PROYECTO DE SOFTWARE

Cursada 2020

TEMARIO

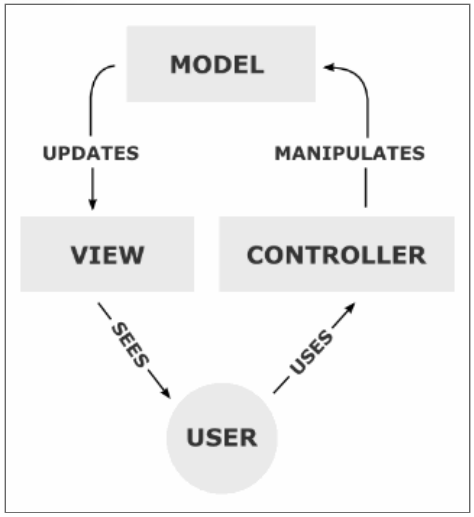
- MVC.
- Templates
- Api Rest.

PATRÓN MVC

MODEL – VIEW - CONTROLLER

- Tres componentes:
    - Modelo
    - Vista
    - Controlador
- El principio más importante de la arquitectura MVC es la **separación del código del programa en tres capas**, dependiendo de su naturaleza.
- La lógica relacionada con los datos se incluye en el modelo, el código de la presentación en la vista y la lógica de la aplicación en el controlador.

MVC



- Reduce la complejidad, facilita la reutilización y acelera el proceso de comunicación entre capas.

APLICACIÓN TÍPICA SIN MVC

- Aplicación típica que no sigue MVC tiene todo el código en el mismo lugar. Ver ejemplo **Cliente Servidor sin MVC**

```
#encoding: utf-8
import pymysql

from flask import Flask, g



app = Flask(__name__)

@app.route('/')
def hello_world():
    #CREATE USER 'proyecto'@'localhost' IDENTIFIED BY 'password1';
    #GRANT ALL PRIVILEGES ON *.* TO 'proyecto'@'localhost';
    SECRET_KEY = "dev"
    DEBUG = True
    DB_HOST = 'localhost'
    DB_USER = 'proyecto'
    DB_PASS = 'password1'
    DB_NAME = 'proyecto'
    #Creamos la conexión
    g.db = pymysql.connect(
            host=DB_HOST,
            user=DB_USER,
            password=DB_PASS,
            db=DB_NAME,
            cursorclass=pymysql.cursors.DictCursor
            )

    mensaje = ''
    mensaje +=  ' '
    mensaje +=  ''
    mensaje += ''
    mensaje += ''

    sql = "SELECT * FROM issues"

    cursor = g.db.cursor()
    cursor.execute(sql)
    issues = cursor.fetchall()
    mensaje += ''

    mensaje += ''
    for field in issues[0].keys():
        mensaje += ''
    mensaje += ''

    for issue in issues:
        mensaje += ''
        for field in issue.values():
            mensaje += ''
        mensaje += ''
    mensaje += '<table><tbody><tr><th>'
        mensaje += field
        mensaje += '</th></tr><tr><td>'
            mensaje += str(field)
            mensaje += '</td></tr></tbody></table>'
    mensaje += ''
    mensaje += ''

    #Cerramos conexión a la BBDD
    db = g.pop('db', None)
    if db is not None:
        db.close()
```

```
    return mensaje
```

MVC - CONFIGURACIÓN

app_template/config.py

```
from os import environ

class BaseConfig(object):
    """Base configuration."""

    DEBUG = None
    DB_HOST = "bd_name"
    DB_USER = "db_user"
    DB_PASS = "db_pass"
    DB_NAME = "db_name"
    SECRET_KEY = "secret"

    @staticmethod
    def configure(app):
        # Implement this method to do further configuration on
your app.
        pass


class DevelopmentConfig(BaseConfig):
    """Development configuration."""

    ENV = "development"
    DEBUG = environ.get("DEBUG", True)
    DB_HOST = environ.get("DB_HOST", "localhost")
    DB_USER = environ.get("DB_USER", "MY_DB_USER")
    DB_PASS = environ.get("DB_PASS", "MY_DB_PASS")
    DB_NAME = environ.get("DB_NAME", "MY_DB_NAME")
```

