COMP519 Web Programming
Lecture 28: REST (Part 2)
Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Example (2)

- The web service we want to provide maintains information on Computer Science students at the University of Liverpool
- The web service should allow us to
  - Retrieve information on all students

    ```
    GET https://api.liv.ac.uk/v1/students
    ```
  - Retrieve information on a specific student (via student id)

    ```
    GET https://api.liv.ac.uk/v1/students/201912345
    ```
  - Add information on a new student

    ```
    POST https://api.liv.ac.uk/v1/students
    + information on a new student in the request body
    ```
  - Modify information on an existing student

    ```
    PATCH https://api.liv.ac.uk/v1/students
    + partial information on a student in the request body
    ```
  - Delete (information on) a student

    ```
    DELETE https://api.liv.ac.uk/v1/students/201912345
    ```

URIs and PHP Scripts

- We need a server-side program or programs that deal with all these HTTP requests
- Problem:
  None of the HTTP requests explicitly refers to a server-side program

```
POST https://api.liv.ac.uk/v1/students
+ information on a new student in the request body
```

- Contrast:

```
<form action="process.php" method="post">
  <label>Address: <input type="text" name="addr"></label>
  <input type="hidden" name="item"
         value="<?php echo $_REQUEST['item'] ?>">
</form>
```

  Resulting in a HTTP request

```
POST https://student.csc.liv.ac.uk/~sqxyz/process.php
addr=...&item=...
```

- How do we solve this problem?

URIs and PHP Scripts

Option 1: Multitude of scripts

- For URIs where the last URI component maps to a directory, web servers will typically look for a file `index.html` or `index.php` in that directory
- We could create a directory hierarchy corresponding to the resource hierarchy
- In each directory we place a PHP script `index.php` that deals with requests relating to the corresponding resource
- The scripts would all be identical and use `$_SERVER['SCRIPT_URI']` to determine the corresponding resource

$\rightsquigarrow$ thousands of identical scripts
$\rightsquigarrow$ directory structure changes as resources are added and deleted
$\rightsquigarrow$ very difficult to maintain all the scripts

URIs and PHP Scripts

Option 2: One script plus URI rewriting to trailing pathnames

- We have just one PHP script with an arbitrary name, for example, `REST.php`, in the directory corresponding to the base URL of the web service
- Using the rewriting facility of the web server, we rewrite URIs so that a resource becomes a trailing pathname

```
RewriteRule ^\/?(.*)$ REST.php/$1
```

- The script can use `$_SERVER['PATH_INFO']` to determine the trailing pathname and thereby the resource that a request refers to

⇝ Rewrite rules are not always enabled,
Trailing pathnames are not always enabled,
might require administrator rights for the web server

URIs and PHP Scripts

Option 3: One script plus URI rewriting to query

- We have just one PHP script with an arbitrary name, for example, `REST.php`, in the directory corresponding to the base URL of the web service
- Using the rewriting facility of the web server, we rewrite URIs so that a resource becomes (part of) a query

```
RewriteRule ^\/?(.*)$ REST.php?resource=$1 [L,QSA]
```

The option `QSA` ensures that if a query is already part of the URI, it will be preserved

- The script can use `$_REQUEST['resource']` to determine the trailing pathname and thereby the resource that a request refers to

⇝ Rewrite rules are not always enabled,
might require administrator rights for the web server

URIs and PHP Scripts

Option 4: One script plus URI in query

- We have just one PHP script with an arbitrary name, for example, `REST.php`, in the directory corresponding to the base URL of the web service
- The resource must be passed to the script via a query parameter, for example, `resource`
- The script can use `$_REQUEST['resource']` to determine the trailing pathname and thereby the resource that a request refers to

⇝ Instead of

```
GET http://api.liv.ac.uk/v1/students
```

users of the web service need to use

```
GET http://api.liv.ac.uk/v1/REST.php?resource=students
```

⇝ Not compliant with the expected format of URIs

Web server configuration and `.htaccess` Files

- Rewrite rules for URLs are part of the web server configuration
- `.htaccess` files provide a way to make configuration changes to a web server on a per-directory basis
- `.htaccess` files contain configuration directives
  - Rewrite rules are one of many kinds of configuration directives
- The directives apply to directory containing the `.htaccess` file and all its subdirectories
- Directives are applied in the order that they are found

On our web server `.htaccess` files need to be world-readable

URL Rewrite Rules

The general format of rewrite rules is

```
RewriteRule pattern substitution [flags]
```

- *pattern* is a Perl compatible regular expression
- *pattern* may contain capture groups
- *pattern* is matched relative to the current directory
  not the original, full URI in the request
- *substitution* is a string that replaces the original URL-path that was matched by *pattern*
  not the part of original URL-path that it matched
- *substitution* may contain back-references ($*N*) to *pattern*
- *flags* is a comma-separated list of flags, modifying the behaviour of the rewrite rule

URL Rewrite Rules: Patterns

- *pattern* is matched relative to the current directory
  not the original requested URI
  ⇝ the directory path where the rule is defined is stripped from the currently mapped filesystem path before matching
  (up to and including a trailing slash)

```
# ~sgxyz/public_html/a/.htaccess
RewriteRule b/file1.html file2.htm
```

- `http://student.csc.liv.ac.uk/~sgxyz/A/file1.html`
  - Mapped path is `~sgxyz/public_html/A/file1.html`
  - URL is not affected by the rewrite rule as `~sgxyz/public_html/A/` is a not `~sgxyz/public_html/a/` nor a subdirectory of it
- `http://student.csc.liv.ac.uk/~sgxyz/a/file1.html`
  - Mapped path is `~sgxyz/public_html/a/file1.html`
  - Prefix up to directory `a/` will be stripped `file1.html`
  - Pattern `b/file1.html` does not match `file1.html`, the rule has no effect

URL Rewrite Rules: Patterns

- *pattern* is matched relative to the current directory
  not the original requested URI
  ⇝ the directory path where the rule is defined is stripped from the currently mapped filesystem path before matching
  (up to and including a trailing slash)

```
# ~sgxyz/public_html/a/.htaccess
RewriteRule b/file1.html file2.htm
```

- `http://student.csc.liv.ac.uk/~sgxyz/a/b/file1.html`
  - Mapped path is `~sgxyz/public_html/a/b/file1.html`
  - Prefix up to directory `a/` will be stripped
  - Pattern `b/file1.html`
    does match `b/file1.html`
  - Mapped path becomes `~sgxyz/public_html/a/file2.htm`

URL Rewrite Rules: Substitutions

- *substitution* is a string that replaces the original URL-path that was matched by *pattern*
  not the part of original URL-path that it matched
- *substitution* may contain back-references ($*N*)

```
# ~sgxyz/public_html/a/.htaccess
RewriteRule  b  c
```

- `http://student.csc.liv.ac.uk/~sgxyz/a/b/file1.html`
  - Mapped path is `~sgxyz/public_html/a/b/file1.html`
  - Prefix up to directory `a/` will be stripped
  - Pattern `b`
    does match `b/file1.html`
  - Mapped path becomes `~sgxyz/public_html/a/c`
    ⇝ Not only `b` is replaced by `c`
    ⇝ *pattern* does not identify what is replaced
    ⇝ it is safer to write patterns that match the whole remaining path

URL Rewrite Rules: Substitutions

- *substitution* is a string that replaces the original URL-path that was matched by *pattern*
  not the part of original URL-path that it matched
- *substitution* may contain back-references ($*N*)

```
# ~sgxyz/public_html/a/.htaccess
RewriteRule ^b(\/?.*)$ c$1
```

- `http://student.csc.liv.ac.uk/~sgxyz/a/b/file1.html`
  - Mapped path is `~sgxyz/public_html/a/b/file1.html`
  - Prefix up to directory `a/` will be stripped
  - Pattern `^b(\/?.*)$`

    does match `b/file1.html`
  - Mapped path becomes `~sgxyz/public_html/a/c/file1.html`

URL Rewrite Rules: Substitutions

- We can use rewriting to turn (part of) a path into a query

```
# ~sgxyz/public_html/a/.htaccess
RewriteRule ^b\/?(.*)$ file2.php?path=$1
```

- `http://student.csc.liv.ac.uk/~sgxyz/a/b/c/file1.html`
  - Mapped path is `~sgxyz/public_html/a/b/c/file1.html`
  - Prefix up to directory a/ will be stripped
  - Pattern `^b\/?(.*)$`

    does match `b/c/file1.html`
  - Mapped path becomes

    `~sgxyz/public_html/a/file2.php?path=c\/file1.html`

URL Rewrite Rules: Flags

- Flags modify the behaviour of rewrite rules

Flag	Function
`B`	Escape/encode non-alphanumeric characters in backreferences before applying the transformation
`BNP`	If backreferences are being escaped/encoded, spaces should be escaped to %20 instead of +
`QSA`	Appends any query string from the original request URL to any query string created in the rewrite target
`QSD`	Discard any query string attached to the incoming URI
`NC`	Makes the pattern comparison case-insensitive
`L`	Stop the rewriting process immediately and do not apply any more rules
`R`	Forces an external redirect, optionally with the specified HTTP status code
`PT`	Forces the resulting URI to be passed back to the URL mapping engine for processing

URL Rewrite Rules: Flags

- Flags modify the behaviour of rewrite rules

Flag	Function
B	Escape/encode non-alphanumeric characters in backreferences before applying the transformation
QSA	Appends any query string from the original request URL to any query string created in the rewrite target

```
# ~sgxyz/public_html/a/.htaccess
RewriteRule ^b\/?(.*)\/\w+\.php$ file2.php?path=$1 [B,QSA]
```

- `http://student.csc.liv.ac.uk/~sgxyz/a/b/c/d/file1.php?sort=true`
  - Mapped path is `~sgxyz/public_html/a/b/c/d/file1.php?sort=true`
  - Prefix up to directory a/ will be stripped
  - Pattern `^b\/?(.*)\/\w+\.php$`

    does match `b/c/file1.php`
  - Mapped path becomes

    `~sgxyz/public_html/a/file2.php?path=c\/d&sort=true`

Example (3)

- The web service we want to provide maintains information on Computer Science students at the University of Liverpool
- The web service should allow us to
  - Add information on a new student

    ```
    POST https://api.liv.ac.uk/v1/students
    + information on a new student in the request body
    ```

  - ...
- We will use Option 3 to deal with HTTP request to our web service
- We will rewrite HTTP requests so that

  ```
  method https://api.liv.ac.uk/v1/resource?query
  ```

  becomes

  ```
  method https://api.liv.ac.uk/v1/REST.php?resource=resource
      ↪&query
  ```

- The script `REST.php` can use `$_REQUEST['resource']` to determine the resource that a request refers to

Revision and Further Reading

- Read
  - Apache HTTP Server Tutorial: `.htaccess` files
    `https://httpd.apache.org/docs/2.4/howto/htaccess.html`
  - Apache Module `mod_rewrite`
    `https://httpd.apache.org/docs/2.4/mod/mod_rewrite.html`
  - RewriteRule Flags
    `http://httpd.apache.org/docs/current/rewrite/flags.html`

  of The Apache Software Foundation: Apache HTTP Server Version 2.4 Documentation. The Apache Software Foundation, 2020. `http://httpd.apache.org/docs/current/` [accessed 20 Dec 2020]