REST Interfaces and Flask

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand the concept of Representational State Transfer (REST) and the Flask module
2. Demonstrate your knowledge of REST interfaces in a body of work
3. Demonstrate your knowledge of Flask in a body of work

RESTful Interfaces

RESTful Interfaces (1)

- Abbreviation for: Representational State Transfer
  - also commonly referred to as a RESTful API
- An architectural style for providing standards between computer systems and the web
- Characterised by being **stateless**
- Concerned with the separation of client and server
  - both the client and server can be developed independently
  - changes on either platform will not affect each other
- REST works with resources and not commands
  - a resource is an object or document
  - no reliance upon interfaces

RESTful Interfaces (2)

Communicating between a Client and a Server i

- **Client** sends a request to receive or modify a resource
- **Server** responds to the request made by a *client*
- A client's request is made up of the following:
  - a HTTP verb
  - a header and accept parameter
  - path to the resource
  - an optional message body

RESTful Interfaces (3)

Communicating between a Client and a Server ii

HTTP Verbs

- Four basic HTTP verbs:
  1. GET
  2. POST
  3. PUT
  4. DELETE

Headers and Accept Parameters

- Sends information about the content to be received from the server
- Stipulated inside the `Accept` field of the header
- Defined by a MIME Type:
  - application: `application/json`, `application/pdf`, `application/xml`
  - audio: `audio/mpeg`, `audio/wav`
  - image: `image/png`, `image/jpeg`, `image/gif`
  - text: `text/plain`, `text/html`
  - video: `video/ogg`, `video/mp4`

RESTful Interfaces (4)

Communicating between a Client and a Server iii

Paths

- Requests must contain a path to the resource
- Paths should be designed to ensure the client knows what is happening
  - e.g. `/modules/5062/`
- Paths should always contain necessary information to locate the resource
  - a degree of specificity if required
- `POST` request to `/modules` may not need an identifier
  - the `id` is generated on the server-side

RESTful Interfaces (5)

Sending Responses from the Server

- Server responses must also include a MIME type
  - stored in the `content-type` of the header
- Response data must match what is stored in the `Accept` field of the client's request header
- Status codes are sent as part of the response
  - indicates the success of an operation
  - popular status codes are shown in the table below

Status Code	Definition
200 (OK)	Successful operation
201 (CREATED)	Successful creation of an item
204 (NO CONTENT)	Successful operation but no data returned
400 (BAD REQUEST)	Failed operation as bad request was sent
403 (FORBIDDEN)	Failed operation as the client does not have permission
404 (NOT FOUND)	Failed operation as the resource was not found
500 (INTERNAL SERVER ERROR)	Failed operation as an unexpected error occurred

RESTful Interfaces (6)

REST End-Points

- REST APIs expose a set of public URLs to access resources of a web service
  - these are known as an **endpoint**
- Examples of REST endpoints:

HTTP Verb	API Endpoint	Description
`GET`	/modules	Get a list of modules
`GET`	/modules/	Get a single module
`POST`	/modules	Creates a new module
`PUT`	/modules/	Updates a module
`DELETE`	/modules/	Deletes a module

Flask

Flask (1)

- Flask is a third-party module that enables you to build web applications
- Based upon the WSGI toolkit and Jinja2 template engine
- Often referred to as a micro-framework
  - makes use of extensions to add extended functionality
  - e.g. database handling and management, form validation etc.

Flask (2)

Flask Extensions

- Extensions can provide additional functionality
  - i.e. database handling, form validation and session management
- Common extensions are:
  - Mail
  - SQLAlchemy
  - WTF
- Installation of an extension is handled via the Python package manager

```
$ python3 -m pip install Flask-Mail Flask-SQLAlchemy Flask-WTF
```

Flask (3)

Flask and Python

- Install Flask via the Python package manager
  - **Note**: You should be using virtual environments when working on your Python projects

```
$ python3 -m pip install Flask
```

- Flask can be imported into the module using the `from` and `import` keywords
  - i.e. `from flask import Flask`

Flask (4)

Creating a Flask Application

- A Flask application is created using the `Flask` class
  - i.e. `app = Flask(__name__)`
- Create a function for a simple web-page
  - i.e. `def hello()`
- Use a decorator on the function to denote the path/URL of the page
  - i.e. `@app.route('/')`
- Run the application on a local development server
  - i.e. `app.run()`
  - contains four optional parameters: `host`, `port`, `debug`, and `options`

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return "Hello 5062CEM"

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

Flask (5)

Creating a Flask Application with a Template

- Flask uses Jinja2 for templating
  - a directory called `templates` will store your Jinja templates
- Create a file named `index.html` with the HTML source of your template
  - this should be located in the `templates` directory
- Jinja uses two curly braces (`{{` and `}}`) to dynamically import data from the backend
  - between these curly braces will be a variable name
    - i.e. `{{ module }}`
  - this will have been defined as a parameter in the `render_template` function in the Python source-code

Python Flask Application

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html", module="5062CEM")

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

HTML Template

```html
<html>
<head>
    <title>{{ module }}</title>
    <style>
        body {
            background-color: magenta;
        }
    </style>
</head>
<body>
Hello {{ module }}!
</body>
```

Flask (6)

Creating a Flask Application with a Form

Python Flask Application

```
from flask import Flask, render_template, request
from flask_wtf import Form
from wtforms import StringField, IntegerField, SubmitField
app = Flask(__name__)
app.secret_key = 'potato'

class SampleForm(Form):
    moduleCode = IntegerField("Code")
    moduleName = StringField("Name")
    school = StringField("School")
    submit = SubmitField("Send")

@app.route('/', methods=['GET', 'POST'])
```

HTML Template

```
<html>
<head>
    <title>{{ title }}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bo
</head>
<body>
<form action="{{ url_for('index') }}" method="POST">
    {{ form.hidden_tag }}
    <div class="card w-50 m-2">
        <h5 class="card-header">New Module Form</h5>
        <div class="card-body">
```

Flask (7)

Creating a RESTful Interface within a Flask Application

```
from flask import Flask, jsonify, make_response
app = Flask(__name__)

modules = [
    {"code": "5062CEM", "title": "Programming and Algorithms 2", "leader": "Dr Ian Cornelius"},
    {"code": "4061CEM", "title": "Programming and Algorithms 1", "leader": "Dr Ian Cornelius"}
]

@app.route("/modules", methods=['GET'])
def list_modules():
    response = make_response(jsonify(modules), 200)
    response.headers['Content-Type'] = "application/json"
    return response

@app.route("/modules/<id>", methods=['GET'])
```

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk