Welcome

About Me

- Using Django since 2018
- PSF Contributing member
- Lead dev for a Django-based commercial application for utilities districts
- Part of HTMX community since 2020

Server-Side is Dead! Long Live Server-Side (+ HTMX)

Jack Linke

Web 2.0

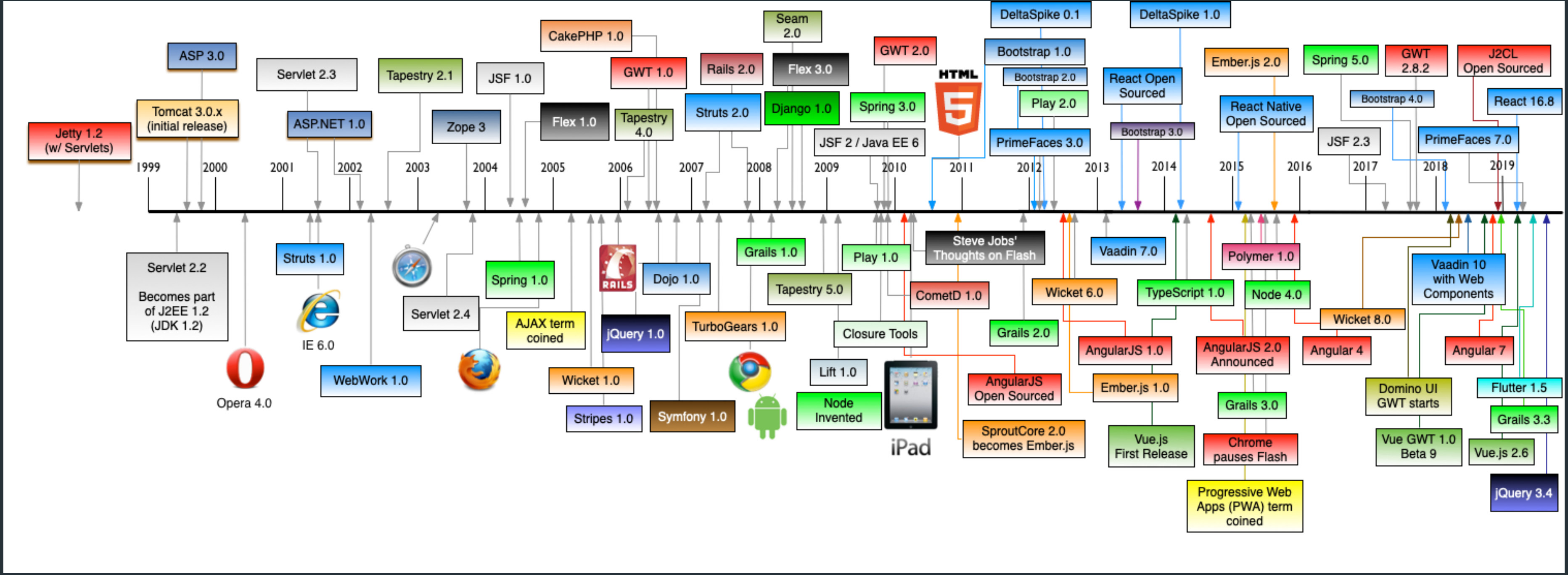
- Dynamic content
- User-generated
- Interactive & ‘social’
- Web server & database on the back-end, serving html

The current state

Django templates

- How to apply interactivity?
  - Vanilla JavaScript
  - jQuery
  - Single-Page Applications
  - Use Vue directly

