Pesquisa e desenvolvimento de funcionalidade de reconhecimento de imagens para aplicações mobile

Mestrado em Engenharia Informática – Computação Móvel

Rodrigo Ramos Sá Pessoa

Leiria, Novembro de 2020

Pesquisa e desenvolvimento de funcionalidade de reconhecimento de imagens para aplicações mobile

Mestrado em Engenharia Informática – Computação Móvel

Rodrigo Ramos Sá Pessoa

Trabalho de Projeto realizado sob a orientação da Professora Doutora
Sónia Maria Almeida da Luz

Leiria, Novembro de 2020

Originalidade e Direitos de Autor

O presente relatório de projeto é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionado o Autor e feita referência ao ciclo de estudos no âmbito do qual o mesmo foi realizado, a saber, Curso de Mestrado em Engenharia Informática – Computação Móvel, no ano letivo 2019/2020, da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos.

Dedicatória

Aos meus pais Brigite e António por acreditarem sempre em mim. Devo o meu sucesso a vocês, que me acompanharem desde o início até ao fim desta etapa da minha vida. Obrigado por serem o melhor que tenho.

Agradecimentos

Começo por agradecer ao António Lopes por ter sido um excelente mentor no decorrer deste projeto. Não só garantiu que a minha integração na md3 seria feita da melhor forma, como também me equipou com ferramentas e ensinamentos que me ajudaram imensamente neste projeto. A ti devo grande parte do meu sucesso, e do meu crescimento como profissional. Um enorme obrigado por toda a disponibilidade, confiança e amizade ao longo de todo este tempo.

À minha orientadora Professora Sónia Luz pelo excelente acompanhamento que me proporcionou. Sou-lhe imensamente grato por toda a disponibilidade e apoio nesta etapa do meu mestrado, a si devo grande parte do meu sucesso e tudo se deve à sua excelente orientação. Muito obrigado por tudo.

A toda a equipa da *md3* envolvida no projeto da *APM* por me receberem de braços abertos com todo o seu companheirismo e espírito de equipa. Um grande obrigado.

A todos os meus amigos, com quem posso sempre contar, obrigado por tudo.

Por fim, um importante agradecimento à minha família por acreditarem em mim e me apoiarem sempre. Em especial. à minha mãe e pai por estarem sempre do meu lado e por me darem a força para lutar pelos meus sonhos. Devo tudo a vocês e sou-vos eternamente grato. Do fundo do coração, obrigado.

Resumo

Pretende-se com o projeto documentado neste documento a criação de uma prova de conceito capaz de extrair a informação contida numa tabela de presenças. Esta solução enquadra-se no contexto de uma empresa francesa com o nome *APM*, cuja missão inclui a facilitação de eventos para crescimento profissional. No final de cada um destes eventos, haverá uma folha de presenças que necessita ser assinada por todos os participantes e, posteriormente, registada manualmente na plataforma online da empresa. Por este processo ser atualmente manual, proporciona uma experiência desagradável aos aderentes da *APM.* Com este projeto foi possível o desenvolvimento de um *package python* que recorre a técnicas de visão computacional para interpretar o conteúdo de uma dada imagem da folha de presenças, identificando os participantes que a assinaram. Foram utilizadas técnicas já conhecidas como *OCR* para este fim, contudo, surgiu também a necessidade pela criação de técnica originais, como a criação do *Margin Patrol* para a deteção do esqueleto da tabela. De modo a assegurar o sucesso deste trabalho, foi necessária a integração com a equipa da *md3*, responsável pelos projetos da *APM*. Parte deste processo exigiu a adoção da metodologia em prática, o *Scrum*, e o estudo das tecnologias utilizadas internamente. A aplicação *mobile* a ser integrada com este projeto foi desenvolvida com recurso à *framework Ionic*, destinada à criação de aplicações multiplataforma, o que proporcionou o estudo e a realização de testes relacionados com aplicações híbridas, principalmente no âmbito da sua capacidade de realizar a captura da imagem da tabela.

Palavras-chave: Visão Computacional, OCR, OpenCV, Ionic, Scrum, Margin Patrol

Abstract

The goal of the project described throughout this document is the development of a proof-of-concept capable of extracting the information contained in a presence sheet. This solution fits in the context of a French company named *APM*, whose mission includes the facilitation of professional growth events. At the end of these events, there is a presence sheet that has to be signed by every participant and, after that, registered on the company's online platform. Because this process is manual, it provides an unpleasant experience to *APM's* subscribers. With this project, it was possible to develop a python package that uses computer vision techniques to interpret the contents of a given image containing a presence sheet. Know techniques such as *OCR* were used, however, there was also the need to come with original solutions, as was the case for identifying the skeleton of the table, which prompted the creation of the *Margin Patrol* solution. In order to guarantee the success of this project, it was necessary to integrate the team at *md3* responsible for *APM*'s projects. Part of this process required the adoption of the methodology in use, *Scrum*, and the study of the technologies used in-house. The mobile app to be integrated with this project was developed using the Ionic framework, used for making cross-platform apps. This motivated the study and testing of hybrid applications with a focus on the process of capturing the image of the presence sheet.

Keywords: Computer Vision, OCR, OpenCV, Ionic, Scrum, Margin Patrol

Índice

Lista de Figuras

Lista de Código

Lista de Tabelas

Lista de Siglas e Acrónimos

API	Application Programming Interface
APM	Association Progès du Management
APP	Aplicação
CV	Computer Vision
ESTG	Escola Superior de Tecnologia e Gestão
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
MP	Margin Patrol
MVVM	Model View ViewModel
NR	NameReader
OCR	Optical Character Recognition
POC	Proof of concept
PWA	Progressive Web App
REST	Representational State Transfer
SO	Sistema Operativo

1. Introdução

Nos dias de hoje, é possível observar a evolução das tecnologias móveis e o aumento da sua presença no nosso dia a dia. Considerando este domínio, o dispositivo que mais se destaca em termos de popularidade é o *smartphone*, que veio a substituir o telemóvel inovando drasticamente nas suas capacidades e consequentemente revolucionando todo o sector tecnológico. Por definição, um dispositivo móvel é aquele que possui a capacidade de realizar tarefas de computação de forma portátil, sendo que existem muitos mais aparelhos, para além do *smartphone*, que se caracterizam como móveis. Os aspetos mais atrativos deste tipo de dispositivo são a sua capacidade de aceder à internet, a possibilidade de instalar aplicações e de usufruir dos vários sensores que vêm integrados nos dispositivos.

As capacidades destes dispositivos em conjunto com a sua popularidade justificam o valor que este meio consegue gerar para um negócio. Destaca-se a forma como uma aplicação (*app)* permite criar uma relação bastante próxima entre empresa e cliente, devido a conceitos como *gamification* e a ferramentas como notificações *PUSH*. Outra forma como estas *apps* geram valor para um negócio é através da sua possibilidade de integrar funcionalidades complexas recorrendo a sensores do próprio dispositivo. Deste modo, torna-se possível para estas *apps* a integração de técnicas como as de computação aplicada ao contexto e de visão computacional, que quando usadas corretamente tem a capacidade de facilitar trabalho.

A área da visão computacional tem sido bastante popularizada, surgindo cada vez mais projetos que aplicam os seus princípios. O objetivo desta tecnologia está em equipar máquinas com a capacidade de interpretar informação do mundo real, convertendo-a em dados legíveis e trabalháveis. Uma das áreas mais estudadas deste domínio é a de sistemas óticos de caracteres, cuja função é a de detetar texto presente numa imagem. Encontra-se presente em sistemas como os de digitalização, permitindo a conversão de documentos impressos em ficheiros com texto editável.

