



# Security Documentation

INFO2222 ASSIGNMENT2

*RE11-T14A: Jayden Zhang (490193404), Faye He (500030507),  
Wenxi Xu (500175196), Scarlett Hu (480114491), Rui Wang (500174384)*

*GitHub link: <https://github.sydney.edu.au/INFO2222-2021S1/RE11-T14A3>*

*Link to Last Commit: <https://github.sydney.edu.au/INFO2222-2021S1/RE11-T14A3/commit/fe296d4a68270e3f049c79a64e7adf0837d43e68>*

# Table of Contents

<b>1. Deployment.....</b>	<b>3</b>
a. VM SSH Settings.....	3
b. Different Users and Groups .....	3
c. Storage of Important Keys .....	4
e. Start-up Script.....	5
<b>2. Virtual Users.....</b>	<b>6</b>
a. Ordinary user who visits our website for the very first time .....	6
b. The admin user .....	7
c. The tourist.....	7
<b>3. Options.....</b>	<b>8</b>
a. Confidentiality - HTTPS.....	8
b. Availability - WSGI + Web server deployment .....	11

# 1. Deployment

## a. VM SSH Settings

In order to protect our VM from unwanted SSH logins, we have configured the `sshd_config` file and modified some of the settings.

- **Permitrootlogin no:**  
By changing this setting to no, we have disabled the ability to SSH login as root completely. By doing so, we can prevent hackers from gaining the highest permission of the system if they somehow get the password or bypass the authentication process.
- **PasswordAuthentication no & ChallengeResponseAuthentication no:**  
By changing these two settings to no, we disabled the ability to SSH login using plain text password completely for any users. The only way to login to a user is through SSH keys that we set up before disabling these options.
- **Match User groupAcc  
PasswordAuthentication yes:**  
By enabling this option, it only turns on password authentication for the user `groupAcc`. This is for the conveniences of our members as this user is mainly used to git pull and also start the server. This user has been stripped of many permissions, for example, it can't perform operations requiring sudo permission. This is done to improve the security of our server.

## b. Different Users and Groups

Inside the VM, there are two types of users: one that has sudo access, and one that doesn't. For the user that has sudo access, we put an extra layer of protection on them by hashing their passwords. By doing so, the time to use brute force to gain sudo access would be extremely long, and would result in a timeout after several attempts by the system anyway.

For the user that doesn't have sudo access, it is mainly used for simple server-related operations such as pulling code from the Git repository and starting the uwsgi application. They only have permissions directly required to perform these operations and nothing else.

### c. Storage of Important Keys

To pull project files directly from the repository to the VM, we have used the deployment key feature of GitHub. We first generated a pair of RSA keys by using ssh-keygen. We then copied the value of the public key to the deployment key section of the GitHub.

This deployment key only has read access to the repository and can thus prevent hackers from uploading malicious files into our repository. There are two sets of such key pairs, one for each user that we use on a daily basis. Sharing of keys is definitely not recommended, and since we have two users that need to perform git pull, we then created two different sets of key pairs for each user. The keys are also only readable and accessible by the user who created them.

### d. Domain

The domain name is purchased at **Alibaba Cloud**.

Domain Name	Domain Type	Status	Registration Date	Expiration Date
<input type="checkbox"/> abaweb.work	New gTLD	Normal	2021-05-07 16:23:37	2022-05-07 16:23:37

Figure 1.1 Domain name purchased at Alibaba Cloud

Add Record		Import & Export		Query Volume		Quick Start		ALL ▾		Exact Search ▾		Search by keyword. <input type="text"/>		Advanced Search ▾	
<input type="checkbox"/>	Host ▾	Type ▾	Line(ISP) ▾	Value		TTL		Status		Remark		Actions			
<input type="checkbox"/>	www	A	Default	10.86.225.118		10 minute(s)		Normal				<a href="#">Edit</a>   <a href="#">Disable</a>   <a href="#">Delete</a>   <a href="#">Remark</a>			

Figure 1.2 Domain Name Resolution

The SSL certification is also purchased at **Alibaba Cloud**.

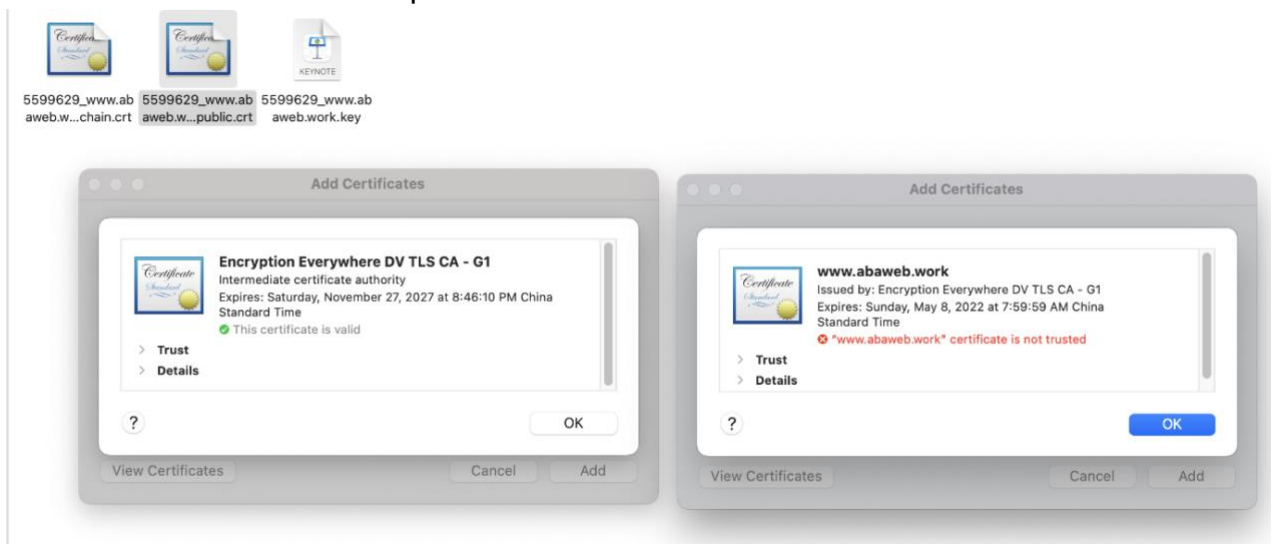


Figure 1.3 Certificates

## e. Start-up Script

In case of unexpected system reboot, we want to make sure our server is back up and running as soon as possible. However, the system administrator, me, is unable to know if the system reboots and when it happens immediately. I would then need to automate the start-up process of our server.

We then wrap the commands we need to start the server in a custom service called `my_uwsgi.service`:

```
[Unit]
Description=A service to automatically run a bash script that starts the uwsgi application on boot.
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/bash -c 'cd /home/rh/RE11-T14A3/code; source py36-venv/bin/activate; uwsgi --ini uwsgi.ini'

[Install]
WantedBy=multi-user.target
```

Coupled with the command:

```
sudo systemctl enable my_uwsgi.service
```

The system will now automatically start this service on system boot and we no longer need to manually start the server whenever the system reboots.

## 2. Virtual Users

We have created three virtual users mimicking real users to perform actions which we consider a normal person will go through when visiting our website. We use selenium for automated functional testing.

You can perform the action by goes to the VirtualUser folder and running `python3 ./main.py`.

### a. Ordinary user who visits our website for the very first time

This user will firstly sign up with xuhe4716 as her username. She is interested in learning the bottle package, so she clicks on Bottle in the header immediately after signing up, she has managed to finish reading tutorial content up to *How to handle GET/POST requests?*

