

# PANDEMIC INFORMATION DISSEMINATION WEB APPLICATION: A MANUAL DESIGN FOR EVERYONE

O. Nolasco-Jáuregui<sup>1\*</sup> and L.A. Quezada-Téllez<sup>2</sup>

<sup>1\*</sup>Department of Biostatistics, Tecana American University, Fort Lauderdale, FL, USA

<sup>2</sup>Department of Physics and Mathematics, Superior School at Apan, UAEH, Mexico

## ABSTRACT

*The aim of this research is to generate a web application from an inedited methodology with a series of instructions indicating the coding in a flow diagram. The primary purpose of this methodology is to aid non-profits in disseminating information regarding the COVID-19 pandemic, so that users can share vital and up-to-date information. This is a functional design, and a series of screenshots demonstrating its behaviour is presented below. This unique design arose from the necessity to create a web application for an information dissemination platform; it also addresses an audience that does not have programming knowledge. This document uses the scientific method in its writing. The authors understand that there is a similar design in the bibliography; therefore, the differences between the designs are described herein; it is very important to point out that this proposal can be taken as an alternative to the design of any web application.*

## KEYWORDS

*Infodemic, web application, Python, Flask, Social Media, COVID-19 & Analysis.*

## 1. INTRODUCTION

The term infodemic [1-3] has been used to highlight the risks of misinformation phenomena during the management of a pandemic; the infodemic itself could accelerate the contagious process by influencing and fragmenting the social response [4].

In the case of the COVID-19 epidemic, the media has had a critical impact on this new information environment [5]. The dissemination of information can strongly influence people's behavior and alter the effectiveness of measures implemented by governments to contain and mitigate the virus [6]. Thus, new models for predicting the spread of the virus are taking into account the population's behavioral response, public health interventions, and the media dynamic behind information consumption [7].

The global spread of the coronavirus has been affected by the spread of misinformation, making populations more vulnerable to the disease and countering health experts and their mitigation efforts [8].

M. Cinelli et al. documented their analysis of COVID-19 of social media and the infodemic in [9], where they studied information spreading from conventional platforms such as Twitter, Instagram, and YouTube, as well non-regulated social media such as Gab and Reddit. In this analysis, they found that the largest increase in the number of registered posts was on January 21<sup>st</sup> on Gab, January 24<sup>th</sup> on Reddit, January 30<sup>th</sup> on Twitter, January 31<sup>st</sup> on YouTube, and

February 5<sup>th</sup> on Instagram. Therefore, social media platforms seem to have specific times for content consumption; such patterns may depend on the difference in audience and interaction mechanisms across platforms. From the data analyzed by filtering content related to the corona virus, the pandemic, etc., the resultant data set consists of 1,342,103 posts and 7,465,721 comments produced by 3,734,815 users. In this work, they found that information spreading is highly questionable for Gab and Reddit, as they are environments more susceptible to spreading misinformation.

According to C. Cuello-García et al. in [10], there are more than 3.8 billion people who use social networks around the world, and the amount of information received through these platforms affects how it is perceived and how people deal with the pandemic [11]. Patients, doctors, and scientists share information related to COVID-19 on Twitter, Facebook, or other social media channels [12]. Nowadays, health professionals can communicate with each other in different parts of the world [13]. Likewise, scientists from all fields can quickly interconnect and disseminate the results of their research, thus enhancing their scientific reach through easy access to information. Social media is now a part of our lives which we use to cope with social distancing [14].

Several social network research projects are documented in [10], which studied misinformation on social networks, with a focus on detecting its sources and containing them efficiently to reduce any possible damage [15]. However, they should explore certain gaps in these investigations, including the detection of susceptible populations, socio demographic characteristics, and ideological asymmetries, to eliminate disinformation [16]; this will certainly benefit from an interdisciplinary approach. There are studies and analyses of social networks with big-data, data mining, and intelligent surveillance that can be used to detect patterns [17]. Artificial intelligence can help develop data-driven algorithms and machine learning methods for acquiring COVID-19 patient experiences [18-19]. Artificial intelligence can be used as a research tool. The increased use of artificial intelligence for patients and healthcare staff is encouraging as a secondary analysis (this questionnaire is not a formal instrument, but the staff is in direct contact with the illness and their perspective is important) of social media data [20]. This field is still in its infancy and is not without error, but it is an area worth exploring.

In [21], K. C. Yang et al. analyzed the prevalence and dissemination of links with low credibility pandemic content on two major social media platforms, characterizing the similarities, differences, dissemination patterns, and people known as influencers, etc. In comparing these two platforms, divergence was found between the prevalence of low-credibility and suspicious videos in the popular sources. In fact, the accounts and pages which exert the most influence and dominance on these platforms are few. On both platforms, there is evidence of coordinated exchanges of infodemic content. This evident manipulation points to the necessity for mitigation strategies at the societal level. It also underscores the lack of limits due to inconsistent data access policies and users' inability to manage and manipulate the information. This article found evidence of coordination between accounts spreading infodemic content on both platforms. During the first months of the pandemic, similar waves of low credibility content were seen on both platforms. The strong correlation between the low- and high-credibility content timelines reveals that these spikes were driven by public attention rather than bursts of malicious content.

In [22], M. S. Islam et al. extracted social media data from December 31<sup>st</sup>, 2019 to April 5<sup>th</sup>, 2020; analyzing it to compare and contrast data collected from other sources and finding 2,311 reports of rumors, stigma, and conspiracy in 25 languages from 87 countries. The results were related to COVID-19 and comorbidities, transmission, and mortality (24%); control measures (21%); treatments and cure (19%); cause of the contagion, including its origin (15%); violence (1%); and others (20%). Of the 2,276 reports, 1,856 statements were false (82%). The study in

[22] shows that rumors affect trust in health workers, in governments, and in health experts, even affecting people's health. The authors also show that conspiracy theories can motivate people to not get vaccinated or take antibiotics and recommends that governments and international health agencies continue to publish correct and appropriate information supported by scientific evidence about COVID-19 on their websites. Even national and international agencies, as well as fact-checking agencies, must not only identify rumors, conspiracy theories, and discredit them, but must also involve social media companies in spreading the information.

This WAF (web application FrameWork) arose from the need to disseminate correct and appropriate information which is scientific in nature and comes solely from official and reliable sources. It is a fact that WAF will receive feedback from the information that citizens provide, since it is a public service, but only those users registered and verified by a human will be able to use the posting service. For malicious content, we use tokens that trigger alarms for the WAF administrator.

This proposal provides the reader with a useful pseudocode flow diagram (see Figure 1), as well as a comprehensive explanation, so anyone with basic knowledge in software programming will be able to reproduce, code, and design this WAF. The reader should not be concerned about Python 3 scripting or HTML syntax because these are described in this document. In addition, this WAF arises from the necessity to disseminate information for the authors' recent work studying COVID-19 data [23] and [24].

