



Deggendorf Institute of Technology

Final Report Of TheProject

“COSMOS INFINITYBLOG”

(Advanced ProgrammingTechniques)

Professor

Dr. Andreas Fischer

COSMOS BLOG APPLICATION

Brief description of the software developed

The blog uses the Flask Web Framework version 1.0.3 - A Web Application Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management, etc. -

Flask is a web application framework written in Python. It is developed by **Armin Ronacher**, who leads an international group of Python enthusiasts. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine.

WSGI

Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

Werkzeug

It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

Jinja2

Jinja2 is a popular templating engine for Python. A web templating system combines a template with a certain data source to render dynamic web pages.

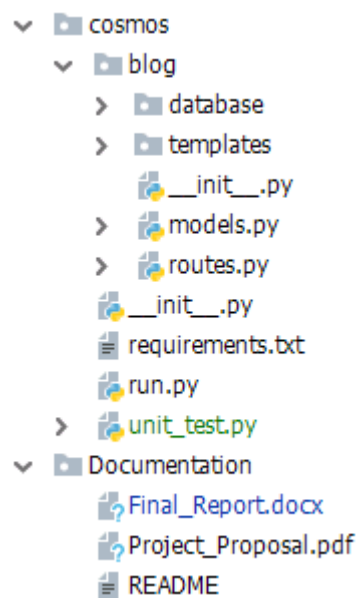
Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support. Instead, Flask supports the extensions to add such functionality to the application.

1. CODE DOCUMENTATION

The blog software contains several elements in order to work, python scripts, HTML templates, CSS and JS files for the HTML files and 3rd-party modules have been included in the repository. Into the repository the following files are allocated:

<i>run.py</i>	Script that allows running the software
<i>__init__.py</i>	Script that initialise the database and modules

<i>unit_test.py</i>	Contains methods for testing the main code functions.
<i>route.py</i>	Contains all the methods that show the html files
<i>models.py</i>	Contains the User, Article, Comment classes and the connection between the database and the application
<i>requirements.txt</i>	Contains the name and the version of the modules needed for running the software



Structure and files in the COSMOS package

For the present project, Flask is being used to:

- Connecting the App with the SQLite3 database; create, edit and delete entries into the User, Comments and Article tables.
- Rendering HTML templates and show information and manage URL routes.
- Managing the user sessions and authenticate user.

Modules and Libraries

The following modules are been used for the deployment of the blog software:

Module	Version	Description
Click	7.0	Required for running Flask

dominate	2.3.5	Required for running Flask
Flask	1.0.3	Web Framework (core of the project)
Flask-Bootstrap	3.3.7.1	Manage templates
Flask-Login	0.4.1	Manage user sessions
Flask-SQLAlchemy	2.4.0	Manage the database connections
Flask-WTF	0.14.2	For creating forms and add validators
itsdangerous	1.1.0	Required for run Flask
Jinja2	2.10.1	Render HTML files
MarkupSafe	1.1.1	Required for run Flask
SQLAlchemy	1.3.3	Engine for Flask-SQLAlchemy
visitor	0.1.3	Required for run Flask
Werkzeug	0.15.4	Required for run Flask
WTForms	2.2.1	Validators for form fields

1.1. Database management

The blog uses a SQLite3 database, with three tables: comment table, user table and article table. This three tables store the information of users, articles and comments. Additionally, the software has a directory for the image articles.

Database Accessing

The blog software uses a SQLite 3 database. For accessing the database, perform insertions and queries is needed the module *SQLAlchemy*. It is necessary configure a variable named <SQLALCHEMY_DATABASE_URI> that states the directory where the database file database.db is allocated.

```
from flask_sqlalchemy import SQLAlchemy
if platform.system()=='Linux':
    db_path = dirpath + '/database/database.db'
else:
    db_path = dirpath + '\\database\\database.db'
print('database path: ',db_path,'*****')
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///'+db_path
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
Bootstrap(app)
db = SQLAlchemy(app)
```

In Python 3 is necessary import the *SQLAlchemy* module, then depending on the type of OS (Windows or Linux), the code manipulates the path towards the database. The <db_path> variable

contains the directory where the database is allocated. Finally, This variable is used as an argument to create an instance <db> that is an instance of the *SQLAlchemy* Class.

1.1.1. User table

The <User> table allows the blog software recognize and remember the personal user data. The table stores data as name, last name, email, password and the level access number that could have the following values.

Variable <user.access>	0	1	2	3
Type of User	No authorized user	User	Author	Administrator

Table name: user		
	Name	Data type
1	id	INTEGER
2	username	VARCHAR (15)
3	email	VARCHAR (50)
4	password	VARCHAR (80)
5	access	INTEGER
6	allow	BOOLEAN
7	name	VARCHAR (15)
8	lastname	VARCHAR (15)
9	role	INTEGER

```
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(15))
    name = db.Column(db.String(15))
    lastname = db.Column(db.String(15))
    email = db.Column(db.String(50), unique=True)
    password = db.Column(db.String(80), unique=True)
    access = db.Column(db.Integer)
    role = db.Column(db.Integer)
    allow = db.Column(db.Boolean)
```

User table scheme and Class *User*

For performing an insertion into the database, the code uses the following statements.

```
hashed_password = generate_password_hash(form.password.data,method='sha256')
new_user = User(username=form.username.data,
                 email=form.email_.data,
                 password=hashed_password,
                 access = 0,
                 allow = False, name=form.name_.data,
                 lastname=form.lastname.data,
                 role = 1)
db.session.add(new_user)
db.session.commit()
```

A `<new_user>` object is created, this object is an instance of the *Class User*. The `<new_user>` is used as an argument for the method `db.session.add(<new_user>)` for creating a new entry in the *table User*. Finally, the code calls the `db.session.commit()` method for writing into the database.

For performing a query to the database, the code uses the following statements.

```
users = User.query.filter_by().all()
print(users)
```

The method `User.query.filter_by().all()` retrieves all the objects into the database to a list named `<users>`. This list can be used for showing all the users in a web page.

1.1.2. Article table

This table is used to store the data from the user posts. This table allocates variables as `<title>`, `<tags>`, `<content>`, `<timestamp>`, `<author>` and others. The variable `<id_article>` is used as index for perform queries and edit the articles, the variable `<id>` corresponds to the `<User.id>` variable, it is used for filter the articles by user.

