



Școala
informală
de IT

Python Development

Web Apps Intro



Școala
informală
de IT

Cuprins

1. Ce este o aplicație web?
2. Arhitectură client-server
3. HTML & CSS
4. Django Framework
5. Arhitectură MVC/MTV
6. Django ORM
7. Django fixtures
8. Django views & templates



Ce este o aplicație web?

Web Apps Intro



Ce este o aplicație web?

- O aplicație web este un program care rulează în browser.
- Aplicația web este diferită de un website, dar această diferență este foarte mică.
- Pentru a diferenția cele două concepte, o să avem în vedere următoarele caracteristici ale unei aplicații web:
 - adresează o problemă concretă (chiar dacă aceasta este simpla afișare a unor informații)
 - este interactivă
 - are un CMS (Content Management System)
- Internetul funcționează pe baza unei rețele de rutare a pachetelor de date în conformitate cu anumite protocoale: Internet Protocol (IP), Transport Control Protocol (TCP) și altele.
- Un protocol este un set de reguli care specifică cum calculatoarele trebuie să comunice între ele în cadrul unei rețele.



Ce este o aplicație web?

- Internet Protocolul (IP) este un set de reguli necesare rutării și adresării pachetelor de date ca acestea să poată ajunge la destinație.
- Datele care parcurg Internetul sunt divizate în părți mici numite pachete.
- Fiecare mașină din rețea are un identificator unic. Acesta este folosit în comunicarea tuturor echipamentelor conectate la internet.
- Acest identificator unic este numit adresă IP.
- Există două standarde pentru adresele IP:
 - IP Version 4 (IPv4)
 - IP Version 6 (IPv6)



Ce este o aplicație web?

- IPv4 utilizează 32 de biți binari cu ajutorul cărora crează o adresă unică. Este reprezentată de patru numere separate prin spațiu, fiecare număr este un număr zecimal (baza 10) care reprezintă un număr binar pe 8 biți (numit octet).

Ex: 192.169.2.105

- IPv6 utilizează 128 de biți binari cu ajutorul cărora crează o adresa unică. O adresă IPv6 este compusă din opt grupuri de numere hexazecimale (baza 16) separate de semnul :

Ex: 2001 : cdba : 0000 : 0000 : 0000 : 0000 : 3257 : 9652

- Fiecare grup format din 0 zero-uri poate fi omis pentru a economisi spațiu, lăsând semnul : pentru a marca lipsa.

Ex: 2001 : cdba :: 3257 : 9652

- Compresia unei adrese IPv6 prin eliminarea grupurilor formate din zero-uri poate fi aplicată o singură dată, altfel nu s-ar putea reface adresa completă.



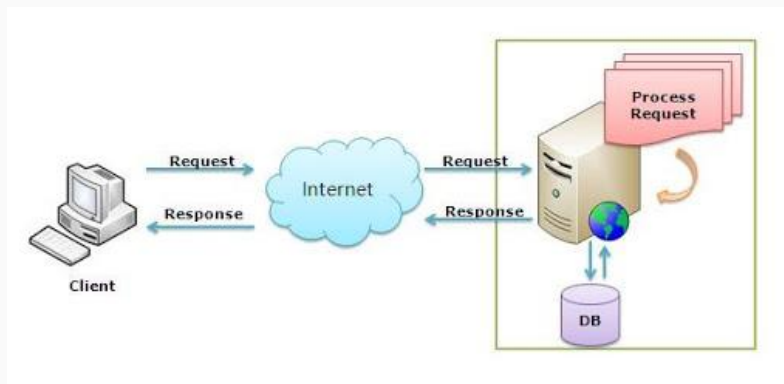
Ce este o aplicație web?

- Transfer Control Protocol (TCP) este un protocol care definește și menține modul de transmitere a datelor într-o rețea în care echipamentele schimbă date.
- Împreună cu protocolul IP, cele două protocoale sunt conceptele de bază care definesc regulile conform cărora funcționează internetul.
- TCP este un protocol orientat pe conexiunea dintre echipamentele conectate la o rețea, prin care o conexiune este stabilită între două echipamente până când cele două echipamente au terminat de schimbat date (mesaje).
- Determină:
 - cum se împart datele în pachete pe care rețeaua le poate distribui.
 - trimite pachete de date către rețea și primește pachete de date de la aceasta.
 - gestionează fluxul de date.
 - asigură retransmiterea pachetelor pierdute/alterate și confirmarea tuturor pachetelor primite.



