Project Description

Aim: To create a blog website with a contact form, you can use the **Flask web framework along with MongoDB for data storage**. The Flask application will have several routes defined to handle different URLs. The homepage ("/") will render an "index.html" template, the about page ("/about") will render an "about.html" template, and there will be separate routes for individual blog posts ("/post", "/post1", "/post2") that render corresponding post templates.

For the contact form, there will be a route ("/contact") that accepts both GET and POST requests. When a user visits this URL with a GET request, the contact page ("/contact") will be rendered. If the user submits a POST request by filling out the contact form, the form data (name, email, phone, message) will be extracted from the request. The data will then be inserted into the MongoDB database using the `db.blogContacts.insert_one()` method. If the insertion is successful, the user will be redirected to the contact page ("/contact"), and if an error occurs, an "error_page.html" template will be rendered.

Overall, this code sets up the necessary routes and templates for a blog website with a contact form, providing a basic structure for further development.

Whole code is available on GitHub: https://github.com/SIDVJSINGH/blogSite

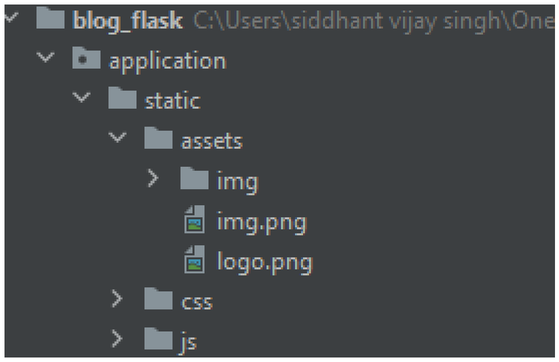
Steps:

1. Creating a directory in pycharm(python code editor).

2. Installing flask for server-side scripting and flask-pymongo for database of contacts from the form.

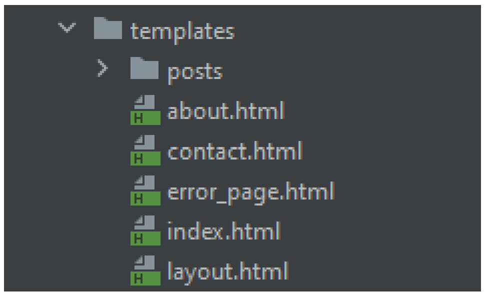
3. Making html, css, and js(javascript) from bootstrap.

4. Creating 2 different folders.



```
blog_flask  C:\Users\siddhant vijay singh\One
  application
    static
      assets
        img
        img.png
        logo.png
      css
      js
```

1 for all css and js code.

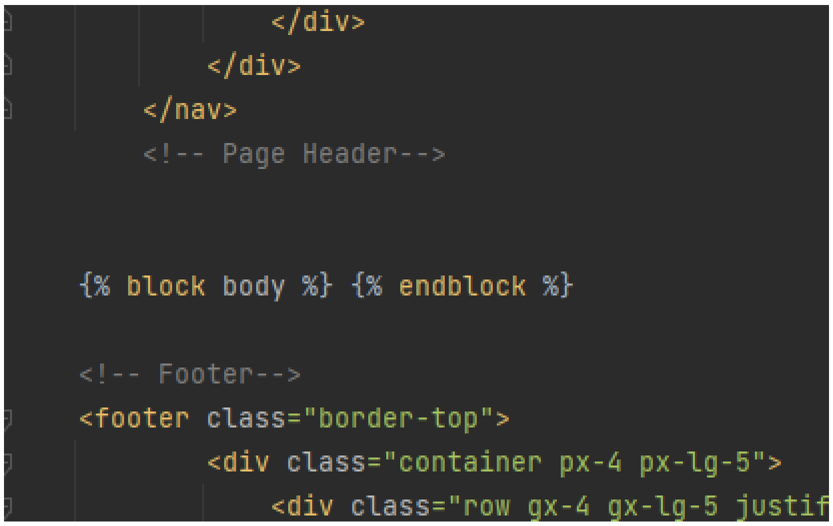


```
templates
  posts
  about.html
  contact.html
  error_page.html
  index.html
  layout.html
```

2nd for all html file codes

5. The “layout.html” file holds a general layout of the page that is common for all the website.

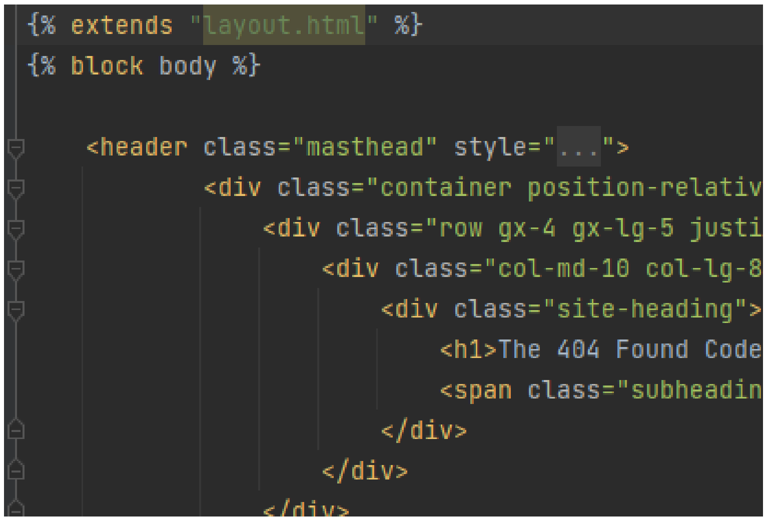
Here I used Jija templating.



```
            </div>
        </div>
    </nav>
    <!-- Page Header-->


{% block body %} {% endblock %}

<!-- Footer-->
<footer class="border-top">
        <div class="container px-4 px-lg-5">
            <div class="row gx-4 gx-lg-5 justif
```

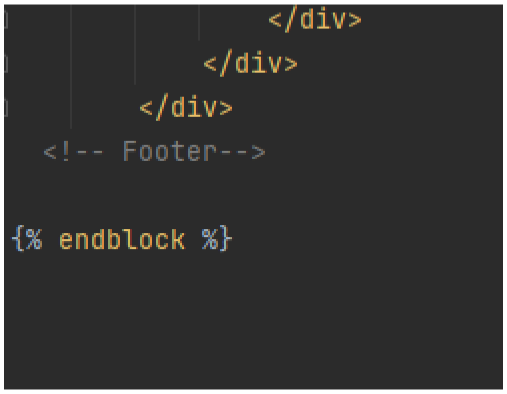
{% block body %} {% endblock %} acts as a general template for all other files to get their header and footer part from.

6. Other html files will use this “layout.html” file as a template.



```
{% extends "layout.html" %}
{% block body %}

    <header class="masthead" style="...">
            <div class="container position-relativ
                <div class="row gx-4 gx-lg-5 justi
                    <div class="col-md-10 col-lg-8
                        <div class="site-heading">
                            <h1>The 404 Found Code
                            <span class="subheadin
                        </div>
                    </div>
                </div>
```

Above {% extends “layout.html” %} will add all the code of layout.html to the current html file and add mid section (body part) in {% block body %} start of body.



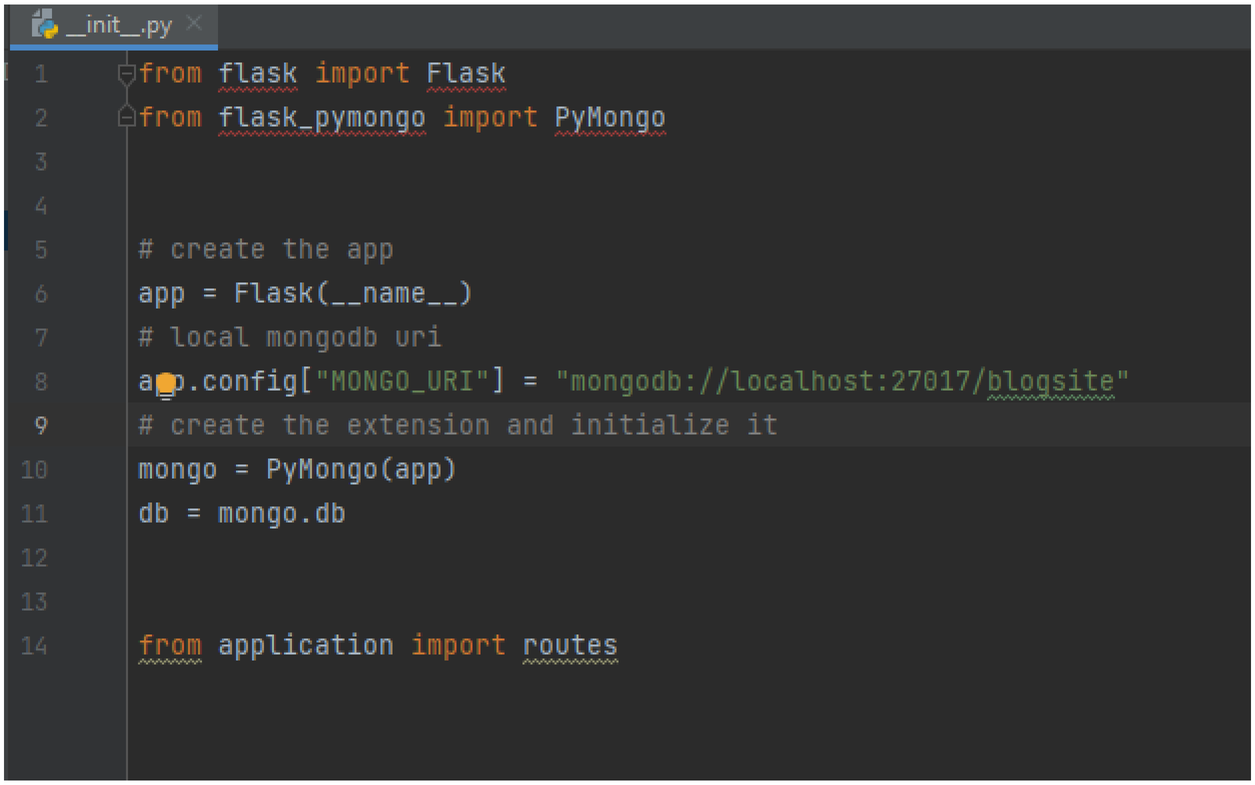
```
            </div>
        </div>
    </div>
  <!-- Footer-->

{% endblock %}
```

{% endblock %} end body.

Let’s come to the application part of the directory.

7. There are 2 python files that serves as initialization and routes of the webpage.

1st we have __init__.py file, that is a initialization file.


__init__.py

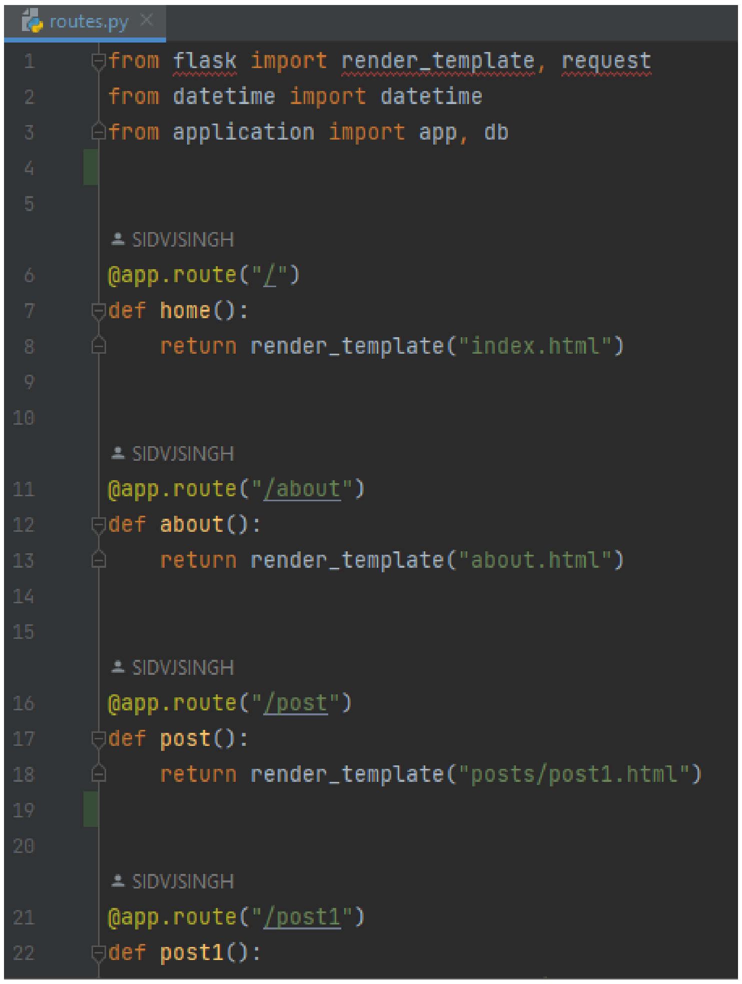
```python
from flask import Flask
from flask_pymongo import PyMongo


# create the app
app = Flask(__name__)
# local mongodb uri
app.config["MONGO_URI"] = "mongodb://localhost:27017/blogsite"
# create the extension and initialize it
mongo = PyMongo(app)
db = mongo.db


from application import routes
```

Code Description:

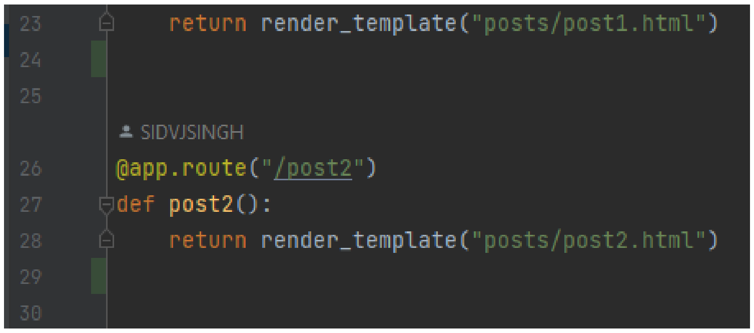
1. **from flask import Flask**: Imports the Flask module, which is a web framework used for building web applications in Python.
2. **from flask_pymongo import PyMongo**: Imports the PyMongo module, which is a Flask extension that provides integration with MongoDB.
3. **app = Flask(__name__)**: Creates a Flask application instance, with the name set to the module name or package name.
4. **app.config["MONGO_URI"] = "mongodb://localhost:27017/blogsite"**: Sets the MongoDB URI for the Flask application's configuration. In this case, it specifies a local MongoDB server running on the default port 27017, and the database name is "blogsite".
5. **mongo = PyMongo(app)**: Creates a PyMongo extension instance, passing the Flask application as an argument. This initializes the PyMongo extension with the Flask application's configuration.
6. **db = mongo.db**: Retrieves the MongoDB database instance from the PyMongo extension. This allows you to interact with the MongoDB database using the **db** object.
7. **from application import routes**: Imports the **routes** module from the **application** package. This is likely where the application's route handlers and views are defined.

Overall, these lines of code set up a Flask application with MongoDB integration, configure the MongoDB connection, and import the application's routes for handling different URLs.