This document has been written as a history and describes a step-by-step flow diagram (Figure 1) for generating the WAF. The pseudocode is written in italics. This document is organized as follows: Introduction, Methodology, and Conclusions.

## 2. METHODOLOGY

The methodology is divided into five sections in descending order according to their coding. When making changes to the coding lines, it is important to ensure that its behavior will not affect the whole design because this flow diagram has a feedback loop (see Figure 1).

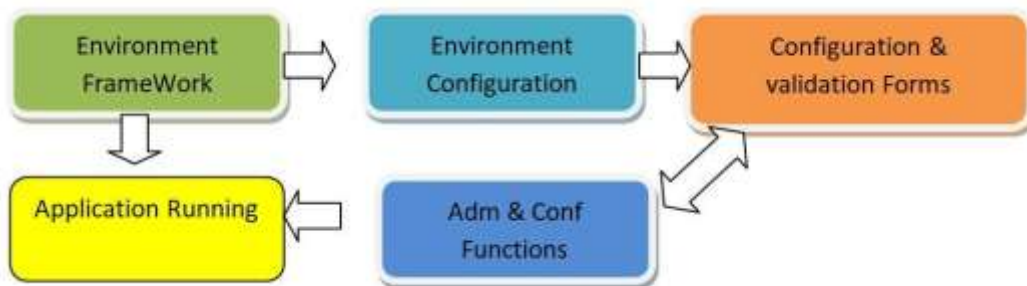


Figure 1. A flowchart of diagram represents the methodology.

Figure 1 shows the following steps: 1) programming the framework, 2) creating and modifying the environmental variables for the application, 3) configuring the application settings and form features, 4) running the application configuration file, and 5) generating a configuration file for Administrator functions.

## **A. Python Flask Framework**

Framework Software [25] from JetBrains [26] with the IDE de PyCharm 2020.3.2 for Windows [27] with ID license T35185X9L2 was selected, with a monthly cost of approximately \$20 USD. This WAF uses the Flask framework [28] and PEP 3333 (Python Web Server Gateway Interface v1.0.1) [29]. Flask is an excellent framework for developing Web applications; Flask version 1.0 was used in this work.

This WAF has a blog area, where multiple registered users can publish posts (with date and time stamps) to provide information about the pandemic. The WAF administrator can delete user posts with untrue and/or malicious content. Please consult the Terms and Conditions section of the WAF.

## **B. Environment Configuration**

This is the first step of the WAF configuration. In this case, the IDE from PyCharm [28] with Python, PHP (HTML), and Flask were selected. Therefore, when creating the project file, the Python interpreter was selected in order to recognize the .py extension files; in the project's main configuration, it is very important to select or install the list of packages from the libraries. This list of packages will be mentioned throughout this document.

The first package to be installed is PIP, which is the basis for handling Python 3. This tool allows installing and administrating Python libraries and dependencies. Once PIP is installed, Flask and its dependencies can then be installed. To verify correct installation, it is necessary to open the command prompt and run Python and import Flask; if there are no errors, it is already installed.

First, it is necessary to create the project in the root with an exclusive folder named App. Then, choose the option empty Project, select Python as the interpreter, create another folder inside this folder named static for static files such as main.css, forms.py, models.py, run.py, and routes.py, and another folder named templates for HTML files.

### **1) Template Configuration**

The main.css file is a type of Cascading Style Sheet (CSS) containing the characteristics that define the style for multiple web pages that share layout, colors, fonts, etc; this file is encoded in languages such as HTML and JavaScript — properties of World Wide Web technology. This main.css file has the coding for the multiple pages with these properties, providing more flexibility and control when presenting features that share format syntax.

The routes.py file defines the routes of the multiple web pages and their methods, which are named home, about, register, login, logout, account, create post, post\_id, post\_id/delete, user, reset\_password, and reset\_password/token.

The form.py file defines the 6 FlaskForms by class type, which are named RegistrationForm, LoginForm, UpdateAccountForm, PostForm, RequestResetForm, and ResetPasswordForm.

Inside the templates folder are the HTML files that have the .html extension as layout.html. This file describes the characteristics of the multiple web pages; it has a single location, divided into sections with different names, facilitating control and organization. When coding changes are made to the template, every change is made on all pages; this is known as template inheritance, in which layout.html is a parent page and the remaining templates are the children pages.

The extension of a template is defined by blocks. The start and end of a block is with brace and percentage characters, coded as `{% block content %}` y `{% endblock content %}`; for the extension of the layout.html file it is necessary to add the following coding: `{% extends "layout.html" %}`.

An extremely detailed tutorial describing several tips that were useful for this WAF was presented by Corey Schafer in [30]. Corey's tutorial differs in that he works with a different environment — a Mac operating system — and he has an IDE; therefore, all configurations are different from the here presented WAF.

#### **a) Bootstraps**

Boostraps are template styles that contain meta tags, JavaScripts, and pre-configured CSS files donated by Open-Source community programmers in different programming languages such as JS, CSS, or HTML. The bootstrap style can be customized, since the authors allow it, but it is very important to read the terms and conditions for their codes in [10]. It is possible to modify the containers, page divisions, or the navigation bars, change the size of the window, etc. It is even possible to bootstrap code snippets for each page extension and modify their characteristics in each section.

To change the location inside the page extensions, add the method in the HTML code as `url_for("about.html")` from the Flask library; the file name is included in parenthesis.

The about.html file contains a brief bibliography of the research group; account.html shows user accounts, displaying the user's name and photo or uploaded image.

The create\_post.html file describes the methods that only create a post if the user is active, that is, if the user is registered in the database or if the user's session is open. The home.html code from the front page shows the default form with the posts and even organizes them by date, time, and page.

The layout.html file contains the head and body code for all web page routes; this file defines the class characteristics for the containers, scroll bars, buttons, tabs, etc. The login.html file contains the form groups that will be used to acquire the user's data and verify its validity if registered using the POST method. The register.html file contains the form groups that will be used to acquire new user's data, check if it already exists in the database, and send an error if FALSE; it uses the POST method.

The reset\_request uses the content section and the group form to reset password and email validity in the database using the submit form. The user\_post.html file contains the delete\_post form; the post can only be deleted by the user who created the post.

#### **b) Emulation Configuration**

To emulate the application and run it error free, the authors chose the localhost IP 127.0.0.1, which has IPV4 and port 4555 [31]. This port is routed automatically by sending and receiving messages from the TCP/IP stack.

Application properties were verified at the following url: <http://127.0.0.1:4555/> from the web browser; every change to the application code can be loaded by tapping the F5 button in the web browser.

The code can also be run from the command prompt in conjunction with the Flask library properties, encoded as `app.run(port=4555, debug=True)`. This will trigger a debug session with a PIN number.

## 2) Flask Module Configuration

To call up the `render_template` function belonging to the Flask library, it is necessary to work with properties such as `import` and `from`.

