Flask Redirects ,URL handling,and Error Pages

Introduction

- Flask provides easy ways to:
  - **Redirect users** from one route to another.
  - **Generate dynamic URLs** using `url_for()`.
- Both features help in **navigation**, **form submissions**, and **clean code management**.

✅ Core part of every web application.

What is a Redirect?

- A **redirect** automatically sends a user from one URL to another.
- Flask provides a helper function:

```
from flask import redirect
```

- Used after actions like:
  - Form submission
  - Login success/failure
  - Updating or deleting data

Simple Redirect Example

```
from flask import Flask, redirect

app = Flask(__name__)

@app.route('/')
def home():
    return "<h2>Welcome to Home Page</h2>"

@app.route('/start')
def start():
    return redirect('/')

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

Visiting `/start` → redirects automatically to `/`.

Redirect with `url_for()`

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

@app.route('/')
def home():
    return "Welcome to Home Page"

@app.route('/login')
def login():
    return "Please log in"

@app.route('/go_home')
def go_home():
    return redirect(url_for('home'))

@app.route('/go_login')
def go_login():
    return redirect(url_for('login'))
```

✅ `url_for('function_name')` builds URLs dynamically — safer than hardcoding.

Why Use `url_for()` Instead of Hardcoding

Method	Example	Drawback / Advantage
Hardcoded	`redirect('/login')`	Breaks if route changes
`url_for()`	`redirect(url_for('login'))`	Auto-updates when route changes ✅

✅ Always use `url_for()` for maintainability and dynamic routing.

Syntax of url_for() url_for(endpoint, **values)

endpoint → name of the function handling route

values → dynamic parts of the URL

Example:

url_for('profile', username='praveen')

Generates: /user/praveen

Example with Dynamic URLs

```
@app.route('/user/<username>')
def profile(username):
    return f"Hello, {username}!"

@app.route('/goto/<username>')
def goto(username):
    return redirect(url_for('profile', username=username))
```

✅ Visiting `/goto/John` → redirects to `/user/John`.

Flask URL Building

- `url_for()` can be used to build URLs for:
  - Static pages
  - Dynamic routes
  - Templates
  - Redirects

Example:

```
url_for('static', filename='style.css')
```

✅ Resolves path → `/static/style.css`

Redirect with Query Parameters

```
@app.route('/search')
def search():
    query = request.args.get('q')
    return f"Searching for {query}"

@app.route('/go_search')
def go_search():
    return redirect(url_for('search', q='Flask Tutorial'))
```

✅ Visiting `/go_search` → redirects to `/search?q=Flask+Tutorial`

Permanent Redirects

- Flask redirect defaults to **302 Temporary Redirect**.
- You can make it **permanent (301)**:

```
@app.route('/old-page')
def old_page():
    return redirect(url_for('new_page'), code=301)

@app.route('/new-page')
def new_page():
    return "This is the new page!"
```

✅ `code=301` tells browsers and search engines it's a permanent redirect.

Using `url_for()` in Templates

Template: `home.html`

```
<a href="{{ url_for('contact') }}">Contact Us</a>
```

Flask:

```
@app.route('/contact')
def contact():
    return "Contact Page"
```

✅ The link updates automatically if route names change.

Redirect Best Practices

✅ Use `url_for()` instead of hardcoding URLs.
✅ Return meaningful HTTP codes.
✅ Avoid infinite redirect loops.
✅ Always handle user input validation before redirecting.
✅ For secure redirects (external sites), validate domain first.

Flask Error Page Handling

- Flask allows developers to **handle errors gracefully** by showing **custom error pages** instead of default error messages.
- This improves:
  - ✅ User experience
  - ✅ Professional appearance
  - ✅ Application security

What is Error Handling?

- Error handling means catching errors during request processing and responding appropriately.
- Flask supports both:
  - **Default error pages**
  - **Custom error handlers**

Examples of common HTTP errors:

Error Code	Meaning
400	Bad Request
401	Unauthorized
403	Forbidden
404	Page Not Found
500	Internal Server Error

Default Flask Error Pages

By default, Flask shows technical error pages like:

```
404 Not Found
500 Internal Server Error
```

✅ Helpful during development
❌ Not user-friendly for production

Creating Custom Error Pages

You can use Flask’s decorator:

```
@app.errorhandler(error_code)
def function_name(error):
    return render_template('error_page.html'), error_code
```

✅ Used to define **custom responses** for different HTTP errors.

Example – Custom 404 Page

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return "Welcome to Home Page"

@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

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

✅ Handles 404 errors with a custom HTML template.

Template – 404.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Not Found</title>
  <style>
    body { text-align: center; font-family: Arial; margin-top: 50px; }
    h1 { color: #FF5733; }
  </style>
</head>
<body>
  <h1>404 - Page Not Found</h1>
  <p>The page you are looking for doesn't exist.</p>
  <a href="/">Go Back Home</a>
</body>
</html>
```

✅ This provides a friendly UI instead of plain text.

500 Error Page Template

```
<!DOCTYPE html>
<html>
<head>
  <title>Server Error</title>
</head>
<body>
  <h1>500 - Internal Server Error</h1>
  <p>Something went wrong on our end.</p>
  <p>Please try again later.</p>
</body>
</html>
```

Using `abort()` to Trigger Errors

You can manually raise an error using Flask's **abort()** method:

```
from flask import abort

@app.route('/admin')
def admin():
    abort(403)  # Forbidden access
```

Flask automatically displays the **custom 403.html** page if defined.

Returning JSON Error Responses (API)

For REST APIs, return JSON instead of HTML.

```
from flask import jsonify

@app.errorhandler(404)
def not_found(e):
    return jsonify(error="Resource not found"), 404

@app.errorhandler(500)
def internal_error(e):
    return jsonify(error="Server error occurred"), 500
```

✅ Useful for API-based Flask apps.

Best Practices for Error Handling

✅ Always define custom 404 and 500 pages.
✅ Hide technical error details from users.
✅ Log all exceptions internally.
✅ Return JSON for API routes.
✅ Keep error pages consistent with your website's design.
✅ Use `abort()` for manual error triggering.