

Bases de Dados

Lab 09 : Desenvolvimento de Aplicações e Transações

O guião de laboratório de Desenvolvimento de Aplicações e Transações é uma continuação dos guiões de Introdução às Bases de Dados e de SQL. Já deverá, portanto, ter uma base de dados do exemplo do Banco criada na sua conta no servidor **db.tecnico.ulisboa.pt**.

Nota: Se ainda não criou esta base de dados deverá seguir as instruções no Lab 1.

1ª Parte – Consulta simples da base de dados usando uma framework de Python o Flask

O código seguinte contém um script Python que se liga à base de dados e executa uma consulta SQL usando a framework de Flask¹, para obter todas as contas.

```
#!/usr/bin/python3
from wsgiref.handlers import CGIHandler

from flask import Flask

## PostgreSQL database adapter
import psycopg2
import psycopg2.extras

## SGBD configs
DB_HOST="db.tecnico.ulisboa.pt"
DB_USER="istxxxxxxx"
DB_DATABASE=DB_USER
DB_PASSWORD="xxxxxxx"
DB_CONNECTION_STRING = "host=%s dbname=%s user=%s password=%s" %
(DB_HOST, DB_DATABASE, DB_USER, DB_PASSWORD)

app = Flask(__name__)
```

¹ <https://flask.palletsprojects.com/>

```
# Na raiz do site '/' vamos listar as contas
@app.route('/')
def list_accounts():
    dbConn=None
    cursor=None
    try:
        dbConn = psycopg2.connect(DB_CONNECTION_STRING)
        cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
        query = "SELECT * FROM account;"
        cursor.execute(query)
        rowcount=cursor.rowcount

    # Python 3.6 introduced f-strings
    # We will use them here to build the HTML string
    html = f'''
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="utf-8">
        <title>List accounts - Python</title>
    </head>
    <body style="padding:20px">
        <table border="3">
            <thead>
                <tr>
                    <th>account_number</th>
                    <th>branch_name</th>
                    <th>balance</th>
                </tr>
            </thead>
            <tbody>
    '''
    for record in cursor:
        html += f'''
            <tr>
                <td>{record[0]}</td>
                <td>{record[1]}</td>
                <td>{record[2]}</td>
            </tr>
        '''
```

```
html += '''
        <tbody>
        </table>
    </body>
</html>
'''

return html # Renders the html string
except Exception as e:
    return str(e) # Renders a page with the error.
finally:
    cursor.close()
    dbConn.close()

CGIHandler().run(app)
```

Código 01: *test.cgi* - Flask framework

1. Grave este código num ficheiro **test.cgi** na sua máquina local.
2. Substitua os valores das variáveis **DB_USER** e **DB_PASSWORD** pelas suas credenciais de login no Postgres (username do Fénix, password do psql_reset)
3. Localize a pasta "~/web" no seu directório no sigma.tecnico.ulisboa.pt. Esta pasta "~/web" é o local onde deverá colocar os scripts Python.
4. Mova ou copie o ficheiro **test.cgi** para a sua pasta "~/web" no sigma.ist.utl.pt
5. Torne o ficheiro executável com o seguinte comando:

```
~/web$ chmod +x test.cgi
```

Nota: Verifique que se encontra na directoria do ficheiro (pode usar pwd, ls).

6. Este código tem algumas diferenças do utilizado no Lab 01. Neste caso são importados os seguintes módulos adicionais:
 - **from wsgiref.handlers import CGIHandler** é um interface standard para execução de aplicações em servidores web (e.g., Flask/Python em Apache).
 - **from flask import Flask** importa a web framework que vamos usar.

- `import psycopg2, psycopg2.extras` é um adaptador para PostgreSQL.
- 7. Em seguida instanciamos a classe Flask na variável *app* . Como vamos utilizar apenas um módulo, o argumento da classe Flask deve ser a variável `__name__`.
- 8. Na última linha com `CGIHandler().run(app)` definimos qual é a aplicação Python que o Web Server Gateway Interface (WSGI) deve executar.
- 9. Na framework Flask podemos usar o decorador² **route** para definir a função a executar para cada URL. Neste exemplo vamos usar o URL raiz `"/`.
- 10. Abra o Web browser na sua máquina local e aceda ao endereço³ incluindo a `"/` final:

`http://web2.ist.utl.pt/istxxxxxx/test.cgi/`

onde istxxxxxx deve ser substituído pelo seu nome de utilizador.

