INFO5002: Intro to Python for Info Sys

Week 9

Slides created by: Zachary Doucet

Northeastern University

Week 9

I. CSS

II. Django

Recap

APIs as Interface

- Interface to share data between codebases.
- Is user defined and follows a strict user defined format.

Regular Expressions

Mini string pattern language

- Character classes with [] and the complement with [^].
- Escape sequences with \: e.g \d, \D, \s, \S, \w, \W, \\
- Ranges with -.
- Reptition with *, +, ?, {a, b}, {a,}, {,b}, {x}
- Use in python by import re and creating a pattern object by passing a regular expression r”” to re.compile

HTML

- **HTML** describes the layout of a webpage. i.e. What is the content? How is that content organized?
- The webpage is divided into tags which instructs the browser how to render the content.
- Tags are defined with angled brackets <tag> while a closing tag is defined with angled brackets with a slash </tag>.

HTML Tags

- Headers: <h1></h1> through <h6></h6>
- Paragraph:
- Unordered lists:
- Ordered lists:
- Tables: <table><tr><th></th></tr><tr><td></td></tr></table>
- Anchors:

Forms: collect user data

- <form></form>: create a form
- <input type=”” />
  - type=”text”: single-line input
  - type=”radio”: selecting one of many choices
  - type=”checkbox”: one or more selections
  - type=”submit”: submit button for the form
  - type=”button”: display clickable button

- <label for=””></label>: creating an input for input id=for
- <select><option></option></select>: dropdown menu
- <textarea rows=”” cols=””></textarea>: multi-line resizable input field.
- <button type=””></button>: define a clickable button
  - type=”button”: clickable
  - type=”submit”: submits form
  - type=”reset”: re-initialises form

All input types

- button
- checkbox
- color
- date
- datetime-local
- email
- file
- hidden
- image
- month
- number
- password
- radio
- range
- reset
- search
- submit
- tel
- text
- time
- url
- week

CSS

MDN, W3Schools

CSS as style

- CSS is to style as HTML is to layout.
- You first select a tag, class, id, or combination of such and define a style onto the target.

HTML Integration (3)

I. Write styling in html tag.

II. Can write styling in-between <style></style> tag in html header.

III. Write in a separate file and load the file.

- I takes precedence over II and II takes precedence over III.
  - If I define a style in III it will be overwritten if I redefine in I.

Inline CSS

- Within the html open tag you add an attribute style where you define declarations as a semi-colon seperated list of property: value pairs.

```
<tag style="property1: value1; property2: value2">…</tag>
```

Internal CSS

- In the head between style tags define semi-colon separated property: value pairs associated to a selector between braces.

```
<head>
    <style>
        selector {
            property1: value1;
            property2: value2;
        }
    </style>
</head>
```

External CSS

- Create an external css file following same pattern of internal CSS without need of HTML tags.
- Import into html using link rel=”stylesheet”

```
<head>
    <link rel="stylesheet" href="styles.css" />
</head>
```

- Best practice!

Selectors

I. Tag by specifying tag name.

```
h1 {
    property: value;
}
```

II. Class by specifying .{class-name}

```
.dark {
    property: value;
}
```

III. Id by specifying #{id}

```
#big-title {
    property: value;
}
```

Common properties

** Property names follow American spelling **

color	Text colour.
background-color	The background colour.
background-image	e.g url(“image.jpg”)
opacity	How opaque. 0 (transparent), 1 (solid)
font-size	Size of text (px, em, rem)
font-family	Name of typeface (e.g Arial, Times)
font-weight	bold, normal, or numbers [1, 1000]
text-align	left, center, right
line-height	Spacing between lines (e.g 2.5, 3em)

