You might not need a frontend framework

DjangoCon EU 2021

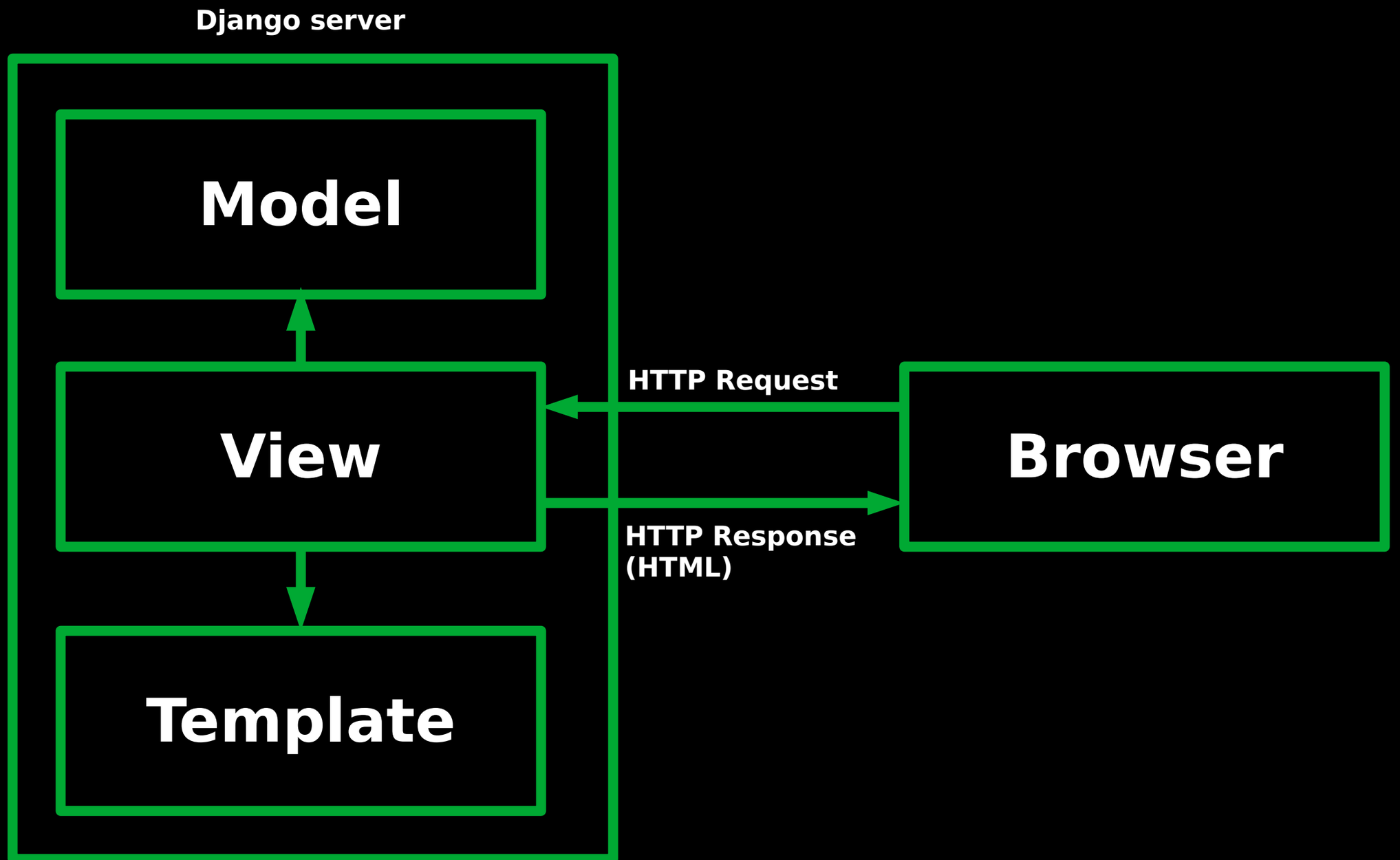
Afonso Cerejeira

DevOps engineer @ Collabora

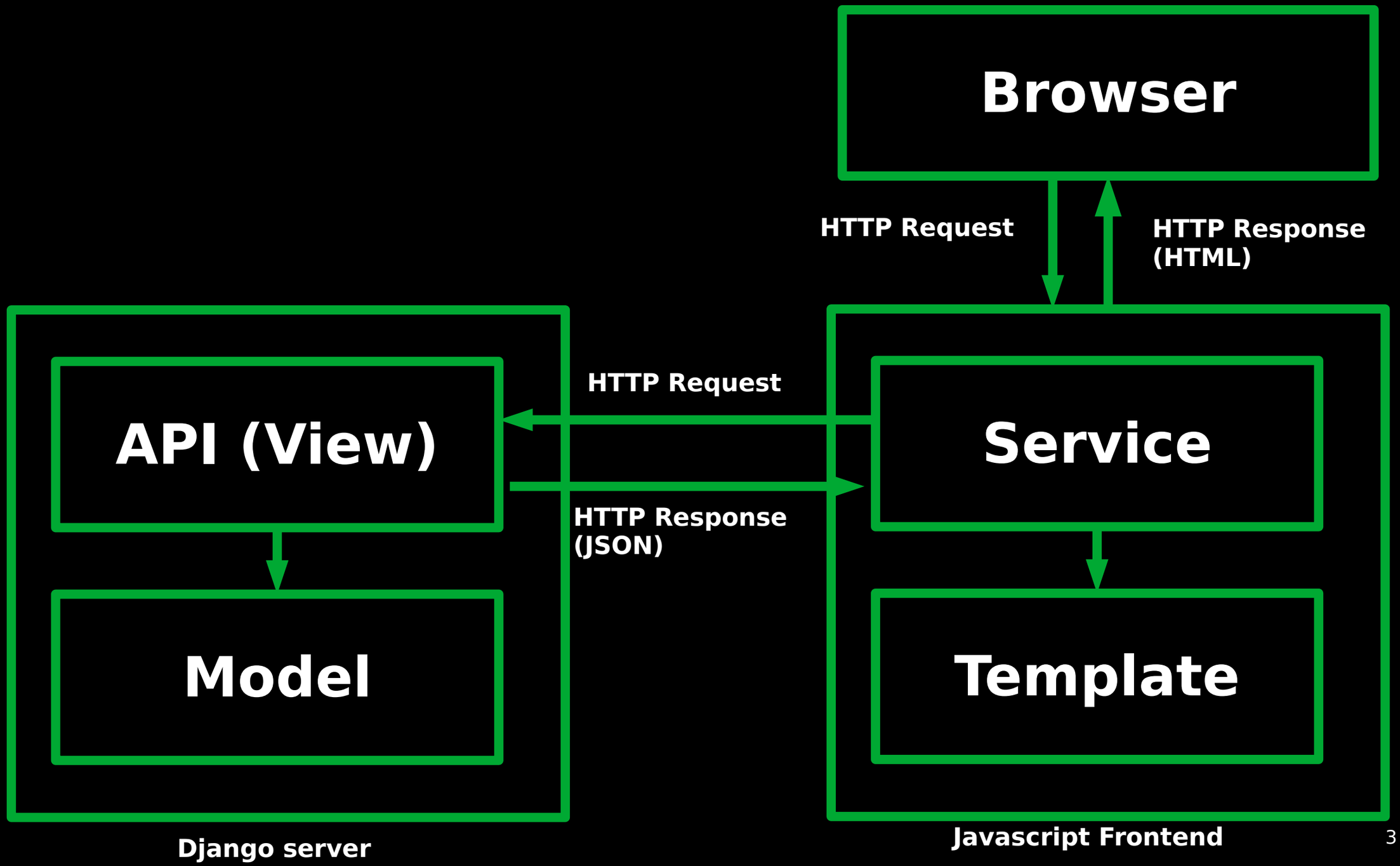
afonso.cerejeira@collabora.com

Open First

Django SSR architecture



Django SPA architecture

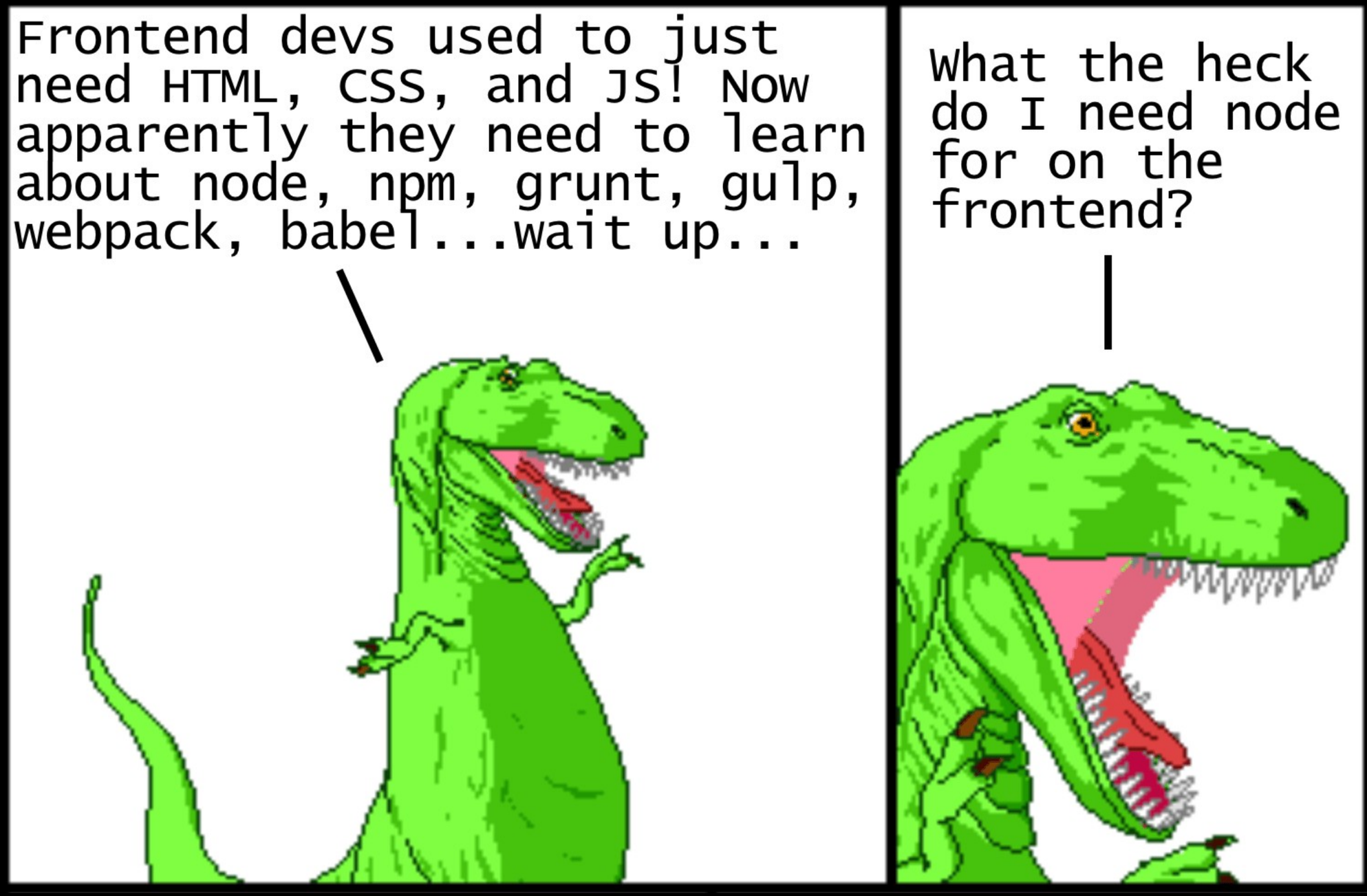


Benefits of Django + SPA

- Clear separation between frontend and backend
- Modern UX with better reactivity and no page reloads
- API can be consumed by other clients (e.g, mobile apps)
- Easy to deploy the frontend
- Access to the whole Node.js ecosystem

Drawbacks of Django + SPA

- Another programming language stack to learn:
  - node.js
  - npm
  - webpack
  - babel
  - React/Vue/Angular
- Requires Javascript for the whole site
- Context switching between Python and Javascript