Some commonly used Flask functions are `render_template`, `flash`, `redirect`, `request`, `url_for`, and `abort`, which are in the `routes.py` file; the `models.py` file uses the `flask_login` method, which imports the `UserMixin` property. For the initialization file, there are elemental functions such as `flask_sqlalchemy`, `flask_bcrypt`, and `flask_login`. For the configuration file, there are elemental functions for `FlaskForm` such as `flask_wtf` and `flask_wtf.file`.

## 3) Setting up Other Modules

An elementary Python module is `os`, which provides for interaction with the operating system, for example, `os.getcwd()`, which allows the programmer to locate it at the project file folder address and know if the session is active.

The `datetime` method contains the query to the operating system in real time, capturing the local date and time.

It is important to mention that you can import previously edited files, classes, or methods, modules at any time. The `routes.py` file includes modules encoded in the `models.py` file as follows: `from App.models import User, Post`; similarly, the `forms.py` file includes modules defined in `models.py` as `from App.models import User`.

## C. Configuration Forms

The reason for creating a form is to be able to receive and send parameters and interact with the user; internally, each file has a class and method created with its respective handler, objects.

Next, add the Flask libraries to configure and use the `flask_wtf` forms, encoded as `from flask_wtf import FlaskForm`.

### 1) User Forms

The user registration form (`RegistrationForm`) will be created in a class that receives the `FlaskForm` parameters. There are several fields in this form in which the data of the user to be registered will be acquired through the following methods: `username`, `email`, `password`, `confirm_password`, `accept_tos`, and `submit`. To edit these fields, some methods must be incorporated from the `wtf` forms, such as `StringField`, `PasswordField`, `SubmitField`, `BooleanField`, `validators`, and `TextAreaField`; `flask_wtf.file` is also required to import `FileField`.

There are some restrictions that must be considered for the fields to be received by the `StringField` type variables; an error must be generated if the field is found empty and sending flashing messages or danger warnings. There is also a restricted number of characters. For the `username` field, which receives data from the `StringField` function, the restricted number of characters is 25, with a minimum of 4 and a maximum of 25 characters, For the `email` field,

which also receives data from the StringField function, the restricted number of characters is 35, with a minimum of 6 and a maximum of 35 characters.

To verify the data, methods such as username, email, password, and confirm\_password are used; these methods validate the strings introduced by the user before they are saved in the database; it is necessary to use the predefined functions in the validation libraries (see section D).

LoginForm, which is similar to RegistrationForm, is necessary to create a class that receives FlaskForm characteristics as parameters. Thus, there are several fields containing data from users already registered in the database, such as email, password, remember, and submit. To edit these fields, some methods must be incorporated from the libraries, such as wtforms, StringField, PasswordField, SubmitField, and BooleanField.

There are some restrictions that must be considered for StringField type variables in LoginForm, such as email; an error must be generated if this field is left empty, and messages sent when the registered user does not match the database; it also is necessary to use validation functions in the database for the password variable named PasswordField; the remember field — a type of BooleanField — is activated when the user decides whether to leave the session by receiving TRUE or FALSE.

Likewise, the post upload form (PostForm) has several data fields for the user to register a new post, such as title, content, and submit. To edit these fields, some methods must be incorporated from the wtforms, such as StringField, SubmitField, and TextAreaField; There is only one restriction when it comes to create post: an error is generated if any field is left empty.

If the user is already registered but does not remember their username or password, a RequestResetForm registration form is created which receives the FlaskForm properties as the parameters. The RequestResetForm has two fields where the user data cannot be retrieved without the registration email.

To edit these fields, some methods must be incorporated from the wtforms, such as StringField and SubmitField; for the email field that receives the characters with StringField, the restricted number of characters is 35, with a minimum of 6 and a maximum of 35 characters. Note that an error will be generated if the field is left empty or if the email field does not match the database. When the user sends a change of password request, instructions are sent by email. Then, the form that appears on the link sent by email is the form named ResetPasswordForm, which receives the FlaskForm properties as the parameters.

There are some restrictions that must be considered for the PasswordField variable in ResetPasswordForm. The password parameter generates an error if the field is left empty and it is essential to use validation functions. For the confirm\_password variable, which is also a PasswordField type, an error must be generated if the field is left empty and messages sent when password does not match confirm\_password.

## **D. Validation Functions**

### **1) User Authentication Function**

The User Authentication function is defined in the user registration form named RegistrationForm. This RegistrationForm is a kind of class that generates a dictionary inside the forms.py file, and this function has validation and error parameters defined in the

wtforms.validators and ValidationError libraries, respectively. To code these, first enter the username and email fields for each label to receive

StringField('Username',[validators.Length(min=4,max=25)],  
StringField('EmailAddress',[validators.Length(min=6,max=35)]). For the password and confirm\_password fields, it is important to activate the validation function as PasswordField from the wtforms library. Until the validation parameters password and confirm\_password match, the following code can be used:

PasswordField('NewPassword',[validators.DataRequired(), validators.EqualTo('confirm', message='Passwords must match'))). In RegistrationForm, the accept\_tos field receives a boolean type field from BooleanField and validates that this field is not empty and is encoded as follows: TOS (Terms Of Service), BooleanField('I accept the TOS', [validators.DataRequired()]). For the submit field using the SubmitField('Sign Up') method (this function does not require a validation function activation).

Similarly, the LoginForm generates a dictionary in the file forms.py. This form needs to activate the validation and error functions. The validation functions are from the wtforms.validators library, as is the imported ValidationError. For the EmailAddress field, the receiving label is coded as follows: StringField('EmailAddress',[validators.Length(min=6,max=35)]). For the LoginForm receives the field PasswordField of wtforms and its validation parameters, are those coded as: PasswordField('Password',[validators.DataRequired(),

validators.EqualTo('confirm',message='Passwords must match'))). The remember field in LoginForm receives a boolean type field from BooleanField and validates that this field is not empty and is encoded as follows: BooleanField('Remember Me', [validators.DataRequired()]). For the field submit, its definition is coding inside of the method SubmitField('Sign Up') (and it does not require a validation function).

The form for PostForm is defined as a class that generates a dictionary in the forms.py file, and it needs validation and error functions with the parameters from the wtforms.validators library with the following coding: import ValidationError. Then, enter the field of Title and the label fields with this coding: StringField('Title', [validators.DataRequired()]). The PostForm receives the field content from wtforms library. For the validation of content parameter it is important to check that it is not empty, with the coding: ('Content', [validators.DataRequired()]). For the field submit, it has the method SubmitField('Post') (and it does not require a validation function).

The form for RequestResetForm is a class that generates a dictionary in the forms.py file that needs the validation and error functions with the parameters from the wtforms.validators library. For the email field, the receiving label is coded as follows: User.query.filter\_by(email=email.data).first(); this will ensure the existence of a user in the database that matches the email field.

