Tarea 8

No hace falta instalar nada

Autentificación, Autorización, y registro de eventos

El Django allouth no tiene Bootstrap (ponerlo en Bootstrap)
https://github.com/pennersr/django-allauth

En la carpeta

Podemos personalizar los templates, usando los que tiene en templates y poniendolos en una carpeta templates/accounts.

es donde busca los TEMPLATES: TENER CUIDADO CON LOS TEMPLATES

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

EN LOS TEMPLATES puedes poner user-name, user authentificate

No tenemos que hacer lo del principio (https://docs.djangoproject.com/en/2.2/topics/auth/default/), ya lo hace Django solo.

- Authentication in Web requests: si queremos autentificarnos

```
if request.user.is_authenticated:
    # Do something for authenticated users.
    ...
```

```
else:
    # Do something for anonymous users.
    ...
```

- How to log a user in: esto esta hecho ya
- The login_required decorator: para saber que funciones tienen acceso a la autentificación, poner "@login_required"

from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
…

- Ejemplo de HTML:

```
{% extends "base.html" %}

{% block content %}

{% if form.errors %}
<p>Your username and password didn't match. Please try again.</p>
{% endif %}

{% if next %}
    {% if user.is_authenticated %}
    <p>Your account doesn't have access to this page. To proceed,
    please login with an account that has access.</p>
    {% else %}
    <p>Please login to see this page.</p>
    {% endif %}
{% endif %}

<form method="post" action="{% url 'login' %}">
{% csrf_token %}
<table>
<tr>
    <td>{{ form.username.label_tag }}</td>
    <td>{{ form.username }}</td>
</tr>
<tr>
    <td>{{ form.password.label_tag }}</td>
    <td>{{ form.password }}</td>
</tr>
</table>

<input type="submit" value="login">
<input type="hidden" name="next" value="{{ next }}">
```

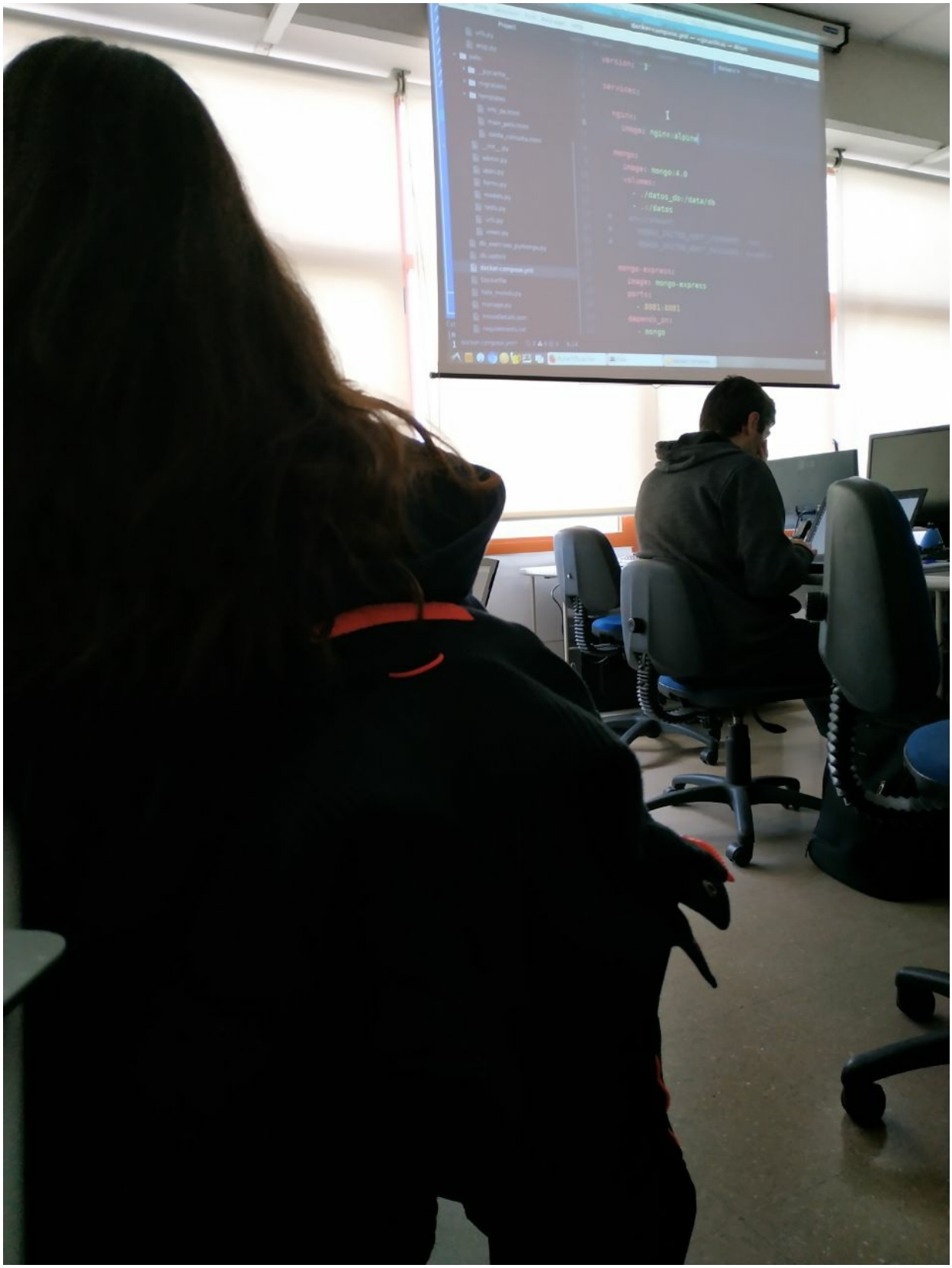
```
</form>

{# Assumes you setup the password_reset view in your URLconf #}
<p><a href="{% url 'password_reset' %}">Lost password?</a></p>

{% endblock %}
```

