Client-Server Interaction

Language Technology and Web Applications

Jannis Vamvas [1]

Department of Computational Linguistics
University of Zurich

October 9, 2024

University of Zurich[UZH]

[1] Parts of this lecture are based on the MDN web docs, which are licensed under CC-BY-SA 2.5.

Milestone I

16th October 2024, 10:15 am

- Briefly present your concept to the class
- One presenter per team, one slide
- 2 minutes for presenting the slide, 2 minutes for questions

Where we are

Past weeks: **Static** web application

- Document structure and styles
- Manipulation of document using JavaScript

Now: **Dynamic** web application

- Sending user input to the server
- Retrieving data from the server
- Processing and storing data on the server

Learning Goals for this Week

- You can write a simple HTML form
- You can explain the basic elements of an HTTP request
- You can perform asynchronous requests in JavaScript (with the help of documentation)

Topics

1. Web Forms
2. HTTP
3. Ajax
4. Backend Programming

Topics

1. Web Forms
2. HTTP
3. Ajax
4. Backend Programming

Elements of an HTML Form

- Fields
- A submit button
- Metadata on what should happen with the form data

Example

```
<form action="/my-handling-form-page" method="post">

    <label for="name">Name:</label>
    <input type="text" id="name" name="username">

    <button type="submit">Send!</button>

</form>
```

Name: Send!

Types of Text Inputs

```
<input type="text">
<input type="number">
<input type="password">
<input type="email">
<input type="tel">
<input type="search">
<input type="url">
```

Multi-Line Text Input

```
<textarea name="message" rows="3" cols="30">
    Write something here
</textarea>
```

Write something here

Check Boxes

```
<p>Choose your monster's features:</p>
<div>
  <input type="checkbox" id="scales" name="scales" checked>
  <label for="scales">Scales</label>
</div>
<div>
  <input type="checkbox" id="horns" name="horns">
  <label for="horns">Horns</label>
</div>
```

Choose your monster's features:

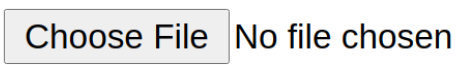
- [x] Scales
- [ ] Horns

Boolean Attributes

- `checked` (for checkboxes)
- `required`
- `readonly`
- `disabled`
- `hidden`

Uploading files

```
<form method="post" enctype="multipart/form-data">
    <input type="file" name="profile-pic">
</form>
```



Practical Exercise

Analyze the form on the federal health insurance premium calculator:

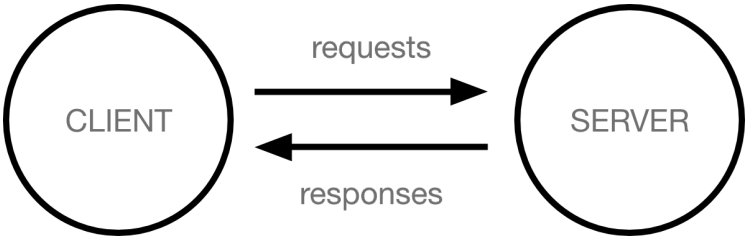
`https://priminfo.ch`

1. What HTML elements do you recognize from the previous slides?
2. Can you manipulate the elements using the browser's developer tools (e.g., enable a disabled field)?

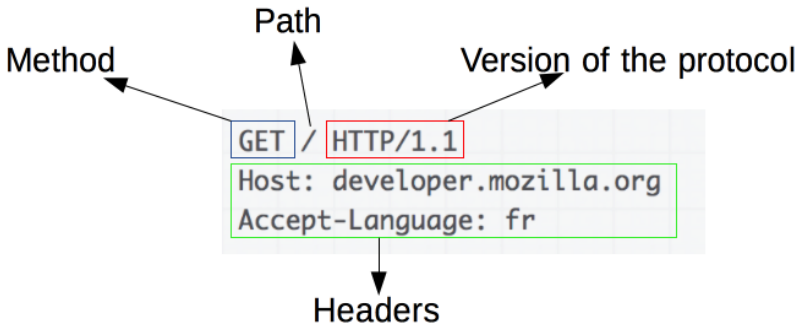
Topics

1. Web Forms
2. HTTP
3. Ajax
4. Backend Programming

Clients and Servers



Metadata of an HTTP Request



Examples of a Host

- example.com
- subdomain.example.com

- example.com:80
- 172.23.66.232:53402

- localhost
- 127.0.0.1

Examples of a Path

- /
- /index.html
- /shop/

“Query Parameters”:

- /shop/?key1=value1
- /shop/?key1=value1&key2=value2

Encoding of Query Parameters

`{'q': 'hello'}` → `q=hello`

`{'q': 'foo bar'}` → `q=foo%20bar` or `q=foo+bar`

`{'q': 'grüezi'}` → `q=gr%C3%BCezi`

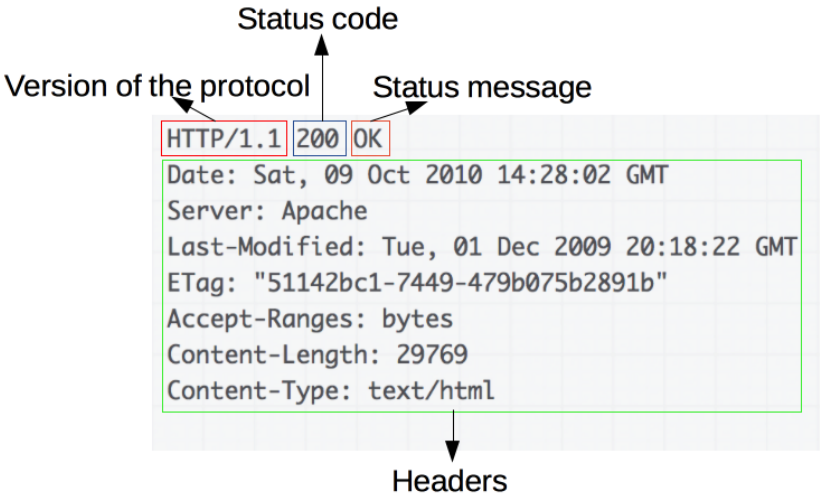
Examples of Request Headers

- `Accept-Language: de-CH`
  Informs the server about the human language the server is expected to send back.

- `Referer: https://www.google.com/`
  The address of the previous web page

- `User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X x.y; rv:42.0) Gecko/20100101 Firefox/42.0`
  Application type, operating system, software vendor or software version of the client

Metadata of an HTTP Response



Examples of HTTP Status Codes

- `200 OK`
- `302 Found` (= a redirect)
- `403 Forbidden`
- `404 Not Found`
- `500 Internal Server Error`

Examples of Response Headers

- `Content-Type: text/html; charset=utf-8`
  Media type of the resource
- `Last-Modified: Mon, 18 Jul 2016 02:36:04 GMT`
  The last modification date of the resource
- `Set-Cookie: mykey=myvalue; expires=Mon, 17-Jul-2017 16:06:00 GMT; Max-Age=31449600; Path=/; secure`
  Send cookies from the server to the client

Examples of Content Types (“MIME Types”)

- `text/html`
- `text/css`
- `text/javascript`
- `image/png`
- `image/gif`
- `application/zip`
- `application/pdf`
- `application/json`

Content of an HTTP Message

Also called: *body*, *payload*

Content of **responses**:

- A file of the specified content type (e.g. `text/html`)

Content of **requests**:

- Form data
- Uploaded file(s)

Difference between GET and POST

GET

- “Read”: Retrieve data from the server
- No request body

POST

- “Write”: Submit data to the server
- Request body (“payload”)

Topics

1. Web Forms
2. HTTP
3. Ajax
4. Backend Programming

Motivation

- So far, posting the form requires a complete reload of the page.
- But many modern web applications exchange information all the time.
- Javascript can be used to perform an HTTP request at any time (“asynchronous request” / “Ajax”).

Ajax GET Request

```
const request = new Request(
    'https://example.com',
    {method: 'GET'}
);

fetch(request).then(function(response) {
    if (response.status === 200) {
      console.log(response);
    }
});
```

fetch returns a Promise

```
var callback1 = function() {
    // ...
};

var callback2 = function() {
    // ...
};

Promise.then(callback1).catch(callback2);
```

Alternative Callback Syntax

```
Promise.then(function(data) {
    // ...
})
```

can also be written as:

```
Promise.then(data => {
    // ...
})
```

Ajax POST Request

```
const request = new Request(
    'https://example.com',
    {
        method: 'POST',
        body: '{"foo": "bar"}',
    }
);

fetch(request).then(response => {
    if (response.status === 200) {
      console.log(response);
    }
});
```

Form Submit Events

```
<form id="my-form">
  <input type="text" name="username">
  <button type="submit">Send!</button>
</form>

<script>
  const form = document.querySelector('#my-form');
  form.addEventListener('submit', function(event) {
      console.log('The form was submitted.')
      // Disable the synchronous POST request
      event.preventDefault();
      // Perform an asynchronous request
      const request = ...
  });
</script>
```

Repetition Quiz

https://t.uzh.ch/1BW

Topics

1. Web Forms
2. HTTP
3. Ajax
4. Backend Programming

Goal

- Use Python to write a server script that processes HTTP requests

Code Structure

```
from http.server import BaseHTTPRequestHandler, HTTPServer

class HTTPRequestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        ...

    def do_POST(self):
        ...

if __name__ == '__main__':
    server = HTTPServer(
        ("localhost", 5000),
        RequestHandlerClass=HTTPRequestHandler,
    )
    server.serve_forever()
```

Handling a GET Request

```
def do_GET(self):
    html = """
    <html>
        <head>
            <meta charset="utf-8">
            <title>My first form</title>
        </head>
        <body>
            <!-- Write something here -->
        </body>
    <html>"""
    response_body = html.encode("utf-8")
    self.send_response(200)
    self.send_header("Content-Type", "text/html")
    self.send_header('Content-Length', str(len(response_body)))
    self.end_headers()
    self.wfile.write(response_body)
```

Handling a POST Request

```
import urllib

def do_POST(self):
    content_length = int(self.headers['Content-Length'])
    request_body = self.rfile.read(content_length).decode("utf-8")
    request_data = urllib.parse.parse_qs(request_body)
    html = """
    <html>
        <!-- ... -->
    <html>
    """
    response_body = html.encode("utf-8")
    self.send_response(200)
    self.send_header("Content-Type", "text/html")
    self.send_header('Content-Length', str(len(response_body)))
    self.end_headers()
    self.wfile.write(response_body)
```