Table name: article			class Article(db.Model):	
	Name	Data type		
1	id_article	INTEGER		<code>id_article = db.Column(db.Integer, primary_key=True)</code>
2	id	INTEGER		<code>id = db.Column(db.Integer)</code>
3	title	VARCHAR (100)		<code>title = db.Column(db.String(40))</code>
4	tags	VARCHAR (50)		<code>tags = db.Column(db.String(20))</code>
5	article	VARCHAR (6000)		<code>article = db.Column(db.String(600))</code>
6	url_image	VARCHAR (300)		<code>url_image = db.Column(db.String(300))</code>
7	timestamp	DATETIME		<code>data = db.Column(db.LargeBinary)</code>
8	allow	BOOLEAN		<code>timestamp = db.Column(db.DateTime)</code>
9	data	BLOB		<code>allow = db.Column(db.Boolean)</code>
10	author	VARCHAR (30)		<code>author = db.Column(db.String(30))</code>

Article table scheme and Class Article

For performing an insertion into the articles table, the following statements are being used.

```
now = datetime.now()
url_image = ''
new_article = Article(id=current_user.id,
                      title=form.title.data,
                      tags=form.tags.data,
                      article=form.article.data,
                      url_image = url_image,
                      timestamp = now,
                      allow = False,
                      author = str(current_user.name)+' '+str(current_user.lastname))
db.session.add(new_article)
db.session.commit()
```

First, the code request the actual date by using the method `datetime.now()`. The code creates an object `<new_article>`, it is an instance of the Class `Article`. The `<new_article>` object has as arguments the new article data.

For performing a query to the database, the code uses the following statements.

```
article = Article.query.filter_by(id_article=id_article).first()
```

The method `Article.query.filter_by().first()` retrieves the first element of a query. The variable `<id_article>` is used to filter the article by the article index.

1.1.3. Comment table

The <Comment> table scheme is showed below; in the main code uses the Class named Comment that contains the <id_article> variable that is used as an index for retrieving the comments according a specific article. The <id_comment> variable is as index for the elements in the comment table.

Table name: comment			<pre> class Comment(db.Model): id_comment = db.Column(db.Integer, primary_key=True) id_article = db.Column(db.Integer) id = db.Column(db.Integer) name = db.Column(db.String(15)) lastname = db.Column(db.String(15)) comment = db.Column(db.String(300)) timestamp = db.Column(db.DateTime) allow = db.Column(db.Boolean) </pre>
	Name	Data type	
1	id_comment	INTEGER	
2	id	INTEGER	
3	name	VARCHAR (15)	
4	lastname	VARCHAR (15)	
5	comment	VARCHAR (300)	
6	timestamp	DATETIME	
7	allow	BOOLEAN	
8	id_article	INTEGER	

Comment table and Class Comment

For inserting a new entry into the comment table, the following statements are used.

```

new_comment = Comment(id=0,
    id_article=id_article,
    name=form.name__.data,
    lastname=form.lastname.data,
    comment=form.comment__.data,
    timestamp=now,
    allow=False)
db.session.add(new_comment)
db.session.commit()

```

The code uses as parameters the data provided when the user performs a post in the comment form. if the user has an active account the variable <allow> is set to True.

For performing a query from the comment table, the following statements are used.

```

comments = Comment.query.filter_by(id_article=id_article).all()

```

1.2. ./images/uploads Directory

Additional, for saving the images that are used in the articles, there is a directory that is created when the system starts. The following statements are used to check and create the directory.

```
if not os.path.exists('templates/blog/assets/images/uploads'):
    os.makedirs('templates/blog/assets/images/uploads')
    print('Uploads directory created')
else:
    print('Uploads directory already exists')
```

The following statements are used to catch the image data post sent when the user upload an new image in the selection page.

```
if request.method == 'POST':
    f = request.files['inputFile']
    UPLOAD_FOLDER = '../blog/templates/blog/assets/images/uploads'
    f.save(os.path.join(UPLOAD_FOLDER, 'img'+str(id_article)+'.jpg'))
    #print('sentro',id_article)
    return redirect(url_for('detail',id=id_article))
```

1.3. Rendering a template and sending variables

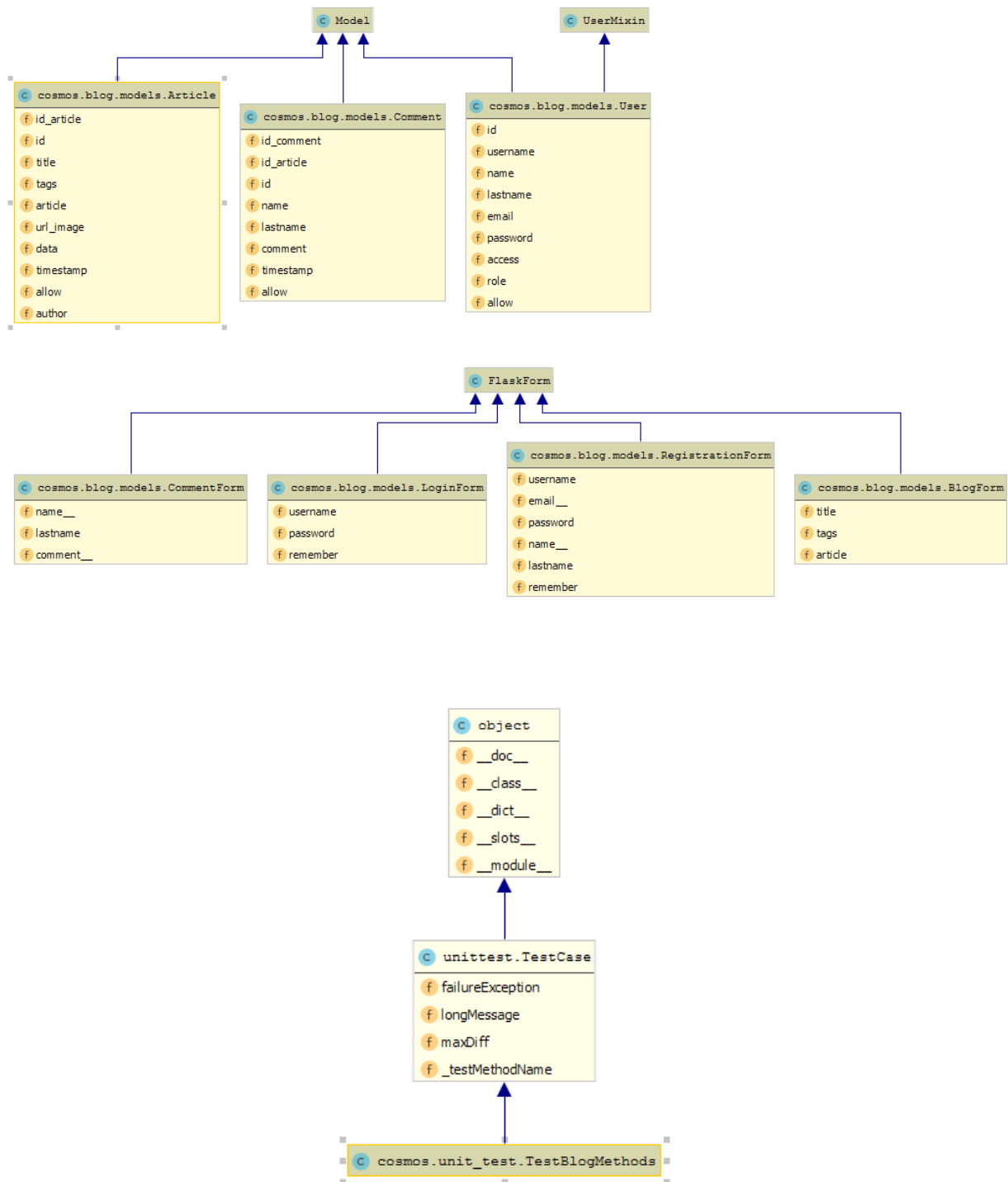
The blog uses the method *render_template()* for showing an HTML web page when a requests is performed. The following statements send an HTML response when the user ask for the '/contact' page.

```
@app.route('/contact')
def contact():
    head = check_session()
    return render_template('contact.html',head=head)
```