Este detallito es necesario para el despliegue, siempre que se usen contraseñas es necesario usar HTTPS, para su cifrado.

Añadir nuevo serviciod al docker-compose.yml:

```
nginx:
          image: nginx:alpine
          ports:
            - "80:80"
            - "443:443"
          volumes:
            - ./conf:/etc/nginx/conf.d:ro
            - ./cert:/etc/ssl/private:ro
          depends_on:
            - web
```

Docker compose se añade lo de nginx

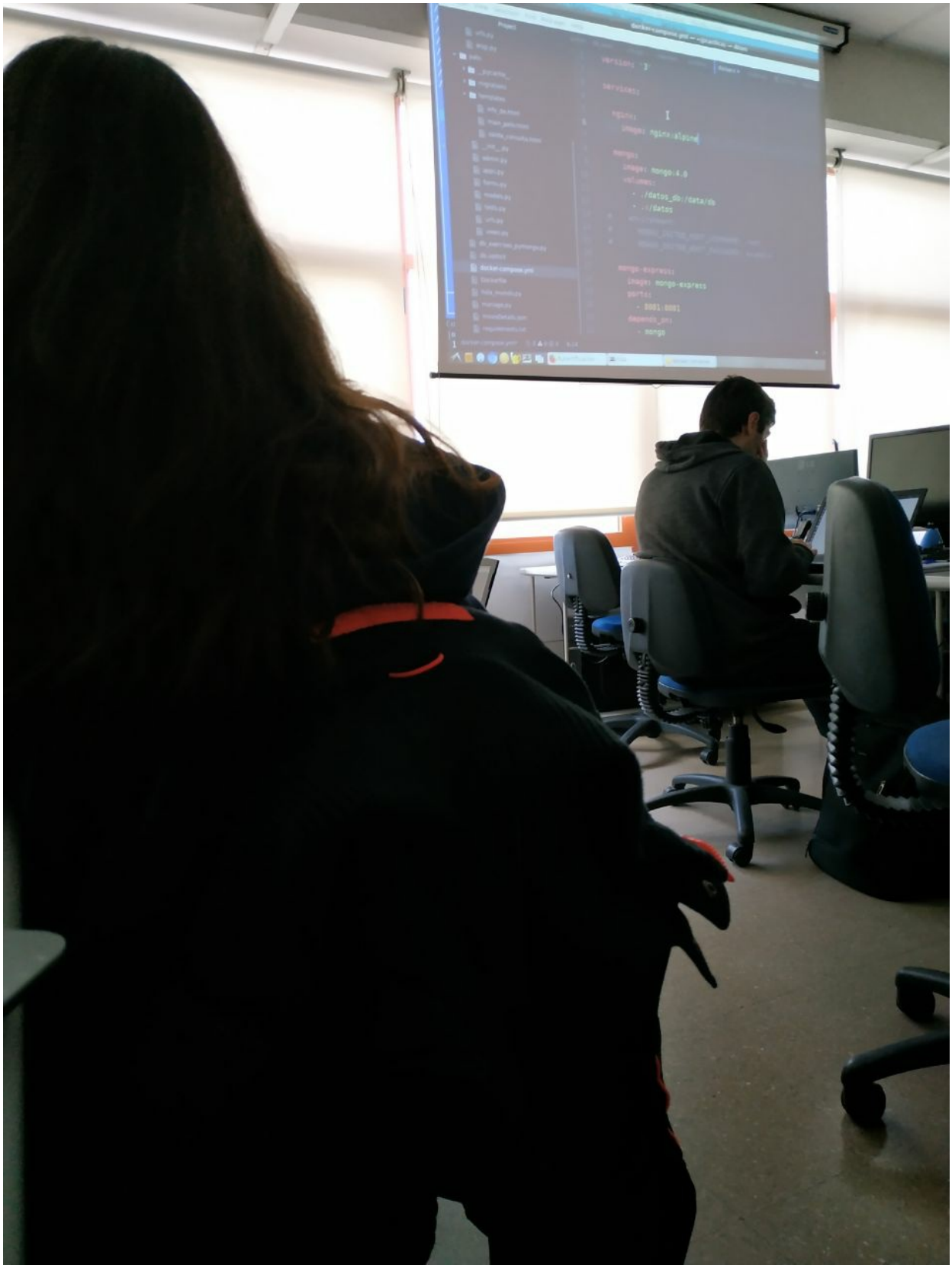


Y el puerto 8000 no hace falta ponerlo, entra internamente y ya estamos desde nginx

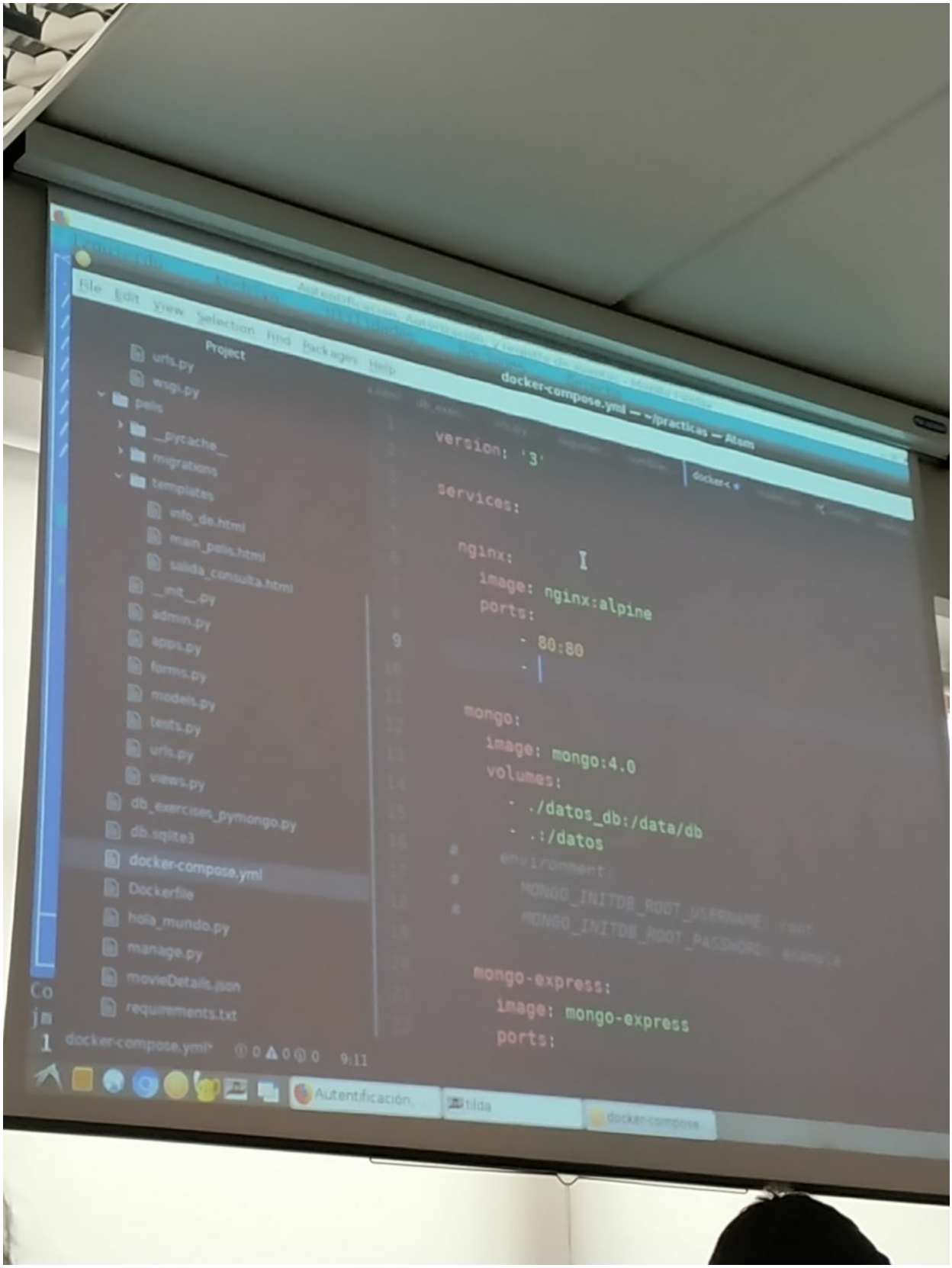
web:
build: .
command: python manage.py runserver 0.0.0.0:8000

```
volumes:
- .:/code
#ports:
# - 8000:8000
links:
- mongo
depends_on:
- mongo
```

