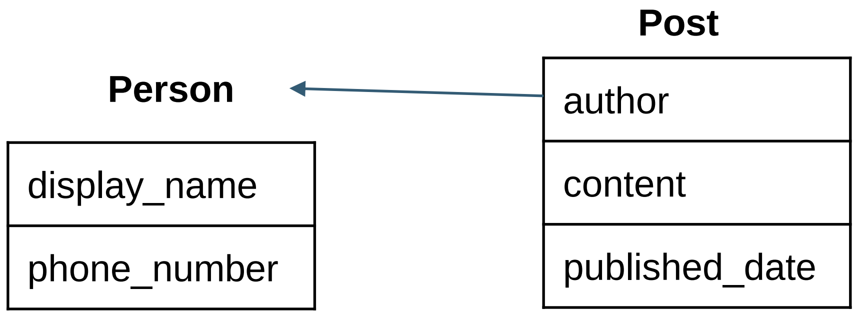
# Form, Authentication, Session, and Cookie

Tim Dosen PBP

## Review

# Django Models

- Models are a set of data stored to be processed in our apps. We store them in form of tables that connect to one another. We can create, read, update, and delete (CRUD) data from the tables using several specific command instructions called SQL.
- Do you know or remember what is object?



- A model class == a database table
- A model instance == a database table row

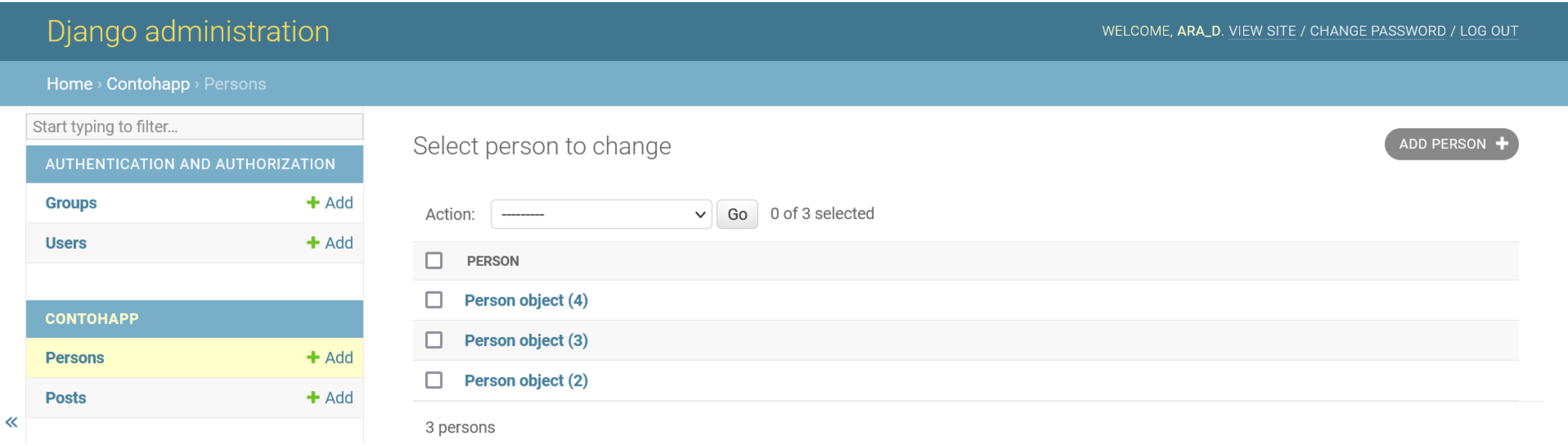
**Person**

| display_name | phone_number |
|---|---|
| Kak PeBePe | +628123456 |
| Budi | +6281676732 |

- https://docs.djangoproject.com/en/4.1/topics/db/models/

# Django Admin

- By default, the admin site url is http://localhost:8000/admin/



Django administration

WELCOME, ARA_D. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Contohapp › Persons

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

| | |
|---|---|
| Groups | + Add |
| Users | + Add |

CONTOHAPP

| | |
|---|---|
| Persons | + Add |
| Posts | + Add |

Select person to change

ADD PERSON +

Action: --------- Go 0 of 3 selected

| ☐ | PERSON |
|---|---|
| ☐ | Person object (4) |
| ☐ | Person object (3) |
| ☐ | Person object (2) |

