Projeto 2

Universidade de Aveiro

Luís Couto, Francisco Silva, Joaquim Andrade

Projeto 2

DETI

Universidade de Aveiro

Luís Couto, Francisco Silva, Joaquim Andrade
(89078), (93400), (93432)
labi2019-p2-g19

14/06/2019

Resumo

Seja no facebook, seja no instagram, ou até nas nossas próprias câmaras de telemóvel, é possível ver que existe um reconhecimento facial automático. O reconhecimento não só facial mas também como de outros objetos em imagens, vídeos e até mesmo na vida real é um tema muito atual e alvo de muita pesquisa e foco científico.

Este trabalho vai ao encontro desse tema, pois tem como objetivo desenvolver uma Aplicação Web, que fazendo uso de um site capaz de reconhecer objetos em imagems[1], permita tratar essas imagens, guardá-las numa base de dados e fazer várias pesquisas com elas. Neste relatório falaremos da Aplicação Web que desenvolvemos, do código que fizemos para a Interface Web, da maneira que implementámos o processador de imagens e o tipo de base de dados que criámos.

Iremos falar de cada módulo desenvolvido, assim como apresentaremos imagens que demonstram o funcionamento correto do código desenvolvido. No final iremos analisar os resultados e tirar conclusões.

Conteúdo

Lista de Figuras

Acrónimos

DETI - Departamento de Electrónica, Telecomunicações e Informática

LI - Laboratórios de Informática

Capítulo 1

Introdução

1.1 Motivação e Objetivos

No âmbito da avaliação da cadeira de LI, foi-nos proposto efetuar um projeto que aplicava todos os conhecimentos adquiridos ao longo dos dois semestres da cadeira (*Python*, HTML, CSS, JavaScript, entre outros) e que tinha como objetivo principal criar uma Aplicação Web.

O que era pedido desta Aplicação Web era que fosse possivel carregar imagens para o sistema, que depois iriam ser reencaminhadas para o site fornecido no guião[1]. Este site iria devolver um ficheiro JSON com os diversos objetos presente na imagem, incluindo as coordenadas necessárias para recortar cada um dos objetos e a confiança. Este ficheiro iria depois ser usado pelo processador para fazer o respetivo tratamento de imagem. Estas imagens e objetos iriam depois ficar guardados numa base de dados, sendo possível ao utilizador fazer pesquisas sobre eles de várias maneiras.

1.2 Conteúdo disponibilizado

Quanto ao trabalho, existem 2 pastas, cada uma com o seguinte conteúdo:

1. codigo
    (a) Interface (ficheiros necessário para o website)

(b) Original (imagens originais)

(c) Objects (objetos recortados das imagens originais)

(d) images.db (base de dados)

(e) processador.py (ficheiro com as funções para tratar as imagens)

(f) main.py (ficheiro CherryPy que é a aplicação)

2. relatorio

(a) TrabalhoP2.pdf

(b) Todas as imagens usadas

(c) Todos os ficheiros fonte necessários

Quando a este relatório, está dividido em 6 capítulos. Depois desta introdução, no Capítulo 2 falamos da implementação da Interface Web e como navegar pelo website. No Capítulo 3 é esclarecido o algoritmo do processador de imagens, sendo apresentada uma explicação detalhada e imagens dos resultados.
No Capítulo 4 é feita uma breve referência à base de dados criada, no Capítulo 5 é falado na Aplicação Web e sendo que por fim, no Capítulo 6, são tiradas conclusões.

Capítulo 2

Interface Web

2.1 Estrutura da Interface

Para a implementação da Interface Web, foi usado um *template bootstrap*[2] como base fundamental, no entanto foram feitas alterações a nível de código, de estrutura e de *display* do website.