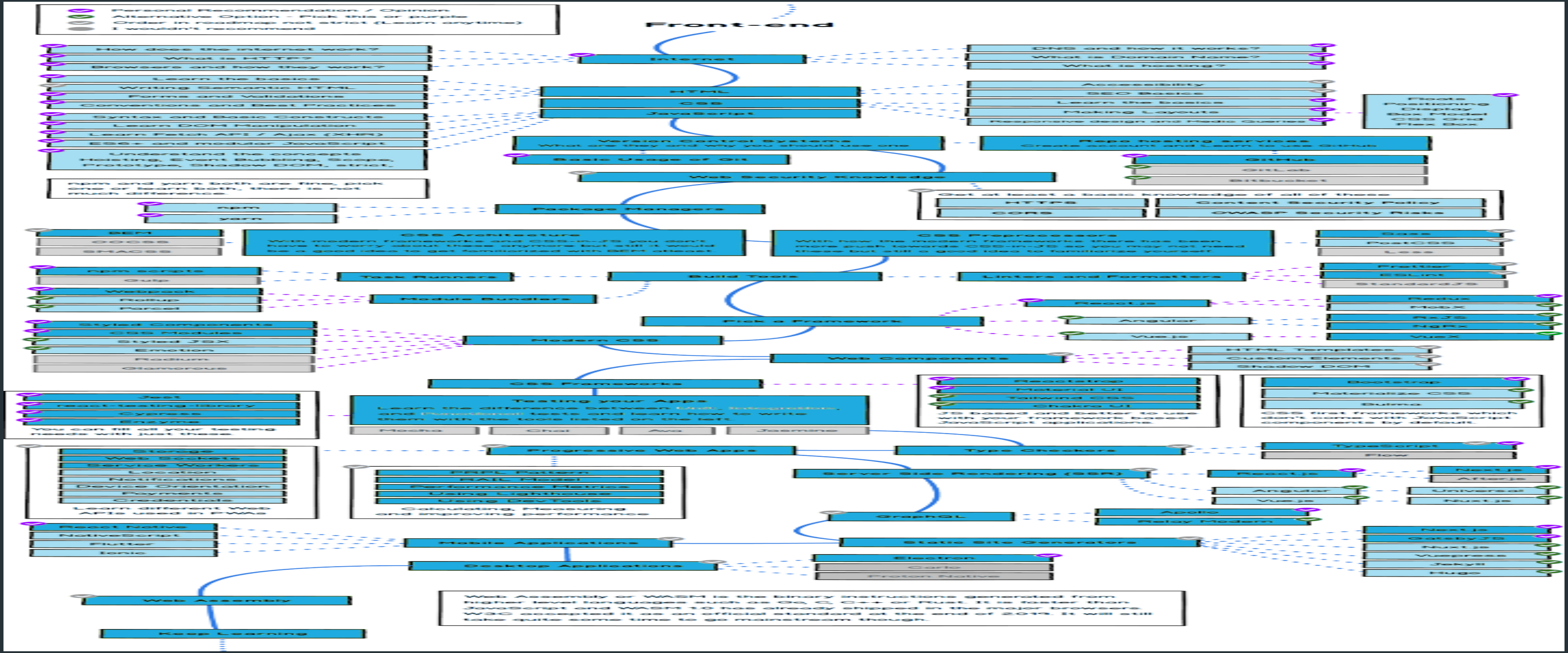
Frameworks for days



@JackDLinke

https://github.com/mraible/history-of-w

Front-End Development Roadmap



Personal Recommendation / Opinion
Alternative Option - Pick this or purple
Order in roadmap not strict (Learn anytime)
I wouldn't recommend
Front-end
How does the internet work?
What is HTTP?
Browsers and how they work?
Internet
DNS and how it works?
What is Domain Name?
What is hosting?
Learn the basics
Writing Semantic HTML
Forms and Validations
Conventions and Best Practices
HTML
CSS
JavaScript
Accessibility
SEO Basics
Learn the basics
Making Layouts
Responsive design and Media Queries
Floats
Positioning
Display
Box Model
CSS Grid
Flex Box
Syntax and Basic Constructs
Learn DOM Manipulation
Learn Fetch API / Ajax (XHR)
ES6+ and modular JavaScript
Understand the concepts Hoisting, Event Bubbling, Scope, Prototype, Shadow DOM, strict,
Version Control Systems
What are they and why you should use one
Basic Usage of Git
Repo hosting services
Create account and Learn to use GitHub
GitHub
GitLab
Bitbucket
Web Security Knowledge
Get at least a basic knowledge of all of these
HTTPS
Content Security Policy
CORS
OWASP Security Risks
npm and yarn both are fine, pick one or learn both, there is not much difference
npm
yarn
Package Managers
BEM
OOCSS
SMACSS
CSS Architecture
With modern frameworks and CSS-in-JS you don't have to worry about these anymore but still it would be a good idea to familiarize with BEM atleast.
CSS Preprocessors
With how the modern frameworks there has been more push towards CSS-in-JS so you may not need these but still a good idea to familiarize yourself
Sass
PostCSS
Less
npm scripts
Gulp
Task Runners
Build Tools
Linters and Formatters
Prettier
ESLint
StandardJS
Webpack
Rollup
Parcel
Module Bundlers
React.js
Angular
Vue.js
Pick a Framework
Redux
MobX
RxJS
NgRx
VueX
Styled Components
CSS Modules
Styled JSX
Emotion
Radium
Glamorous
Modern CSS
Web Components
HTML Templates
Custom Elements
Shadow DOM
CSS Frameworks
Reactstrap
Material UI
Tailwind CSS
Chakra UI
JS based and better to use with your framework based JavaScript applications.
Bootstrap
Materialize CSS
Bulma
CSS first frameworks which don't come with JavaScript components by default.
Jest
react-testing-library
Cypress
Enzyme
You can fill all your testing needs with just these
Testing your Apps
Learn the difference between Unit, Integration, and Functional tests and learn how to write them with the tools listed on the left.
Mocha
Chai
Ava
Jasmine
Progressive Web Apps
Type Checkers
TypeScript
Flow
Storage
Web Sockets
Service Workers
Location
Notifications
Device Orientation
Payments
Credentials
Learn different Web APIs used in PWAs
PRPL Pattern
RAIL Model
Performance Metrics
Using Lighthouse
Using DevTools
Calculating, Measuring and improving performance
Server Side Rendering (SSR)
React.js
Angular
Vue.js
Next.js
After.js
Universal
Nuxt.js
GraphQL
Apollo
Relay Modern
React Native
NativeScript
Flutter
Ionic
Mobile Applications
Static Site Generators
Next.js
GatsbyJS
Nuxt.js
Vuepress
Jekyll
Hugo
Desktop Applications
Electron
Carlo
Proton Native
Web Assembly
Web Assembly or WASM is the binary instructions generated from higher level languages such as Go, C, C++ or Rust. It is faster than JavaScript and WASM 1.0 has already shipped in the major browsers. W3C accepted it as an official standard at the end of 2019. It will still take quite some time to go mainstream though.
Keep Learning

@JackDLinke

https://www.decipherzone.com/blog-detail/front-e

Django templates

- So, what is the ‘standard’ Django approach for *modern* front-end development?

Django templates

- So, what is the ‘standard’ Django approach for *modern* front-end development?

? ? ?

One potential approach - HTMX is…

An extension to existing HTML

Backend agnostic (bring-your-own-backend)

Focused - does a few things very well

Allows you to access

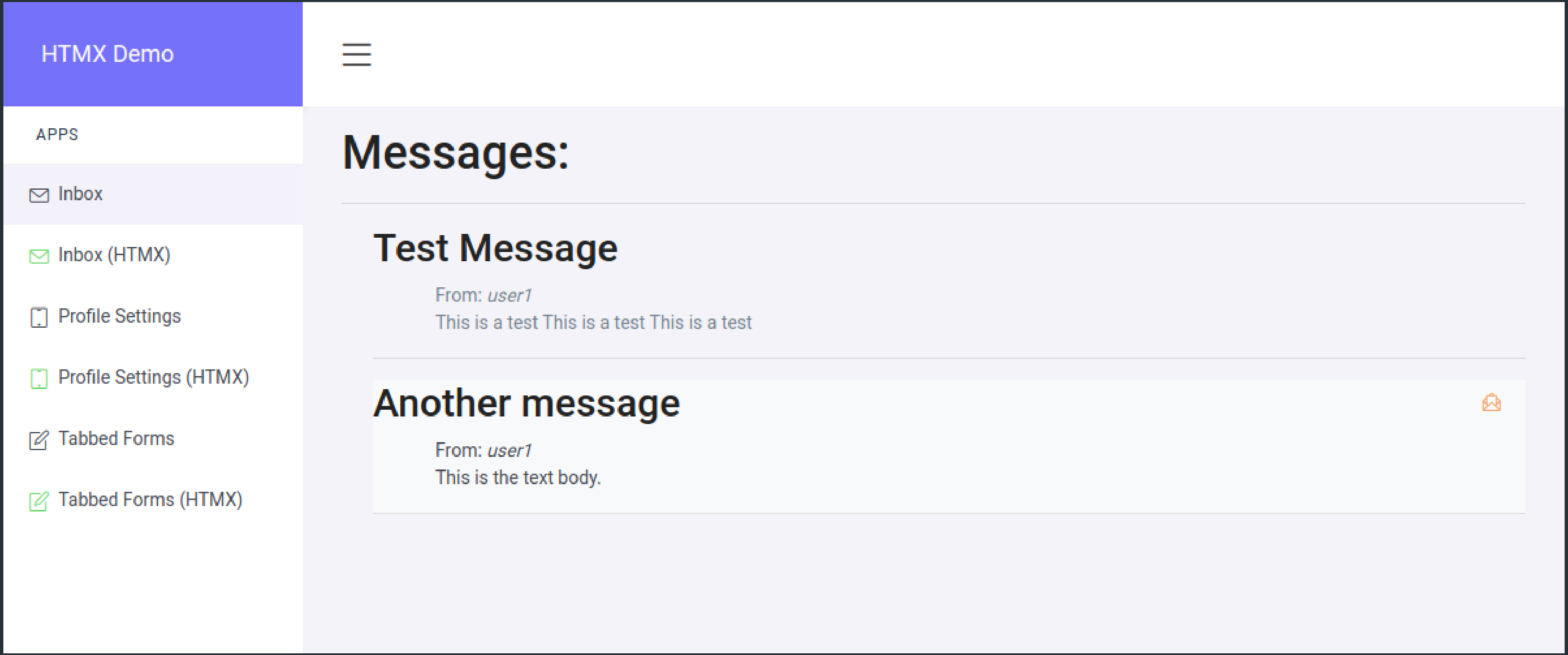
AJAX, CSS Transitions, WebSockets and Server Sent Events

Directly in your HTML

Small - 11 kB

Feature and approach walkthroughs

Inbox functionality (marking read, archive)



Messages:

Test Message

From: *user1*
This is a test This is a test This is a test

Another message

From: *user1*
This is the text body.

Inbox functionality (marking read, archive)

```
def inbox(request):
    template = "messaging/inbox.html"

    if request.method == "POST":
        message_id = request.POST.get('item')
        message = get_object_or_404(Message, id=message_id)
        if message.to_user == request.user:
            message.mark_read()






    messages = Message.objects.filter(to_user=request.user)
    context = {'messages': messages}


    return TemplateResponse(request, template, context)
```

```
def inbox_htmx(request):
    template = "messaging/inbox_htmx.html"

    if request.method == "POST":
        message_id = request.POST.get('item')
        message = get_object_or_404(Message, id=message_id)
        if message.to_user == request.user:
            message.mark_read()

        template = "messaging/fragments/message_fragment.html"
        context = {'message': message}
        return TemplateResponse(request, template, context)


    messages = Message.objects.filter(to_user=request.user)
    context = {'messages': messages}


    return TemplateResponse(request, template, context)
```

Inbox functionality (marking read, archive)

```
<form method="POST">
    {% csrf_token %}
    <input type="hidden" name="item" value="{{ message.id }}">
    <button
        class="btn mb-1 ml-2 btn-rounded btn-outline-light"
        type="submit"
        title="Mark Read">
        <i class="icon-envelope-letter menu-icon text-warning"></i>
    </button>
</form>
```

```
<div class="ml-4 {% if not message.read_at %} bg-light{% endif %}" id="li_list_{{ message.id }}">
    ...
            <input type="hidden" name="item" value="{{ message.id }}">
            <span
                data-hx-post="{% url 'messaging:inbox_htmx' %}"
                data-hx-swap="outerHTML"
                data-hx-target="#li_list_{{ message.id }}"
                data-hx-include="[name='item']"
                class="btn mb-1 ml-2 btn-rounded btn-outline-light" type="submit" title="Mark Read">
                <i class="icon-envelope-letter menu-icon text-warning"></i>
            </span>
    ...
</div>
```