Ahora lo del puerto 80 de la web no hace falta



Ponemos los puertos, además ahí ponemos el 443 también

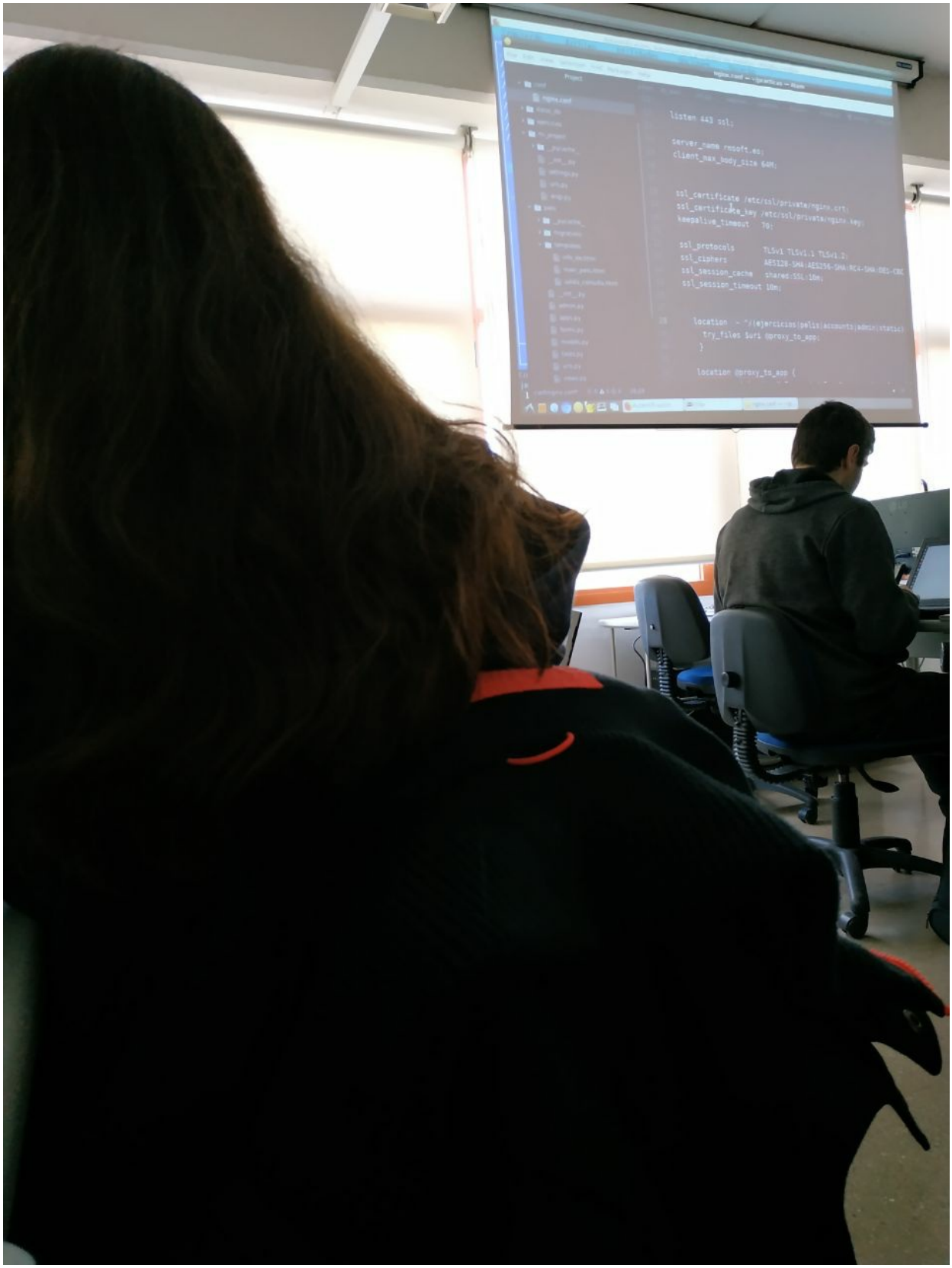


En el directorio cert, pondremos un par de archivos con una pareja de claves generadas a este próposito, y en el directorio conf el archivo de configuración de nginx:

clave publica - clave privada

```
server {
  listen 80 default_server;
  server_name _;

  # redirecciona todo a https
  return 301 https://$host$request_uri;
}

server {
  listen 443 ssl;
  server_name _;

  # la pareja de claves
  ssl_certificate /etc/ssl/private/nginx.crt;
  ssl_certificate_key /etc/ssl/private/nginx.key;
  keepalive_timeout   70;

  ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers         AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
  ssl_session_cache   shared:SSL:10m;
  ssl_session_timeout 10m;

  location  ~ ^/(miapp|admin|accuonts) {
        try_files $uri @proxy_to_app;
  }

  # proxy inverso
  location @proxy_to_app {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass   http://web:8000;
  }
```

