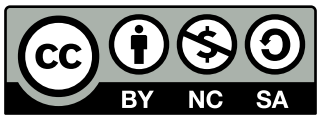
Desarrollo Web con Flask

Jesús Arias Fisteus

Aplicaciones Web (OpenCourseWare, 2023)

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática



Parte I

El patrón Modelo–Vista–Controlador en Flask

El patrón Modelo–Vista–Controlador

El patrón **Modelo–Vista–Controlador (MVC)** es un patrón de diseño de *software* para interfaces de usuario que divide la aplicación en tres componentes: **modelo**, **vista** y **controlador**.

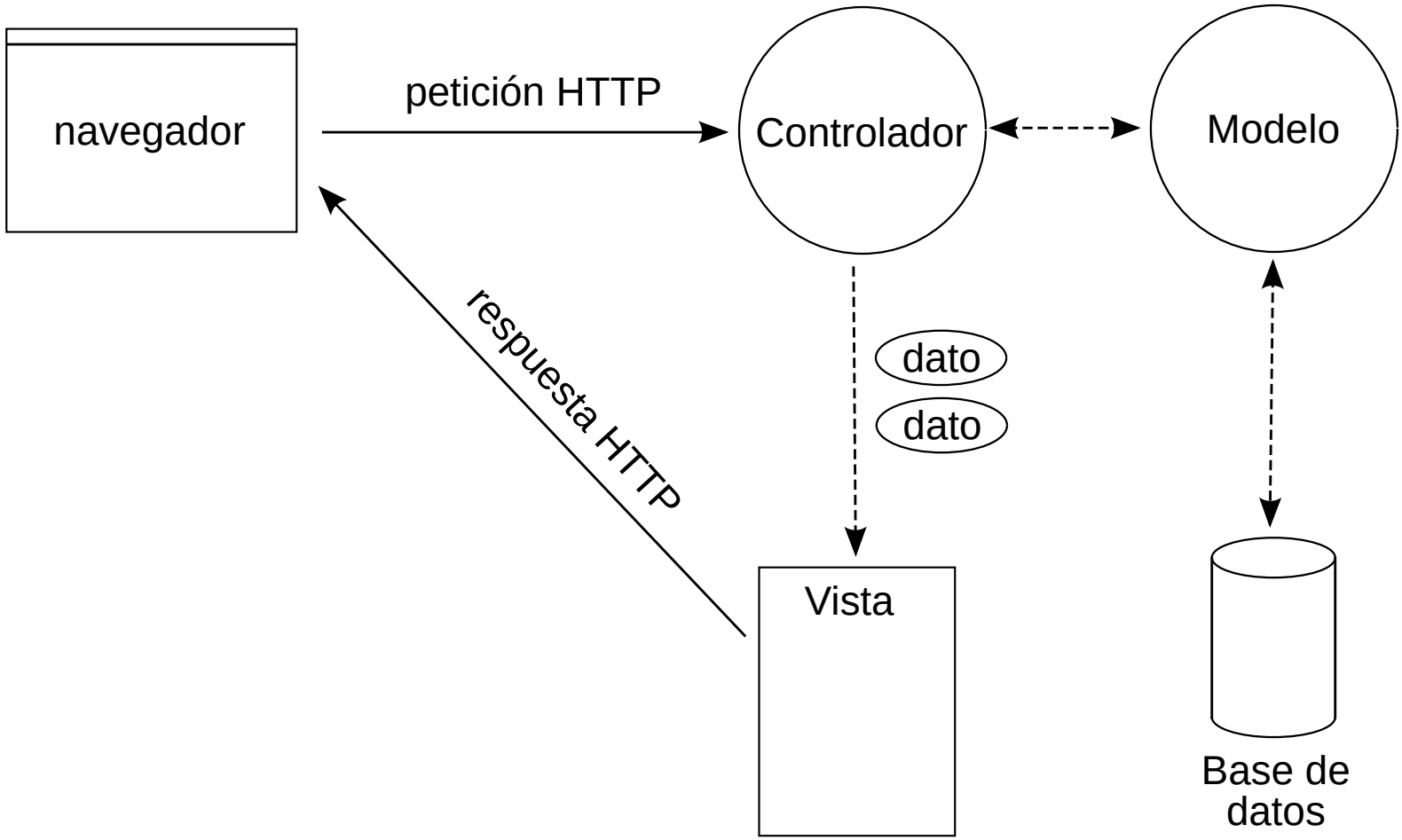
The Model–View–Controller pattern

El componente del **modelo** gestiona los datos, lógica y reglas de la aplicación, incluyendo normalmente, en el caso de las aplicaciones Web, el almacenamiento de datos en una base de datos.

El componente de la **vista** presenta los datos a los usuarios (normalmente con HTML y CSS en el caso de las aplicaciones Web) y recoge la entrada de los usuarios.

El componente del **controlador** acepta la entrada de los usuarios e invoca la funcionalidad apropiada en los componentes del modelo y de la vista.

El patrón MVC en aplicaciones Web



El patrón MVC en aplicaciones Web

1. Mediante una página HTML creada por la vista anterior, el cliente envía una petición HTTP al servidor.
2. El controlador recibe la petición e interacciona con el modelo.
3. El controlador invoca a la vista con los datos que necesita presentar al usuario.
4. La vista recoge dichos datos, crea la página HTML que los presenta y construye la respuesta HTTP a enviar al cliente.

Flask

Flask[1] es un *framework* minimalista para el desarrollo de aplicaciones Web en lenguaje Python, con soporte para extensiones que añaden muchas de las funciones típicamente presentes en otros *frameworks* Web.

[1] `https://flask.palletsprojects.com/`

El componente del modelo

El componente del **modelo** suele construirse en *frameworks* Web sobre sistemas de persistencia de objetos, conocidos normalmente como ORM[2], los cuales proporcionan automáticamente el código necesario para **crear**, **leer**, **actualizar** y **borrar** (CRUD) objetos en la base de datos.