MVC – SEPARANDO EL MODELO

app_template/app/db.py

```
import pymysql

from flask import current_app
from flask import g
from flask import cli


def connection():
    if "db_conn" not in g:
        conf = current_app.config
        g.db_conn = pymysql.connect(
            host=conf["DB_HOST"],
            user=conf["DB_USER"],
            password=conf["DB_PASS"],
            db=conf["DB_NAME"],
            cursorclass=pymysql.cursors.DictCursor,
        )

    return g.db_conn


def close(e=None):
    conn = g.pop("db_conn", None)

    if conn is not None:
        conn.close()


def init_app(app):
    app.teardown_appcontext(close)
```

MVC – SEPARANDO EL MODELO

app_template/app/models/issue.py

```
class Issue(object):
    @classmethod
    def all(cls, conn):
        sql = "SELECT * FROM issues"

        cursor = conn.cursor()
        cursor.execute(sql)

        return cursor.fetchall()

    @classmethod
    def create(cls, conn, data):
        sql = """
            INSERT INTO issues (email, description, category_id,
status_id)
            VALUES (%s, %s, %s, %s)
        """

        cursor = conn.cursor()
        cursor.execute(sql, list(data.values()))
        conn.commit()

        return True
```

MVC - EL CONTROLADOR

app_template/app/resources/issue.py

```
from flask import redirect, render_template, request, url_for
from app.db import connection
from app.models.issue import Issue

# Public resources
def index():
    conn = connection()
    issues = Issue.all(conn)

    return render_template("issue/index.html", issues=issues)


def new():
    return render_template("issue/new.html")


def create():
    conn = connection()
    Issue.create(conn, request.form)

    return redirect(url_for("issue_index"))
```

MVC – SEPARANDO LA VISTA

app_template/app/templates/layout.html

```
    {% block head %}
    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
    <title>{% block title %}{% endblock %}</title>
    {% endblock %}


    <div id="navbar">
      {% block navbar %}
      {% endblock %}
    </div>
    <div id="content">
      {% with messages = get_flashed_messages() %}
        {% if messages %}
          <ul class="flashes">
          {% for message in messages %}
            <li>{{ message }}</li>
          {% endfor %}
          </ul>
        {% endif %}
      {% endwith %}

      {% block content %}
      {% endblock %}
    </div>
    <div id="footer">
      {% block footer %}
      {% endblock %}
    </div>
```

MVC – SEPARANDO LA VISTA - VISTA

app_template/app/templates/issue/index.html

```
{% extends "layout.html" %}
{% block title %}Consultas{% endblock %}
{% block head %}
  {{ super() }}
{% endblock %}
{% block content %}
  <h1>Consultas</h1>
  {% for issue in issues %}
    <li>{{ issue.email }} - {{ issue.description }} - {{
issue.category_id }} - {{ issue.status_id }}</li>
  {% endfor %}
  <a href="{{ url_for('home') }}" class="link">Volver</a>
  <a href="{{ url_for('issue_new') }}" class="link">Nuevo</a>
{% endblock %}
```

MVC - SEPARANDO LA VISTA - TEMPLATES

- ¿Qué es?
- Lo vemos más adelante.

VARIACIONES DEL MVC ORIGINAL

- MTV en Django - Particularidades
- Model
- Template
- View

BUENAS PRÁCTICAS MVC

La idea central detrás de MVC es la **reutilización de código** y la **separación de intereses**.

BUENAS PRÁCTICAS MVC - MODELO

- Puede contener:
    - La lógica necesaria para asegurar que los datos cumplen los requerimientos (validaciones).
    - Código de manipulación de datos.
- NO puede contener:
    - En general, nada que se relacione con el usuario final directamente.
    - Se debe evitar HTML embebido o cualquier código de presentación.

BUENAS PRÁCTICAS MVC - VISTA

- Puede contener:
    - Código de presentación, formatear y dibujar los datos.
- NO puede contener:
    - Código que realice consultas a la BD.