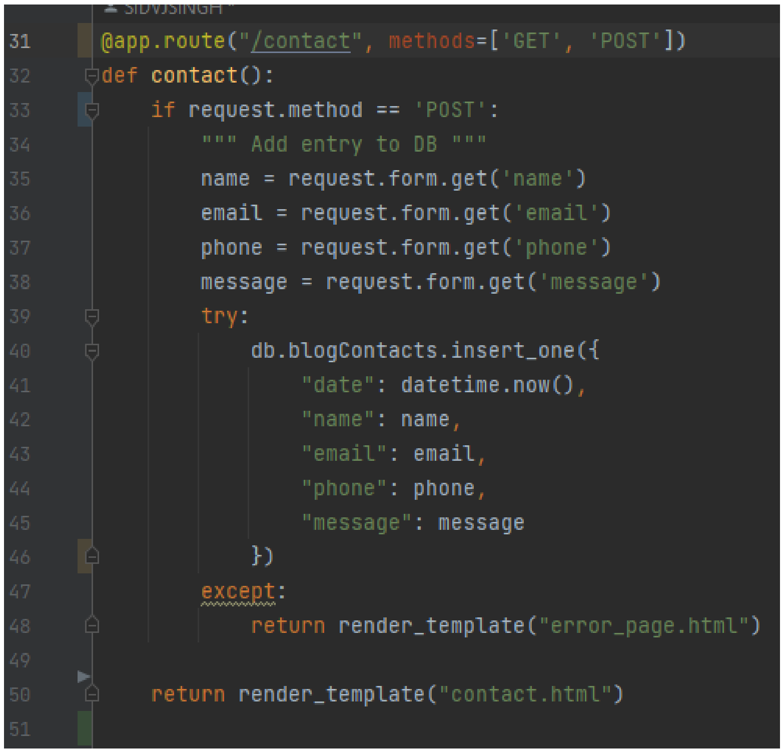
2nd we have routes.py file



```
routes.py

from flask import render_template, request
from datetime import datetime
from application import app, db


SIDVJSINGH
@app.route("/")
def home():
    return render_template("index.html")


SIDVJSINGH
@app.route("/about")
def about():
    return render_template("about.html")


SIDVJSINGH
@app.route("/post")
def post():
    return render_template("posts/post1.html")


SIDVJSINGH
@app.route("/post1")
def post1():
```



```
    return render_template("posts/post1.html")


SIDVJSINGH
@app.route("/post2")
def post2():
    return render_template("posts/post2.html")

```



```python
@app.route("/contact", methods=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        """ Add entry to DB """
        name = request.form.get('name')
        email = request.form.get('email')
        phone = request.form.get('phone')
        message = request.form.get('message')
        try:
            db.blogContacts.insert_one({
                "date": datetime.now(),
                "name": name,
                "email": email,
                "phone": phone,
                "message": message
            })
        except:
            return render_template("error_page.html")

    return render_template("contact.html")
```

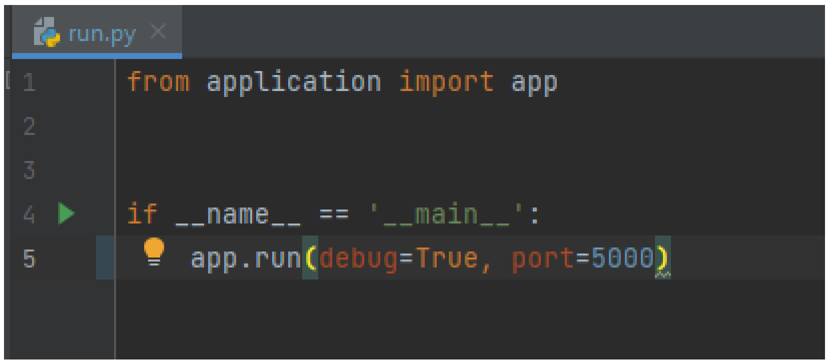
Code Description:

1. **@app.route("/")**: Defines a route handler for the root URL ("/"). When a user visits the root URL, the **home()** function is executed. It renders the "index.html" template and returns the rendered HTML page.
2. **@app.route("/about")**: Defines a route handler for the "/about" URL. When a user visits this URL, the **about()** function is executed. It renders the "about.html" template and returns the rendered HTML page.
3. **@app.route("/post")**: Defines a route handler for the "/post" URL. When a user visits this URL, the **post()** function is executed. It renders the "posts/post1.html" template and returns the rendered HTML page.
4. **@app.route("/post1")**: Defines a route handler for the "/post1" URL. When a user visits this URL, the **post1()** function is executed. It renders the "posts/post1.html" template and returns the rendered HTML page. Note that this route handler has the same behavior as the previous one.
5. **@app.route("/post2")**: Defines a route handler for the "/post2" URL. When a user visits this URL, the **post2()** function is executed. It renders the "posts/post2.html" template and returns the rendered HTML page.
6. **@app.route("/contact", methods=['GET', 'POST'])**: Defines a route handler for the "/contact" URL, which accepts both GET and POST requests. When a user visits this URL with a GET request, the **contact()** function is executed. It renders the "contact.html" template and returns the rendered HTML page. When a user submits a POST request to this URL (typically through a form submission), the function checks for the request method and processes the form data. It retrieves the form data (name, email, phone, message) from the request, inserts it into the MongoDB database using the **db.blogContacts.insert_one()** method, and redirects to the "contact.html" template if successful. If an error occurs during the database insertion, it renders an "error_page.html" template.

These route handlers handle different URLs in the Flask application, rendering specific HTML templates and performing actions based on the request method (GET or POST).

Note: *by* ***default*** *a route is a* ***GET*** *method.*

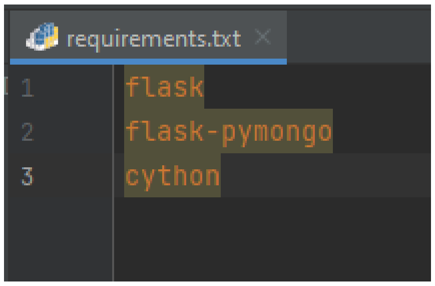
There is a 1 more file that runs all these folders and files, that is run.py file.



```python
from application import app


if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

This file simply run the application for us on local port 5000 and with all debug logs on, all error logs will be visible on webpage itself with debug=True.

I have also attached a requirement.txt file for all the libraries to install before starting this application.

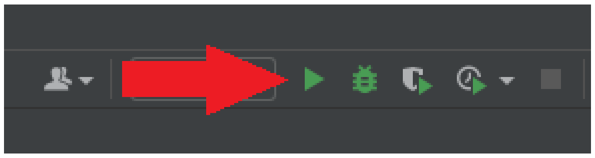


```
flask
flask-pymongo
cython
```

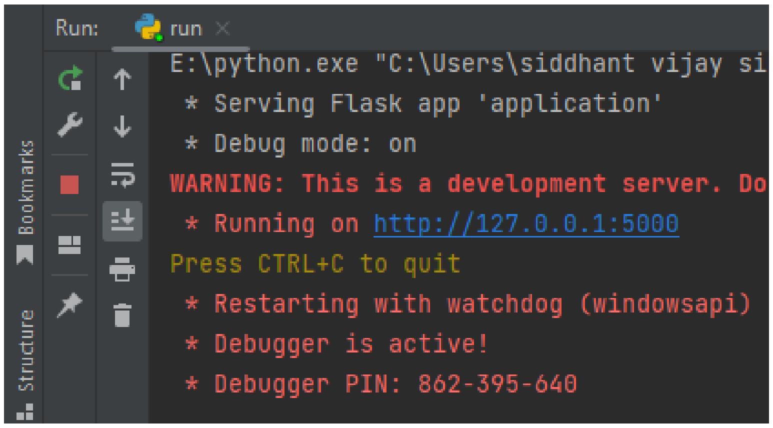
Above is all about the code

Now let's see the results

- We can Run the app by simple clicking on the play button

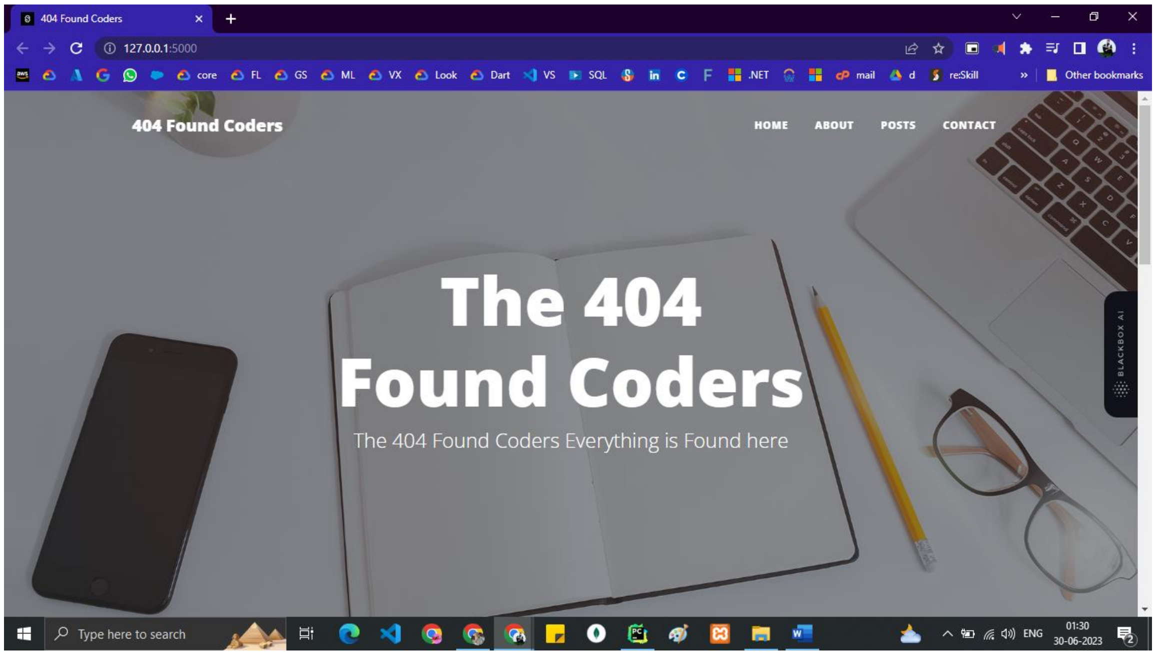


- The app is deployed in local now.



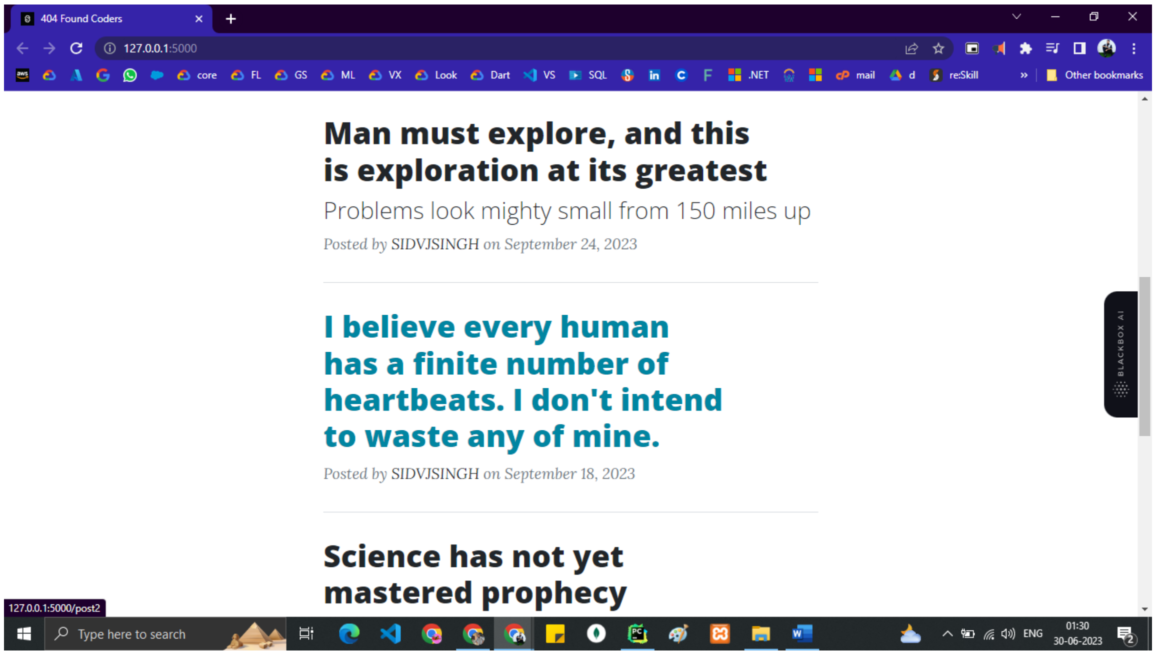
```
E:\python.exe "C:\Users\siddhant vijay si
 * Serving Flask app 'application'
 * Debug mode: on
WARNING: This is a development server. Do
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 862-395-640
```

Home Page: Where all pages are accessible.



The 404 Found Coders

The 404 Found Coders Everything is Found here



Man must explore, and this is exploration at its greatest

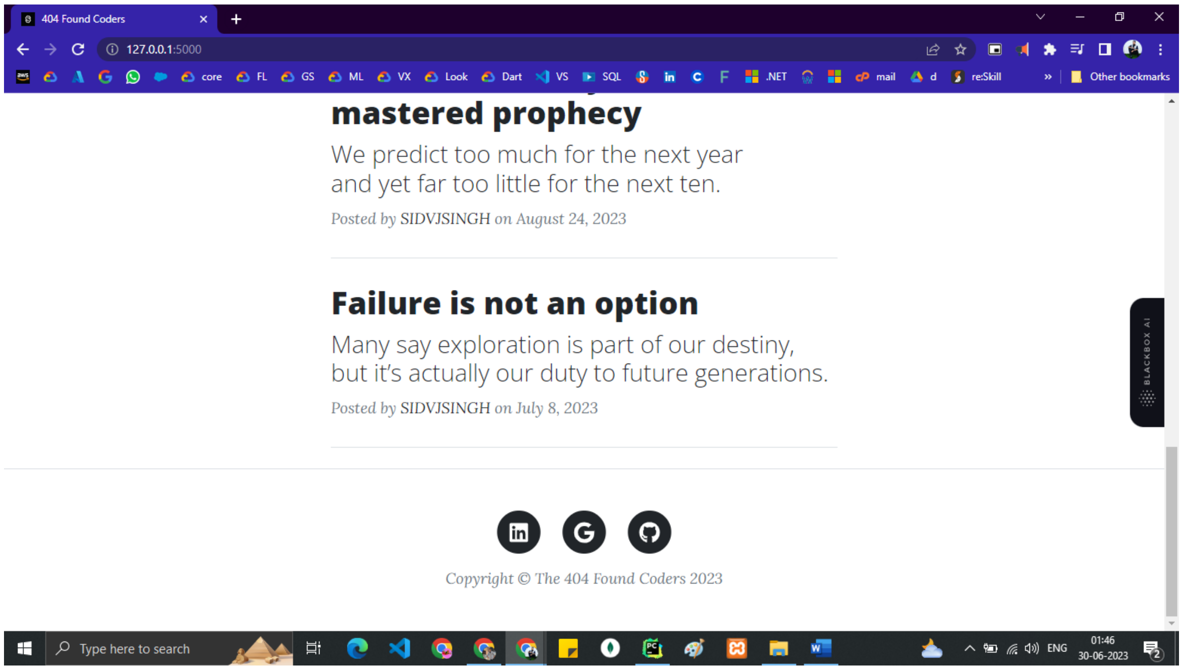
Problems look mighty small from 150 miles up

Posted by SIDVJSINGH on September 24, 2023

I believe every human has a finite number of heartbeats. I don't intend to waste any of mine.

Posted by SIDVJSINGH on September 18, 2023

Science has not yet mastered prophecy



mastered prophecy

We predict too much for the next year and yet far too little for the next ten.

Posted by SIDVJSINGH on August 24, 2023

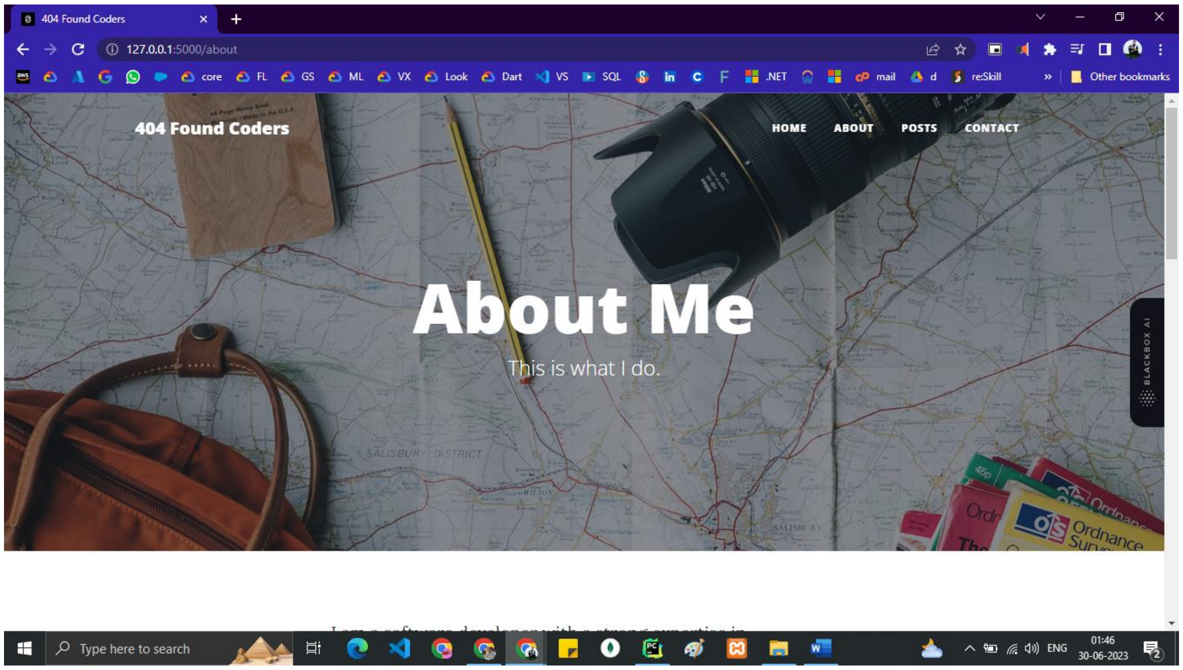
Failure is not an option

Many say exploration is part of our destiny, but it's actually our duty to future generations.

Posted by SIDVJSINGH on July 8, 2023

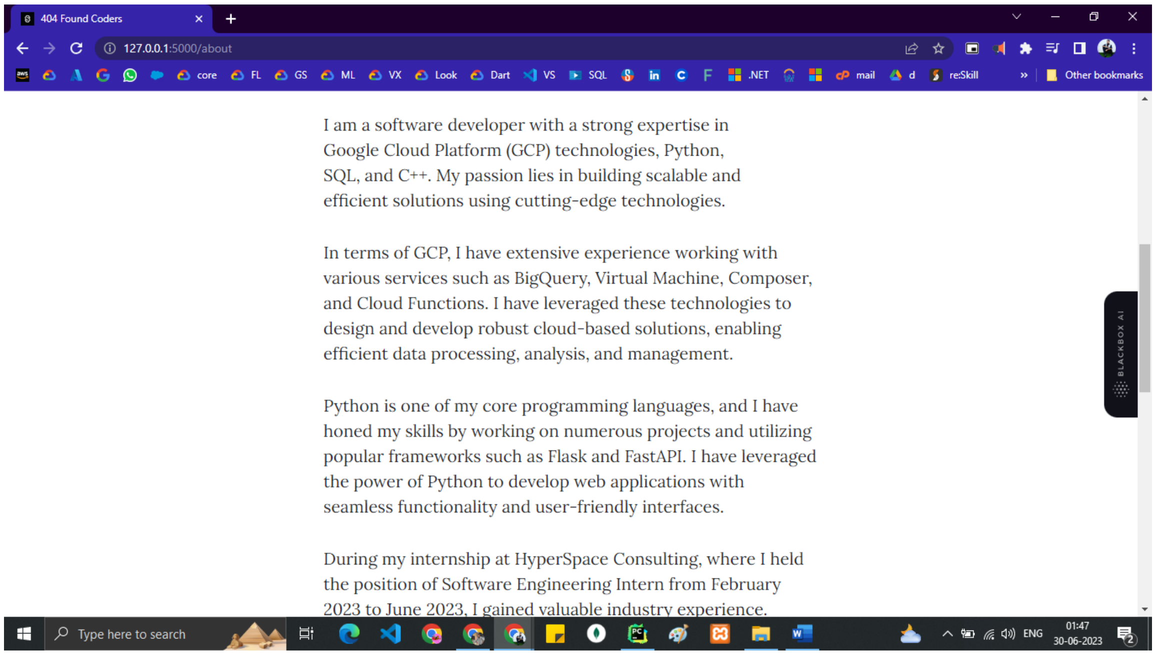
Copyright © The 404 Found Coders 2023

About page: About me is written on this page



