Codes for the web app

1. **Web App Structure**
   a) **Static Intro Page:**

- Create a simple page (index.html) with a link or button that redirects users to the home page.

```
<!DOCTYPE html>
<html>
<head>
   <title>Intro Page</title>
</head>
<body>
   <h1>Welcome to the Web App</h1>
   <a href="/login">Go to Home</a>
</body>
</html>
```

   b) **Static Alternate Home Pages:**

- Create two simple home pages:

i. **Version A (home_A.html)**:

```
<!DOCTYPE html>
<html>
<head>
   <title>Home A</title>
</head>
<body>
   <h1>Welcome to Version A</h1>
   <button onclick="recordClick()">Click Me</button>
</body>
<script>
   function recordClick() {
      console.log('Button clicked on Version A');
   }
</script>
</html>
```

ii. **Version B (home_B.html)**:

```
<!DOCTYPE html>
<html>
<head>
<title>Home B</title>
</head>
<body>
<h1>Welcome to Version B</h1>
<button onclick="recordClick()">Click Me</button>
```

```
</body>
<script>
function recordClick() {
        console.log('Button clicked on Version B');
         }
</script>
</html>
```

2. **Implementing A/B Testing**
    a) **User Distribution and Stickiness:**

Use a backend logic to assign a user to a version consistently. Below is an example in Python using Flask:

```
from flask import Flask, render_template, request, redirect, session
import random
        app = Flask(__name__)
        app.secret_key = "your_secret_key"
@app.route('/')
        def index():
        return render_template('index.html')
@app.route('/login')
def login():
 # If user is already assigned, redirect to their version
if 'version' in session:
  return redirect(f"/home_{session['version']}")
 # Assign a random version (A or B)
         session['version'] = 'A' if random.random() < 0.5 else 'B'
        return redirect(f"/home_{session['version']}")
@app.route('/home_A')
        def home_a():
         return render_template('home_A.html')
@app.route('/home_B')
def home_b():
return render_template('home_B.html')
if __name__ == '__main__':
app.run(debug=True)
```

3. Metrics Capture

Button Click Tracking:

- **Frontend**: Capture button clicks using JavaScript and send them to the backend.

```
<script>
        function recordClick(version) {
        fetch('/track_click', {
                method: 'POST',
                headers: {
        'Content-Type': 'application/json'
                },
```

```
            body: JSON.stringify({ version: version })
        });
        }
</script>
```

Update buttons to include *onclick="recordClick('A')"* or *onclick="recordClick('B')"* based on the version.

Backend: Handle interaction tracking and store it in a database.

```
        @app.route('/track_click', methods=['POST'])
        def track_click():
        data = request.get_json()
        version = data.get('version')
# Save the version and timestamp to a database or file
        print(f"Button clicked on Version {version}")
        return 'OK', 200
```

4. Database Integration

- Use a database like SQLite to log user interactions:

```
CREATE TABLE interactions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    version TEXT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Insert interaction data from the backend:

```
import sqlite3
def log_interaction(version):
    conn = sqlite3.connect('data.db')
    cursor = conn.cursor()
    cursor.execute("INSERT INTO interactions (version) VALUES (?)", (version,))
    conn.commit()
    conn.close()
```

5. Testing

- Run the app and verify:
    - Users are consistently redirected to the same version.
    - Button clicks are logged correctly in the database or console.