Choose HTMX

and avoid learning too much JavaScript.

Martin Borus @ 🐍🍕 Hamburg Python Pizza 2023

about me

- writes software for ferry operators (and supports it, too)
- mainly booking systems
- using legacy versions of MS Dynamics NAV (aka. Navision)
- using Python for connectors and everything else

Classic HTML

Classic HTML

A web page looks like this

```
<html>
  <head>
  …
  </head>
  <body>
  …
  </body>
</html>
```

Classic HTML - Interactive Elements

- Links
  click here

- Forms
  <form action="/login">
  <input type="text" id="name" name="name" value="Martin">
  <input type="submit" value="Submit">
  </form>

- Page Redirections/Reloads
  <meta http-equiv="refresh" content="5;url=page.html" />

Current Interactive Web Pages

- are commonly driven by JavaScript
- require understanding of the JavaScript environment and tools
- use frameworks like React, Vue.js many others
- often communicate with backend by JSON, not HTML
- Browser modifies the DOM (Document Object Model) on the client

HTMX

HTMX in a nutshell

- is a JavaScript library which helps you avoid writing Javascript

- is removes the constraints of classic HTML
  - every element can be interactive
  - server responses can be fragments of pages

- created by Carson Gross (Big Sky Software) with a business friendly BSD-2 license.

- is available at htmx.org with good documentation

How to install

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Python Pizza Hamburg</title>

    <script src="https://unpkg.com/htmx.org@1.9.8"></script>

</head>
<body>

    ...
```

HTMX gives you attributes that control interactivity

- “Define which events trigger an action”
  hx-**trigger**
  e.g. click, load, mouse-over and many more

- “Define what kind of ajax request is sent to the server”
  hx-**get**, hx-**post**, hx-**put**, hx-**delete**

- “Define where the server response is displayed”
  hx-**target**

- “Let’s convert regular links into htmx-links”
  hx-**boost**

Let’s see 3 examples…

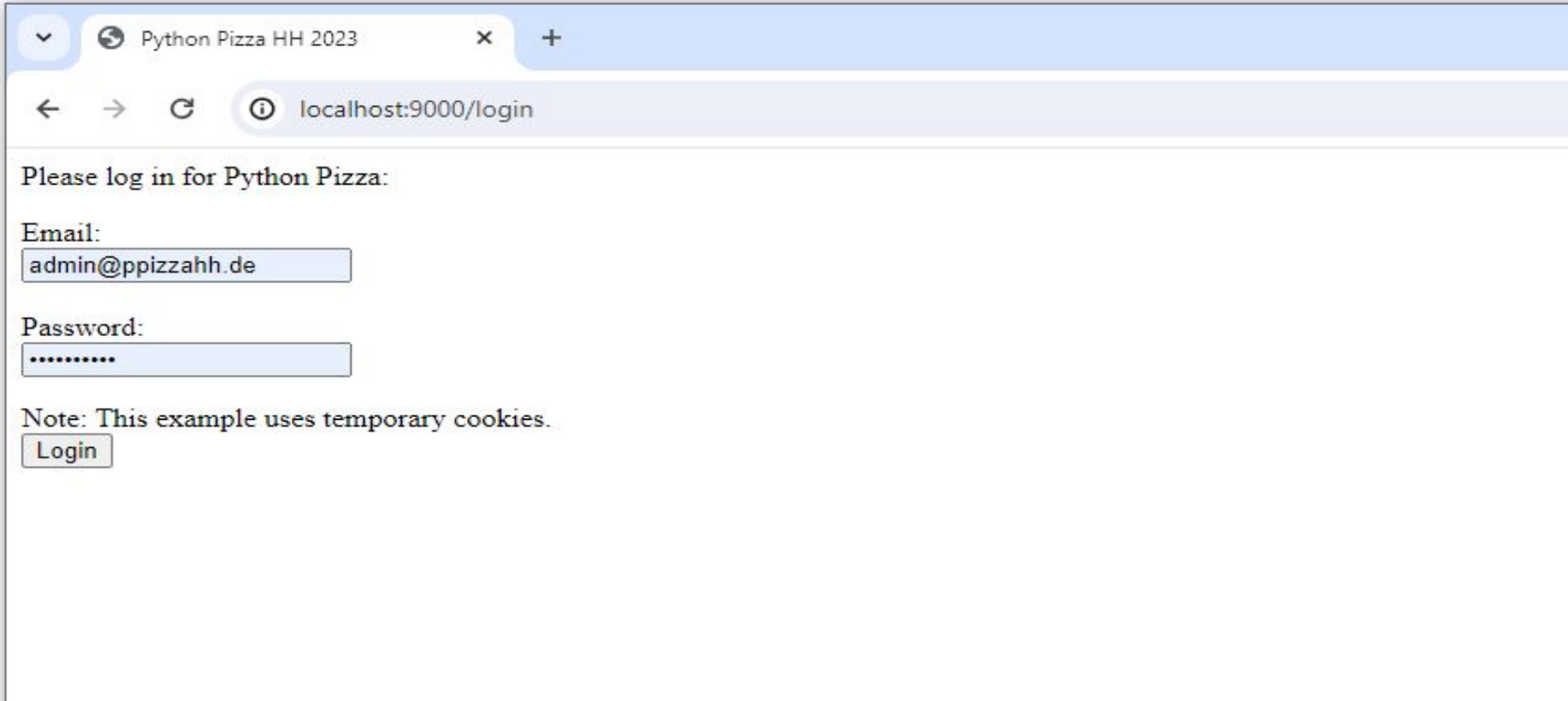
…with a FastAPI backend.

1. User Login Example

User Login Example - Backend