3 persons

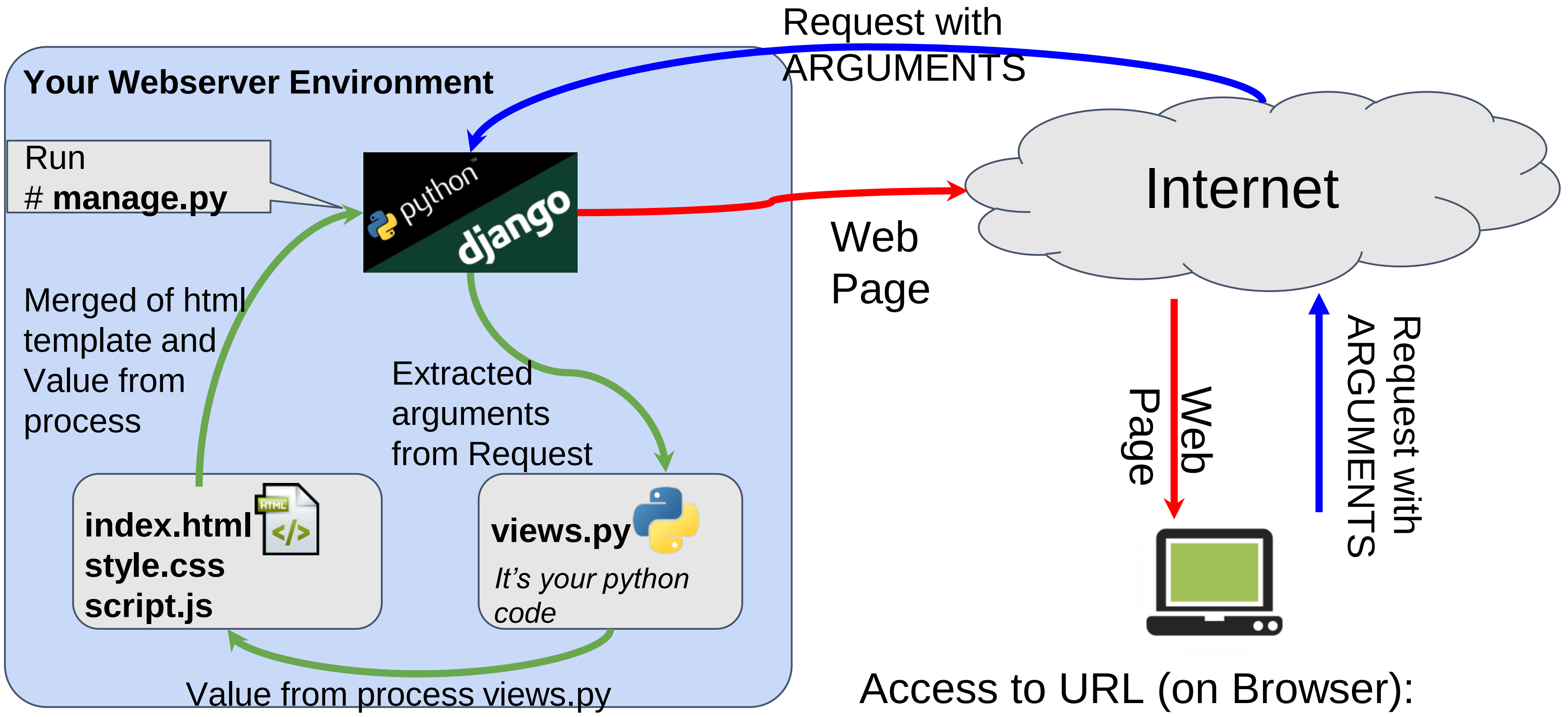
# HTTP Requests

- The most commonly used HTTP request methods are **GET**, **POST**, **PUT**, **PATCH**, and **DELETE**. These are equivalent to the **CRUD operations (create, read, update, and delete)**.
  - **GET**: GET request is used to read/retrieve data from a web server. GET returns an HTTP status code of **200 (OK)** if the data is successfully retrieved from the server.
  - **POST**: POST request is used to send data (file, form data, etc.) to the server. On successful creation, it returns an HTTP status code of **201**.
  - **PUT**: A PUT request is used to modify the data on the server. It replaces the entire content at a particular location with data that is passed in the body payload. If there are no resources that match the request, it will generate one.
  - **PATCH**: PATCH is similar to PUT request, but the only difference is, it modifies a part of the data. It will only replace the content that you want to update.
  - **DELETE**: A DELETE request is used to delete the data on the server at a specified location.

https://www.geeksforgeeks.org/different-kinds-of-http-requests/

## Passing Argument

## Passing Arguments



Access to URL (on Browser):
http://[appname].herokuapp.com/

## Passing Argument Methods

**POST**:

- Appends form-data inside the body of the HTTP request (data is not shown is in URL)
- Form submissions with POST cannot be bookmarked

**GET**:

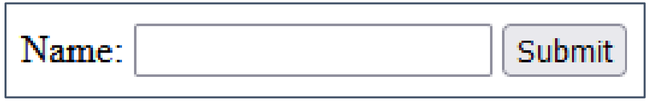
- Never use GET to send sensitive data! (will be visible in the URL)
- Useful for form submissions where a user want to bookmark the result
- GET is better for non-secure data, like query strings in Google

## Form

## About Form

```
<form action=[URL DESTINATION] method=[METHOD]>
            <input type=[INPUT TYPE] other attributes>
            ....
            ....
            <input type=[INPUT TYPE] other attributes>
</form>
```

```
<form action="http://www.sesuatu.com/proses"
method="POST">
            Name: <input type="text">
            <input type="submit" value="Submit">
</form>
```