About Me

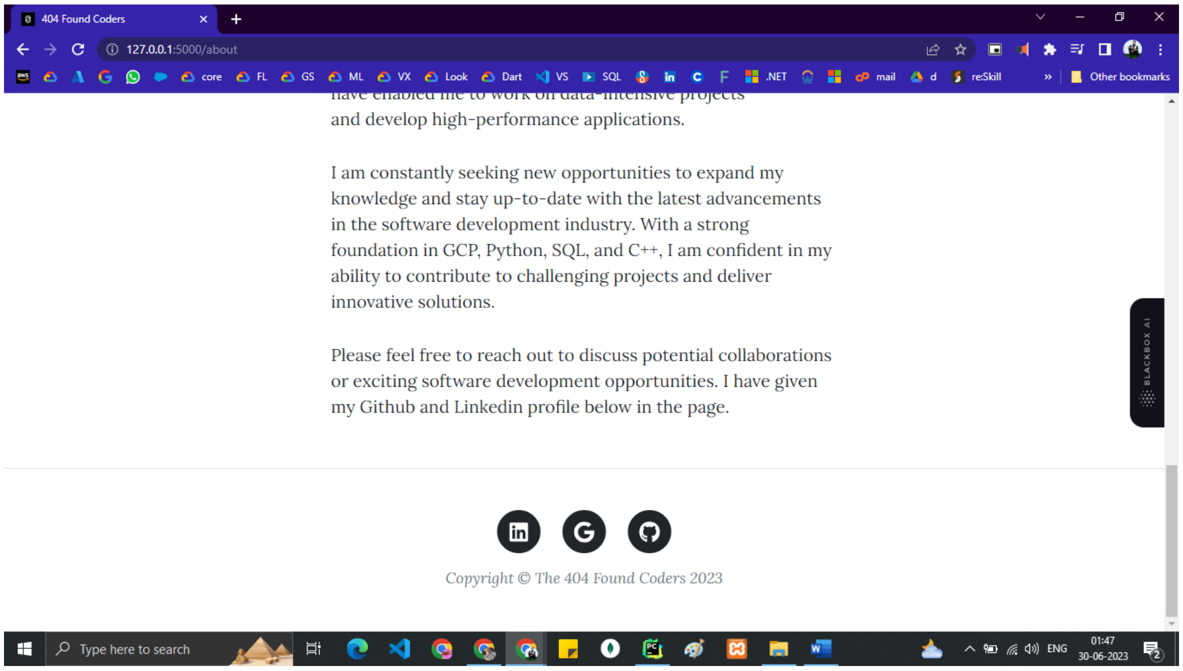
This is what I do.


I am a software developer with a strong expertise in Google Cloud Platform (GCP) technologies, Python, SQL, and C++. My passion lies in building scalable and efficient solutions using cutting-edge technologies.

In terms of GCP, I have extensive experience working with various services such as BigQuery, Virtual Machine, Composer, and Cloud Functions. I have leveraged these technologies to design and develop robust cloud-based solutions, enabling efficient data processing, analysis, and management.

Python is one of my core programming languages, and I have honed my skills by working on numerous projects and utilizing popular frameworks such as Flask and FastAPI. I have leveraged the power of Python to develop web applications with seamless functionality and user-friendly interfaces.

During my internship at HyperSpace Consulting, where I held the position of Software Engineering Intern from February 2023 to June 2023, I gained valuable industry experience.



404 Found Coders

127.0.0.1:5000/about

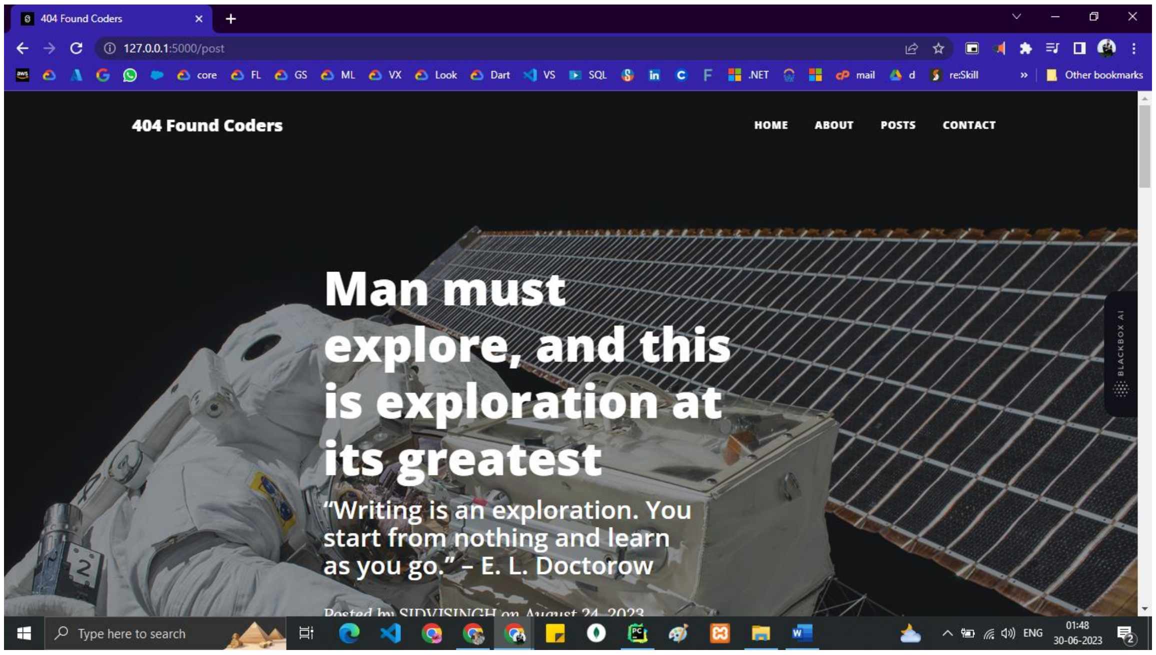
and develop high-performance applications.

I am constantly seeking new opportunities to expand my knowledge and stay up-to-date with the latest advancements in the software development industry. With a strong foundation in GCP, Python, SQL, and C++, I am confident in my ability to contribute to challenging projects and deliver innovative solutions.

