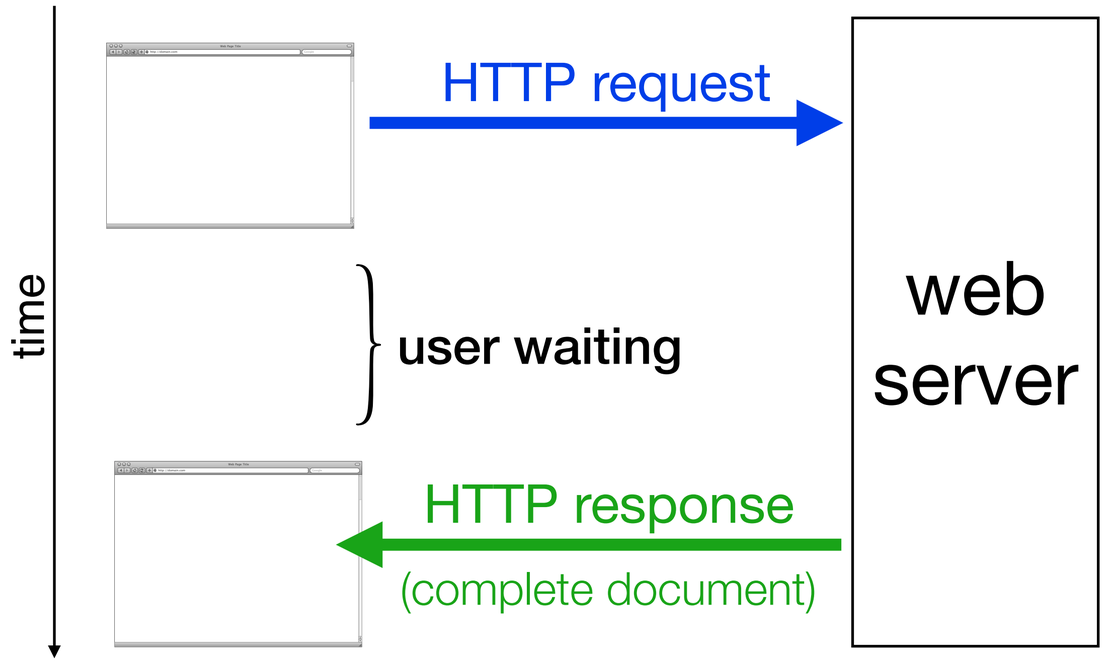
Web Programming
AJAX

Leander Jehl | University of Stavanger

Traditional web interaction

- User requests a page = browser (client) sends HTTP request to server
- Browser is “blocked” from activity while it waits for the server to provide the document
- When the response arrives, the browser renders the document



synchronous request-response communication

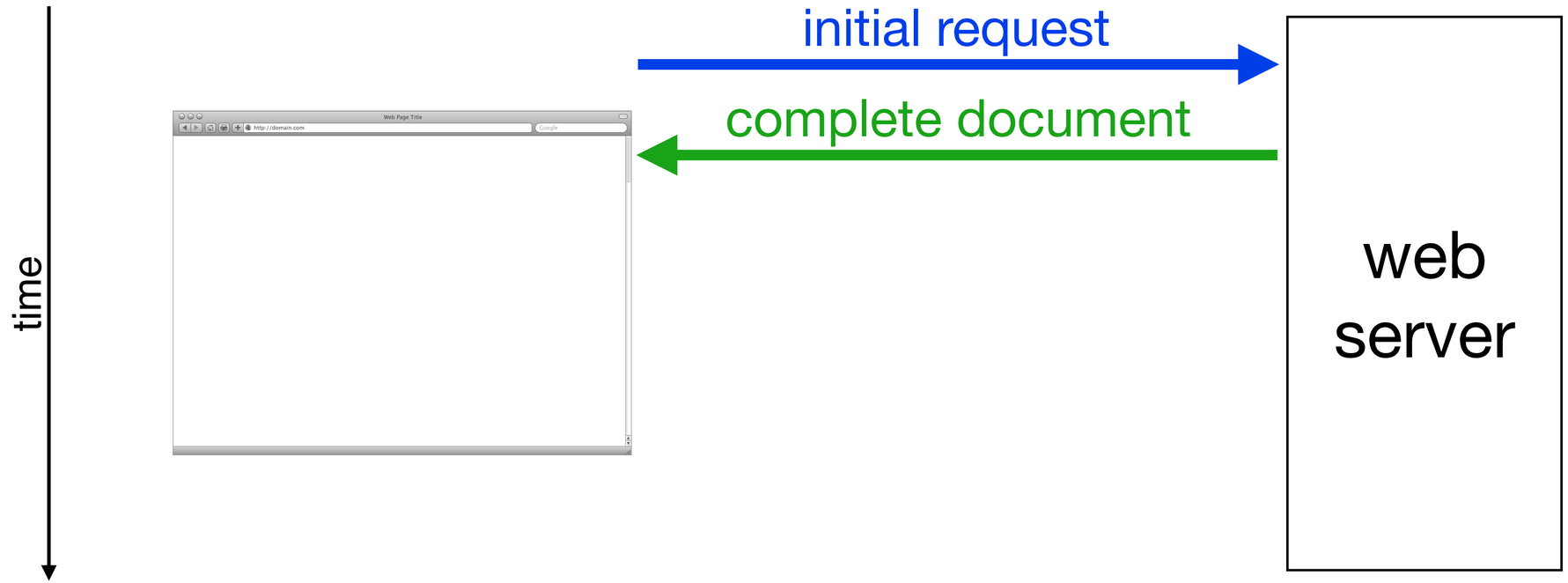
Motivation

- Provide web-based applications with rich user interfaces and responsiveness
- This requires frequent interactions between the user and the server
  - Speed of interactions determines the usability of the application!
- Often, only (relatively small) parts of the documents are modified or updated. No need to reload the entire page
- Client might want to send data to the server in the background

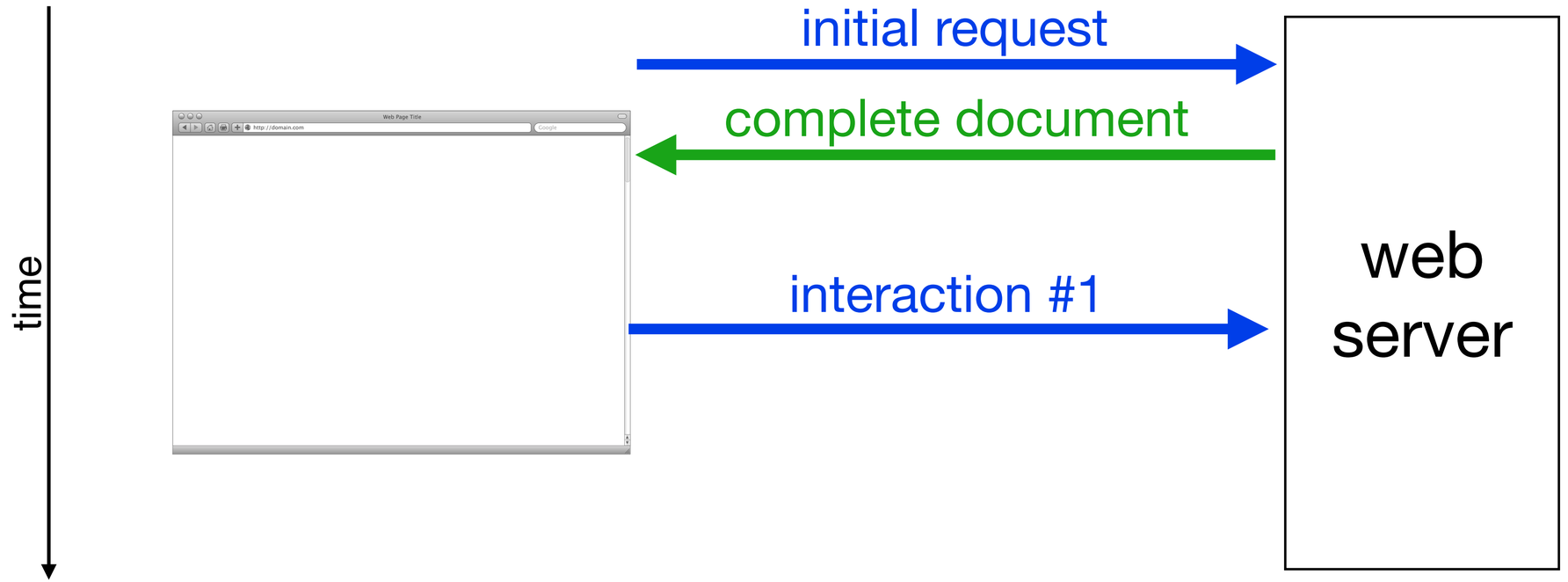
AJAX

- **A**synchronous **J**avaScript **a**nd **X**ML
- Combination of web technologies
  - Client side: HTML, JavaScript
  - Server side: any programming language
  - Despite the name, XML is not required!
- Two key features
  - Retrieve data, not pages
  - Asynchronous, i.e., no need to "lock" the document while waiting for the response

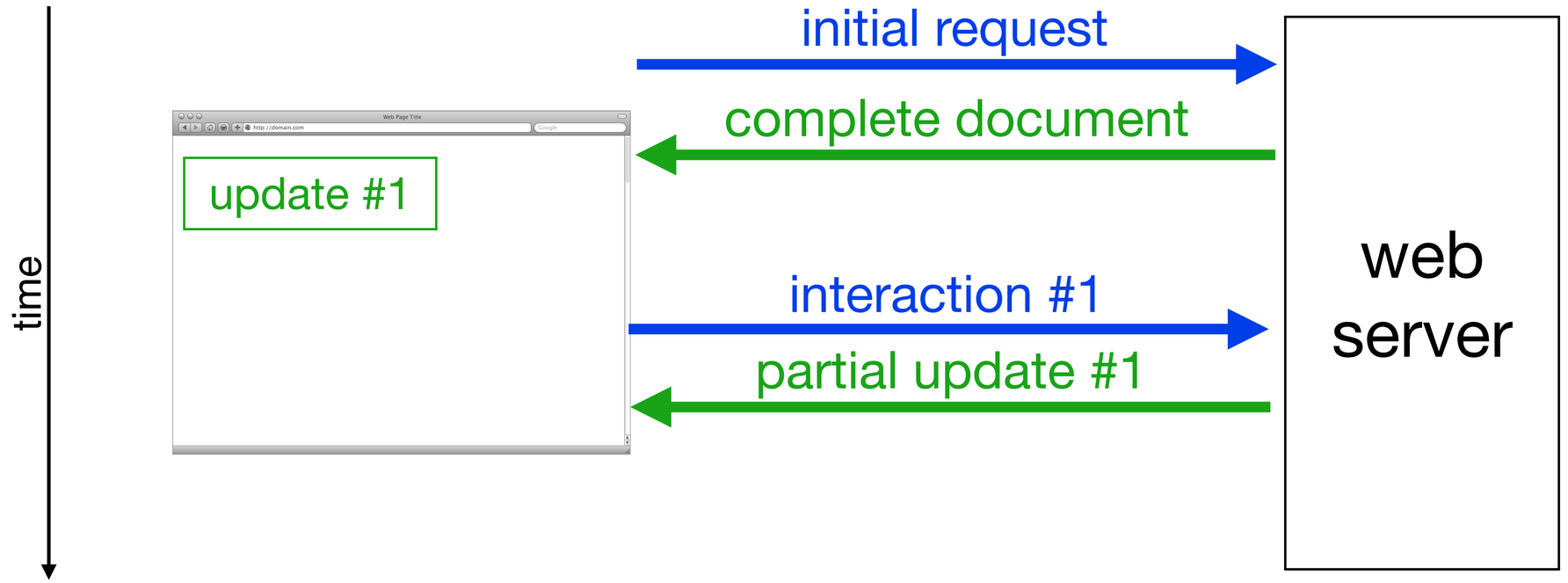
AJAX interaction



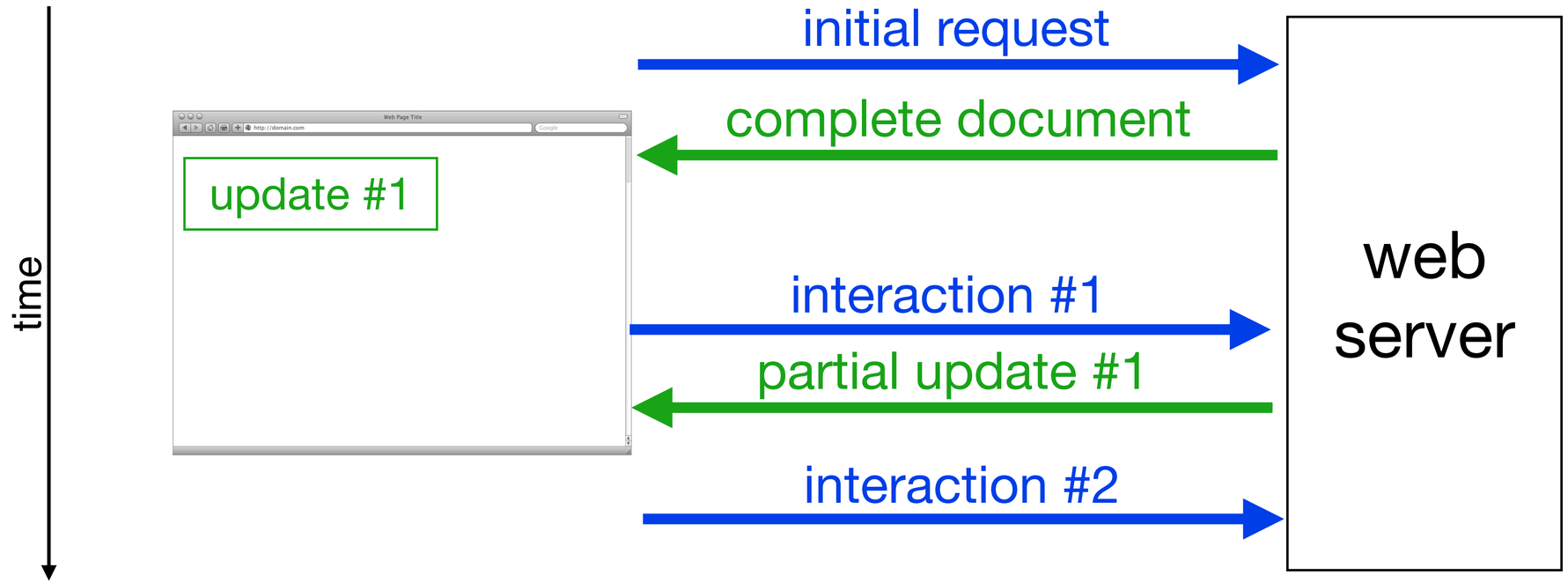
AJAX interaction



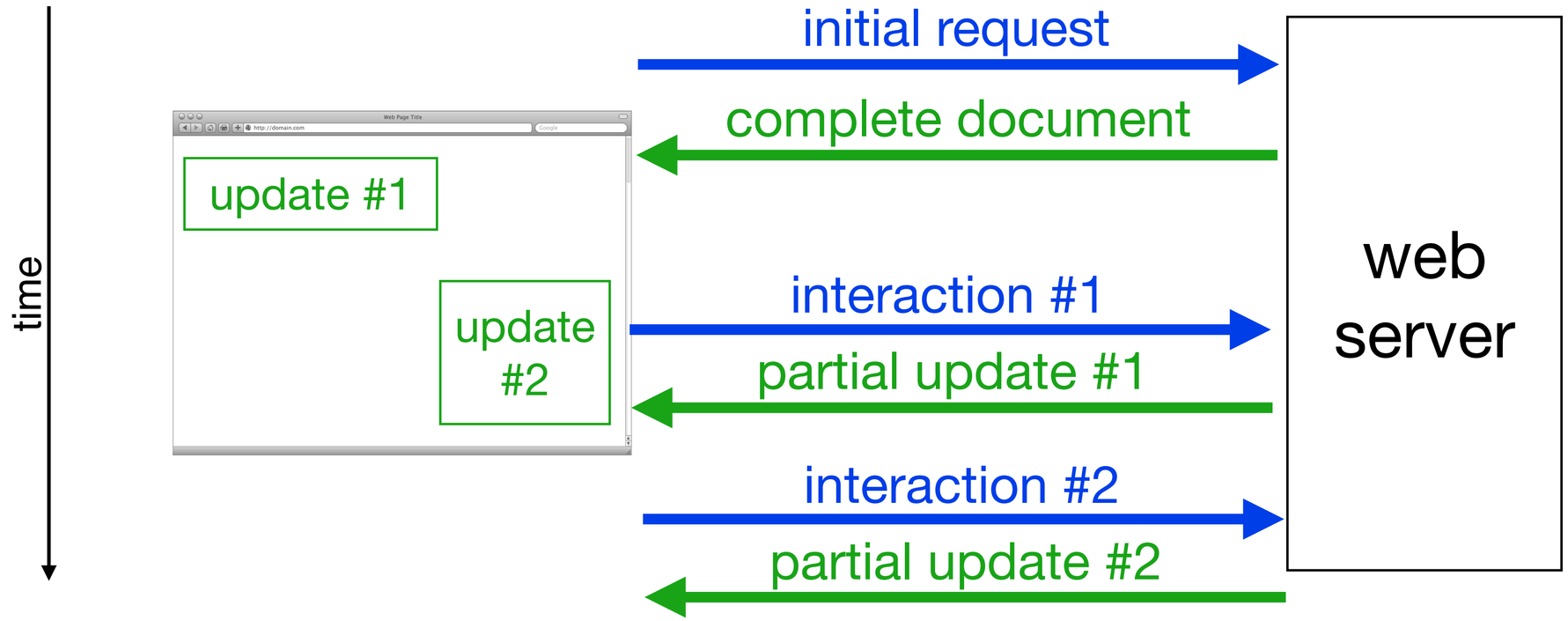
AJAX interaction



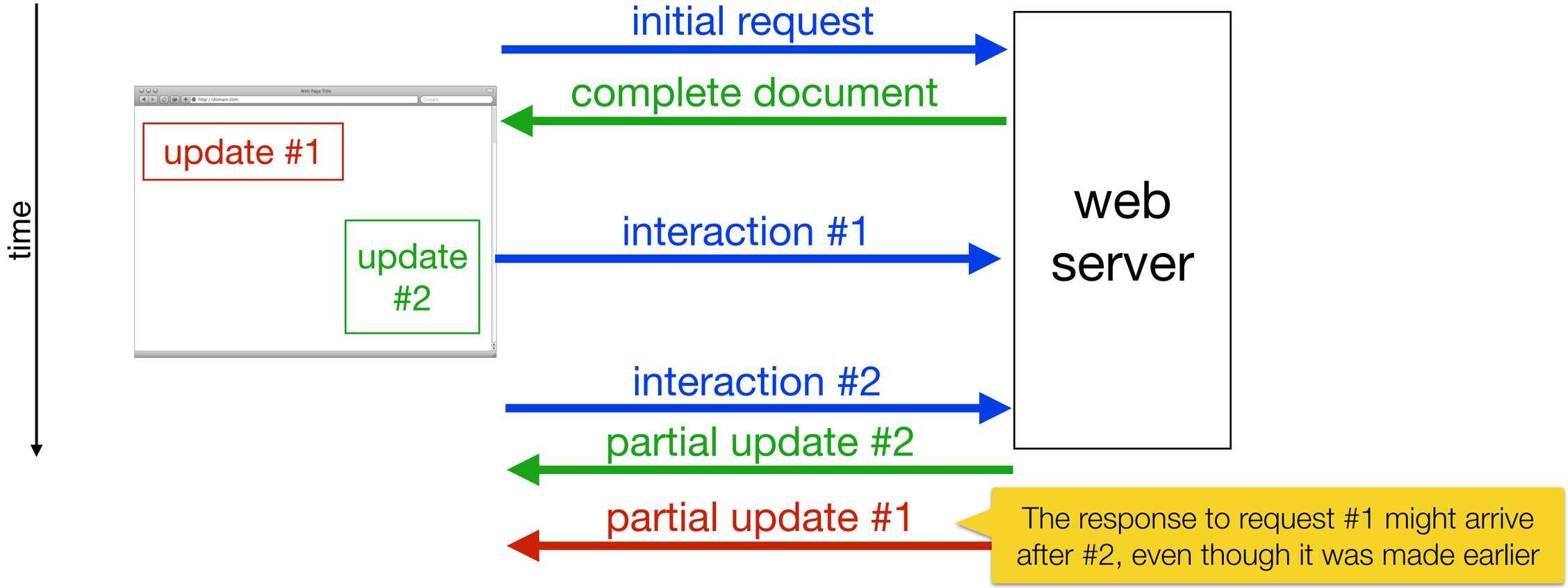
AJAX interaction



AJAX interaction

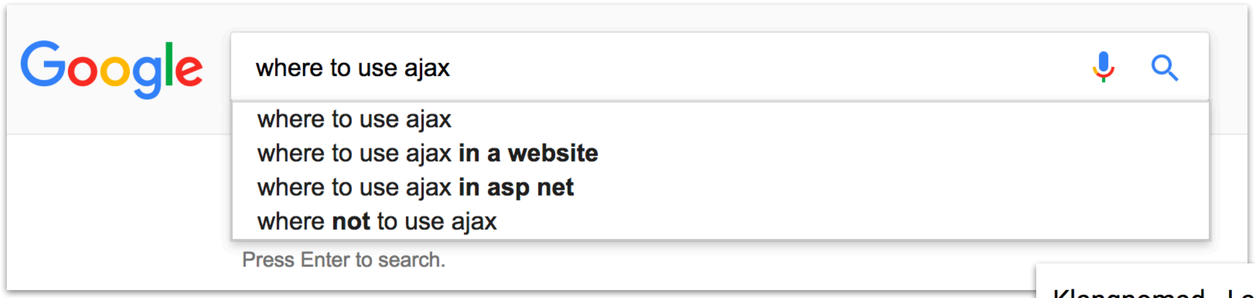


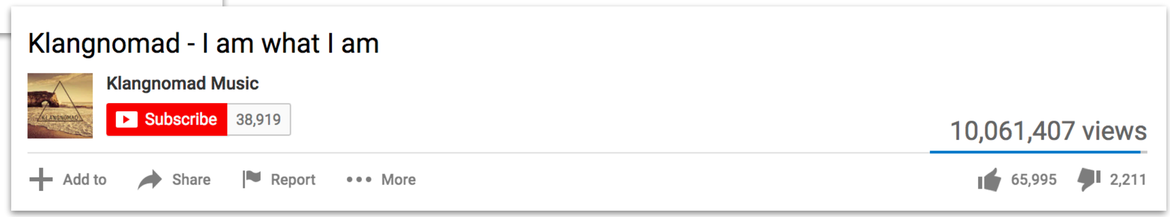
Note that responses are asynchronous

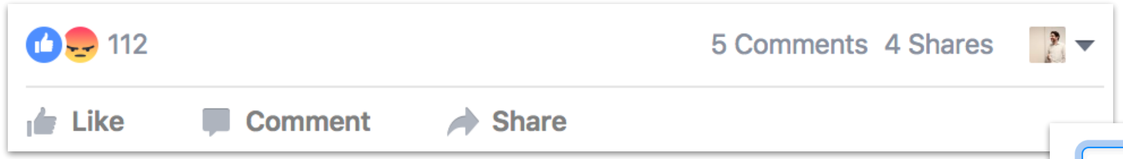


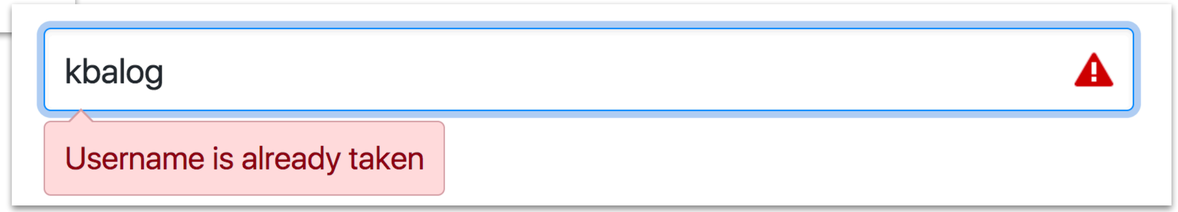
Where to use AJAX?

Where to use AJAX?







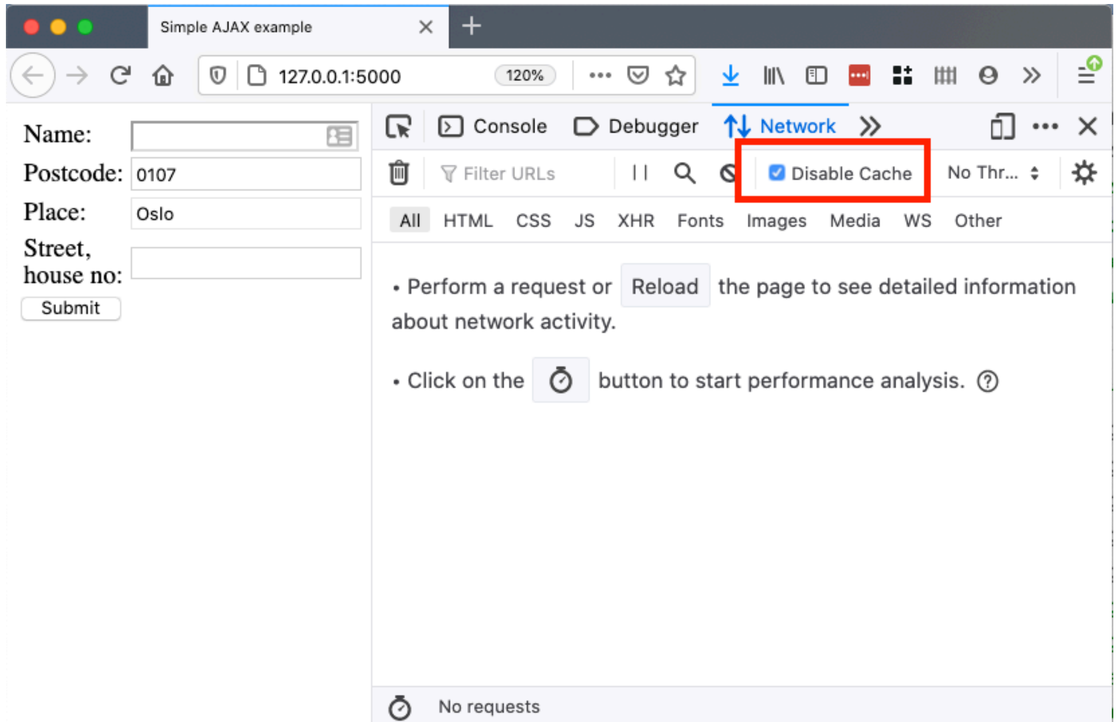


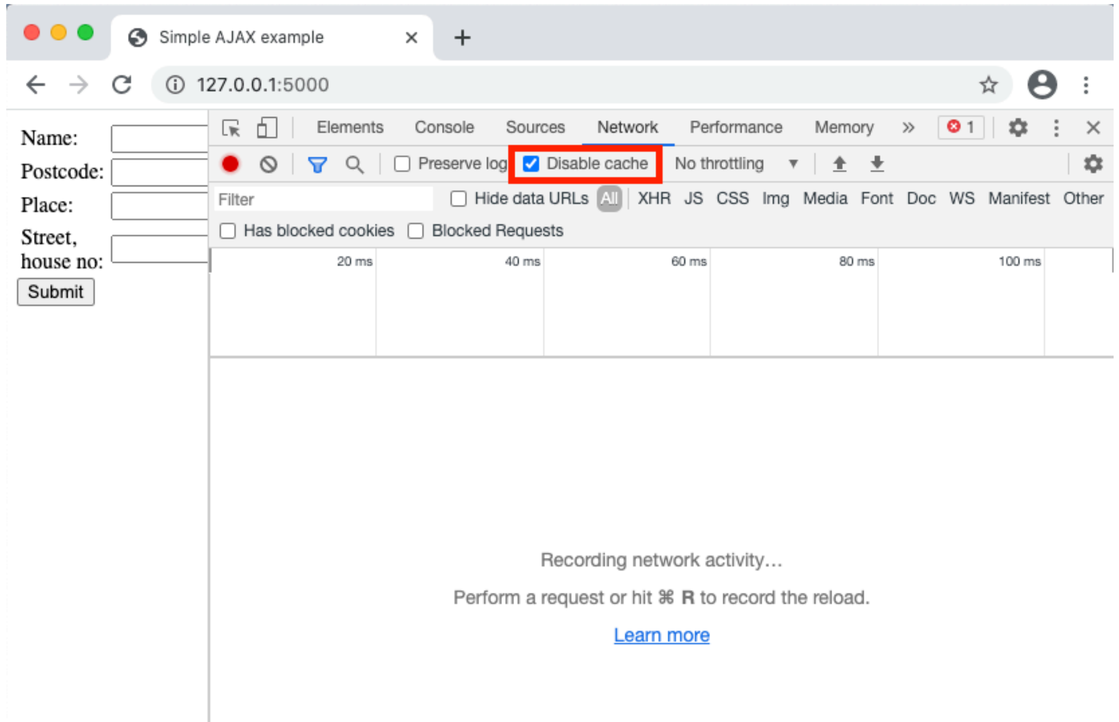
Four main parts

1. Initial HTML document (may be generated using Python)
2. JavaScript to send the AJAX request to the server
3. Server-side program to receive the request and produce the requested data
4. JavaScript to receive the new data and integrate it into the original document being displayed

Tips

When working with AJAX, open the developer tools in your browser, go to network tab, and **disable the cache.**

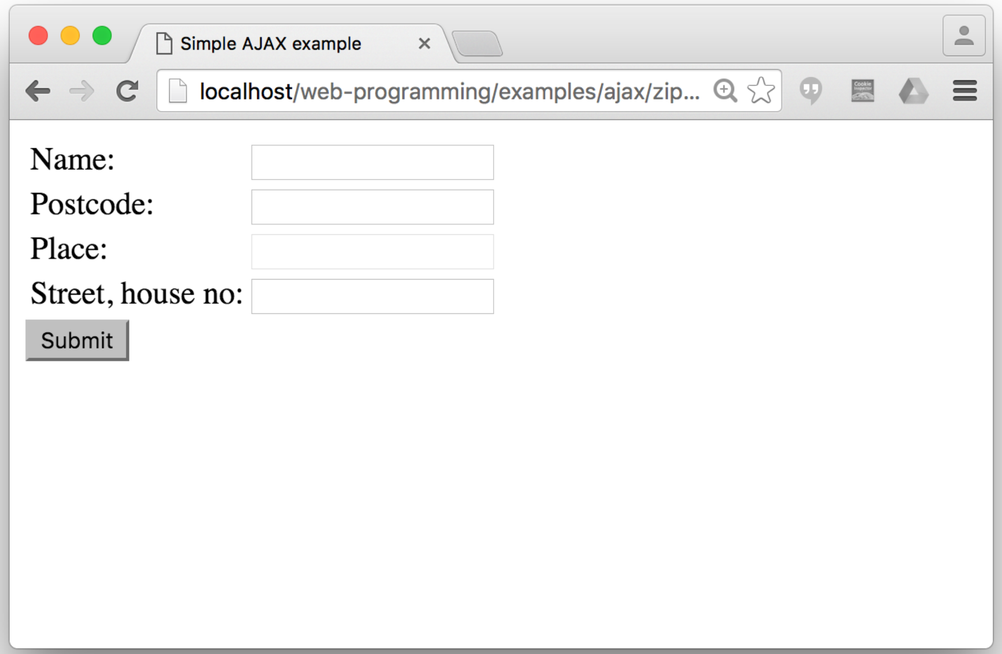


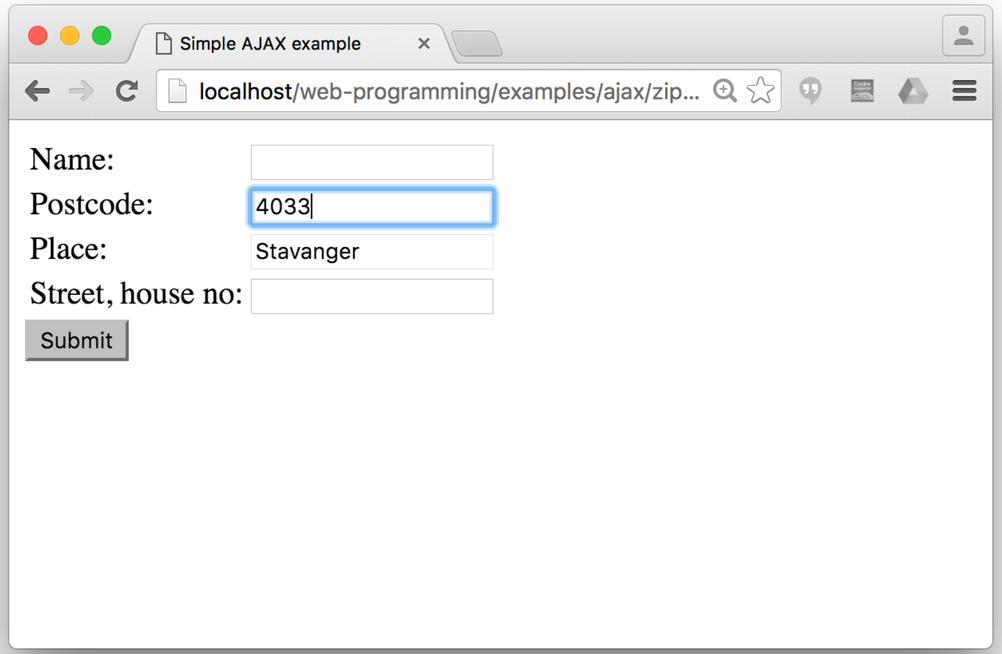


Example walkthrough

https://github.com/dat310-2025/info/tree/master/
examples/ajax/zipcode

Example


Simple AJAX example
localhost/web-programming/examples/ajax/zip...
Name:
Postcode:
Place:
Street, house no:
Submit


Simple AJAX example
localhost/web-programming/examples/ajax/zip...
Name:
Postcode:
4033
Place:
Stavanger
Street, house no:
Submit

1. Initial HTML document

- Register JavaScript handler function on onkeyup event
  - I.e., whenever the user presses a key

zipcode.html

```
<input type="text" name="postcode" onkeyup="getPlace(this.value);"/>
```

2. Request phase

- Make call using `fetch`
- Wait for reply using `await`

zipcode.js

```
async function getPlace(postcode){
    let url = "/getplace?postcode=" + postcode;
    let response = await fetch(url);
    ...
}
```

3. Response document

- Flask app generates simple text response

app.py

```
@app.route("/getplace", methods=["GET"])
def getplace():
    POSTCODES = {
        "0107": "Oslo",
        "0506": "Oslo",
        "4090": "Hafrsfjord",
        …
   }
    postcode = request.args.get("postcode", None)
    # look up corresponding place or return empty string
    if postcode and (postcode in POSTCODES):
        return POSTCODES[postcode]
    return ""
```

4. Receiver phase

- Status is 200 if the request was successfully completed
- `text()` returns a promise, which is resolved to the response text.

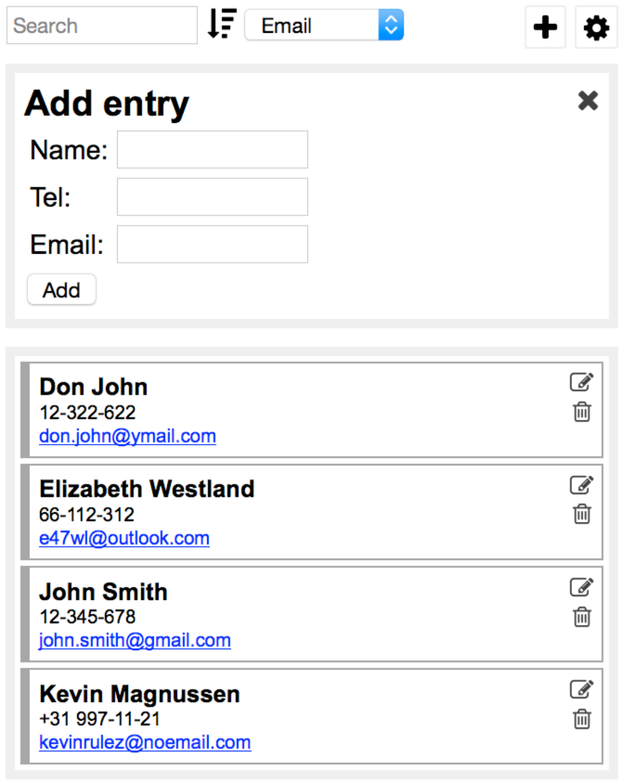
zipcode.js

```
async function getPlace(postcode){
    let url = "/getplace?postcode=" + postcode;
    let response = await fetch(url);
    if (response.status == 200){
        let result = await response.text();
        updatePlace(result);
    }
}
```

returns a `Promise`, thus we need to `await` the result.

Comparison

JS application



Flask application

Courses

Course Id	Name
MAT100	Mathematical methods I.
MAT200	Mathematical methods II.
DAT100	Object-oriented programming
DAT200	Algorithms and data structures
DAT220	Databases
DAT310	Web programming
DAT320	Operating Systems

Students

Student no	Name
111111	John Smith
222222	Mary Jane
333333	Lars Kongen

add new student

AJAX application

Name:

Postcode: 4033

Place: Stavanger

Street, house no:

Submit

Comparison

JS application

1. Load program (JS) from server.
2. Run program on browser.

Flask application

Courses

Course Id	Name
MAT100	Mathematical methods I.
MAT200	Mathematical methods II.
DAT100	Object-oriented programming
DAT200	Algorithms and data structures
DAT220	Databases
DAT310	Web programming
DAT320	Operating Systems

Students

Student no	Name
111111	John Smith
222222	Mary Jane
333333	Lars Kongen

add new student

AJAX application

Name:

Postcode: 4033

Place: Stavanger

Street, house no:

Submit

Comparison

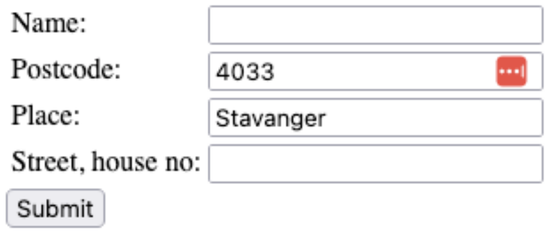
JS application

1. Load program (JS) from server.
2. Run program on browser.

Flask application

1. Run program (python) on server.
2. Browser displays.

AJAX application



Comparison

JS application

1. Load program (JS) from server.
2. Run program on browser.

Flask application

1. Run program (python) on server.
2. Browser displays.

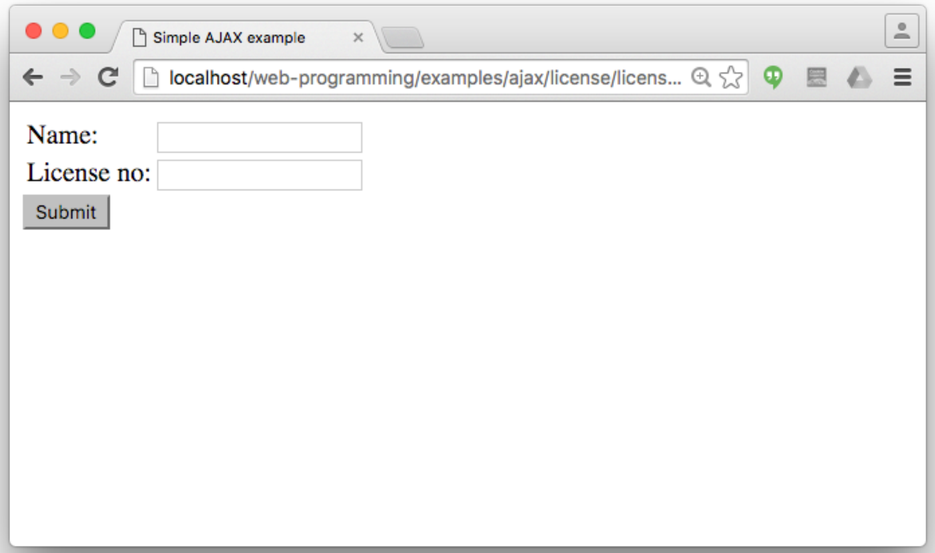
AJAX application

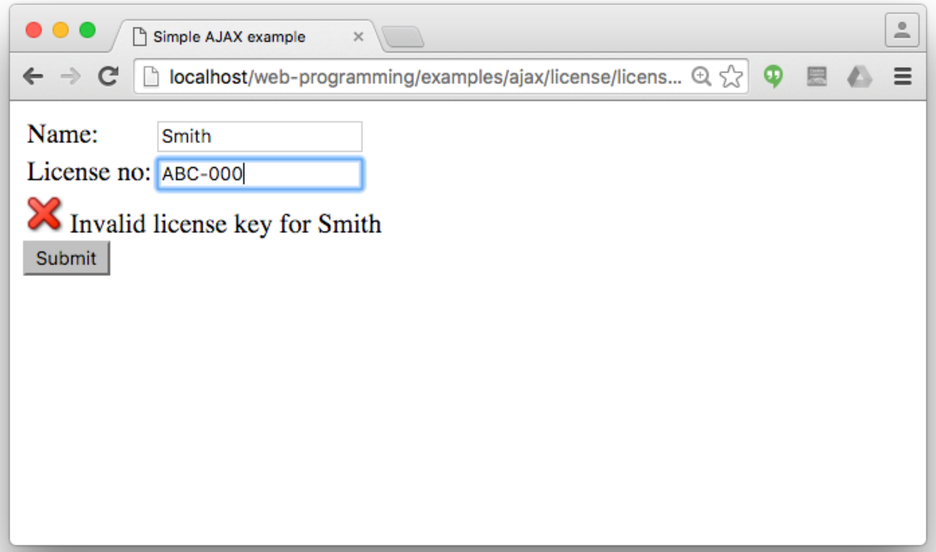
1. Load program (JS) from server.
2. Run program (python) on server.
3. JS and python communicate via AJAX