https://peterxjang.com/blog/modern-javascript-explained-for-dinosaurs.html

Problems with Django + SPA

- Requires creating a (REST) API
- No Django forms (rendering and validation)
- No Django templates (filters, includes, blocks)
- Increased project complexity
- Harder to test
- Can affect accessibility and SEO
- Single repository vs multiple projects challenges

Do we really need a full Javascript frontend?
(it depends)

Progressive enhancement

- Usable without Javascript
- Better UX when Javascript is enabled
- Use conventional Django patterns
- Lightweight frontend

Example: “talks” app

- CRUD talks management app
- github.com/ajcerejeira/talksapp/
- talksapp.herokuapp.com/

Example: “talks” app

```
# talks/models.py
class Talk(models.Model):
    title = models.CharField(max_length=200, unique=True)
    description = models.TextField(blank=True)
    speaker = models.ForeignKey(User, on_delete=models.CASCADE)

    class Meta:
        verbose_name = "talk"
        verbose_name_plural = "talks"

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse("talk_detail", kwargs={"pk": self.pk})
```

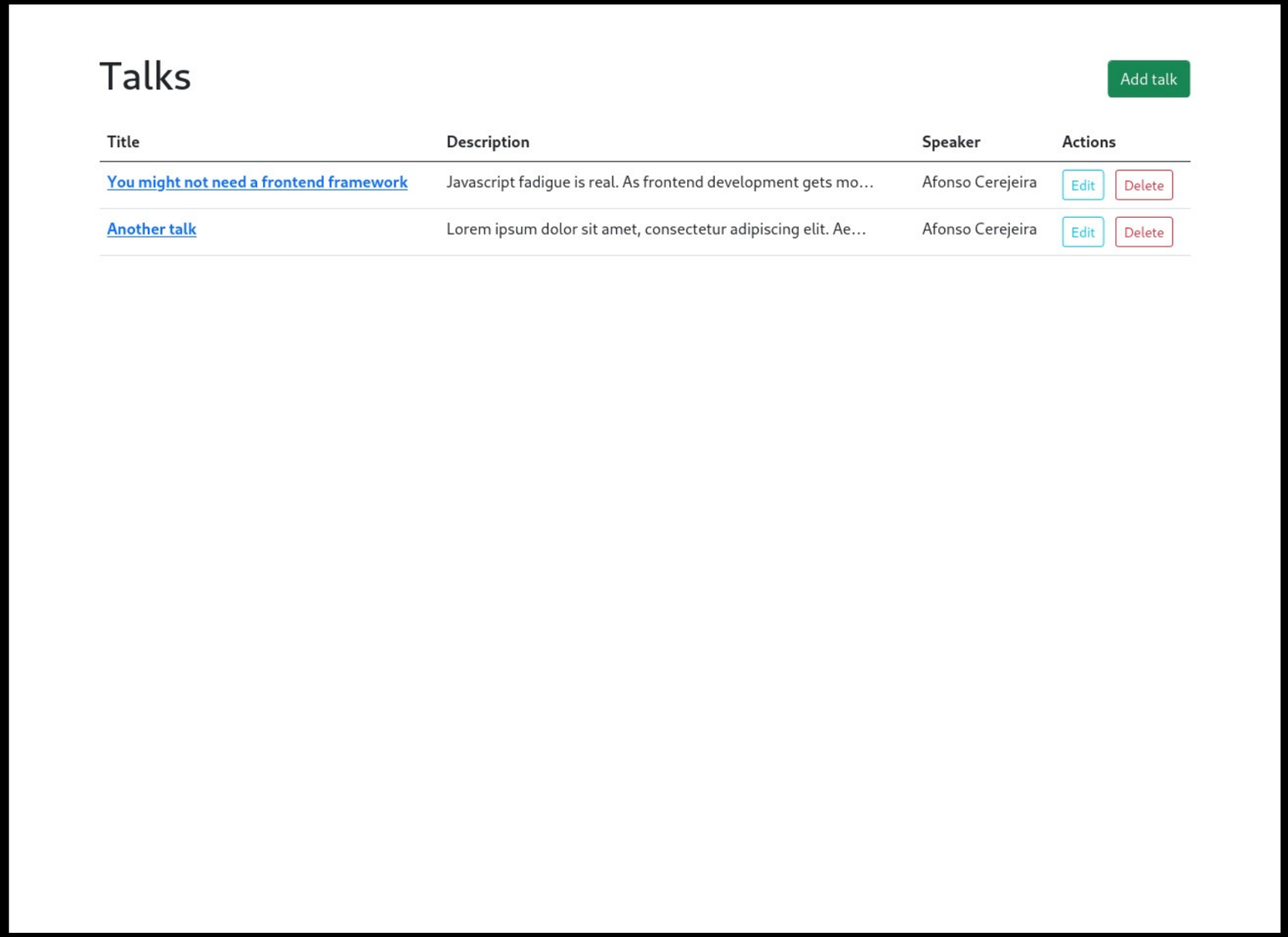
Example: “talks” app

```
# talks/views.py
class TalkListView(ListView):
    model = Talk

class TalkDetailView(DetailView):
    model = Talk

class TalkCreateView(CreateView):
    model = Talk
    fields = ["title", "description", "speaker"]

class TalkUpdateView(UpdateView):
    model = Talk
    fields = ["title", "description", "speaker"]

class TalkDeleteView(DeleteView):
    model = Talk
    success_url = reverse_lazy("talk_list")
```

