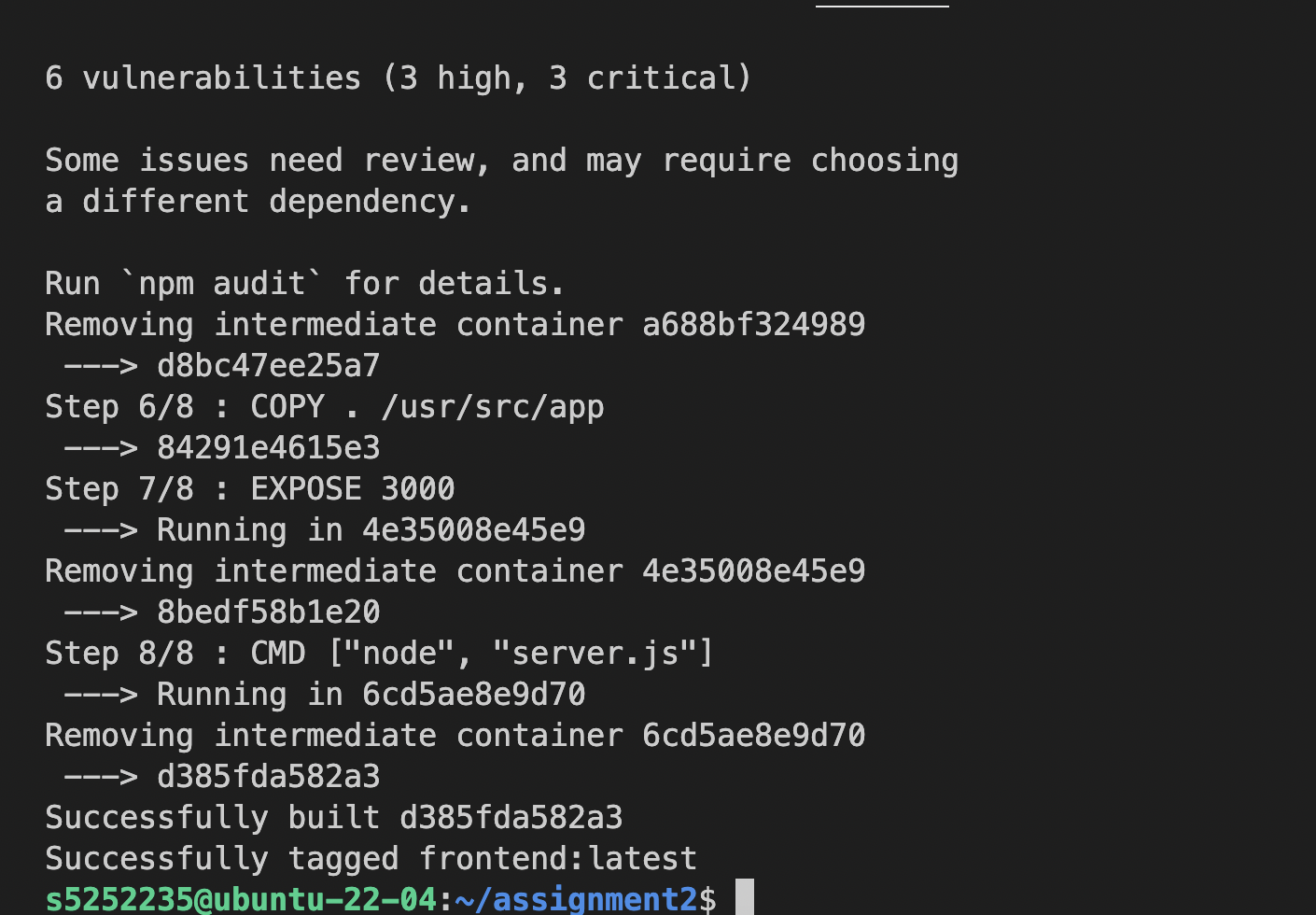


Fig 1a. Successful built of frontend images

Text

Description automatically generated

Fig 1b. Successful built of mongo-express images

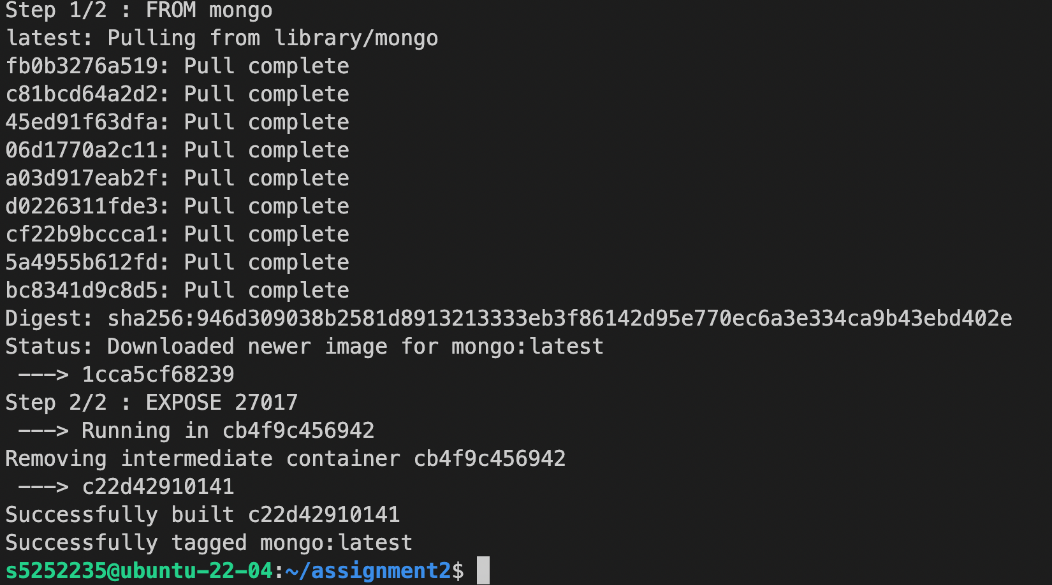


Fig 1c. Successful built of mongoDb images

Text

Description automatically generated

Fig 1d. Successful built of proxy images

Text

Description automatically generated

Fig 1e. Successful launch of docker-compose & proxy image

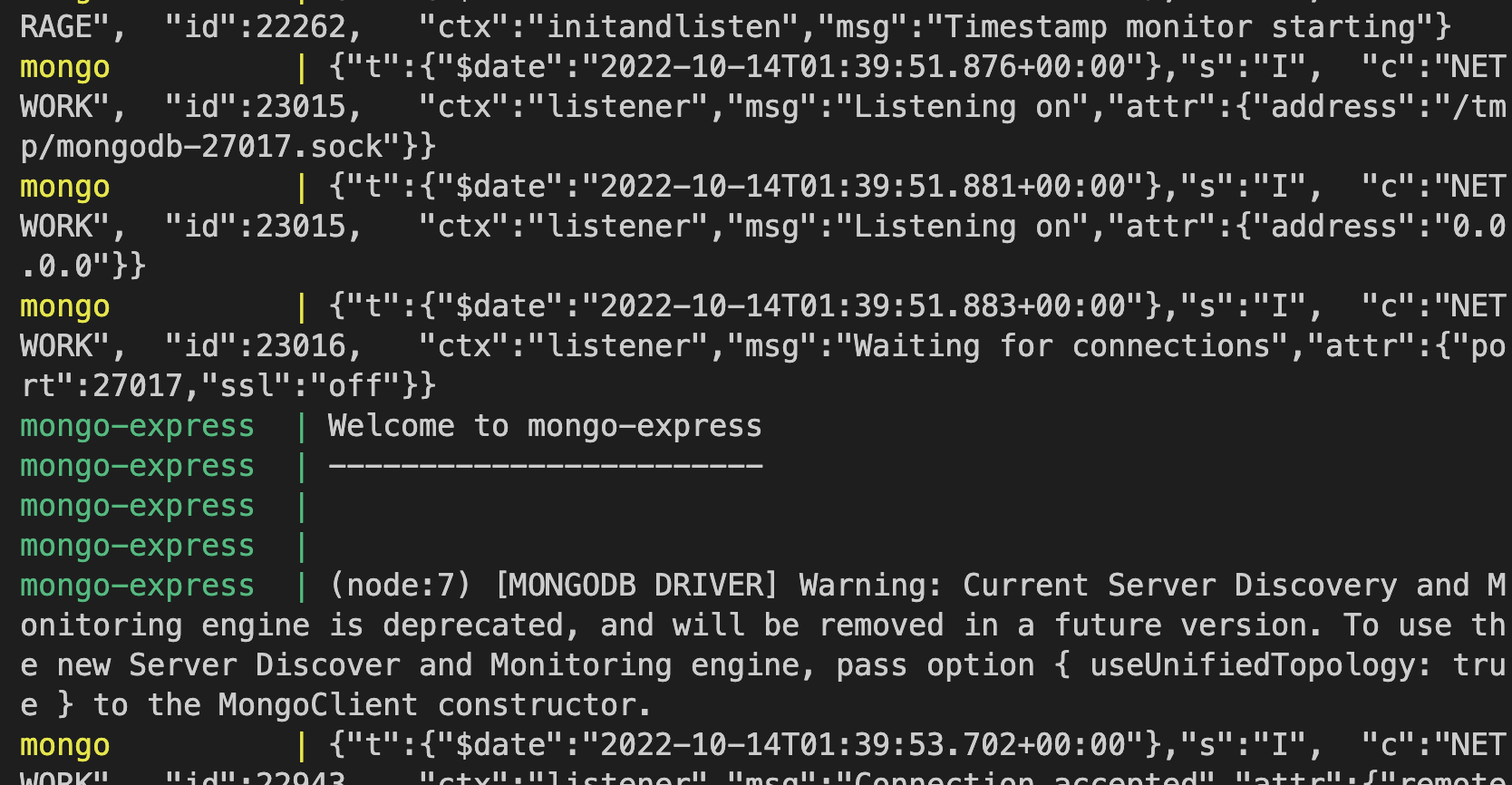


Fig 1f. Successful launch of mongo image & mongo-express images

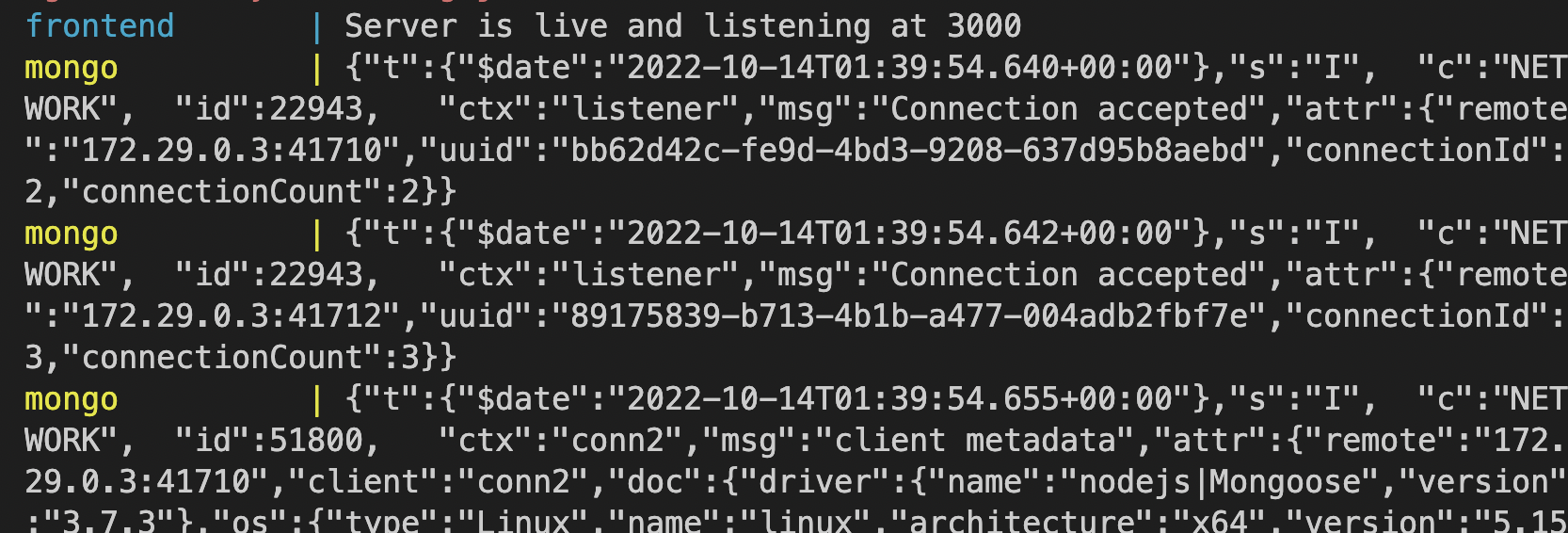


Fig 1g. Successful launch of frontend images