A *Association Progrès du Management* (*APM)* [1] é um exemplo de uma empresa que decidiu investir numa aplicação móvel híbrida como complemento à sua plataforma de criação e gestão de eventos, facilitando o modo como membros da *APM* interagem com o

complexo ecossistema da empresa. Deste modo acrescenta-se valor a funcionalidades pré-existentes através do uso da computação portátil da aplicação que permite o acesso a sensores como a câmara e o *Global Positioning System* (GPS).

Pretende-se com este projeto o desenvolvimento de uma funcionalidade na aplicação móvel que permita facilitar o processo de registo de assinaturas em eventos, sendo que tal deve ser feito recorrendo a técnicas de visão computacional. Para alcançar este objetivo, será realizado um processo de investigação no qual serão estudadas as tecnologias e ferramentas que podem ser úteis na resolução este problema. Para este projeto serão usadas as *frameworks Ionic* [2] e *Django* [3], pois a *app* em causa é multiplataforma e a *Application Programming Interface* (*API)* com que este interage encontra-se escrita em *Django*. A metodologia a adotar para o processo de desenvolvimento será o *Scrum*, permitindo de forma ágil que toda a equipa esteja sempre atualizada do estado do projeto e possa trabalhar em cooperação.

1.1. Enquadramento

Este projeto enquadra-se no contexto do segundo ano do mestrado em Engenharia Informática - Computação Móvel da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, onde os alunos têm a possibilidade de escolher o tipo de trabalho que pretendem realizar, sendo este relatório referente a um trabalho de projeto. O tema deste trabalho foi proposto pela *md3* [4], empresa de *outsourcing* sediada em Leiria e cliente de longa data da *APM*.

1.2. Problema

Atualmente, eventos de caracter presencial exigem um trabalhoso processo de gestão. Internamente na *APM* existem grupos de membros que a eles tem associado um organizador, responsável pelos eventos desse mesmo grupo. Um organizador terá de solicitar formações a um *expert*, aquele que irá aos eventos realizar a formação. Após criado um evento, os membros do grupo terão que confirmar a sua disponibilidade, sendo esta informação utilizada para gerar um documento onde serão registadas as assinaturas dos participantes no evento. Na data do evento, os participantes terão que colocar a sua assinatura nesse mesmo documento, indicando se estiveram presentes no período da manhã ou da tarde no evento.

Após todo este processo, o organizador terá que registar manualmente os membros presentes no evento com base na folha de presenças que obteve do evento.

Este processo de registo de assinaturas implica uma perda de tempo aos organizadores que coloca obrigatoriamente os membros do clube em espera, não contribuindo para a qualidade da experiência de todos os elementos envolvidos no processo. Como tal, pretende-se a criação de uma alternativa a este processo, recorrendo a técnicas de visão computacional como o *optical character recognition* (*OCR*) para conseguir registar na plataforma online as presenças em eventos através de uma fotografia da sua respetiva folha de presenças.

1.3. Objetivos e Motivação

Pretende-se com este projeto a atualização e evolução da aplicação móvel multiplataforma *DirectAPM*, propriedade da empresa *APM*. Entre os vários requisitos estabelecidos, destaca-se como principal foco deste projeto o desenvolvimento de um *proof of concept* (*POC*) baseado em visão computacional, que permita a transformação do atual processo de registo de assinaturas.

A principal motivação para a realização deste projeto é a vertente investigativa que este mesmo proporciona. A área da visão computacional foi algo estudado durante o mestrado, possibilitando a compreensão do funcionamento de algumas tecnologias deste domínio e o conhecimento das possibilidades e atenção que o tem acompanhado. Isto resultou num interesse pelo tema que, juntamente com a oportunidade de aprender a trabalhar com tecnologias híbridas de desenvolvimento móvel foram a razão pela escolha deste projeto.

1.4. Estrutura do documento

O presente relatório tem início com esta secção introdutória ao projeto, seguindo-se o capítulo 2 com a descrição do estado da arte, onde são expostos alguns conceitos, tecnologias e trabalhos importantes para a contextualização e compreensão deste projeto. No capítulo seguinte, o capítulo 3, são abordados o planeamento adotado e a metodologia posta em prática durante o processo de desenvolvimento. No capítulo 4 é apresentada a descrição dos requisitos, casos de uso e ferramentas utilizadas, de modo a expor as características pretendidas para o *POC*, assim como os artefactos que permitiram a sua concretização. A

descrição do processo de desenvolvimento, onde é detalhada a criação e o funcionamento da solução de deteção de presenças encontra-se no capítulo 5. No capítulo 6 encontram-se descritos alguns dos principais testes realizados ao longo deste projeto, assim como a explicação do seu papel para o melhoramento da solução final. Por fim, no capítulo 7, é apresentada a conclusão do trabalho realizado acompanhada do seu balanço qualitativo e uma descrição do trabalho futuro.

2. Estado da arte

Nesta secção serão descritos os principais conceitos necessários para a compreensão do projeto desenvolvido. Este capítulo tem início com a descrição do que é desenvolvimento *mobile* e quais os modelos de aplicações existentes, por serem importantes para a compreensão do modelo híbrido, central no desenvolvimento da aplicação trabalhada neste projeto. Segue-se uma explicação da área da visão computacional (*CV - Computer Vision*) onde é detalhando o processo de *OCR*, ambos domínios onde se insere a solução desenvolvida para reconhecimento de presenças. Será explicado o que são serviços *RESTful*, de modo a dar a entender a arquitetura adotada na *API*. Por fim, são explorados alguns exemplos de aplicações que, de alguma forma, têm um funcionamento semelhante ao pretendido com este projeto.

2.1. Desenvolvimento Mobile

Atualmente podemos constatar que tem aumentado significativamente a presença de dispositivos móveis no nosso dia a dia, de entre os quais, destacam-se os *smartphones*. Os números de utilizadores de smartphones têm aumentado exponencialmente ao longo dos anos, verificando-se em 2016 a existência de 2,5 biliões de utilizadores e estimando-se que este número tenha crescido até aos 3,5 biliões no ano de 2020 [5]. É, de facto, inegável a utilidade que este tipo de tecnologia é capaz de fornecer, possibilitando o acesso prático e portátil a funcionalidades características de computadores. Destacam-se como mais importantes as capacidades de acesso à internet, a posse de uma interface baseada em comandos *touch*, o acesso a vários sensores integrados no dispositivo e a possibilidade de instalar *software* sob a forma de aplicações móveis (*apps*).

É possível afirmar que o mercado das *apps* se encontra também em constante evolução, tendo-se verificado um crescimento global do número de downloads para os 204 biliões no ano de 2019 [6]. A grande maioria do tempo gasto em *smartphones* é para o uso de *apps* [7]. Devido à sua popularidade, hoje em dia, a escolha de desenvolver uma *app* é bastante aliciante para qualquer empresa. Não só é um meio bastante rentável, estimando-se que os seus consumidores gastaram mundialmente 50,1 biliões de dólares na primeira metade de 2020 [8], como também, por natureza, traz vantagens aos próprios negócios. As empresas

podem utilizar *apps* para aumentar o interesse na sua marca, pois, ao contrário de outros meios de publicidade, os consumidores adotam *apps* como sendo úteis, resultando num meio de divulgação da marca que constrói confiança com os consumidores [9]. É importante, portanto, reconhecer o seu potencial de comprometimento com utilizadores e a forma como contribuem para o aumento de vendas [10].

Uma das primeiras decisões a ser tomada antes do desenvolvimento de uma *app* é a escolha do sistema operativo (*SO*) a que esta se destina, algo que irá determinar o tipo de desenvolvimento necessário. Atualmente os *SO* mais populares são o *Android* da *Google* e *iOS* da *Apple* [11], o que faz com que sejam a primeira escolha de negócios que pretendem alcançar o maior número de utilizadores.

Distinguem-se vários modelos de *app*, representados na Figura 1, cada um com as suas vantagens e desvantagens. Um dos aspetos importantes de cada modelo é o alcance que este possibilita ao nível de *SO*, tendo para este projeto sido adotado o modelo híbrido que abrange tanto *iOS* como *Android*. Contudo, não existe um tipo que seja universalmente adotado, porque a escolha certa irá depender dos objetivos, recursos e necessidades de cada negócio. Em seguida, serão apresentados os modelos de aplicação predominantes, assim como as suas principais vantagens e desvantagens.

Figura 1 - Tipos de modelo de app

2.1.1. Aplicações Nativas

Aplicações nativas são contruídas para um *SO* específico, destinando-se a uma única plataforma como *Android* ou *iOS*. O design destas plataformas é caracteristicamente diferente, sendo necessária a utilização das devidas linguagens e ambientes de

desenvolvimento. Apesar de representarem estratégia atualmente dominante [12] implicam complicações quando se pretende a disponibilização de uma *app* em várias plataformas.

Entre os modelos que serão discutidos nesta secção, o nativo é o que proporciona o melhor desempenho e experiência para o utilizador, tirando o melhor proveito das qualidades do *SO* e dos *softwares* nele presentes [13]. Contudo, como mencionado anteriormente, a desvantagem deste modelo é quando se pretende a disponibilização da *app* em múltiplas plataformas. Para alcançar este objetivo será necessário o desenvolvimento e manutenção de duas *apps* distintas, como consequência das incompatibilidades entre os *SO*, o que poderá exigir mais recursos ao nível de equipa e de tempo necessário para os projetos. Por estes motivos, considera-se que o modelo nativo é financeiramente exigente, especialmente para pequenas empresas [12].

2.1.2. Aplicações Web

Aplicações *web* são acessíveis através de *browsers* e desenvolvidas com recurso a tecnologias web como *HTML*, *CSS* e *Javascript* [14]. Por este motivo são consideradas multiplataforma, contornando o problema das aplicações nativas. Comparativamente com o modelo anterior, aplicações *web* conseguem ter um custo de desenvolvimento mais baixo, recorrendo a uma única base de código. São também mais fáceis de manter.

Existem algumas desvantagens associadas a este modelo, destacando-se como sendo a mais limitante a dependência dos *browsers* onde correm. Por existir esta dependência, então está implícito que as *apps* terão os mesmos problemas que o *browser*. Um exemplo de um destes problemas é o acesso aos componentes do dispositivo porque este encontra-se, atualmente, limitado [15]. Estas *apps* também não são acessíveis sem que haja ligação há internet. Por estes motivos, é ainda impossível ter o mesmo nível de qualidade de experiência comparativamente com as aplicações nativas.

2.1.3. PWAs

O modelo *Progressive Web App* (*PWA*) pode ser definido como uma transformação do modelo *web app.* Ao contrário das aplicações *web*, as *PWAs* não são dependentes de uma ligação à internet, têm a capacidade de ser instaladas, de receber notificações *PUSH* e de fornecer uma experiência semelhante à de uma *app* nativa [16]. Estas características são

tornadas possíveis devido à existência de um *script*, denominado de *service worker*, que funciona como uma camada entre o *browser* e o servidor [17]. Esta camada permite que o *browser* seja capaz de continuar a fazer pedidos ao *service worker*, mesmo sem acesso à internet. Da mesma forma, possibilita que *o service worker* comunique com o servidor mesmo que o utilizador não esteja com a *app* aberta, permitindo a concretização das ações que o utilizador realizou *offline* e a existência de funcionalidades como notificações *PUSH*.

Apesar de se apresentar como uma evolução do modelo *web app*, este tipo de aplicação continua a sofrer com restrições impostas pelos *browsers*, nomeadamente, o acesso limitado aos componentes dos dispositivos e *APIs* das plataformas [16]. Também é importante mencionar que existe falta de suporte a *service workers* por parte de alguns *browsers* [18]. Contudo, ao longo dos anos, têm-se verificado melhorias neste aspeto.

2.1.4. Aplicações Híbridas

O modelo híbrido existe como um meio termo entre aplicações nativas e aplicações *web*. Permite que, através de uma única base de código, seja possível criar aplicações multiplataforma, que forneçam uma experiência semelhante à do modelo nativo e com acesso ilimitado às capacidades dos dispositivos [19]. É por este motivo uma abordagem largamente adotada, sendo capaz de contornar grande parte dos problemas característicos dos modelos referidos anteriormente.

Existem várias *frameworks* destinadas à criação de *app*s híbridas, destacando-se como pertencendo às mais populares *React Native*[1], *Flutter*[2] e *Ionic* [20]. Todas estas ferramentas têm o objetivo comum de criar *app*s híbridas, contudo, a forma como atingem este objetivo é a variável que as distingue. Justamente por haver diferenças no modo como cada *framework* lida com o seu código, é possível fazer a distinção entre dois subgrupos dentro deste domínio. Existem, portanto, aplicações híbridas-nativas e aplicações híbridas-*web* [21]. *Frameworks* híbridas-nativas, como *React Native* e *Flutter*, fazem uma ponte entre o código produzido e os elementos das plataformas destino, resultando numa interface verdadeiramente nativa. Em contraste, *frameworks* híbridas-*web* como *Ionic* optam por fazer o encapsulamento do seu código, resultando essencialmente numa *web app* capaz de aceder

[1] https://reactnative.dev
[2] https://flutter.dev

aos recursos dos dispositivos [22]. Para este projeto foi utilizada a *framework Ionic*, detalhada na secção 4.4.2.

O desenvolvimento híbrido é ideal para empresas que pretendam ter a sua *app* compatível com múltiplas plataformas, diminuindo os recursos necessários para o seu desenvolvimento. A sua manutenção é bastante facilitada por existir uma única base de código, ao contrário das aplicações nativas. Permite a disponibilização das *apps* nas lojas de cada plataforma e o funcionamento sem ligação à internet, ao contrário das aplicações *web*. Tem o potencial de aceder totalmente aos componentes dos dispositivos, ao contrário das *PWA*s. Contudo, este modelo nem sempre poderá ser o ideal, dependendo das intenções do negócio. Por exemplo, no caso de um jogo ou outras *apps* que tenham a necessidade de executar tarefas intensas, as aplicações híbridas não serão capazes de assegurar o desempenho característico das aplicações nativas. É possível observar na Tabela 1 uma comparação dos vários modelos descritos nesta seção.

Tabela 1 - Comparação entre modelos de aplicações

	Nativo	Web	PWA	Híbrido Web	Híbrido Nativo
Podem ser instaladas	Sim	Não	Sim[d]	Sim	Sim
Funcionam offline	Sim	Não	Sim[b]	Sim	Sim
Multiplataforma	Não	Sim	Sim	Sim	Sim
Acesso a componentes	Total	Limitado[a]	Limitado[a]	Potencial[c]	Potencial[c]
Presente nas lojas de apps	Sim	Não	Sim	Sim	Sim

a) Devido às restrições impostas pelos *browsers*

b) Com recurso a *service workers*

c) Aplicações multiplataforma têm a capacidade de aceder aos componentes nativos dos dispositivos. Em *Ionic* isto é conseguido através de *Plugins* [23], enquanto que em *React Native* são utilizados componentes [24]. Muitos dos componentes disponibilizados para as plataformas referidas são produzidos pela comunidade, resultando em alguma dependência para a sua manutenção e criação. Por este motivo, o acesso a componentes foi considerado como potencial ao invés de total.

d) Desde que a *Google* introduziu o conceito de *Trusted Web Activities* é oficialmente possível integrar uma *PWA* com *Android*, permitindo a publicação de *PWAs* na *App Store* [25].

2.2. Visão computacional

Define-se *CV* como sendo a área do conhecimento que tem como objetivo a atribuição da capacidade de compreender e interpretar conteúdo visual a máquinas. De acordo com Szeliski [26], com *CV* "pretende-se a descrição do mundo visível em uma ou mais imagens e a reconstrução das suas propriedades...". Como meio para alcançar este objetivo, são utilizados algoritmos e técnicas provenientes de várias áreas como a matemática, física, inteligência artificial, *machine learning*, etc.

Este tema encontra-se bastante presente nos dias de hoje, sendo uma componente chave em tecnologias como reconhecimento facial, condução automática e *OCR*. Devido ao seu potencial promissor, ao longo dos anos tem sido o alvo de várias pesquisas que têm como fim avançar o seu desenvolvimento, expandindo-o para domínios como o da segurança [27] e medicina [28]. Prevê-se que a presença de *CV* venha a aumentar em diversas áreas, resultando na evolução e criação de novas indústrias [29].

Pretende-se com este projeto o desenvolvimento de uma solução que recorra a este domínio para resolver o problema da identificação de presenças. Como tal, foi realizada uma pesquisa por soluções semelhantes para que fosse possível estudar as abordagens normalmente adotadas e descobrir como esta solução pretendida poderia contribuir para a área.

Um dos principais desafios do atual problema encontra-se na extração e segmentação da tabela contida na imagem, algo que da mesma forma se encontra presente em problemas como o da resolução de sudokus através de *CV* [30] [31] [32]. Não só este tipo de problema foi importante para compreender como poderiam ser isoladas as células das tabelas para este projeto, mas também permitiu reconhecer a possibilidade de integração da técnica de *OCR* dentro desse processo. Uma representação do produto resultante da extração de uma grelha de sudoku encontra-se na Figura 2. Ao serem isoladas as células contidas na grelha do jogo, é feita a identificação dos dígitos em si contidos. Essencialmente, é algo similar com aquilo que se pretende com este projeto, a segmentação das células da tabela de presenças, identificação dos nomes e registo das presenças em si assinaladas. Por este motivo, tendo por base exemplos como os referidos, foi possível identificar estratégias que viriam a ser implementadas na solução final. Do mesmo modo, foi possível traçar a sequência das etapas

que seriam necessárias para a resolução do problema, começando com o isolamento da tabela, seguindo-se a sua segmentação e finalizando com a interpretação dos seus conteúdos.

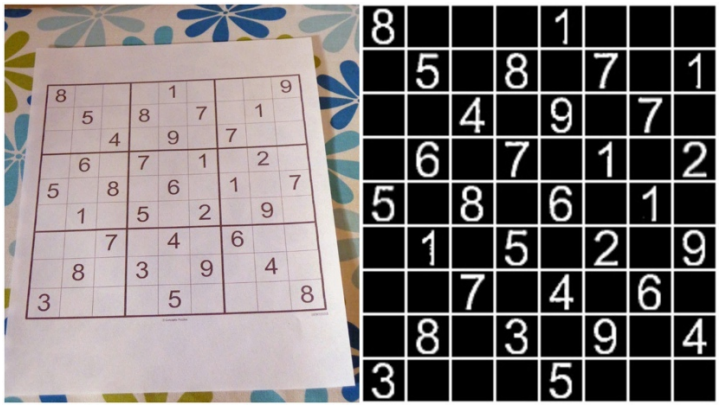


Figura 2 - Exemplo de extração de uma grelha de sudoku[3]

Relativamente à identificação dos conteúdos da tabela, foi realizado o estudo de vários trabalhos em conjunto com a realização de vários testes, com o intuito de descobrir a melhor abordagem a aplicar neste projeto. As tabelas de presenças criadas pela *APM* são geradas cumprindo um determinado formato, detalhado na secção 5.5.1.1. Sabendo que estes documentos são gerados automaticamente, é possível determinar alguns fatores que possibilitam a construção de uma solução ideal para o problema. Espera-se que na folha esteja contida uma listagem de todos os participantes, cada um acompanhado por dois espaços correspondentes à sua presença no período da manhã e da tarde do evento. A identificação dos utilizadores é preenchida à máquina e por este motivo, após a segmentação da tabela, pretende-se que a sua leitura seja possível com recurso a *OCR*, técnica aprofundada na secção 2.2.1. Contudo, existe ainda a necessidade por uma técnica que permita avaliar se um determinado utilizador assinalou, ou não, a sua presença num dos espaços da tabela a si correspondentes.

De modo a determinar como poderia ser feita a determinação de presenças neste projeto, foi necessário perceber como os utilizadores realizavam esta ação por si. Observou-se que na maioria dos casos cada utilizador marcava a sua presença na folha com a sua assinatura ou rúbrica, o que inicialmente proporcionou a pesquisa por uma solução que possibilitasse o seu reconhecimento e correspondência com o utilizador. Contudo, esta opção foi descartada após ter sido determinado que não era importante para o *POC* a correspondência entre o

[3] https://aakashjhawar.medium.com/sudoku-solver-using-opencv-and-dl-part-2-bbe0e6ac87c5

participante e a sua assinatura. Ou seja, um sujeito apenas precisa de assinalar a sua presença na folha, mesmo que com uma cruz, para que seja considerada a sua presença. Por este motivo, a solução desenvolvida utiliza uma técnica que não discrimina entre os tipos de registos efetuados, sendo apenas considerada a sua presença na tabela.

2.2.1. Optical Character Recognition

Define-se *OCR* como sendo a tecnologia que fornece a máquinas a capacidade de extrair texto contido numa imagem digital, possibilitando posteriormente a sua edição e processamento. A ideia de equipar uma máquina com esta capacidade interpretativa de texto não é propriamente recente, podendo a sua origem ser datada aos anos 20 e 30 [33]. Eventualmente, com o passar do tempo, *OCR* evoluiu ao ponto de se encontrar presente em diversas áreas comerciais, sendo uma tecnologia que conseguiu manter a sua relevância ao longo de várias décadas. Atualmente é possível identificar diversas das suas aplicações nomeadamente no processamento de documentos, *scan* de passaportes, embalamento de produtos alimentares, produção automóvel, etc. [34].

É possível fazer a distinção entre diversas fases que integram um sistema de *OCR* [35] [36] [37]. As fases que integram este processo variam de projeto para projeto dependendo das suas necessidades, contudo, é possível destacar algumas cuja ocorrência se verificou na maioria dos casos estudados. No exemplo da Figura 3 encontra-se representado um sistema *OCR* que irá servir como base para a explicação do funcionamento da tecnologia. Cada uma das suas etapas contribui de forma única e importante para a extração do texto contido na imagem, encontrando-se as principais enumeradas e descritas em seguida.

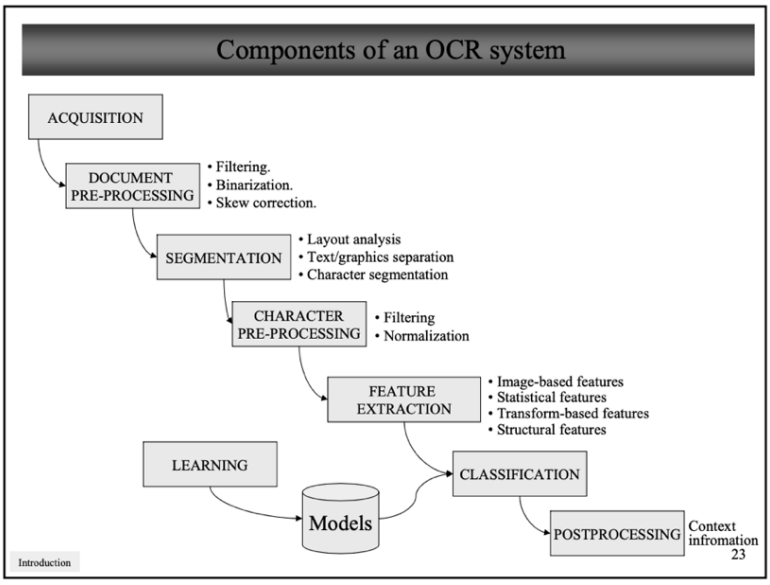


Figura 3 - Etapas de um sistema *OCR*[4]

1. Aquisição da imagem – O processo começa com a aquisição da imagem que contém o texto. A sua origem pode ser proveniente de aparelhos de digitalização, câmaras fotográficas, telemóveis, etc. No caso específico deste projeto, as imagens a analisar são obtidas através da segmentação da tabela contida numa fotografia da folha de presenças.
2. Pré-processamento – Etapas de pré-processamento poderão repetir-se ao longo da sequência de eventos. Essencialmente, tem como objetivo transformar a imagem para que seja mais fácil fazer a análise do seu conteúdo. Para tal, são utilizadas diversas técnicas, encontrando-se aquelas que foram utilizadas neste projeto explicadas na secção 5.5.2.1.
3. Segmentação – Recorrendo novamente a técnicas de pré-processamento, surge esta etapa com o objetivo de preparar a informação para ser analisada. Como tal, espera-se o isolamento de secções do documento e caracteres em si contidos, de modo a remover redundância e possibilitar o trabalho individual de cada segmento.
4. Extração de características – A partir do conteúdo resultante da etapa de segmentação, será efetuado nesta etapa o registo das suas características. Dependendo do tipo de problema, poderá ser feita uma análise topológica, geométrica e até mesmo estatística de cada caracter. Estas características serão a base para a identificação de cada caracter que será analisado.

[4] https://pdfs.semanticscholar.org/0641/566e9f65eacabbf10994410f5c3578eadc1d.pdf

5. Classificação – Nesta etapa é feita a correspondência entre as características detetadas e o caracter que pretendem representar. Para este fim, existem várias abordagens que recorrem a métodos como o de correlação [38], algoritmos como o *K-Nearest Neighbor* [39], redes neuronais [40], etc.
6. Pós-processamento – Devido à margem de erro que existe em sistemas de *OCR*, são frequentemente implementadas técnicas de pós-processamento com o objetivo de mitigar falhas que possam ter ocorrido ao longo do processo. Um exemplo simples deste tipo de técnica é o uso de métodos baseados em dicionários, que permitem assimilar os resultados das leituras a palavras já existentes, ou até mesmo corrigir possíveis erros gramaticais.

2.3.Serviços RESTful

Um serviço *RESTful* é qualquer serviço que cumpra com a arquitetura *Representational State Transfer* (*REST*). Essencialmente, permitem a imposição de um conjunto de regras que detalham como devem ser desenhados e processados os pedidos efetuados à *API* [41]. A sua relevância para este projeto, surge do seu uso na *API* da empresa para a qual o *POC* foi construído. Nesta secção serão descritos os seus princípios mais importantes.

Tendo por base o protocolo *Hypertext Transfer Protocol* (*HTTP)*, a comunicação entre o cliente e o servidor é realizada com base nas operações *GET*, *POST*, *PUT* e *DELETE*. Isto é importante pois o *REST* atribui significado a cada um destes pedidos, de forma a representar as intenções do cliente. É por isso introduzido o acrónimo *CRUD*, representativo dos termos *Create*, *Read*, *Update*, *Delete*. Cada um destes designa operações que podem ser exercidas sobre os dados da *API*, e podem ser executados através dos pedidos *HTML* referidos.

- *POST* – Correspondente à operação *Create*. É utilizado quando se pretende a criação de novos dados.
- *GET* – Correspondente à operação *Read*. É utilizado quando se pretende apenas devolver dados da *API*.
- *PUT* – Correspondente à operação *Update*. É utilizado para atualizar dados já existentes na *API*.
- *DELETE* – Correspondente à operação *Delete*. É utilizado quando se pretende apagar dados da *API*.

Outro aspeto característico da arquitetura *REST* é a forma como os seus *endpoints* são estruturados. Estes, deverão ser construídos para ser representativos da lógica dos dados. Por exemplo, num cenário em que existe uma *API* para uma empresa fictícia cujo endereço é *https://www.empresa.pt/*. Um exemplo de um *endpoint* que permitiria acesso aos dados dos seus funcionários poderia ser *https://www.empresa.pt/funcionarios/.* Ao ser realizado um pedido *GET* a este *endpoint*, seriam devolvidos os dados de todos os funcionários. Contudo, no caso de se pretender o acesso a um determinado indivíduo, então este *endpoint* deverá continuar a ser especificado conforme a lógica da empresa. Assumindo que cada funcionário tem um *id* único na base de dados, um possível *endpoint* poderia ser *https://www.empresa.pt/funcionarios/{id}*, possibilitando deste modo a devolução, atualização ou remoção do funcionário especificado.

Ao serem respeitadas as regras desta arquitetura, é possível construir *API*s com baixa curva de aprendizagem, descomplicadas, descritivas, intuitivas e rápidas.

2.4. Soluções semelhantes

Nesta secção são exploradas algumas aplicações móveis que de alguma forma possuem a capacidade de extrair informação contida em estruturas do tipo tabela através de técnicas de visão computacional.

A aplicação *Microsoft Office Lens – PDF Scanner*[5] (Figura 4) é talvez o paralelo ideal para comparar com a solução desenvolvida para este projeto. Esta aplicação existe com o objetivo de fornecer aos seus utilizadores a capacidade de digitalizar documentos reais a partir de um dispositivo móvel. Ao abrir a aplicação o utilizador poderá escolher uma fotografia da sua galeria ou tirar uma com a câmara do dispositivo. Caso escolha a segunda opção terá a hipótese de escolher entre 4 modos de fotografias, sendo estes respetivamente "quadro", "documento", "cartão de visita" e "fotografia". Cada modo permite otimizar o resultado obtido conforme o meio de onde o utilizador pretende extrair a informação. De modo a exemplificar o processo de extração de informação de documentos, será apesentado um exemplo demonstrativo. Após a obtenção de uma imagem, a aplicação irá automaticamente detetar as margens do documento, contendo a informação relevante. Estas margens poderão

[5] https://play.google.com/store/apps/details?id=com.microsoft.office.officelens&hl=en_US&gl=US

ser ajustadas pelo utilizador caso este o deseje. A partir deste ponto, o utilizador terá à sua disponibilidade algumas funcionalidades de edição da imagem, assim como a capacidade de exportar o resultado para vários formatos, sendo os de maior notoriedade para foto, *pdf* e diretamente para *word*. No caso da extração do documento para *word*, será gerado um documento que contém a imagem original acompanhada de uma versão onde todo o texto presente é editável, extraído com recurso à tecnologia de *OCR*.

Considerou-se que entre os seus pontos fortes está a integração com os programas do *office* (permitindo uma grande facilidade de uso), o uso de realidade aumentada para pré-visualizar as margens detetadas (o que não foi possível replicar com *Ionic* sendo que este necessita recorrer a *plugins* para ter acesso à câmara) e a variedade de casos diferentes em que é aplicável (o problema atual apenas é feito para detetar as tabelas utilizadas na empresa). Contudo, é também necessário destacar alguns dos aspetos negativos como é o caso da sua incapacidade de interpretar resultados e agrupá-los logicamente. Apesar da sua eficiência para traduzir documentos para ficheiros editáveis, não existe uma forma direta de interpretar os dados obtidos, algo vital para o problema de reconhecimento de presenças numa tabela.

Em síntese, tal como a solução desenvolvida para este problema, a aplicação *Microsoft Office Lens* permite que seja possível a um dispositivo móvel a extração de texto a partir de uma fotografia.

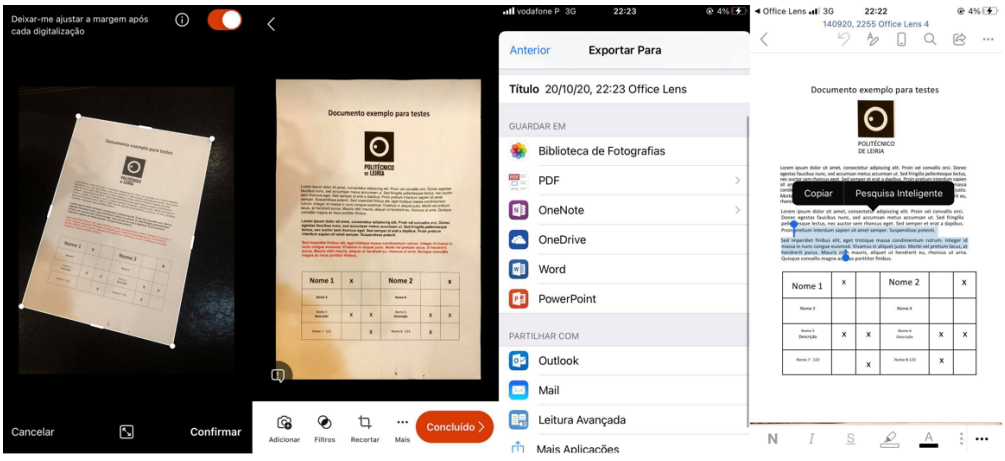


Figura 4 - Imagens da app Microsoft Office Lens

A aplicação *Adobe Scan*[6] (Figura 5) é também uma aplicação para digitalização de documentos. É considerada uma das melhores soluções gratuitas (inserir citação).

Ao entrar na aplicação o utilizador irá ter acesso à sua câmara e galeria, podendo à semelhança do *Microsoft Office Lens* escolher diferentes modos para as fotografias consoante a fonte da informação a ser digitalizada. Após a obtenção de uma fotografia a aplicação irá determinar as margens do documento, sendo estas ajustáveis pelo utilizador. Haverá a este ponto a opção de extrair imediatamente o texto da imagens, algo que não se encontra presente nas outras soluções exploradas para este relatório e que de facto acaba por tornar esta aplicação bastante mais prática, descartando a necessidade da exportar o texto da fotografia para um documento de texto e tendo esta informação muito mais acessível. Contudo, o utilizador continua a ter a oportunidade de prosseguir para a edição da fotografia capturada, podendo aplicar algumas técnicas de pré-processamento na imagem. No final, o documento resultante poderá ser extraído para *pdf*. O ponto forte desta aplicação é o facto de possuir uma interface bastante intuitiva e completa, conseguido também detetar de forma correta o texto presente nos documentos utilizados para testar a aplicação. Contudo, à semelhança da aplicação anterior, não é possível haver uma integração direta entre esta aplicação e as estruturas da *APM*, pelo que a automatização que é pretendida com este projeto seria inalcançável.

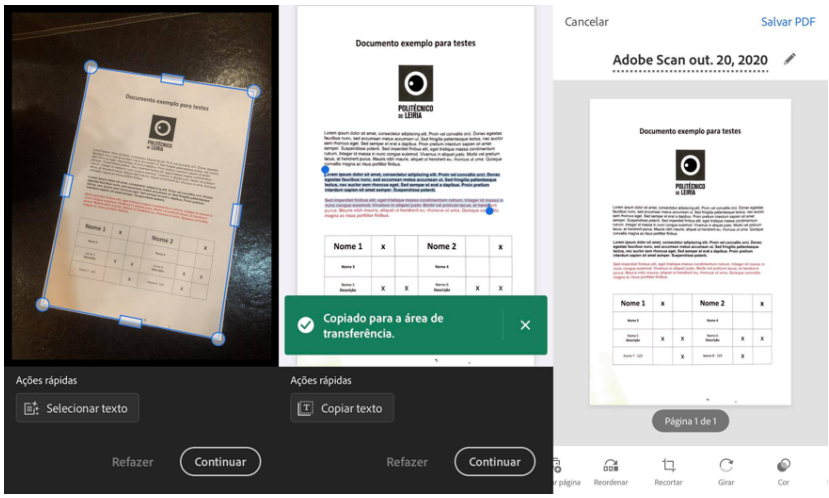


Figura 5 - Imagens da app Adobe Scan

[6] https://acrobat.adobe.com/pt/pt/mobile/scanner-app.html

A aplicação *iScanner*[7] (Figura 6) é a aplicação de digitalização de documentos que se encontra no topo da lista de aplicações para empresas da *App Store* na data da escrita deste relatório. Tal como as outras aplicações mencionadas anteriormente, a principal função deste programa é a digitalização de documentos, traduzindo o texto presente numa fotografia para um formato editável. As funcionalidades desta aplicação são comuns às dos exemplos anteriores, com exceção das capacidades de proteger digitalizações através de um sistema de passwords, de assinar documentos, e de resolver equações presentes nas digitalizações. Surge, contudo, uma grande barreira que coloca esta aplicação em desvantagem aquando comparada com as outras. Existe uma versão "PRO" da aplicação, acessível através da aderência a um plano de subscrição. Utilizadores que não paguem este valor terão acesso limitado às funcionalidades do aplicativo, inclusive a capacidade de utilizar *OCR* para identificar o texto presente nos documentos. Por esta razão esta aplicação não poderia ser considerada como alternativa para resolver o problema do projeto.

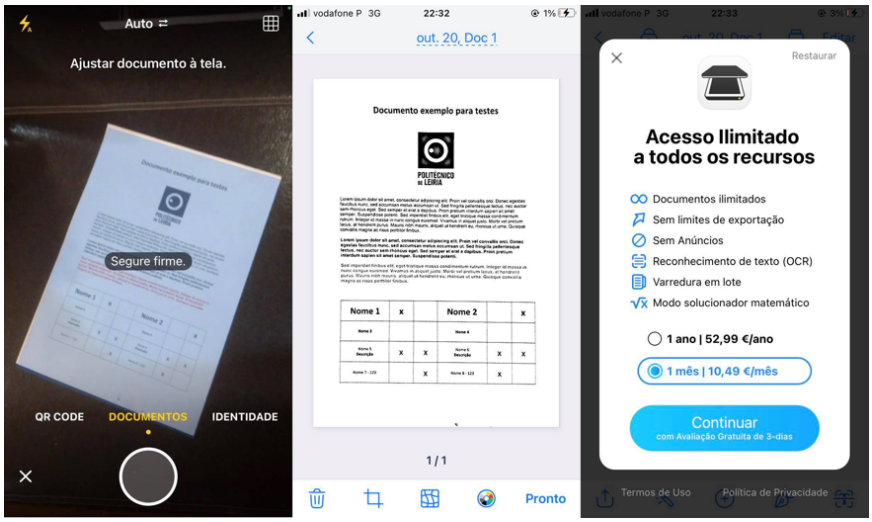


Figura 6 - Imagens da app iScanner

[7] https://apps.apple.com/pt/developer/bpmobile/id1085442365

3. Planeamento e Metodologia

Antes de dar início a um projeto é importante formular uma ideia de como este será feito num contexto temporal. Existem imensas condicionantes que afetam o modo como é realizado o desenvolvimento, contudo aquela que se destaca frequentemente como sendo a mais importante é o tempo estabelecido para a realização do projeto. No caso deste projeto em concreto, existem dois aspetos importantes a considerar, sendo estes as exigências do cliente e o limite estabelecido pelos órgãos da *ESTG* para entrega dos trabalhos de mestrado. Estas condicionantes exigiram a construção de um plano que permitisse organizar o tempo a ser investido nas suas diferentes etapas, pois deste modo é possível ter uma constante noção do estado do projeto e ajustar o foco de trabalho conforme esse estado.

Juntamente com o planeamento, outro elemento essencial para assegurar o cumprimento das tarefas necessárias para a conclusão deste projeto foi o uso de uma metodologia de desenvolvimento de *software*, de modo a permitir a gestão e organização de todo o processo. A metodologia adotada foi o *Scrum*, utilizada devido à sua adoção pela *md3* como principal *framework* para gestão do processo de desenvolvimento dos seus processos.

Neste capítulo será primeiramente descrito o planeamento elaborado para este projeto acompanhado pela descrição do trabalho realizado ao longo das várias etapas, seguindo-se a explicação da metodologia aplicada, o *Scrum,* de acordo com a forma que este foi utilizado neste projeto.

3.1. Planeamento

O plano inicial, proposto pela *md3*, é composto por várias etapas que orientam o trabalho a ser realizado ao longo de um período de 9 meses. Este trabalho segue uma lógica que permite assegurar não só o desenvolvimento atempado das tarefas, mas também a qualidade do trabalho realizado. Na Figura 7 encontra-se um cronograma que apresenta a verdadeira divisão de tempo feita no projeto, que apesar de não idêntica à original, é baseada nas suas fases. Sendo que o projeto realizado pretendia complementar um outro projeto de maior escala já composto por várias soluções, justifica-se a necessidade de uma primeira etapa que pretende a integração na equipa de desenvolvimento e uma introdução da metodologia a

adotar (3.1.1). De seguida, sendo que este projeto exige investigação, surge uma etapa que pretende o estudo do estado da arte e das ferramentas que poderão ser utilizadas no desenvolvimento de uma solução (3.1.2). Seguidamente, foi necessário estudar a arquitetura do ecossistema de aplicações da *APM* e iniciar o trabalho na aplicação, para verificar como os vários componentes se integram no projeto e para entender a estrutura e funcionamento da aplicação (3.1.3). Só após estas fases concluídas se avançou para a principal etapa deste trabalho, o desenvolvimento e implementação da solução, quer no lado da *API* quer na própria aplicação móvel (3.1.4). Por fim, após a concluído o projeto, deu-se continuidade ao desenvolvimento da aplicação (3.1.5). Este relatório foi escrito ao longo destas várias fases.

Face à situação excecional vivida pelo país, como consequência do surgimento do vírus COVID-19, surgiram algumas medidas para assegurar o sucesso dos trabalhos dos vários estudantes, nomeadamente o alongamento do prazo de entrega deste mesmo relatório, algo que surgiu como medida de auxílio por parte do Instituto Politécnico de Leiria. Esta expansão dos prazos provou-se bastante útil, pois foi necessária uma adaptação a novas condições de trabalho. Toda a equipa da *md3* passou a trabalhar remotamente, algo que exigiu a introdução de novos meios de comunicação e a adoção de novas ferramentas de trabalho remoto. Apesar destes contratempos, a *md3* conseguiu transitar para o trabalho remoto com bastante facilidade, algo que possibilitou a conclusão atempada deste projeto.

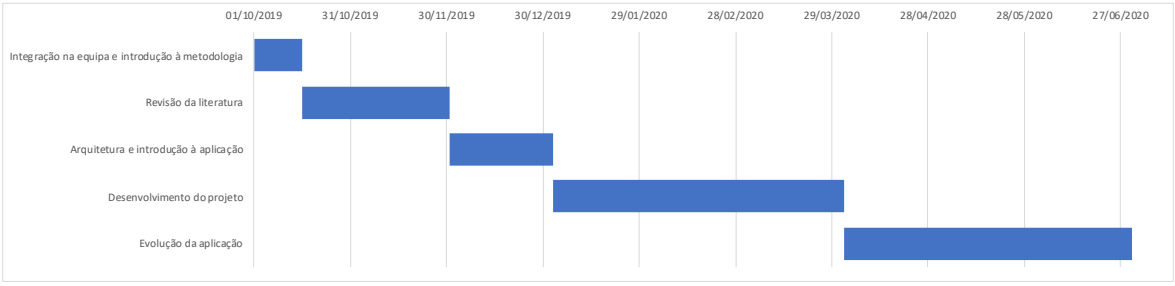


Figura 7 - Cronograma do projeto

3.1.1. Integração na equipa e introdução à metodologia

Sendo que este projeto foi realizado em grande proximidade com a *md3*, houve a necessidade de começar este projeto com uma fase introdutória através da qual foi possível conhecer os membros que integram a empresa. Ao longo do projeto foi necessário trabalhar várias vezes em equipa, em grande parte devido ao facto que a aplicação móvel ser apenas um dos

componentes que integram o ecossistema de aplicações da *APM*, pelo que existem também um *front office* e um *back office*. Havendo esta amplitude de projetos internos e tendo em conta que estes se encontram interligados, foi bastante importante conhecer antecipadamente todas estas estruturas e compreender as suas funções no sistema, porque apenas deste modo e através da cooperação com a equipa foi possível assegurar a integridade entre todas as soluções.

Os projetos da *md3* seguem uma metodologia ágil que permite manter todos aqueles que integram os seus projetos atualizados relativamente ao estado dos vários projetos. Parte do processo de integração incluiu o estudo e a aprendizagem dessa mesma metodologia, denominada de *Scrum* (descrita na secção 3.2). Um dos seus aspetos mais importantes é o conjunto de reuniões que ocorrem com frequências que podem variar desde semanalmente a diariamente, nas quais houve uma participação ativa desde o início deste projeto como forma de perceber melhor a metodologia e permitir a familiarização com o representante da *APM*.

3.1.2. Revisão da literatura

De modo a possibilitar o desenvolvimento deste *POC*, foi necessária a realização de uma extensa pesquisa que permitisse aprofundar o conhecimento acerca das áreas, tecnologias e ferramentas que poderiam ser relevantes para este projeto. Essa mesma investigação, apesar de ter sido um processo contínuo ao longo do desenvolvimento, teve início nesta etapa.

Primeiramente, foi necessário estudar as tecnologias híbridas de modo a compreender melhor a aplicação móvel da *APM* e a sua forma de programação. Foram analisadas e comparadas as principais *frameworks*, seguindo-se o estudo da documentação do *Ionic* e a realização de vários tutoriais acerca do seu funcionamento juntamente com *Angular*. O estudo das aplicações híbridas culminou na criação de uma aplicação exemplo onde foram aplicadas todas as técnicas estudadas, nomeadamente o envio e tratamento de notificações *PUSH*, a realização de chamadas e o envio de mensagens a partir da aplicação, a possibilidade de realizar pesquisas e filtrar listas de dados, e o uso de componentes do próprio dispositivo como, por exemplo, a câmara.