Inbox functionality (marking read, archive)

- Uses the same template fragment for
  - Initial display of each message
  - Swapped content

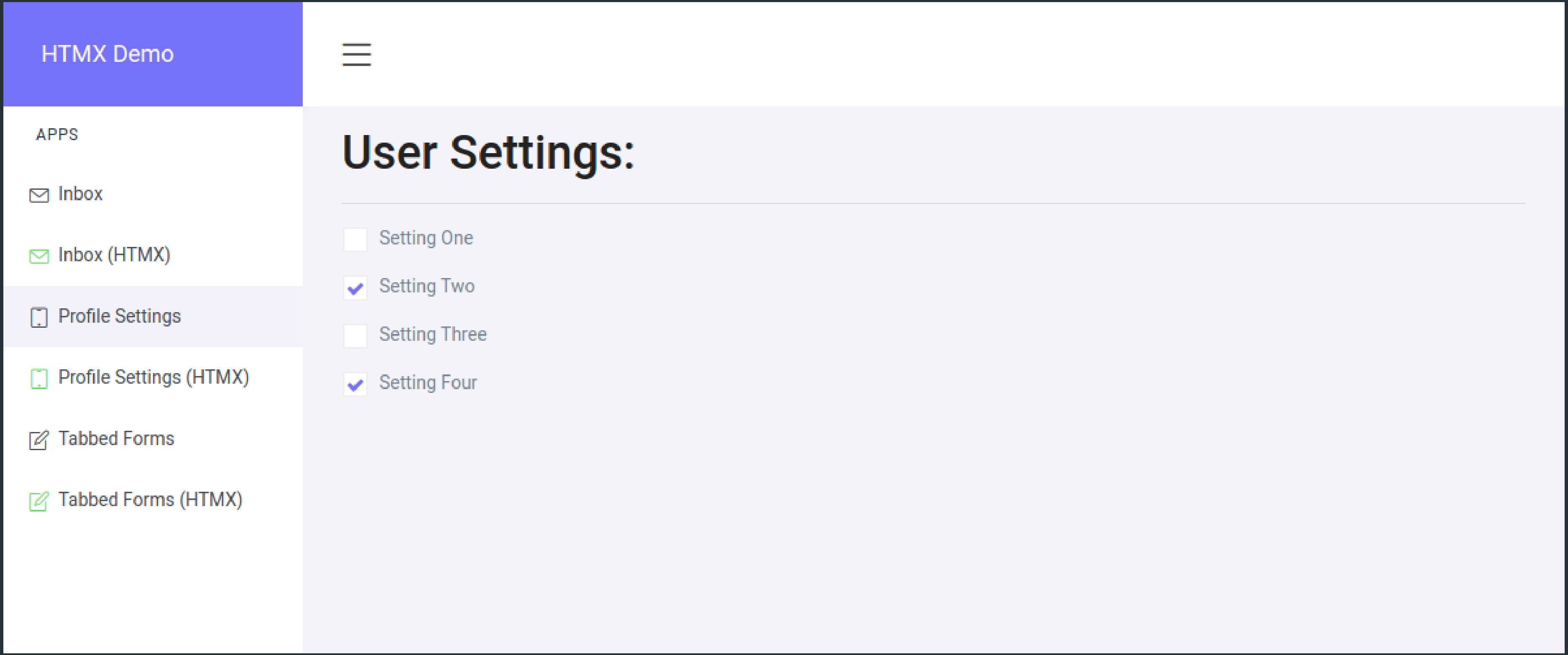
```
{% for message in messages %}
    {% include "messaging/fragments/message_fragment.html" %}
{% endfor %}
```

- Swapped content replaces original content

Inbox functionality (marking read, archive)

Demo

One-click settings changes



User Settings:

- [ ] Setting One
- [✓] Setting Two
- [ ] Setting Three
- [✓] Setting Four

@JackDLinke

One-click settings changes

- Initial example uses ajax
- Includes small amount of _*hyperscript* for animation
- Example of returning HTML directly from view vice fragment
- Loading the response to different location (div)

One-click settings changes

```
<div class="custom-control custom-switch col-lg-3 mt-3 mb-1">
    <form method="POST">
        <input type="checkbox"
            class="form-check-input"
            {% if request.user.setting_one == True %}checked{% endif %}
            name="set_value"
            id="setting_one">
        <label class="form-check-label" for="setting_one">Setting One</label>
        {% csrf_token %}
    </form>
</div>
```

```
$("#setting_one").click(function(){
    var csrfToken = $( "input[name='csrfmiddlewaretoken']");
    $.ajax({
        url: '{% url "users:settings" %}',
        type: "POST",
        dataType: "json",
        data: {
            'settings':'setting_one',
            'csrfmiddlewaretoken':csrfToken.val()
        },
        cache: false
    }).done(function(data) {
        if (data.result === true){
            alert(data.message);
        }
    });
});
```