BUENAS PRÁCTICAS MVC - CONTROLADOR

- Puede contener:
    - Creación de instancias del Modelo para pasarle los datos necesarios.
- NO puede contener:
    - Código que realice consultas a la BD (SQL embebido).
    - Se debe evitar HTML embebido o cualquier código de presentación.

PARA SEGUIR VIENDO:

- Lenguaje SQL -> **http://www.w3schools.com/sql/default.asp**
- MVC -> **http://es.wikipedia.org/wiki/Modelo_Vista_Controlador#Frameworks_MV**

LA VISTA

PROGRAMANDO CON TEMPLATES – ALTERNATIVAS PARA EL VIEW

TEMPLATES EN FLASK

- El uso de templates o plantillas permite separar la aplicación de la presentación, pero ....

No asegura MVC. Ésa es NUESTRA responsabilidad

- En flask, Jinja: **https://flask.palletsprojects.com/en/1.1.x/templating/**

TEMPLATES EN FLASK - JINJA2

- Jinja2 es un **motor de templates** en Python promocionado como un motor de plantilla **rápido**, **moderno** y **seguro**.

https://palletsprojects.com/p/jinja/

- Desarrollado y distribuido bajo **licencia BSD**.
- ¿Por qué lo elegimos en la cátedra?
    - Porque es la alternativa que propone Flask.
    - **Es muy similar a otros motores con lo cual el traspaso a otro motor es inmediato.**

TEMPLATES EN FLASK - JINJA2

- Flask utiliza por defecto las plantillas de Jinja 2 mediante el módulo flask.render_template().
- Configurado para buscar las plantillas en el directorio templates.

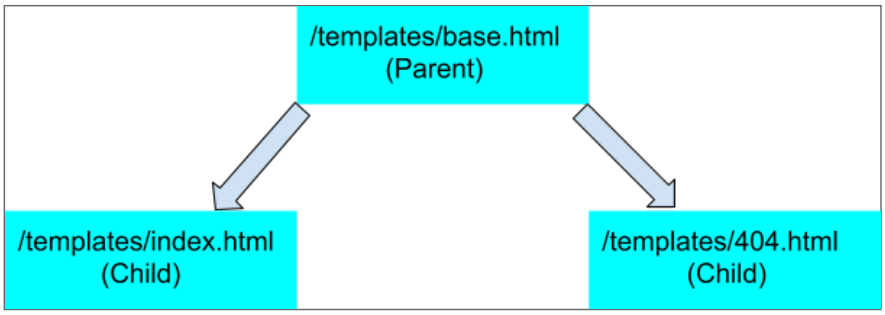
```
from flask import redirect, render_template, request, url_for
from flaskps.db import get_db
from flaskps.models.issue import Issue


def index():
    Issue.db = get_db()
    issues = Issue.all()

    return render_template('issue/index.html', issues=issues)
```

JINJA2 - HERENCIA

- Jinja2 admite herencia.
- Nos permite tener un diseño básico, común a todas las plantillas.
- La herencia se realiza a través de **{% extend %}**




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

JINJA2 - HERENCIA

A través de **{% block %}** indicamos al motor de plantilla que se debe reemplazar el contenido por los de la plantilla hija.

```
{% block title %}Home{% endblock %}
{% block head %}
  {{ super() }}
{% endblock %}
{% block content %}
  <h1>Home</h1>
  <p>Welcome to my awesome page.</p>
{% endblock %}
```

JINJA2 - ESTRUCTURAS DE CONTROL

- Las estructuras de control se señalan con el delimitador **{% ... %}**

```
<ul>
{% for user in users %}
    <li>{{ user.username|e }}</li>
{% else %}
    <li><em>no users found</em></li>
{% endfor %}
</ul>
```

- Ver:
    - Fuente: **Jinja2 Estructuras de Control**

JINJA2 - VARIABLES

- Podemos acceder a atributos de las variables pasados al template de dos maneras:

```
{{ foo.bar }}
{{ foo['bar'] }}
```

JINJA2 - FILTROS

- Las variables pueden ser modificadas utilizando filtros.

```
{{ name|striptags|title }}
```

- Fuente: **jinja2_filtros**
- Algunos filtros: **capitalize()**, **format()**, **replace()**, **urlencode()**, **json_encode**, **upper()**, **lower()**, **join()**, **sort()**, **truncate()**, **trim()**, etc.