**URL DESTINATION :**

Posting data to URL endpoint

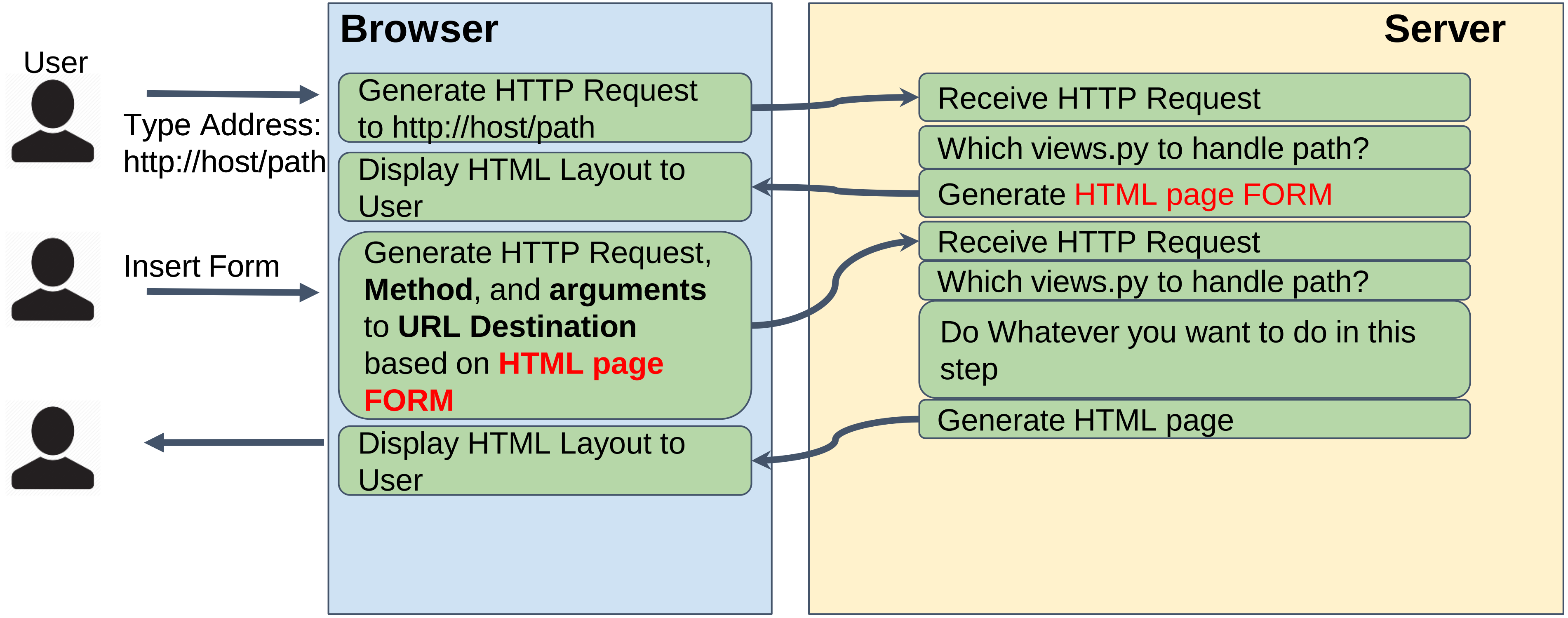
**METHOD :**

Method on passing variables to URL DESTINATION (GET or POST)

**INPUT & INPUT TYPE:**

Data attributes from Browser to Server

# Submission Flows


User
Type Address: http://host/path
Insert Form
Browser
Generate HTTP Request to http://host/path
Display HTML Layout to User
Generate HTTP Request, **Method**, and **arguments** to **URL Destination** based on **HTML page FORM**
Display HTML Layout to User
Server
Receive HTTP Request
Which views.py to handle path?
Generate HTML page FORM
Receive HTTP Request
Which views.py to handle path?
Do Whatever you want to do in this step
Generate HTML page

## Django HTML Form
## (Example Form definition in **forms.py**)

```
class Input_Form(forms.ModelForm):
        class Meta:
                model = Person
                fields = ['display_name']
        error_messages = {
                'required' : 'Please Type'
        }
        input_attrs = {
                'type' : 'text',
                'placeholder' : 'Nama Kamu'
        }
        display_name = forms.CharField(label='', required=False, max_length=27,
                widget=forms.TextInput(attrs=input_attrs))
```

https://docs.djangoproject.com/en/4.1/topics/db/models/#meta-options

https://docs.djangoproject.com/en/4.1/topics/forms/

## Django HTML Form
(Import from **forms.py** in **views.py**)

```
from .forms import Input_Form


def formulir(request):
    response = {'input_form' : Input_Form}
    return render(request, 'index.html', response)
```

## Django HTML Form
(Merge HTML parts from **forms.py** in HTML template)

Full html to render

```
<form action="savename" method="POST">
    {% csrf_token %}
  Name: {{ input_form.as_p }}
  <input type="submit" value="Submit">
</form>
```

penting untuk security
***Cross-Site Request Forgery***

# Django HTML Form
(Generated HTML parts from **forms.py**)

forms.py

```
class Input_Form(forms.ModelForm):
      class Meta:
            model = Person
            fields = ['display_name’]
      error_messages = {
            'required' : 'Please Type’
            }
      input_attrs = {
            'type' : 'text',
            'placeholder' : 'Nama Kamu’
            }
      display_name = forms.CharField(label='', required=False,
            max_length=27, widget=forms.TextInput(attrs=input_attrs))
```

HTML template

```
<form action="savename" method="POST">
    {% csrf_token %}
  Name: {{ input_form.as_p }}
  <input type="submit" value="Submit">
</form>
```

Generated HTML parts

```
        Name: <p> <input type="text" name="display_name"
placeholder="Nama Kamu" maxlength="27" id="id_display_name"></p>
```