Ce este o aplicație web?

- Un web request începe din momentul în care utilizatorul cere încărcarea unei pagini/aplicații web prin introducerea unei adrese în browser.
- Browser-ul preia URL-ul și îl parsează pentru a ști adresa serverului căruia îi corespunde resursa respectivă.
- Astfel URL-ul ajunge la DNS (Domain Name Center), un repository de domenii și adresele IP asociate acestora.
- Dacă adresa a mai fost accesată înainte, browser-ul folosește memoria cache.
- În final, browser-ul trimite request-ul către server.
- O dată ajuns la server, request-ul nostru este preluat de acesta și interpretat pentru a localiza pagina/aplicația cerută și toate datele aferente acesteia.
- După identificarea paginii/aplicației dorite, datele sunt trimise înapoi către browser, iar acesta afișează și formatează datele obținute.



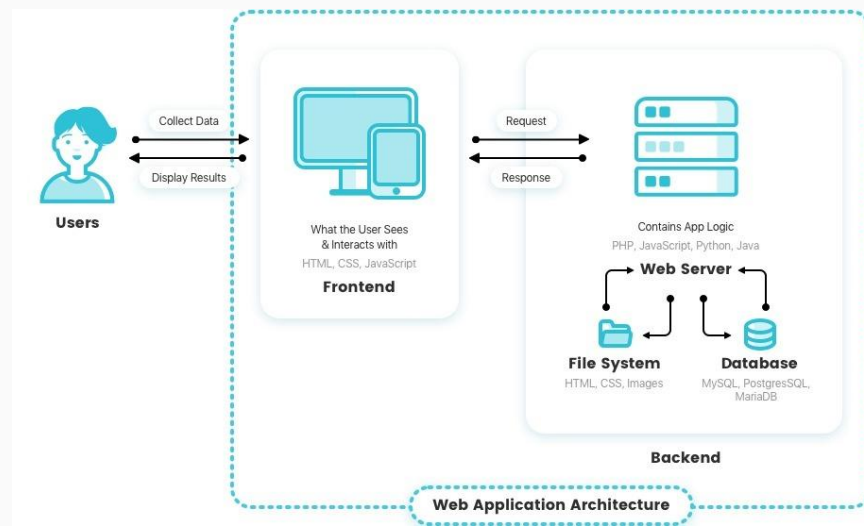
Arhitectură client-server

Web Apps Intro



Arhitectură client-server

- Arhitectura unei aplicații web reprezintă mecanismul prin care componentele unei aplicații comunică între ele.
- Cu alte cuvinte, reprezintă modul prin care userul și serverul comunică.
- O arhitectură web este împărțită în două componente:
 - clientul (frontend) - reprezintă calculatorul, telefonul, tableta sau orice alt device pe care îl folosește clientul pentru a naviga într-o aplicație web.
 - serverul (backend) - are rolul de a servi datele cerute de către client.
- Comunicarea dintre ele reprezintă modelul client-server, al cărui scop este de a rezolva un request al unui utilizator.



Arhitectură client-server

- Un server conține mai multe straturi (layere) care comunică între ele:
 - presentation layer
 - business/application layer
 - persistent storage layer
- **Presentation layer.** Este accesibil utilizatorului prin intermediul browser-ului și este format din componente UI care suportă interacțiunea cu sistemul. Este dezvoltat folosind următoarele tehnologii:
 - **HTML.** Determină ce va conține pagina web.
 - **CSS.** Determină cum va arăta conținutul paginii.
 - **JavaScript.** Determină interactivitatea cu utilizatorul a aplicației web.
- **Business layer.** Acest layer preia request-ul utilizatorului, îl procesează și determină pașii care trebuie făcuți pentru a îl îndeplini.
- **Persistence layer.** Denumit și layer de stocare reprezintă o locație centralizată care primește apelurile de date și oferă acces la structura folosită pentru stocarea datelor.



