Project Report (Computer Networks)

Topic: Simple Web Server Implementation

Group Members

Syed Ali Jodat (20K-0155)

Abdul Ahad Shaikh (20K-0319)

Teacher

Sir Shoaib Raza

Table of contents

Introduction:

The Simple Web Server and Client project aims to develop a basic web server and client using Python programming language. The server will serve static HTML pages to the client on request, and the client will be able to make GET and POST requests to the server. Additionally, the client will have the option to register new websites with the server, which will be stored in a database for future use. The project will utilize Flask web development framework and threading to handle multiple requests simultaneously. This report will cover the project's methodology, requirements analysis, system design, implementation, testing, and results, and provide a conclusion and references used during the development process.

GET Method:

The GET method is used to retrieve data from a server. When a client makes a GET request, the request URL contains parameters in the form of key-value pairs. The server retrieves the requested data and sends it back to the client in the response body. GET requests are typically used to retrieve data that does not change frequently, such as web pages, images, or documents.

POST Method:

The POST method is used to send data to a server to create or update a resource. When a client makes a POST request, the request data is sent in the request body. The server receives the data and performs the necessary actions to create or update the resource. POST requests are typically used to submit data that changes frequently, such as user input or form submissions.

Some key **differences** between GET and POST methods are:

Data Encoding: In GET method, data is encoded in the URL, while in POST method, data is encoded in the request body.

Security: GET requests are less secure than POST requests because data is visible in the URL, which can be intercepted and modified. POST requests are more secure as data is not visible in the URL.

Caching: GET requests can be cached by web browsers and servers, while POST requests cannot be cached.

In summary, GET requests are used to retrieve data from a server, while POST requests are used to send data to a server for creation or update of a resource. Both methods are essential in HTTP communication and are widely used in web development.

Simple Web Server- Our application:

The GET method will be used to request static HTML pages from the server. For example, when a client enters a URL into their web browser, the browser will send a GET request to the server for that specific HTML page. The server will then respond with the requested HTML page, which will be displayed in the client's web browser.

The POST method will be used to submit data from the client to the server. For example, when a client submits a form on a web page, the data entered into the form will be sent to the server using a POST request. The server will then receive the data and store it in a database for future retrieval.

So, in summary, the GET method will be used for retrieving static HTML pages, while the POST method will be used for submitting data from the client to the server.

```
PROBLEMS 4   OUTPUT   TERMINAL   DEBUG CONSOLE

Press CTRL+C to quit
127.0.0.1 - - [10/May/2023 21:44:54] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2023 21:44:56] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2023 21:46:46] "POST /register HTTP/1.1" 302 -
127.0.0.1 - - [10/May/2023 21:46:46] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2023 21:46:55] "GET /admin HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2023 21:48:15] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2023 21:49:32] "POST /register HTTP/1.1" 302 -
127.0.0.1 - - [10/May/2023 21:49:32] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [10/May/2023 21:49:37] "GET /admin HTTP/1.1" 200 -
```

Fig 2- GET and POST method implementation.

Tools and Languages used:

Python: It is the main programming language used for developing the web server and client.

Flask: It is a web framework in Python that will be used to build the web server.

SQLAlchemy: It is an SQL toolkit and ORM that will be used for database integration.

Python Requests: It is a Python library that will be used for making HTTP requests from the client to the server.

The project will be developed using a combination of these tools and languages to achieve the desired functionality of serving static HTML pages, handling GET and POST requests from the client, allowing the client to register new websites with the server, storing the registered websites in a database, and using threading to handle multiple requests simultaneously.

Register a Website:

Once the project is hosted on the localhost, user can access it by opening any web browser and entering "localhost" followed by the assigned port number. Upon accessing the website, the user will be presented with the following page on their screen:

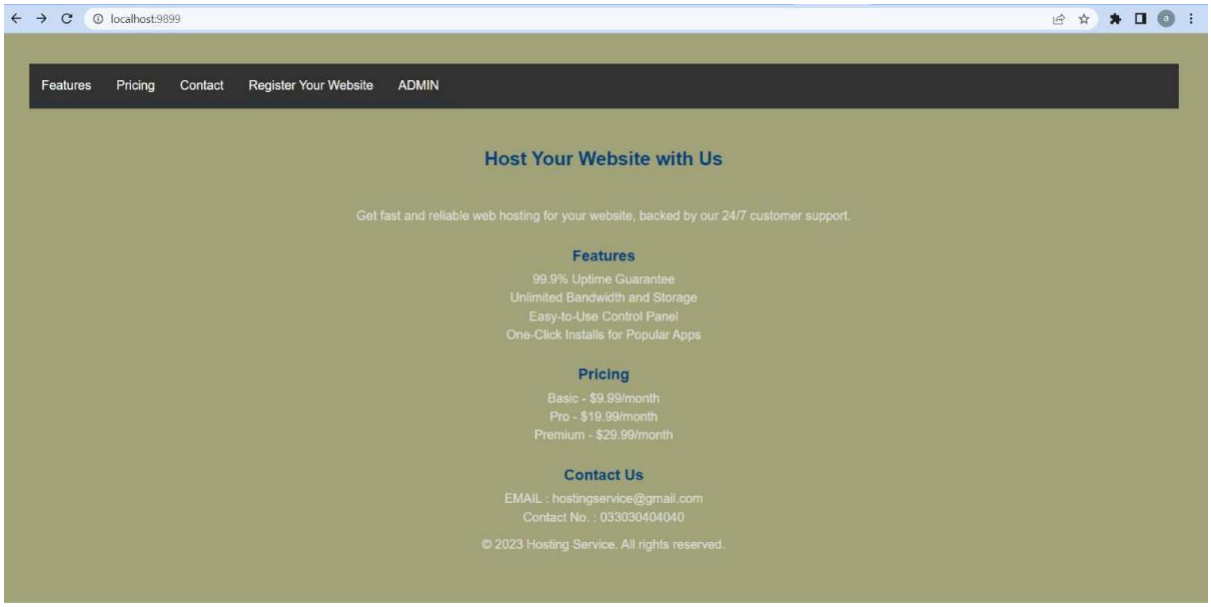

Fig 4.1 – Initial Screen of Simple Web Server

Now, the user Can **View** the following:

1. Features
2. Pricing
3. Contact Numbers

The User can choose to **Register** their desired domain or login as admin.

Fig 4.2 – Navigation Bar

After Clicking On “Register Your Website” Button, the user will be redirected to the following **Registration** page:

Fig 4.3 Website Registration Page

Admin Page:

Admin can also choose to delete any website from the registered websites.

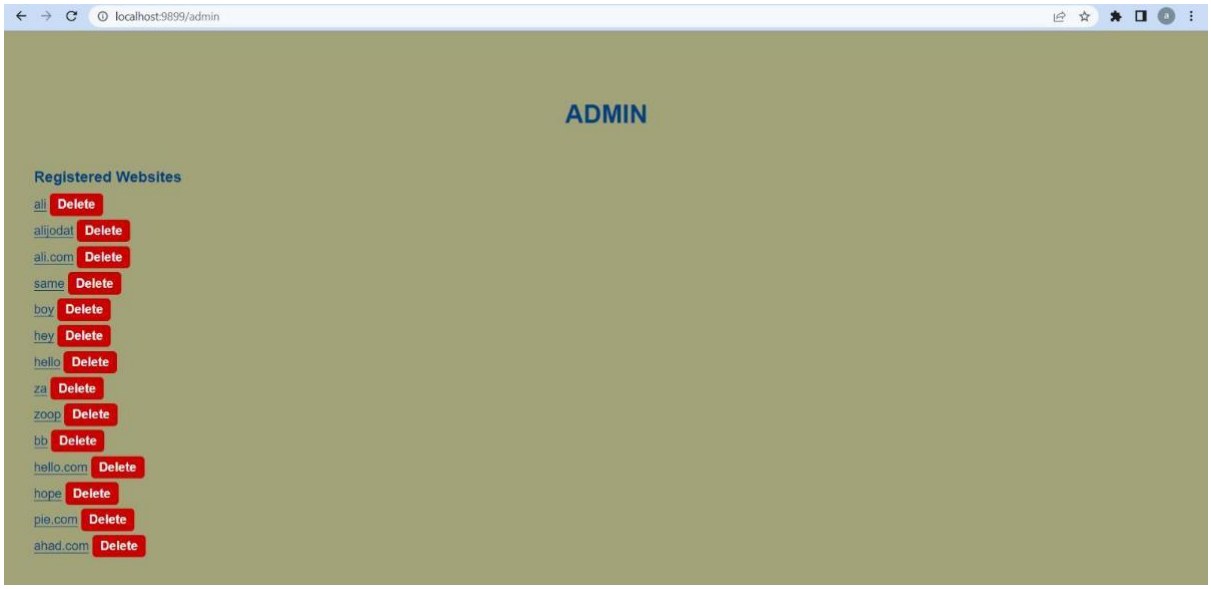

Fig 4.4 Admin Screen

Database of Registered Websites:

Once the website is registered, its html file name is saved in the database along with its URL/Domain. The HTML file itself is saved in a directory named ‘webs’ in the project folder. SQLAlchemy is used to store data. The database can be accessed through ‘sqlite’ viewer.

id	domain_name	html
1	ali	ali.html
2	alijodat	alijodat.html
3	ali.com	ali.com.html
4	same	same.html
5	boy	boy.html
6	hey	hey.html
7	hello	hello.html
8	za	za.html
9	zoop	zoop.html
10	bb	bb.html
11	hello.com	hello.com.html
12	hope	hope.html

1 / 1

Fig 5.1 Database of Registered Websites.

```python
# Configure the database
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///websites.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# Define the Website model
class Website(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    domain_name = db.Column(db.String(200), unique=True, nullable=False)
    html = db.Column(db.Text, nullable=False)
```

Fig 5.2 Initializing the Database.

Coding Snaps:

```python
fin > web_server.py > register
# Define the route for registering a new website
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':

        domain_name = request.form['domain_name']
        html = request.form['html']

        # Create the HTML file for the website
        filename = f'{domain_name}.html'
        try:
            # Save the website to the database
            website = Website(domain_name=domain_name, html=filename)
            db.session.add(website)
            db.session.commit()
        except:
            print('DOMAIN NOT AVAILABLE')
            return render_template('error.html')

        try:
            with open(f'fin/webs/{filename}', 'w') as file:
                file.write(html)
        except:
            print('html file not added')

        # websites = Website.query.all()
        # print(websites)
        return redirect(url_for('register'))
    else:
        websites = Website.query.all()
```

Fig 6.1 Coding Part-1

```
fin > web_server.py > register
@app.route('/admin/error')
def error():
    return 'DOMAIN ALREADY TAKEN'

@app.route('/admin', methods=['GET', 'POST'])
def admin():
    if request.method == 'POST':
        return redirect(url_for('admin'))
    else:
        websites = Website.query.all()
        return render_template('admin.html', web_pages=websites)

@app.route('/admin/<path:path>', methods=['POST'])
def delete_page(path):
    web_pages = Website.query.all()
    for website in web_pages:
        if website.domain_name == path:
            web_pages.remove(website)
            os.remove(f'webs/{path}.html')
            break
    return redirect(url_for('home'))

# Define the route for serving static HTML pages
@app.route('/<path:path>', methods=['GET', 'POST'])
def serve_page(path):
    if request.method == 'POST':
        print('POST request done ')

    website = Website.query.filter_by(domain_name=f'{path}').first()
    if website is None:
```

Fig 6.2 Coding Part -2

```
if __name__ == '__main__':
    # Create the database tables if they don't exist
    with app.app_context():
        db.create_all()

    # Start the server on port 5000 and use threading to handle multiple requests
    thread = threading.Thread(target=run_simple, kwargs={'application': app, 'hostname': 'localhost', 'port': PORT, '
    thread.start()
```

Fig 6.3 Coding Part -3

Conclusion:

In conclusion, our project to develop a simple web server and client using Python has been successfully completed. We have implemented all the functional features as outlined in our proposal, including serving static HTML pages, handling GET and POST requests from the client, allowing the client to register new websites with the server, storing the registered websites in a database for future retrieval, and using threading to handle multiple requests simultaneously.

We used the Flask web development framework, Python threading library, and SQLAlchemy for database integration. Our development process followed an iterative approach, with regular testing and debugging to ensure the functionality and performance of the system.

Overall, this project has provided us with a valuable learning experience in Python web development and project management. We hope that this web server and client can be useful for anyone who needs a simple and efficient tool for serving HTML pages and managing website registration.

References:

1. Flask web development framework documentation: (https://flask.palletsprojects.com/en/2.1.x/)
2. Python threading documentation: (https://docs.python.org/3/library/threading.html)
3. SQLAlchemy documentation for database integration: (https://docs.sqlalchemy.org/en/14/)

4. Python requests library for making HTTP requests: (https://docs.python-requests.org/en/latest/)