DOCUMENTACION LEC API

Guillermo López Sanz

INDICE

1. INTRODUCCIÓN

1.1 ¿Para qué es la API?

La idea para la API es poder gestionar equipos, resultados, y jugadores de la LEC (League of Legends European Championship) en fase regular para la temporada 2023. En ella podremos acceder y modificar la base de datos incluida en el archivo (con sqlite) para poder actualizar los campos necesarios.

Es una API desarrollada con Python (usando flask, requests y sqlite).

1.2 ¿Qué se puede hacer con la API?

Hay muchas opciones desarrolladas para gestionar los datos de la LEC con la API, las cuales voy a enu merar a continuación y más tarde explicaré en detalle. Son las siguientes:

- Iniciar la BBDD
- GET equipos.
- GET equipo.
- GET jugadores.
- GET posiciones.
- GET resultados.
- INSERT jugador.
- DELETE jugador.
- UPDATE jugador.
- UPDATE equipo.
- INSERT resultado.
- Vista clasificación
- Vista equipo.
- Reset BBDD.

1.3 Introducción a la LEC.

La LEC es la liga europea del videojuego League of Legends, una de las más prestigiosas a nivel mundial.

El League of Legends (más conocido como LoL) es un videojuego en el que, sin entrar en mucho detalle, se enfrentan dos equipos de cinco jugadores en el que cada jugador tiene una posición. Estas posiciones son TOP, JUNGLE, MID, ADC, SUPPORT (añadidas en la tabla de posiciones, la cual no se puede modificar salvo que se edite a mano la bbdd, ya que no varían en el juego).

En fase regular, los equipos juegan un partido a la semana entre ellos, sumando una victoria si ganan, y una derrota si pierden. Al final de temporada, el equipo con más victorias y menos derrotas es el ganador.

2. EXPLICACIÓN DE MÉTODOS

2.1 INICIO

Hay un método implementado para crear la bbdd. Lo primero es crear el entorno virtual en nuestra carpeta. Para ello hay que ejecutar el siguiente comando dentro de la carpeta donde vayamos a tener el archivo app.py:

python3 -m venv env

Después activaremos el entorno virtual con el siguiente comando:

call env/Scripts/activate

En la carpeta donde se encuentre el archivo.py debe haber un archivo txt requirements.txt en el que introduciremos las siguientes librerías para instalarlas en el proyecto: Flask, requests y flask-cors. Instalamos mediante el siguiente comando:

pip install -r requirements.txt

Una vez instaladas, ejecutamos la aplicación con el siguiente comando:

flask –app app.py –debug run

Este comando activará el servidor y te dará el endpoint, el cual necesitarás para escribir cada ruta para ejecutar los métodos.

2.1.1 INIT (NO ES NECESARIA, YA ADJUNTO BBDD CON DATOS CON LA API)

Init es un método que si no existe la bbdd lec.db te la crea y crea las tablas si no existen.

Se ejecuta con la siguiente ruta (en mi caso mi endpoint es http://127.0.0.1:5000):

http://127.0.0.1:5000/init

```python
#funcion inicial para crear o conectar a la bbdd y crear tablas si no existen
@application.route("/init")
def iniciar():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    cur.execute('CREATE TABLE IF NOT EXISTS TEAMS(NAME,WINS,DEFEATS)')
    cur.execute('CREATE TABLE IF NOT EXISTS PLAYERS(NAME,TEAM,POSITION)')
    cur.execute('CREATE TABLE IF NOT EXISTS POSITIONS(POSITION)')
    cur.execute('CREATE TABLE IF NOT EXISTS RESULTS(TEAM1,TEAM2,WINNER)')
    con.close()
    return "API STARTED CORRECTLY!",200
```

2.2 MÉTODOS GET

El método GET se utiliza como petición por parte del front recursos al back. Los métodos GET implementados en la API son los siguientes:

2.2.1 GET TEAMS.

Este método devuelve un json con todos los equipos que hay en la BBDD. El método se ejecuta con la siguiente ruta:

http://127.0.0.1:5000/teams

```python
#Obtener json de todos los equipos con sus datos
@application.route("/teams",methods = ['GET'])
def get_teams():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    teams = cur.execute('SELECT * FROM TEAMS;')
    cont = 0
    json={}
    for row in teams:
        json[cont]={}
        json[cont]["name"]=row[0]
        json[cont]["wins"]=row[1]
        json[cont]["defeats"]=row[2]
        cont+=1
    con.close()
    return json,200
```

2.2.2 GET RESULTS

Este método devuelve un json con todos los resultados que hay en la BBDD. El método se ejecuta con la siguiente ruta:

http://127.0.0.1:5000/results

```python
#Obtener JSON con resultados
@application.route("/results",methods = ['GET'])
def get_results():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    results = cur.execute('SELECT * FROM RESULTS;')
    cont = 0
    json={}
    for row in results:
        json[cont]={}
        json[cont]["local"]=row[0]
        json[cont]["visitor"]=row[1]
        json[cont]["winner"]=row[2]
        cont+=1
    con.close()
    return json,200
```

2.2.3 GET PLAYERS

Este método devuelve un json con todos los jugadores que hay en la BBDD. El método se ejecuta con la siguiente ruta:

http://127.0.0.1:5000/players

```
#Obtener json de jugadores
@application.route("/players",methods = ['GET'])
def get_players():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    players = cur.execute('SELECT * FROM PLAYERS;')
    cont = 0
    json={}
    for row in players:
        json[cont]={}
        json[cont]["name"]=row[0]
        json[cont]["team"]=row[1]
        json[cont]["position"]=row[2]
        cont+=1
    con.close()
    return json,200
```

2.2.4 GET POSITIONS

Este método devuelve un json con todas las posiciones que hay en la BBDD. El método se ejecuta con la siguiente ruta:

http://127.0.0.1:5000/positions

```
#Obtener json de posiciones
@application.route("/positions",methods = ['GET'])
def get_positions():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    players = cur.execute('SELECT * FROM POSITIONS;')
    cont = 0
    json={}
    for row in players:
        json[cont]={}
        json[cont]["position"]=row[0]
        cont+=1
    con.close()
    return json,200
```

2.2.5 CARGAR TEMPLATE INSERT PLAYER

El método insert player es un método POST, pero ese POST se envía desde un template que se carga cuando llamamos a la siguiente ruta:

http://127.0.0.1:5000/insert_player

Esta ruta va a cargar obtener los equipos y posiciones que existen en la bbdd para cargar el template con estos datos, evitando así que el usuario pueda introducir datos de equipos o posiciones que no están en la bbdd.

```python
#Insert player a traves de un template
@application.route("/insert_player",methods=['GET'])
def render_insertplayer():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    teams = cur.execute('SELECT NAME FROM TEAMS;')
    array_teams=[]
    for row in teams:
        array_teams.append(row[0])
    positions = cur.execute('SELECT POSITION FROM POSITIONS;')
    array_positions = []
    for row2 in positions:
        array_positions.append(row2[0])
    con.close()
    data = {
        "teams" :array_teams,
        "positions": array_positions
    }
    return render_template('insert_player.html',**data),200
```

El template generado se ve de la siguiente manera:

Donde en –SELECT TEAM—están todos los equipos y en –SELECT POSITION—las posiciones.

Al pulsar el botón INSERT, se ejecutará la misma ruta pero esta vez con el método POST.

2.2.6 CARGAR TEMPLATE DE INSERT RESULT

De la misma manera que el método anterior uso este método. Cuando llamas a la siguiente ruta con el método GET se llama al template al cual se le pasan los equipos de la bbdd para poder generar resultados de la forma team1, team2, winner:

http://127.0.0.1:5000/new_result

```python
#Insert resultado a través de template
@application.route('/new_result',methods=['GET'])
def render_result():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    teams = cur.execute('SELECT * FROM TEAMS;')
    teams_array = []
    for row in teams:
        teams_array.append(row[0])
    con.close()
    return render_template('insert_result.html',teams=teams_array)
```

Se ve de la siguiente manera:

En cada SELECT aparecen todos los equipos que hay en la BBDD.

Al pulsar el botón INSERT se ejecuta la misma ruta pero cambiando el método a POST.