HTML & CSS

Web Apps Intro



HTML & CSS

- **HTML** este acronimul de la **H**yper **T**ext **M**arkup **L**anguage.
- Este un limbaj standardizat pentru etichetarea fișierelor text în vederea asigurării font-ului, culorilor, graficii și link-urilor într-o pagină/aplicație web.
- Este un fișier cu extensia **.html** care poate fi vizualizat cu ajutorul unui browser.
- Are o structură bazată pe tag-uri pentru definirea elementelor din document.
- Fiecare tag are rolul lui.
- Fiecare tag poate primi un set de atribute.
- Limbajul este în permanență revizuit de către comunitatea W3C (**W**orld **W**ide **W**eb **C**onsortium)

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```



HTML & CSS

- CSS este acronimul de la **C**ascading **S**tyle **S**heet.
- Este un limbaj standardizat care descrie cum trebuie afișate elemente HTML.
- Pentru stilizarea unui element HTML, acesta trebuie selectat folosind selector CSS.
- Un fișier ce conține cod CSS este salvat cu extensia **.css**.
- Stilul aferent fiecărui selector este scris sub forma **proprietate: valoare;** și este scris între acolade.
- CSS-ul poate fi folosit sub 3 forme:
 - extern - codul CSS se află într-un fișier extern.
 - Intern - codul CSS se află în documentul HTML (tag-ul **<style />**).
 - Inline - codul CSS se află în atributul **style=""**.



Django Framework

Web Apps Intro



Django Framework

- Django este un framework web bazat pe Python care are rolul de a încuraja dezvoltarea aplicațiilor web într-un mod rapid având în același timp un design pragmatic.
- Fiind scris de developeri cu experiență, acesta se ocupă de substraturile dezvoltării web astfel încât oferă luxul de a ne concentra pe dezvoltarea aplicației web în sine în loc de a ne concentra pe lucrurile repetitive de la un proiect la altul.



A fost conceput astfel încât să ajute developerii să își pună ideile în aplicare cât mai repede.



Respectă standardele de securitate și ajută developerii în a evita cât mai multe greșeli de securitate.



Aplicațiile web cu mult trafic se pot baza pe abilitatea framework-ului de a fi scalabil într-un mod rapid și flexibil.

Django Framework

- Fiind un framework de Python, pentru a instala Django trebuie să ne asigurăm ca avem instalat Python (versiunea 3+).
- Python instalează automat și **pip** care este managerul de referință pentru pachetele Python. Dacă acesta lipsește trebuie instalat manual.
 - Windows: `py -m pip install --upgrade pip`
 - Linux & MacOS: `python3 -m pip install --user --upgrade pip`
- Pentru a gestiona mai ușor pachetele instalate pentru diferite proiecte avem nevoie de **virtualenv/venv**.
 - Windows: `py -m pip install --user virtualenv`
 - Linux & MacOS: `python3 -m pip install --user virtualenv`
- Trebuie creat un nou virtual environment pentru noul proiect:
 - Windows: `py -m venv path_to_new_env`
 - Linux & MacOS: `python3 -m venv path_to_new_env`
- Activarea unui anumit virtual environment se face cu:
 - Windows: `.\path_to_env\Scripts\activate`
 - Linux & MacOS: `source path_to_env/bin/activate`
- Dezactivarea se face folosind `deactivate`



Django Framework

- Pentru a instala Django vom folosi:
 - Windows: `py -m pip install Django`
 - Linux & MacOS: `python3 -m pip install Django`
- Pentru a crea un nou proiect trebuie rulat:
 - Windows: `django-admin startproject nume_proiect`
 - Linux & MacOS: `django-admin startproject nume_proiect`
- Pentru a preciza calea unde să creăm proiectul ne putem folosi de un al doilea parametru al comenzii:
 - Windows: `django-admin startproject nume_proiect pathpath`
 - Linux & MacOS: `django-admin startproject nume_proiect path`
 - Pentru a crea o aplicație nouă trebuie să mergem în proiectul pe care îl avem și să rulăm:
 - Windows: `py manage.py startapp app_name`
 - Linux & MacOS: `python3 manage.py startapp app_name`



Django Framework

- În urma creării unui nou proiect vom observa următoarea structură de fișiere:
 - **settings.py** - fișierul cu setările proiectului. Aici se definesc aplicațiile folosite în proiect, conexiunile cu baza de date, 3rd party-uri și altele.
 - **urls.py** - fișierul unde sunt definite URL-urile (rutele) valabile în proiect/aplicație.
 - **manage.py** - core-ul prin care putem interacționa cu framework-ul Django.
 - **admin.py** - fișierul cu modelele disponibile în interfața administratorului.
 - **apps.py** - fișierul de configurare a aplicației.
 - **models.py** - fișierul unde sunt definite modele folosite în aplicație.
 - **tests.py** - fișierul unde sunt definite testele pentru aplicație.
 - **views.py** - fișierul unde sunt definite paginile aplicației.



