DJANGO MODEL FORM + API BASICS

06016322 - WEB PROGRAMMING

CREATING FORMS FROM MODELS

```python
from django.db import models
from django.forms import ModelForm

TITLE_CHOICES = (
    ('MR', 'Mr.'),
    ('MRS', 'Mrs.'),
    ('MS', 'Ms.'),
)

class Author(models.Model):
    name = models.CharField(max_length=100)
    title = models.CharField(max_length=3, choices=TITLE_CHOICES)
    birth_date = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.name

class Book(models.Model):
    name = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
```

```python
class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ['name', 'title', 'birth_date']

class BookForm(ModelForm):
    class Meta:
        model = Book
        fields = ['name', 'authors']
```

```python
from django import forms

class AuthorForm(forms.Form):
    name = forms.CharField(max_length=100)
    title = forms.CharField(
        max_length=3,
        widget=forms.Select(choices=TITLE_CHOICES),
    )
    birth_date = forms.DateField(required=False)

class BookForm(forms.Form):
    name = forms.CharField(max_length=100)
    authors = forms.ModelMultipleChoiceField(queryset=Author.objects.all())
```

THE SAVE() METHOD

- Every ModelForm also has a save() method. This method creates and saves a database object from the data bound to the form.
- Note that if the form hasn't been validated, calling save() will do so.

```
>>> from myapp.models import Article
>>> from myapp.forms import ArticleForm

# Create a form instance from POST data.
>>> f = ArticleForm(request.POST)

# Save a new Article object from the form's data.
>>> new_article = f.save()

# Create a form to edit an existing Article, but use
# POST data to populate the form.
>>> a = Article.objects.get(pk=1)
>>> f = ArticleForm(request.POST, instance=a)
>>> f.save()
```

SELECTING THE FIELDS TO USE

- It is strongly recommended that you explicitly set all fields that should be edited in the form using the fields attribute.

```
from django.forms import ModelForm

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = '__all__'
```

```
class PartialAuthorForm(ModelForm):
    class Meta:
        model = Author
        exclude = ['title']
```

Since the Author model has 3 fields name, title and birth_date, this will result in the fields name and birth_date being present on the form.

OVERRIDING THE DEFAULT FIELDS

```
from django.utils.translation import gettext_lazy as _

class AuthorForm(ModelForm):
    class Meta:
        model = Author
        fields = ('name', 'title', 'birth_date')
        labels = {
            'name': _('Writer'),
        }
        help_texts = {
            'name': _('Some useful help text.'),
        }
        error_messages = {
            'name': {
                'max_length': _("This writer's name is too long."),
            },
        }
```

OVERRIDING THE DEFAULT FIELDS

```
from django.forms import CharField, ModelForm
from myapp.models import Article

class ArticleForm(ModelForm):
    slug = CharField(validators=[validate_slug])

    class Meta:
        model = Article
        fields = ['pub_date', 'headline', 'content', 'reporter', 'slug']
```

DJANGO FORMSET

FORMSETS

- A formset is a layer of abstraction to work with multiple forms on the same page.
- Let's say you have the following form:

```
>>> from django import forms
>>> class ArticleForm(forms.Form):
...     title = forms.CharField()
...     pub_date = forms.DateField()
```

- You might want to allow the user to create several articles at once. To create a formset out of an ArticleForm you would do:

```
>>> from django.forms import formset_factory
>>> ArticleFormSet = formset_factory(ArticleForm)
```

FORMSETS

- You now have created a formset named ArticleFormSet. The formset gives you the ability to iterate over the forms in the formset and display them as you would with a regular form:

```
>>> formset = ArticleFormSet()
>>> for form in formset:
...     print(form.as_table())
<tr><th><label for="id_form-0-title">Title:</label></th><td><input type="text"
name="form-0-title" id="id_form-0-title"></td></tr>
<tr><th><label for="id_form-0-pub_date">Pub date:</label></th><td><input type="text"
name="form-0-pub_date" id="id_form-0-pub_date"></td></tr>
```

USING INITIAL DATA WITH A FORMSET

```
>>> import datetime
>>> from django.forms import formset_factory
>>> from myapp.forms import ArticleForm
>>> ArticleFormSet = formset_factory(ArticleForm, extra=2)
>>> formset = ArticleFormSet(initial=[
...     {'title': 'Django is now open source',
...      'pub_date': datetime.date.today(),}
... ])
```

```
>>> for form in formset:
...     print(form.as_table())
<tr><th><label for="id_form-0-title">Title:</label></th><td><input type="text"
name="form-0-title" value="Django is now open source" id="id_form-0-title"></td></tr>
<tr><th><label for="id_form-0-pub_date">Pub date:</label></th><td><input type="text"
name="form-0-pub_date" value="2008-05-12" id="id_form-0-pub_date"></td></tr>
<tr><th><label for="id_form-1-title">Title:</label></th><td><input type="text"
name="form-1-title" id="id_form-1-title"></td></tr>
<tr><th><label for="id_form-1-pub_date">Pub date:</label></th><td><input type="text"
name="form-1-pub_date" id="id_form-1-pub_date"></td></tr>
<tr><th><label for="id_form-2-title">Title:</label></th><td><input type="text"
name="form-2-title" id="id_form-2-title"></td></tr>
<tr><th><label for="id_form-2-pub_date">Pub date:</label></th><td><input type="text"
name="form-2-pub_date" id="id_form-2-pub_date"></td></tr>
```

USING A FORMSET IN VIEWS AND TEMPLATES

```
from django.forms import formset_factory
from django.shortcuts import render
from myapp.forms import ArticleForm

def manage_articles(request):
    ArticleFormSet = formset_factory(ArticleForm)
    if request.method == 'POST':
        formset = ArticleFormSet(request.POST, request.FILES)
        if formset.is_valid():
            # do something with the formset.cleaned_data
            pass
    else:
        formset = ArticleFormSet()
    return render(request, 'manage_articles.html', {'formset': formset})
```

- The manage_articles.html template might look like this:

```
<form method="post">
    {{ formset.management_form }}
    <table>
        {% for form in formset %}
        {{ form }}
        {% endfor %}
    </table>
</form>
```

- However there's a slight shortcut for the above by letting the formset itself deal with the management form:
- The {{ formset }} calls the as_table method on the formset class.

```
<form method="post">
    <table>
        {{ formset }}
    </table>
</form>
```

USING VUE JS WITH DJANGO

INSTALLING VUE JS

- Direct <script> Include
  - Simply download and include with a script tag. Vue will be registered as a global variable.

    https://vuejs.org/js/vue.js

- CDN
  - For prototyping or learning purposes, you can use the latest version with:

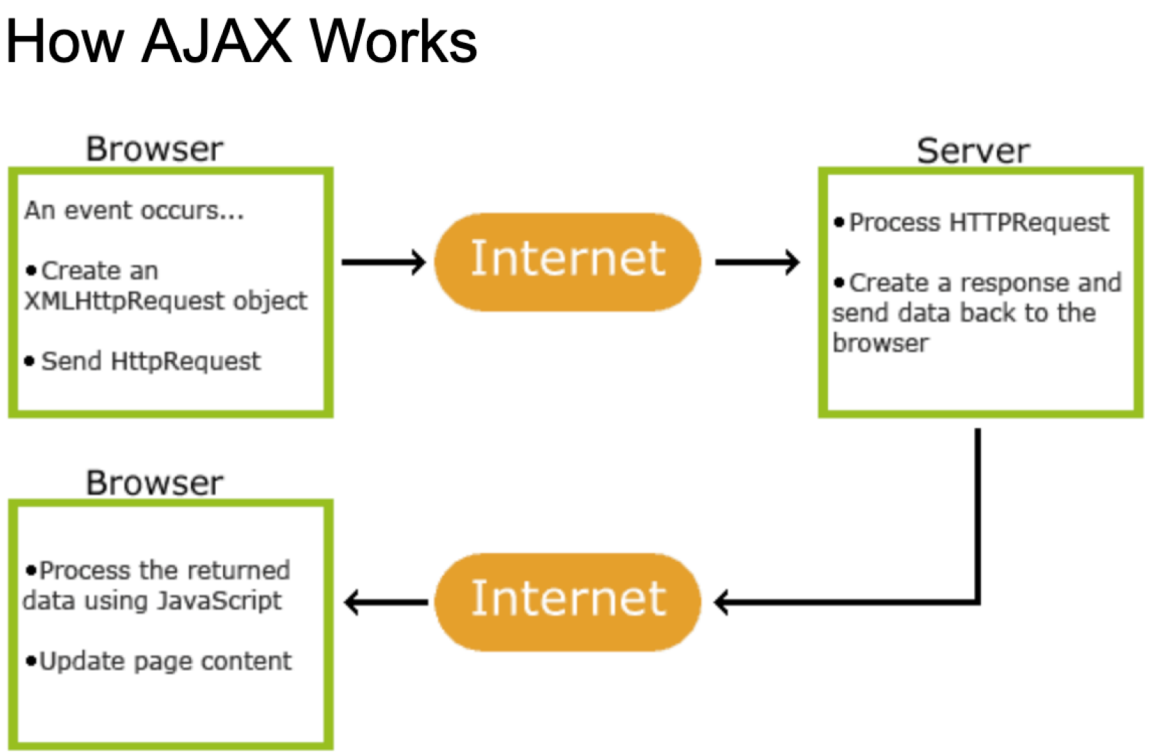
```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

WHAT IS AJAX?

WHAT IS AJAX?

- **Asynchronous Javascript and XML** (AJAX), is a way of communicating to a web server from a client-side application through the *HTTP* or *HTTPS* protocol.

- Even though AJAX holds XML in the name, the way data is sent through requests or received doesn't have to be XML, but also plain text, or in most cases JSON, due to it being lighter and a part of JavaScript in and of itself.



WHY IS AJAX USEFUL?

- Vue.js is used as a front-end framework, and if you ever want to communicate with a server, to retrieve or store information to a database or perform some calculations on your data you will most likely need **AJAX**.

- Even though AJAX can be used by creating an XMLHttpRequest object, which will be available through the browser. There are certain packages that can help us communicate with our server.

- Vue.js has an official package which is called **vue-resource** which works as an HTTP client, but the official documentation suggests using **Axios**.

AJAX USING VANILLA JS

- Making AJAX requests with the XMLHttpRequest() method, often referred to as XHR, is a three step process:
  1. Set up our request by creating a new XMLHttpRequest().
  2. Create an onload callback to run when the request completes.
  3. Open and send our request.

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

AJAX USING AXIOS

- There are many times when building application for the web that you may want to consume and display data from an API. There are several ways to do so, but a very popular approach is to use **axios**, a promise-based HTTP client.

- In this exercise, we'll use the **CoinDesk API** to walk through displaying Bitcoin prices, updated every minute.

```
new Vue({
  el: '#app',
  data () {
    return {
      info: null
    }
  },
  mounted () {
    axios
      .get('https://api.coindesk.com/v1/bpi/currentprice.json')
      .then(response => (this.info = response))
  }
})
```

DEALING WITH ERRORS

```
  mounted () {
    axios
      .get('https://api.coindesk.com/v1/bpi/currentprice.json')
      .then(response => {
        this.info = response.data.bpi
      })
      .catch(error => {
        console.log(error)
        this.errored = true
      })
      .finally(() => this.loading = false)
  }
})
```

DJANGO'S API VIEWS

REST API WITH DJANGO

```
from django.utils import simplejson
from django.http import HttpResponse

def some_view(request):
    to_json = {
        "key1": "value1",
        "key2": "value2"
    }
    return HttpResponse(simplejson.dumps(to_json), mimetype='application/json')
```

```
from django.http import JsonResponse
def some_view(request):
    return JsonResponse({"key": "value"})
```

EXERCISE IDEAS

- Delete question
- Create question
- Create choice(s)