Please feel free to reach out to discuss potential collaborations or exciting software development opportunities. I have given my Github and Linkedin profile below in the page.

Copyright © The 404 Found Coders 2023

Post page: where most recent post is shown/ available.



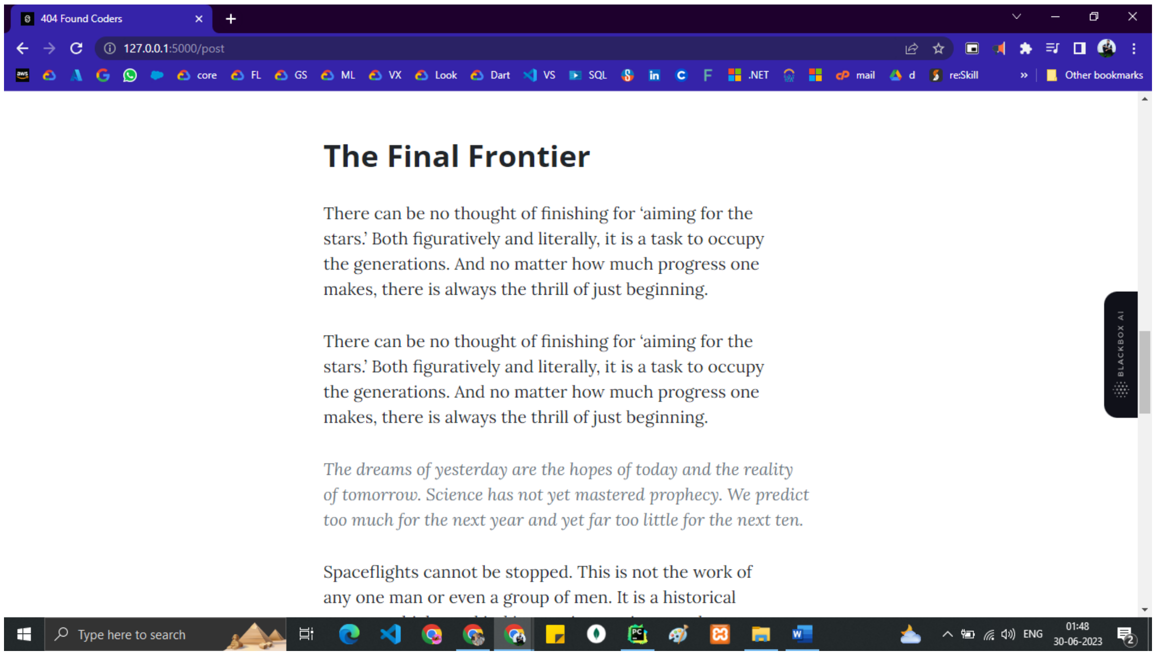
404 Found Coders

HOME ABOUT POSTS CONTACT

Man must explore, and this is exploration at its greatest

"Writing is an exploration. You start from nothing and learn as you go." – E. L. Doctorow

Posted by SIDVISINGH on August 24, 2023



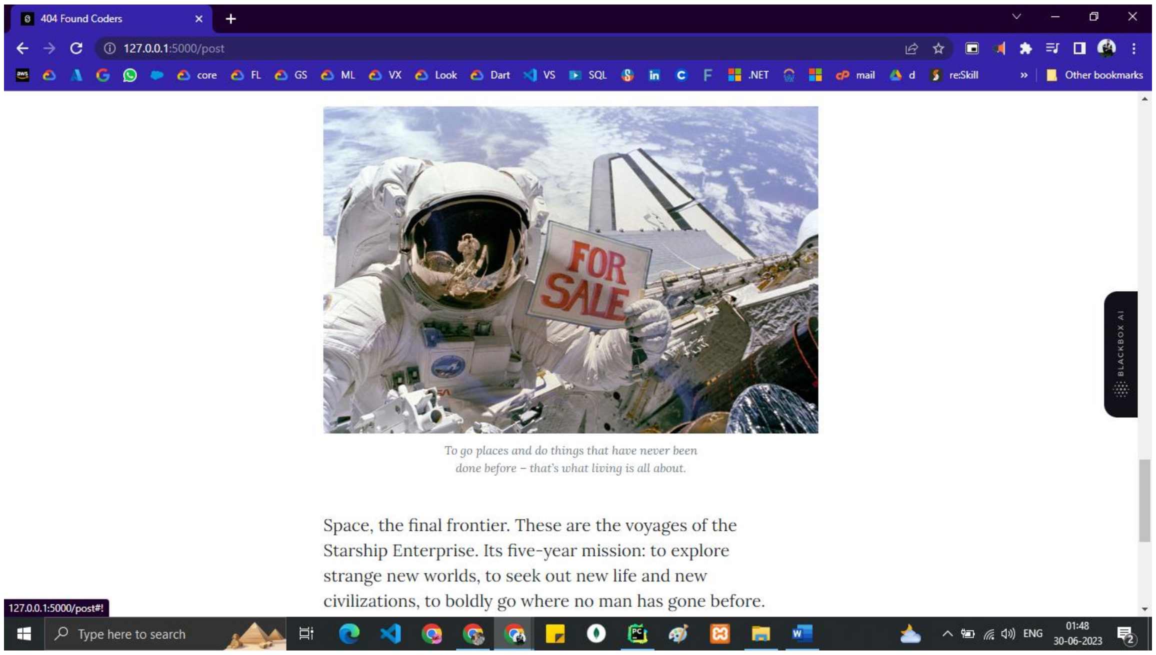
The Final Frontier

There can be no thought of finishing for 'aiming for the stars.' Both figuratively and literally, it is a task to occupy the generations. And no matter how much progress one makes, there is always the thrill of just beginning.

There can be no thought of finishing for 'aiming for the stars.' Both figuratively and literally, it is a task to occupy the generations. And no matter how much progress one makes, there is always the thrill of just beginning.

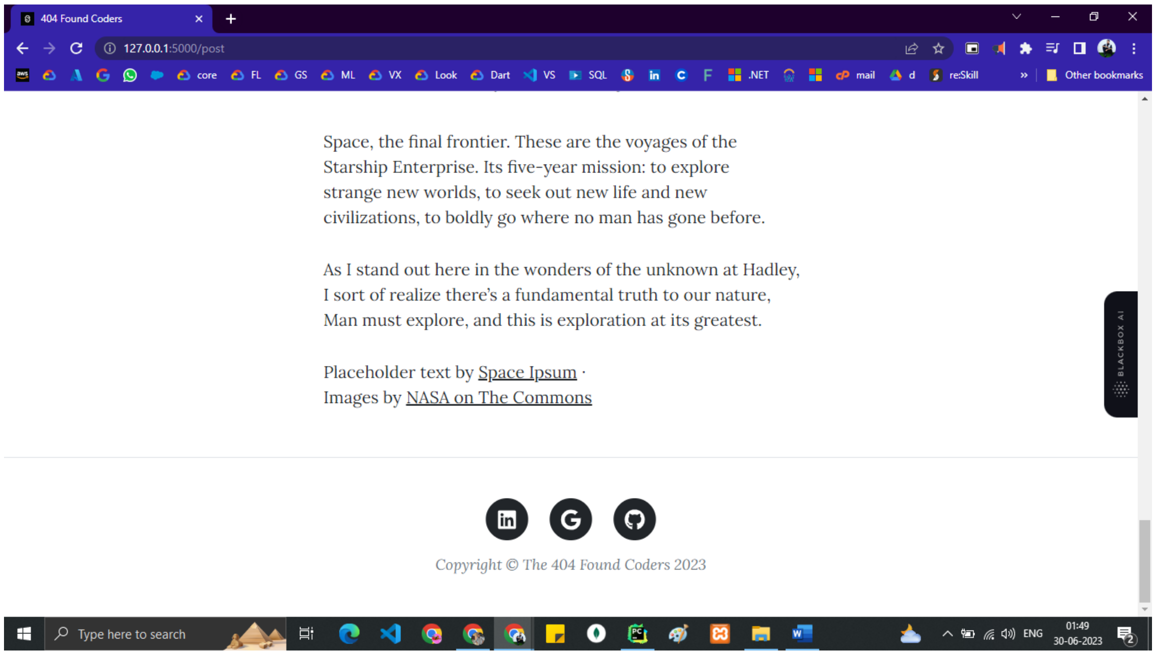
The dreams of yesterday are the hopes of today and the reality of tomorrow. Science has not yet mastered prophecy. We predict too much for the next year and yet far too little for the next ten.

Spaceflights cannot be stopped. This is not the work of any one man or even a group of men. It is a historical