Fichero de configuración nginx.conf copia y pega



y hay que poner en location:

```
location  ~ ^/(ejercicios|pelis|admin|accuonts) {
  try_files $uri @proxy_to_app;
```

```
    }
```

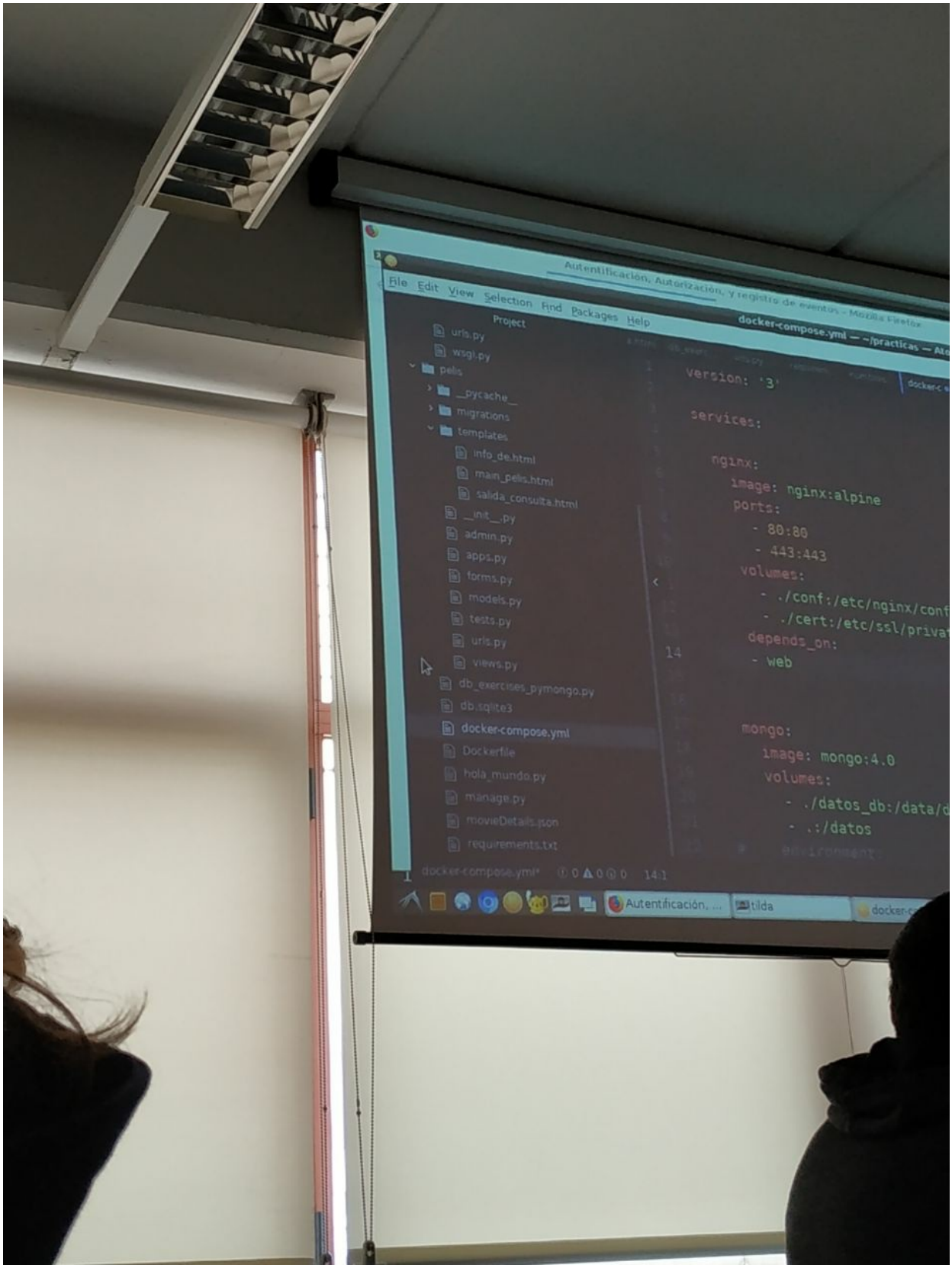
location ~ ^/(ejercicios|pelis|admin|accuonts) {
try_files $uri @proxy_to_app;
}

proxy inverso: pasar el rquirimiento tal cual a nuestra aplicación, y lo pasa al 8000