The <head> variable is sent as an argument that is used in the contact.html file for show different text according to the session status.

```
<ul class="navbar-nav ml-auto">
  <a href="/{{head.url1}}" class="nav-link">{{head.text1}}</a>
  <a href="/{{head.url2}}" class="nav-link">{{head.text2}}</a>
</ul>
```

UML DIAGRAMS



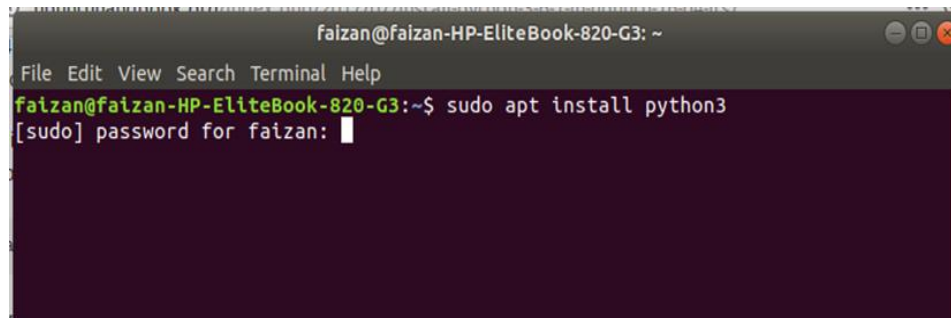
2. INSTALLATIONS AND DEPLOYMENT

2.1. Installation on Linux

To run COSMOS blog application in Ubuntu OS, first, you need to install python 3.6 or higher. You need to be logged in as a user with sudo access to be able to install packages on your system. Follow the following instructions to Install Python 3 on Ubuntu 18.04: By using the standard apt tool from the [deadsnakes](#) PPA:

1. Right click on your desktop and select terminal or Open terminal via Ctrl+Alt+T or searching for "Terminal" from app launcher.
2. Start by updating the packages list and installing the prerequisites:

```
sudo apt update  
sudo apt install software-properties-common
```

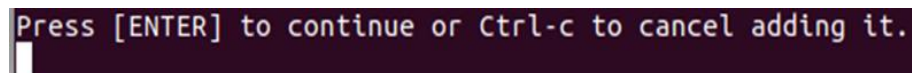
A terminal window titled 'faizan@faizan-HP-EliteBook-820-G3: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'faizan@faizan-HP-EliteBook-820-G3:~\$' and the command 'sudo apt install python3' has been entered. The next line shows '[sudo] password for faizan:' followed by a cursor.

```
faizan@faizan-HP-EliteBook-820-G3:~$ sudo apt install python3  
[sudo] password for faizan:
```

3. Next, add the deadsnakes PPA to your sources list:

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

After pressing enter the following screen will appear and you press enter once again.

A terminal window showing the prompt 'Press [ENTER] to continue or Ctrl-c to cancel adding it.' followed by a cursor.

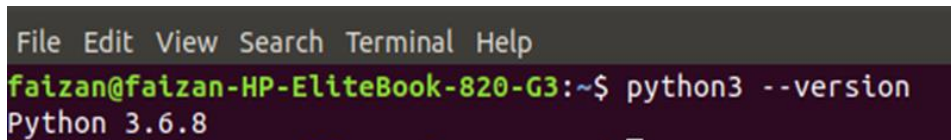
```
Press [ENTER] to continue or Ctrl-c to cancel adding it.
```

4. Next, you install python 3:

```
sudo apt install python3
```

5. You can verify python version by typing:

```
python3.7 --version
```

A terminal window with the prompt 'faizan@faizan-HP-EliteBook-820-G3:~\$' and the command 'python3 --version'. The output is 'Python 3.6.8'.

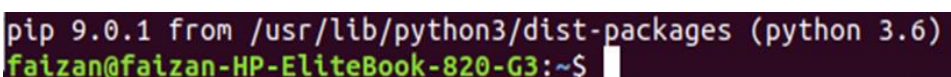
```
faizan@faizan-HP-EliteBook-820-G3:~$ python3 --version  
Python 3.6.8
```

6. Next you need to install pip:

```
sudo apt install python3-pip
```

7. You can verify pip version by typing:

```
pip3 --version
```

A terminal window showing the output 'pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)' followed by the prompt 'faizan@faizan-HP-EliteBook-820-G3:~\$' and a cursor.

```
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)  
faizan@faizan-HP-EliteBook-820-G3:~$
```

8. To install the particular requirements for COSMOS app. Locate the requirements.txt file in cosmos folder and then you need to change your directory to this location from terminal by using command that looks like this:

```
cd ../blog-project/cosmos
```

```
faizan@faizan-HP-EliteBook-820-G3:~$ cd /home/faizan/Desktop/blog-project/cosmos
faizan@faizan-HP-EliteBook-820-G3:~/Desktop/blog-project/cosmos$
```

9. Next you need to install required modules in requirements file:

```
pip3 install -r requirements.txt
```

```
Successfully installed Click-7.0 Flask-1.0.3 Flask-Bootstrap-3.3.7.1 Flask-Login-0.4.1 Flask-SQLAlchemy-2.4.0 Flask-WTF-0.14.2 Jinja2-2.10.1 MarkupSafe-1.1.1 SQLAlchemy-1.3.3 WTForms-2.2.1 Werkzeug-0.15.4 dominate-2.3.5 itsdangerous-1.1.0 visitor-0.1.3
```

10. Now it's time to deploy the server:

```
Py run.py runserver
```

```
Running on http://127.0.0.1:8000/
```

your server will be running on IP 127.0.0.1 (localhost) and port 8000.