ALGUNAS COSAS MÁS

- Nos permite incluir comentarios a través de **{#.....#}**
- Se pueden incluir otros templates

```
{% include 'header.html' %}
    Body
{% include 'footer.html' %}
```

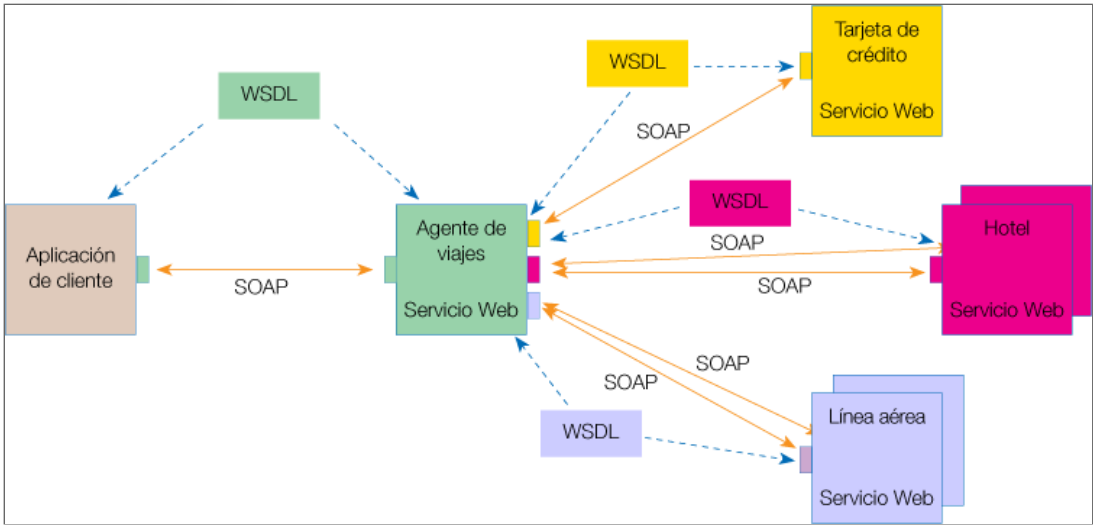
¿SEGUIMOS?

- ¿Qué es una API REST?

SERVICIOS WEB

- Un **servicio web** es una tecnología que utiliza un conjunto de **protocolos** y **estándares** para intercambiar datos entre aplicaciones.
- Algo importante: lograr **interoperabilidad**.
- Uso de estándares abiertos
    - XML-RPC, JSON-RPC
    - WSDL: Web Services Description Language
    - SOAP: Simple Object Access Protocol
    - ¿Rest?

SERVICIOS WEB



- Ejemplo típico con WSDL y SOAP

¿QUÉ ES REST?