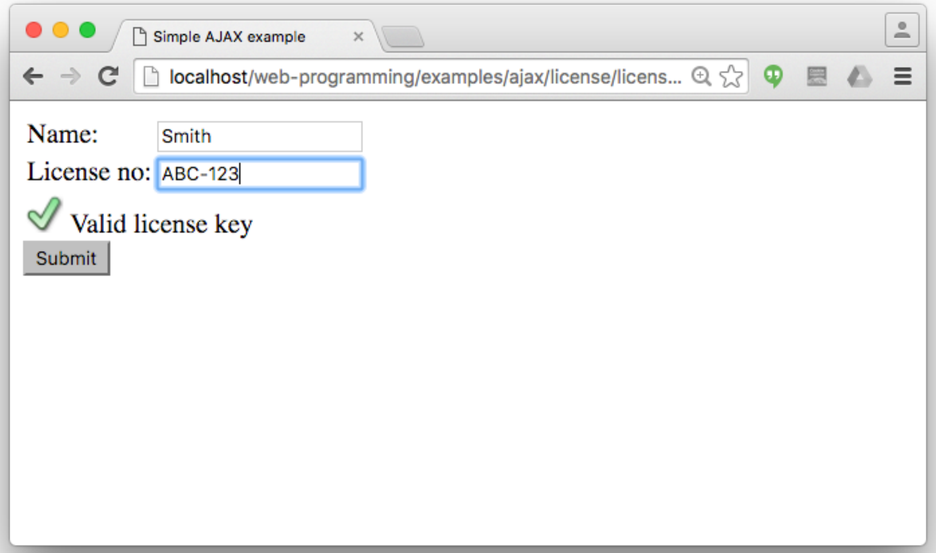
Example walkthrough #2

https://github.com/dat310-2025/info/tree/master/
examples/ajax/license

Example #2


Simple AJAX example
localhost/web-programming/examples/ajax/license/licens...
Name:
License no:
Submit


Simple AJAX example
localhost/web-programming/examples/ajax/license/licens...
Name:
Smith
License no:
ABC-000
Invalid license key for Smith
Submit


Simple AJAX example
localhost/web-programming/examples/ajax/license/licens...
Name:
Smith
License no:
ABC-123
Valid license key
Submit

Example #2

- Request can be POST as well
- It is also possible for the server to send back a HTML snippet
- The client updates part of the page (i.e., the DOM) with the received snippet

1. Initial HTML document

- Register JavaScript handler function on onkeyup events
  - I.e., whenever the user presses a key in the name or license fields

`license.html`

```
<input type="text" name="name" id="name" onkeyup="checkLicense();" />

<input type="text" name="license" id="license" onkeyup="checkLicense();" />
```

2. Request phase

- Make asynchronous call using POST
  - Need to add a HTTP header to make it as if it was a form submission

license.js

```
async function checkLicense(){
    var name = document.getElementById("name").value;
    var license = document.getElementById("license").value;

    let result = await fetch("/check_license",{
        method: "POST",
        headers: {
            "Content-Type": "application/x-www-form-urlencoded",
        },
        body: "name=" + name + "&license=" + license
    });
```

3. Response document

- Flask app generates a HTML snippet

app.py

```
@app.route("/check_license", methods=["POST"])
def check_license():
    VALID_LICENSES = {…}
    name = request.form.get("name", None)
    license = request.form.get("license", None)
    # check if name and license match
    if name and license:
        if VALID_LICENSES.get(name, None) == license:
            return "<img src='/static/images/yes.png' /> Valid license key"
        else:
            return "<img src='/static/images/no.png' />
                    Invalid license key for {}".format(name)
    return ""
```

4. Receiver phase

- Status is 200 if the request was successfully completed
- `text()` returns a promise, which is resolved to the response text.

license.js

```
if (response.status == 200){
        let result = await response.text()
        document.getElementById("license_check").innerHTML = result;
    }
```

Fetch

Fetch

- Takes as argument the URL to send request to
- Returns a promise
- Use await to wait for reply

```
let reply = await fetch("/getplace?postcode=" + postcode);
```

Sends **GET** request if no additional arguments are given.

Encode parameters, just as when sending form.

Fetch response

- Access response text using **`response.text()`**
- **`response.text()`** returns another promise
- **await** for actual text result

```
let reply = await fetch("/getplace?postcode=" + postcode);
let result = await reply.text();
```

Fetch POST

- Fetch takes as second argument, an object

```
let data = "name=" + name + "&license=" + license;
let result = await fetch("/check_license",{
            method: "POST",
            headers: {
                "Content-Type": "application/x-www-form-urlencoded",
            },
            body: data,
        })
```

Encode parameters, just as when sending form.

- Response is handled as with GET request.

```
if (response.status == 200){
        let result = await response.text()
        document.getElementById("license_check").innerHTML = result;
    }
```

Exercises #1, #1b

github.com/dat310-2025/info/tree/master/
exercises/ajax

What can be the response document?

- Data as a simple string
- HTML snippet
- Data as "object"
  - Both the client and the server need to speak the same language, i.e., how to *encode* and *decode* the object

JSON

- JavaScript Object Notation
- Lightweight data-interchange format
- Language independent
- Two structures
  - Collection of name-value pairs (object)
    - a.k.a. record, struct, dictionary, hash table, associative array
  - Ordered list of values (array)
    - a.k.a. vector, list

JSON

- Values can be
  - string (in between "…")
  - number
  - object
  - array
  - boolean (true/false)
  - null

Example JSON

```
{
 "name":"John Smith",
 "age":32,
 "married":true,
 "interests":[1,2,3],
 "other":{
        "city":"Stavanger",
        "postcode":4041
         }
}
```

JSON with Python

examples/ajax/json/json_python.py

- **`json`** is a standard module
- **`json.dumps(data)`**
  - returns JSON representation of the data
- **`json.loads(json_value)`**
  - decodes a JSON value
- **`json.dumps()`** and **`json.loads()`** work with strings
- **`json.dump()`** and **`json.load()`** work with file streams

JSON with JavaScript

examples/ajax/json/json_js.html

- **`JSON.stringify(value)`**
  - returns JSON representation of a value (encode)
- **`JSON.parse(json)`**
  - parses a JSON value into a JavaScript object (decode)

Example

https://github.com/dat310-2025/info/tree/master/
examples/ajax/json/student

Example

examples/ajax/json/student

app.py

```
@app.route("/get_data", methods=["GET"])
def check_license():
    DATA = {
        "name":"John Doe",
        "student_no": 111111
    }
    return json.dumps(DATA)

@app.route("/post_data", methods=["POST"])
def print_data():
    print(request.get_json())
    return "OK"
```

Reply with json data.

Receive json data.

Example

examples/ajax/json/student

student.js

```
async function getStudent(){
    let reply = await fetch("/get_data");
    if (reply.status == 200){
        let result = await reply.json();
 ...
```

Receive json data.

student.js

```
async function sendStudent(){
    let student = { name: name, student_no: student_no };

    let reply = await fetch("/post_data",{
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(student)
    });
   ...
}
```

Include JSON data in request

Exercise #2

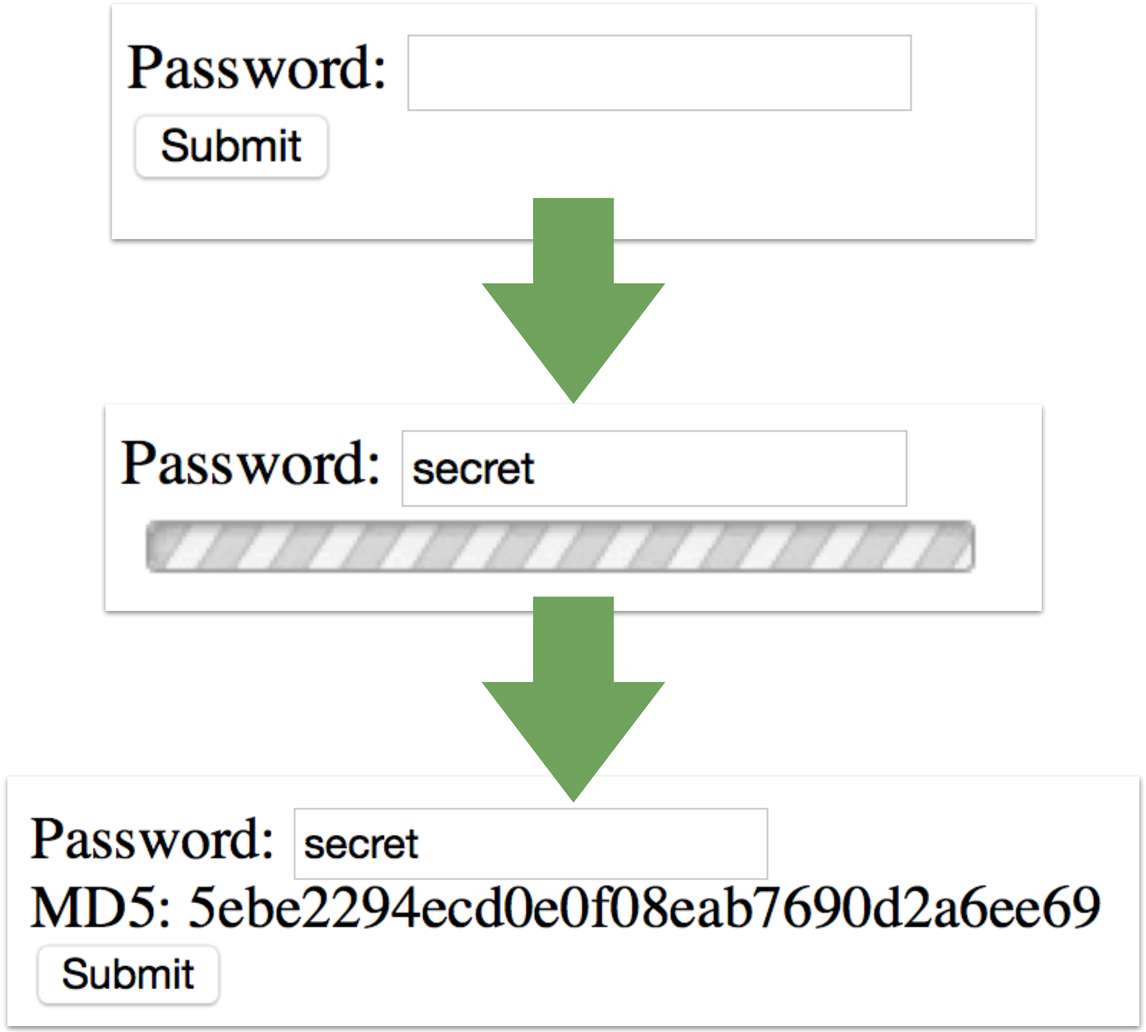
github.com/dat310-2025/info/tree/master/
exercises/ajax

Example

https://github.com/dat310-2025/info/tree/master/
examples/ajax/loading

Indicating waiting

- An animated gif is displayed until the response arrives
- In this example there is an artificial delay of 1sec is added to the Python code

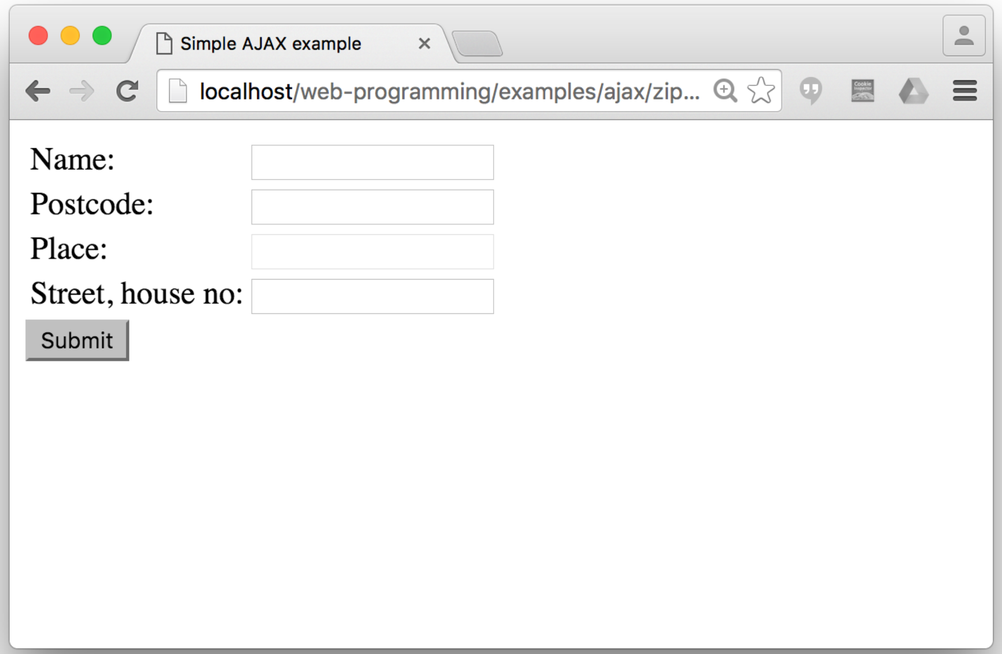


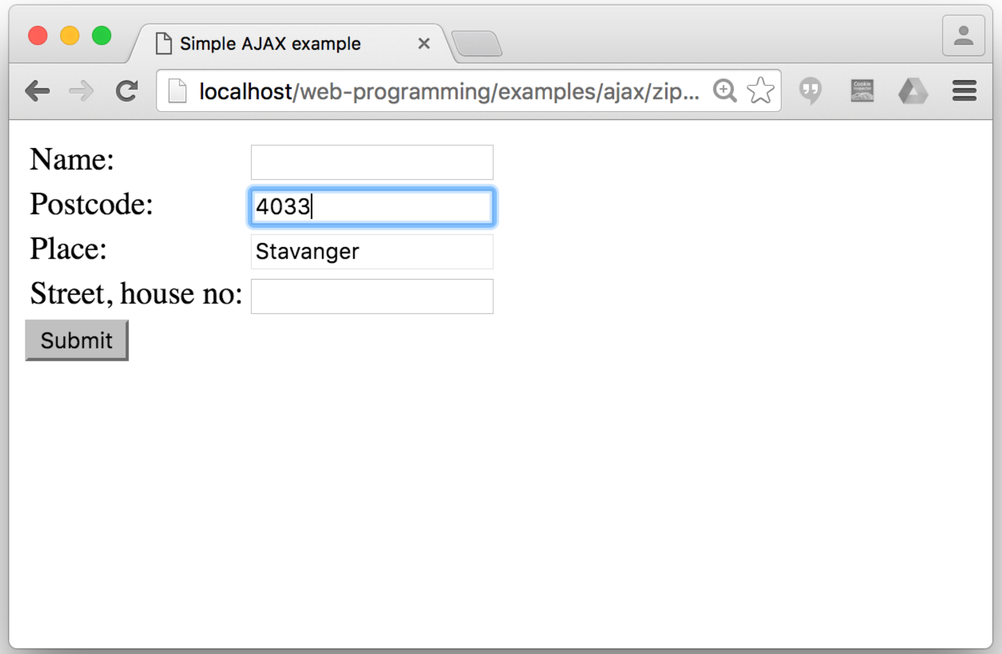
AJAX without async

Example walkthrough

https://github.com/dat310-2024/info/tree/master/**examples/ajax/zipcode**

Example


Simple AJAX example
localhost/web-programming/examples/ajax/zip...
Name:
Postcode:
Place:
Street, house no:
Submit


Simple AJAX example
localhost/web-programming/examples/ajax/zip...
Name:
Postcode:
4033
Place:
Stavanger
Street, house no:
Submit

1. Initial HTML document

- Register JavaScript handler function on onkeyup event
  - I.e., whenever the user presses a key

zipcode.html

```
<input type="text" name="postcode" onkeyup="getPlace(this.value);"/>
```

2. Request phase

- Register callback function
- Make asynchronous call

zipcode.js

```
function getPlace(postcode) {
    var xhr = new XMLHttpRequest();
    /* register an embedded function as the handler */
    xhr.onreadystatechange = function () {
        […]
        }
    };
    /* send the request using GET */
    xhr.open("GET", "/getplace?postcode=" + postcode, true);
    xhr.send(null);
}
```

setting this parameter to **`true`** means making an asynchronous request

3. Response document

- Flask app generates simple text response

app.py

```
@app.route("/getplace", methods=["GET"])
def getplace():
    POSTCODES = {
        "0107": "Oslo",
        "0506": "Oslo",
        "4090": "Hafrsfjord",
        …
   }
    postcode = request.args.get("postcode", None)
    # look up corresponding place or return empty string
    if postcode and (postcode in POSTCODES):
        return POSTCODES[postcode]
    return ""
```

4. Receiver phase

- Callback is called multiple times, readyState indicates the progress (0..4)
- Status is 200 if the request was successfully completed

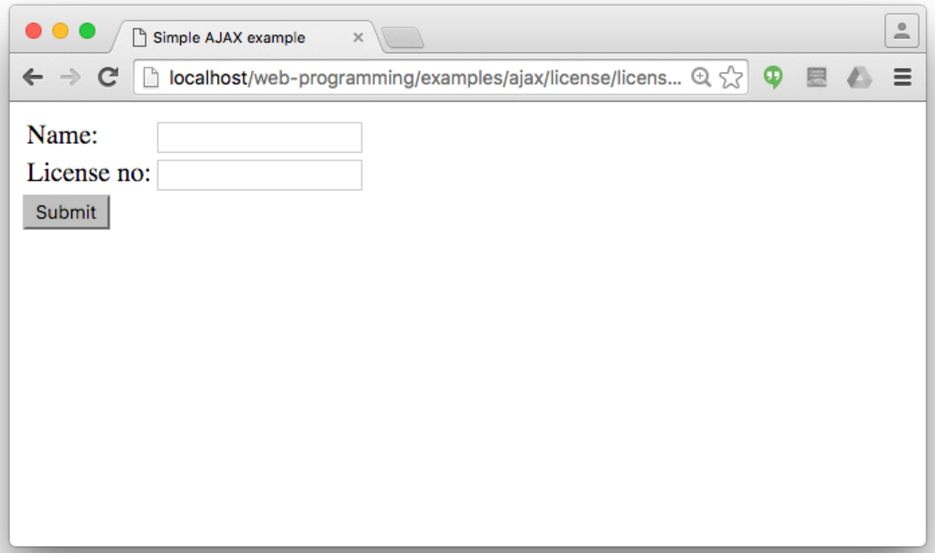
zipcode.js

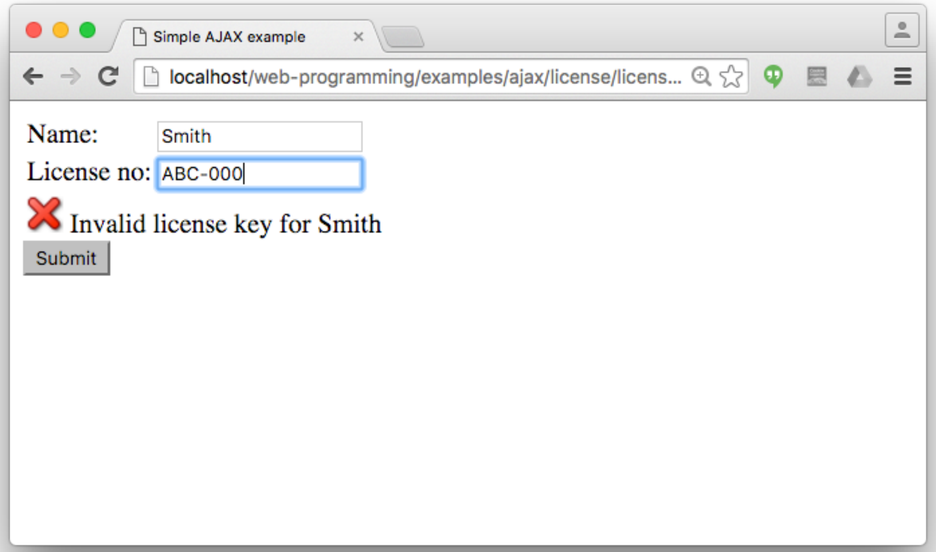
```
xhr.onreadystatechange = function () {
    /* readyState = 4 means that the response has been completed
     * status = 200 indicates that the request was successfully completed */
    if (xhr.readyState == 4 && xhr.status == 200) {
        var result = xhr.responseText;
        document.getElementById("place").value = result;
    }
};
```

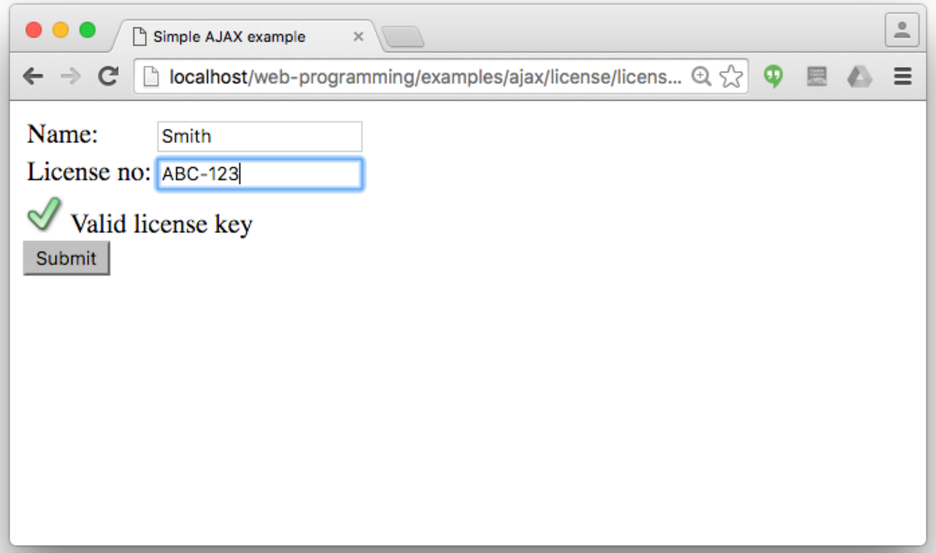
Example walkthrough #2

https://github.com/dat310-2024/course-info/tree/master/
examples/ajax/license

Example #2


Simple AJAX example
localhost/web-programming/examples/ajax/license/licens...
Name:
License no:
Submit


Simple AJAX example
localhost/web-programming/examples/ajax/license/licens...
Name:
Smith
License no:
ABC-000
Invalid license key for Smith
Submit


Simple AJAX example
localhost/web-programming/examples/ajax/license/licens...
Name:
Smith
License no:
ABC-123
Valid license key
Submit

Example #2

- Request can be POST as well
- It is also possible for the server to send back a HTML snippet
- The client updates part of the page (i.e., the DOM) with the received snippet

1. Initial HTML document

- Register JavaScript handler function on onkeyup events
  - I.e., whenever the user presses a key in the name or license fields

`license.html`

```
<input type="text" name="name" id="name" onkeyup="checkLicense();" />

<input type="text" name="license" id="license" onkeyup="checkLicense();" />
```

2. Request phase

- Make asynchronous call using POST
  - Need to add a HTTP header to make it as if it was a form submission

license.js

```
function checkLicense() {
     […]

    /* send the request using POST */
    xhr.open("POST", "/check_license", true);
    /* To POST data like an HTML form, add an HTTP header */
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    /* variables go in the request body */
    xhr.send("name=" + name + "&license=" + license);

     […]
}
```

3. Response document

- Flask app generates a HTML snippet

app.py

```
@app.route("/check_license", methods=["POST"])
def check_license():
    VALID_LICENSES = {…}
    name = request.form.get("name", None)
    license = request.form.get("license", None)
    # check if name and license match
    if name and license:
        if VALID_LICENSES.get(name, None) == license:
            return "<img src='/static/images/yes.png' /> Valid license key"
        else:
            return "<img src='/static/images/no.png' />
                    Invalid license key for {}".format(name)
    return ""
```

4. Receiver phase

- Callback is called multiple times, readyState indicates the progress (0..4)
- Status is 200 if the request was successfully completed

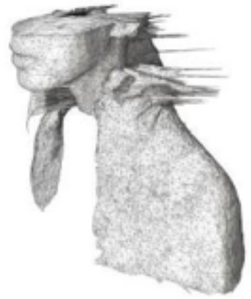
license.js

```
xhr.onreadystatechange = function () {
    /* readyState = 4 means that the response has been completed
     * status = 200 indicates that the request was successfully completed */
    if (xhr.readyState == 4 && xhr.status == 200) {
        var result = xhr.responseText;
        document.getElementById("license_check").innerHTML = result;
    }
};
```

Assignment 7

- Use vue or js
- Check next lecture on how to combine vue and flask

Coldplay - coldplay-cover.jpg
Guns N' Roses - Greatest Hits
Nightwish - Century Child
U2 - No Line On The Horizon



No.	Title	Length
1.	Politik	5:18
2.	In My Place	3:48
3.	God Put a Smile upon Your Face	4:57
4.	The Scientist	5:09
5.	Clocks	5:07
6.	Daylight	5:27
7.	Green Eyes	3:43
8.	Warning Sign	5:31
9.	A Whisper	3:58
10.	A Rush of Blood to the Head	5:51
11.	Amsterdam	5:19
Total length:		**54:08**