The user registration form ResetPasswordForm is a class that generates a dictionary in forms.py files. It also needs the validation and error parameters from the wtforms.validators library; they are encoded as follows: first enter the username and email fields of for each receiving label as StringField('Username',[validators.Length(min=4,max=25)],

StringField('EmailAddress',[validators.Length(min=6,max=35)]). The ResetPasswordForm receive the confirm\_password and password fields by the function PasswordField of wtforms with the validation functions; those parameters are indicate that it is not empty and they must be match, shown in the following coding:



PasswordField('ConfirmPassword',[validators.DataRequired(), validators.EqualTo('confirm')]). For the field submit by the method SubmitField('Reset Password'), and it does not require a validation function.

## **E. Administrator Configurations**

This is a series of functions configured in the WAF. Here, the secret\_key has the coding of the modifications of cookies characteristics cross-site request forgery attacks, and all situations involving sensitive data. The reality is that these situations are constantly changing and hackers always find new routes.

A secret\_key with random characters can be generated in the Python terminal and imported using the secrets method followed by secrets.token\_hex(64), where 64 is the number of characters to be generated. The token acquired is assigned to configure properties such as app.config['SECRET\_KEY'].

## **F. Configuration Functions**

### **1) Route Configurations**

The route () function needs the Flask HTTP methods properties because it is an instance from URL locations [28]. The route () functions are defined in the forms described in section B and the templates from section B.1.

For the route() function of the root page it is defines the methods with the following coding: @app.route("/") and @app.route("/home"); the instance that connects the home location and its label is coded as follows: df home. And at the end of this function, coding this: return render\_template ("home.html", post=posts, page=page), where class post is the list of posts by all users, and the page number is an integer corresponding to the page containing the post list organized by date and time. This number — required for pagination — is assigned within the next function and is coding as follows: page=request.args.get('page', type=int).

To define the location of route () and about(), use the following method: @app.route("/about"); the instance that connects the location and label of the function its defined as follows: df about(): then, at the end of the function: return render\_template ("about.html", title= 'About'). The descriptive text of the WAF content is described in the about.html template.

The GET manager sends a message and the server returns the requested data, while the POST sends HTML form data to the server, which is received in the POST method if not discovered by the server.

The route() function in the register() function uses the GET and POST methods as requests handlers in the form, as follows: form=RegistrationForm(); this form uses the following method: @app.route("/register", methods=['GET', 'POST']); then, if the function register() makes a request, the following will occur: if the POST method returns a value of TRUE, then validation functions are activated. Consequently, the email data form is first requested via email and then encoded as form.email.data.rfind("@"), form.email.data.rfind(".com") in such a way that the data received in the email field is verified to be a real email. Then, the email field is compared; if the email is already exists in the database, then it is TRUE, and the following message is sent: flash('The account already exists!'); if the user is not registered, the fields are added to the dictionary as follows: db.session.add(user) and db.session.commit(). Finally, the new user is registered and redirected to login; this is the coding: return redirect('/login').

The login() function in the register handlers field uses GET and POST request methods in the form, encoded as follows: form=LoginForm(); this form uses the following method: @app.route("/login", methods=['GET', 'POST']); if the function register() has a request, the following will occur : if the POST method returns a value of TRUE, then a validation is carried out on the field, first via email, which is encoded as form.email.data.rfind("@") and form.email.data.rfind(".com") in such a way that the data received in the email field is verified as a real email. Then, the email field is compared; if the email is already exists in the database, then it is TRUE, and the following message is sent: flash('Log In', 'Successful!'); if the user is not registered, the fields are added to the dictionary as follows: flash('Login Unsuccessful! Please check email and password', 'danger'); When the user has logged in, the session must remain open, encoded as follows: login\_user(user, remember=form.remember.data) from flask\_login; finally, the new user is registered and redirected to login, encoded as follows: return redirect('/home').

The @app.route("/logout") method in the logout() function uses logout\_user() imported from flask\_login, the method responsible for inactivating the current user, and finally a return is called return redirect("/home").

In the @app.route("/account",methods=['GET','POST']) method, GET and POST are used for request methods in the form, encoded as follows: form=UpdateAccountForm(); this form uses the following method: @app.route("/register", methods=['GET', 'POST']); in this function, a new field is added — the user picture — which is refreshed each time the user begins the session and can be changed if a method for changing the user image or photograph is generated, encoded as follows: f=form.picture.data. If the user wants to change the image, the new image is saved in the dictionary, but the file to be uploaded must be a verified image type, and the previous image must be removed, using the following code: os.remove(old\_image),current\_user.image\_file=f.filename, and db.session.add(current\_user). Subsequently, the location is redirected to the template account.html with the coding render\_template("account.html",title= 'Account', image\_file=image\_file, form=form).

The route() function in the create\_post handlers field use GET and POST for request methods in the form, encoded as follows: form=PostForm(). This form uses the following method: @app.route("/create\_post",methods=['GET', 'POST']); then, the form.title.data and form.content.data are validated as not empty; if this is TRUE, the following message is sent: flash('Your post is empty!'); if FALSE, then flash('Your post has been created', 'success'); if a post is generated, the new fields title and content are added as db.session.add(post), db.session.commit(). Finally, the new user is registered and redirected to login, encoded as follows: return redirect('/home').

When a user clicks on a published post, the function is named route, so this method is called @app.route("/<int:post\_id> "); the post field receives the number of posts the argument post\_id is a dictionary function with the registered user name as an argument; then, the template is displayed with the post list and said function is defined in posting.html.

The route()function also describes the situation when the active user wants to delete a post, carried out using the following coding: @app.route("<int:post\_id>/delete", methods=['POST']) the arguments delete\_post and post\_id are necessary to identify the post to delete. Thus, it is important to verify that the current user is still active using the following coding: if post.author != current\_user, next, the function is called: db.session.delete(post) and finally: return redirect('/home').

The route()function also describes the situation when the active user wants to reset\_password, carried out using the following coding: `@app.route("/reset_password", methods=['GET','POST'])` the email field needs a validation from the `form=RequestResetForm(fields)`; if FALSE, then `flash('Only for users registered')`; if TRUE, `send_reset_email(user)` is activated and the following fields are necessary: token, sender\_email, password, and smtp\_server [34]. Finally, the library is named `email.message` import `EmailMessage` [35].

When the registered user receives password reset instructions via email and follows the instructions, then `@app.route("/reset_password/<token>/", methods=['GET', 'POST'])` with `form=ResetPasswordForm()`; it is necessary to use the user authentication field with `current_user.is_authenticated` and receive a variable token type as a parameter. If the new password field is not registered in the database, then it is replaced in the dictionary with the new password: `user.password = hashed_password` and `db.session.commit()` is called return: `redirect(url_for('login'))`; if the user tries to use a password previously registered in the dictionary then, `flash('This password must different!')`.

Note: The use of `Flask-Bcrypt()` properties is very important for password use and verification [36].

## 2) Route Configurations

To decide on the domain name, it is important to visit sites such as [www.namecheap.com](http://www.namecheap.com) to determine if the domain name is available.