```
20
    Martin Borus
21  @router.get( path: "/login", response_class=HTMLResponse, include_in_schema=False)
22  def login_form(request: Request):
23      # allow body to contain html tags
24      body = Markup(Path( *args: TEMPLATE_PARTIAL_FOLDER, "login.html").read_text())
25
26      # if it's a HTMX request, return just the part
27      if request.headers.get("hx-request") == "true":
28          return HTMLResponse(body)
29
30      # if it's not a HTMX request, return a full page
31      return templates.TemplateResponse(
32          name=f"base.html",
33          context={
34              "request": request,
35              "title": "Python Pizza HH 2023",
36              "body": body,
37          },
38      )
```

User Login Example



User Login Example - Backend - Base Template

base.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ title }}</title>

    <script src="https://unpkg.com/htmx.org@1.9.8"></script>

    <!-- this is where you put your own JavaScript -->
    <script src="static/js/my.js"></script>

</head>
<body>

    {{ body }}

</body>
</html>
```

User Login Example

```
20 <body>
21
22     <div id="login-area">
23     Please log in for Python Pizza:
24     <p/>
25
26     <form hx-post="/login"
27           hx-target="#login-area"
28           hx-swap="outerHTML"
29     >
30         <label for="email">Email:</label><br>
31         <input type="email" id="email" name="email" required><br>
32         <br/>
33         <label for="password">Password:</label><br>
34         <input type="password" id="password" name="password" required><br>
35         <div id="login-fail"></div>
36
37         <p/>Note: This example uses temporary cookies.
38         <br/>
39         <button>Login</button>
40
41     </form>
42
43 </div>
44
45 </body>
```

User Login Example - Backend Post

```
Martin Borus
41  @router.post( path: "/login", response_class=HTMLResponse, include_in_schema=False)
42  def login(
43      request: Request, email: Annotated[str, Form()], password: Annotated[str, Form()]
44  ):
45      # it's a not HTMX request, display JavaScript missing warning
46      if request.headers.get("hx-request") != "true":
47          body = Markup(Path( *args: TEMPLATE_FOLDER, "partials", "no_htmx.html").read_text())
48          return templates.TemplateResponse(
49              name=f"base.html",
50              context={...},
55          )
56
57      if fake_backend.user_login_possible(email=email, password=password):
58          body_html = Path( *args: TEMPLATE_PARTIAL_FOLDER, "login_success.html").read_text()
59          template = Template(body_html)
60          session_id = str(uuid4())
61          request.app.state.sessions[session_id] = email
62          body = template.render(session_id=session_id)
63          return HTMLResponse(body, status_code: 200)
```

User Login Example - Backend Post - Success fragment

login_success.html

```html
<div id="login-area">

    Login successful.

    <!-- there's a 1 second delay here  -->
    <!-- change it to 15 seconds later to give you time to examine the html -->

    <div id="set-session"
        hx-trigger="load delay:1s"
        hx-on="htmx:beforeRequest: document.cookie = 'session_id={{ session_id }}; path=/;'"
        hx-get="/"
        hx-target="#login-area"
        hx-swap="outerHTML"
        hx-push-url="true"
    ></div>

</div>
```

User Login Example - Backend Post (OOB, Out of bound)

login_failed.html

```html
<div id="login-fail">
    <p style="...">Login unsuccessful. Please try again.</p>
</div>
```

```python
        return HTMLResponse(body, status_code=200)
    else:
        # allow body to contain html tags
        body = Markup(Path(*args: TEMPLATE_PARTIAL_FOLDER, "login_failed.html").read_text())

        # Note: alternative for this at https://htmx.org/extensions/response-targets/
        #       as of 2023.11.15 this doesn't yet work with 1.9.8, use 1.9.7 instead

        # originally requested: #login-area
        # but I can send the response somewhere else
        response = HTMLResponse(body, status_code=200)
        response.headers["HX-Retarget"] = "#login-fail"
        return response
```

User Login Example

Python Pizza HH 2023

localhost:9000/login

Please log in for Python Pizza:

Email:
admin@ppizzahh.de

Password:
•••••

Login unsuccessful. Please try again.