Example: “talks” app

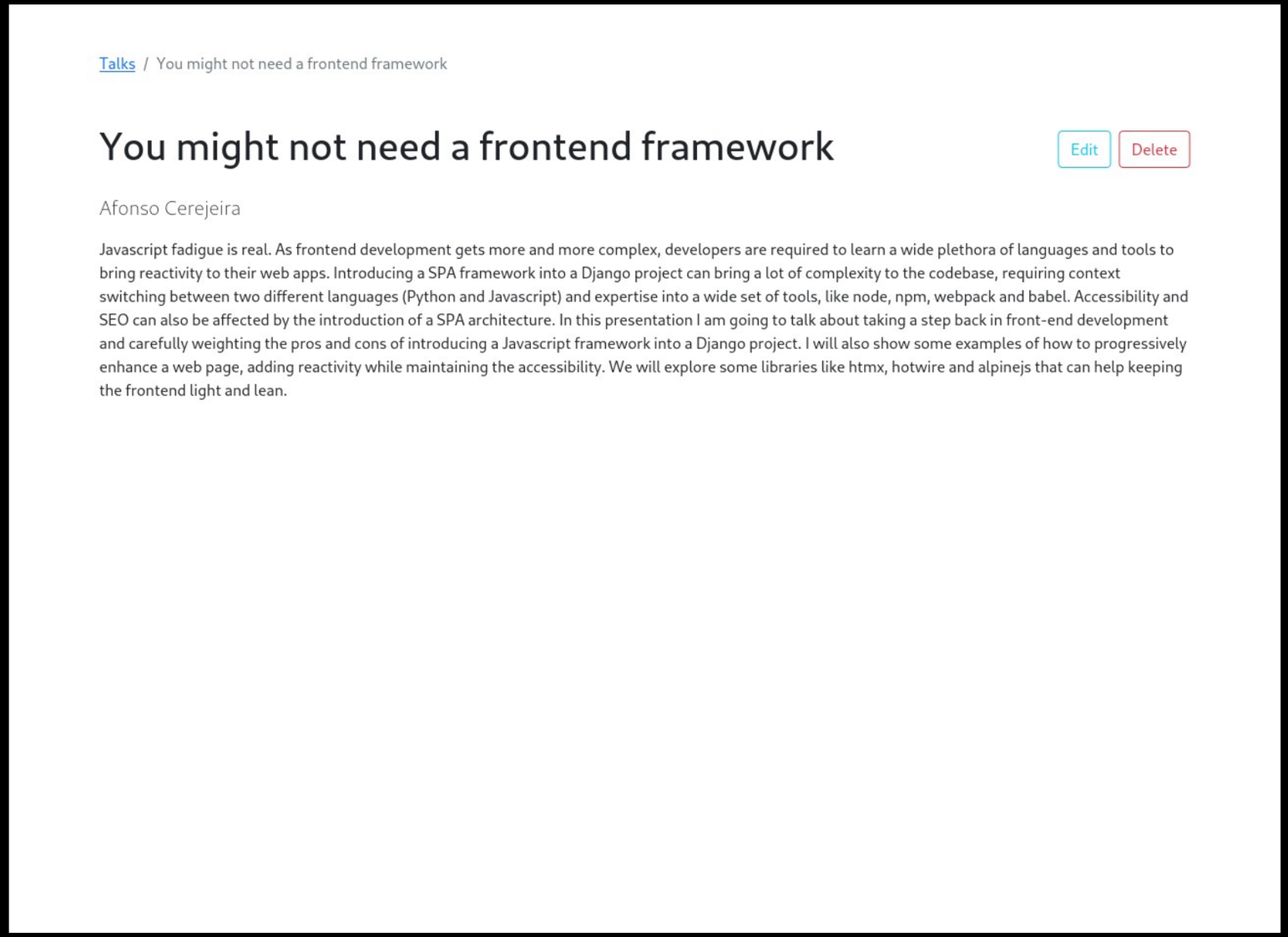
```
# talks/urls.py
urlpatterns = [
    path("", TalkListView.as_view(), name="talk_list"),
    path("new/", TalkCreateView.as_view(), name="talk_create"),
    path(
        "<int:pk>/", TalkDetailView.as_view(), name="talk_detail"
    ),
    path(
        "<int:pk>/edit",
        TalkUpdateView.as_view(),
        name="talk_update",
    ),
    path(
        "<int:pk>/delete",
        TalkDeleteView.as_view(),
        name="talk_delete",
    ),
]
```

Example: “talks” app



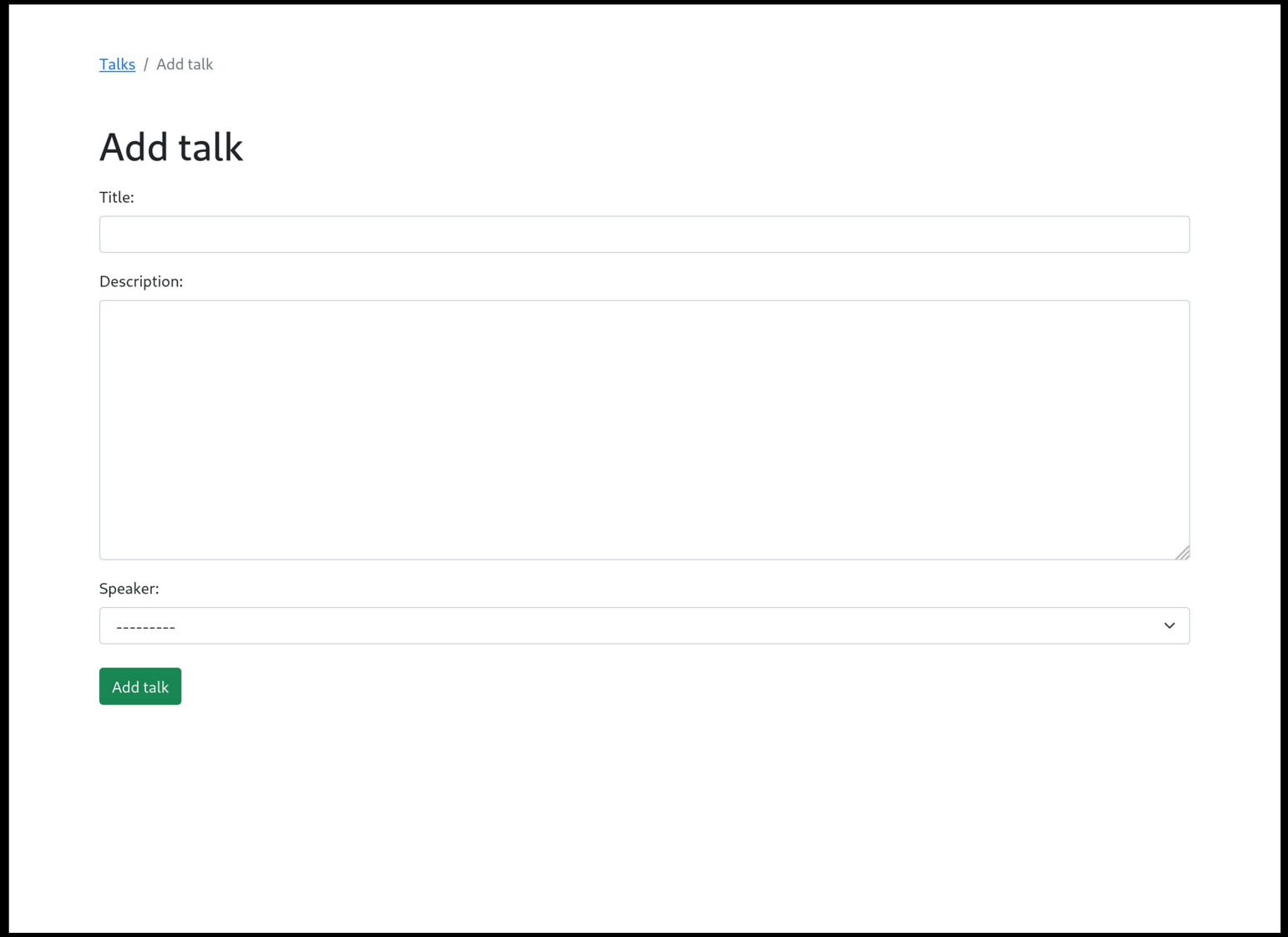
GET /talks/

Example: “talks” app



GET /talks/1

Example: “talks” app



GET /talks/new

htmx

- Uses **data attributes** for accessing:
  - AJAX
  - CSS transitions
  - WebSockets
  - Server Sent Events
- Small (~10kB min.gz'd)
- Dependency-free
- Extendable
- IE11 compatible

htmx: quick example

```
<!-- "when a user clicks on this button, issue an ajax request
to /clicked, and replace the entire button with the response" -->

<!-- Load from unpkg -->
<script src="https://unpkg.com/htmx.org@1.3.3"></script>
<!-- have a button POST a click via AJAX -->
<button hx-post="/clicked" hx-swap="outerHTML">
  Click Me
</button>
```

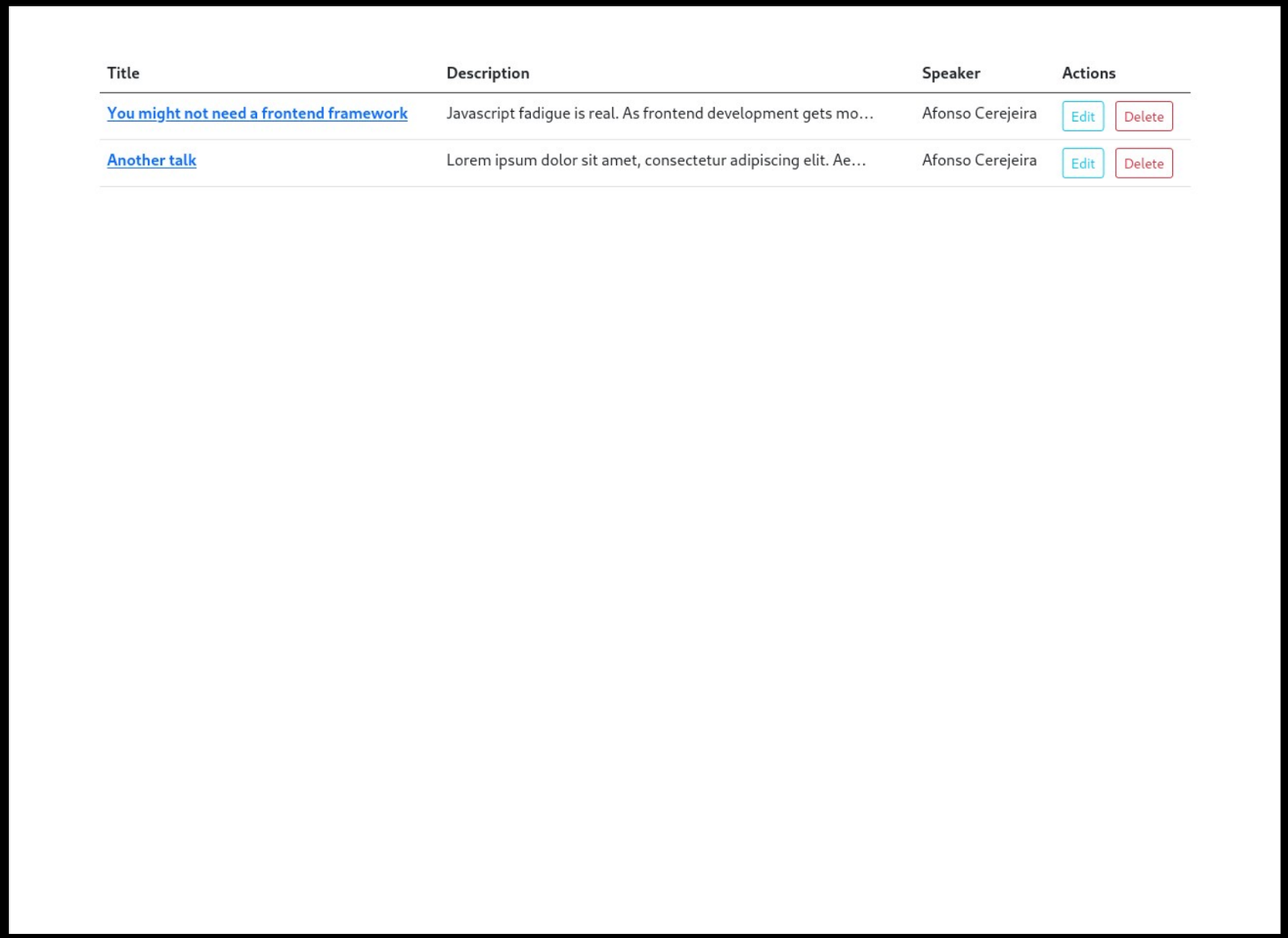
htmx: adding active search

```
# talks/views.py
class TalkSearchView(TalkListView):
    template_name = "talks/talk_table.html"

    def get_queryset(self):
        talks = super().get_queryset()
        if "q" in self.request.GET:
            q = self.request.GET["q"]
            talks = talks.filter(
                Q(title__icontains=q)
                | Q(description__icontains=q)
            )
        return talks

# talks/urls.py
urlpatterns = [
    ...
    path("search/", TalkSearchView.as_view(), name="talk_search"),
]
```

htmx: adding active search

```
# templates/talks/talk_table.html
<table id="talks-table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Description</th>
      <th>Speaker</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    {% for talk in talks %}
      <tr>
        <th>{{ talk.title }}</th>
        <td>{{ talk.description|truncatechars:60 }}</td>
        <td>{{ talk.speaker.get_full_name }}</td>
        <td>
          <a href="{% url 'talk_update' talk.pk %}">Edit</a>
          <a href="{% url 'talk_delete' talk.pk %}">Delete</a>
        </td>
      </tr>
    {% endfor %}
  </tbody>
</table>
```

htmx: adding active search

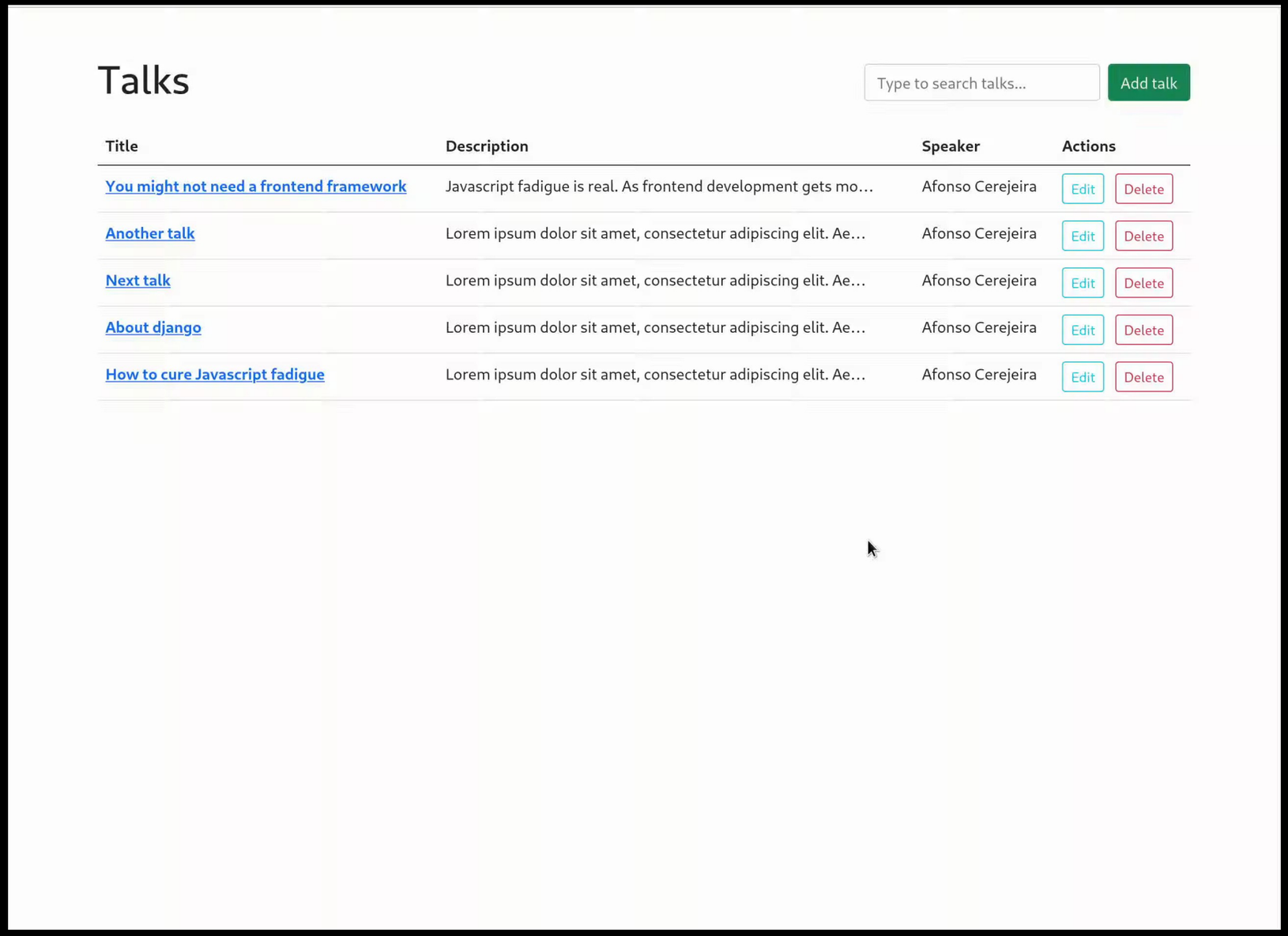


GET /talks/search?q=talk

htmx: adding active search

```
# templates/talks/talk_list.html
<h1>Talks</h1>
<input
  type="text"
  name="search" placeholder="Type to search talks..."
  hx-get="{% url 'talk_search' %}"
  hx-trigger="keyup changed delay:500ms"
  hx-target="#talks-table">
<a href="{% url 'talk_create' %}">Add talk</a>

{% include "talks/talk_table.html" %}
```

htmx: adding active search



GET /talks

Alpine.js

- Minimal framework for composing JavaScript behavior in markup
- Suitable for simple user interactions
- Does not use virtual-dom
- Small (~7kB min.gz'd)
- Syntax inspired by Vue template expressions
- No build step required

Alpine.js: quick example

```
<!-- tab selector -->

<script
src="https://cdn.jsdelivr.net/gh/alpinejs/alpine@v2.8.2/dist/alpine
.min.js" defer></script>

<div x-data="{ tab: 'foo' }">
  <button @click="tab = 'foo'">Foo</button>
  <button @click="tab = 'bar'">Bar</button>

  <div x-show="tab === 'foo'">Tab Foo</div>
  <div x-show="tab === 'bar'">Tab Bar</div>
</div>
```