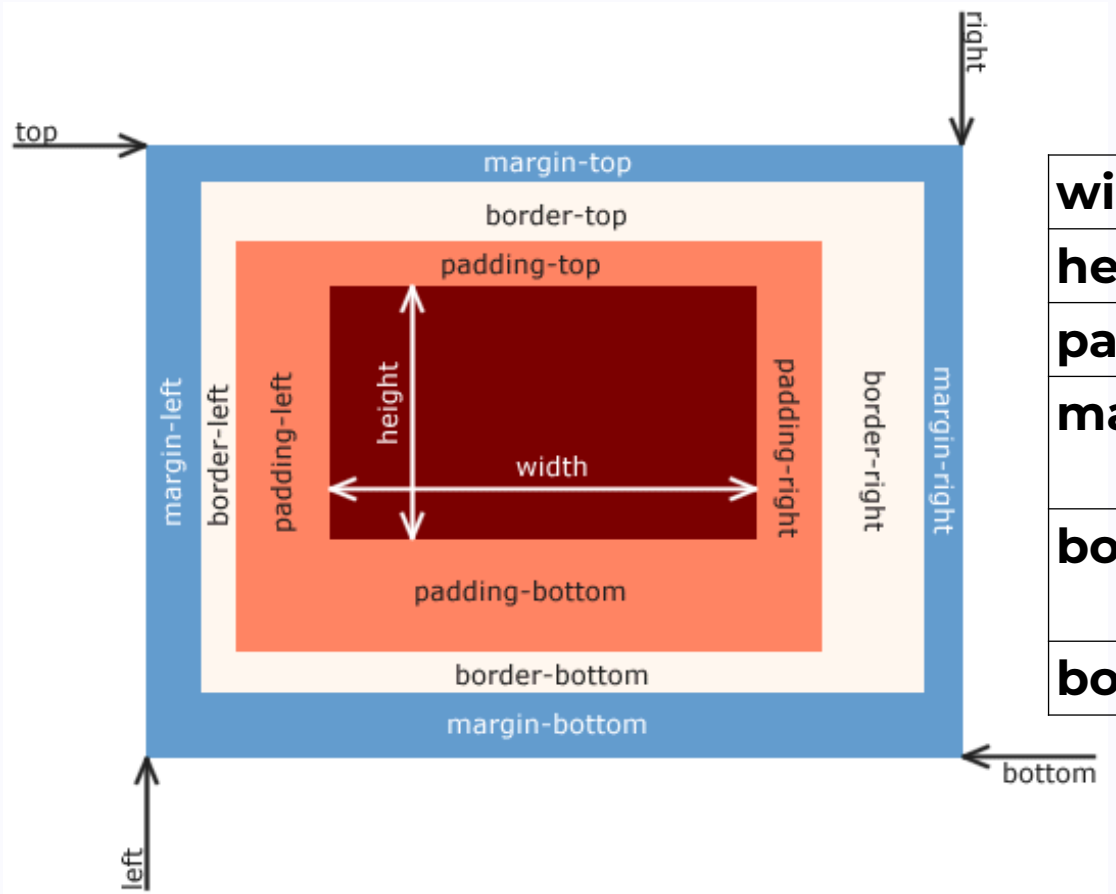
Colours can be written as an html name (e.g. red, blue), as a hexademical (e.g #ff0000), or in rgb (e.g rgb(255, 99, 71))

Sizing

Unit	Example	Relative To
px	12px	Nothing (fixed)
%	50%	Parent element
em	6em	Parent font-size
rem	0.75rem	Root font-size
vh/vw	100vw	Viewport
Unitless (line-height only)	1.5	Font-size

Be careful of compounding using em! If you have a parent tag with 2em and your current tag uses 2em that's a total of 4em.

Box Model



width	Elem width
height	Elem height
padding	Space inside elem
margin	Space outside elem
border	Border around elem
border-radius	Rounded corners

```
margin-left: 1rem;
margin-right: 1rem;
margin-top: 1rem;
margin-bottom: 1rem;

/* short hands */
margin: 10px;                /* all sides */
margin: 10px 20px;           /* top/bottom left/right */
margin: 10px 20px 30px 40px;  /* top right bottom left (clockwise) */

/* padding same pattern */
```

Layout

- Display controls how elements are rendered on page.
- **block**: take up full width available, starts on new line (e.g headers and paragraphs).
- **inline**: takes up only amount needed, stays on new line until width full (e.g anchors).
- **none**: takes up no space on screen (for hiding info).
- **flex**: create flexible layout (default to horizontal stacking).

```
.container {
    display: block;
}
```

Let’s practice

Taking the website your designed potentially last class style it! Or take your ps8 html file and style it!

Django

PCC 373-401

* Using version 5.2.7

A Web Framework

- You can service your website through HTML pages in a static fashion.
- You can service your web pages dynamically with the help of a web framework: Django for Python.
- Install with **pip install django** in your terminal.

```
pip install django
```