2.2.7 VISTA DE CLASIFICACIÓN:

Método que muestra una tabla con la clasificación de la LEC en base a los resultados registrados en la BBDD. Es un template al que se le pasan los datos de cada equipo y se genera una tabla con estos datos.

Se ejecuta con la siguiente ruta:

http://127.0.0.1:5000/leaderboard

```python
#Vista de la clasificacion
@application.route('/leaderboard', methods=['GET'])
def clasificacion():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    tabla = cur.execute('SELECT * FROM TEAMS ORDER BY WINS DESC, DEFEATS ASC')
    tabla_array = []
    for i in tabla:
        equipo = []
        for j in i:
            equipo.append(j)
        tabla_array.append(equipo)
    con.close()
    return render_template("table.html",tabla=tabla_array),200
```

La vista de la clasificación es la siguiente:

LEADERBOARD

TEAM	WINS	DEFEATS
VITALITY	2	0
SK GAMING	2	0
G2	1	0
ASTRALIS	1	0
BDS	1	1
EXCEL	1	1
KOI	0	1
HERETICS	0	1
MAD LIONS	0	2
FNATIC	0	2

Los equipos están ordenados por las victorias de manera descendente y derrotas de manera ascendente.

2.2.8 TEAM VIEW

Este método es un para cargar un template con una tabla con las stats de un equipo y otra tabla con sus jugadores. El equipo se elige a través de un template que tiene un select con los equipos que hay en la bbdd.

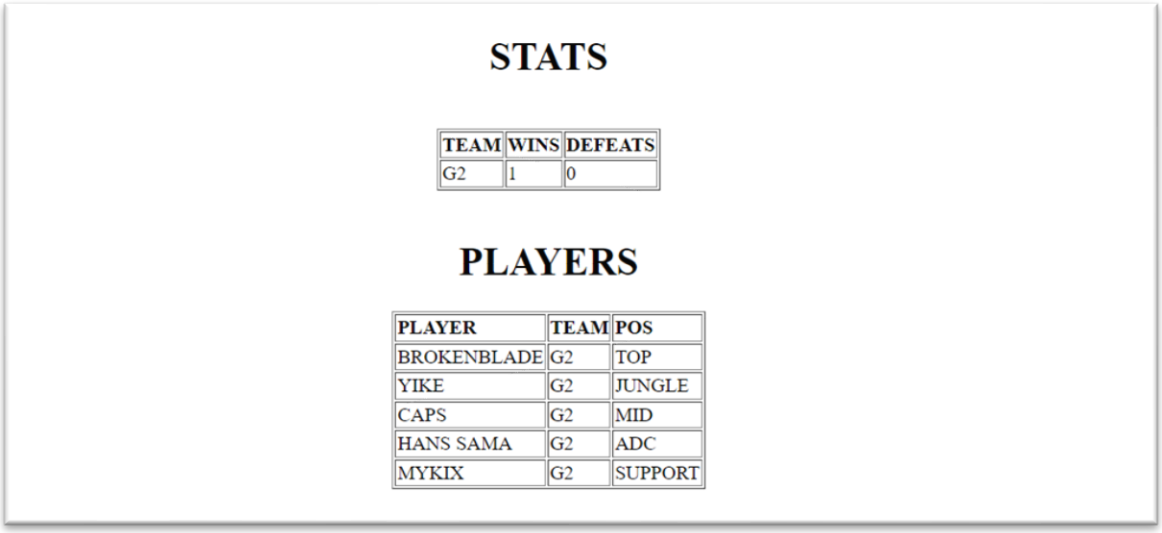
Ruta: *http://127.0.0.1:5000/team_view*

Al igual que los anteriores métodos, al pulsar el botón SELECT se vuelve a llamar la ruta en método POST y se muestra el template con la tabla.

```python
#Vista de un equipo (stats y jugadores)
@application.route('/team_view',methods=['GET'])
def render_team():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    teams = cur.execute('SELECT NAME FROM TEAMS;')
    array_teams=[]
    for row in teams:
        array_teams.append(row[0])
    con.close()
    return render_template('select_team.html',teams=array_teams),200
```