To go places and do things that have never been done before – that's what living is all about.

Space, the final frontier. These are the voyages of the Starship Enterprise. Its five-year mission: to explore strange new worlds, to seek out new life and new civilizations, to boldly go where no man has gone before.



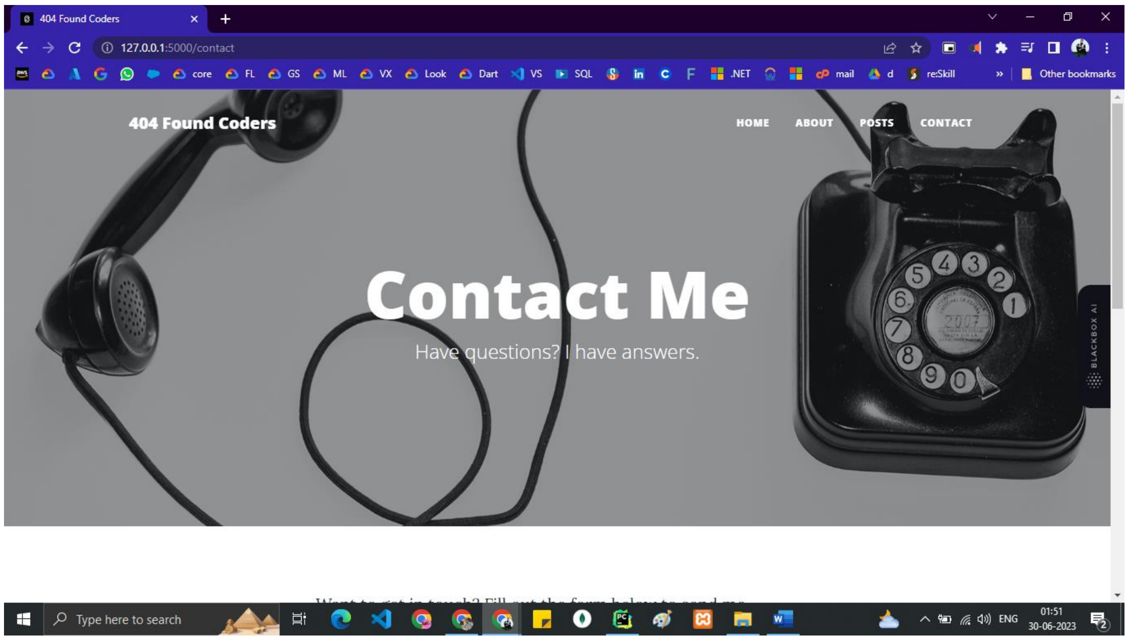
Space, the final frontier. These are the voyages of the Starship Enterprise. Its five-year mission: to explore strange new worlds, to seek out new life and new civilizations, to boldly go where no man has gone before.

As I stand out here in the wonders of the unknown at Hadley, I sort of realize there's a fundamental truth to our nature, Man must explore, and this is exploration at its greatest.

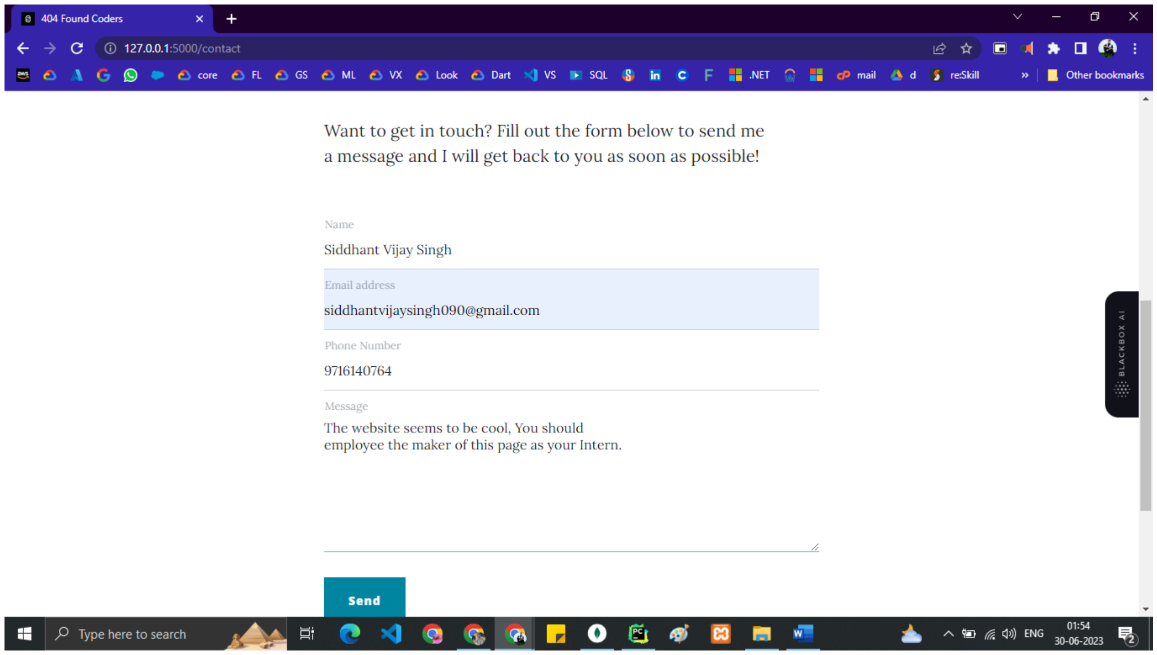
Placeholder text by Space Ipsum ·
Images by NASA on The Commons

Copyright © The 404 Found Coders 2023

Contact page: where you can put your queries.



You can write your name, email address, phone number and query for the maker, and hit send.


Want to get in touch? Fill out the form below to send me a message and I will get back to you as soon as possible!

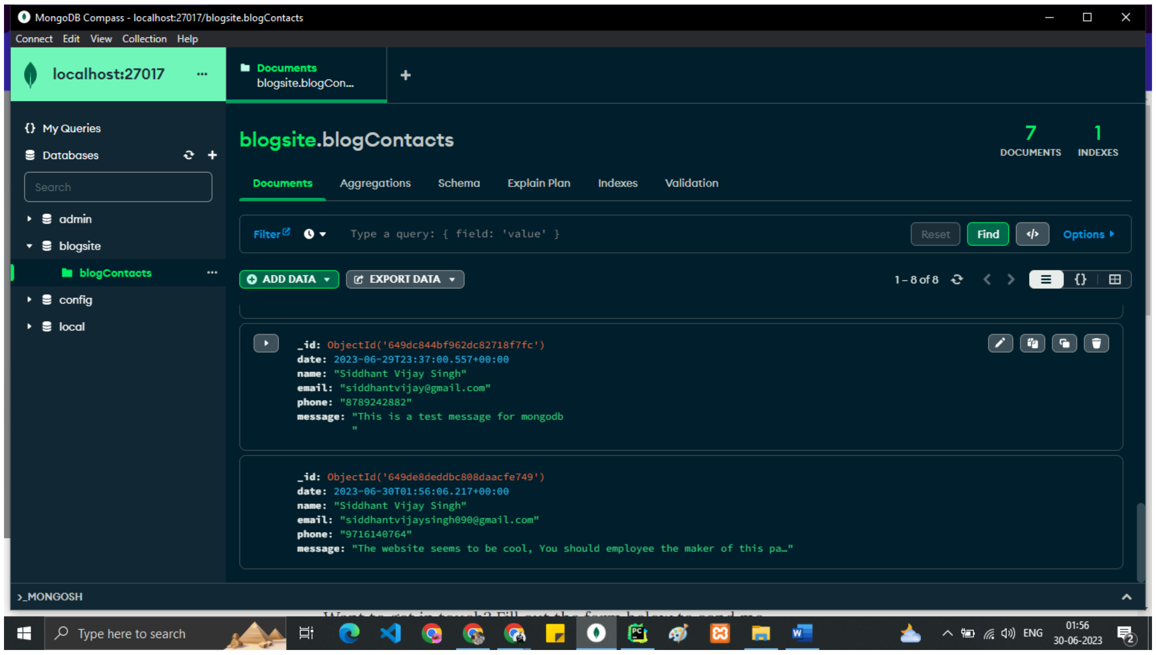
Name
Siddhant Vijay Singh

Email address
siddhantvijaysingh090@gmail.com

Phone Number
9716140764

Message
The website seems to be cool, You should employee the maker of this page as your Intern.

This send button will make a post request to the server and all your contact information will be put into the database.