2.2 Página Inicial

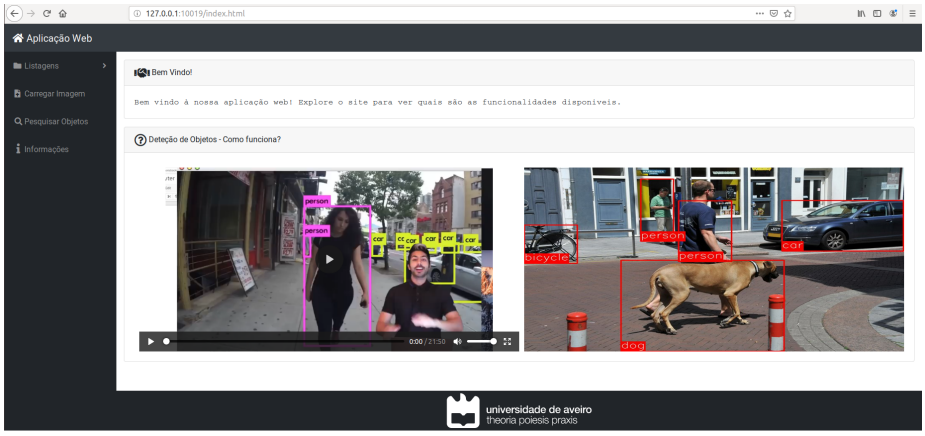


Figura 2.1: index.html

Esta é a página inicial do website. O utilizador pode ver um vídeo informativo sobre a deteção de objetos ou então pode navegar para outras páginas usando a *sidebar*.

2.3 Listar Objetos

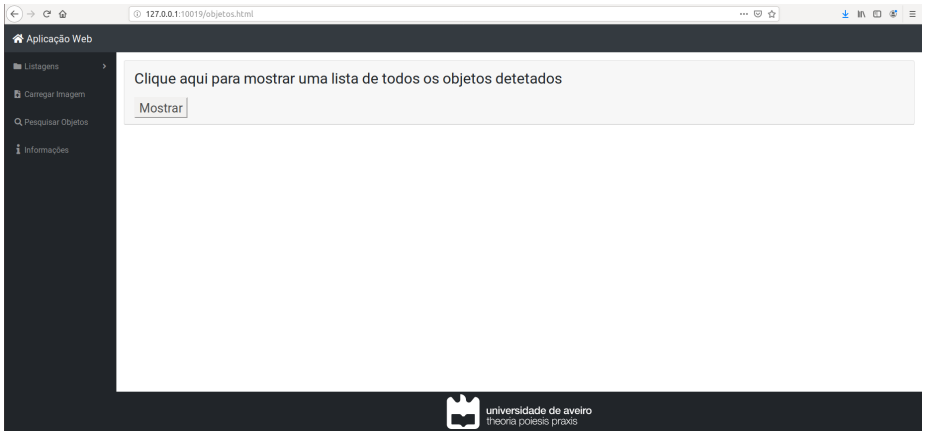


Figura 2.2: objetos.html

Nesta página, o utilizador pode listar todos os objetos que já foram detetados pressionando o botão indicado.

2.4 Listar Imagens

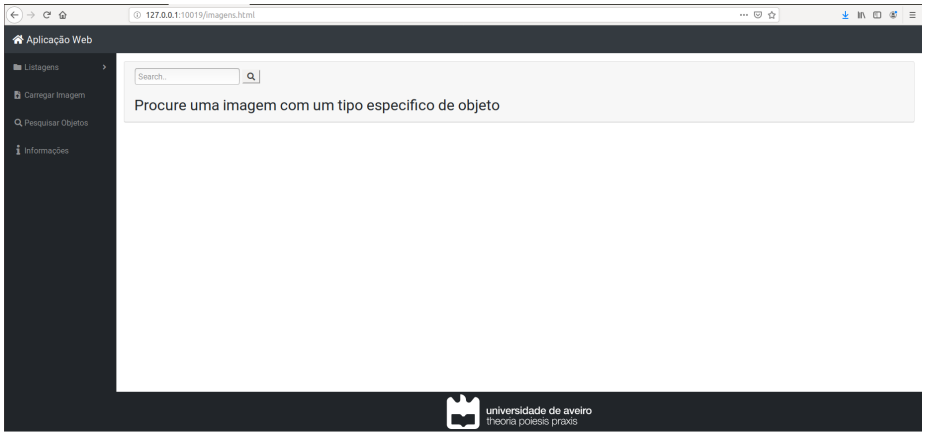


Figura 2.3: imagens.html

Nesta página, o utilizador pode procurar por um determinado objeto (car, person,...). O website vai então mostrar todas imagens

que contenham esse objeto, com um limiar de confiança de 50.

2.5 Carregar Imagem

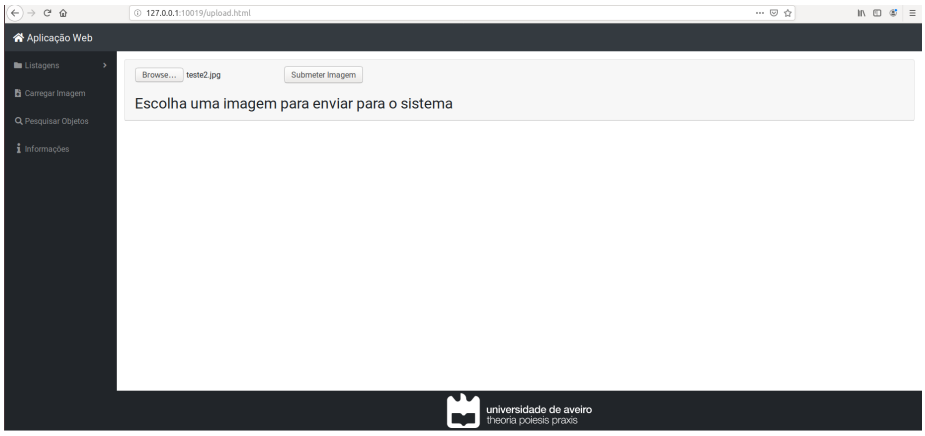


Figura 2.4: upload.html

Nesta página, o utilizador pode enviar qualquer imagem para o sistema. Esta imagem vai depois ser tratada segundo as especificações pedidas e guardada na base de dados, assim como qualquer objeto recortado a partir desta.

2.6 Pesquisar Objetos

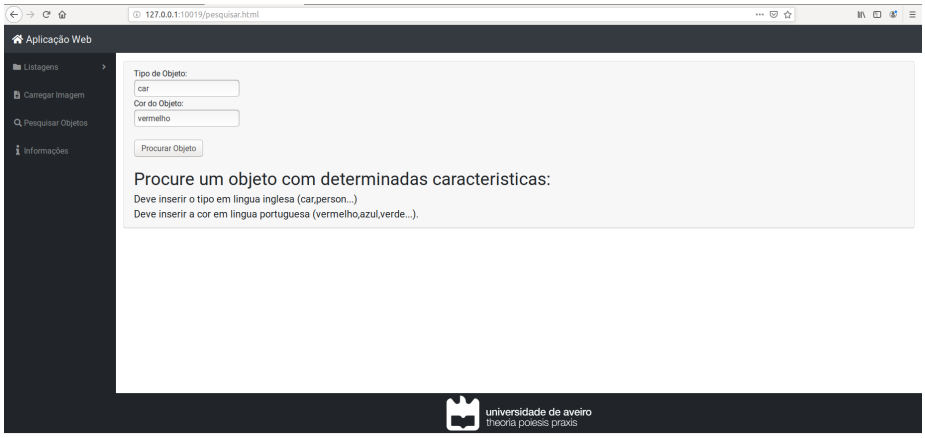


Figura 2.5: pesquisar.html

Nesta página, o utilizador pode procurar por um determinado objeto (car, person,...) com uma determinada cor (vermelho,verde,azul,..). Devido à maneira como a nossa base de dados está feita, o objeto tem de ser un input em inglês e a cor um input em português.

2.7 Informações

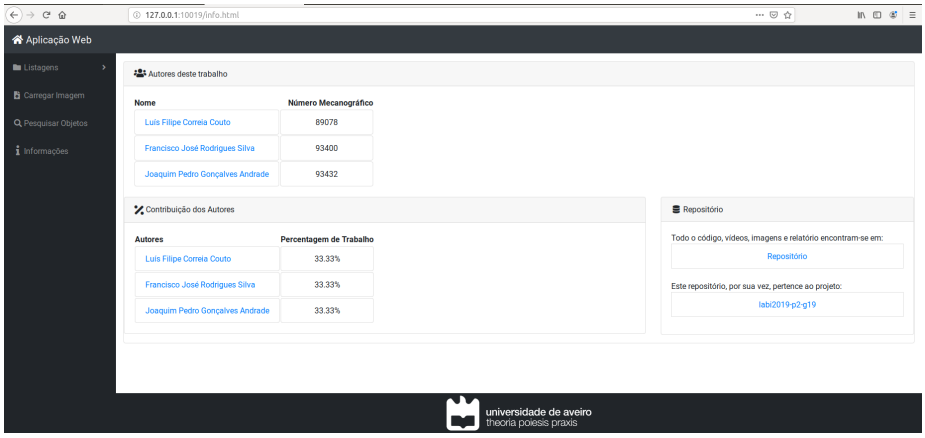


Figura 2.6: info.html

Nesta página, o utilizador pode ler informação sobre os autores, a sua contribuição ou o repositório onde o código do projeto se encontra.

Capítulo 3

Processador de Imagens

Neste capítulo vai ser explicada a implementação do processador de imagens, que se encontra implementado no ficheiro processador.py. Devido a extensão do código não vamos por imagens dele neste relatório, no entanto vamos fazer uma descrição detalhada e mostrar resultados ao fim de dar *run* a este programa usando uma imagem.

3.1 Implementação

O processador de imagens é um programa em *python* que recebe uma imagem enviada e efetua várias operações com ela.

Em primeiro lugar, fazendo uso do módulo "requests", o programa envia a imagem para o URL disponibilizado no guião[1] e guarda numa variável o ficheiro json que é devolvido.

O ficheiro json contém informações sobre os objetos identificados pelo serviço, como a class (tipo de objeto), coordenadas (x, y, x1, y1) que representam uma caixa de identificação dos objetos que foram encontrados e identificados e um valor de confiança que é a confiança que o serviço tem de que identificou um objeto e a sua class corretamente.

Em segundo lugar o programa utiliza as coordenadas devolvidas pelo serviço e cria novas imagens com um recorte das imagens originais, um para cada objeto encontrado. A função **creatbox** guarda

os valores da class, confiança e guarda num *tuple* as coordenadas, representando uma caixa que vai ser usada para o recorte das imagens e devolve todas estas informações.

Com a função **crop** do módulo Image que recebe como argumento um *tuple* com as coordenadas, é feito o recorte dos objetos encontrados na imagem original. As imagens geradas dos objetos identificados são então guardadas num diretório à parte (pasta **Objects)**, tal como as imagens originais(pasta **Original**.
Numa terceira fase o programa analisa cada uma das imagens que representam os objetos encontrados e tenta classificar o objeto com uma cor. Nesta classificação considerámos as cores: vermelho, laranja, amarelo, verde, azul, violeta, preto, branco e cinzento.

Para a implementação do problema das cores considerámos cada pixel da imagem como uma combinação das cores vermelho, verde e azul. Podemos saber os valores de RGB usados em cada pixel da imagem e usámos isso para classificar cada um dos pixéis com uma cor.
Considerámos limites de representação para cada um dos canais RGB. Num pixel uma cor com valor menor que o limite inferior(85) é classificado como baixo, com valor maior que o limite superior(170) é classificado como alto e caso esteja dentro dos limites é classificado como médio.

Destas combinações de cores e classificações tivemos de considerar $(3^3) = 27$ possibilidades e atribuir uma cor a cada uma. Classificado cada pixel, basta fazer a contagem, e a cor associada ao maior número de pixéis é considerada a cor dominante na imagem. Face a alguns problemas que encontrámos na implementação desta solução decidimos criar um filtro que funciona como uma variável booleana e que limita a cor preta.

Face aos limites que considerámos a cor preta era considerada dominante na maioria das imagens visto que algumas cores próximas do preto e presentes em várias imagens eram classificadas como preto adulterando de certa forma os resultados.

Calculada a cor temos todas as informações que precisamos e o programa procede.

3.2 Resultados

Ao correrm os este código com uma imagem original que contenha vários objetos, obtemos o seguinte resultado.

Figura 3.1: Imagem para testar processador

```
luis@luis-VirtualBox:~/labi2019-p2-g19$ python3 processador.py teste.jpg
Class: car
Confianca: 0.999
x: 641
y: 416
x1: 1225
y1:593
Cor dominante: amarelo

Class: car
Confianca: 0.999
x: 608
y: 134
x1: 1216
y1:285
Cor dominante: verde

Class: car
Confianca: 0.997
x: 32
y: 395
x1: 608
y1:614
Cor dominante: azul

Class: car
Confianca: 0.978
x: 13
y: 84
x1: 613
y1:307
Cor dominante: vermelho

luis@luis-VirtualBox:~/labi2019-p2-g19$
```

Figura 3.2: Resultado impresso na consola

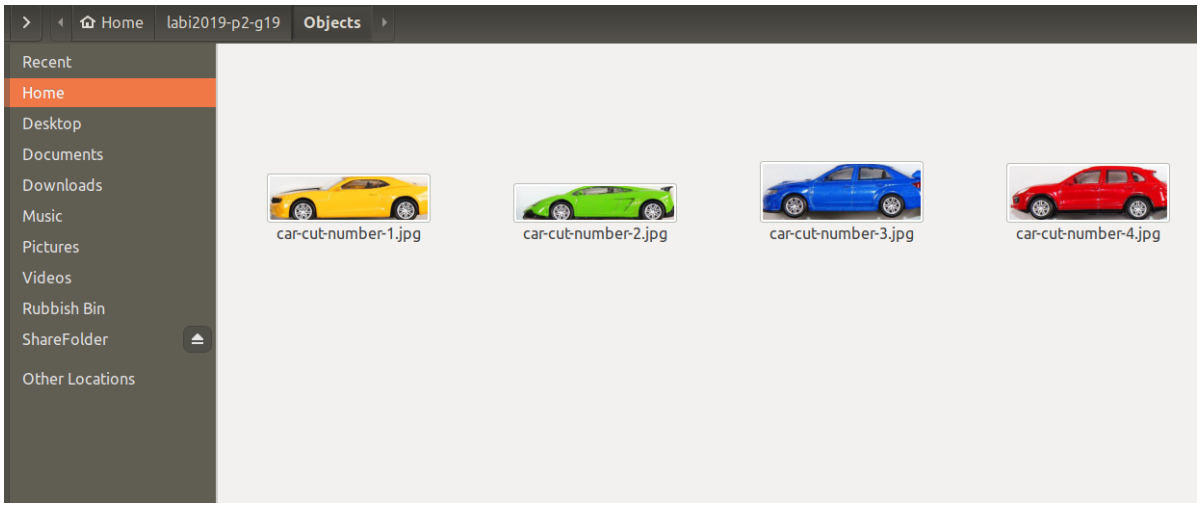


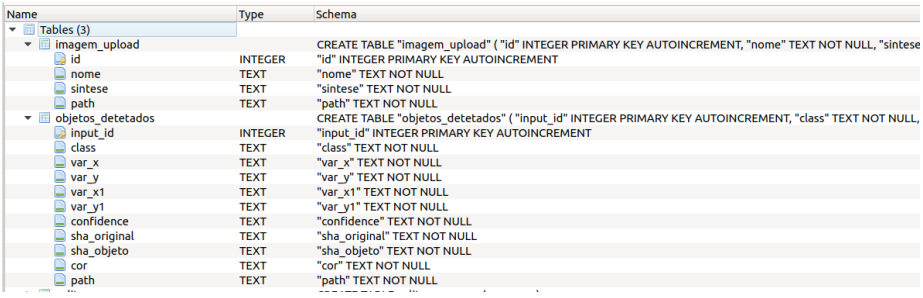
Figura 3.3: Objetos recortados pelo processador a partir da imagem original, guardados no respetivo diretório

Capítulo 4

Base de Dados

A primeira tabela é referente às imagens originais. Contém 4 campos que são o **id** que incrementa automáticamente à medida que vão sendo introduzido dados, o **nome** da imagem, a **síntese** que é a síntese (SHA-512) da imagem original e o **path** que contém o caminho do diretório onde a imagem está armazenada.

A segunda tabela é referente às imagens que contém objetos recortados a partir das imagens originais. Contém 11 campos que são o **input_id** que incrementa automáticamente à medida que vão sendo introduzido dados, a **class** que é o tipo de objeto detetado, a **var_x**, **var_y**, **var_x1** e **var_y1** que são as coordenadas de uma caixa na imagem original onde se encontra o objeto, a **confidence** que é a confiança, o **sha_original** que é a síntese (SHA-512) da imagem original, o **sha_objeto** que é a sintese (SHA-512) da imagem que contém o objeto recortado, a **cor** do objeto e o **path** que contém o caminho do diretório onde o objeto está armazenado.



Name	Type	Schema
Tables (3)		
imagem_upload		CREATE TABLE "imagem_upload" ("id" INTEGER PRIMARY KEY AUTOINCREMENT, "nome" TEXT NOT NULL, "sintese
id	INTEGER	"id" INTEGER PRIMARY KEY AUTOINCREMENT
nome	TEXT	"nome" TEXT NOT NULL
sintese	TEXT	"sintese" TEXT NOT NULL
path	TEXT	"path" TEXT NOT NULL
objetos_detetados		CREATE TABLE "objetos_detetados" ("input_id" INTEGER PRIMARY KEY AUTOINCREMENT, "class" TEXT NOT NULL,
input_id	INTEGER	"input_id" INTEGER PRIMARY KEY AUTOINCREMENT
class	TEXT	"class" TEXT NOT NULL
var_x	TEXT	"var_x" TEXT NOT NULL
var_y	TEXT	"var_y" TEXT NOT NULL
var_x1	TEXT	"var_x1" TEXT NOT NULL
var_y1	TEXT	"var_y1" TEXT NOT NULL
confidence	TEXT	"confidence" TEXT NOT NULL
sha_original	TEXT	"sha_original" TEXT NOT NULL
sha_objeto	TEXT	"sha_objeto" TEXT NOT NULL
cor	TEXT	"cor" TEXT NOT NULL
path	TEXT	"path" TEXT NOT NULL

Figura 4.1: Base de Dados criada

Capítulo 5

Aplicação Web

A aplicação principal deste trabalho é o ficheiro **main.py**. Para iniciar a aplicação, o utilizador tem de correr na consola o ficheiro **main.py** e aceder ao endereço que aparece indicado na consola.

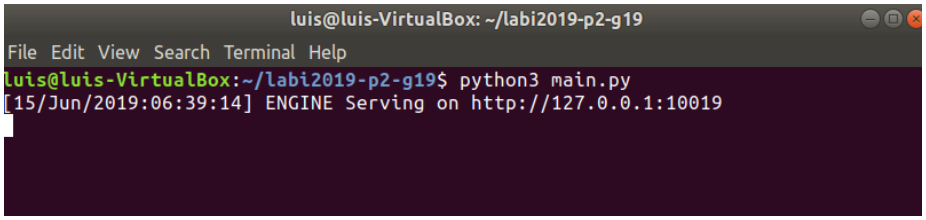


Figura 5.1: Iniciar a Aplicação

Para formarmos a aplicação web usamos então o módulo cherrypy em python. O cherrypy é um servidor aplicacional. Temos então a declaração de uma classe **Root**. Esta classe é composta por um método chamado **index**. **@cherrypy.expose** determina que o método **index** deverá ser exposto ao cliente Web. O método **index** devolve um html.

Após esse método temos o método **listnames** que se conecta à base de dados e devolve um objeto json com uma lista de todos os objetos já detectados. Temos também o método **listdetected** que devolve um objeto json com as sínteses das imagens dos objetos cortados, juntamente com a imagem original e a confiança. Neste método foi aplicada a identação do objeto json de modo a que o

resultado final seja de acordo com o pedido no guião. Após este método temos dois métodos semelhantes, no entanto com filtros. Estes filtros foram aplicados através da alteração do comando execute da base de dados, que nos permitiu facilmente alcançá-los.

O método **put** por sua vez serve para inicializar uma nova imagem. Este processo leva-nos diretamente para o nosso processador de imagens. Este processador tem também vários métodos dos quais falaremos mais à frente. Este método envia no final, toda a informação requerida para a base de dados. O método **identifier**, mais uma vez conecta-se à base de dados e procura, novamente, usando uma particularidade do execute, o caminho de uma imagem através da sua síntese.

Capítulo 6

Conclusão:

Este foi um trabalho desafiante que pôs à prova todos os conhecimentos adquiridos ao longo do ano. Apesar de termos conseguidos fazer os blocos inviduais do trabalho e eles funcionarem, o produto final não é o esperado. Devido à menor dimensão do nosso grupo e consequente falta de tempo, não foi possível escrever código JavaScript, e sem este código não foi possível por a aplicação a funcionar como desejado, pois a Interface Web não comunica com o resto dos blocos.

6.1 Contribuição dos autores

O aluno Luís Couto ficou encarregue da Interface Web e base de dados.
O aluno Francisco Silva ficou encarregue do processador de imagens.
O aluno Joaquim Andrade ficou encarregue da Aplicação Web.

O relatório foi escrito pelo aluno Luís Couto com a assistência dos outros dois membros.

Tendo isto em conta, a contribuição de cada aluno foi de 33.33 %.

A área utilizada no XCOA é https://xcoa.av.it.pt/labi2019-p2-g19.
A projeto no Code.UA é https://code.ua.pt/projects/labi2019-p2-g19.

Bibliografia

[1] J. Barraca. (2019), URL: http://image-dnn-sgh-jpbarraca.ws.atnog.av.it.pt/.

[2] B. Digital. (2019), URL: https://startbootstrap.com/templates/sb-admin/.