Note: This example uses temporary cookies.
Login

2. Live Updating Search Example

Updating Search Example

Python Pizza HH 2023

localhost:9000

Welcome at Python Pizza, admin@ppizzahh.de. (Logout)

Let's order some pizza!

Pizza search by toppings:
mozzar

PZ1 - The Classic Margherita
A timeless delight with ripe tomatoes, creamy mozzarella, and fresh basil on a crispy crust.
Toppings: Tomato, Mozzarella, Basil
Click to order this pizza.

PZ2 - Salami Simplicity
Classic and straightforward, this pizza offers a generous helping of salami paired with melted mozzarella.
Toppings: Salami, Mozzarella
Click to order this pizza.

PZ3 - Mushroom Melt

Updating Search Example

Python Pizza HH 2023

localhost:9000

Welcome at Python Pizza, admin@ppizzahh.de. (Logout)

Let's order some pizza!

Pizza search by toppings:
mozzar onio

PZ6 - Onion Overture
A melody of caramelized onions on a canvas of melted mozzarella.
Toppings: Onions, Mozzarella
Click to order this pizza.

Updating Search Example

```
19  </head>
20  <body>
21
22      <div id="main">
23      Welcome at Python Pizza, admin@ppizzahh.de.
24      <a href="" id="logout"
25         hx-trigger="click"
26         hx-get="logout"
27         hx-target="#main"
28         hx-swap="outerHTML"
29      >(Logout)</a>
30      <p/>Let's order some pizza!
31
32      <p/>
33      <label for="pizza-search">Pizza search by toppings:</label><br/>
34      <input type="text" name="pq" id="pizza-search"
35             hx-get="/search_pizza"
36             hx-trigger="keyup delay:500ms changed"
37             hx-target="#search-results"
38             placeholder="Search by typing toppings..."
39             autofocus
40             autocomplete="off"
41      >
42      <p/>
43      <div id="search-results">
44          Please enter your favorite topping(s) to select a pizza.
45      </div>
46
47  </div>
48
49  </body>
```

Updating Search Example (Backend)

search_pizza.html

```
<p/> {{ pizza.order_id }} - {{ pizza.name }}
<br/> {{ pizza.description }}
<br/> Toppings: {{ pizza.toppings }}
<br/>Click to <a href="" id="order-pizza-{{ pizza.order_id }}t"
        hx-trigger="click"
        hx-get="order_pizza/{{ pizza.order_id }}"
        hx-target="#main"
        hx-swap="outerHTML"
>order</a> this pizza.
```

Updating Search Example (Backend)

```
21
    Martin Borus *
22  @router.get( path: "/search_pizza", response_class=HTMLResponse, include_in_schema=False)
23  def search_pizza(request: Request, pq: str, session_id: str = Cookie(None)):
24      matched_pizzas = fake_backend.find_pizzas(
25          search_text=pq, max_results=MAX_SEARCH_RESULTS + 1
26      )
27
28      if not matched_pizzas:
29          return HTMLResponse("No pizza found. Try other ingredients.")
30
31      html_parts = []
32
33      template = Template(Path( *args: TEMPLATE_PARTIAL_FOLDER, "search_pizza.html").read_text())
34      for pizza in matched_pizzas[:MAX_SEARCH_RESULTS]:
35          html_parts.append(template.render(pizza=pizza))
36
37      if len(matched_pizzas) > MAX_SEARCH_RESULTS:
38          html_parts.append("<p/>There are more. Enter more search terms...")
39
40      body = "\n".join(html_parts)
41      return HTMLResponse(body)
42
```

3. Dashboards

Dashboards - Server Side Events (SSE)

Python Pizza HH 2023

localhost:9000

Thank you, admin@ppizzahh.de

Your order is "Onion Overture".
Toppings: Onions, Mozzarella

Please wait while we make it. Or order more.