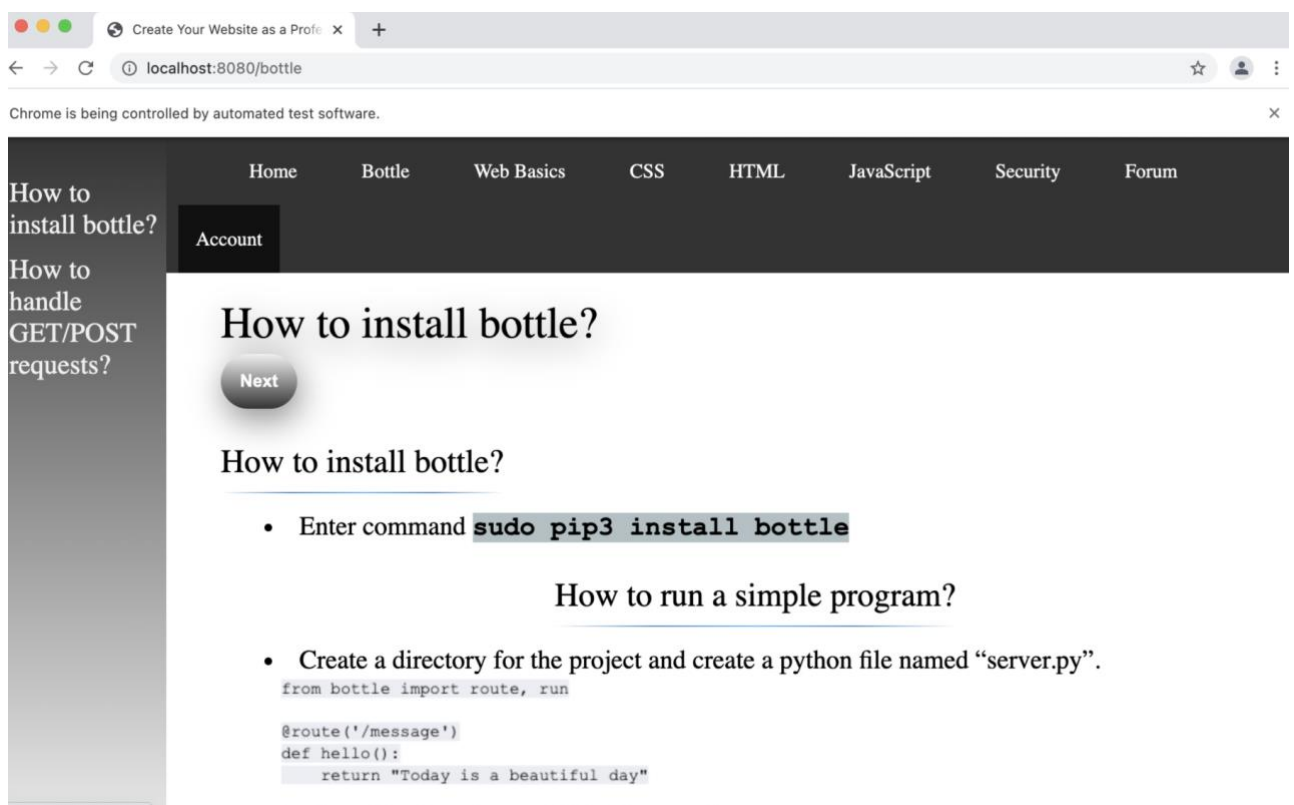


Figure 2.1 Automation using selenium

She then reviews the first tutorial and then jumps to the web basics section by clicking on the corresponding button in the header on the top of the page. She goes through every topic that we have provided back and forth. She feels pretty satisfied with what we have done so decides to learn about us, thus she clicks on About and starts to read the information about the website. She then logs out by redirecting herself to the Account page and click on the "Log out" button.

## b. The admin user

The admin user named "BillyKings" is fond of using his power. After logging in successfully, he goes straight to the Account page by clicking on it in the header and is excited to manage users. He firstly clicks on add users to create new accounts for his friends *jzha5330*, *sihu9591*, *rwan4990* and *wexu0321*. However, since the username *wexu0321* has been taken, he fails to add her.

Since the other users are added successfully, he goes back to the Account page, clicks on Manage Users and mute the newly added users because he does not want his friends to expose him on the internet. He then goes back and deletes the existing user *wexu0321* merely because *wexu0321* has taken the place of his friend which annoys BillyKings. He then promoted *sihu9591* to be an admin.