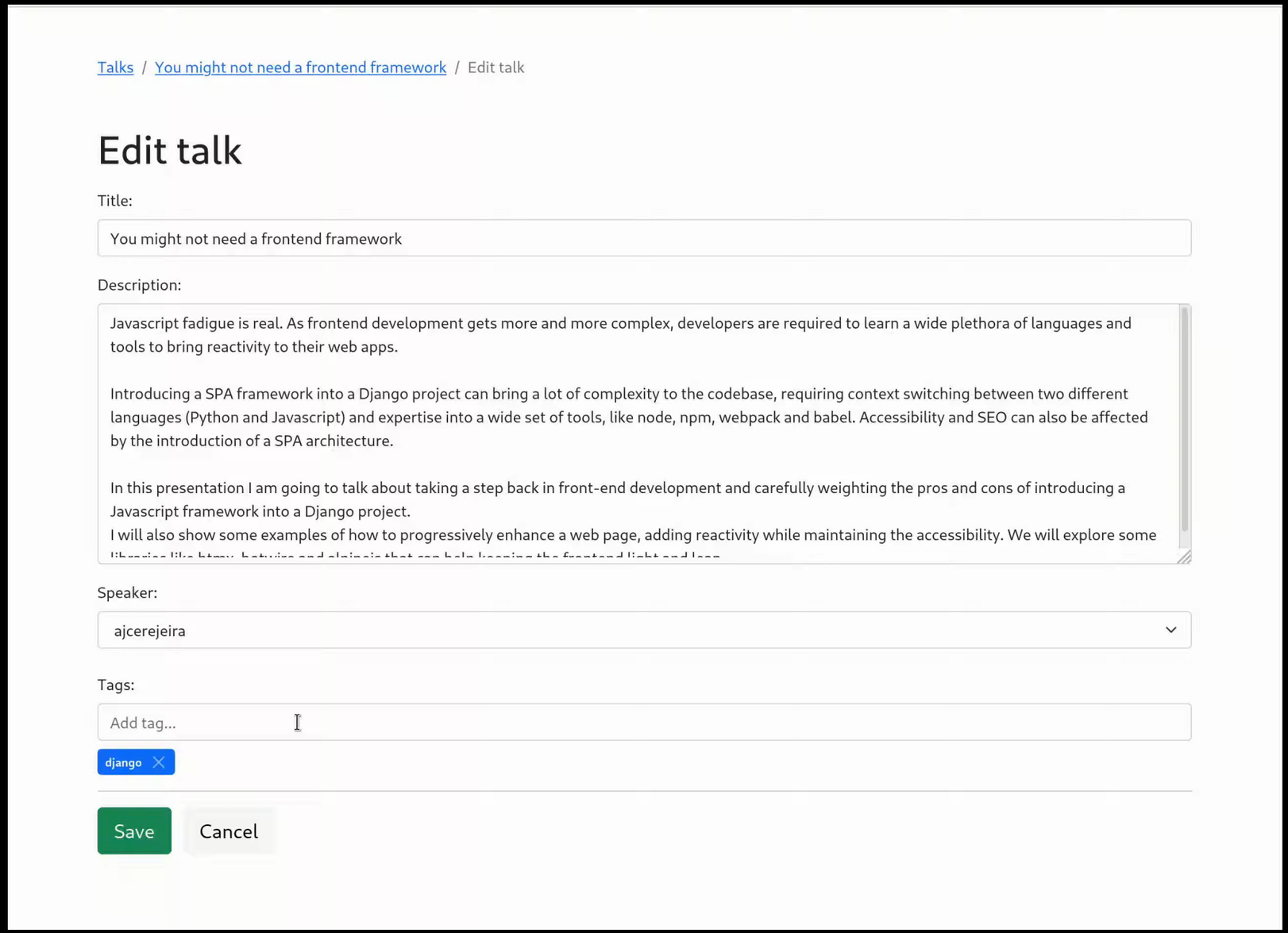
Alpine.js: adding tags input

```
# talks/models.py
class Talk(models.Model):
    title = models.CharField(max_length=200, unique=True)
    description = models.TextField(blank=True)
    speaker = models.ForeignKey(User, on_delete=models.CASCADE)
    tags = models.JSONField(default=list)
```

Alpine.js: adding tags input

```
# templates/talks/talks_form.html
<form method="POST">
  {% csrf_token %}
  ...
  {% autoescape off %}
  <div x-data="{tags: {{ talk.tags|default:"[]" }}, newTag: ''}">
    <input type="hidden" name="tags"
        x-bind:value="JSON.stringify(tags)">
    <input x-model="newTag" @keydown.enter.prevent=
      "if (newTag.trim() !== '')
          tags.push(newTag.trim());
       NewTag = ''">
    <template x-for="tag in tags" :key="tag">
      <span x-text="tag" class="me-2"></span>
      <button type="button" class="btn-close btn-close-white"
          @click="tags = tags.filter(i => i !== tag)"></button>
    </template>
  </div>
  {% endautoescape %}
</form>
```

Alpine.js: adding tags input



GET /talks/1/edit

Hotwire: HTML over the wire

- Successor of **turbolinks**
- SPA page load speed experience without writing Javascript
- Server renders the pages, **hotwire** handles navigation
- Currently in beta (non-official Django package here: github.com/hotwire-django/turbo-django)

“Simplicity is the ultimate sophistication”
Leonardo da Vinci