```
_id: ObjectId('649dc844bf962dc82718f7fc')
date: 2023-06-29T23:37:00.557+00:00
name: "Siddhant Vijay Singh"
email: "siddhantvijay@gmail.com"
phone: "8789242882"
message: "This is a test message for mongodb
         "

_id: ObjectId('649de8deddbc808daacfe749')
date: 2023-06-30T01:56:06.217+00:00
name: "Siddhant Vijay Singh"
email: "siddhantvijaysingh090@gmail.com"
phone: "9716140764"
message: "The website seems to be cool, You should employee the maker of this pa…"
```

At MongoDB Compass you can see that a "blogsite" named database is created and "blogContacts" named table is also created with the contacts of the customer.

Whole code is available on GitHub: https://github.com/SIDVJSINGH/blogSite

- **In conclusion (about myself as developer of this project),** the project involves setting up a Flask application with MongoDB integration. The Flask application handles various routes to render HTML templates for the homepage, about page, blog posts, and contact page. The contact page includes a contact form that accepts user input and stores it in the MongoDB database. The project showcases the developer's skills in Google Cloud Platform (GCP), Python, SQL, and C++. Additionally, the developer has gained practical experience during their internship at HyperSpace Consulting, where they worked with Python frameworks such as Flask and FastAPI, along with Google Cloud Platform. With their expertise in these technologies, the developer is well-equipped to build scalable web applications and handle data-intensive projects.

- **Potential Future Improvements:**

While the project demonstrates a functional blog website with a contact form, there are several potential future improvements that can be considered:

1. User Authentication: Implement user authentication and authorization functionality to allow registered users to log in, create personalized profiles, and interact with the website based on their roles (e.g., admin, regular user).

2. User Input Validation: Enhance the contact form by implementing client-side and server-side input validation to ensure that the submitted data is accurate and follows the desired format.

3. Error Handling and Feedback: Improve error handling by providing meaningful error messages or alerts to users when they encounter issues, such as failed form submissions or database errors. This helps users understand and troubleshoot problems more effectively.

4. Data Pagination: Implement pagination for blog posts to improve performance when handling a large number of posts. This allows users to navigate through blog posts more efficiently and improves the overall user experience.

5. Search Functionality: Add a search feature that enables users to search for specific blog posts based on keywords, tags, or categories. Implementing a search functionality can enhance user engagement and make it easier for users to find relevant content.

6. Comments Section: Introduce a comments section for blog posts, allowing users to engage in discussions, provide feedback, and share their thoughts. This promotes community interaction and enhances user engagement on the website.

7. Responsive Design: Optimize the website's design and layout to ensure it is responsive and mobile-friendly. This ensures a seamless user experience across various devices and screen sizes.

8. Testing and Bug Fixes: Conduct thorough testing to identify and fix any bugs, usability issues, or security vulnerabilities that may arise. Regularly updating and maintaining the project will ensure its stability and security over time.

9. Performance Optimization: Optimize the website's performance by implementing caching mechanisms, optimizing database queries, and leveraging techniques such as minification and compression of static assets to improve loading times and overall responsiveness.

10. Analytics and Monitoring: Incorporate analytics and monitoring tools to track website usage, user behavior, and performance metrics. This data can provide valuable insights for making data-driven decisions and continuously improving the website.

By considering these potential future improvements, the project can be enhanced with additional features, improved usability, and enhanced performance, providing a more robust and engaging experience for users.

Thanks for your time and giving me this opportunity.

Regards

Siddhant Vijay Singh

siddhantvijaysingh090@gmail.com

http://www.github.com/sidvjsingh

http://www.linkedin.com/in/sidvjsingh