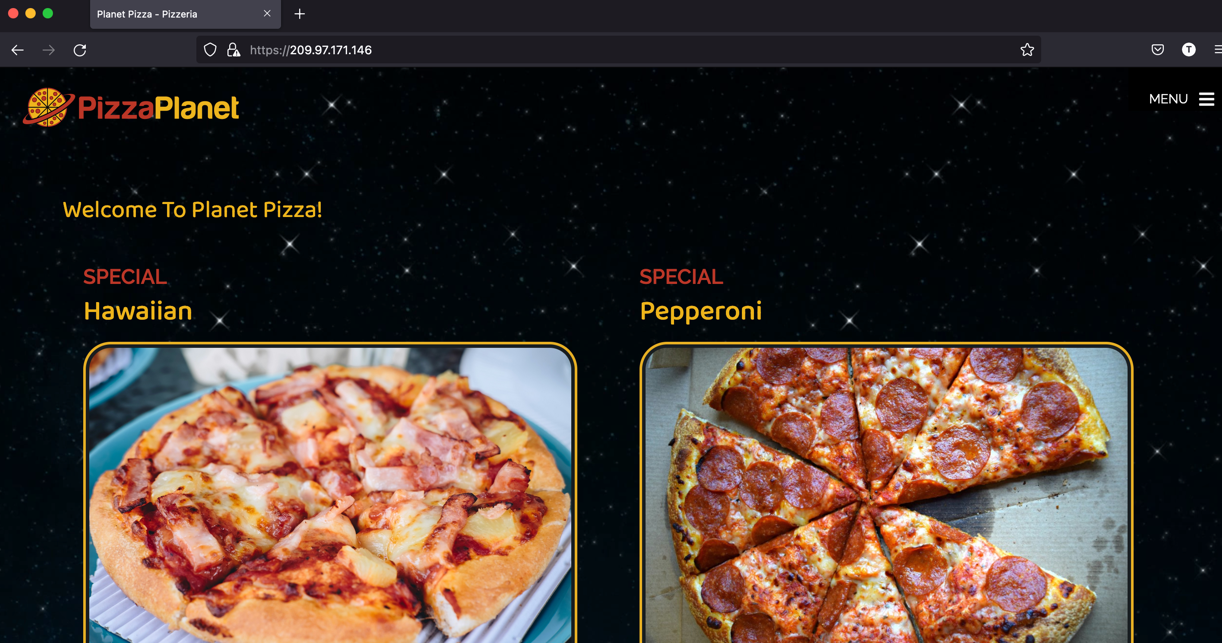


Fig 1h. Successful launch of frontend application



Fig 1g. Successful launch of mongo-express application

# Content of docker-compose.yml

version: "3.9"

services:

frontend:

container\_name: frontend

image: frontend

build: ./frontend

environment:

- MONGODB\_URI=mongodb://mongo:27017/pizzeria

- PORT=3000

- SECRET=aslbljwerljwer

depends\_on:

- mongo

networks:

- frontend

- backend

mongo:

container\_name: mongo

image: mongo

build: ./mongodb

ports:

- "27017:27017"

volumes:

- mongodata:/data/db

networks:

- backend

mongo-express:

container\_name: mongo-express

image: mongo-express

build: ./mongo-express

environment:

- ME\_CONFIG\_MONGODB\_SERVER=mongo

- ME\_CONFIG\_MONGODB\_PORT=27017

links:

- mongo

networks:

- frontend

- backend

proxy:

container\_name: proxy

image: nginx

build: .

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf:ro

- ./ssl/localhost.crt:/etc/ssl/certs/localhost.crt

- ./ssl/localhost.key:/etc/ssl/private/localhost.key

ports:

- "3000:3000"

- "8081:8081"

- "443:443"

networks:

- frontend

volumes:

mongodata:

networks:

frontend:

backend:

internal: true

# Automate Security Testing of the Project

Automate Test on [***https://209.97.171.146/***](https://209.97.171.146/)

Graphical user interface, text, application

Description automatically generated

After an automate test on our website, we can find 11 alerts. Among these alerts we have 5 Medium Level, 4 Low Level and 2 Informational Level. However, the test is not complete.

The automate test can only access the basic functionalities and requests of our website. To get deeper in our analyse, we need to make a manual test. With this manual scan, ZAP can identify more errors in certain domains that the automate test can’t access like the functionalities that requires an authentication. A manual scan is also mandatory in order to identify false positive errors.

Graphical user interface, text, application

Description automatically generated

## Manual Scan:

For the Manual Scan, I start by visiting the home page to see if any errors can be found. As you can see in the screenshot, we can find 2 medium level errors, 1 low level error and 1 informational error. The 2 medium errors are linked to CSP.

Graphical user interface, application

Description automatically generated

After looking at the welcome page, I analyse the Register/Login page. One major error identified by the ZAP manual scan is the lack of Anti-CSRF Token. The absence of this Anti-CRS Token can be very harmful for the user in case of cyber-attack because the client is asked to enter his bank information. If a hacker managed to access the information contained in this form, he would retrieve all this personal data linked to a client.

Graphical user interface, application

Description automatically generated

## Final Results:

Graphical user interface, text, application

Description automatically generated

## Code issue 1:

Project: 7623ICT-Assignment2-2022

Code Issue Number: CWE-352

Code Sample and Location: Registration Form from line 10 to line 165 in *register.ejs*

Text

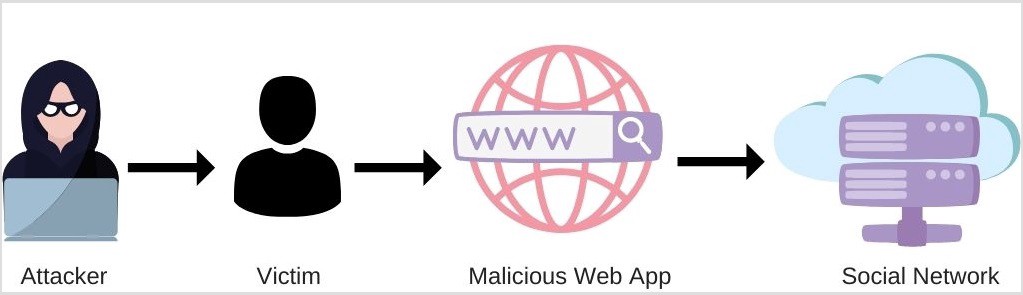
Description automatically generated

Issue Type/Class: Absence of Anti-CSRF Token

Description: The principle of Anti-CSRF Token allows to secure the exchange between the client and the server. When communicating with our server, through a form for example, we generate a token that will be attached to this form. Once the user has filled in his information and submitted the form back, it is received by the server. Before manipulating the data, the server will check the value of the token. If the token received is the same as the one generated previously by the server, then the form is accepted, otherwise it is rejected.

The objective of the hacker is to create a malicious form hosted on another domain. If the user is logged-in on the original website and is tricked to access the malicious site, the form created by the hacker can be submitted. With this form, the hacker can execute scripts, access sensitive data or even modify some user values like his username or password.

**CSRF Attack Diagram**



*Source :* [*https://blog.sessionstack.com/how-javascript-works-csrf-attacks-7-mitigation-strategies-757dfb08e7a6*](https://blog.sessionstack.com/how-javascript-works-csrf-attacks-7-mitigation-strategies-757dfb08e7a6)

Verification: To verify this issue, we should create another website hosted on another domain, with a form that will do certain malicious actions when it is submitted.

Impact: With no protection from this type of attack, cross-site request forgery can compromise and expose the confidentiality and integrity of the website’s user data. In our case, it can be their address and even worse their card number.

Proposed Remediation: The solution to counter CSRF attacks is to implement anti-CSRF tokens in every form used by the client. The easiest way to implement anti-CSRF token is through hidden fields or headers.

Here is an example on line 11:

Text

Description automatically generated

## Code issue 2:

Project: 7623ICT-Assignment2-2022

Code Issue Number: CWE-829

Code Sample and Location:

Text

Description automatically generated

Issue Type/Class: Vulnerable JS Library

Description: Some scripts executed on the website are using a library called jquery-3.4.1 which is identified as a vulnerable library. After looking on Google, this version of jQuery is vulnerable because it allows the user to pass HTML containing elements from untrusted sources to one of jQuery’s DOM manipulation methods. By doing so, an untrusted user can execute code

Impact: By using this version of the jQuery library, the site is now vulnerable to malicious attacks like Cross-Site Scripting (XSS).

Proposed Remediation: Update to a latest and safest version of jQuery. To find a better version, we can look on the latest jQuery releases and look the patch notes. jQuery website and other articles talking about this vulnerability suggest using the 3.5.0 or higher version.