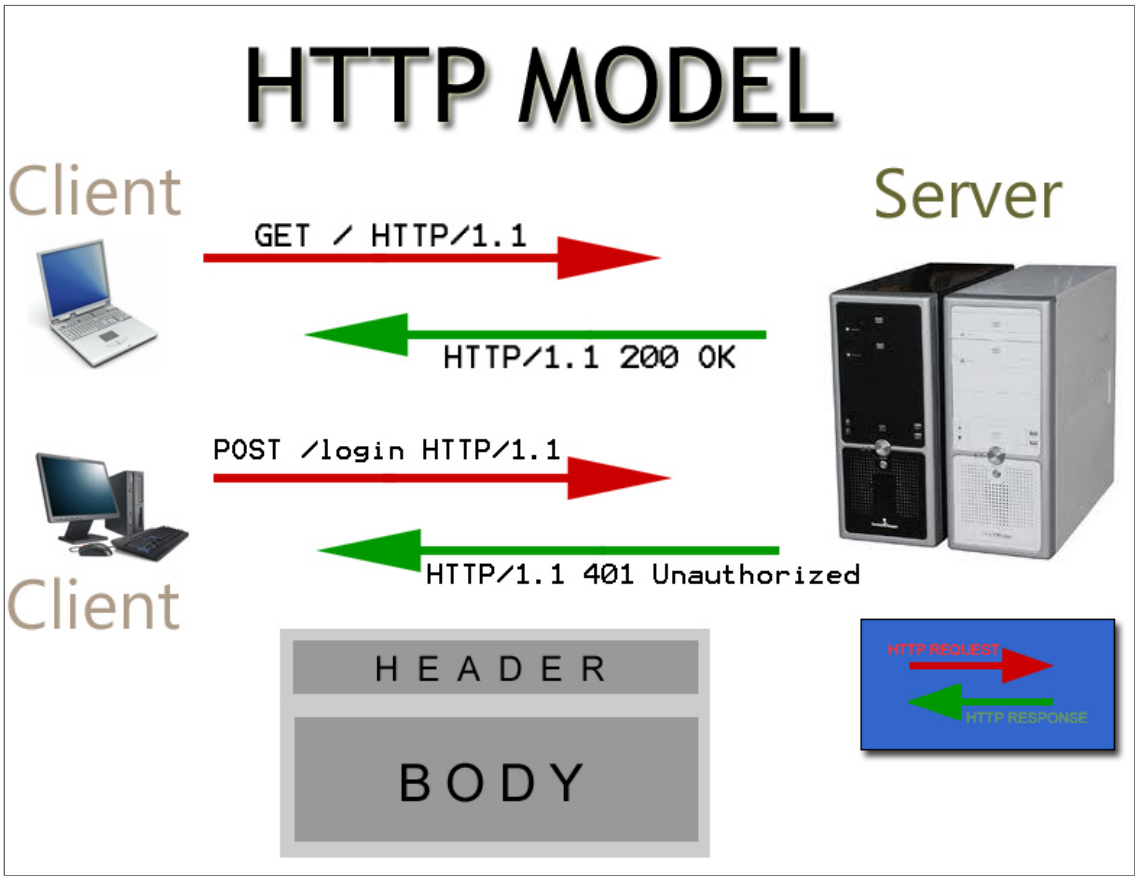
- **REpresentational State Transfer**: arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.
- REST nos permite crear servicios y aplicaciones que pueden ser usadas por **cualquier dispositivo o cliente que entienda HTTP**, por lo que es más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.
- REST fue definido por Roy Fielding (coautor de la especificación HTTP) en el año 2000.

REST

- Los sistemas que siguen los principios REST se los denomina también RESTful.
- Se basa en **HTTP** para intercambiar información.
- **SIN estado**.
- Se piensa en los **recursos** como una entidad que puede ser accedido públicamente.
- Cada objeto tiene su propia URL y puede ser fácilmente cacheado, copiado y guardado como marcador.

RECORDEMOS QUE ....

- Una petición HTTP consta de:
    - Una **URL** y un **método** de acceso (GET, POST, PUT,...).
    - **Cabeceras**. Meta-información de la petición.
    - **Cuerpo del mensaje** (opcional).


HTTP MODEL
Client
Server
GET / HTTP/1.1
HTTP/1.1 200 OK
POST /login HTTP/1.1
HTTP/1.1 401 Unauthorized
Client
HEADER
BODY
HTTP REQUEST
HTTP RESPONSE

MÉTODOS HTTP

- **GET**: usado para solicitar un recurso al servidor.
- **PUT**: usado para modificar un recurso existente en el servidor.
- **POST**: usado para crear un nuevo recurso en el servidor.
- **DELETE**: usado para eliminar un recurso en el servidor.

¿POST = GET + PUT + DELETE?

- Por lo general, las peticiones de tipo **PUT** y **DELETE** son realizadas a través de peticiones **POST**.
- La petición **POST** se utiliza tanto para crear, borrar o actualizar un recurso.
- Pero hay una diferencia: **POST NO es idempotente**.