- 11. Deverá aparecer no browser uma tabela HTML com todas as contas.
- 12. No browser deverá aceder a "View Source" ou "View page source" e verificar o código HTML da gerado. Compare este HTML com o conteúdo do script **test.cgi**.

² <https://www.programiz.com/python-programming/decorator>

³ https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

2ª Parte - Utilização de template para gerar o HTML

Agora vamos adaptar o código anterior para utilizar um template. Para este efeito, utilizaremos o código apresentado na figura seguinte.

```
#!/usr/bin/python3
from wsgiref.handlers import CGIHandler
from flask import Flask
from flask import render_template
import psycopg2
import psycopg2.extras

## SGBD configs
DB_HOST="db.tecnico.ulisboa.pt"
DB_USER="istxxxxxx"
DB_DATABASE=DB_USER
DB_PASSWORD="XXXXXX"
DB_CONNECTION_STRING = "host=%s dbname=%s user=%s password=%s" %
(DB_HOST, DB_DATABASE, DB_USER, DB_PASSWORD)

app = Flask(__name__)

@app.route('/')
def list_accounts():
    dbConn=None
    cursor=None
    try:
        dbConn = psycopg2.connect(DB_CONNECTION_STRING)
        cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
        query = "SELECT * FROM account;"
        cursor.execute(query)
        return render_template("index.html", cursor=cursor)
    except Exception as e:
        return str(e) #Renders a page with the error.
    finally:
        cursor.close()
        dbConn.close()

CGIHandler().run(app)
```

Código 02: *app.cgi* - Flask com Templates

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>List accounts - Flask</title>
</head>
<body style="padding:20px">
{% if cursor %}
  <table border="2px">
    <thead>
      <tr>
        <th>Conta</th>
        <th>Agência</th>
        <th>Saldo</th>
      </tr>
    </thead>
    <tbody>
      {% for record in cursor %}
        <tr>
          <td>{{ record[0] }}</td>
          <td>{{ record[1] }}</td>
          <td>{{ record[2] }}</td>
        </tr>
      {% endfor %}
    </tbody>
  </table>
{% else %}
  <p> Erro: ao obter dados da base de dados!</p>
{% endif %}
</body>
</html>
```

Código 03: *index.html* - Template

1. Vamos copiar o código acima e guardar localmente num ficheiro *app.cgi* e *index.html* respectivamente o código 02 e 03.
2. Substitua os valores das variáveis **DB_USER** e **DB_PASSWORD** pelas suas credenciais (username do Fénix, password do *psql_reset*) em *app.cgi*

3. Novamente transferimos para o servidor **sigma.tecnico.ulisboa.pt** e colocamos em “~/web” e metemos executável: `chmod +x app.cgi`
4. A primeira alteração que vemos nesta versão é nas importações, importamos a classe **render_template** do módulo **Flask**
5. Em vez de definir a página HTML dentro da função com uma f-string vamos utilizar a função `render_template`. O Flask usa Jinja2⁴, uma biblioteca para gerar as páginas de HTML a partir de templates, que recorre aos seguintes tipos de delimitadores.
 - `{% ... %}` para [Declarações](#), ex:if, for, while
 - `{{ ... }}` para [Expressões](#) a serem substituídas no página gerada
 - `{# ... #}` para [Comentários](#) que não devem ser incluídos na página gerada.
 - `# ... ##` para [Declarações de linha](#)
6. O Flask importa os templates por nome de ficheiro de uma directoria de templates. Vamos então criar esta directoria em “~/web” no servidor **sigma.tecnico.ulisboa.pt** e mover o nosso ficheiro de template *index.html* para templates, com o comando:

```
~/web$ mkdir templates/  
~/web$ mv index.html templates/
```

7. Para que o template Jinja tenha acesso aos dados é necessário definir explicitamente as variáveis que vamos exportar no método **render_template**. Vamos neste caso exportar apenas o cursor da base de dados.
8. Agora no template (*index.html*) verificamos se recebemos a variável `cursor` com o statement `{% if %}` caso contrário `{% else %}` apresentamos um erro.
9. Caso tenha recebido o cursor vamos iterar com um `{% for %}` o cursor para obter os registos da base de dados.
10. Abra o Web browser na sua máquina local e aceda ao endereço⁵:

`http://web2.ist.utl.pt/istxxxxxx/app.cgi/`

onde istxxxxxx deve ser substituído pelo seu nome de utilizador.

11. Deverá aparecer uma tabela HTML com todas as contas como antes.

⁴ <https://jinja.palletsprojects.com/en/2.11.x/templates/>

⁵ Nota é importante que coloque a barra “/” final

3ª Parte – Alterações à Base de Dados

Agora vamos criar uma nova página para dar a possibilidade de modificar o saldo de uma conta e vamos utilizar um template para gerar a página. Para este efeito, utilizaremos o código apresentado na figura seguinte.

1. Adicione no final do *app.cgi* o Código 04 que adiciona um novo route *"/accounts"*

```
@app.route('/accounts')
def list_accounts_edit():
    dbConn=None
    cursor=None
    try:
        dbConn = psycopg2.connect(DB_CONNECTION_STRING)
        cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
        query = "SELECT account_number, branch_name, balance FROM account;"
        cursor.execute(query)
        return render_template("accounts.html", cursor=cursor)
    except Exception as e:
        return str(e)
    finally:
        cursor.close()
        dbConn.close()
```

Código 04: route('accounts')

2. Grave o código apresentado em seguida num ficheiro **accounts.html** na sua máquina local. Mova ou copie o ficheiro **accounts.html** para a sua pasta *"~/web/templates"* no **sigma.tecnico.ulisboa.pt**, e aceda à página:

<http://web2.ist.utl.pt/istxxxxxx/app.cgi/accounts>


```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>List accounts - Flask</title>
</head>
<body style="padding:20px">
{% if cursor %}
  <table border="2px">
    <thead>
      <tr>
        <th>Conta</th>
        <th>Agência</th>
        <th>Saldo</th>
        <th>Alterar</th>
      </tr>
    </thead>
    <tbody>
      {% for record in cursor %}
        <tr>
          <td>{{ record[0] }}</td>
          <td>{{ record[1] }}</td>
          <td>{{ record[2] }}</td>
          <td><a href="balance?account_number={{ record[0] }}">Alterar
saldo</a></td>
        </tr>
      {% endfor %}
    </tbody>
  </table>
{% else %}
  <p> Erro: não foi possível obter dados da base de dados!</p>
{% endif %}
</body>
</html>
```

Código 05: accounts.html - Template

- Deverá aparecer no browser uma lista de contas, com um link "Alterar montante" para cada uma delas.

Numero de Conta	Nome da Agência	Montante	Alterar
A-101	Downtown	500.0000	Alterar saldo
A-102	Uptown	700.0000	Alterar saldo
A-201	Uptown	900.0000	Alterar saldo
A-215	Metro	600.0000	Alterar saldo
A-217	University	650.0000	Alterar saldo
A-222	Central	550.0000	Alterar saldo
A-305	Round Hill	800.0000	Alterar saldo
A-333	Central	750.0000	Alterar saldo
A-444	Downtown	850.0000	Alterar saldo

4. Com o botão direito do rato sobre a janela do browser, aceda ao código HTML da página. Compare este HTML com o conteúdo do script **accounts.html**. Em particular, observe com atenção como são construídos os links "Alterar saldo".
5. Vamos adicionar uma nova função em **app.cgi** para gerar o template **balance.html**, para tal vamos introduzir o código abaixo (Código 06) depois da função introduzida no passo 1) **list_accounts_edit()**

```
@app.route('/balance')
def change_balance():
    try:
        return render_template("balance.html", params=request.args)
    except Exception as e:
        return str(e)
```

Código 06: route('/balance')

6. Com o seguinte template, vamos criar um formulário para submeter o novo saldo de uma dada conta.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Change balance - Flask</title>
</head>
<body>
  <h3>Alterar saldo da conta {{ params.get("account_number") }}</h3>
  <form action="update" method="post">
    <p><input type="hidden" name="account_number" value="{{
params.get('account_number') }}" /></p>
    <p>Novo saldo: <input type="text" name="balance" /></p>
    <p><input type="submit" value="Submit" /></p>
  </form>
</body>
</html>

```

Código 07: *balance.html* - Template para alterar saldos

7. Repare que o código `{{ params.get("account_number") }}` está a ser usado para preencher o valor do número de conta. De onde vem este número de conta?
8. Precisamos de importar o objeto Request. Vamos alterar o código de **app.cgi** na linha 5 onde está **from flask import render_template** necessário adicionar **request**. A linha deve ficar:


```
from flask import render_template, request
```

 Repare que no método **render_template** adicionamos a variável **params** que usa propriedade **args**⁶ do objeto **request** importado na linha 5 de **app.cgi**
9. Grave o código acima (Código 07 num ficheiro **balance.html** na sua máquina local. Mova ou copie o ficheiro **balance.html** para a sua pasta "~/web/template" no sigma.tecnico.ulisboa.pt e regresse à página:

<http://web2.ist.utl.pt/istxxxxxx/app.cgi/accounts>
10. Clique no link "Alterar saldo" do lado direito da conta A-305. Deverá aparecer o seguinte formulário:

⁶ Retorna os parâmetros decompostos do URL - <https://flask.palletsprojects.com/en/2.1.x/api/#flask.Request.args>

Alterar saldo para a conta A-305

Novo saldo:

11. Vamos agora criar uma função para receber os dados deste formulário e atualizar o saldo da conta. Para este efeito, usamos o código seguinte:

```
@app.route('/update', methods=["POST"])
def update_balance():
    dbConn=None
    cursor=None
    try:
        dbConn = psycopg2.connect(DB_CONNECTION_STRING)
        cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
        balance=request.form["balance"]
        account_number=request.form["account_number"]
        query = 'UPDATE account SET balance=%s WHERE account_number = %s'
        data=(balance, account_number)
        cursor.execute(query,data)

        return query
    except Exception as e:
        return str(e)
    finally:
        dbConn.commit()
        cursor.close()
        dbConn.close()
```

Código 08: route('/update')

12. Insira o código acima no ficheiro **app.cgi** depois da função **change_balance()**. Regresse à página:

<http://web2.ist.utl.pt/istxxxxxx/app.cgi/accounts>

13. Clique no link "Alterar saldo" para a conta A-305. Deverá aparecer o formulário.

14. Preencha o formulário, indicando o novo saldo para a conta A-305 (por exemplo: 600).
15. Pressione o botão “submit”. Deverá aparecer no browser a instrução de UPDATE que foi executada sobre a base de dados.
16. O saldo da conta A-305 foi atualizado. Aceda ao endereço:

<http://web2.ist.utl.pt/istxxxxxx/app.cgi/>

17. Deverá aparecer no browser uma tabela HTML com todas as contas. Confirme que a conta A-305 tem o novo saldo que indicou no formulário.

account_number	branch_name	balance
A-101	Downtown	500.0000
A-102	Uptown	700.0000
A-201	Uptown	900.0000
A-215	Metro	600.0000
A-217	University	650.0000
A-222	Central	550.0000
A-333	Central	750.0000
A-444	Downtown	850.0000
A-305	Round Hill	600.0000

4ª Parte – Atualizações Concorrentes e Transações

1. Abra um terminal para o sistema Postgres e use a base de dados “bank” (istxxxxxx).
2. Escreva uma consulta para obter os dados das contas do cliente *Cook*.
3. O cliente *Cook* pretende transferir 500€ da conta A-102 para a conta A-101. Inicie uma transação com o comando `START TRANSACTION;`
4. Coloque 500€ na conta A-101.
5. Consulte os saldos das contas do cliente *Cook*. Neste momento, quanto dinheiro tem o cliente *Cook* no banco?
6. Abra um segundo terminal e ligue-se à base de dados com esta transação em curso.
7. Consulte os saldos das contas do cliente *Cook*. Afinal quanto dinheiro tem no banco?