[2] ORM: *mapeo objeto relacional*.

El componente del modelo

Sin embargo, también existen alternativas a los ORM ampliamente usadas, como son:

- Construir consultas SQL directamente desde el código de la aplicación.
- Usar bases de datos no basadas en el modelo relacional (ni en el lenguaje SQL) como, por ejemplo, MongoDB.

El componente del modelo en Flask

```
# Declaración de una clase del modelo
class User(db.Model):
    user_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), nullable=False)
    email = db.Column(
        db.String(128),
        unique=True,
        nullable=False
    )

# Inserción de un objeto
new_user = model.User(email="mary@example.com", name="Mary")
db.session.add(new_user)
db.session.commit()

# Consulta de un objeto
user = model.User.query.filter_by(
    email="mary@example.com"
).first()
```

El componente de la vista

Las **vistas** suelen ser construidas en *frameworks* Web mediante **plantillas** HTML que, a partir de un conjunto de variables que contienen los datos a mostrar, producen la página HTML que los presenta.

Las plantillas consisten fundamentalmente en marcado HTML con pequeños fragmentos de código incrustado para inyectar datos, mostrar contenido condicionalmente, iterar sobre los datos de una colección, etc.

Plantillas

Las aplicaciones Flask usan normalmente el lenguaje de plantillas **Jinja**[3].

[3] https://jinja.palletsprojects.com/

Jinja

```
<div class="messages">
    {% for post in posts %}
        <div class="message">
            <div class="text">{{ post.text }}</div>
            <div class="metadata">
                <span class="author">
                    {{ post.user.name }}
                </span>
                <span class="date">
                    {{ post.timestamp }}
                </span>
            </div>
        </div>
    {% endfor %}
</div>
```

El componente del controlador

Normalmente se programan los **controladores** mediante estructuras normales de código (métodos / funciones) en el lenguaje de programación del *framework*.

