TÉCNICO LISBOA

Desenvolvimento de Aplicações com Bases de Dados

Bases de Dados 2021/2022

Mário Gaspar da Silva

Bibliografia

- **Capítulo 9.**
  - Avi Silberschatz, Henry F. Korth, S. Sudarshan. *Database System Concepts*, Seventh Edition. McGraw-Hill ISBN 9780078022159.
- Driver para acesso a Postgres em Python
  - https://www.psycopg.org/docs/usage.html
- Framework para desenvolvimento de Aplicações Web em Python
  - https://flask.palletsprojects.com/en/1.1.x/quickstart/

Aplicações e Interfaces de Utilizador

- A maioria dos utilizadores não usa uma linguagem de interrogação como SQL
  - “power users” dominam-na e usam-na para produzir ficheiros com dados para serem trabalhados noutras aplicações (como folhas de cálculo)
- O programa de aplicação (ou serviço) faz de intermediário entre a base de dados e os utilizadores (ou utentes).
- Aplicações organizadas com
  - Front-end (interface de utilizador)
  - Backend (retaguarda)
  - Middle-layer

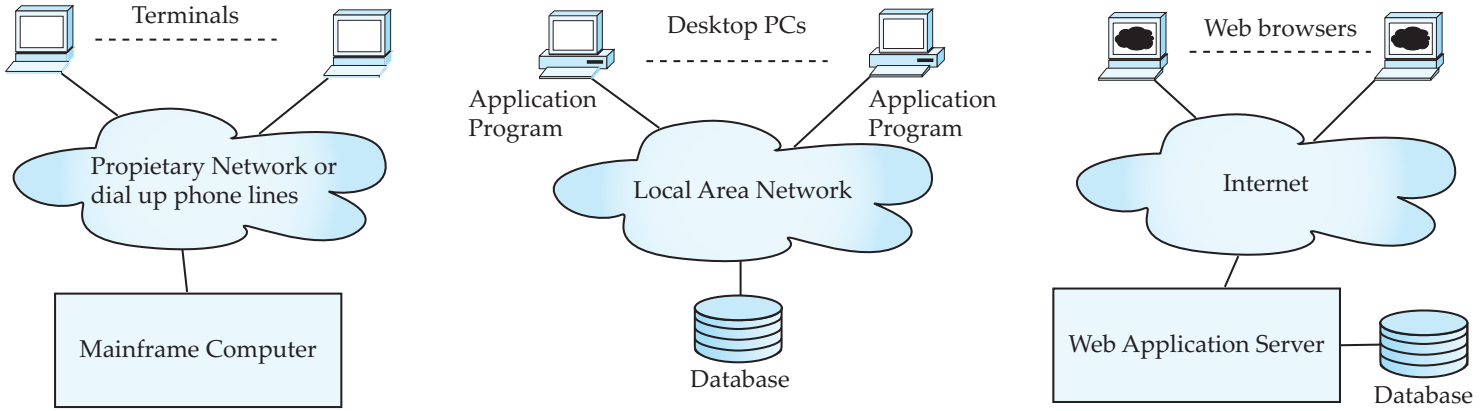
Front-end: user interface

- Forms
- Graphical user interfaces
- Many interfaces are Web-based

SQL em Aplicações

- Os comandos SQL podem ser invocados através de uma **aplicação** informática
  - Escrita numa qualquer linguagem
    - Java, C, PHP, Perl, Python, …

Arquitectura de Aplicações: 4 Eras



(a) Mainframe Era (b) Personal Computer Era (c) Web era

1. Mainframe (1960´s and 70's)
2. Personal computer (1980's)
3. Web (mid 1990's onwards)
4. Web and Smartphone (2010 onwards)

Interface de Utilizador Web

- Standard de facto em interfaces humanas para acesso a bases de dados.
- Evita instalação de programas, com interface de razoável qualidade.
  - Código HTML/JavaScript descarregado dinamicamente

Exemplo de Pedido

GET **index.html** HTTP/1.1

User-agent: Mozilla/4.0

Accept: text/html, image/gif, image/jpeg

Exemplo de Resposta

HTTP/1.1 200 OK

Date: Mon, 04 Mar 2002 12:00:00 GMT

Content-Length: 1024

Content-Type: text/html

Last-Modified: Mall, 22 JUll 1998 09:23:24 GMT

<HTML>

<HEAD>

</HEAD>

<BODY>

<H1>Barns and Nobble Internet Bookstore</H1>

Our inventory:

<H3>Science</H3>

<B>The Character of Physical Law</B>

</BODY>

</HTML>

HTML

- É uma ***markup language*** porque anota o texto com etiquetas (*tags*):
  - <HTML> … </HTML>
  - <TITLE> … </TITLE>
  - <H3> … </H3>
- Web Browsers apresentam o conteúdo do de uma forma "gráfica" ao utilizador

https://www.w3schools.com/html/default.asp

Web API

- O cliente é uma aplicação e não um Web Browser
  (ou um programa, por ex em JavaScript, a correr no browser)
- O resultado do servidor é normalmente XML
  - HTML limitado a um conjunto de etiquetas
  - Extensible Markup Language (XML) – aplicação define etiquetas
  - HTML5 é um dialecto de XML

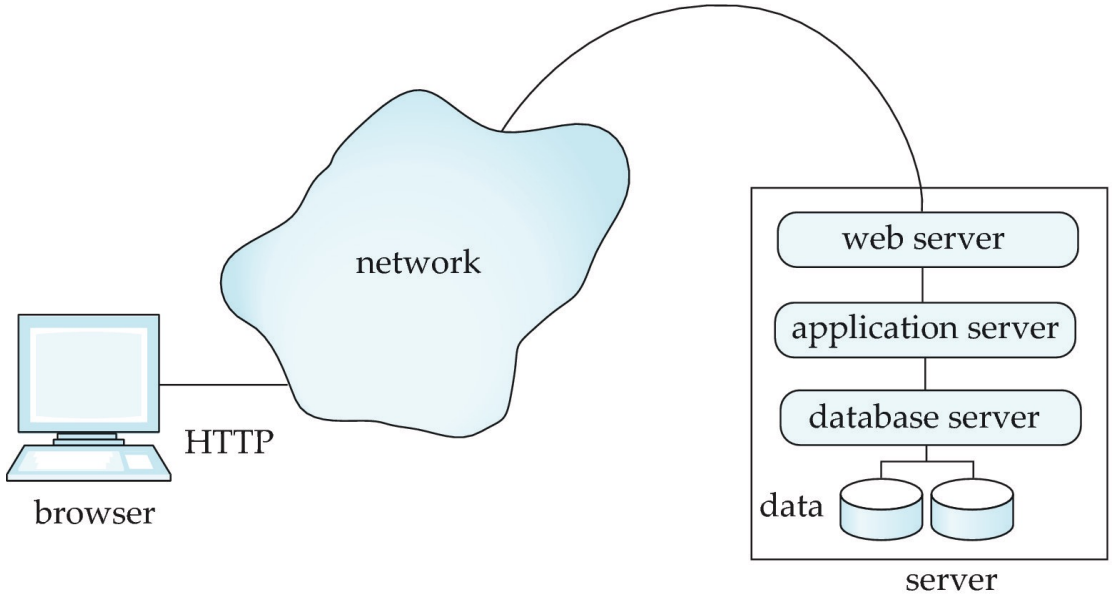
Exemplo de texto fonte HTML

```
<html>

<body>

 <table border>
   <tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
   <tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
   ....

 </table>

  <form action="PersonQuery" method=get>
   Search for:
     <select name="persontype">
        <option value="student" selected>Student </option>
        <option value="instructor"> Instructor </option>
     </select> <br>
   Name: <input type=text size=20 name="name">
   <input type=submit value="submit">

 </form>

</body> </html>
```

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

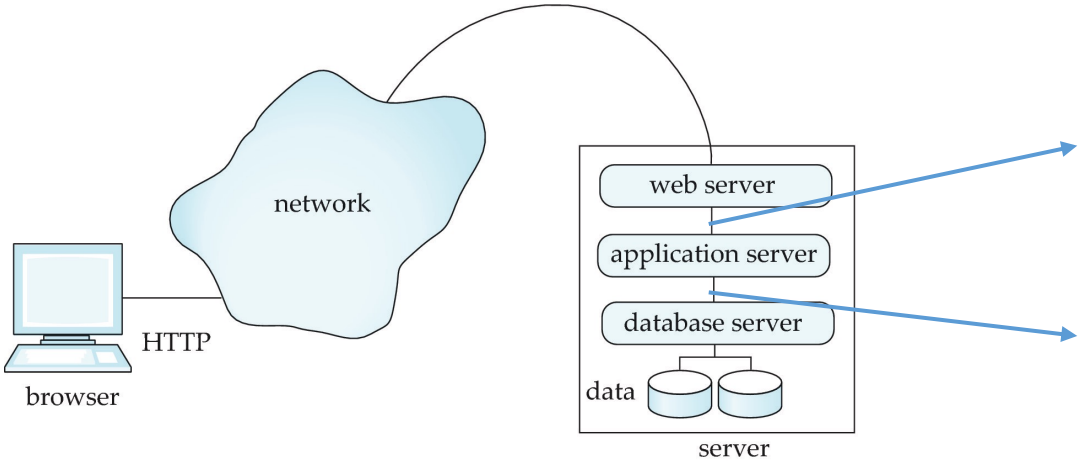
Search for: Student
Name:
submit

Arquitectura de Aplicações Web



(3 camadas)

API (Application Program Interface)



- CGI (Common Gateway Interface)

  Interface standard entre

  servidor web ←→ aplicação

- Open Database Connectivity (ODBC)/JDBC

  Interface standard entre

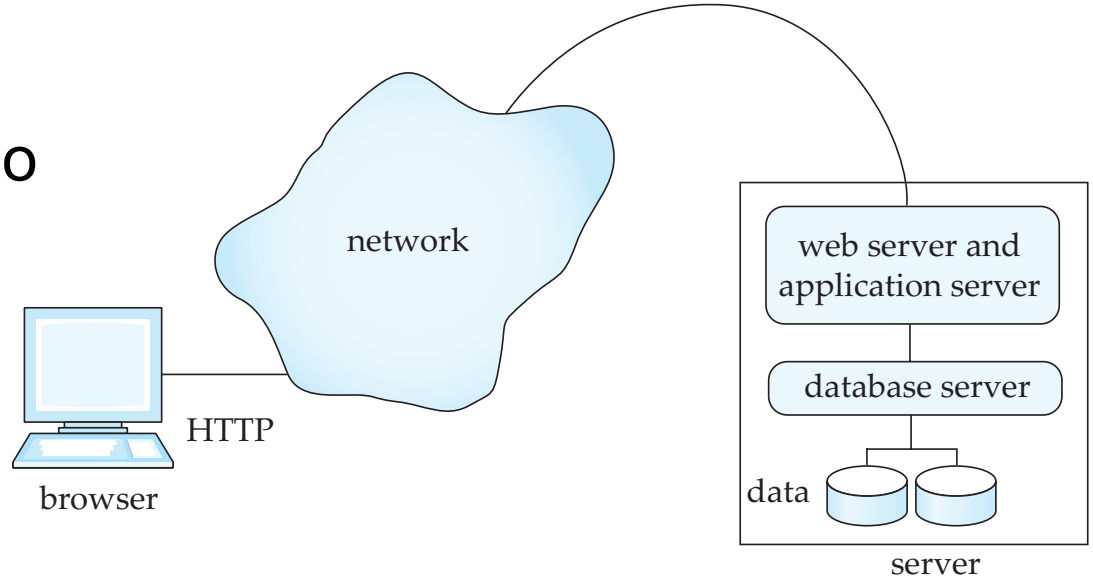
  Aplicação ←→ SGBD

Arquitectura de Aplicações Web

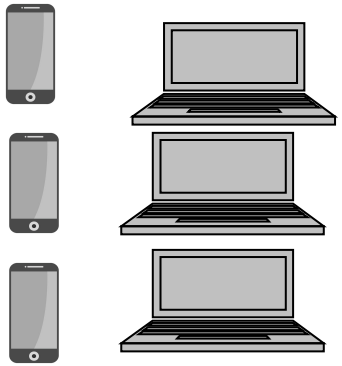
Cada indirecção
entre níveis de abstração

tem custos acrescidos

Arquitectura com 2 camadas



Distribuição Física (3 Fileiras)



Web Browsers
JavaScript

HTML, XML
HTTP

Servidor Aplicacional
Java, Python, PHP

EX: web2.tecnico.ulisboa.pt

SQL
JDBC, ODBC

Servidor de Dados – SGBD
MySQL/MariaDB, Postgres

EX: db.tecnico.ulisboa.pt

TÉCNICO LISBOA

Programação bi-linguagem

SQL+X

Adaptação de Dados

- O resultado de qualquer interrogação SQL é um **conjunto de tuplos**
  - Este tipo de dados normalmente não existe nas linguagens de programação imperativas.
- Integração com SQL usa o **cursor** como mecanismo de adaptação. Permite iterar sobre os tuplos de um resultado.

Cursores

```
EXEC SQL DECLARE sinfo CURSOR FOR
        SELECT  S.sname
        FROM  Sailors S, Boats B, Reserves R
        WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color='red'
        ORDER BY  S.sname
```

1. Declaração
2. Abertura
3. Operação sobre o tuplo para onde aponta
4. Deslocação do cursor para o tuplo seguinte
5. Fecho

Duas abordagem de integração

Estática:

SQL embebido na linguagem

- Embedded SQL
- SQLJ, uma extensão ao Java

Dinâmica:

API para invocar comandos SQL

- Dynamic SQL
- JDBC (Java DataBase Connectivity)
- PDO (PHP Data Objects)
- **Psycopg**

Embedded SQL

- Ligação à base de dados:

  EXEC SQL CONNECT

- Declaração de variáveis:

  EXEC SQL BEGIN (END) DECLARE SECTION

- Comandos

  EXEC SQL Statement;

Variáveis

EXEC SQL BEGIN DECLARE SECTION

```
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
```

EXEC SQL END DECLARE SECTION

Variáveis de erro (pré-definidas):

- **SQLCODE** (long, negativo se existir um erro)
- **SQLSTATE** (char[6], indica o tipo do erro)

Exemplo em C com SQL

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
     SELECT S.sname, S.age     FROM Sailors S
     WHERE S.rating > :c_minrating
     ORDER BY S.sname;
do {
     EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
     printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Embedded SQL

Construção de instruções SQL em tempo de execução (run-time)

```
char c_sqlstring[]=
  {“DELETE FROM Sailors WHERE rating>5”};
EXEC SQL PREPARE readytogo FROM
  :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

Combinação Java + SQL

Estática

- SQLJ

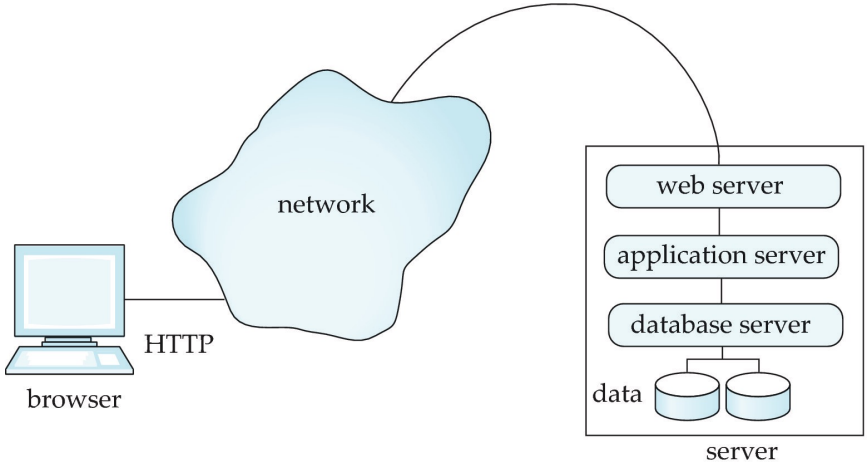
Dinâmica

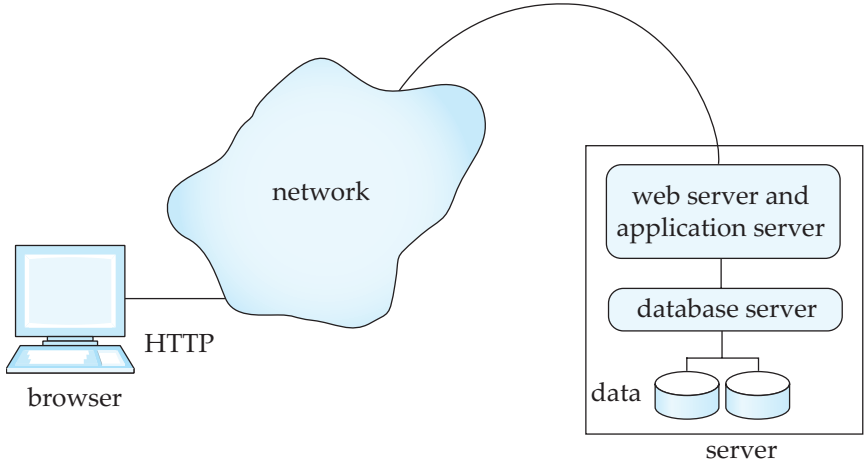
- JDBC - Java Database Connectivity

TÉCNICO LISBOA

Programação de Aplicações Web

Arquitectura de Aplicações Web





Sessões com HTTP

- The HTTP protocol is **connectionless**
  - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - In contrast, Unix logins, and JDBC/ODBC connections stay connected until the client disconnects
    - retaining user authentication and other information
  - Motivation: reduces load on server
    - operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - E.g., user authentication should be done only once per session
- Solution: use a **cookie**

Sessões com HTTP: Cookies

- A **cookie** is a small piece of text containing identifying information
  - Sent by server to browser
    - Sent on first interaction, to identify session
  - Sent by browser to the server that created the cookie on further interactions
    - part of the HTTP protocol
  - Server saves information about cookies it issued, and can use it when serving a request
    - E.g., authentication information, and user preferences
- Cookies can be stored permanently or for a limited time

Server-Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
  - Define an HTML document with embedded executable code/SQL queries.
  - Input values from HTML forms can be used directly in the embedded code/SQL queries.
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
  - JSP, PHP
  - General purpose scripting languages: VBScript, Perl, **Python**

Python – Linguagem Interpretada

Frameworks

- **Flask**
- Django

Modelo de programação equivalente a JSP com Python.

- Driver
  - **Psycopg**

Oferece modelo de acesso a SGBD equivalente a ODBC/JDBC em Python

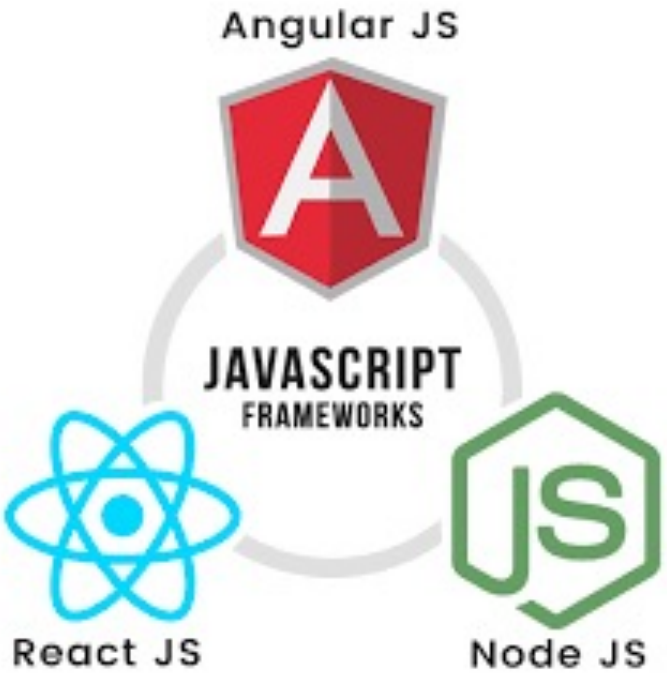
Client-Side Scripting

- Browsers can fetch certain scripts (client-side scripts) or programs along with documents, and execute them in "safe mode" at the client site
  - **Javascript**
  - ~~Adobe Flash and Shockwave~~ for animation/games (now defunct, → **HTML5**)
  - ~~Applets~~ (now defunct, painful)
- **Client-side scripts/programs allow documents to be active**
  - E.g., animation by executing programs at the local site
  - E.g., ensure that values entered by users satisfy some correctness checks
  - Permit flexible interaction with the user.
    - Executing programs at the client site speeds up interaction by avoiding many round trips to server

Validação de regras de integridade com JavaScript (Exemplo)

```
<html> <head>
   <script type="text/javascript">
      function validate() {
         var credits=document.getElementById("credits").value;
         if (isNaN(credits)|| credits<=0 || credits>=16) {
            alert("Credits must be a number greater than 0 and less than 16");
            return false
        }
      }
   </script>


</head> <body>
   <form action="createCourse" onsubmit="return validate()">
      Title: <input type="text" id="title" size="20"><br />
      Credits: <input type="text" id="credits" size="2"><br />
      <Input type="submit" value="Submit">
   </form>
</body> </html>
```

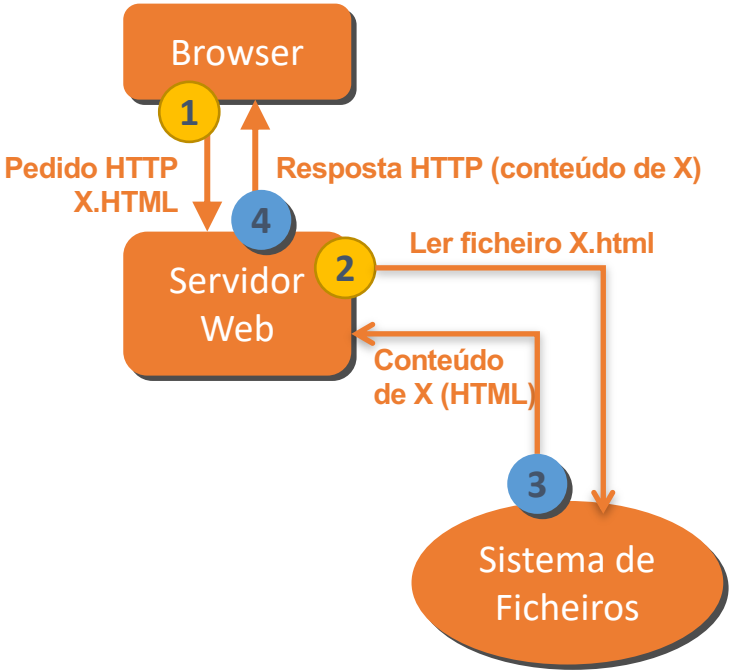
JavaScript Frameworks

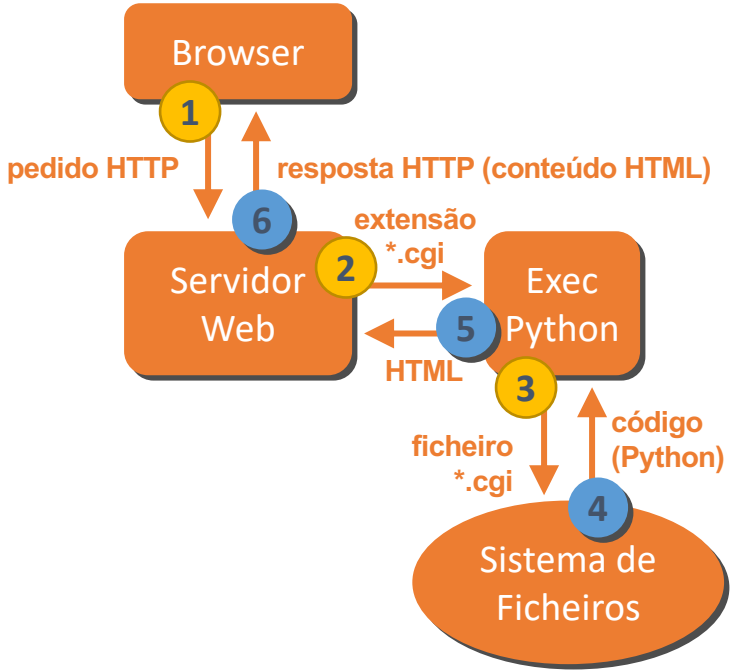


TÉCNICO LISBOA

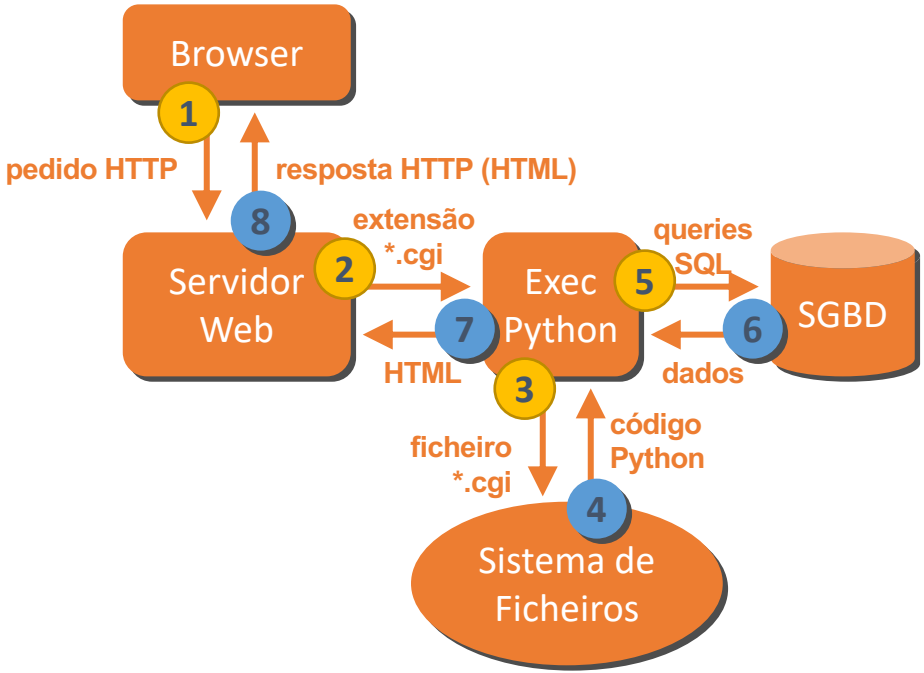
Psycopg+Flask

Execução Servidor Web





Execução Servidor Web + SGBD



Python CGI

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

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

Flask

```
app = Flask(__name__)   # __name__ ➔ ~/web/test.cgi.
@app.route('/')         # @? Decorator -- Meta-programming
                        # https://www.programiz.com/python-programming/decorator
                        # <test.cgi> /  ➔ def list_accounts()
def list_accounts():    #
  …a completar…
   return <html-string>
CGIHandler().run(app)
```

list_accounts()

```
def list_accounts():
 dbConn=None;  cursor=None
  try:
    dbConn = psycopg2.connect(DB_CONNECTION_STRING)
    cursor = dbConn.cursor(cursor_factory = psycopg2.extras.DictCursor)
    cursor.execute("SELECT * FROM account;")

    html = ….. A completer com código que gera HTML com conteúdo do cursor
    return html        #Renders the html string
  except Exception as e:
    return e #Renders a page with the error.
  finally:
    cursor.close();    dbConn.close()
```

list_accounts()

```
html=f'''
      <!DOCTYPE html>
      <title>List accounts - Python</title>  <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>
       '''
```

https://realpython.com/python-f-strings/

List_accounts() com Flask Templates

```
@app.route('/').   # <app.cgi> → list_accounts()
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).  # index.html ➔ ~/web/templates/ /index.html
  except Exception as e:
    return str(e) #Renders a page with the error.
  finally:
    cursor.close();     dbConn.close()
```

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

Templates Jinja2

O flask procura os templates por nome de ficheiro na pasta **templates**

Delimitadores:

- {% ... %} Declarações, ex: if, for, while
- {{ ... }} para Expressões a serem substituídas no página gerada
- {# ... #} para Comentários que não devem ser incluídos na página gerada.
- # para linha com instrução

Template index.html

```
<!doctype html>
<title>List accounts - Flask</title>  <body style="padding:20px">
  {% if cursor %}
    <table border="2px">
     <thead>  <tr>  <th>account_number</th>  <th>branch_name</th> <th>balance</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> Error: failed to retrieve data from database!</p>
  {% endif %}
</body>
```

Listas contas (com link para alteração)

@app.route('/accounts') # <app.cgi>/accounts ➔ list_accounts_edit()

```
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()
```

Template accounts.html (c/ link para alterar saldo)

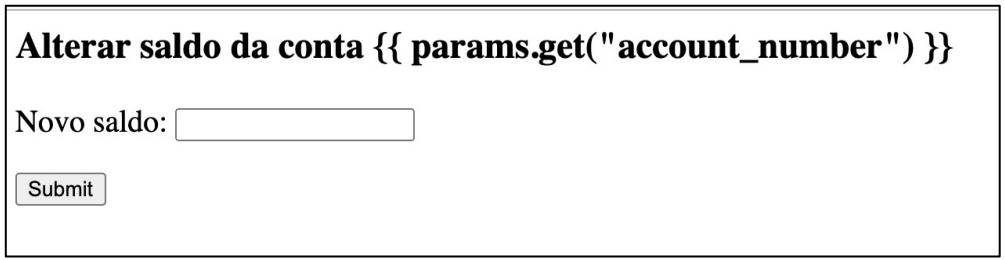
```
<!doctype html>
<title>List accounts - Flask</title> <body style="padding:20px">
  {% if cursor %}
    <table border="2px">
      <thead> <tr>   <th>Numero de Conta</th>  <th>Nome da Agência</th>  <th>Saldo</th>    <th>Alterar</th>    </tr>   </thead>
      <tbody>
        {% for record in cursor %}
          <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: ao obter dados da base de dados!</p>
  {% endif %}
</body>
```

Consultar Saldos

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

Template balance.html

```
<html>
    <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>
```



Forms

```
@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)
    query = f"'UPDATE account SET balance={request.form["balance"]} WHERE account_number = '{request.form["account_number"]}';"'
    cursor.execute(query)
    return query
  except Exception as e:
    return str(e)
  finally:
    dbConn.commit();    cursor.close();    dbConn.close()
```

Processamento de Form

```
@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)
    query = f"'UPDATE account SET balance={request.form["balance"]} WHERE account_number = '{request.form["account_number"]}';"'
    cursor.execute(query)
    return query
  except Exception as e:
    return str(e)
  finally:
    dbConn.commit();    cursor.close();    dbConn.close()
```

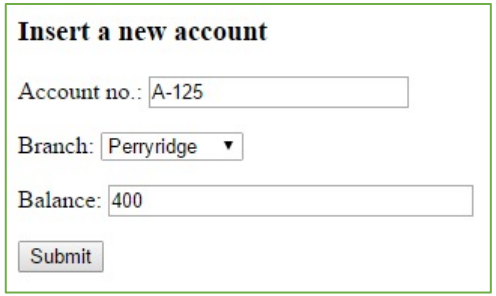

ZU 0666',0,0); DROP DATABASE TABLICE;--

Injecção de SQL (ex. Em PHP)

```
...
$account_number = $_REQUEST['account_number'];
$branch_name = $_REQUEST['branch_name'];
$balance = $_REQUEST['balance'];

$connection = new PDO($dsn, $user, $pass);

$sql = "INSERT INTO account VALUES
('$account_number', '$branch_name', $balance)";
1$
$nrows = $connection->exec($sql);

echo("<p>Rows inserted: $nrows</p>");
...
```

Injecção de SQL

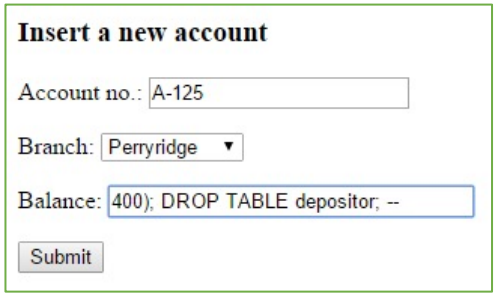
```
...
$account_number = $_REQUEST['account_number'];
$branch_name = $_REQUEST['branch_name'];
$balance = $_REQUEST['balance'];

$connection = new PDO($dsn, $user, $pass);

$sql = "INSERT INTO account VALUES
('$account_number', '$branch_name', $balance)";

$nrows = $connection->exec($sql);

echo("<p>Rows inserted: $nrows</p>");
...
```




```
INSERT INTO account VALUES ('A-215', 'Perryridge',
400)
```

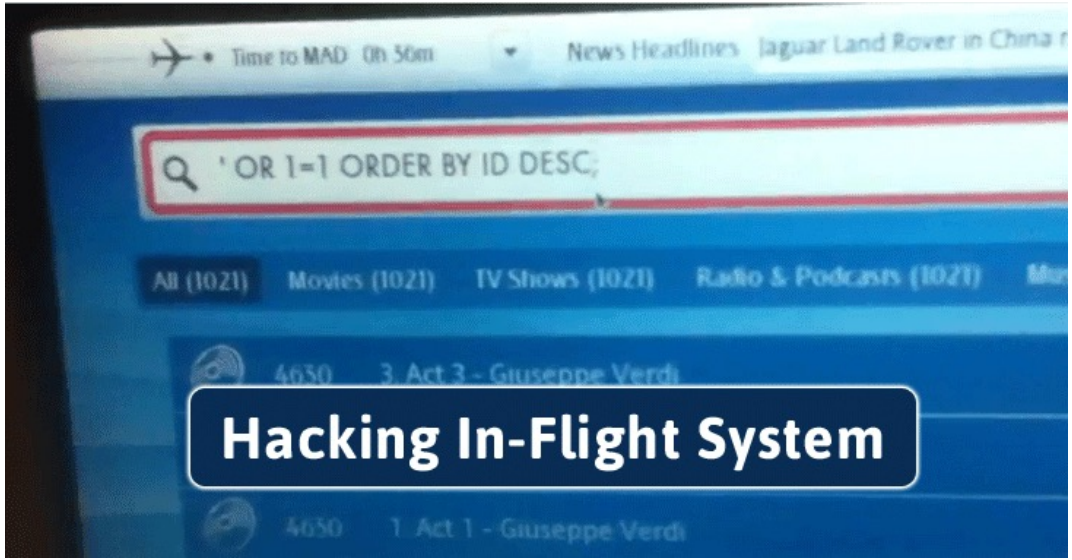
Injecção de SQL

```
...
$account_number = $_REQUEST['account_number'];
$branch_name = $_REQUEST['branch_name'];
$balance = $_REQUEST['balance'];

$connection = new PDO($dsn, $user, $pass);

$sql = "INSERT INTO account VALUES
('$account_number', '$branch_name', $balance)";

$nrows = $connection->exec($sql);

echo("<p>Rows inserted: $nrows</p>");
...
```




```
INSERT INTO account VALUES ('A-125', 'Perryridge', 400); DROP TABLE depositor; --)
```

Hacking in-flight system



https://thehackernews.com/2016/12/hacking-in-flight-system.html

Como impeder injecção de SQL

Melhor pática: usar SEMPRE prepared statements

```
...
$account_number = $_REQUEST['account_number'];
$branch_name = $_REQUEST['branch_name'];
$balance = $_REQUEST['balance'];

$stmt = $connection->prepare(
      "INSERT INTO account VALUES (:account_number, :branch_name, :balance)");

$stmt->bindParam(':account_number', $account_number);
$stmt->bindParam(':branch_name', $branch_name);
$stmt->bindParam(':balance', $balance);

$stmt->execute();
...
```

Como impeder injecção de SQL

Melhor pática: usar SEMPRE prepared statements

```
...
$account_number = $_REQUEST['account_number'];

$stmt = $connection->prepare(
     "SELECT balance FROM account WHERE account_number=:account_number");

$stmt->bindParam(':account_number', $account_number);

$stmt->execute();

foreach($stmt as $row)
{
        echo($row['balance']);
}
...
```

Psycopg

Psycopg can automatically convert Python objects to and from SQL literals: using this feature your code will be more robust and reliable. We must stress this point:

Warning: Never, **never**, **NEVER** use Python string concatenation (+) or string parameters interpolation (%) to pass variables to a SQL query string. Not even at gunpoint.

Psycopg – Injecção SQL

The correct way to pass variables in a SQL command is using the **second argument** of the execute() method:

```
>>> SQL = "INSERT INTO authors (name) VALUES (%s);" # Note: no quotes
>>> data = ("O'Reilly", )
>>> cur.execute(SQL, data) # Note: no % operator
```

Processamento de Form HTML (versão à prova de Injecção de SQL)

```
query =  "UPDATE account SET balance=%s WHERE   account_number = %s”
data = (request.form["balance"], request.form["account_number"])
cursor.execute(query, data)
```

Recomendações Básicas de SEGURANÇA

- Forms com Method=POST (não GET)
- Prepared Statements (evitar “SQL Injection”)
- SSL
- Certificados

TÉCNICO LISBOA