Note that each form field has an ID attribute set to `id_<field-name>`

## Django HTML Form
(Validate POST arguments based on **forms.py**)

views.py

```
def savename(request):
    form = Input_Form(request.POST or None)
    if (form.is_valid and request.method == 'POST'):
        form.save()
        return HttpResponseRedirect('/')
    else:
        return HttpResponseRedirect('/')
```

## Web Application Threats

## Web Application Threats

- SQL Injection – the goal of this threat could be to bypass login algorithms, sabotage the data, etc.
- Denial of Service Attacks– the goal of this threat could be to deny legitimate users access to the resource
- Cross Site Scripting **XSS–** the goal of this threat could be to inject code that can be executed on the client side browser.
- Cookie/Session Poisoning– the goal of this threat is to modify cookies/session data by an attacker to gain unauthorized access.
- Form Tampering – the goal of this threat is to modify form data such as prices in e-commerce applications so that the attacker can get items at reduced prices.
- Code Injection – the goal of this threat is to inject code such as PHP, Python, etc. that can be executed on the server. The code can install backdoors, reveal sensitive information, etc.
- Defacement– the goal of this threat is to modify the page been displayed on a website and redirecting all page requests to a single page that contains the attacker's message.
- CSRF - the goal is to induce users to perform actions that they do not intend to perform
- Etc.

https://www.guru99.com/how-to-hack-website.html

## CSRF (Cross Site Request Forgery)

- Django has built-in protection against most types of Cross Site Request Forgery (CSRF) attacks
- Activated by default on MIDDLEWARE parameter in *settings.py*. The '***django.middleware.csrf.CsrfViewMiddleware***' will be executed before any view

https://docs.djangoproject.com/en/4.1/topics/security/

# CSRF in Web Form

- CSRF in HTML Template Form field:

```
<form action="savename" method="POST">
    {% csrf_token %}
  Name: {{ input_form.as_p }}
  <input type="submit" value="Submit">
</form>
```

- CSRF in *views.py*:

**With** CSRF check:

```
def my_view(request):
        # my implementation goes here
        return render(request, "a_template.html")
```

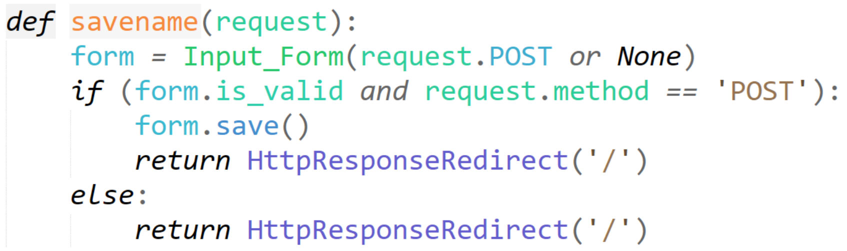
**Without** CSRF check:

```
@csrf_exempt
def my_view(request):
        # my implementation goes here
        return render(request, "a_template.html")
```

# XSS (Cross Site Scripting)

- Django’s Built in Security Features Cross Site Scripting (XSS) Protection
- XSS attacks allow a user to inject client side scripts into the browsers of other users
- Using Django templates protects you against the majority of XSS attacks
- Using is_valid() in Django templates, forms.py and views.py

```
>>> data = {“subject”: “hello”,
...         “message”: “<script>alert(‘Hacked’);</script>“,
...         “sender”: “foo@example.com”,
...         “cc_myself”: True}
>>> f = ContactForm(data)
>>> f.is_valid()
```



```
def savename(request):
    form = Input_Form(request.POST or None)
    if (form.is_valid and request.method == 'POST'):
        form.save()
        return HttpResponseRedirect('/')
    else:
        return HttpResponseRedirect('/')
```

## Discussion about XSS

- Django templates escape specific characters which are particularly dangerous to HTML
- While this protects users from most malicious input, it is not entirely foolproof, eg:

```
<div class={{ var }} >Halo</div>
```

- What happened if “var” was set to

```
“class1 onmouseover=alert(‘Hacked’)”
```

```
<div class=class1 onmouseover=alert(‘Hacked’)> Halo </div>
```

**Note**: be very careful when storing HTML in the database, especially when that HTML is retrieved and displayed.

## Authentication and Authorization

## Authentication and Authorization

- **Authentication** is the process of verifying who you are: login
- **Authorization** is the process of verifying that you have access to something

Question

Do we need both authentication and authorization?

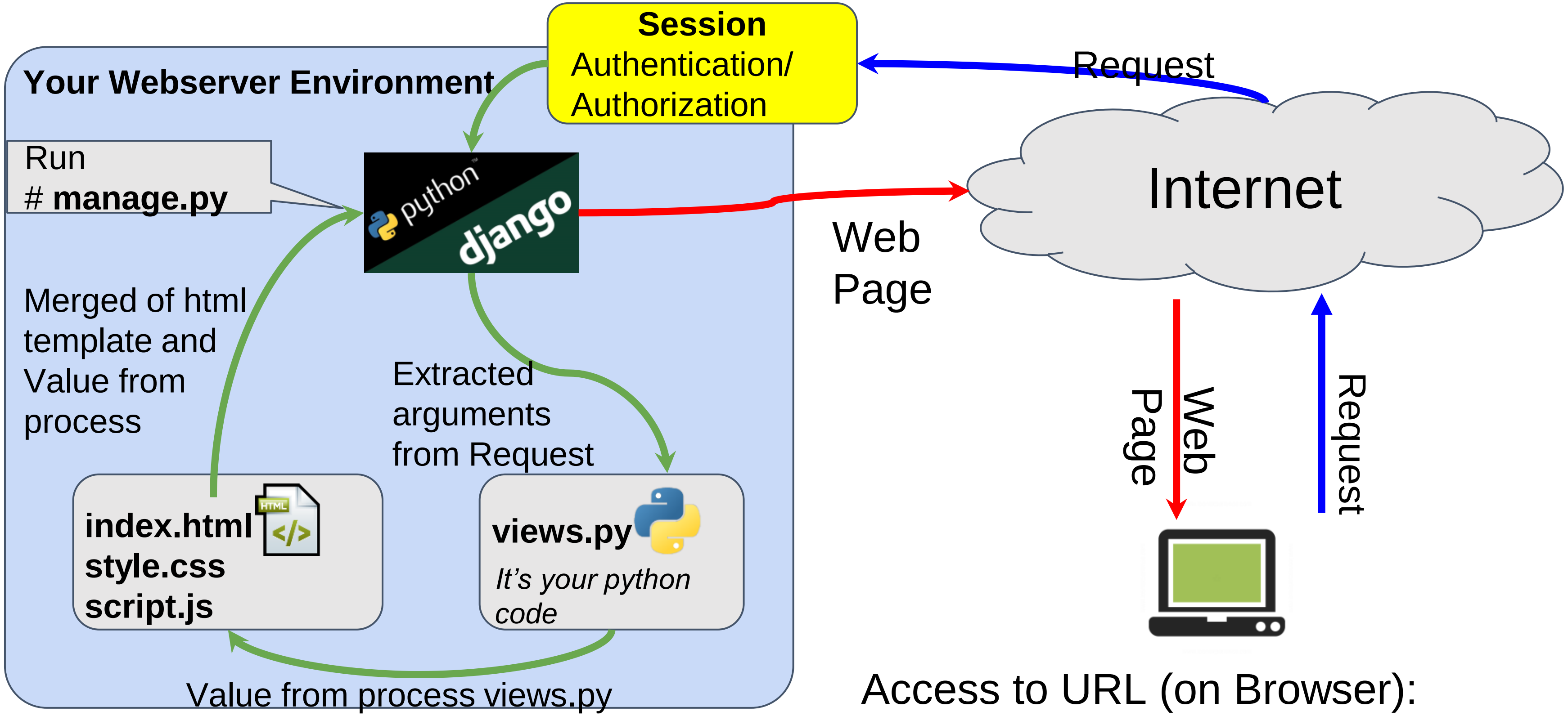
## Authentication is hard

Trying to write your own login system is difficult:

- How are you doing to save your password securely?
- How do you help with forgotten password?
- How do you make sure users set a good password?

Solution: Reuse Django User/Login module.
https://docs.djangoproject.com/en/4.1/topics/auth/default/

# Django Framework With Authentication



Access to URL (on Browser):
http://[appname].herokuapp.com/

# Reuse Django Framework Users Object

- Core of the authentication system
- **Primary attributes** of the default users are:

  username, password, email, first_name, last_name

- Creating users

```
from django.contrib.auth.models import User
user = User.objects.create_user(‘pebepe', ‘pebepe@pbp.com', ‘pbppassword’)
user.save()
```

- Creating superusers

```
$ python manage.py createsuperuser --username=kakpebebepe --email=pebepe21@pbp.com
```

## Changing Password

- Django does not store raw (clear text) passwords on the user model, but only a hash
- Do not attempt to manipulate the password attribute of the user directly.
- A helper function is used when creating a user.

```
>>> from django.contrib.auth.models import User

>>> u = User.objects.get(username=‘pewe')

>>> u.set_password('new password')

>>> u.save()
```

Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Eg: hash(‘ini password saya’) == ‘AF3627731ABE63FF’

## How to log a user in?

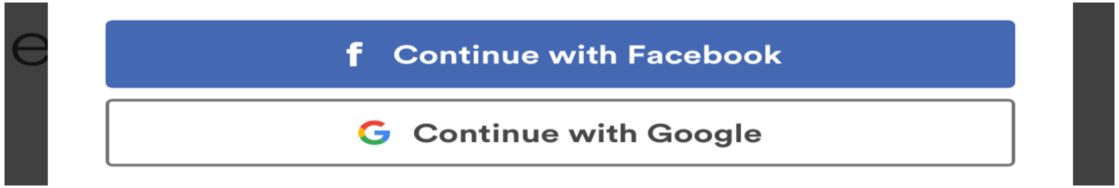
```
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(request, username=username, password=password)
    if user is not None:
        # Register in Session
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        # Sometimes return HTTP 403

           ...
```

## Alternative login mechanism

- By using available libraries ?
- Open Authentication (OAuth) ?

## OAuth (Open Authentication)

- OAuth is a standard for user authentication
- For users:
  - It allows a user to log into a website like AirBnB via some other service, like Gmail or Facebook
- For developers:
  - It lets you authenticate a user without having to implement log in
  - Examples: "Log in with Facebook"



## OAuth2

- Companies like Google, Facebook, Twitter, and GitHub have OAuth2 APIs:
  - Google Sign-in API
  - Facebook Login API
  - Twitter Login API
  - GitHub Apps/Integrations

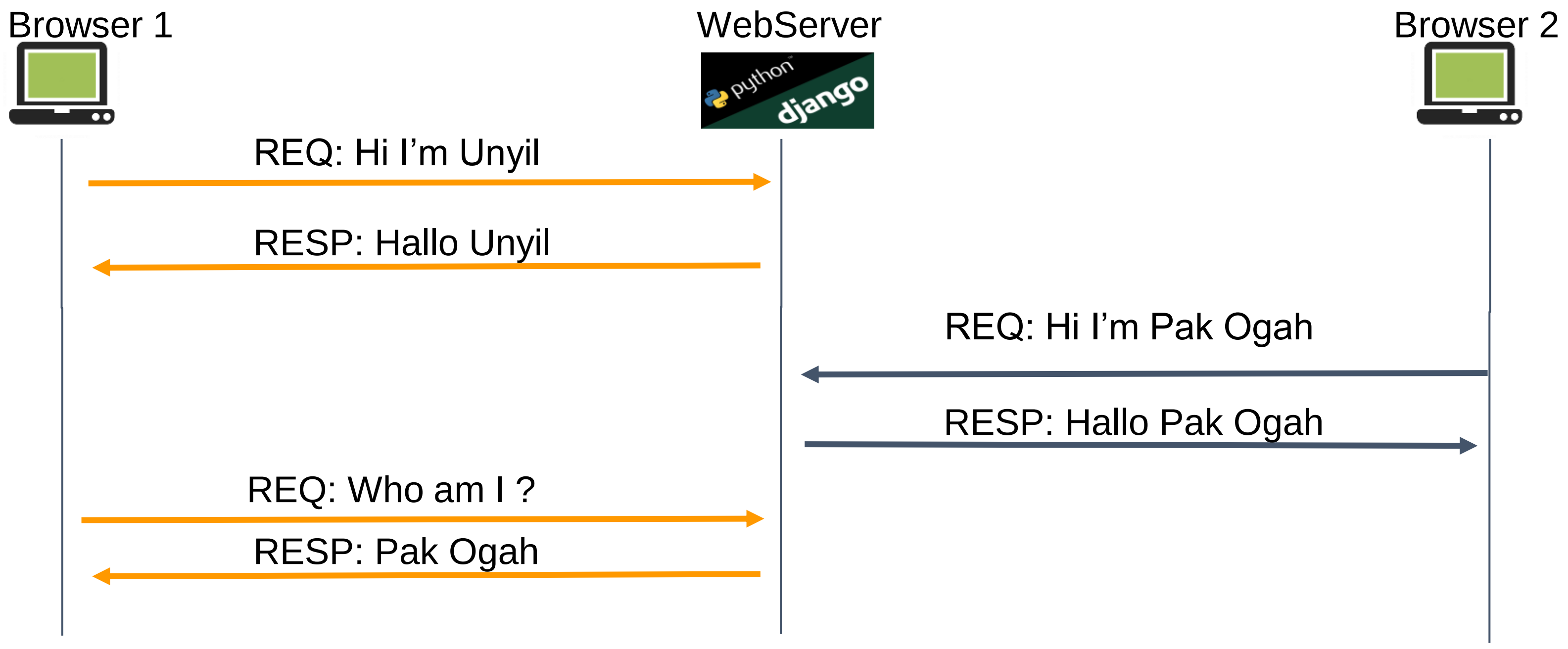
- OAuth2 is standardized, but the libraries that these companies provide are all different.
- You must read the documentation to understand how to connect via their API.

## Session and Cookie

## Session

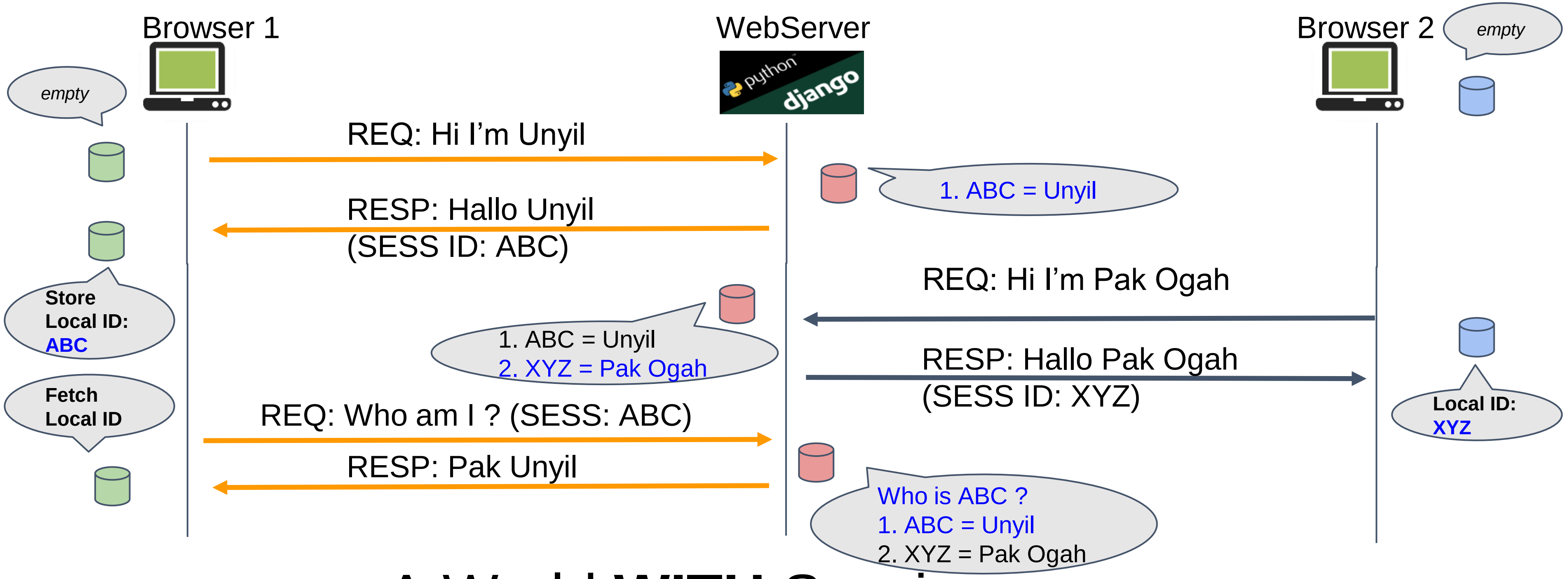
- Session: an abstract concept to represent a series of HTTP request and responses between a specific Web browser and server
  - HTTP doesn’t support the notion of a session
- How to implement Session concept?
  - Implement some code in ServerSide and ClientSide programming
    - ServerSide programming using **Server Session Database**
    - ClientSide programming using **Cookie** or **LocalStorage**
  - Server Session and Cookies need to work together to keep the session alive