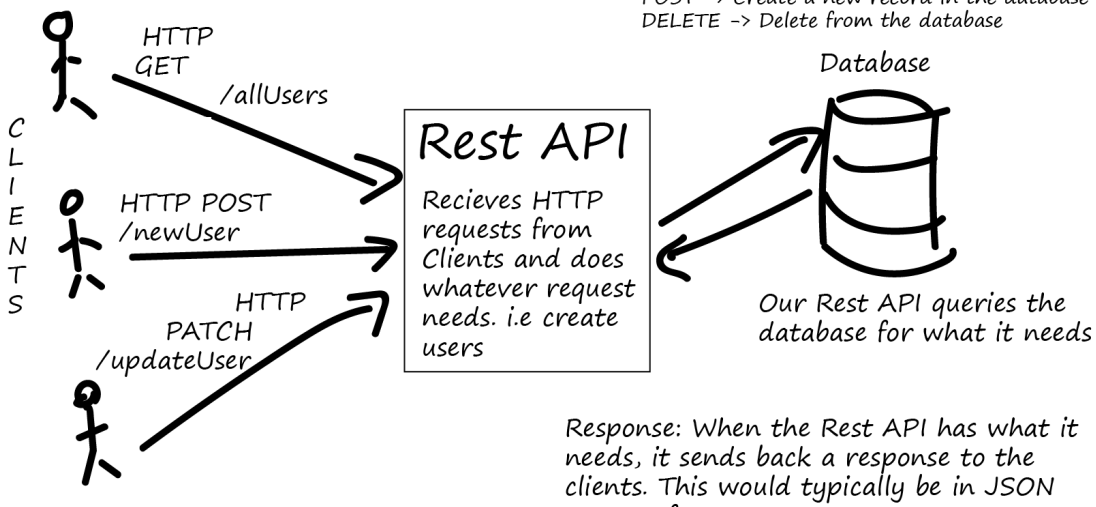
MÉTODOS HTTP

HTTP Method	Idempotent	Safe
OPTIONS	yes	yes
GET	yes	yes
HEAD	yes	yes
PUT	yes	no
POST	no	no
DELETE	yes	no
PATCH	no	no

- Los métodos HTTP **Seguros** son aquellos que no modifican los recursos.
- Los métodos HTTP **Idempotentes** son aquellos que pueden ser llamados múltiples veces y generarán el mismo resultado.
- Más info ver: **rfc7231** y **rfc5789**

Rest API Basics

Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database



Our Rest API queries the database for what it needs

Our Clients, send HTTP Requests and wait for responses

Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.

ACCEDIENDO A LOS RECURSOS

- La implementación del recurso decide qué información es visible o no desde el exterior, y qué representaciones de dicho recurso se soportan.
- Podríamos pensar en:
    - HTML
    - XML
    - JSON
- Ejemplo: ¿consultamos los **feriados de 2020**?

ACCEDIENDO A LOS RECURSOS

- Probemos con **curl**:

```
curl -X GET https://api.mercadolibre.com/categories/MLA5725
```

- Otros ejemplos:
    - API REST de Mercado Libre: **http://developers.mercadolibre.com/es/api-docs-es/**
    - Google Translate (es necesario API_KEY): **https://developers.google.com/apis-explorer/#p/translate/v2/**
    - El clima en OpenWeatherMap (es necesario API_KEY): **clima en La Plata**

VENTAJAS / DESVENTAJAS

- Separación cliente/servidor.
- Simplicidad.
- Seguridad.
- Uso de estándares.
- Escalabilidad.
- Cambio de esquema: usando REST podemos tener varios servidores donde unos no saben que los otros existen.

+Info: **http://www.desarrolloweb.com/articulos/ventajas-inconvenientes-apirest-desarrollo.html**

GENERANDO API REST

- A mano.... o,
- Muchos frameworks que facilitan el desarrollo:
    - Django: **https://www.django-rest-framework.org/tutorial/quickstart/**
    - Flask: **https://flask-restful.readthedocs.io/en/latest/quickstart.html**

REFERENCIAS REST

- http://martinfowler.com/articles/richardsonMaturityModel.html
- http://restcookbook.com/Miscellaneous/richardsonmaturitymodel/
- http://www.restapitutorial.com/lessons/whatisrest.html
- http://asiermarques.com/2013/conceptos-sobre-apis-rest/
- http://rest.elkstein.org/
- http://restfulwebapis.org/rws.html
- https://restfulapi.net/
- https://www.paradigmadigital.com/dev/introduccion-django-rest-framework/
- https://flask-restful.readthedocs.io/en/latest/

FIN