This is the current kitchen status:

Customer: admin@ppizzahh.de
Pizza ordered: "Onion Overture"
Preparition status: Prepared.
Preparation time: 1:13

The current time is 18:15:11

Dashboards - Server Side Events (SSE)

You need to import the “sse.js” extension

```
<script src="https://unpkg.com/htmx.org@1.9.8"></script>
<script src="https://unpkg.com/htmx.org/dist/ext/sse.js"></script>
```

The backend needs to support SSE - FastAPI uses starlette and does it.

```
from sse_starlette.sse import EventSourceResponse
```

Then you can use SSE via htmx.

dashboard.html

```
<div hx-ext="sse" sse-connect="/kitchen-dashboard" sse-swap="message">
    ...
</div>
```

Dashboards - Server Side Events (SSE)

The Jinja2 dashboard template for an order is like this

dashboard_job.html

```
Customer: {{ email }}
<br/>Pizza ordered: "{{ pizza.name }}"
<br/>Preparition status: {{ status }}
<br/>Preparation time: {{ age }}
<p/>
```

Dashboards - Server Side Events (SSE)

```
Martin Borus *
22  @router.get("/kitchen-dashboard")
23  async def message_stream(request: Request, session_id: str = Cookie(None)):
        Martin Borus *
24      async def event_generator():
25          template = Template(
26              Path( *args: TEMPLATE_PARTIAL_FOLDER, "dashboard_job.html").read_text()
27          )
28
29          while True:
30              # If client closes connection, stop sending events
31              if await request.is_disconnected():
32                  break
33
34              current_jobs = []
35  >           ...
37              for order in request.app.state.orders:
38                  # templating pizza order html here
39  >               ...
49                  current_jobs.append(order_html)
50
```

Dashboards - Server Side Events (SSE)

```
        # while loop continues

        if current_jobs:
            body = "\n".join(current_jobs)
        else:
            body = "There are no pizzas ordered at the moment."

        # The time is updated as an out of bound element
        oob_body = (
            '<span id="current-time" hx-swap-oob="true">'
            f"{datetime.datetime.now():%H:%M:%S}"
            "</span>"
        )
        yield oob_body + body

        await asyncio.sleep(1)

    return EventSourceResponse(event_generator())
```

Dashboards - Server Side Events (SSE)

Python Pizza HH 2023

DevTools - localhost:9000/login

Elements Console Sources Network Performance Memory Application Lighthouse Recorder

Thank you, admin@ppizza

Your order is "Onion Overt
Toppings: Onions, Mozzar

Please wait while we make

This is the current kitchen

Customer: admin@ppizzah
Pizza ordered: "Onion Ove
Preparition status: Baking.
Preparation time: 8:56

The current time is 18:22:5

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▼<div id="main">
     " Thank you, admin@ppizzahh.de "
    ▶<p>…</p>
    ▶<p>…</p>
    ▶<p>…</p>
    ▼<div hx-ext="sse" sse-connect="/kitchen-dashboard" sse-swap="message" class>
       "Customer: admin@ppizzahh.de "
       <br>
       "Pizza ordered: "Onion Overture" "
       <br>
       "Preparition status: Baking. "
       <br>
       "Preparation time: 8:56 "
       <p></p>
     </div>
    ▼<p> == $0
       " The current time is "
       <span id="current-time" hx-swap-oob="true">18:22:54</span>
     </p>
```

Styles Computed Lay

Filter

```
element.style {
}
p {
  display: block;
  margin-block-start:
  margin-block-end: 1
  margin-inline-start:
  margin-inline-end:
}
```

margin
border
padding
126

Bonus Content

(if there’s time)

Timeouts and Errors

You can use the hx-request attribute to set a timeout

```
<div ... hx-request='\"timeout\":10000' >

<div ... hx-request="js: timeout:10000 " >
```

Timeouts and Errors

You can catch errors and run your own JavaScript code on error

```
<div id="..."
    hx-on="htmx:onLoadError: myjsfunction()
           htmx:responseError: myjsfunction()
           htmx:timeout: myjsfunction()"
>
```

Note: There’s an upcoming syntax change for all hx-on attributes.

```
hx-on:htmx:on-load-error="myjsfunction()"
hx-on:htmx:response-error="myjsfunction()"
hx-on:htmx:timeout="myjsfunction()"
```

Final thoughts.

Thanks for your attention.

Get the code here:

https://github.com/mborus/choose_htmx

Find me here:

linkedin.com/in/mborus-de

github.com/mborus

mastodon.social/@mborus

Get HTMX here:

https://htmx.org

Source for “Confused” Image: ChatGPT