Then you can go type the above address in the address bar of your favorite browser and you will find the website running. However you can change the IP and port from run.py file according to your requirement.

Note: To uninstall the Modules use this command

```
pip uninstall -r requirements.txt -y
```

2.2. Installation on Windows

1. To run blog application in Windows OS you need to install python 3 first. You can download python 3 by visiting your favourite web browser. Open a browser window and navigate to the [Download page for Windows](https://www.python.org/downloads/windows/) at [python.org](https://www.python.org).

2. Underneath the heading at the top that says **Python Releases for Windows**, click on the link for the **Latest Python 3 Release - Python 3.x.x**. (As of this writing, the latest is Python 3.7.3).

3. Scroll to the bottom and select either **Windows x86-64 executable installer** for 64-bit or **Windows x86 executable installer** for 32-bit.

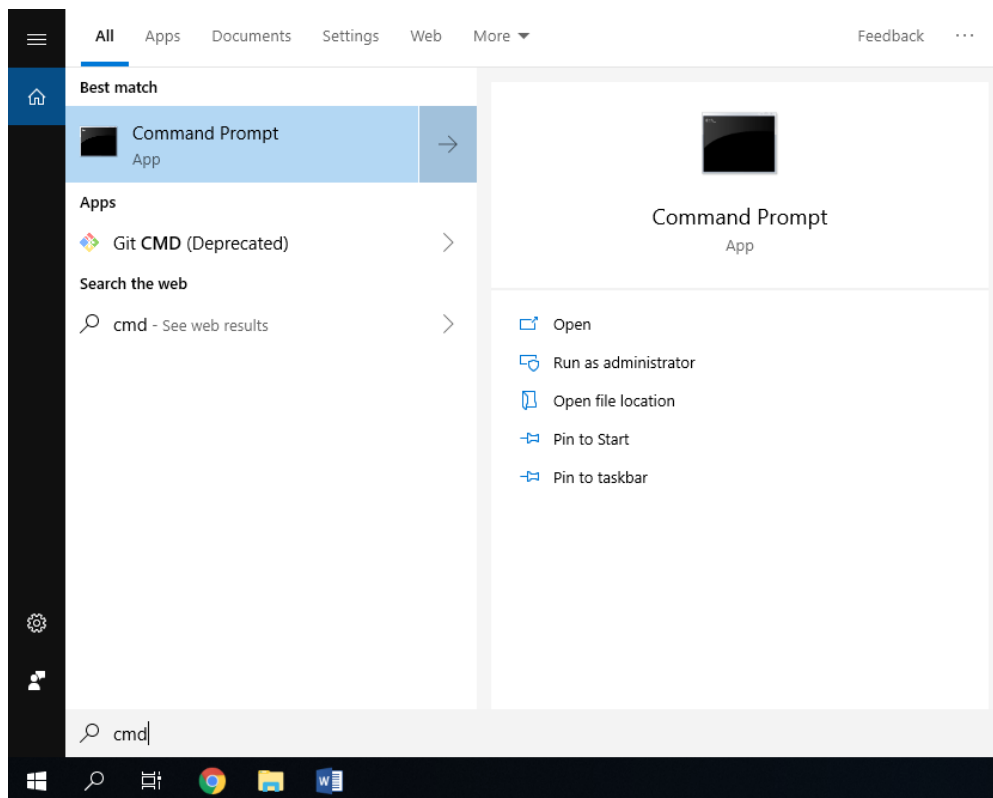
Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file. A dialog should appear that looks something like this:



Important: You want to be sure to check the box that says **Add Python 3.x to PATH** as shown to ensure that the interpreter will be placed in your execution path.

Then just click **Install Now**. A few minutes later you should have a working Python 3 installation on your system.

4. Next, on desktop search for command prompt and click to open it



5. Check your Python version by this command:

`py --version`

```
C:\Users\FAIZAN MAHMOOD>py --version
Python 3.7.3
```

6. Now upgrade pip to the latest version by typing:

`python -m pip install --upgrade pip`

```
C:\Users\FAIZAN MAHMOOD>pip --version
pip 19.1.1 from c:\users\faizan mahmood\
```

7. Now change the directory to the directory with requirements.txt file using the command that would look something like this

`cd...\blog-project\cosmos`

8. Now install the requirements:

`pip install -r requirements.txt.`

9. After the prerequisites installed it's time to run the server and u can do it by using this command :

```
py run.py runserver
```

and your server will be running on IP 127.0.0.1 (localhost) and port 8000.

```
Running on http://127.0.0.1:8000/
```

Then you can go type the above address in the address bar of your favorite browser and you will find the website running. However you can change the IP and port from run.py file according to your requirement.

Note: To uninstall the Modules use this command

```
pip freeze > requirements.txt
```

3. TEMPLATING

3.1. Jinja 2 Templates

Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment:

```
<title>{% block title %}{% endblock %}</title><ul>
{% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}</ul>
```

Features:

- sandboxed execution
- powerful automatic HTML escaping system for XSS prevention
- template inheritance
- compiles down to the optimal python code just in time
- optional ahead-of-time template compilation
- easy to debug. Line numbers of exceptions directly point to the correct line in the template.
- configurable syntax

3.2. Definitive HTML files

Detail.html:

```
<div class="blog-left">
  <h1>{{article.title}}</h1>
  <div class="posted-on">
    <p>by <span>{{user.name}} {{user.lastname}}</span> on
  <span>{{article.timestamp.strftime("%m/%d/%Y %H:%M:%S")}}</span></p>
  </div>
  <div class="detail-img">
    
  </div>
  <div class="blog-desc">
    <p>{{article.article}}</p>
  </div>
</div>
```

We are getting the article title by using article.title and then we are getting first name and last name of user who uploaded the article and then next we are getting the time ,when this article uploaded.

```
<div class="pt-5">
<h3 class="mb-5">{{number}} Comments</h3>
```

Getting the number of comments on the article.

```
<ul class="comment-list">
  {% for comment in comments %}
  {% if comment.allow: %}
    <li class="comment">
      <div class="vcard">
        
      </div>
      <div class="comment-body">
        <h3>{{comment.name}} {{comment.lastname}}</h3>
        <div class="meta">{{comment.timestamp}}</div>
        <p>{{comment.comment}}</p>
        <!--p><a href="#" class="reply rounded">Reply</a></p-->
      </div>
    </li>
  {% endif %}
  {% endfor %}
</ul>
```