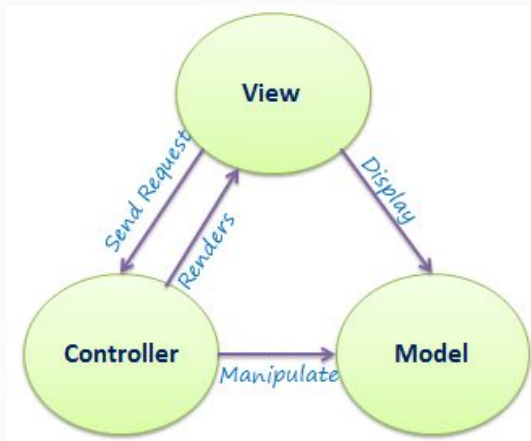
Arhitectură MVC/MTV

Web Apps Intro



Arhitectură MVC/MTV

- **MVC** este acronimul pentru **Model-View-Controller**.
- **MTV** este acronimul pentru **Model-Template-View**.
 - Mai poate fi găsit și sub denumirea de **MVT (Model-View-Template)** - este același concept.
- **MTV** este o altă denumire pentru **MVC** asociată framework-ului Django.
- Este un stil de arhitectură care separă o aplicație în trei componente logice: Model, View și Controller.
- Fiecare componentă se ocupă de un anumit aspect al aplicației.
- Este o arhitectură arhiprezentă în cele mai noi framework-uri și aplicații (în special cele web).



Arhitectură MVC/MTV

- Oferă un mod mai ușor de a testa aplicația.
- Oferă control total asupra HTML-ului și URL-urilor.
- Este în conformitate cu funcționalitățile oferite de ASP.NET, JSP, Django ș.a.
- Oferă separare între logica aplicației și modul de gestionare și prezentare a datelor.
- Suportă TDD (Test Driven Development).

Model - V - C

- Este componenta care se ocupă de reprezentarea datelor și logica aferentă lor.
- Reprezintă datele manipulate de aplicație și prezentate cu ajutorul componentei View.

M - View - C

- Este componenta care prezintă utilizatorului date.
- Reprezintă interfața cu care interacționează utilizatorul.

M - V - Controller

- Este componenta care manipulează request-ul făcut de utilizator.
- Este responsabil de toată logica aplicației.



Django ORM

Web Apps Intro

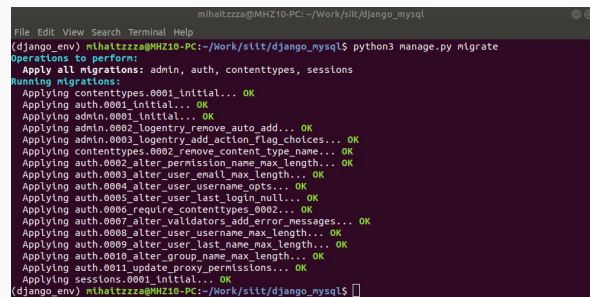


Django ORM

- Pentru a seta MySQL ca bază de date a aplicației noastre avem nevoie să instalăm **mysqlclient**. Acesta este un client MySQL care are rolul să comunice cu serverul nostru.
- Pentru a instala **mysqlclient** rulați: **pip install mysqlclient**.
- Pentru a defini informațiile pentru conectarea la baza de date modificați fișierul **settings.py**, secțiunea **DATABASES**, astfel:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'HOST': 'localhost',  
        'NAME': 'django_db',  
        'USER': 'root',  
        'PASSWORD': 'root',  
    }  
}
```

- Pentru a testa conexiunea cu baza de date, rulați comanda: **python manage.py migrate**
- Aceasta va rula toate migrările neaplicate încă. By default, Django conține migrări pentru interfața de administrator.