For handling the domain and uploading on the web the WAF it is necessary to rent a cloud manager [37]. The most important characteristic is connection via SSH [38]. There are services that offer node servers, which also offer a large volume of storage. In this work, a node (1 CPU) was rented with 25G of storage and 1GB of RAM, which is sufficient for the here presented WAF; another parameter to select when making the payment is the region, since costs vary. That is, the WAF content can be viewed in America, Europe, Asia, etc. Note that each added feature increases the monthly rent, but the service used for this WAF is modest in storage and the cost for the domain is approximately \$20USD per year. For an additional cost, an administrator password is also available for increased performance at any time. Finally, when the payment is made, the url is named by command line and will communicate with the rented server using the SSH commands, although it is necessary to install an intermediary such as PuTTY [37].

It is crucial to make a check-list of the elements that should be running visually when the domain is activated. Make a list of the of the elements when the domain configuration is done and check each of them by command line, that is, make a DNS manager [39].

## 3) Putty Configuration

The use of PuTTY is different depending on the operating system to be used, and it is not the only option for schemes that work with SSH; thus, PuTTY was chosen for this handler [37].

PuTTY is a SSH telnet client which is used when the user needs to generate a key field and its authorization is configured within an `authorized_key` [38].

#### **4) Server Flask Application Load**

There are multiple ways to upload the flask application to the server. One is to use the PIP library and the freeze method [40], a type of encryption of all the dependencies that comprise the here presented WAF, that is, all the libraries and packages that were necessary to configure the environment of the flask application. It is also essential to create a file like req.txt with the list of dependencies so that it will load automatically when running. This file must be inside the root Project folder.

To configure the virtual environment of App (root name project folder) on the server, pythonenv and python-pip must be installed [41-42]. With the last one configuration the next commands can be able to activate the virtual environment and its properties: source venv/bin/activate next, for the virtual environment activate with this coding: pip install -r req.txt.

##### **a) Sensitive Information**

When emulating the WAF on the local server, keys were used that must be modified before loading the App project to the web. It is necessary to modify the following sensitive information: The SECRET\_KEY, SQALCHEMY\_DATA\_URI, EMAIL\_USER, and EMAIL\_PASS, which have the personal administrator data. Note: do not forget to change this information and generate it again randomly in the Command Prompt. All this information will be added to a configuration file that must be created on the server with the JSON method, as config.json, using the touch method.

#### **G. WAF Running**

Continuing with the configuration of the virtual environment on the server, to start the App project, it is necessary to use the export command, which loads the Python run.py file; the code to run the App project must include the following host number: flask run --host=0.0.0.0.

Once the App project is installed on the server, a supervisor such as gunicorn is necessary from the library. pip is used to configure the alias location of the routes, and the server\_name can also be changed; gunicorn is a Python and CSS code handler.

Next, with the gunicorn supervisor installed on the server, the autostart=true method from the App project is necessary; in the event of a crash, App project is repaired with the following code: sutorestart=true, stopasgroup=true. Also, kill processes are repaired with the following coding: killasgroup=true. The above can also be added to the lines of code in the configuration file.

With the DNS configured [39], the server service working, and storage for data control, the WAF is up and running correctly.

## 1) Templates Running

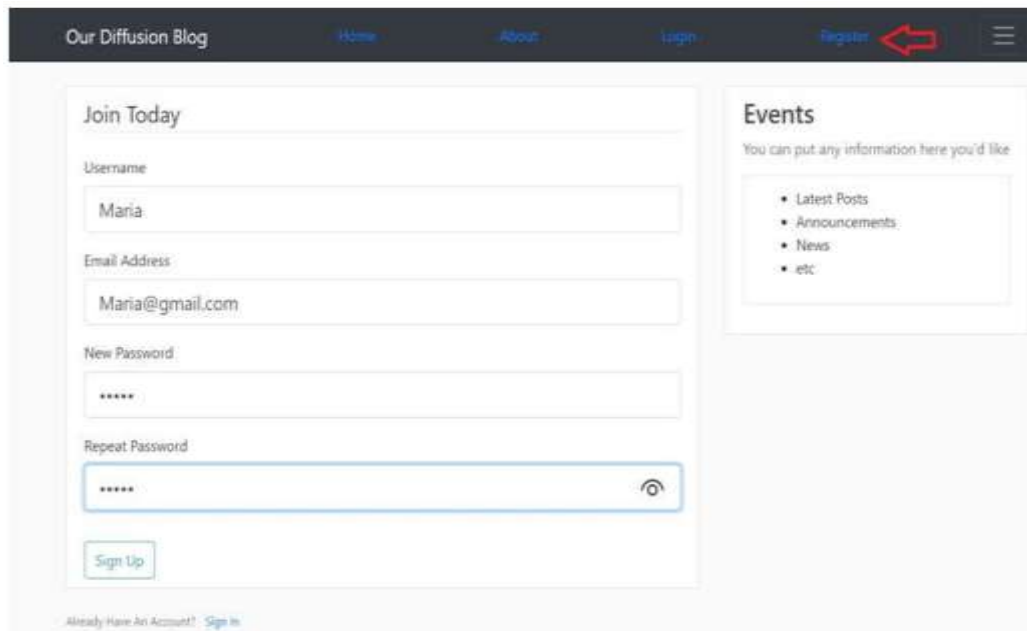


Figure 2. The red arrow indicates activation of the Register template using the POST method from the RegistrationForm().

The register.html describes the RegisterForm template (see section B). register.html defines the content-section type form-group activated by the POST method which it calls a series of border-bottom with mb-4 with the label Join Today with the properties of division general group, as form.hidden\_tag() with a fieldset of the form-control- label, form.username.label, form.email.label, form.password.label, form.confirm.password, and form.submit of the btnoutline-info. It is configured as a muted and small class with the following label: Already Have Account?; after that, this method is trigger: href=/login. Then, it is redirected to the location of the login.html template, see Figure 2.

When the user tries to register, it Sign Up event activates the redirect to the login.html template location.

login.html describes the LoginForm template (see Figure 3) and defines the content-section type form-group activated by the POST method. Consequently, it calls a series of border-bottom with mb-4 with the label Log In. Inside of this class with the division general group it has a form.hidden\_tag() with a fieldset; for the form-control- label, form.email.label, form.password.label, form.remember.label, and form.submit they have their btn-outline-info control. Also, the login.html has a muted and small class with the Need an Account? label; if this action is activated, then this method is called href=/register, which redirects to the register.html template location (see Figure 3).

When the user has registered, the Login event is activated, which directs the user to the location home.html. If the user wants to recover their password, then the href=/reset\_request function is activated (with text muted ml-2). Finally, it redirects to the reset\_request.html template location.

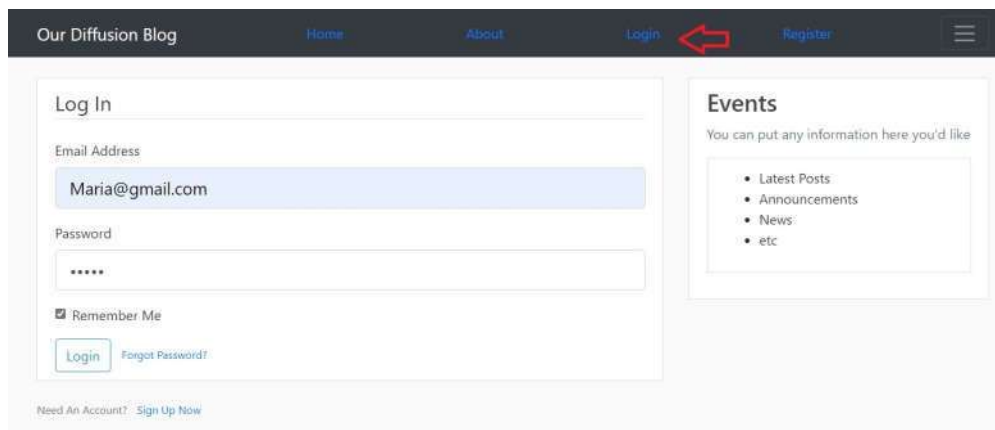


Figure 3. The red arrow indicates activation of the Login template, which uses the POST method of LoginForm().

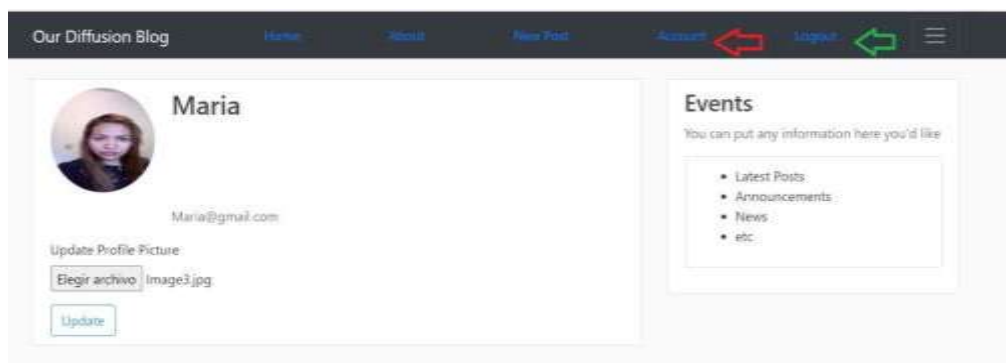


Figure 4. The red arrow indicates activation of the Account template using the POST method from the UpdateAccountForm(). The green arrow indicates a new redirect to the location logout().

Remember that each field is verified from this Login template (see section C) if the user is already registered in the database; if the users do not match, flash type messages are activated, preventively or in error (see Figure 3).

The account.html file is a content-section media which has rounded-circle properties of accountimag using src for the image\_file field. Note that a field is added for the image in each user dictionary as current\_user.image, If the user does not have a profile photo selected, a default photo is uploaded for all users. Figure 4 shows the default image for this WAF.

The image file upload process begins with the f=form.picture.data function, which validates the image file extension, only accepting .pdf, .png, .jpeg, .gif, and .pdf extensions, for example: f.file.rfind('.pdf'). Next, it is replaced and saved in the user dictionary with f.save(f.filename) and old\_image=Image.open(f.filename).

An interesting tip is to configure of the dimensions of the image to be uploaded to the profile, coded as old\_image.thumbnail(125,125); then, the previous image is eliminated with os.remove(f.filename); next, the new image is added to the dictionary with db.session.add(current\_user) and db.session.commit(). Once the new image is uploaded, it is important to return to the same account.html template; finally, the new image uploaded by the user is displayed on the profile (see Figure 5).

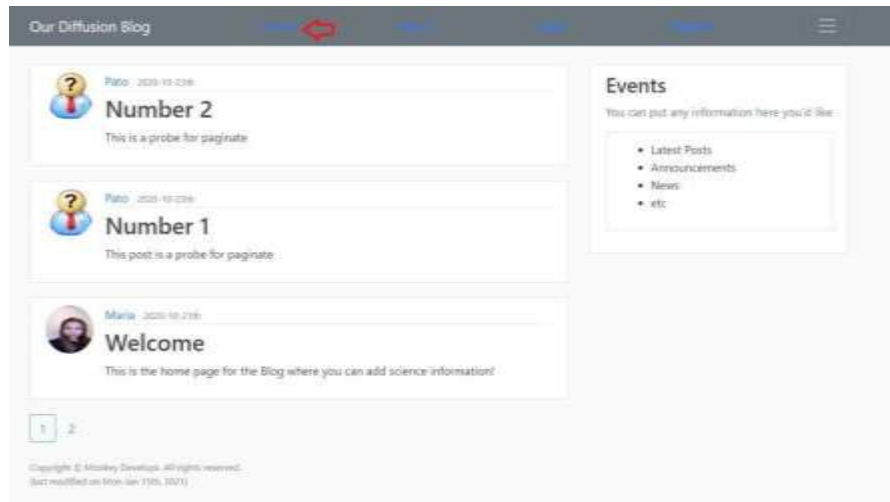


Figure 5. The red arrow indicates activation of the home.html template; the author has uploaded a welcome post.

In the home.html template, content is governed by a cycle for that controls the post.items; that is, this template has all posts from all users organized by date, starting from the most recent (see Figure 5). Figure 5 shows the last three posts from two different users organized by date. Each post has 5 fields: post.author.image, post.author.username, post.data\_posted, post.title, and post.content. Consequently, the chronological list of posts to appear in the home.html template use post.iter\_page, and page\_num.

The text in Figure 5. shows the essential information from the OurDisseminationBlog.com.mx page with the following characteristics: text-muted with small type, which depends on a footer that reports on the last modification, version, etc., as well as All Rights Reserved and Copyrights links, where the authors describe the terms and conditions of Our Dissemination Blog.

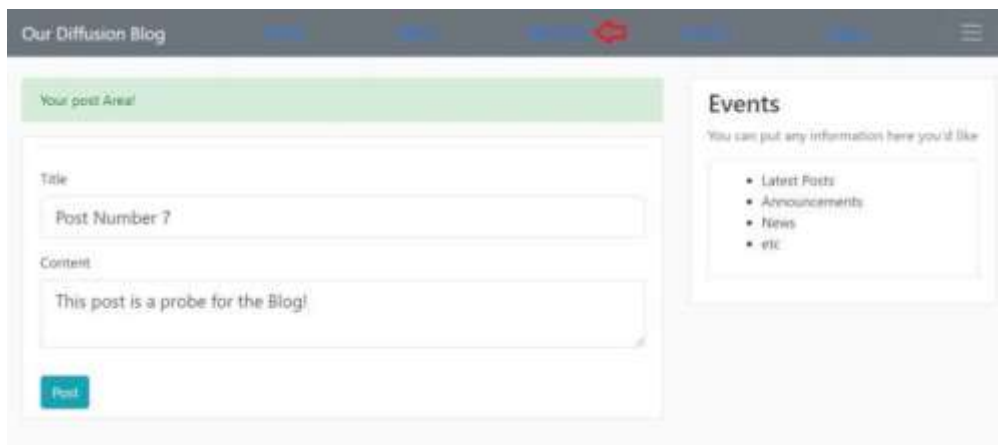


Figure 6. The red arrow indicates activation of the posting.html template, which uses the GET and POST methods from the PostForm().

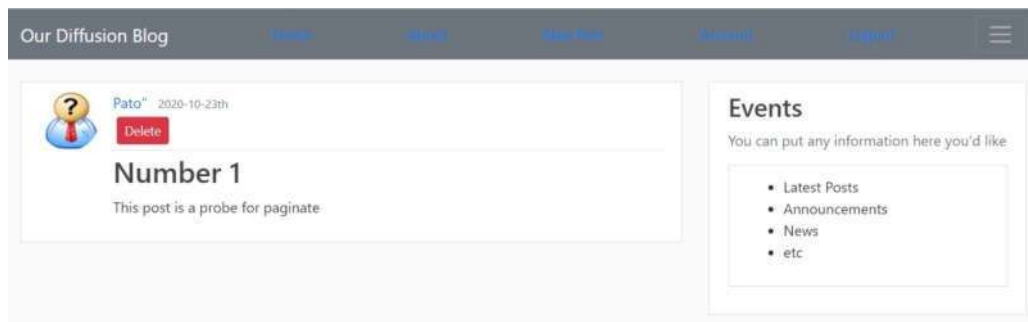


Figure 7. The delete\_post(post\_id) function needs @login\_required and it has the following route: @app.route("<init:post\_id>/delete", methods=['POST']).

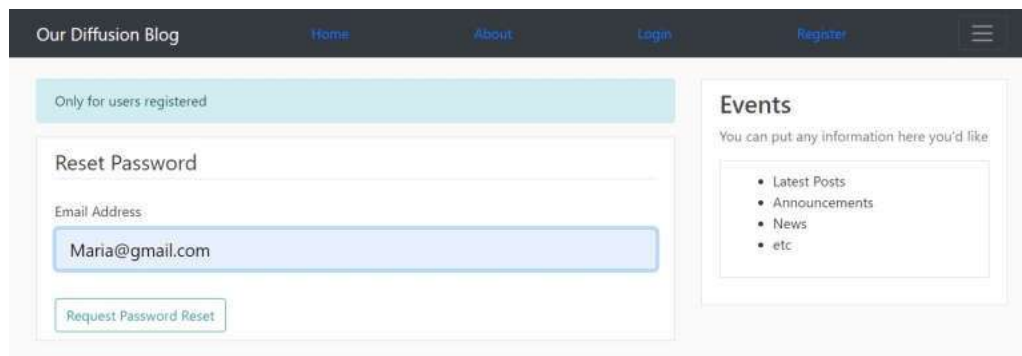


Figure 8. The reset\_request.html template is activated by login.html with RequestResetForm() from the GET and POST methods. The author of OurDisseminationBlog is requesting a password reset.

The create\_post.html file uses a media content-section which is activated using the POST method and is of the form.hidden\_tag() type, and then the form new post appears with the next coding: new\_post(), finally it redirects the location of /create\_post page, see Figure 6.

Figure 7 shows the posting.html template containing the block content, which has the media content-section properties received from post\_id to be deleted. The post\_id is essential for



deleting a post; if `post.author` is equal to `current_user`, then `deleteModal` is activated; if `FALSE`, then `delete_post` in `routes.py` is encoded as `post=Post.query.get_or_404(post_id)`.

The following instruction activates the function in charge of deleting from the database: `db.session.delete(post)`; then it is redirected to the location with `redirect(/home)`, so the user can verify that their post was deleted (see Figure 7).

Figure 8 shows the `reset_request.html` template, activated when the user requests a password reset; consequently, it is necessary to authenticate the current user with the following coding: `current_user.is_authenticated`; next, the following filter is created in the database with the email field: `User.query.filter_by(email=form.email.data)`. If all the above is `TRUE`, then `send_reset_email(email)`.

### 3. CONCLUSIONS

This application is not intended as a competitor for any application on the market, and it is not for commercial purposes. The goal of this document is to present the design proposed in Figure 1. This document is also meant to serve as a study guide for any individual who wishes to reproduce the pseudocode in this document.

The WAF was created by the authors' research group with the aim of disseminating advances in the line of research working on COVID-19 data. After fulfilling this need, it was decided to add a blog on the social platform for sharing essential information about this pandemic.

The WAF is running, as evidenced by the screenshots in Figures 2 to 8. One node (1CPU) with 25G of Storage and 1GB of RAM was rented and a domain purchased at `OurDisseminationBlog.com.mx` in Mexico—characteristics that were sufficient for this WAF.

Unfortunately, due to the economic crisis brought about by the pandemic, it was not possible to continue paying for these services, and financial support was not forthcoming. The WAF is active on a local server, and the next version is in the works. The application's functionality was verified free of charge by working at `http://127.0.0.1:4555` in the Google web browser.

### ACKNOWLEDGEMENTS

The authors are grateful to Corey Schafer's tutorial in [30], which describes several tips that were useful for their WAF.

### REFERENCES

- [1] Eysenbach, G. "How to fight an infodemic: the four pillars of infodemic management", *Journal of medical Internet research*, 22(6), e21820, 2020.
- [2] World Health Organization. "Director-General's remarks at the media briefing on 2019 novel coronavirus on 8 February 2020", 2020.
- [3] Mendoza, M., Poblete, B., & Castillo, C. "Twitter under crisis: Can we trust what we RT?". In *Proceedings of the first workshop on social media analytics* (pp. 71-79, 2010).
- [4] Starbird, K., Maddock, J., Orand, M., Achterman, P., & Mason, R. M. "Rumors, false flags, and digital vigilantes: Misinformation on twitter after the 2013 boston marathon bombing". *IConference Proceedings*, 2014.
- [5] Kim, L., Fast, S. M., & Markuzon, N. "Incorporating media data into a model of infectious disease transmission". *PloS one*, 14(2), e0197646, 2019.

- [6] Shaman, J., Karspeck, A., Yang, W., Tamerius, J., & Lipsitch, M. "Real-time influenza forecasts during the 2012–2013 season", *Nature communications*, 4(1), 1-10, 2013.
- [7] Viboud, C., & Vespignani, A. "The future of influenza forecasts", *Proceedings of the National Academy of Sciences*, 116(8), 2802-2804, 2019.
- [8] Buchanan, M. "Managing the infodemic", Doctoral dissertation, Nature Publishing Group., 2020
- [9] Cinelli, M., Quattrocchi, W., Galeazzi, A., Valensise, C. M., Brugnoli, E., Schmidt, A. L., ... & Scala, A. "The COVID-19 social media infodemic", *Scientific Reports*, 10(1), 1-10, 2020.
- [10] Cuello-Garcia, C., Pérez-Gaxiola, G., & van Amelsvoort, L. "Social media can have an impact on how we manage and investigate the COVID-19 pandemic", *Journal of clinical epidemiology*, 127, 198, 2020.
- [11] Jurkowitz, M., & Mitchell, A. "Americans who primarily get news through social media are least likely to follow COVID-19 coverage, most likely to report seeing made-up news", *Pew Research Center*, 2020.
- [12] Hitlin, P., & Olmstead, K. "The science people see on social media", *Pew Research Center*, 2019.
- [13] Smailhodzic, E., Hooijsma, W., Boonstra, A., & Langley, D. J. "Social media use in healthcare: A systematic review of effects on patients and on their relationship with healthcare professionals", *BMC health services research*, 16(1), 1-14, 2016.
- [14] Zarocostas, J. (2020). How to fight an infodemic. *The lancet*, 395 (10225), 676.
- [15] Chou, W. Y. S., Oh, A., & Klein, W. M. "Addressing health-related misinformation on social media", *Jama*, 320(23), 2417-2418, 2018.
- [16] Wang, Y., McKee, M., Torbica, A., & Stuckler, D. "Systematic literature review on the spread of health-related misinformation on social media", *Social science & medicine*, 240, 112552, 2019.
- [17] Neumann, G., & Kawaoka, Y. "Predicting the next influenza pandemics", *The Journal of infectious diseases*, 219, S14-S20, 2019.
- [18] Chitra, U., & Musco, C. "Analyzing the impact of filter bubbles on social network polarization"., In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 115-123, 2020.
- [19] Holone, H. "The filter bubble and its effect on online personal health information", *Croatian medical journal*, 57(3), 298, 2016.
- [20] Lau, A. Y., & Staccini, P. "Artificial intelligence in health: new opportunities, challenges, and practical implications2, *Yearbook of medical informatics*, 28(01), 174178, 2019.
- [21] Yang, K. C., Pierri, F., Hui, P. M., Axelrod, D., Torres-Lugo, C., Bryden, J., & Menczer, F. (2021). "The COVID-19 Infodemic: Twitter versus Facebook". *Big Data & Society*, 8(1), 20539517211013861.
- [22] Islam, M. S., Sarkar, T., Khan, S. H., Kamal, A. H. M., Hasan, S. M., Kabir, A., ... & Seale, H. "COVID-19–related infodemic and its impact on public health: A global social media analysis", *The American journal of tropical medicine and hygiene*, 103(4), 1621, 2020.
- [23] Molina-Torres, R., Nolasco-Jáuregui, O., Rodríguez-Torres, E. E., Itzá-Ortiz, B. A., & Quezada-Téllez, L. A. "A comparative analysis of urban development, economic level, and COVID-19 cases in Mexico City", *Journal of Urban Management*, 2021.
- [24] Nolasco-Jauregui, O., Quezada-Tellez, L. A., Rodríguez-Torres, E. E., & TetlalmatziMontiel, M. "COVID-19 Patients Analysis using Superheat Map and Bayesian Network to identify Comorbidities Correlations under Different Scenarios". *medRxiv*, 2021.
- [25] GoodFirms (2020). What is a Web Framework?. Retrieved January 7th 2021 from GoodFirms: <https://www.goodfirms.co/glossary/web-framework/>
- [26] JetBrains (2021).The Python IDE for Professional Developers. Retrieved January 8th 2021 from JetBrains S.R.O.: <https://www.jetbrains.com/pycharm/>
- [27] JetBrains (2021).Download PyCharm for Windows. Retrieved January 8th 2021 from JetBrains S.R.O.: <https://www.jetbrains.com/pycharm/download/#section=windows>
- [28] Armin Ronacher and Contributors (2015). The Pallets Projects (Flask). Retrieved January 9th 2021 from WSGI.org: <https://palletsprojects.com/p/flask/>
- [29] Python Software Foundation (2021) Python Web Server Gateway Interface v1.01. Retrieved January 11th 2021 from python.org: <https://www.python.org/dev/peps/pep3333>
- [30] Corey Schafer (2021). Development, Design, DIY and more [Blog]. Retrieved January 10th 2021 from CoreyMS: <https://coreyms.com/>
- [31] Bradley Mitchel (2020, October 23). 127.0.0.1 IP Address Explained. Retrieved January 11th 2021 from LifeWire: <https://www.lifewire.com/network-computer-special-ipaddress-818385>

- [32] Getbootstrap (2020). Introduction. Retrieved January 12th 2021 from The Bootstrap Authors and MIT license: <https://getbootstrap.com/docs/4.3/gettingstarted/introduction/>
- [33] PythonBasic.org (2020). Flask HTTP methods, handle GET & POST. Retrieved January 13th 2021 from Python Basic.org: <https://pythonbasics.org/flask-httpmethods/#:~:text=A%20GET%20message%20is%20send,not%20cached%20by%20the%20server>
- [34] PythonBasic.org (2020). Send an E-mail with Python Flask. Retrieved January 14th 2021 from PythonBasic.org: <https://pythonbasics.org/flask-mail/>
- [35] Python Software Foundation (2021). email: Examples. Retrieved January 16th 2021 from python.org: <https://docs.python.org/3/library/email.examples.html>
- [36] Andromeda (2020, November 28th). Flask-Bcrypt. Retrieved January 17th 2021 from Andromeda.local Debian Linux Version 10.7: <http://lira.no-ip.org:8080/doc/pythonflask-bcrypt-doc/html/>
- [37] Simon Tatham (2020, November 22). Download PuTTY. Retrieved January 18th 2021 from putty.org: <https://www.putty.org/>
- [38] Linode (2020, October 7th). Use Public Key Authentication with SSH. Retrieved January 18th 2021 from Linode: <https://www.linode.com/docs/guides/use-public-keyauthentication-with-ssh/#windows>
- [39] Linode (2020, November 24). DNS Manger. Retrieved January 18th 2021 from Linode: <https://www.linode.com/docs/guides/dns-manager/>
- [40] Hanson, C., & Walcott, K. R. (2018). "NETCDL: The Network Certification Description Language". International Journal of Computational Science, Information Technology and Control Engineering, 5(3).
- [41] Kaur, J., & Sahni, V. (2016). "Energy Efficient Linear Cluster Handling Protocol For WSN". International Journal of Computational Science, Information Technology and Control Engineering, 3(3).
- [42] Kularia, Y., Kohli, S., & Bhattacharya, P. P. (2016). "Analysis Of Acoustic Channel Characteristics For Underwater Wireless Sensor Networks". International Journal of Computational Science, Information Technology and Control Engineering, 3(1/2).