Starting new project

- You can create a new Django project by typing

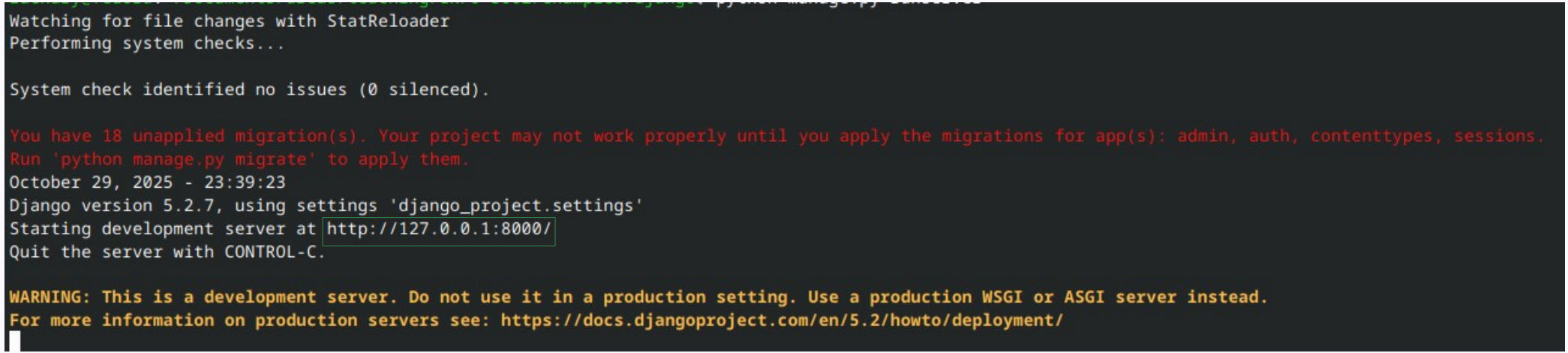
```
django-admin startproject calendar_project .
```

  - Will create a folder with name *calendar_project* and a *manage.py* file which helps you run django commands. `calendar_project  db.sqlite3  manage.py`
  - Inside are 4 files: `asgi.py  __init__.py  settings.py  urls.py  wsgi.py`
    - *settings.py*: defines the settings Django will use.
    - *urls.py*: defines which pages to use for each browser request.
    - *wsgi.py* (web server gateway interface): helps Django serve the web pages to the client.
    - *asgi.py* (asynchronous server gateway interface): async version.

Starting server

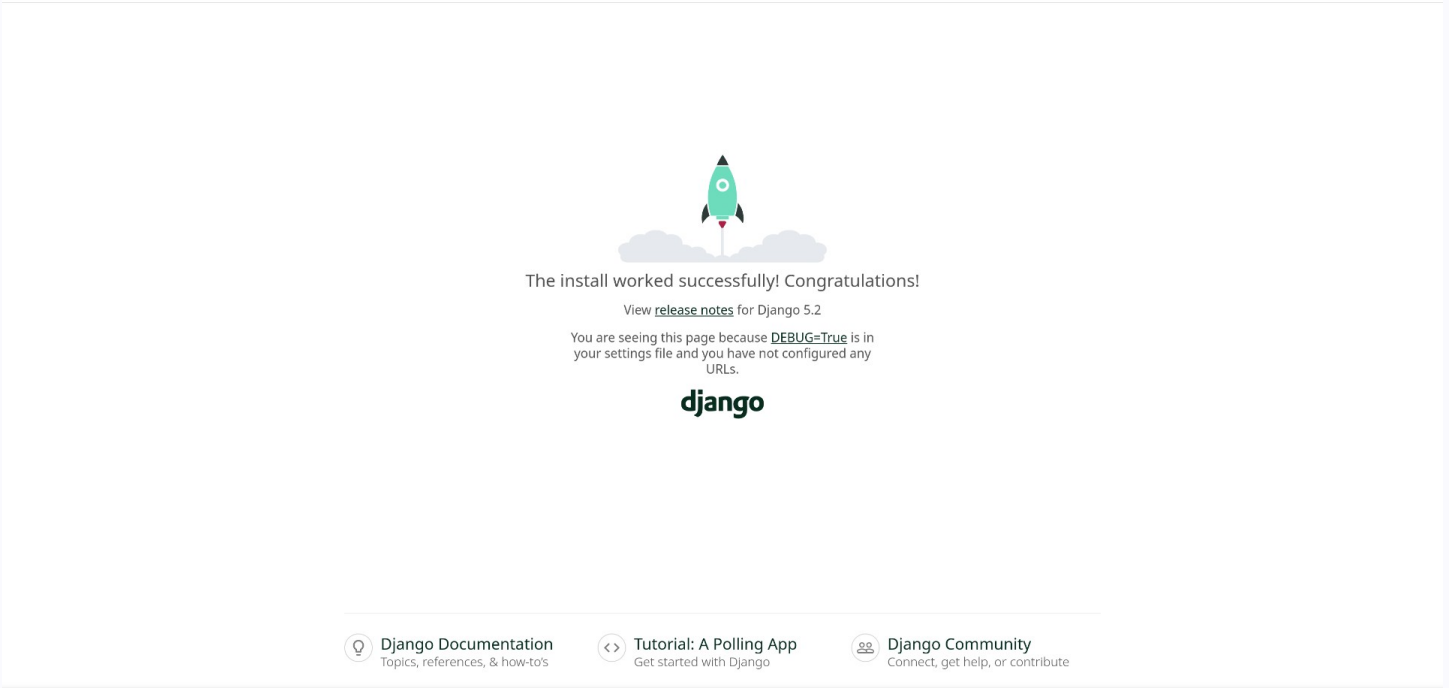
- You can start server with **runserver** and optionally specify a port.

```
python manage.py runserver [port]
```



View with web browser

```
http://127.0.0.1:8000
http://localhost:8000
```



Start an app in our project

- Django project works as a collection of apps working together.

```
python manage.py startapp calendar_events
```

- Can create an app with **startapp**.
- Creates many files:

```
admin.py  apps.py  __init__.py  migrations  models.py  tests.py  views.py
```

  - *models.py*: define the data to manage.
  - *admin.py*: customise admin panel.
  - *views.py*: where you define view functions.

Enabling App

- In your settings.py add the created app under INSTALLED_APPS.

```
INSTALLED_APPS = [
    # My apps
    "calendar_events",
    # Default apps
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
]
```

Let’s Practice

I. Install Django

II. Create a new project called ml_project

III. Create an app in ml_project called mad_libs

Models

- Instructs Django how to work with the data in the app.
- Created as a class that inherits from **models.Model**
- Attributes are assigned to the expected data type found under **models.{date-type}Field**

```
from django.db import models

class Event(models.Model):
    """A Calendar event that the user has."""
    start_datetime = models.DateTimeField()
    end_datetime = models.DateTimeField()
    event_name = models.CharField(max_length=255)
    notes = models.TextField()
```

Each **field** will be a database entry. More on databases later.

Field Types

AutoField	IntegerField auto increments	**BigAutoField**	64bit auto increments
BigIntegerField	64bit integer	**BinaryField**	Raw binary data
BooleanField	Boolean	**CharField**	Small amt text up to max_length
DateTimeField	Python's datetime.datetime	**DateField**	Python's datetime.date
DurationField	Time periods Python's timedelta	**DecimalField**	Fixed-precision decimal number
EmailField	CharField that validates email	**FileField**	File
FilePathField	Filepath choices from path	**FloatField**	Floating-point number
GenericIpAddressField	IPv4 or IPv6 address	**ImageField**	FileField that validates image
IntegerField	Integer	**JSONField**	Json
PositiveBigIntegerField	64bit positive integer	**PositiveIntegerField**	Positive integer
SlugField	Short label	**SmallAutoField**	AutoField for smaller nums

SmallIntegerField	Small Integer	**TextField**	Large text field
TimeField	Python’s datetime.time	**UrlField**	CharField validates URL
UUIDField	Python’s UUID		

Can find all with the explanations at:
https://docs.djangoproject.com/en/5.2/ref/models/fields/#field-types

Pushing models to DB

```
python manage.py makemigrations calendar_events
```

App name

```
Migrations for 'calendar_events':
  calendar_events/migrations/0001_initial.py
    + Create model Event
```

Migration file tells Django how to update database

```
python manage.py migrate
```

Apply any migrations

```
Operations to perform:
  Apply all migrations: admin, auth, calendar_events, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying calendar_events.0001_initial... OK
  Applying sessions.0001_initial... OK
```

* Do this whenever you make updates to ***models.py***

Admin Site

- Place for admins to modify the site easily.
- Create admin with createsuperuser:

```
python manage.py createsuperuser
```

```
Username (leave blank to use 'zachary'): admin
Email address:
Password:
Password (again):
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

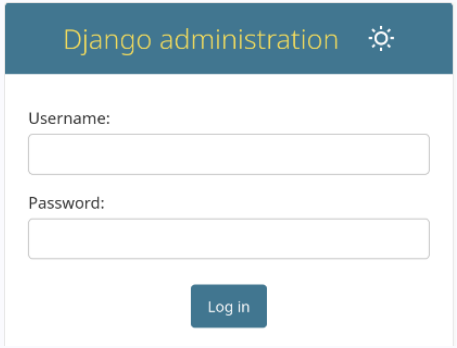
Registering model w/ admin site

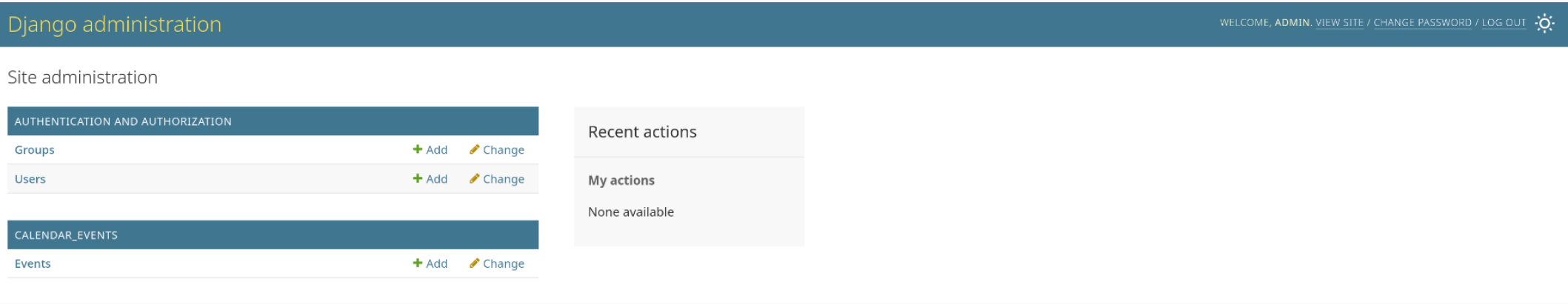
- Register models in *admin.py*.

```
from django.contrib import admin

from .models import Event

admin.site.register(Event)
```

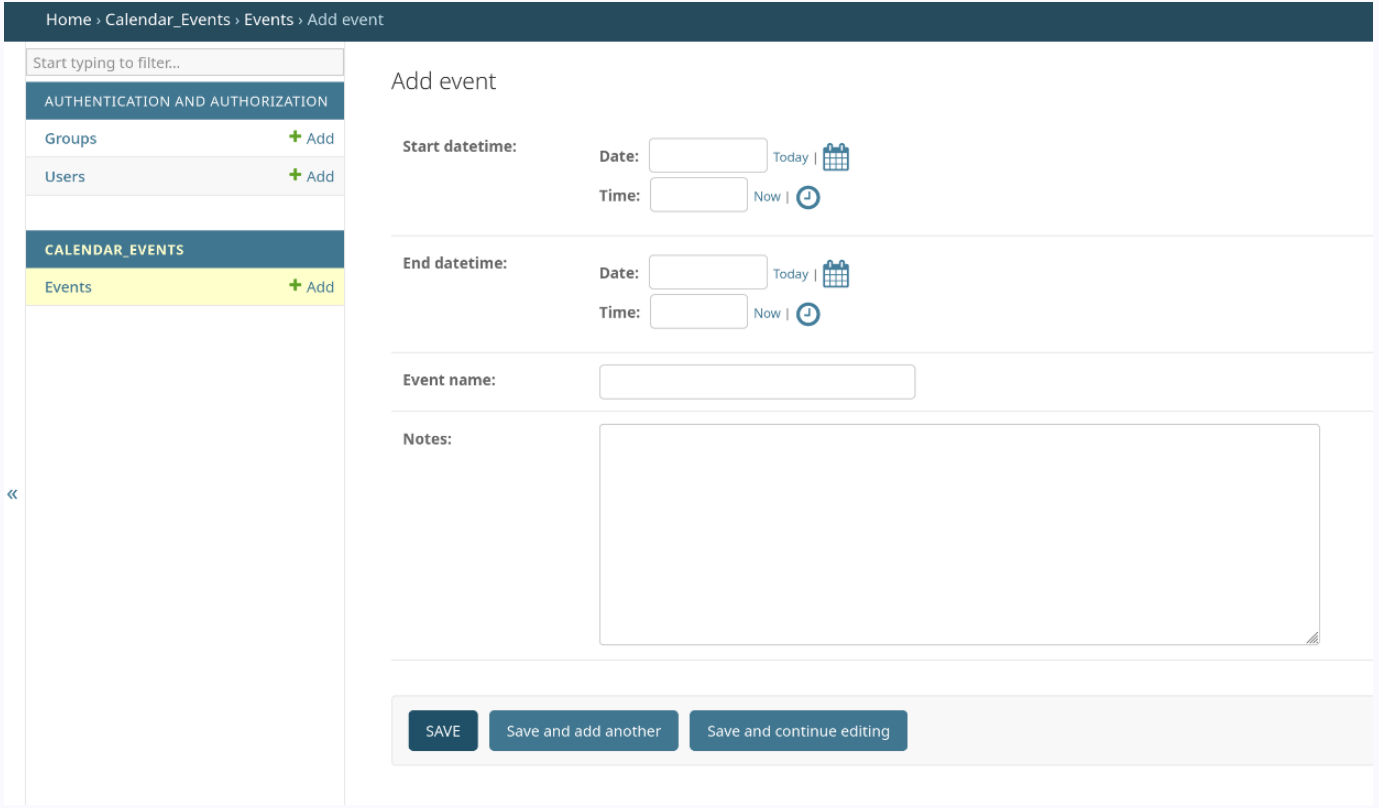
Login to admin site

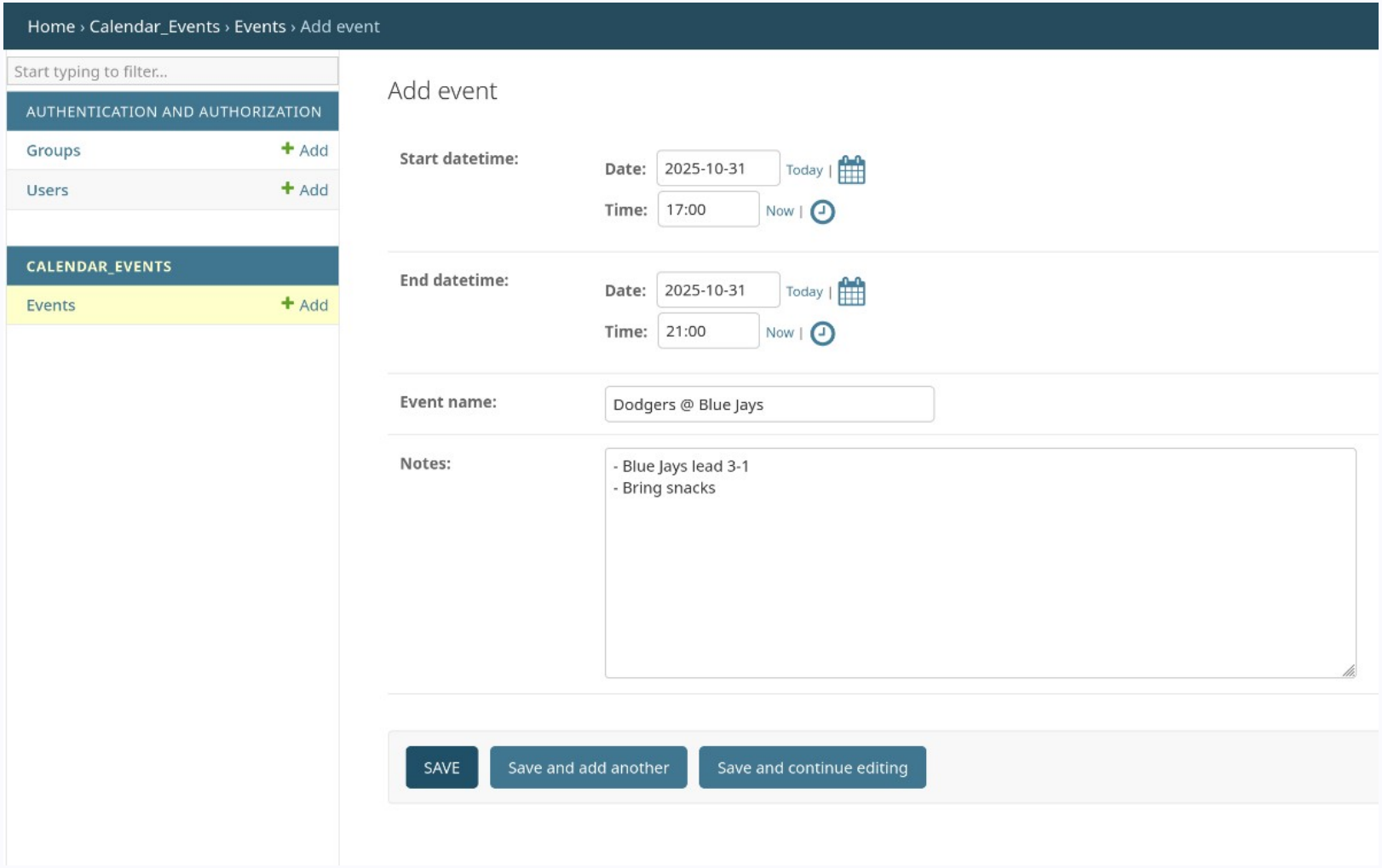
http://localhost:8000/admin

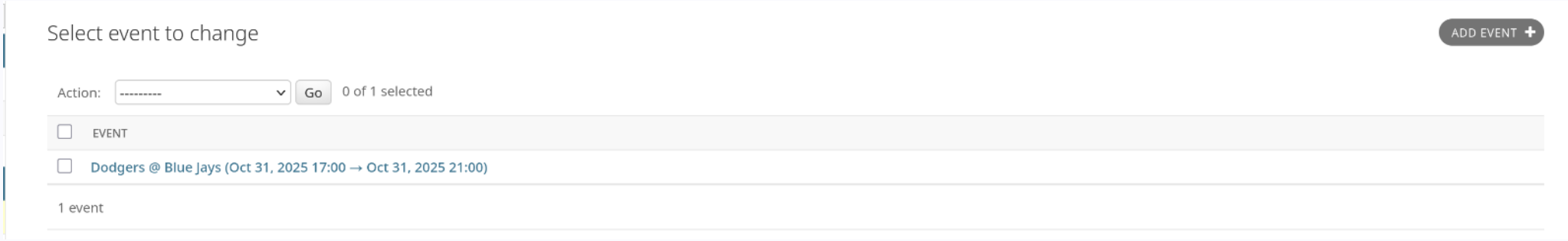




Adding to model



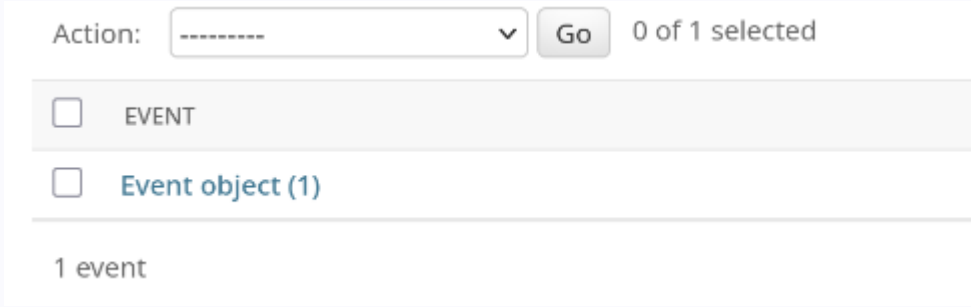

Home › Calendar_Events › Events › Add event
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
CALENDAR_EVENTS
Events + Add
Add event
Start datetime:
Date: 2025-10-31 Today |
Time: 17:00 Now |
End datetime:
Date: 2025-10-31 Today |
Time: 21:00 Now |
Event name:
Dodgers @ Blue Jays
Notes:
- Blue Jays lead 3-1
- Bring snacks
SAVE
Save and add another
Save and continue editing



Can have a nice identifier by creating a __str__(self) method

```
def __str__(self):
    start = timezone.localtime(self.start_datetime).strftime("%b %d, %Y %H:%M")
    end = timezone.localtime(self.end_datetime).strftime("%b %d, %Y %H:%M")
    return f"{self.event_name} ({start} → {end})"
```

Otherwise you get default:



Relationships: one-to-many

- Many of one thing can be associated to a single other.
- Use a ForeignKey field on the model that can only be associated to a single other model.

```
calendar = models.ForeignKey(Calendar,
    on_delete=models.CASCADE,
    null=True)
```

```python
from django.db import models
from django.utils import timezone
from django.core.validators import RegexValidator

HEX_COLOR_VALIDATOR = RegexValidator(
    regex=r"[0-9a-fA-F]{6}", message="Enter a valid hex color code, e.g. F100AB"
)

class Calendar(models.Model):
    name = models.CharField(max_length=255)
    colour_hex = models.CharField(max_length=6, validators=[HEX_COLOR_VALIDATOR])
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return str(self.name).title()

class Event(models.Model):
    start_datetime = models.DateTimeField()
    end_datetime = models.DateTimeField()
    event_name = models.CharField(max_length=255)
    notes = models.TextField()
    calendar = models.ForeignKey(Calendar, on_delete=models.CASCADE, null=True)
```

Other relationships

- One-to-one: use OneToOneField instead of ForeignKey.
- Many-to-many: use ManyToManyField instead of ForeignKey.

Django Shell

- We saw the python shell we can create with the **python** command. We can do something similar to get django's.

```
python manage.py shell
```

- The shell makes it easier for us to test out code and verify models without needing to use webpages.

Creating pages

1. Define URL(s)

2. Create view(s)

3. Create template(s)

Creating pages

1. Define URL(s)

2. Create view(s)

3. Create template(s)

Generally good practice to have each app its own urls.py

- Create a urls.py for each app and then include into the main project's urls.py

```
"""Defines URL patterns for
calendar_events"""

app_name = "calendar_events"
urlpatterns = []
```

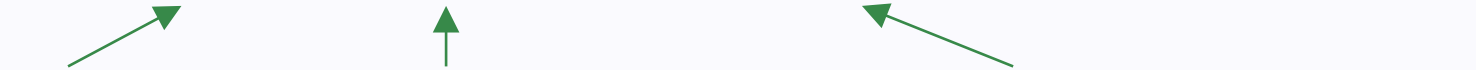
calendar_events/urls.py

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("admin/", admin.site.urls),
    path("",
include("calendar_events.urls")),
]
```

calendar_project/urls.py

Adding URL path

```
path("", views.index, name="index")
```




```
from django.urls import path

from . import views

app_name = "calendar_events"
urlpatterns = [
    # Home page
    path("", views.index, name="index")
]
```

Creating views

- Function that takes in a request object, gets and processed any data and then returns a rendered web page.

```
from django.shortcuts import render


def index(request):
    """The home page for Calendar"""
    return render(request, "calendar_events/index.html")
```

Creating template

- Template is the HTML that shows what the page looks like and can slot in any data that was passed by the view.

`calendar_events/templates/calendar_events`

- Create a folder inside app of *templates/{app_name}*.

```
<h1>Welcome to Calendar Central</h1>
<p>Last refreshed: XXXXX</p>

<h2>Calendars:</h2>
<!-- Show calendars -->
```

Template inheritance

- Some content may be repeated and as such can take advantage of template inheritance.

calendar_events/templates/calendar_events/base.html

```
<p><a href="{% url 'calendar_events:index' %}">Calendar Central</a></p>
<p>Last refreshed: XXXXX</p>

{% block content %}{% endblock content %}
```

```
{% extends 'calendar_events/base.html' %}

{% block content %}
<h1>Welcome to Calendar Central</h1>
<h2>Calendars:</h2>
{% endblock content %}
```

{% %} are template tags

View pass data to template with context

```
def index(request):
    calendars = Calendar.objects.prefetch_related(
        Prefetch("event_set",
queryset=Event.objects.order_by("start_datetime"))
    ).order_by("created_at")

    # Format the current datetime
    now = timezone.localtime(timezone.now())
    last_refreshed = now.strftime("%b %d, %Y at %H:%M:%S")

    context = {"calendars": calendars, "last_refreshed":
last_refreshed}
    return render(request, "calendar_events/index.html", context)
```

Context is just a Python dictionary

Use data in template w/ {{ name }}

```
{% extends 'calendar_events/base.html' %} {% block content %}
<h1>Welcome to Calendar Central</h1>

<h2>Calendars:</h2>
{% for calendar in calendars %}
<h3 style="color: #{{ calendar.colour_hex }}">{{ calendar.name }}</h3>
<ul>
  {% for event in calendar.event_set.all %}
  <li>{{ event }}</li>
  {% endfor %}
</ul>
{% endfor %} {% endblock content %}
```

Calendar Central

Last refreshed: Oct 31, 2025 at 01:23:29

Welcome to Calendar Central

Calendars:

Personal

- Dodgers @ Blue Jays (Oct 31, 2025 17:00 → Oct 31, 2025 21:00)

Get url encoded data

- You can get information from a URL into view by specifying in *urls.py*.

```
path("event/<int:event_id>", views.event, name="event")
```

We expect an integer after event which we will call “event_id”

- event_id is passed to the view function as second argument.

```
def event(request, event_id):
    event = Event.objects.get(id=event_id)
    context = {"event": event, "last_refreshed": get_datetime()}
    return render(request, "calendar_events/event.html", context)
```

All path converters

- str (default): any non-empty string excluding /
- int: 0 or any positive integer
- slug: any slug string (ASCII letters + numbers + hyphen + underscore)
- uuid: must include dashes and letters lowercase
- path: any non-empty string including /

Let’s Practice

- In your created mad_libs app I want you to create a page that outputs random generated mad libs. In Canvas you will find an archive folder with a template of the story and multiple files of different types of words. Your goal is for the root of your program to output a randomised version of the story using the words from the files chosen randomly using Python’s random library.