```
mihaitezza@MHZ10-PC: ~/Work/slt/django_mysql  
(django_env) mihaitezza@MHZ10-PC:~/Work/slt/django_mysql$ python3 manage.py migrate  
Operations to perform:  
Apply all migrations: admin, auth, 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 sessions.0001_initial... OK  
(django_env) mihaitezza@MHZ10-PC:~/Work/slt/django_mysql$
```



Django ORM

- ORM este acronimul de la Object Relational Mapping (mapare obiecte-relație). Este o tehnică de programare pentru convertirea tipurilor de date din date incompatibile ale unui sistem în obiecte ale unui limbaj de programare.
- Cu ajutorul acestei tehnici putem să avem o imagine a bazei de date în contextul obiectelor din limbajul de programare folosit.
- În Django, noțiunea de bază în ceea ce privește gestiunea datelor și interacțiunea cu baza de date o reprezintă **Model**-ul.
- Conform documentației: ***“Modelul este singura și incontestabila sursă de adevăr a datelor noastre. Conține câmpurile și comportamentul esențial al datelor pe care le stocăm.”***
- Rolul ORM-ului în Django este să modeleze baza de date pe baza obiectelor Python definite de noi.
- Ca o clasă Python să reprezinte un model al bazei de date trebuie să:
 - fie o clasă care să moștenească clasa **django.db.models.Model**.
 - fiecare atribut al clasei trebuie să reprezinte un câmp din baza de date.



Django ORM

- Crearea primului model. Modelele își au locul în folderul **models.py**.
- Structura unui model este:

```
class Employer(models.Model):  
    name = models.CharField(max_length=255, unique=True)  
  
    def __str__(self):  
        return '%s %s' % (type(self), self.id)
```

- În Django, câmpul ID este automat folosit de framework pentru identificare datelor (primary key).
- După definirea modelelor rulați comanda **python manage.py makemigrations** pentru a crea fișierele de migrare aferente modelelor respective.
- Pentru rula fișierele de migrare trebuie folosită comanda **python manage.py migrate**.



Django ORM

- Pentru a defini un model de bază care să conțină câmpuri pe care le dorim în toate modelele ne putem folosi de clasa Meta astfel:

```
class MyModel(models.Model):  
    class Meta:  
        abstract = True  
  
    created_at = models.DateTimeField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)
```

- Observați setarea atributului **abstract**. Acesta indică faptul că acest model este unul abstract, deci nu va fi mapat în baza de date.
- Pentru moștenirea câmpurilor de **created_at** și **updated_at** va trebui ca modelul nostru să moștenească această clasă.

```
class Employer(MyModel):  
    class Meta:  
        db_table = 'employers'  
  
    name = models.CharField(max_length=255, unique=True, null=False)
```



Django fixtures

Web Apps Intro



Django fixtures

- Django fixtures reprezintă o modalitate de a popula baza de date.
- Pentru a seta niște date inițiale vom crea următorul fișier JSON:

```
[{
  "model": "employee.employer",
  "fields": {
    "name": "Google",
    "created_at": "2020-12-22 11:00:00",
    "updated_at": "2020-12-22 11:00:00"
  }
}, {
  "model": "employee.employer",
  "fields": {
    "name": "IBM",
    "created_at": "2020-12-22 11:00:00",
    "updated_at": "2020-12-22 11:00:00"
  }
}, {
  "model": "employee.employer",
  "fields": {
    "name": "Facebook",
    "created_at": "2020-12-22 11:00:00",
    "updated_at": "2020-12-22 11:00:00"
  }
}]
```

- Acest fișier trebuie să fie în folderul fixtures.
- Pentru importarea lui se va folosi comanda: **python manage.py loaddata <fixturename>**.



Django views & templates

Web Apps Intro



Django views & templates

- Pentru a defini conținutul unei pagini avem nevoie să definim:
 - o rută - aceasta se definește în **urls.py**.
 - un view - aceasta este o funcție ce primește un request și returnează un răspuns. Se definesc în **views.py**. Poate întoarce fie un răspuns HTTP, fie poate randa un template.
 - un template - acesta este un fișier html care va fi randat de view-ul definit anterior.
- Orice aplicație poate conține un folder **templates** unde putem ține template-urile aferente acesteia.
- Pentru definirea unor template-uri generale avem nevoie să configurăm un folder **templates** la nivel de proiect. Pentru a face acest lucru vom modifica fișierul **settings.py** astfel:
 - importăm modulul os (**import os**) la începutul fișierul **settings.py**.
 - updatăm setarea **TEMPLATES["DIRS"]** astfel:

```
'DIRS': [os.path.join(BASE_DIR, 'templates')],
```



Django views & templates

- Pentru a crea o rută vom adăuga următorul cod în fișierul `urls.py` al proiectului:

```
from django.contrib import admin
from django.urls import path
from djdjango.views import homepage

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', homepage),
]
```

- Un fișier `urls.py` trebuie să conțină o listă denumită **urlpatterns**. Django știe să colecteze toate rutele dintr-un proiect pe baza acestui nume.
- Prima rută din lista este pentru interfața de administrare și poate fi accesată folosind **/admin**.
- Cea de-a doua rută este adăugată de noi și va juca rolul de homepage (prima pagină).
- Pentru a continua va trebui să definim view-ul **homepage**.



Django views & templates

- Pentru a crea view-ul homepage avem nevoie să definim o funcție în fișierul **views.py**. Aceasta va avea următorul cod:

```
from django.shortcuts import render

def homepage(request):
    print(type(render))
    return render(request, 'homepage.html', {
        "name": "DJANGO"
    })
```

- va primi ca parametru obiectul **request**. Acesta conține toate informațiile despre request-ul făcut către această rută.
- va întoarce un răspuns HTTP primit de la funcția render care este apelată cu următorii parametri:
 - ➡ request-ul făcut către rută
 - ➡ calea către template-ul ce trebuie randat
 - ➡ context-ul necesar template-ului (un dicționar ce conține datele necesare template-ului).



Django views & templates

- Ultimul pas în implementarea rutei o reprezintă definirea template-ului.
- În ruta anterioară am folosit template-ul homepage.html, astfel că acesta trebuie creat în folderul **project/templates**.
- Acest fișier este un de tip HTML. Până la definirea unui homepage mai complex vom folosi următorul template:

```
<h3>Project developed in {{name}}</h3>
<p>This is my first template.</p>
```

- Pentru a folosi date trimise din view ne folosim de notația **{{ }}**.
- Template-urile Django pot extinde alte template-uri. Pentru a realiza acest lucru vom folosi tag-ul **extends** astfel:

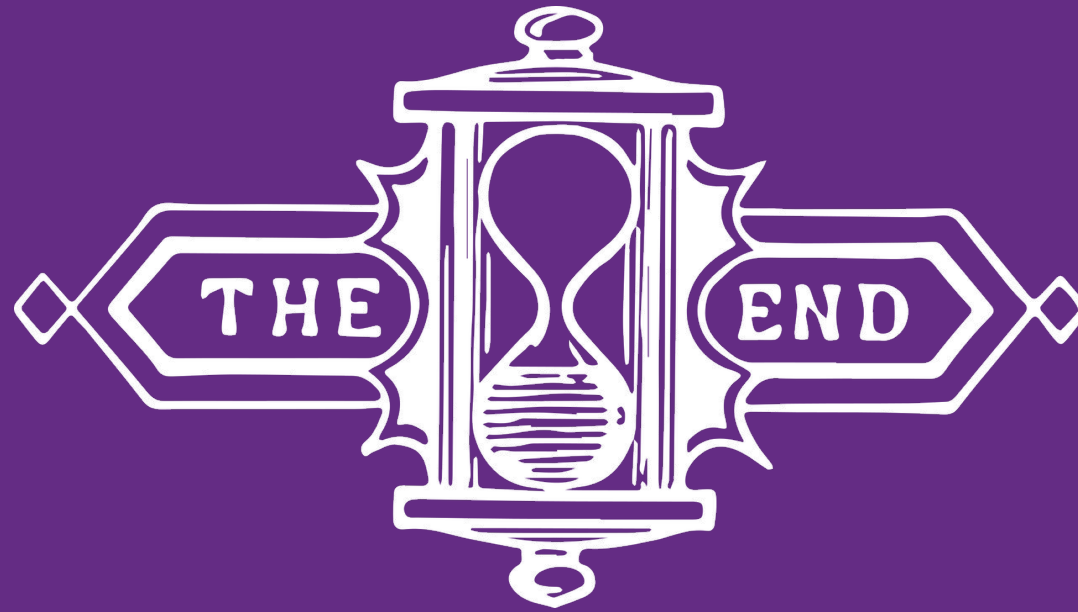
```
<html>
  <head>
    <title>Base template</title>
  </head>
  <body>
    <h1>This is the base template header.</h1>
    {% block content %}
    {% endblock %}
  </body>
</html>
```

- Acest template poate fi extins astfel:

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

{% block content %}
  <h2>Project developed in {{name}}</h2>
  <p>This is my first template.</p>
{% endblock %}
```





Vă mulțumesc!