Aunque el team_view al pulsar el SELECT es un POST, voy a mostrar el resultado y el método en este apartado ya que es un POST sencillo de un equipo solo y al final el resultado es un GET de los datos de un equipo, pero que al necesitar pasarse el equipo como parámetro, es un método POST.

```python
@application.route('/team_view',methods=['POST'])
def team_view():
    team = request.form['team']
    if team!=None and team!="null":
        con = sqlite3.connect("lec.db")
        cur = con.cursor()
        players = cur.execute('SELECT * FROM PLAYERS WHERE TEAM=?',(team,))
        players_array=[]
        for row in players:
            player= []
            for i in row:
                player.append(i)
            players_array.append(player)
        equipo = cur.execute('SELECT * FROM TEAMS WHERE NAME=?',(team,))
        for i in equipo:
            equipo = []
            for j in i:
                equipo.append(j)
        return render_template('team.html',team=equipo,players = players_array,name=team)
    else:
        abort(400)
```



STATS

TEAM	WINS	DEFEATS
G2	1	0

PLAYERS

PLAYER	TEAM	POS
BROKENBLADE	G2	TOP
YIKE	G2	JUNGLE
CAPS	G2	MID
HANS SAMA	G2	ADC
MYKIX	G2	SUPPORT

Ejemplo para el equipo G2.

2.2.9 GET TEAM

Método para a través de query string param “name” devuelve un json con los datos de un equipo si este existe en la bbdd. Un ejemplo de ejecución sería:

Ruta: *http://127.0.0.1:5000/team?name=G2*

Este método devolvería un json con todos los datos del equipo G2.

```python
#Json de equipo (info del club y jugadores) a través de query string params
@application.route('/team',methods=['GET'])
def team():
    team = request.args.get('name')
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    teams = cur.execute('SELECT NAME FROM TEAMS')
    teams_array = []
    for row in teams:
        teams_array.append(row[0])
    if team!=None and team in teams_array:
        json = {}
        players = cur.execute('SELECT * FROM PLAYERS WHERE TEAM=?',(team,))
        players_array=[]
        for row in players:
            player= []
            for i in row:
                player.append(i)
            players_array.append(player)
        equipo = cur.execute('SELECT * FROM TEAMS WHERE NAME=?',(team,))
        for i in equipo:
            equipo = []
            for j in i:
                equipo.append(j)
        json['club'] = {'name':equipo[0],'wins':equipo[1],'defeats':equipo[2]}
        jugadores=[]
        playersjson = {}
        for i in players_array:
            playersjson[i[0]]={'team':i[1],'pos':i[2]}
        jugadores.append(playersjson)
        json['players']=jugadores
        return json
    else:
        abort(400)
```

2.3 MÉTODOS POST

El método HTTP POST envía datos al servidor. El tipo del cuerpo de la solicitud es indicada por la cabecera Content-Type.

2.3.1 POST INSERT PLAYER

Obtiene los datos del template generado con el método GET de la ruta:

http://127.0.0.1:5000/insert_player

Recibe los datos del POST y con ellos hace las comprobaciones necesarias para comprobar si se puede insertar el jugador, y si se cumplen lo inserta en la bbdd.

```
@application.route("/insert_player", methods=['POST'])
def insert_player():
    name = request.form["name"]
    team = request.form["team"]
    position = request.form["position"]
    if name != None and name != '' and team!='null' and position!='null':
        con = sqlite3.connect("lec.db")
        cur = con.cursor()
        data = ()
        data = (name,team,position)
        cur.execute('INSERT INTO PLAYERS VALUES (?,?,?)',data)
        con.commit()
        con.close()
        return "PLAYER "+data[0]+", "+data[1]+", "+data[2]+" INTROUCED",200
    else:
        abort(404)
```