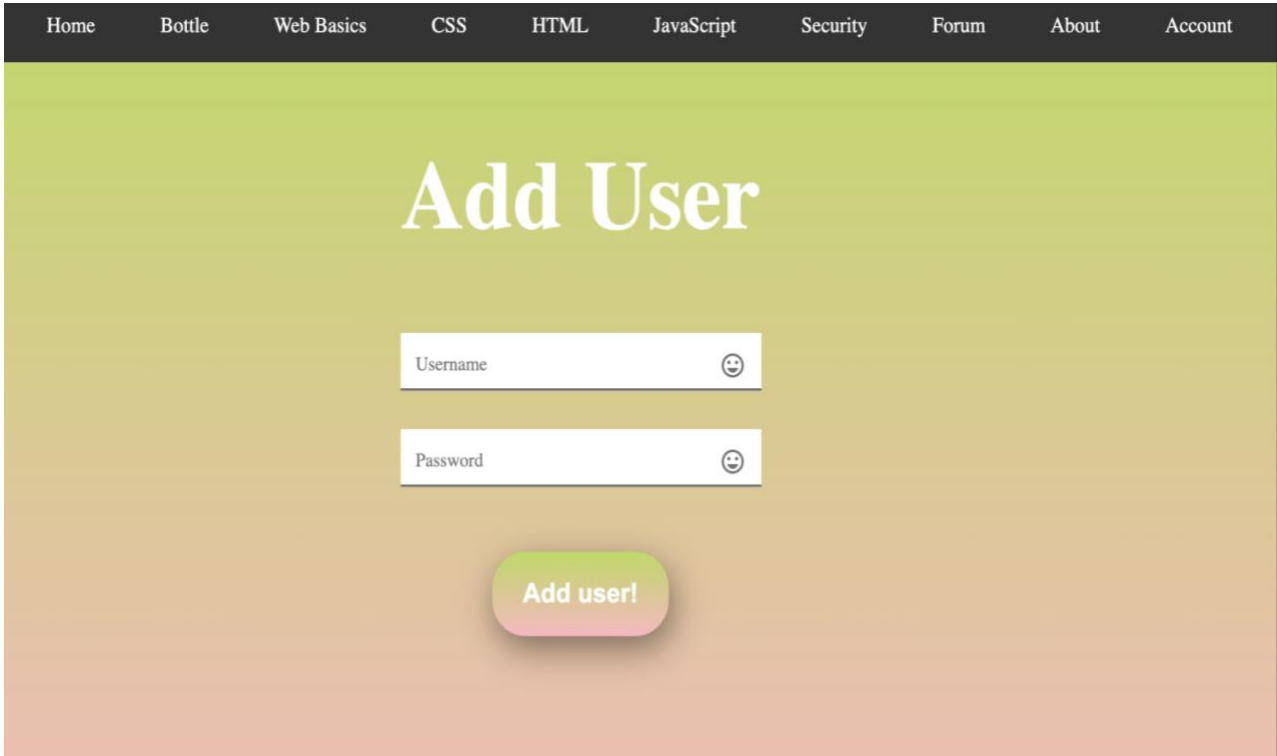
The image shows a web application interface for adding a new user. At the top, there is a dark navigation bar with links: Home, Bottle, Web Basics, CSS, HTML, JavaScript, Security, Forum, About, and Account. The main content area has a light green to yellow gradient background. The title 'Add User' is displayed in a large, white, serif font. Below the title are two input fields: 'Username' and 'Password', both with a small smiley face icon on the right. Below these fields is a large, rounded, green button with a gradient and the text 'Add user!' in white.

Figure 2.2 Add new users as an admin

He then logs in as *rwan4990* because he has got the password! He checks whether *rwan4990* is muted successfully by clicking on Forum in the header and try to post a thread and send messages to *BillyKings*. Of course, both operations fail, which makes his day. He then logs out.

## c. The tourist

Without logging in, the tourist can only check tutorials by clicking on one of the topics in the header. This tourist decides to have a look at **bottle** and then jumps about to learn what the website is about.

## 3. Options

### a. Confidentiality - HTTPS

In order to enable HTTPS for all routes for our server, we have to make use of NGINX and also SSL certificates. Our team first registered a domain name and we set the A records of the DNS server to our server's IP address.

The domain name now points to our server by the DNS server. The requests coming in could be either HTTP or HTTPS, we then need an intermediate layer to redirect the HTTP requests into HTTPS requests and pass the requests to our server. We do so by incorporating NGINX into our project.

In order to use NGINX to enable HTTPS, we also need SSL certificates. There are many ways to generate a SSL certificate. My way of doing this is to use certbot to generate a SSL certificate. However, there is a problem whereas certbot by default uses http-01 challenge to check for validity of the server.

Since our server is in an intranet and we don't have any servers running to catch such requests, we can't pass the http-01 challenge and thus couldn't get the certificate. Fortunately, I managed to find a workaround where we manually force the certbot to use the DNS-01 challenge, it only requires that Certbot can communicate with the DNS server. With some more configurations, we managed to pass both challenges for abaweb.work and [www.abaweb.work](http://www.abaweb.work).



The full configuration is shown as follows:

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile        on;
    tcp_nopush      on;
    tcp_nodelay      on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include         /etc/nginx/mime.types;
    default_type     application/octet-stream;

    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options "SAMEORIGIN";

    server {
        listen 80;
        server_name abaweb.work www.abaweb.work;
        return 301 https://www.abaweb.work$request_uri;
    }

    server {
        listen 443 ssl default_server; #listen on port 80
        listen [::]:443 default_server ipv6only=on;
        ssl_certificate /etc/letsencrypt/live/abaweb.work/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/abaweb.work/privkey.pem;
        server_name abaweb.work www.abaweb.work; #edit 'yourdomain' with your domain name
        root /home/rh/RE11-T14A3/code; #edit to match wherever your bottle-py root folder is
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_prefer_server_ciphers on;

        location / {
            limit_except GET HEAD POST { deny all; }
            include uwsgi_params;
            uwsgi_pass 0.0.0.0:8080;
        }
    }
}
```

```
ssl_certificate /etc/letsencrypt/live/abaweb.work/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/abaweb.work/privkey.pem;
```

After getting a SSL certificate, we need to declare the path of the SSL certificate and also the path of the certificate private key in the configurations of NGINX.

```
add_header X-XSS-Protection "1; mode=block";
add_header X-Frame-Options "SAMEORIGIN";
```

The two uncommented options are security settings that we have included in NGINX to prevent XSS attacks and also clickjacking attacks.

The first header stops pages from loading when they detect reflected cross-site scripting (XSS) attacks. With the second header, we tell the browser not to embed our web page in frame/iframe unless it is from the same origin.

```
ssl_protocols TLSv1.2 TLSv1.3;  
ssl_prefer_server_ciphers on;
```

These two settings ensure that old TLS protocols are not used as it is prone to attacks such as the BEAST attack. We can also let the decision on which ciphers to use be made on server-side not client-side.

```
location / {  
    limit_except GET HEAD POST { deny all; }  
}
```

Lastly, this location block is to redirect the requests from the NGINX virtual server to the uwsgi application that is listening on another port. This setting enables NGINX to filter the requests that are not of the type “GET HEAD POST”.

```
server {  
    listen 80;  
    server_name abaweb.work www.abaweb.work;  
    return 301 https://www.abaweb.work$request_uri;  
}  
  
server {  
    listen 443 ssl default_server; #listen on port 80  
    listen [::]:443 default_server ipv6only=on;  
    ssl_certificate /etc/letsencrypt/live/abaweb.work/fullchain.pem;  
    ssl_certificate_key /etc/letsencrypt/live/abaweb.work/privkey.pem;  
    server_name abaweb.work www.abaweb.work; #edit 'yourdomain' with your domain name  
    root /home/rh/RE11-T14A3/code; #edit to match wherever your bottle-py root folder is  
  
    location / {  
        limit_except GET HEAD POST { deny all; }  
        include uwsgi_params;  
        uwsgi_pass 0.0.0.0:8080;  
    }  
}
```

These two server blocks indicate the layering of the NGINX servers.

In order to intercept requests sent to our domain name, we will have one of the virtual servers to listen on port 80, which is the default port number when the browser sends a HTTP request. The server will intercept the request and redirect it as a HTTPS request with the same request uri to another virtual server.

We then configure another virtual server to catch such requests with a port 443, which is the default port number for HTTPS requests. Naturally, the server names for these servers are abaweb.work and www.abaweb.work, which is the domain name we registered.

Next, we would need to process the requests captured by these virtual servers of NGINX. As such, we have a location block as shown above

This would redirect all the requests to the uwsgi application and let the uwsgi application process these requests. The uwsgi application will be listening on 0.0.0.0:8080 which will accept any request on port 8080.

The rest of the settings are from the default configuration file.

We may continue to configure the NGINX settings to optimize its performance and security, but the current configuration works fine and is able to achieve HTTPS on all routes.

## b. Availability - WSGI + Web server deployment

The settings for NGINX have already been discussed in the deployment section of the report. Now, we will discuss the uWSGI setup.

```
[uwsgi]
socket = 0.0.0.0:8080
chdir = /home/rh/RE11-T14A3/code
master = true
file = run.py
processes=1
threads=4
virtualenv = /home/rh/RE11-T14A3/code/py36-venv
```

- The socket option indicates where the uWSGI application will be listening for requests.
- The chdir option indicates where the working directory of the application is.
- The master option indicates whether to enable a master process which is responsible for pre-forking and threading.
- The file option indicates the name of the runnable file which the application is stored in
- The processes option indicates we only want one process, this is because some features of our server rely on local variables, and processes do not share the values of local variables.
- Since we use a virtual environment to run our server, we have to declare its path so that uwsgi would work correctly.

```
4 import sys
5
6 import bottle
7 import controller
8 import model
9 # -----
10
11 # It might be a good idea to move the following settings to a config file and then load them
12 # Change this to your IP address or 0.0.0.0 when actually hosting
13 host = 'localhost'
14
15 # Test port, change to the appropriate port to host
16 port = 8080
17
18 # Turn this off for production
19 debug = False
20
21 if __name__ == "__main__":
22     bottle.run(host=host, port=port, debug=debug)
23 else:
24     application = bottle.default_app()
25
```

In order to get uwsgi to run our server, we only need to make slight modification, as shown in the screenshot above, to run.py so that uwsgi will recognise the file as an application.