El componente del controlador

Cada método o función se mapea a una o más rutas de peticiones HTTP, recibe todos los datos que necesite de la petición HTTP, interacciona con el modelo y, finalmente, selecciona la plantilla a ser presentada.

Cuando el *framework* recibe una petición HTTP, decide, conforme a la ruta de la petición, qué método o función del controlador necesita ser invocado.

Controladores en Flask

```
@bp.route("/post/<int:message_id>")
@flask_login.login_required
def post(message_id):
    message = model.Message.query.filter_by(
        id=message_id
    ).first()
    if not message:
        abort(
            404,
            "Post id {} doesn't exist.".format(message_id)
        )
    if message.response_to is not None:
        abort(
            403,
            "No view for response messages is available"
        )
    responses = (
        model.Message.query.filter_by(
            response_to=message
        ).order_by(
            model.Message.timestamp.desc()
        ).all()
    )
    return render_template(
        "main/post.html",
        post=message,
        responses=responses
    )
```

Controladores en Flask

```
@bp.route("/new_post", methods=["POST"])
@flask_login.login_required
def new_post():
    text = request.form.get("text")
    message = model.Message(
        user=flask_login.current_user,
        text=text,
        timestamp=datetime.datetime.now(
            dateutil.tz.tzlocal()
        ),
    )
    db.session.add(message)
    db.session.commit()
    return redirect(
        url_for("main.post", message_id=message.id)
    )
```

Parte II

Leyendo datos desde formularios

Controles en formularios HTML

Cuando el usuario presiona el botón de envío de un formulario HTML, el navegador lee los valores de todos los controles del mismo, y los encapsula en una petición HTTP como una **secuencia de parámetros nombre–valor**. La URL de la petición se toma del atributo **`action`** del elemento **`form`**.

El **nombre** de cada parámetro se toma del atributo **`name`** del control correspondiente.

El **valor** de cada parámetro se toma de la entrada del usuario en dicho control.

Controles en formularios HTML

```
<form action="publish" method="get">
  <label>
    Search user:
    <input type="email" name="email"
        placeholder="email␣address">
  </label>
  <input type="submit" value="Search">
</form>
```

Lectura de parámetros de la petición en Flask

Flask proporciona a las funciones del controlador un objeto **`request`** desde el cual estas pueden leer los parámetros de la petición.

```
@bp.route("/find-user")
@flask_login.login_required
def find_user():
    email = request.form.get("email")
    user = model.User.query.filter_by(email=email).first()
    return render_template("view_user", user=user);
```

Referencias

- Miguel Grinberg. *Flask Web Development, 2nd ed.*, O'Reilly Media Inc. (2018):