2.3.2 POST de NEW RESULT

Realiza un insert en la bbdd de un nuevo resultado con los datos recuperados del template generado en el método get de la ruta:

http://127.0.0.1:5000/new_result

Tiene en cuenta que los equipos 1 y 2 no sean iguales y que el equipo ganador sea uno de los dos equipos seleccionados en los selects de equipo 1 y equipo 2.

```
@application.route("/new_result", methods=['POST'])
def insert_result():
    team1 = request.form["team1"]
    team2 = request.form["team2"]
    winner = request.form["winner"]

    if team1 != None and team1 != 'null'and team2!=None and team2!='null' and winner!=None and winner!='null' and (winner == team1 or winner==team2)
        con = sqlite3.connect("lec.db")
        cur = con.cursor()
        teams = cur.execute('SELECT * FROM TEAMS;')
        teams_array = []
        for row in teams:
            teams_array.append(row[0])
        print(team1+" "+team2+" "+winner)
        print(teams_array)
        if team1 in teams_array and team2 in teams_array and team1!=team2:
            data=()
            data= (team1,team2,winner)
            cur.execute('INSERT INTO RESULTS VALUES (?,?,?)',data)
            if winner==team1:
                wins1 = cur.execute('SELECT WINS FROM TEAMS WHERE NAME=?',(team1,))
                wins1_array = []
                for row in wins1:
                    wins1_array.append(row[0])
```

```
    wins1_num = wins1_array[0]
    wins1_num = int(wins1_num)
    wins1_num+=1
    cur.execute('UPDATE TEAMS SET WINS=? WHERE NAME=?',(wins1_num,team1))
    defs2 = cur.execute('SELECT DEFEATS FROM TEAMS WHERE NAME=?',(team2,))
    defs2_array = []
    for row in defs2:
        defs2_array.append(row[0])
    defs2_num = defs2_array[0]
    defs2_num = int(defs2_num)
    defs2_num+=1
    cur.execute('UPDATE TEAMS SET DEFEATS=? WHERE NAME=?',(defs2_num,team2))
    con.commit()
    con.close()
    return "RESULT "+team1+"-"+team2+"  WINNER:"+winner+" INTROUCED",200
else:
    wins2 = cur.execute('SELECT WINS FROM TEAMS WHERE NAME=?',(team2,))
    wins2_array = []
    for row in wins2:
        wins2_array.append(row[0])

    wins2_num = wins2_array[0]
    wins2_num = int(wins2_num)
    wins2_num+=1
    cur.execute('UPDATE TEAMS SET WINS=? WHERE NAME=?',(wins2_num,team2))
    defs1 = cur.execute('SELECT DEFEATS FROM TEAMS WHERE NAME=?',(team1,))
    defs1_array = []
    for row in defs1:
        defs1_array.append(row[0])
    defs1_num = defs1_array[0]
    defs1_num = int(defs1_num)
    defs1_num+=1
    cur.execute('UPDATE TEAMS SET DEFEATS=? WHERE NAME=?',(defs1_num,team1))
    con.commit()
    con.close()
    return "RESULT "+team1+"-"+team2+"  WINNER:"+winner+" INTROUCED",200
```

```
    else:
        abort(400)
else:
    abort(400)
```

2.4 PUT

La petición HTTP PUT crea un nuevo elemento o reemplaza una representación del elemento de destino con los datos de la petición.

2.4.1 MODIFY PLAYER

A través de postman o una herramienta similar, si ejecutamos la ruta con query string params de “name”,”team”,”position” se modificará el jugador de nombre name, con la posición o equipo pasado por parámetros si estos existen en la bbdd. El parámetro obligatorio es name y team y position pueden pasarse los dos, o uno de ellos.

Ejemplos de ejecución:

http://127.0.0.1:5000/modify_player?name=BROKENBLADE&team=KOI&position=JUNGLE

http://127.0.0.1:5000/modify_player?name=BROKENBLADE&team=KOI

http://127.0.0.1:5000/modify_player?name=BROKENBLADE &position=JUNGLE

Cualquiera de estas tres rutas ejecutarían correctamente el método.

```python
#Put player sin template (navegador no coge, hacerlo mediante herramientas como postman)
@application.route("/modify_player",methods=['PUT'])
def modify_player():
    name = request.args.get("name",None,str)
    team = request.args.get("team",None,str)
    position = request.args.get("position",None,str)
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    players = cur.execute('SELECT NAME FROM PLAYERS;')
    players_array = []
    for row in players:
        players_array.append(row[0])
    positions = cur.execute('SELECT POSITION FROM POSITIONS;')
    positions_array=[]
    for row in positions:
        positions_array.append(row[0])
    teams = cur.execute('SELECT NAME FROM TEAMS;')
    teams_array=[]
    for row in teams:
        teams_array.append(row[0])
    if name in players_array:
        if team != None and position !=None and team in teams_array and position in positions_array:
            cur.execute('UPDATE PLAYERS SET TEAM=?,POSITION=? WHERE NAME=?',(team,position,name))
            con.commit()
            con.close()
            return "PLAYER UPDATED",200
        elif team!=None and team in teams_array:
            cur.execute('UPDATE PLAYERS SET TEAM=? WHERE NAME=?',(team,name))
            con.commit()
            con.close()
            return "PLAYER UPDATED",200
        elif position!=None and position in positions_array:
            cur.execute('UPDATE PLAYERS SET POSITION=? WHERE NAME=?',(position,name))
            con.commit()
            con.close()
            return "PLAYER UPDATED",200
        else:
            abort(400)
    else:
        abort(400)
```

2.4.2 PUT TEAM

A través de postman o una herramienta similar, si ejecutamos la ruta con query string params de “new-team”,”team”, se modificará el equipo de nombre team, con el nombre pasado en new-team. Ambos parámetros son obligatorios y team tiene que existir en la BBDD.

```python
#Put team sin template (navegador no coge, hacerlo mediante herramientas como postman)
@application.route('/modify_team',methods=['PUT'])
def modify_team():
    new_team = request.args.get("new-team",None,str)
    team = request.args.get("team",None,str)
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    teams = cur.execute('SELECT * FROM TEAMS;')
    teams_array = []
    for row in teams:
        teams_array.append(row[0])
    if team in teams_array:
        if new_team!=None:
            cur.execute('UPDATE TEAMS SET NAME=? WHERE NAME=?',(new_team,team))
            cur.execute('UPDATE PLAYERS SET TEAM=? WHERE TEAM=?',(new_team,team))
            con.commit()
            con.close()
            return "TEAM AND PLAYERS OF THAT TEAM UPDATED",200
        else:
            abort(400)
    else:
        abort(400)
```

2.5 DELETE

El método DELETE borra un recurso en específico.

2.5.1 DELETE PLAYER

A través de postman o una herramienta similar, si ejecutamos la ruta con query string params de “name”,””, se eliminará el jugador de nombre name en la bbdd.

```python
#Delete player sin template (navegador no coge, hacerlo mediante herramientas como postman)
@application.route("/delete_player", methods=['DELETE'])
def delete_player():
    name = request.args.get("name")
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    players = cur.execute('SELECT NAME FROM PLAYERS;')
    players_array=[]
    for row in players:
        players_array.append((row[0]))
    if name != None:
        if name in players_array:
            cur.execute('DELETE FROM PLAYERS WHERE NAME=?',(name,))
            con.commit()
            con.close()
            return "PLAYER "+name+" WAS DELETED",200
        else:
            abort(404)
    else:
        abort(400)
```

2.5.2 RESET

Método para vaciar la bbdd salvo la tabla de posiciones que nunca va a cambiar. No se recomienda utilizar pues habría que volver a meter todos los datos manualmente o ejecutar una query con todas las “subquerys” para rellenar la bbdd.

```python
#reset de la bbdd
@application.route("/reset",methods=['DELETE'])
def reset():
    con = sqlite3.connect("lec.db")
    cur = con.cursor()
    cur.execute('DELETE * FROM TEAMS')
    cur.execute('DELETE * FROM PLAYERS')
    cur.execute('DELETE * FROM POSITIONS')
    cur.execute('DELETE * FROM RESULTS')
    con.commit()
    con.close()
    return "OK",200
```

3. ERRORES

Se ejecutan en situaciones como cuando no existe la ruta, se pasan como parámetros datos imposibles… Hay dos diseñados en la API.

```python
#Renders para errores 400 y 404
@application.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@application.errorhandler(400)
def page_not_found2(e):
    return render_template('400.html'), 400
```

Si se ejecuta el abort(err) siendo err 404 o 400, se ejecuta el template indicando que se trata de un error 400. El template es el siguiente: