django

3rd SESSION

What will be covered?

- Static files
- Static assets
- Working with forms
  - PUT
  - DELETE
  - Search
- Sessions
- Cookies
- Working with APIs (Application Programming Interface)

Static files / Static assets

Configuring static files

1. Make sure that **`django.contrib.staticfiles`** is included in your **`INSTALLED_APPS`**.
2. In your settings file, define **`STATIC_URL`**, for example:

```
STATIC_URL = '/static/'
```

3. In your templates, use the **`static`** template tag to build the URL for the given relative path using the configured **`STATICFILES_STORAGE`**.

```
{% load static %}
<img src="{% static 'my_app/example.jpg' %}" alt="My image">
```

4. Store your static files in a folder called **`static`** in your app. For example **`my_app/static/my_app/example.jpg`**.

Static files / Static assets

Your project will probably also have static assets that aren't tied to a particular app. In addition to using a **`static/`** directory inside your apps, you can define a list of directories (**`STATICFILES_DIRS`**) in your settings file where Django will also look for static files. For example:

```
STATICFILES_DIRS = [
    BASE_DIR / "static",
    '/var/www/static/',
]
```

Static files / Static assets

Serving static files during development

If you use **`django.contrib.staticfiles`** as explained above, **`runserver`** will do this automatically when **`DEBUG`** is set to **`True`**. If you don't have **`django.contrib.staticfiles`** in **`INSTALLED_APPS`**, you can still manually serve static files using the **`django.views.static.serve()`** view.

This is not suitable for production use! For some common deployment strategies, see Deploying static files.

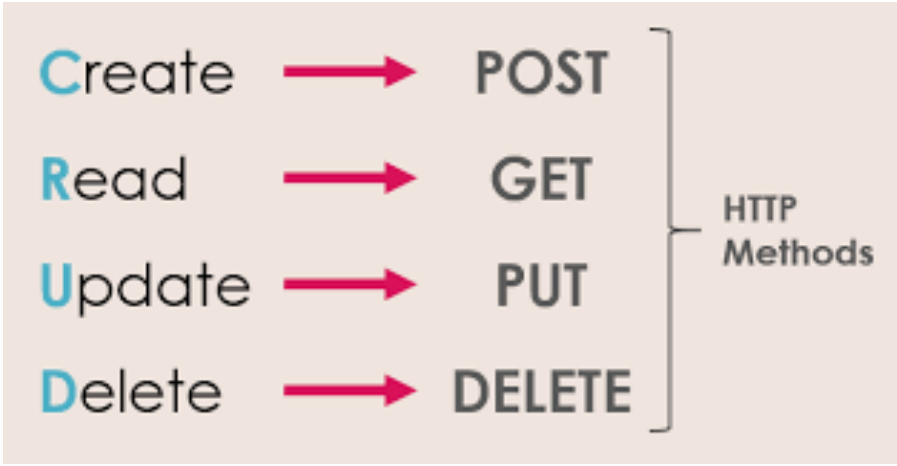
For example, if your **`STATIC_URL`** is defined as **`/static/`**, you can do this by adding the following snippet to your urls.py:

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ... the rest of your URLconf goes here ...
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Working with forms

- GET, POST, PUT, DELETE
- Search



Sessions

How to use sessions

Django provides full support for anonymous sessions. The session framework lets you store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the sending and receiving of cookies. Cookies contain a session ID – not the data itself (unless you're using the cookie based backend).

Enabling sessions

Sessions are implemented via a piece of middleware.

To enable session functionality, do the following:

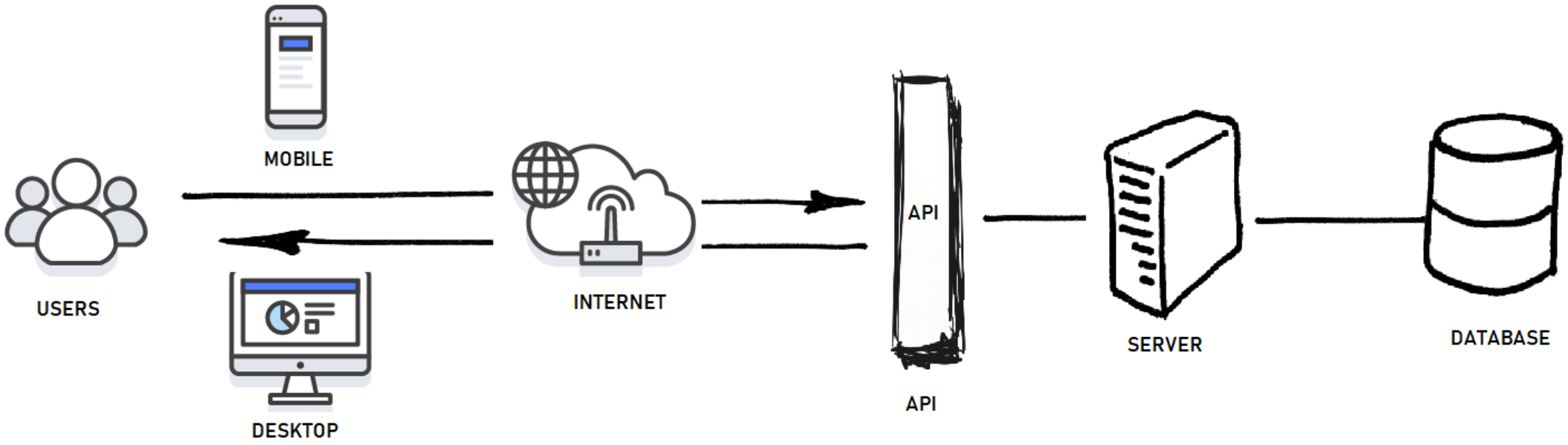
- Edit the **`MIDDLEWARE`** setting and make sure it contains **`'django.contrib.sessions.middleware.SessionMiddleware'`**. The default **`settings.py`** created by **`django-admin startproject`** has **`SessionMiddleware`** activated.

If you don't want to use sessions, you might as well remove the **`SessionMiddleware`** line from **`MIDDLEWARE`** and **`'django.contrib.sessions'`** from your **`INSTALLED_APPS`**. It'll save you a small bit of overhead.

Cookies

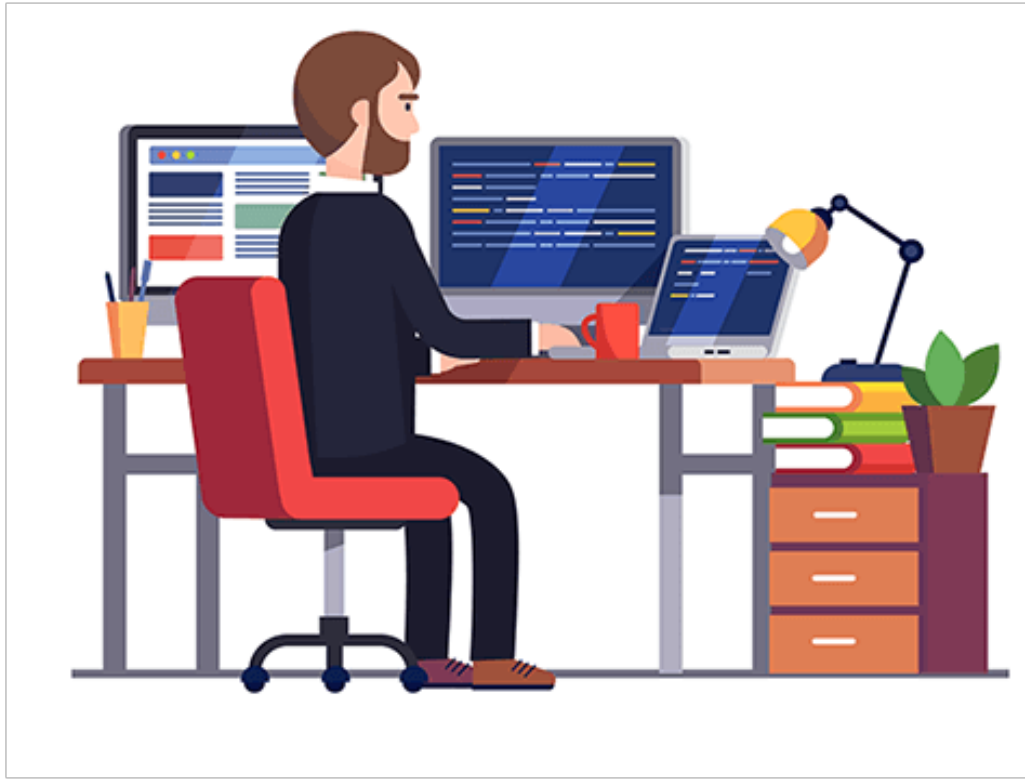
- Cookies are saved on the browser (clients computer)
- Django
  - Create response
  - Create cookies
- How we can get cookie values (through their keys)
  - request.cookies.get(‘key’)

Working with APIs



We can create APIs with: Django REST Framework, FastAPI, etc.

Let’s code



QUESTIONS