Here we are checking if user is allow to comment if condition is right then next he will put his name ,last name and comment he want to write in the article.

```
<div class="comment-form-wrap pt-5">
  <h3 class="mb-5">Leave a comment {{data.name}} {{data.lastname}}</h3>
```

Getting the first and last name of user who comment.

```
<form class="form-signin" method="POST" action="#">
  <div>{{ form.hidden_tag() }}</div>
  {% if logged: %}
    <div>{{ wtf.form_field(form.name) }}</div>
    <div>{{ wtf.form_field(form.lastname) }}</div>
```

```
{% endif %}
<div>{{ wtf.form_field(form.comment__) }}</div>

<button class="btn btn-lg btn-primary btn-block" type="submit">Comment</button>
</form>
```

Above we are setting up our comment form, how its look to our user.

```
<ul>
  <li><a href="/bloglist?tag=older">Oldest articles first</a></li>
  {% for tag in tags%}
    <li><a href="/bloglist?tag={{tag}}">{{tag}}</a></li>
  {% endfor %}
</ul>
```

Getting the tags of article.

SignUp.html

```
<section class="cmspage mtb-40">
  <div class="container">
    <!--ADD CONTENT HERE-->
    {{message}}

    <form class="form-signin" method="POST" action="/signup">
      <h2 class="form-signin-heading">Please Sign Up</h2>
      <div>{{ form.hidden_tag() }}</div>
      <div>{{ wtf.form_field(form.username) }}</div>
      <div>{{ wtf.form_field(form.name__) }}</div>
      <div>{{ wtf.form_field(form.lastname) }}</div>
      <div>{{ wtf.form_field(form.email__) }}</div>
      <div>{{ wtf.form_field(form.password) }}</div>
      <button class="btn btn-lg btn-primary btn-block" type="submit">Sign Up</button>
    </form>
  </div>
</section>
```

Setting up our form for signup, user need to put username, first name , last name, email and password in order to sign up. All these fields are required and must be enter by the user, otherwise user can't sign up.

Login.html

```
{% include 'header.html' %}

{% include 'navbar.html' %}

<body>
  <section class="cmspage mtb-40">
    <div class="container">
      <!--ADD CONTENT HERE-->
      <div>{{message}}</div>
      <form class="form-signin" method="POST" action="/login">
        <h2 class="form-signin-heading">Please sign in</h2>
        {{ form.hidden_tag() }}
        {{ wtf.form_field(form.username) }}
```



```
        {{ wtf.form_field(form.password) }}
        {{ wtf.form_field(form.remember) }}
        <button class="btn btn-lg btn-primary btn-block" type="submit">Sign in</button>
    </form>
</div>
</section>
```

Include header ,navbar and footer from our base file and getting the message and user need to put username ,password in order to login , all these fields are required and must be enter by the user, otherwise contact form will not work and user can also choose remember me if he want to save his password for the next time

Header.html

```
{% if title %}
    <title>Cosmos - {{ title }}</title>
{% else %}
    <title>Cosmos</title>
{% endif %}
```

Setting the header and implementing the condition, if title exist to the page then it will show the title of that page else title will be cosmos

Footer.Html

```
<div class="container">
    <div class="copyright">
        <div>© Copyright 2019, All Rights Reserved</div>
    </div>
</div>
```

Created the footer of website and use simple text inside to div

Navbar.html

```
<nav class="navbar navbar-expand-md navbar-dark bg-blue">
    <div class="container">
        <a class="navbar-brand" href="/index">
            Cosmos Infinity
        </a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarCollapse"
            aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarCollapse">
            <ul class="navbar-nav mr-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="/index">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/about">About</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/contact">Contact</a>
                </li>
            </ul>

            <ul class="navbar-nav ml-auto">
                <a href="{{head.url1}}" class="nav-link">{{head.text1}}</a>
                <a href="{{head.url2}}" class="nav-link">{{head.text2}}</a>
            </ul>
        </div>
    </div>
</nav>
```

```

    </ul>
  </div>
</div>
</nav>

```

Created the navbar by using bootstrap classes and also setting up the link to each menu in navbar. And if head.url1 then it will open the sign up page for url2 it will open login page.

Edit.html

```

<form action="/api/edit_post?id={{article.id_article}}" method="post">
  <div class="form-group">
    <label for="title">Article title</label>
    <input type="text" id="title" name="title" value="{{article.title}}" class="form-control">
  </div>
  <div class="form-group">
    <label for="tags">Tags:</label>
    <input type="text" id="tags" name="tags" value="{{article.tags}}" class="form-control">
  </div>

  <div class="form-group">
    <label for="article">Content:</label>
    <textarea id="article" name="article" class="form-control">{{article.article}}</textarea>
  </div>
  <div class="detail-img">
    
  </div>
  <div>
    <button class="btn btn-lg btn-primary btn-block" type="submit" name="edit" value="Edit">Edit
    Picture</button>
  </div>
  <div>
    <button class="btn btn-lg btn-primary btn-block" type="submit" name="edit" value="False">Keep the
    current image and Save</button>
  </div>
</form>

```

User can edit the article .user can edit the title, content and can edit picture of article or let the picture as it is its all depend on user.

Dashboard.html

```

<div class="dash-left">
  <ul>
    <li class="active"><a href="/dashboard" class="active">Dashboard</a></li>
    <li><a href="/admin-blog-list">Blogs</a></li>
    <li><a href="/change_role">Change your role</a></li>
    <li><a href="/logout">Logout</a></li>
  </ul>
</div>
</div>
<div class="col-md-9">
  <div class="dash-right">
    <h1>Dashboard</h1>
  <p><h1>Welcome {{data.name}} {{data.lastname}}</h1>
  </div>
</div>

```

User can see the dashboard with complete name written on it. Welcome {{data.name}} {{data.lastname}} this code is working to get the user name and lastname.

Contact Us

```
<h1>Contact</h1>
<form>
  <div class="form-group">
    <input type="text" name="name" class="form-control" placeholder="Name">
  </div>
  <div class="form-group">
    <input type="text" name="email" class="form-control" placeholder="E-Mail">
  </div>
  <div class="form-group">
    <input type="text" name="phone" class="form-control" placeholder="Phone">
  </div>
  <div class="form-group">
    <textarea name="message" rows="5" class="form-control" placeholder="Message"></textarea>
  </div>
  <div class="form-group">
    <button class="btn btn-success">Send Message</button>
  </div>
</form>
```

If any user want to contact us, user must put his name, email, phone and message that user want to send us, all these fields are required and must be enter by the user, otherwise contact form will not work

Comment.html

```
{% for comment in comments %}
  <tr>
    <td>{{comment.name}} {{comment.lastname}}</td>
    <td onclick="window.location='/detail?id={{comment.id_article}}'" class="btn btn-info btn-sm">Click
here to read</td>
    <td>{{comment.comment}}</td>
    <td>
      <a href="/api/accept_comment?id={{comment.id_comment}}'" class="btn btn-info btn-
sm">Accept</a>
    </td>
    <td>
      <a href="/api/delete_comment?id={{comment.id_comment}}'" class="btn btn-danger btn-
sm">Reject</a>
    </td>
  </tr>
{%endfor%}
```

Admin can see the user comment who comment on the post. Admin can see user name, last name. And admin can accept and reject the user comment as well.

Changrole.html

```
<section class="cmspage mtb-40">
  <div class="container">
    <h1>{{name}} Please select your desired role</h1>
    <div class="page-desc">
      {% if level == 0: %}
      <p>Your Actual level access is: No User Yet</p>
```

```
{% elif level == 1: %}
    <p>Your Actual level access is: User</p>
{% elif level == 2: %}
    <p>Your Actual level access is: Author</p>
{% elif level == 3: %}
    <p>Your Actual level access is: Administrator</p>
{% endif%}
</div>
<div class="blog-form">
    <form class="form-signin" method="POST" action="/change_role" enctype="multipart/form-data">
        <select name="choice" id="choice">
            <option value="{{level}}" selected>Options</option>
            <option value="1">User</option>
            <option value="2">Author</option>
            <option value="3">Administrator</option>
        </select>
        <button class="btn btn-lg btn-primary btn-block" type="submit">Request</button>
    </form>
</div>
</div>
```

</section>

Admin can change any user role later on, he can make him author administer. In the above code we are changing the roles of users.

Bloglist.html

```
<section class="blog-list mtb-40">
    <div class="container">
        <h1>Blogs {{order_tag}}</h1>
        <div class="row">
            {% for article in articles.items%}
                <div class="col-md-4">
                    <div class="blog-box">
                        
                        <h3>{{article.title}}</h3>
                        <p>by <span>{{article.author}}</span> on <span>{{article.timestamp.strftime("%m/%d/%Y
%H:%M:%S")}}</span></p>
                        <p>{{article.article[:article.article.find('.')+1]}}</p>
                        <a href="detail?id={{article.id_article}}" class="btn btn-danger">Read more...</a>
                    </div>
                </div>
            {%endfor %}
        </div>
    </div>
</section>
```

In the above code first we are getting the list of articles using for loop. And then next we are getting the each article title, author to that article and time when the article uploaded.

```
{% for page_num in articles.iter_pages(left_edge=1,right_edge=1,left_current=1,right_current = 1) %}
    {% if page_num %}
        {% if articles.page == page_num %}
            <a class="btn btn-info mb-4" href="{{ url_for('bloglist', page = page_num,tag=order_tag)
}}">{{page_num}}</a>
        {% else %}
            <a class="btn btn-outline-info mb-4" href="{{ url_for('bloglist', page = page_num,tag=order_tag)
```

```

}}">{{page_num}}</a>
    {% endif %}
{% else %}
    ...
    {% endif %}
{% endfor %}

```

After getting the specific numbers of articles, the next article will display on the next page.

Newuser.htmls

```

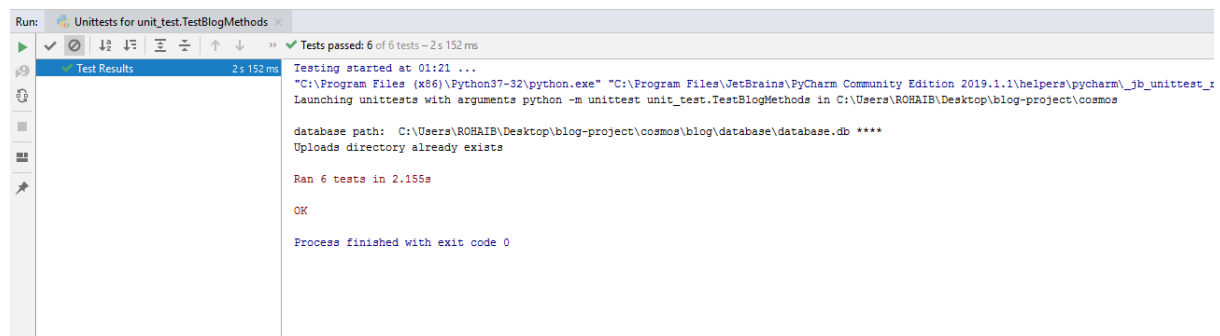
<tr>
<td>{{user.name}} {{user.lastname}}</td>
    <td>{{user.email}}</td>
    {% if user.role == user.access: %}
        <td>N/A</td>
    {% else: %}
        {% if user.role == 1: %}
            <td bgcolor="#FF0000">User</td>
        {% elif user.role == 2: %}
            <td bgcolor="#FF0000">Author</td>
        {% elif user.role == 3: %}
            <td bgcolor="#FF0000">Admin</td>
        {% endif%}
    {% endif%}
    <td>
        {% if user.access == 0:%}
            Current State
        {%else%}
            <a href="/api/non-user?id={{user.id}}" class="btn btn-info btn-sm">OK</a>
        {%endif%}
    </td>
    <td>
        {% if user.access == 1:%}
            Current State
        {%else%}
            <a href="/api/user?id={{user.id}}" class="btn btn-info btn-sm">OK</a>
        {%endif%}
    </td>
    <td>
        {% if user.access == 2:%}
            Current State
        {%else%}
            <a href="/api/author?id={{user.id}}" class="btn btn-info btn-sm">OK</a>
        {%endif%}
    </td>
    <td>
        {% if user.access == 3:%}
            Current State
        {%else%}
            <a href="/api/admin?id={{user.id}}" class="btn btn-info btn-sm">OK</a>
        {%endif%}
    </td>
</tr>

```

Setting up the new user and assigning the role of user if user.role == 1 then he will be User, user.role == 1 then he will be author, user.role == 1 then he will be Admin.

Testing

URL, views, contact, main page and welcome to main page are testing using testing functionality availability in the flask. Test result are shown in figure



4. USAGE DOCUMENTATION

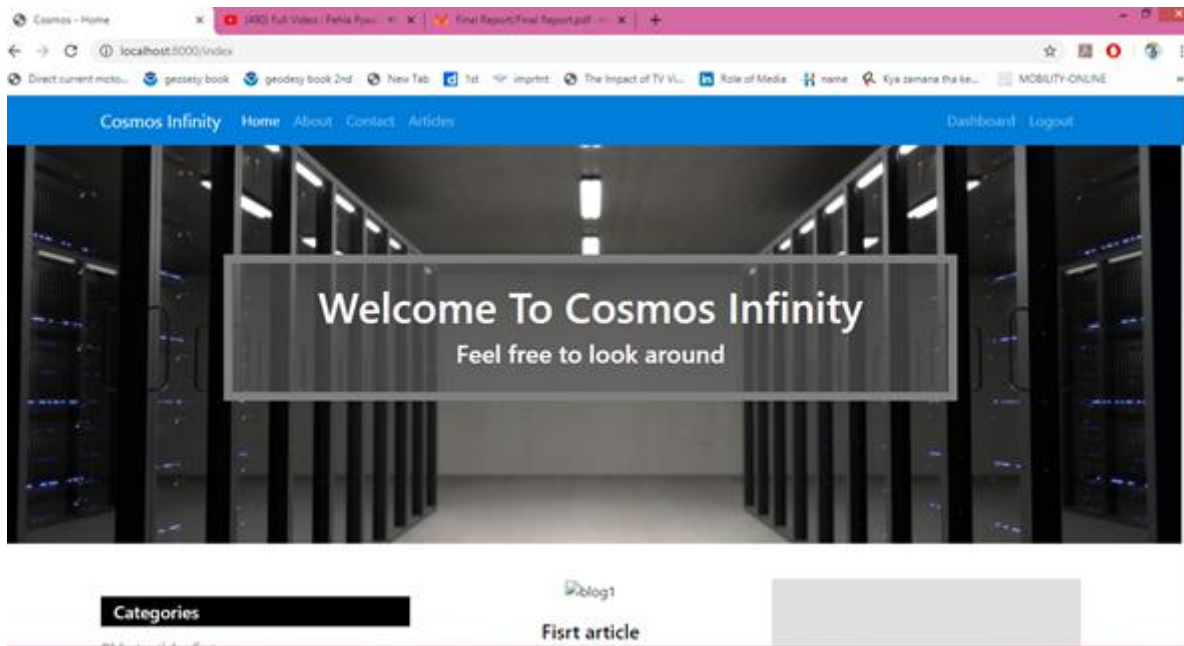
In this documents I will tell that how our blog is working and function of our blog. First of all you have to open all files in the py charm and after this it is open in the command prompt . after this you have to write it cd location of the files then after this py run.py runserver enter then it is running

```
Desktop\blog-project\cosmos
t\cosmos>py run.py runserver
```

4.1. User Manual

Home Page:

After this open the blog as a localhost:8000 so the main page is open in front of us. First of all we have open the main page .clearly there is written at the top of that in which page we are in. when we open the blog we are on the main page of blog so clearly written at the top that we are in the home page .blog consist of several pages one of which is the main page .all the other pages are easily reachable via hyperlink from the main page



Navigation Bar:

A navigation bar to access our home page, Index, About, Contact and Article. When we open another link then there is a change and clearly written that we are on which page of blog. When we click on the about then at the navigation bar, it is change from home to about. Every page is changing just like this. on the home page, user can access latest post



About us

Here you can know about the Blog Website and also know that which type of blog it is.

About

Created by the force of Advanced Programming Techniques.

Linux is user friendly. It's just very particular about who its friends are. ;)

A night before exam is more productive than the whole semester.

Simplicity is the ultimate sophistication

Error 404

If you enter the wrong address which is not in the blog then blog will show 404 error on the page, it shows that you entered the wrong URL. Please enter another address or click on home button to redirect to home page.

404

Sorry, the page you were looking for in this blog does not exist.

[Home](#)

Signup

First person which signup the blog is the admin of our blog. First every user has to be signup in the blog to become admin, authors or registered users. The blog system distinguish between four types of users: Admin, author, registered user and anonymous user.

Please Sign Up

Username *

Please fill out this field.

Name *

Lastname *

Email *

Password *

Sign Up

To sign up fill the details in the user name. User name must be at least 1 character and email must be valid.

Login Page:

Login page where you have to enter your username and password. Then the main page of the blog will open in front of you. You will be given respective rights according to your registration type e.g. admin, author etc.

Please sign in

username

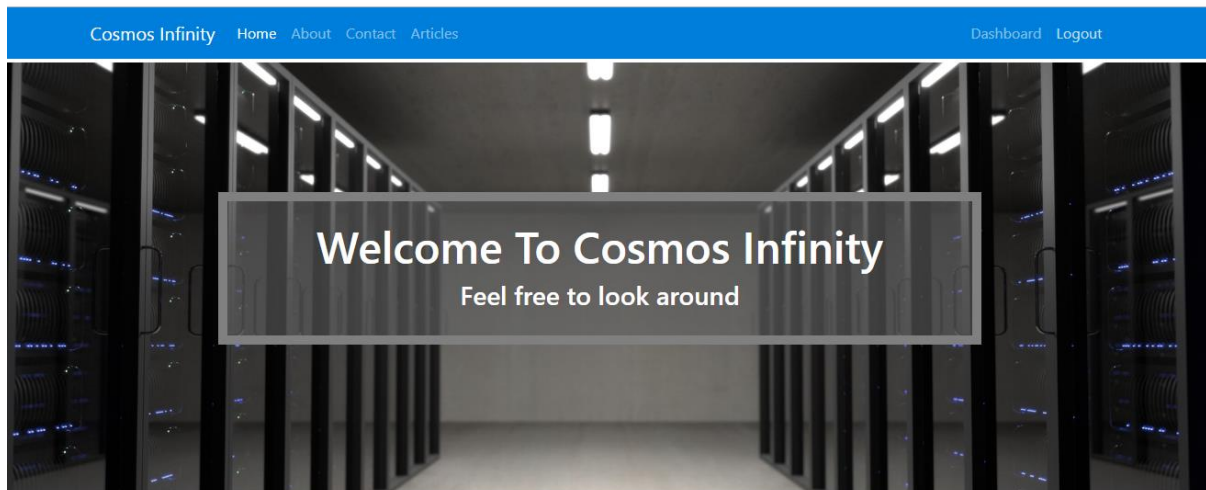
password

☐ remember me

Sign in

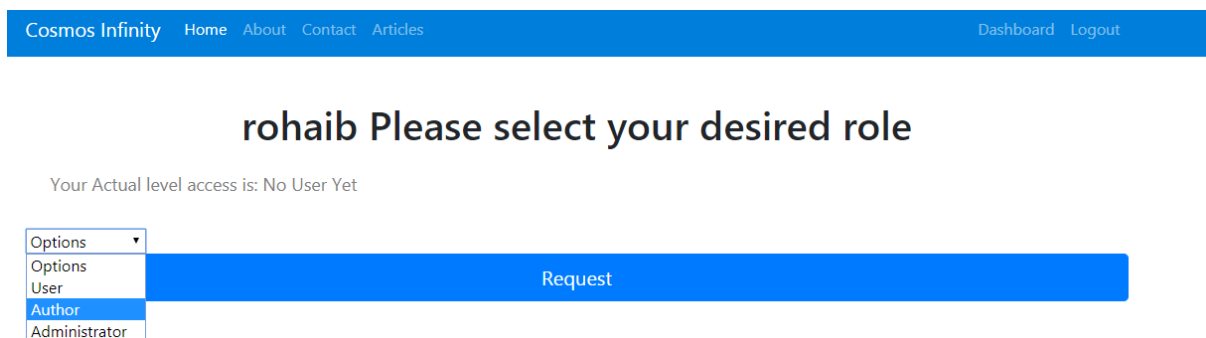
Log out Page:

User is redirected to the home page. User has to give login again to access his account.



Author

If any author wants to write his articles then he has to sign up in the blog. First they have to register himself as they are going to write his article or just they want to become a registered article. The author has to select the author option in the blog.



First this is also the anonymous users after selecting the option from the blog, an anonymous user will become a registered user, author, and also admin of the blog although this request is accepted by admin.

This is the article main page when admin accepts the request. After becoming a registered user, they will publish his article on the blog. An author can only modify or delete his own article which they have published. An author can also add a picture to the article, and the picture should have good quality and be in a prominent position in the article page. An article also contains a picture, which is included in the teaser. An author has to add keywords to the article to simplify it for users to search for their own type of interest from the search bar, and hash tags added are easy for the user to find their own interest type of article. An author and admin have the same authorities, but an author can only add or delete his own articles which they have written and uploaded. Overall, an admin has more authorities than the author.

Cosmos Infinity
Home
About
Contact
Articles
Dashboard
Logout

Dashboard

Articles

Users

Accept Or Remove Comment

Logout

Articles

New Article

#ID	Author	Title	Category	Posted On	Action
-----	--------	-------	----------	-----------	--------

Registered User:

All the user which are publish his articles once time then they will become a registered user just like author next time when this users want to write a article again then they have only have to log in the blog and just write his articles . this has the same authorities on all his own articles they will edit his articles anytime without any problem. Now this users will have the same authorities as the author.

Anonymous User:

This type of user can only read the article and if they want to comment then its goes to the dashboard of admin, after accepting from admin then this comment is public for all the user

4.2. Administration Manual

User name = admin

Password = adminadmin

Admin

First you login into the blog from main page, it is the admin page. Admin has all the authority to remove or delete the articles.

Dashboard
Articles
Accept Or Reject User
Accept Or Remove Comment
Logout

This is admin dash board where admin see all the details of the page what's happen in this blog. Admin click on the articles page and on the page all the article in front of admin. Request also in front of the blog which articles he has to accepted our rejected . only admin has this authority to accept or reject the account and only admin has the authority to change the role of user .

Dashboard	Articles New Article				
Articles					
Users					
Accept Or Remove Comment					
Logout					

#ID	Author	Title	Category	Posted On	Action
1	Cosmos Administrator	Sample Article	tag1;tag2	2019-06-29 10:54:17.364611	Edit Delete
2	David Condor	Fisrt article	Technology;Life;University	2019-06-29 16:03:47.158094	Edit Delete

Admin has the authority to edit or delete the articles which are published by the authors or from registered users. Admin can also write the new articles .only admin has the authority to edit or delete the other users articles.

Dashboard
Articles
Users
Accept Or Remove Comment
Logout

User Name	Email	Current role	Select role
Author Cosmos	author@cosmos.com	Author	User Author Admin
User Cosmos	user@cosmos.com	User	User Author Admin
NoUser Cosmos	nouser@cosmos.com	No authorized yet	User Author Admin Reject
David Condor	wc07685@cosmos.com	Author	User Author Admin

wants to become an User

admin can also see the details of all the users , clearly admin can make any user to become a author or user it totally depends on the admin .

Dashboard	New Comments				
Articles					
Users					
Accept Or Remove Comment					
Logout					
Comment author		Article URL	Comment	Approve	Remove
useruser useruser		Click here to read	Comment from user	Accept	Reject

If any user comment on the article then admin has the authority to accept or reject the comment . if the comment is useful or related to article then admin has to allow the comment on the blog .

User Cosmos	user@cosmos.com	User	User Author Admin	
NoUser Cosmos	nouser@cosmos.com	No authorized yet	User Author Admin Reject	wants to become an User
David Condor	wc07685@cosmos.com	Author	User Author Admin	
rohaib hassan	rohaib@gmail.com	No authorized yet	User Author Admin Reject	wants to become an Author

Admin has to accept this request to become a author. because this person want to publish his own article on the blog and share his opinion .

4.2.1 Create New Article :

when any of the user want to write any article on the blog then he has to first off all register himself and after this they have to upload his article , if this is the author post then this is automatically uploaded and visible for All types of users which are visiting this blog. If any anonymous user want to write his article then its impossible for him to write his article. Author can also edit or delete his article if at the later stage when the author wants to update his article ,this is only possible for author to edit his own article .

Cosmos Infinity	Home	About	Contact	Articles	Dashboard	Logout
Dashboard	Articles					New Article
Articles						
Users						
Accept Or Remove Comment						
Logout						
#ID	Author	Title	Category	Posted On	Action	

5. DEVIATION FROM THE INITIAL PROPOSAL

Project changed from Django to Flask on 2019, April 27th, 2019

Initially, the project was designed to be developed with Django. However, after conducting the first investigations and reading about previous learning experiences. It was concluded that Flask is "possibly" more friendly with people who are learning a new programming language.

*"I would recommend that you go with Flask — you'll figure out how the pieces fit together more easily, and you'll never have functionality lying around that you're not actually using."*¹

*"But if you're a beginner and want to get into web development with python, I'd suggest you start with Flask. The things you learn in Flask can be applied to Django. It's just easier with Flask."*²

¹ <https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v>

² <https://dev.to/benprax/flask-vs-django-1l4k>