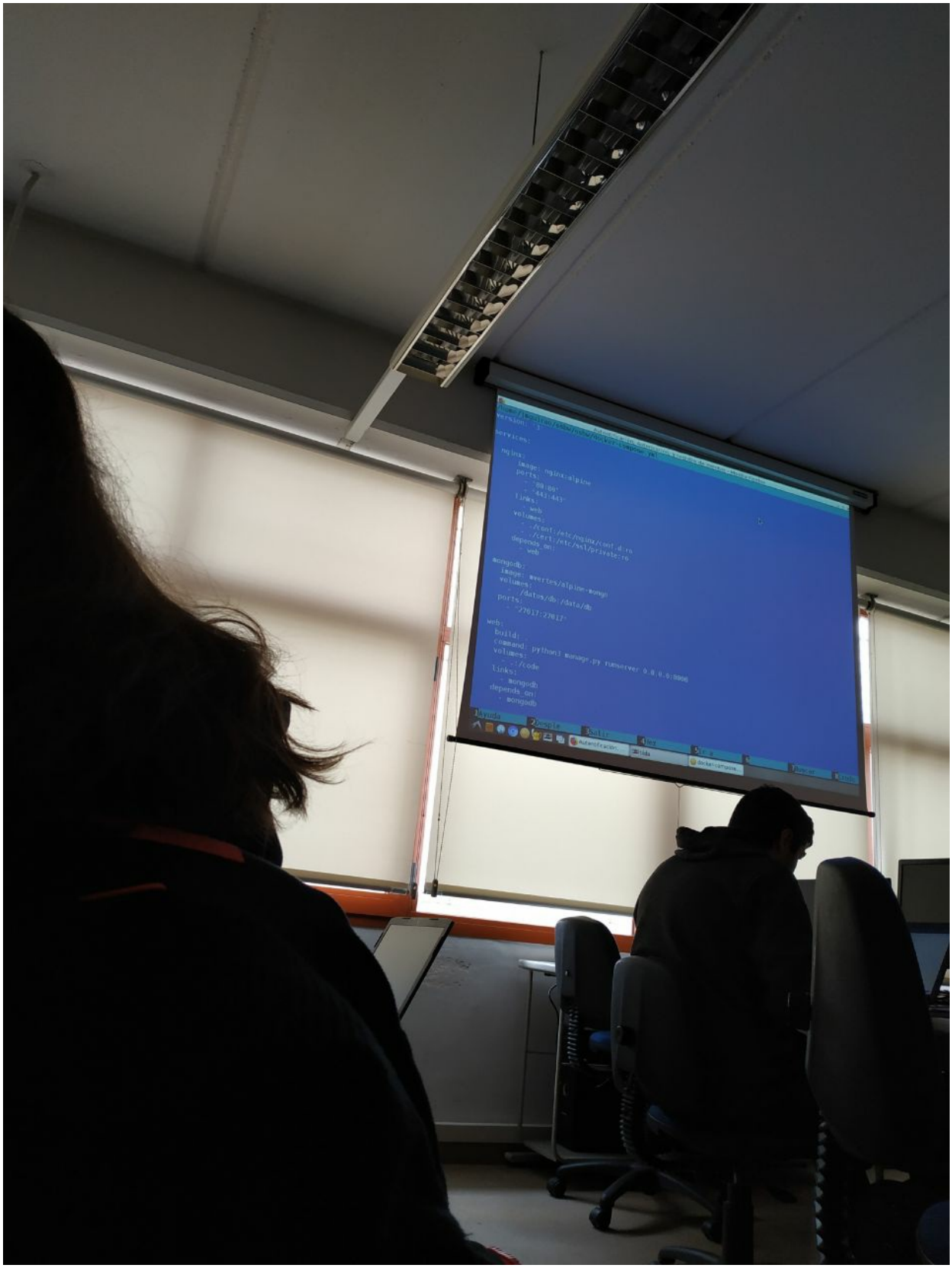
Tienes que generar la clave privada y publica y estaran en una carpeta cert:

ssl_certificate /etc/ssl/private/nginx.crt;
ssl_certificate_key /etc/ssl/private/nginx.key;

En nginx.key está la clave privada



Y en nginx.crt la publica



Para crear en mac la clave privada y publica

https://nickolaskraus.org/articles/how-to-create-a-self-signed-certificate-for-nginx-on-macos/

```
mkdir -p /usr/local/etc/ssl/private
mkdir -p /usr/local/etc/ssl/certs

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /usr/local/etc/ssl
```

Ahora después de aquí va docker compose up

Registro de eventos

No hace falta instalar nada

Para el log, hacemos un import

Y eso muestra logs por pantalla pq hemos usado el debug

Y así está configurado el módulo de python

Ponemos "import logging" en views.py
(https://docs.djangoproject.com/en/2.2/topics/logging/)

y pegamos en settings.py

LOG_FILE = 'mi_archivo_de.log'