Parte III

Rutas relativas

Rutas relativas

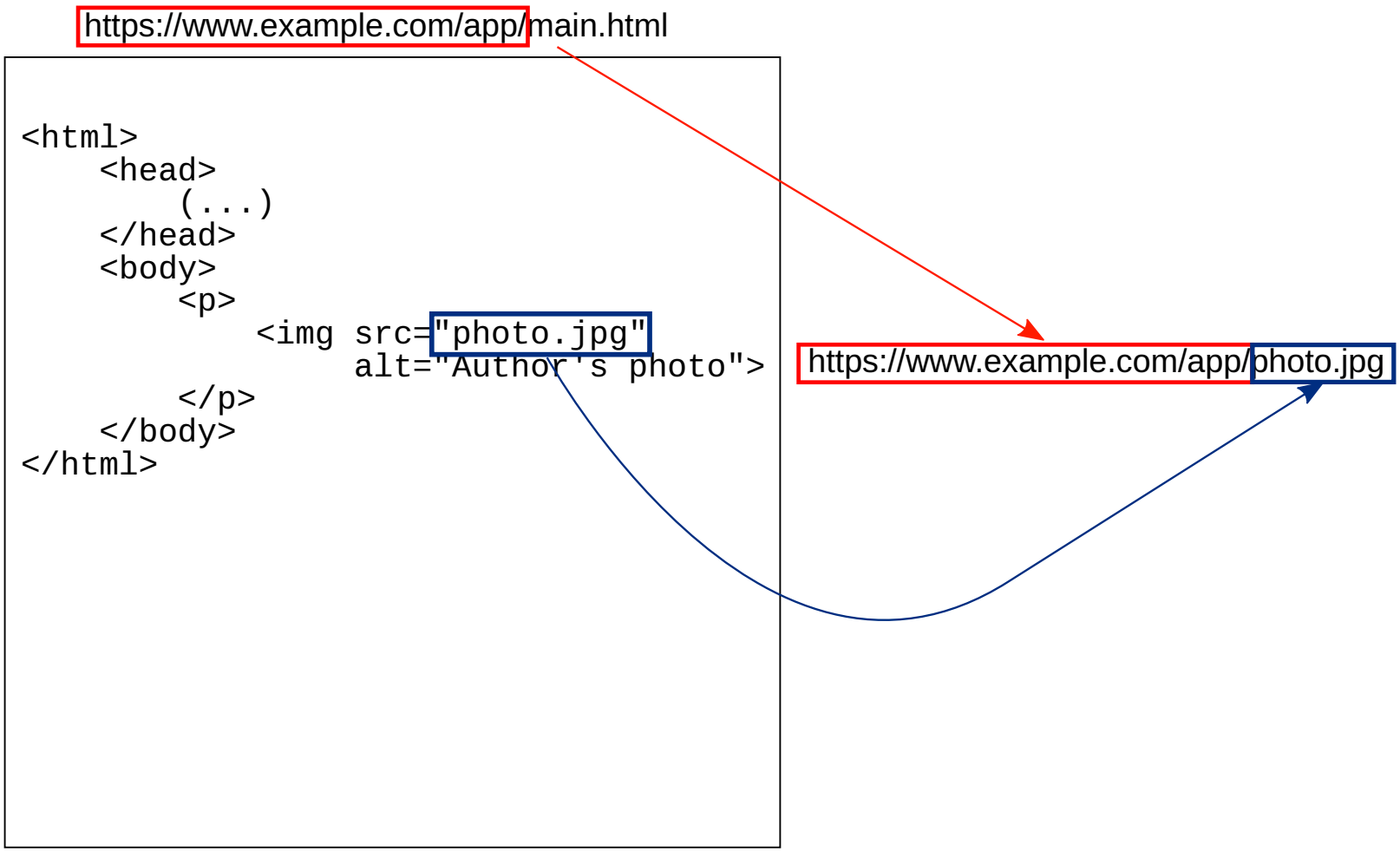
Una aplicación Web consiste típicamente en múltiples recursos, tanto estáticos como dinámicos. Sus URLs suelen compartir el mismo esquema y autoridad. Esto es, suele usarse el mismo protocolo, nombre de dominio y puerto TCP para todas ellas.

Una página HTML puede hacer referencia a otros recursos de la misma aplicación Web desde hipervínculos, formularios, imágenes, referencias a hojas de estilos, referencias a código JavaScript, etc.

Rutas relativas

En una página HTML a la que el cliente haya accedido a través de una URL dada, las **rutas relativas** permiten a los creadores evitar tener que especificar las URLs completas, proporcionando simplemente las partes de las mismas que cambian con respecto a la URL de dicha página.

Rutas relativas



Rutas relativas

En la mayoría de los casos se debería usar rutas relativas para hacer referencia a otras URLs de la misma aplicación:

- Las páginas HTML son más compactas y fáciles de leer, y los errores en URLs son menos frecuentes.
- Se pueden mover las aplicaciones a otro esquema o nombre de dominio sin necesidad de actualizar los enlaces en cada página HTML.
- Durante la fase de desarrollo, se puede acceder a los recursos en despliegues de la aplicación en local o en servidores que no son de producción.