Introduction

Welcome to Module 4! This week, we will learn about creating middleware. Middleware is an application that provides business logic. Business logic is a separate layer or application than the web browser used to access it. The web browser will submit HTTP requests and the middleware will respond with an HTTP request.

Topics and Learning Objectives

Topics

- HTTP
  - GET, POST, PUT/PATCH, DELTE, HEAD
  - Response codes: 1xx, 2xx, 3xx, 4xx, 5xx
- RESTful architectures and CRUD
- Web application URLs and routing
- The request-response cycle
- RESTful APIs
  - Exposing an API endpoint

Learning Objectives

By the end of the module, you should be able to:

1. Create a web form that makes an HTTP POST request to the application.
2. Respond to an HTTP POST with success/failure messaging.
3. Utilize the 6 core HTTP verbs (GET, POST, PUT/PATCH, DELETE, HEAD).
4. Apply HTTP verbs to map to web applications (CREATE, RETRIEVE, UPDATE, DELETE).

Readings & Resources

Reading

Required

- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media, Inc.
  - Chapter 2: Basic Application Structure
  - Chapter 14: Application Programming Interfaces

- Fielding, R. T. (2000). Chapter 5 Representational State Transfer (REST).
- Video: RailsConf 2017: In relentless pursuit of REST by Derek Prior [36:18]

HTTP

- HTTP (Hypertext Transfer Protocol) is an application layer protocol in the Internet protocol suite.
- HTTP is the protocol for the World Wide Web, used to retrieve html documents and their supporting assets (pictures, streams and so on).
- HTTP has gone through three version. HTTP/3 is the latest version, has a lower latency and is faster than HTTP/1.1 and HTTP/2.
- HTTP/3 uses QUIC instead of TCP for the underlying transport protocol. Note HTTP/1.1 and HTTP/2 use TCP/IP as the underlying transport protocol.
- HTTP functions as a request-response protocol in the client-server model.
- A web browser is a client that accesses a server namely the web server. The client submits an HTTP request message to the server. The server provides a response in the form of resources such at HTML files. The response contains completion status information about the request .
- A web browser is an example of a UA (User Agent). Other user agents may be web crawlers, mobile apps or software that consumes or displays web content.

HTTP Data Exchange

- HTTP is a stateless application-level protocol and it requires a reliable network connection to exchange data between client and server.
- In HTTP implementations, TCP/IP connections are used using well knows ports (default is 80 for unencrypted and 443 for encrypted).
- FireFox Development Tools may be used to view the request/response of an HTTP request.

HTTP Authentication

- HTTP provides multiple authentication schemes such as basic access authentication and digest access authentication which operate via a challenge-response mechanism whereby the server identifies and issues a challenge before sending the requested content.

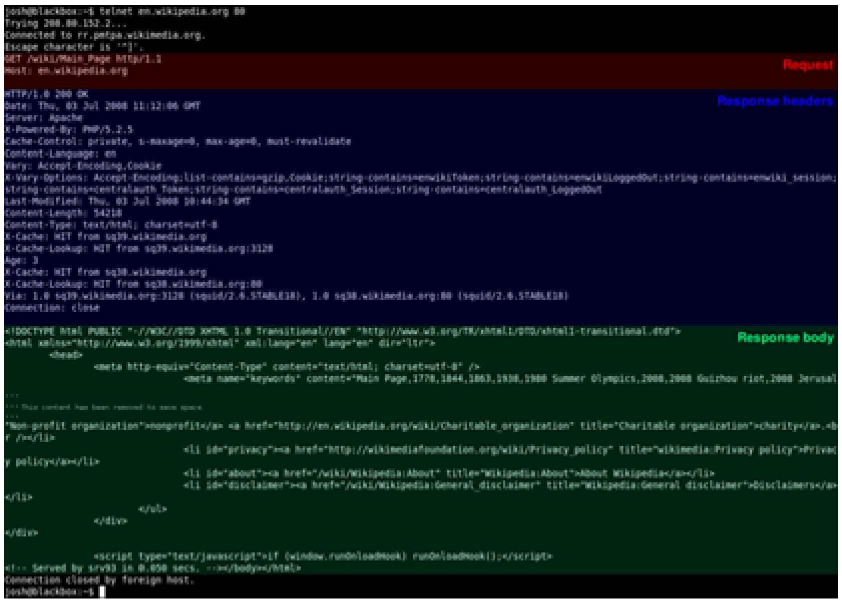
- HTTP provides a general framework for access control and authentication. This framework belongs to HTTP protocol and it is managed by client and server and not by web applications that require a web application session.
- Authentication is configured rather than programmed into a web application.

Request Methods

- HTTP defines methods (referred to as verbs) to indicate the desired action to be performed on the identified resource.
- HTTP/1 defined GET, HEAD, POST.
- HTTP/1.1 defined PUT, DELETE, CONNECT, OPTIONS and TRACE.
- Method names are case sensitive. This is in contrast to HTTP header field names which are case-insensitive.

HTTP Request

- An HTTP/1.1 request made using telnet.
- The request message, response header section and response body are highlighted in the image below.



```
josh@blackbox:~$ telnet en.wikipedia.org 80
Trying 208.80.152.2...
Connected to rr.pmtpa.wikimedia.org.
Escape character is '^]'.
GET /wiki/Main_Page http/1.1
Host: en.wikipedia.org

HTTP/1.0 200 OK
Date: Thu, 03 Jul 2008 11:12:06 GMT
Server: Apache
X-Powered-By: PHP/5.2.5
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding,Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-contains=enwikiToken;string-contains=enwikiLoggedOut;string-contains=enwiki_session;
string-contains=centralauth_Token;string-contains=centralauth_Session;string-contains=centralauth_LoggedOut
Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
Content-Length: 54218
Content-Type: text/html; charset=utf-8
X-Cache: HIT from sq39.wikimedia.org
X-Cache-Lookup: HIT from sq39.wikimedia.org:3128
Age: 3
X-Cache: HIT from sq38.wikimedia.org
X-Cache-Lookup: HIT from sq38.wikimedia.org:80
Via: 1.0 sq39.wikimedia.org:3128 (squid/2.6.STABLE18), 1.0 sq38.wikimedia.org:80 (squid/2.6.STABLE18)
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
                                <meta name="keywords" content="Main Page,1778,1844,1863,1938,1980 Summer Olympics,2008,2008 Guizhou riot,2008 Jerusal
...
... This content has been removed to save space
...
"Non-profit organization">nonprofit</a> <a href="http://en.wikipedia.org/wiki/Charitable_organization" title="Charitable organization">charity</a>.<b
r /></li>
                                <li id="privacy"><a href="http://wikimediafoundation.org/wiki/Privacy_policy" title="wikimedia:Privacy policy">Privac
y policy</a></li>
                                <li id="about"><a href="/wiki/Wikipedia:About" title="Wikipedia:About">About Wikipedia</a></li>
                                <li id="disclaimer"><a href="/wiki/Wikipedia:General_disclaimer" title="Wikipedia:General disclaimer">Disclaimers</a>
</li>
                        </ul>
                </div>
</div>

                <script type="text/javascript">if (window.runOnloadHook) runOnloadHook();</script>
<!-- Served by srv93 in 0.050 secs. --></body></html>
Connection closed by foreign host.
josh@blackbox:~$
```

Figure 4.1. HTTP Request Example

Source: Mihal Miu

Request Methods

Click on each of the request methods below to reveal more about them:

Click-n-reveal: **GET**

The GET method requests that the target resource transfer a representation of its state. GET requests should only retrieve data and should have no other effect.

Click-n-reveal: **HEAD**

The HEAD method requests that the target resource transfer a representation of its state but without the representation data enclosed in the response body. This method is useful for determining if a resource is available though the status codes and finding its size.

Click-n-reveal: **POST**

The POST method requests that the target resource process the representation enclosed in the request according to the semantics of the target resource.

Click-n-reveal: **PUT**

The PUT method requests that the target resource create or update its state with the state defined by the representation enclosed in the request.

Click-n-reveal: **DELETE**

The DELETE method request that the target resource delete its state

Click-n-reveal: **CONNECT**

The CONNECT method requests that the intermediary establish a TCP/IP tunnel to the origin server identified by the request target. It is often used to secure connections with TLS through proxies.

Click-n-reveal: **OPTIONS**

The OPTIONS method requests that the target resource transfer the HTTP methods that it supports.

Click-n-reveal: **TRACE**

The TRACE method requests that the target resource transfer the received request in the response body. This way a client can see what changes have been made by intermediaries.

Click-n-reveal: **PATCH**

The PATCH method requests that the target resource modify its state according to the partial update defined in the representation enclosed request. This saves bandwidth by uploading a part of a file without transferring the entire file.

Safe Methods

- A request method is safe if a request with that method has no intended effect on the server.
- Safe methods are intended to be read-only. This means that these methods do not change any resources on the web server.
- The methods GET, HEAD, OPTIONS and TRACE are defined to be safe.
- The methods POST, PUT, DELETE, CONNECT and PATCH are not safe.
- Unsafe methods modify the state of the server.

Response Status Codes

- The first line of the HTTP response is called the status line and includes a numeric status code and a textual reason phrase.
- The response status code is a three-digit number representing the result of the servers’ attempt to respond to a client request.

- The first digit of the status code defines its class.
    - 1XX (information) – The request was received, continue processing
    - 2XX (success) – The request was successfully received and accepted
    - 3XX (redirect) – Further action needs to be taken in order to complete request
    - 4XX (client error) – The request contains bad syntax or endpoint
    - 5XX (server error) – The server failed to fulfill a valid request

HTTP Client Request Example

HTTP Request

GET / HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

Accept-Language: en-GB,en;q=0.5

Accept-Encoding: gzip, deflate, br

Connection: keep-alive

HTTP Server Response

HTTP/1.1 200 OK

Date: Mon, 23 May 2005 22:38:34 GMT

Content-Type: text/html; charset=UTF-8

Content-Length: 155

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

ETag: "3f80f-1b6-3e1cb03b"

Accept-Ranges: bytes

Connection: close

```
<html>
 <head>
  <title>An Example Page</title>
 </head>
 <body>
  <p>Hello World, this is a very simple HTML document.</p>
 </body>
</html>
```

REST

- REST (Representational State Transfer) is a software architectural style that describes a uniform interface between decoupled components in the Internet is a client-server architecture.
- REST defines four interface constraints
  1. Identification of resources
  2. Manipulation of resources
  3. Self-descriptive messages
  4. Hypermedia as the engine of application state

APIs

- REST has been employed by developers and is a widely accepted set of guidelines for creating stateless, reliable web APIs.
- A web API that obeys the REST constraints is informally described as RESTful.
- RESTful web APIs are typically loosely based HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.
- A web API is an application programming interface for either a web server or web browser. It is a web development concept, and does not include web server or browser implementation details such as SAPI.
- A server-side API is a programming interface consisting of one or more publicly exposed endpoints to a defined request-response message system, expressed in JSON or XML. The request-response is transmitted using HTTP.

- Endpoints are important aspects of interacting with server-side web APIs. Endpoints specify where resource may be accessed by users. Resources are accessed using a URI over HTTP.
- Endpoints need to be static, otherwise correct functionality can not be guaranteed.

Flask and Endpoints

```
import json
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
    return json.dumps({'name': 'alice',
                       'email': 'alice@torontomu.cc'})
app.run()
```

Flask App Routing

- App Routing is the process of mapping URLs to specific functions that handle logic for that URL.
- @app.route is an annotation that defines a URL for the associated method. This means that the method may be executed simply by calling the URL from a web browser.
- Annotations are used to provide supplemental information about a program. Annotations start with “@” and help associate metadata (information) to the program elements.
    - In the previous slide the metadata is the URL and the program element is the Python method.

Flask-Restful with Dynamic URLs

```
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.route('/', methods = ['GET', 'POST'])
def home():
    if(request.method == 'GET'):
        data = "hello world"
        return jsonify({'data': data})

@app.route('/home/<int:num>', methods = ['GET'])
```

```
def disp(num):
    return jsonify({'data': num**2})

app.run()
```

- On the above Python example there is a Python method named “home”. This method may be called by either a GET or POST request from a web browser. The URL would be http://localhost:5000/
- On the above Python example there is a Python method named “disp”. The method “disp” takes a parameter. This parameter may be submitted by incorporating it into the URL. In this example we can say that this URL is dynamic since the parameter may change.
- Possible URLs may be:
  - http://localhost:5000/home/10
  - http://localhost:5000/home/20000
  - Above URLs may be submitted using a GET request.

Sending Info to REST Endpoints

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/user', methods = [ 'POST'])
def user():
          user_data = request.get_json()
          print('user_data: ', user_data)

          res = {'status': 'ok'}
          return jsonify(res)

app.run()
```

- In the above Python example, we are sending information to an endpoint. In our case we are sending JSON from the client to the Python method named “user”.

```
curl --header "Content-Type: application/json"  --request POST  --
data '{"username":"tutorials","password":"secret"}'
http://localhost:5000/user
```

- RESTer may be used to submit JSON request instead of using curl. RESTer is a FireFox plugin.
- See GitHub’s RESTer page
- See Mozilla’s RESTer Add-ons page

Video

Please watch the following video, Web Forms. This video demonstrates how middleware will receive GET and POST requests from a web browser.

Web Forms
TMU Video

Summary

Middleware enables an application to provide business logic that is scalable and accessible over the Internet. Middleware responds to GET and POST requests submitted by web pages running inside a web browser. Middleware allows you to develop your business logic independently of the user interfaces or web pages. Furthermore, middleware can be centralized and operate on servers that are optimized for multi-core computing and very fast memory access.

Assessments

Each week, you'll find here reminders of assignments or labs that you should be working on. For Week 4:

- Lab 4: Data Submission (5% of the final grade) is due in 7 days, i.e., the end of day, Friday, of Week 4.

Reminder

You have a weekly Zoom session. The date/timing for your Zoom session can be found in the Course Schedule in the Course Outline. The link to the Zoom session will be provided separately to you by your instructor. The Zoom session will consist of 20–30 minutes of lecture, followed by questions and answers (Q&A), followed by a lab period. You may ask questions about the previous or current week's content or labs.

Please bring any questions to the weekly Zoom session. However, if you have any questions during the week outside of the Zoom session, you may post them in the Discussion Board. Your instructor may respond in the Discussion board or during the weekly Zoom session.

Click-n-reveal: **Where is the Discussion board?**

Click on the "Communication" area at the top main course menu, and select Discussions from the dropdown menu.

References

Wikipedia. (n.d.). HTTP Protocol.