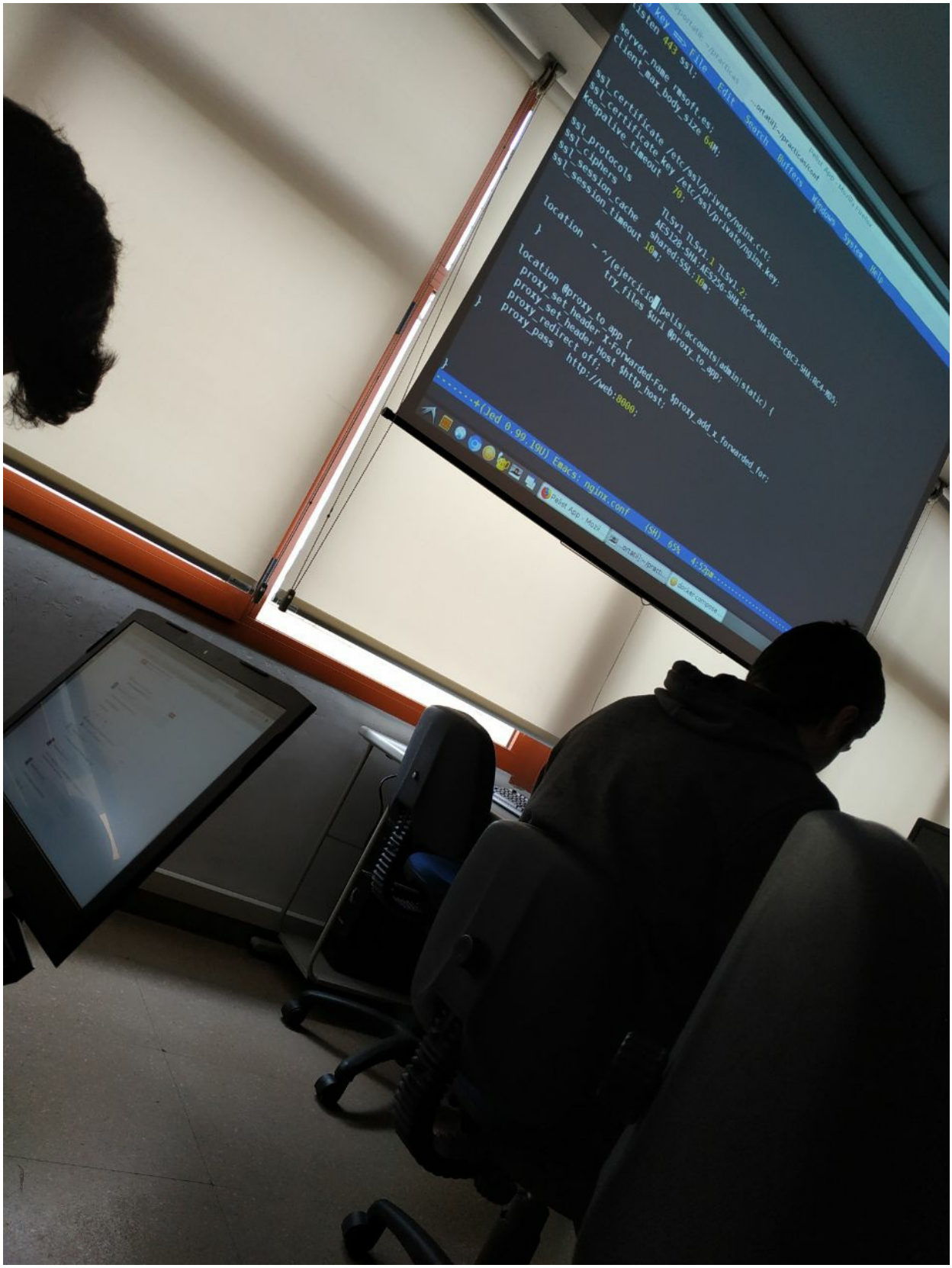
LOGGING = {
'version': 1,
'disable_existing_loggers': False,

```
    'formatters': {
         'verbose': {
           'format' : "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s] %(message)s"
            'datefmt' : "%d/%b/%Y %H:%M:%S"
          },
         'simple': {
           'format': '%(levelname)s [%(name)s:%(lineno)s] %(message)s'
             },
         },

         'handlers': {
             'file': {
                 'level': 'INFO',
```

```
            'class': 'logging.FileHandler',
            'filename': os.path.join(BASE_DIR, LOG_FILE),
            'formatter': 'verbose',
            'mode':'w'
        },
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        }
    },

    'loggers': {
        'django': {
            'handlers':['file'],
            'propagate': True,
            'level':'ERROR',
         },
         'mi_instagram': {
            'handlers': ['file', 'console'],
             'level': 'DEBUG',
          },
      }
}
```

y ponemos:
'pelis': {
'handlers': ['file', 'console'],
'level': 'DEBUG',
},

Ahora copia y pega el registro de eventos de swad. Se copia en settings.py

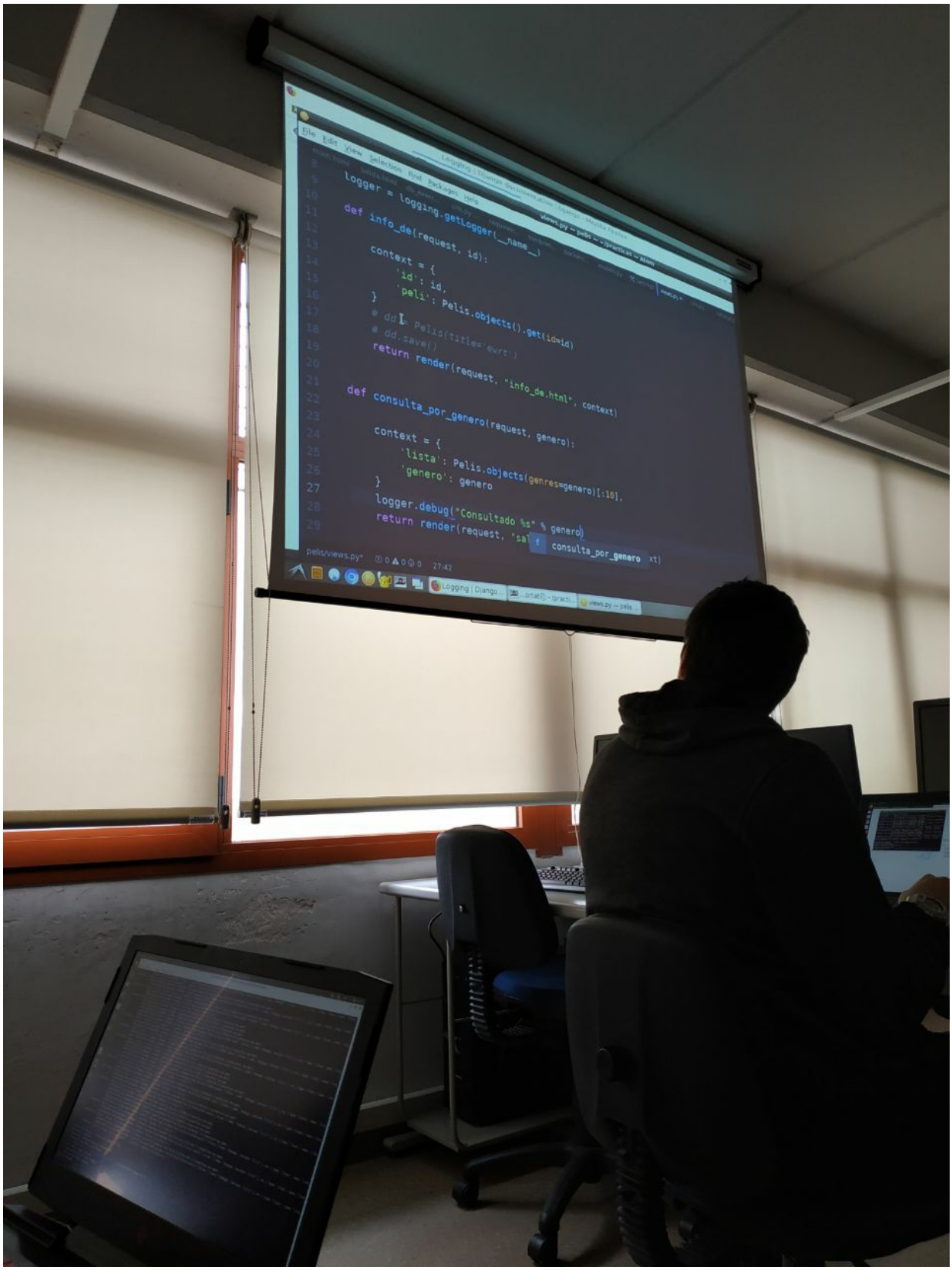


Y para usarlo: "logger.debug" o los que haya:

- logger.debug()
- logger.info()
- logger.warning()

- logger.error()
- logger.critical()

Y ponemos el logging en las funciones:



Y tiene que salir https cuando le das

Welcome to django allouth

Va a instalar el django-allauth

https://django-allauth.readthedocs.io/en/latest/installation.html

Para ello ponemos en el requirements

```
django-allauth
```