8. No primeiro terminal, onde está a correr a transação, retire 500€ da conta A-102.
9. No segundo terminal consulte os saldos das contas do cliente *Cook*.
10. No primeiro terminal confirme a transação (execute a instrução COMMIT)
11. No segundo terminal consulte novamente os saldos das contas do cliente *Cook*. Confirme os resultados da transação.

5ª Parte – Implementação de Transações em Flask

1. Implemente o mecanismo de transações em Python.
2. O cliente *Cook* acabou de abastecer 25€ de gasolina numa estação de serviço e vai pagar com multibanco (conta A-101). Ao mesmo tempo, a sua esposa, que tem um cartão multibanco para a mesma conta, vai levantar 50€ numa máquina multibanco.
3. Abra um terminal para a base de dados e inicie uma transação.
4. Abra uma janela do browser para a página

<http://web2.ist.utl.pt/istxxxxxx/app.cgi/accounts>

5. Na primeira transação retire 25€ da conta (abastecimento).
6. No browser retire 50€ da conta (levantamento). O que se passa quando tenta fazer isto? Porquê?
7. A linha telefónica da estação de serviço está intermitente e a ligação cai. O pagamento que estava a ser feito tem de ser cancelado. Cancele essa transação. (Garantir o ROLLBACK)
8. Ao mesmo tempo que faz a alínea 7, repare no que acontece no browser. Como explica este fenómeno?
9. Consulte novamente os saldos das contas do cliente *Cook*.

No final o seu ficheiro *app.cgi* deve ficar como o código seguinte:

```
#!/usr/bin/python3
from wsgiref.handlers import CGIHandler
from flask import Flask
from flask import render_template, request

import psycopg2
import psycopg2.extras

## SGBD configs
DB_HOST="db.tecnico.ulisboa.pt"
DB_USER="istxxxxxx"
DB_DATABASE=DB_USER
DB_PASSWORD="xxxxxxx"
DB_CONNECTION_STRING = "host=%s dbname=%s user=%s password=%s" %
(DB_HOST, DB_DATABASE, DB_USER, DB_PASSWORD)

app = Flask(__name__)

@app.route('/')
def list_accounts():
    dbConn=None
    cursor=None
    try:
        dbConn = psycopg2.connect(DB_CONNECTION_STRING)
        cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
        query = "SELECT * FROM account;"
        cursor.execute(query)
        return render_template("index.html", cursor=cursor)
    except Exception as e:
        return str(e) # Renders a page with the error.
    finally:
        cursor.close()
        dbConn.close()

@app.route('/accounts')
def list_accounts_edit():
    dbConn=None
    cursor=None
    try:
```

```
dbConn = psycopg2.connect(DB_CONNECTION_STRING)
cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
query = "SELECT account_number, branch_name, balance FROM account;"
cursor.execute(query)
return render_template("accounts.html", cursor=cursor,
params=request.args)
except Exception as e:
    return str(e)
finally:
    cursor.close()
    dbConn.close()

@app.route('/balance')
def change_balance():
    try:
        return render_template("balance.html", params=request.args)
    except Exception as e:
        return str(e)

@app.route('/update', methods=["POST"])
def update_balance():
    dbConn=None
    cursor=None
    try:
        dbConn = psycopg2.connect(DB_CONNECTION_STRING)
        cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
        balance=request.form["balance"]
        account_number=request.form["account_number"]
        query = 'UPDATE account SET balance=%s WHERE account_number = %s'
        data=(balance, account_number)
        cursor.execute(query,data)
        return query
    except Exception as e:
        return str(e)
    finally:
        dbConn.commit()
        cursor.close()
        dbConn.close()

CGIHandler().run(app)
```


Estrutura de ficheiros

No servidor sigma deverá ter a seguinte estrutura de ficheiros que este guião funcione.

```
~  
|  
-web  
|  |-templates  
|  |      |-accounts.html  
|  |      |-balance.html  
|  |      |-index.html  
|  |- app.cgi  
|  |- test.cgi
```

Os ficheiros .cgi não devem ter permissões de escrita para o owner nem nenhum dos diretórios onde se encontra. Se necessário execute no terminal `chmod -R 755 ~/web` para aplicar as permissões à directoria recursivamente.