## Why a webserver need to handle sessions?
## Why a web user needs to keep their session?



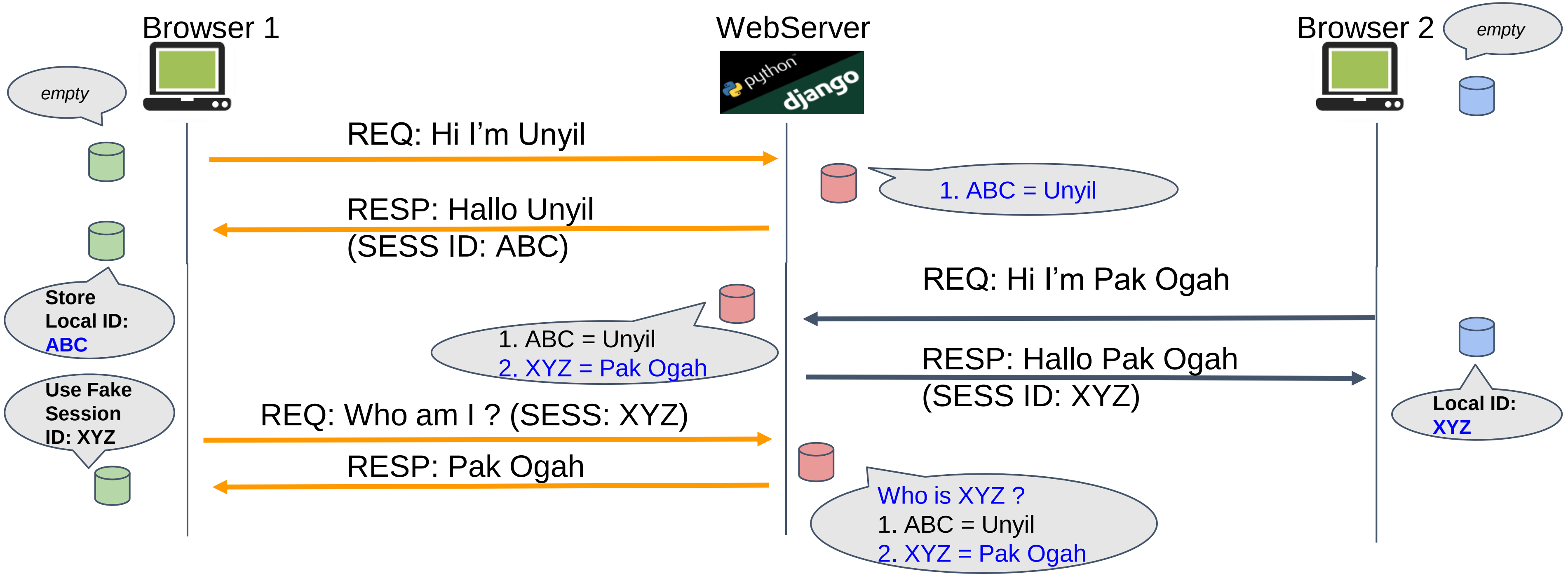
A World **WITHOUT** Session

## Why a webserver need to handle sessions?
## Why a web user needs to keep their session?



A World **WITH** Session

## Why a webserver need to handle sessions? Why a web user needs to keep their session?

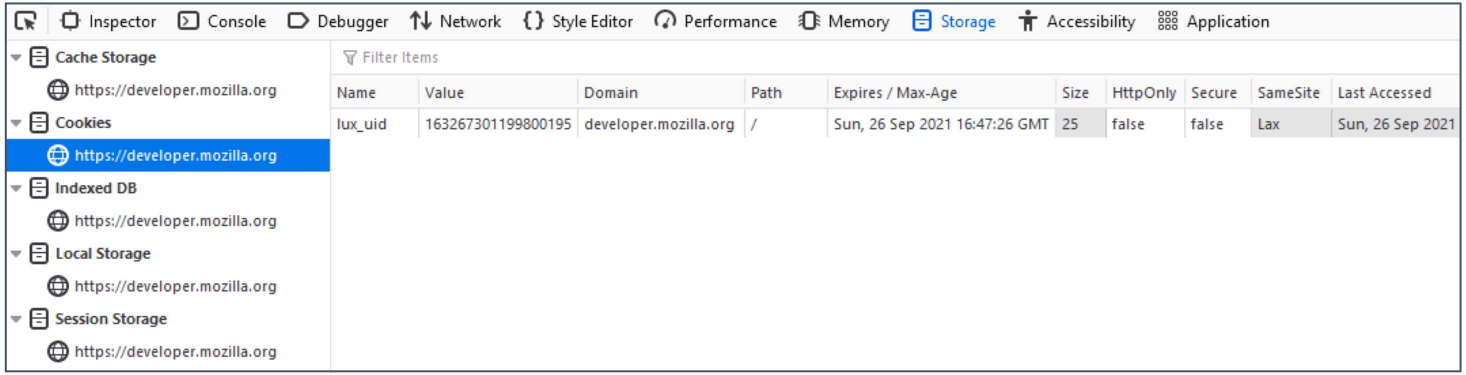

Browser 1
WebServer
Browser 2
empty
empty
python
django
REQ: Hi I'm Unyil
1. ABC = Unyil
RESP: Hallo Unyil (SESS ID: ABC)
Store Local ID: ABC
REQ: Hi I'm Pak Ogah
1. ABC = Unyil 2. XYZ = Pak Ogah
RESP: Hallo Pak Ogah (SESS ID: XYZ)
Local ID: XYZ
Use Fake Session ID: XYZ
REQ: Who am I ? (SESS: XYZ)
RESP: Pak Ogah
Who is XYZ ? 1. ABC = Unyil 2. XYZ = Pak Ogah

Using Fake Session ID (*Session forgery*)

NB: Do not try this at anywhere

# Cookie

- Small amount of information sent by a web server to a browser and then sent back by the browser on future page requests
- Cookies are used to:
  - Authentication
  - User tracking
  - Maintaining user preferences
- A cookie's data consists of a single name/value pair (like dictionary), sent in the header of the client's HTTP GET or POST request



| Name | Value | Domain | Path | Expires / Max-Age | Size | HttpOnly | Secure | SameSite | Last Accessed |
|---|---|---|---|---|---|---|---|---|---|
| lux_uid | 163267301199800195 | developer.mozilla.org | / | Sun, 26 Sep 2021 16:47:26 GMT | 25 | false | false | Lax | Sun, 26 Sep 2021 |

## Myths and Facts about Cookies

| Myths | Facts |
|---|---|
| • Like worms or viruses that can erase data from user harddisk | • Only data, not program code |
| • A form of spyware that can steal personal information | • Cannot erase or read information from the user’s computer |
| • Generate popups and spam | • Usually anonymous (don’t contain personal information) |

## Storing a Cookie

- Session cookie (default): temporary cookie
    - Stored in the browser memory
    - When the browser is closed, it will be erased
    - Can’t be used for tracking long-term information
    - Safer, only browser can access
- Persistent cookie:
    - Stored in a file on the browser’s computer
    - Can track long term information
    - Less secure: users or programs can open cookie files

Please try:
Open multiple tab for the same website
Open multiple browser for the same website,

## Server Session

- Server Session in Django, consists of
  - Database (a model to keep session data)
  - Webserver automatic handling
    - Read cookie parameter passed by Browser
    - Fetch / store data in session model
    - Modify information in the session
    - Send corresponding cookie back to Browser
- Implementation in Django:
  - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Sessions#Enabling_sessions
  - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Sessions#Using_sessions

## Additional Materials

# Model Relationship

Pelajari contoh *model relationship* yang ada di Django Documentation
https://docs.djangoproject.com/en/4.1/topics/db/examples/

- **Many-to-many relationships**
  - To define a many-to-many relationship, use **ManyToManyField**.
  - In this example, an **Article** can be published in multiple **Publication** objects, and a **Publication** has multiple **Article** objects.
- **Many-to-one relationships**
  - To define a many-to-one relationship, use **ForeignKey**.
  - In this example, a **Reporter** can be associated with many **Article** objects, but an **Article** can only have one **Reporter** object.
- **One-to-one relationships**
  - To define a one-to-one relationship, use **OneToOneField**.
  - In this example, a **Place** optionally can be a **Restaurant**.

## References

- Django Documentation:
    - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
    - https://docs.djangoproject.com/en/4.1/topics/forms/
    - https://docs.djangoproject.com/en/4.1/topics/db/models/#meta-options
    - https://docs.djangoproject.com/en/4.1/ref/csrf/
    - https://docs.djangoproject.com/en/4.1/topics/auth/
    - https://docs.djangoproject.com/en/4.1/topics/auth/default/
    - https://docs.djangoproject.com/en/4.1/topics/security/
    - https://docs.djangoproject.com/en/4.1/topics/db/examples/
- https://courses.cs.washington.edu/courses/cse190m/10su/lectures/slides/lecture22-cookies.shtml
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Forms
- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Sessions
- https://www.guru99.com/how-to-hack-website.html
- https://www.w3schools.com/tags/ref_httpmethods.asp