Assignment 2

Task 1
Yes, you can definitely use Flask web framework to serve HTML pages on a server. Here are the steps for a basic implementation:

1. Install Flask using pip:

```python
pip install flask
```

2. Create a Python file called `app.py` and import Flask:

```python
from flask import Flask

app = Flask(__name__)
```

3. Define a route for the index page and return the HTML file:

```python
@app.route('/')
def index():
    return app.send_static_file('index.html')
```

4. Create a directory called `static` in the same directory as your `app.py` file, and put your HTML file(s) in it. In this example, the `index.html` file is in the `static` directory.

5. Run the Flask application:

```python
if __name__ == '__main__':
    app.run(debug=True)
```

This will start the Flask server on localhost at port 5000 (default). Visit `http://localhost:5000` in your browser, and your HTML page should be displayed.

Note: In this example, `send_static_file()` is used to send the HTML file from the `static` directory. You could also use `render_template()` function to create and render templates of HTML pages.

Task 2
To store the user registration details in IBM DB2, you can use the IBM_db API for Python. Here are the steps to do so:

1. Install the IBM_db API using pip:

```python
pip install ibm_db
```

2. Import the IBM_db module and establish a connection to your IBM DB2 database, replacing the placeholders with your database credentials:

```python
import ibm_db

dsn_hostname = "INSERT_HOSTNAME_HERE"
dsn_uid = "INSERT_USER_ID_HERE"
dsn_pwd = "INSERT_PASSWORD_HERE"
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "INSERT_DATABASE_NAME_HERE"
dsn_port = "INSERT_PORT_NUMBER_HERE"
dsn_protocol = "TCPIP"

dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};").format(dsn_driver, dsn_database, dsn_hostname, dsn_port, dsn_protocol,
dsn_uid, dsn_pwd)

conn = ibm_db.connect(dsn, "", "")
```

3. Create a table in the database to store the user registration details:

```python
table_name = "USER_REGISTRATION"

create_table_query = "CREATE TABLE " + table_name + " ("
create_table_query += "USER_ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY
(START WITH 1 INCREMENT BY 1), "
create_table_query += "NAME VARCHAR(255) NOT NULL, "
create_table_query += "EMAIL VARCHAR(255) NOT NULL, "
create_table_query += "PASSWORD VARCHAR(255) NOT NULL, "
create_table_query += "PRIMARY KEY(USER_ID))"

stmt = ibm_db.prepare(conn, create_table_query)
```

```
ibm_db.execute(stmt)
```

4. When a user registers, insert their details into the table:

```python
name = "INSERT_NAME_HERE"
email = "INSERT_EMAIL_HERE"
password = "INSERT_PASSWORD_HERE"

insert_query = "INSERT INTO " + table_name + " (NAME, EMAIL, PASSWORD) VALUES (?, ?, ?)"
stmt = ibm_db.prepare(conn, insert_query)
ibm_db.bind_param(stmt, 1, name)
ibm_db.bind_param(stmt, 2, email)
ibm_db.bind_param(stmt, 3, password)
ibm_db.execute(stmt)
```

5. To retrieve the user registration details, you can use a SELECT query:

```python
select_query = "SELECT * FROM " + table_name
stmt = ibm_db.exec_immediate(conn, select_query)
result = ibm_db.fetch_tuple(stmt)
while (result):
    print(result)
    result = ibm_db.fetch_tuple(stmt)
```

Remember to close the database connection when you're done:

```python
ibm_db.close(conn)
```

Task 3

To validate user login credentials, you can follow these steps:

1. Retrieve the user's inputted login credentials (e.g. username and password) from the login form.

2. Query the database or authentication system to check if the provided username exists and the associated password matches the provided password.

3. If the credentials are correct, redirect the user to the appropriate page or grant them access to the application.

4. If the credentials are incorrect, display an error message to the user and prompt them to try again.

Here's an example of a login validation function in Python using Flask:

```python
from flask import Flask, render_template, request, redirect, session

app = Flask(__name__)
app.secret_key = 'your_secret_key_here'

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # retrieve inputs
        username = request.form['username']
        password = request.form['password']

        # query the authentication system or database
        # to check if the provided username and password is valid
        if username == 'valid_username' and password == 'valid_password':
            # set session variable to keep user logged in
            session['logged_in'] = True
            return redirect('/dashboard')
        else:
            error = 'Invalid login credentials. Please try again.'
            return render_template('login.html', error=error)
    else: # GET request
        return render_template('login.html')
```

In this example, the `login()` function handles both GET and POST requests to `/login`. Upon receiving a POST request, the function retrieves the username and password inputs from the login form and checks if they match the valid credentials. If the credentials are correct, the function sets a `logged_in` session variable to True and redirects the user to the dashboard page. Otherwise, it displays an error message on the login page and prompts the user to try again. If the request to `/login` is a GET request, the function simply renders the login template without any error message.