One-click settings changes

```
<div class="custom-control custom-switch col-lg-3 mt-3 mb-1">
    <form method="POST">
        <input type="checkbox"
            class="form-check-input"
            {% if request.user.setting_one == True %}checked{% endif %}
            name="set_value"
            id="setting_one"
            data-hx-post="{% url "users:settings_htmx" %}"
            data-hx-trigger="click"
            data-hx-target="#setting_one_response">
            data-data-hx-include="[name='settings']"
        <label class="form-check-label" for="setting_one">Setting One</label>
        <input type="hidden" name="settings" value="setting_one">
    </form>
    <div id="setting_one_response"></div>
</div>
```

One-click settings changes

```
def settings(request):
    template = "users/settings.html"
    context = {}

    if request.method == "POST":
        user = request.user

        setting_value = request.POST.get('settings')

        # Set/un-set settings booleans
        if setting_value == "setting_one":
            request.user.toggle_setting_one()

        # ...

    return TemplateResponse(request, template, context)
```

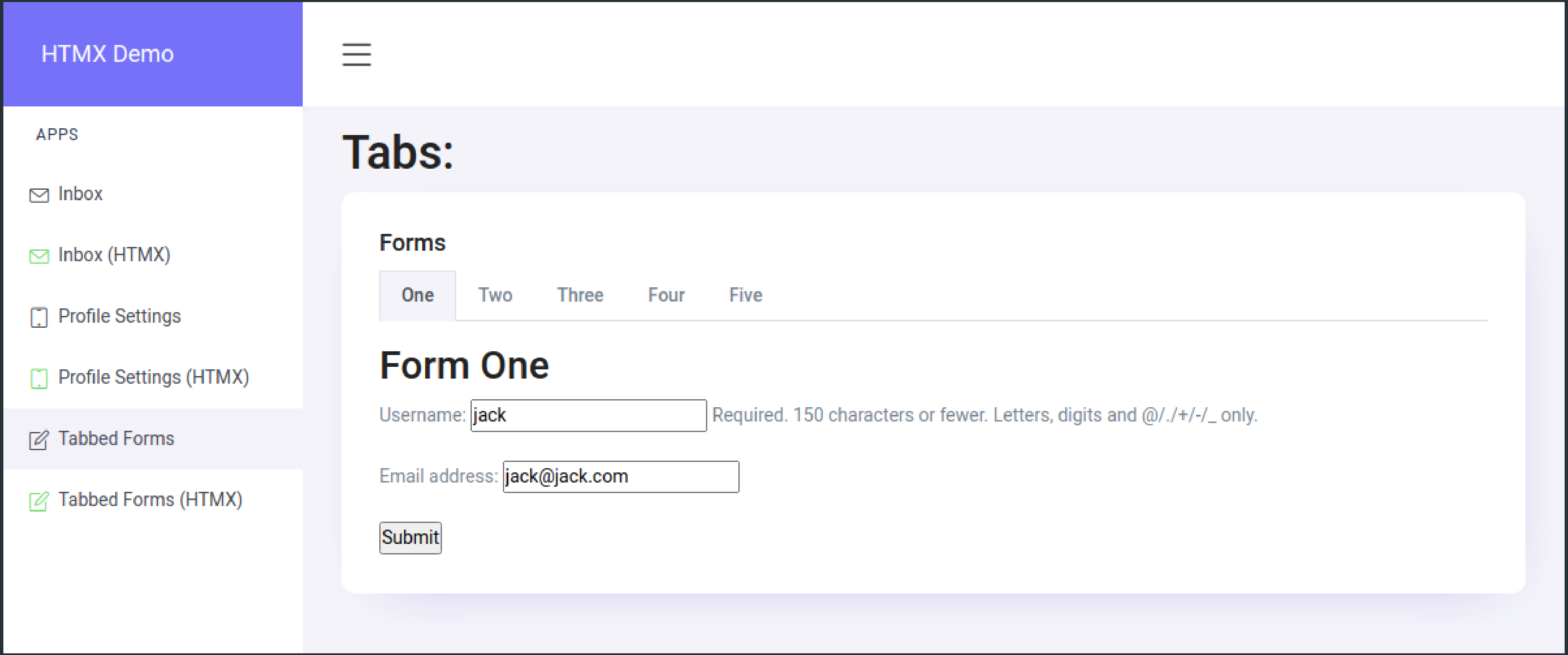
One-click settings changes

```
def settings_htmx(request):
    # ...

        if setting_value == "setting_one":
            request.user.toggle_setting_one()
            successful_toggle = True

        # ...

        if successful_toggle:
            return HttpResponse(
                (
                    '<div _="on load wait 2s then remove me" '
                    '    class="alert alert-success alert-dismissible fade show" '
                    '    role="alert">'
                    '<i class="icon-like menu-icon text-success"></i>'
                    '</div>'
                ),
                status=200,
                content_type="text/html",
            )

    return TemplateResponse(request, template, context)
```

One-click settings changes

Demo

Forms in multiple tabs



Tabs:

Forms

One | Two | Three | Four | Five

Form One

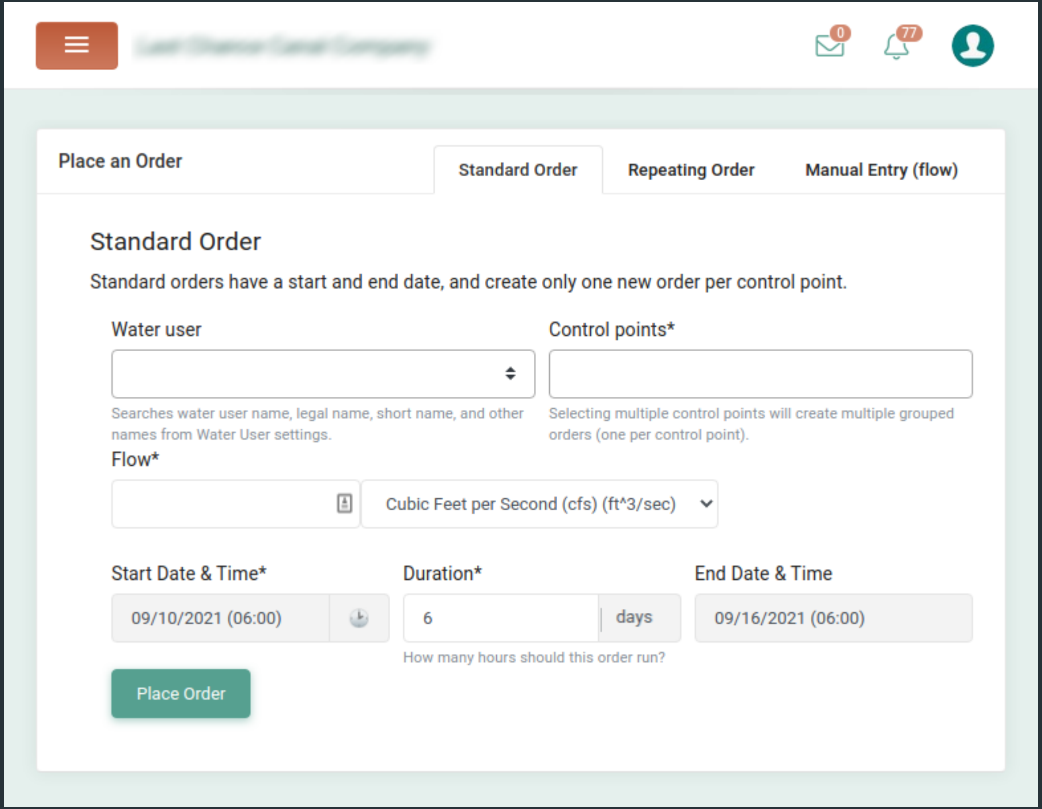
Username: jack Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address: jack@jack.com

Submit

Forms in multiple tabs

- A pattern I use often
  - Profile settings
  - Tenant configuration
  - Order forms variations



Place an Order

Standard Order

Standard orders have a start and end date, and create only one new order per control point.

Water user:

Searches water user name, legal name, short name, and other names from Water User settings.

Control points*:

Selecting multiple control points will create multiple grouped orders (one per control point).

Flow*: Cubic Feet per Second (cfs) (ft^3/sec)

Start Date & Time*: 09/10/2021 (06:00)

Duration*: 6 days

How many hours should this order run?

End Date & Time: 09/16/2021 (06:00)

Forms in multiple tabs

```python
class FormOne(forms.Form):
    username = forms.CharField(max_length=100)
    email = forms.EmailField(max_length=100)
```

... additional forms

Forms in multiple tabs

```
<div class="default-tab">
    <ul class="nav nav-tabs mb-3" role="tablist">
        <li class="nav-item">
            <a class="nav-link active" data-toggle="tab" href="#one">One</a>
        </li>
        ... remaining tabs
    </ul>
    <div class="tab-content">
        <div class="tab-pane fade show active" id="one" role="tabpanel">
            <h2>Form One</h2>
            <div class="p-t-15">
                <form method="post">
                    {% csrf_token %}
                    {{ form_one.as_p }}
                    <input type="submit" value="Submit" name="submit_one">
                </form>
            </div>
        </div>
        ... remaining tab contents
    </div>
</div>
```

Forms in multiple tabs

```
def tabs_view(request):
    template = "tabs/tabs.html"
    context = {}

    post_data = request.POST or None

    # Set defaults
    form_one = FormOne(
        prefix="form_one",
        initial={
            "username": request.user.username,
            "email": request.user.email
        }
    )
    form_two = FormTwo(
        prefix="form_two",
        initial={
            "username": request.user.username,
            "email": request.user.email
        }
    )

    # ... remaining forms

    context = {
        "form_one": form_one,
        "form_two": form_two,
        # ... remaining forms
    }

    return TemplateResponse(request, template, context)
```

Forms in multiple tabs

- Issues with the traditional approach
  - Prefixes
  - Long views
  - All content must load each time (e.g. selects)

Forms in multiple tabs

```
def tabs_view(request):
    template = "tabs/tabs.html"
    context = {}

    post_data = request.POST or None

    # Set defaults
    form_one = FormOne(
        prefix="form_one",
        initial={
            "username": request.user.username,
            "email": request.user.email
        }
    )
    # ... remaining forms

if request.method == 'POST':
    # For each form, if the submit button name is in the POST,
    # process and save that form
    if "submit_one" in request.POST:
        form_one = FormOne(
            post_data, prefix="form_one"
        )
        if form_one.is_valid():
            context["success"] = "Success!"


    context = {
        "form_one": form_one,
        "form_two": form_two,
        # ... remaining forms
    }

    return TemplateResponse(request, template, context)
```

Forms in multiple tabs

- HTMX Approach
  - Multiple tabs, one div for content
  - On-load, populate div with tab 1 content
  - Each tab’s content loads independently

Forms in multiple tabs

```
<div
   id="tabs"
   data-hx-get="{% url 'tabs:form_one' %}"
   data-hx-trigger="load delay:100ms"
   data-hx-target="#tabContent"
   data-hx-swap="innerHTML">
</div>

<li class="nav-item">
   <a href="#"
      data-toggle="tab"
      role="tab"
      data-hx-get="{% url 'tabs:form_one' %}"
      data-hx-target="#tabContent"
      data-hx-swap="innerHTML"
      class="nav-link active">One
   </a>
</li>
... remaining tabs

<div class="tab-pane fade show active" id="tabContent" role="tabpanel">
   Load...
</div>
```

*Tab Contents in separate *.html*

```
<div id="formDiv">
   <h2>Form One</h2>
   <p>
      <h3 class="text-success">{{ success }}</h3>
   </p>

   <form
      data-hx-post="{% url 'tabs:form_one' %}"
      data-hx-target="#formDiv"
      data-hx-swap="outerHTML">
      <div>
         {{ form_one.as_p }}
      </div>
      <input type="submit" value="Submit">
   </form>
</div>
```

@JackDLinke

Forms in multiple tabs

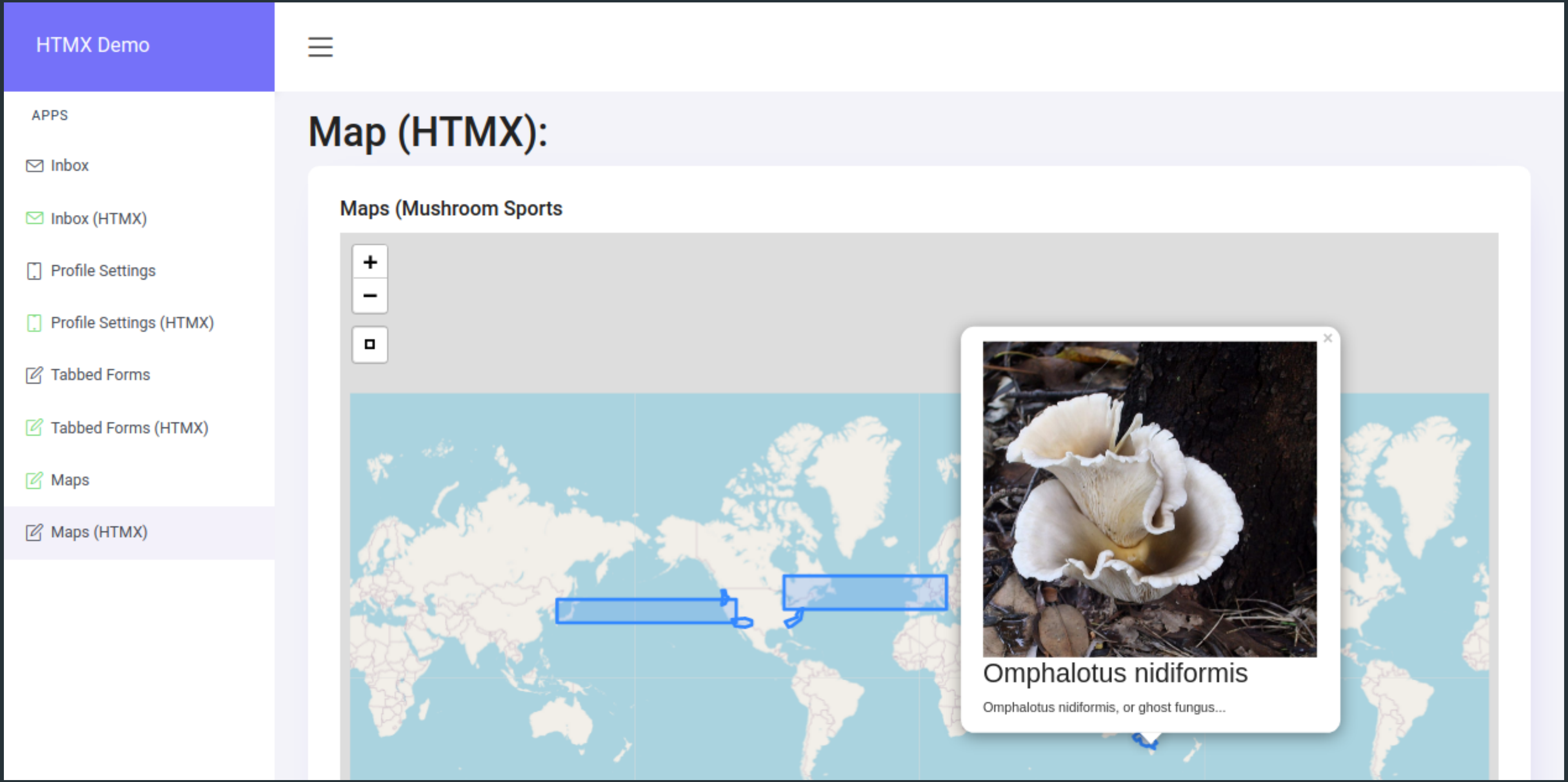
```
def tabs_htmx_view(request):
    template = "tabs/tabs_htmx.html"
    context = {}

    return TemplateResponse(request, template, context)


def form_one_htmx_view(request):
    template = "tabs/fragments/form_one.html"
    context = {}
    form_one = FormOne(
        initial={
            "username": request.user.username,
            "email": request.user.email
        }
    )

    if request.method == 'POST':
        form_one = FormOne(request.POST)
        if form_one.is_valid():
            context["success"] = "Success!"

    context["form_one"] = form_one

    return TemplateResponse(request, template, context)
```

Forms in multiple tabs

Demo

Lazy Data Popovers
(in maps, datatables, etc)

Lazy Data Popovers



Map (HTMX):

Maps (Mushroom Sports

Omphalotus nidiformis

Omphalotus nidiformis, or ghost fungus...

@JackDLinke

Lazy Data Popovers

```
urlpatterns = [
    path('', TemplateView.as_view(template_name='maps/map.html'), name='map'),
    path(
            'Data.geojson',
            GeoJSONLayerView.as_view(model=MushroomSpot, properties=('title', 'description', 'picture_url', 'id')),
            name='data'
    ),
]
```

Lazy Data Popovers

```
<script>
    var dataurl = '{% url "maps:data" %}';

    window.addEventListener("map:init", function (event) {
        var map = event.detail.map;
        // Download GeoJSON data with Ajax
        fetch(dataurl)
        .then(function(resp) {
            return resp.json();
        })
        .then(function(data) {
            L.geoJson(data, {
            onEachFeature: function onEachFeature(feature, layer) {
                var props = feature.properties;
                var content = `<img width="300" src="${props.picture_url}"/><h3>${props.title}</h3><p>${props.description}</p>`;
                layer.bindPopup(content);
            }
        }).addTo(map);
        });
    });
</script>
```

Lazy Data Popovers

- Initial Approach
  - All data for the map features & popovers loaded with initial page

Lazy Data Popovers

```
urlpatterns = [
    path('htmx', TemplateView.as_view(template_name='maps/map_htmx.html'), name='map_htmx'),
    path(
            'Data.geojson',
            GeoJSONLayerView.as_view(model=MushroomSpot, properties=('title', 'description', 'picture_url', 'id')),
            name='data'
    ),
    path("<mushroom_id>/", mushroom_data, name="mushroom_data"),
]
```

Lazy Data Popovers

```python
def mushroom_data(request, mushroom_id):
    template = "maps/fragments/data.html"
    context = {}

    try:
        mushroom = MushroomSpot.objects.get(id=mushroom_id)
        context["mushroom"] = mushroom

    except MushroomSpot.DoesNotExist:
        pass

    return TemplateResponse(request, template, context)
```

Lazy Data Popovers

```
<script>
    var dataurl = '{% url "maps:data" %}';

    function layerOnClick() {
        htmx.process(document.body);
    }

    window.addEventListener("map:init", function (event) {
        var map = event.detail.map;
        // Download GeoJSON data with Ajax
        fetch(dataurl)
        .then(function(resp) {
            return resp.json();
        })
        .then(function(data) {
            L.geoJson(data, {
            onEachFeature: function onEachFeature(feature, layer) {
                var props = feature.properties;
                var content = `
                    <div data-hx-get="/maps/${props.id}/"
                        data-hx-trigger="load"
                        data-hx-target="#data-div-${props.id}"
                        data-hx-swap="outerHTML">
                    </div>
                    <div style="width: 600px" id="data-div-${props.id}"></div>
                `;
                layer.bindPopup(content);
                layer.on('click', layerOnClick);
            }}).addTo(map);
        });
    });
</script>
```

```
# Popover contents in separate html file

{% load static %}
<div style="width: 600px">
    <img width="300" src="{{ mushroom.picture_url }}"/>
    <h3>{{ mushroom.title }}</h3>

    <div style="width: 500px">
        {{ mushroom.description }}
    </div>
</div>
```

@JackDLinke

Lazy Data Popovers

- HTMX Approach
  - Load the map features, but only load the content for each popover when the associated map feature has been clicked.

Lazy Data Popovers

Demo

Tips, best practices, and resources

CSRF Tokens - Inline

Cross-Site Request Forgery

“An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated”

- Django has built-in support
  - Provides a token from back-end
  - Compares to submitted POST information

CSRF Tokens - Inline

```
<div id="htmx-test-div"
   data-hx-post="{% url 'save-like' %}"
   data-hx-headers='{"X-CSRFToken": "{{ csrf_token }}"}'
   data-hx-target="#htmx-test-div"
   data-hx-include="[name='my-data']"
   data-hx-swap="outerHTML">

   <input type="hidden" value="Some Value to POST" name="my-data">
</div>
```

CSRF Tokens - JavaScript Snippet

```
<script>
    // htmx csrf script
    document.body.addEventListener('htmx:configRequest', (event) => {
        event.detail.headers['X-CSRFToken'] = '{{ csrf_token }}';
    })
</script>
```

More Views

Complimentary JS Libraries

- _Hyperscript
  - Built by creator of HTMX
  - Designed to work alongside HTMX
  - Speculative
- Alpine.js
  - Lightweight & focused

django-htmx

- Developed by Adam Johnson
- Provides:
  - Debug Handler when settings.DEBUG is True
  - Boolean for determining if partial or full refresh

```
def my_view(request):
    if request.htmx:
        template_name = "partial.html"
    else:
        template_name = "complete.html"
    return render(template_name, ...)
```

  - Many other useful utilities

Additional Resources

- htmx.org
- awesome-htmx
  - https://github.com/rajasegar/awesome-htmx
- HTMX Discord
  - https://htmx.org/discord
- r/htmx on Reddit
- Thomas Güttler's Best Practices
  - https://github.com/guettli/django-htmx-fun
- Notes & code for this presentation
  - https://github.com/jacklinke/htmx-talk-2021

Thank you

Jack Linke
@JackDLinke
jacklinke.com