Seguiu-se um estudo da área de visão computacional, através do qual se pretendia compreender o problema deste projeto e determinar a melhor abordagem para a sua

resolução. Foram estudados vários artigos científicos, *blogs*, documentações, jornais e livros que de alguma forma possuíam semelhanças com este projeto, a partir dos quais foi possível compreender o funcionamento da tecnologia e como esta poderia ser aplicada ao problema. Da mesma forma, foi feito um estudo comparativo onde se avaliaram várias ferramentas de visão computacional e a partir do qual foi possível determinar as mais adequadas para o contexto deste projeto. Desta etapa resultaram diversos projetos exemplo onde foram testadas as várias técnicas e ferramentas.

Outro aspeto importante desta etapa foi a compreensão de como é realizado o *deploy* das aplicações nas diversas plataformas. Considerando que o trabalho será realizado numa aplicação híbrida, espera-se que esta seja distribuída para ambas a *App Stor*e e *Play Store*. De modo a obter os conhecimentos necessários para ter conforto com ambas as plataformas, houve um período de exploração onde foi possível observar o ciclo de vida da aplicação desde que esta é escrita e continuando com a sua compilação, configuração e envio para as plataformas web onde poderá ser disponibilizada para as *stores* referidas anteriormente.

3.1.3. Arquitetura e introdução à aplicação

Após estar estabelecido o conforto com a empresa, as ferramentas e as tecnologias que foram necessárias para a realização deste projeto, surgiu esta fase como meio de transição para o trabalho com o projeto real. Alguns membros da *md3* realizaram uma demonstração acompanhada pela explicação dos vários projetos que foram desenvolvidos para a *APM*, esclarecendo a lógica de negócio do cliente e demonstrando as principais funcionalidades desses mesmos sistemas e a forma como estes se encontram interligados. Como resultado desta formação, adquiriu-se o conhecimento básico necessário para compreender a *APM* e iniciar o trabalho na aplicação.

Foram feitos vários exercícios para alcançar a familiarização com a estrutura da aplicação e como forma de aplicar os conhecimentos previamente adquiridos sobre *Ionic*. A principal tarefa a contribuir para este objetivo foi a de atualizar a aplicação para a versão mais recente da sua *framework*. A *app* no seu estado inicial encontrava-se na versão 2 do *Ionic* incapaz de sequer compilar, sendo que o motivo para este erro se devia à presença de código obsoleto. Através da atualização da lógica da aplicação e da correção dos vários erros resultantes deste processo, foi possível cumprir as normas da versão 4 do *Ionic*. Mais tarde no processo de

desenvolvimento foi lançada a versão 5 do *Ionic*, que apesar de ter exigido algumas alterações no processo de desenvolvimento de código, não exigiu uma transformação tão drástica como da versão 2 para a 4.

3.1.4. Desenvolvimento do projeto

Determinou-se que o sistema de leitura de tabelas iria utilizar a aplicação como meio de obtenção de imagens, enviando-as para a *API* para que estas pudessem ser sujeitas ao processo de leitura, interpretação e registo de assinaturas. Nesta fase inicial, onde se pretendia a criação do *POC*, a prioridade estava em desenvolver uma solução que fosse capaz de demonstrar o comportamento esperado da solução final, pelo que o primeiro módulo a ser trabalhado foi o que correspondia ao desenvolvimento na *API*.

Este trabalho exigiu a familiarização com a outra principal *framework* deste projeto, o *Django*, tendo havido um período de tempo dedicado unicamente a este fim. Foi feito um estudo da sua documentação assim como a realização de alguns tutoriais, tendo este estudo culminado na criação de uma aplicação exemplo onde se pretendia, através da aplicação de todas as técnicas adquiridas, a criação de uma loja online. Após a conclusão deste pequeno projeto, que permitiu interiorizar os básicos do trabalho com a *framework*, foi então dado início ao trabalho destinado à *API* do cliente.

O desenvolvimento da solução de leitura de tabelas foi um processo de constante pesquisa, tendo sido aplicados e testados diversos métodos e lógicas que foram descobertos mesmo após a fase de revisão de literatura. As imagens utilizadas para testar o funcionamento da aplicação foram produzidas manualmente, pois deste modo era possível obter imitações das tabelas reais, mas sem a presença de ruído ou outros artefactos frequentemente presentes em fotografias. Esta fase do desenvolvimento resultou numa solução que sujeitava as aplicações a um processo composto pelas etapas de extração da tabela, segmentação das células, identificação dos nomes, identificação e registo das presenças. Este módulo foi transformado num *package* e documentado utilizando *sphinx*[8].

Após a criação de uma versão funcional do *POC* pretendido para este projeto, passou-se para o desenvolvimento da solução na parte da aplicação móvel. A comunicação entre ambos

[8] https://www.sphinx-doc.org/en/master/

módulos foi realizada através de pedidos *HTTP,* utilizando *plugins* do *Ionic* para este efeito no lado da *app*. Foram também utilizados *plugins* para permitir o uso da câmara nos dispositivos, sendo este o principal meio para captura de imagens da tabela. Estando concluída esta primeira versão estável da aplicação, foi possível obter um sistema que integrava ambos os módulos do projeto.

Havendo sido concluído uma versão completa do sistema de registo de presenças, foi iniciado um processo de otimização deste mesmo, baseado em vários testes efetuados ao sistema. Foram recolhidas fotografias de folhas de presenças já assinadas, utilizadas em eventos da *APM*. Após terem sido realizados testes com essas mesmas imagens, foi possível observar inúmeros problemas que impediam a presente solução de funcionar num contexto real. Como tal, a solução foi melhorada para conseguir responder a essas mesmas adversidades.

O resultado final desta etapa foi uma versão do *POC* capaz de funcionar com exemplos reais, com uma precisão sempre superior a 90% em todos os testes realizados.

3.1.5. Evolução da aplicação

Após concluída a versão estável do *POC*, deu-se continuidade ao desenvolvimento da aplicação através da implementação de melhorias e de novas funcionalidades requisitadas pela *APM*. Entre as várias alterações efetuadas, destacam-se a criação de um sistema de notificações *PUSH*, a adição de novas páginas e a realização de *spikes* com novas plataformas de testes e de *deployment*. Foi também realizado o *deployment* da aplicação para as plataformas da *Google* e da *Apple*, o que implicou criação e gestão de chaves. Um dos outros aspetos mais importantes desta etapa foi a participação mais ativa no *Scrum*, havendo uma maior participação na estimativa dos valores dos vários *tickets* e o trabalho em conjunto com membros das equipas de *front office*, *back office* e *design*. Manteve-se também o constante contacto com o cliente, pelo que houve também a tarefa de resposta a pedidos de suporte à aplicação.

3.2. Scrum

Para o desenvolvimento deste projeto foi adotada a metodologia *Scrum,* por ser padrão em todos os projetos desenvolvidos na *md3*. Uma metodologia de *software* é uma *framework*

que pretende guiar o processo de desenvolvimento. Tradicionalmente, estas metodologias exigiam um planeamento determinista no início do desenvolvimento e os clientes dos projetos muitas vezes só teriam acesso ao projeto no seu fim. Contudo, o *Scrum* não segue as normas tradicionais, pertencendo a um grupo de metodologias denominadas ágeis, um título habitualmente atribuído quando são adotados os princípios popularizados pelo manifesto ágil [42]. Foram popularizadas pois a prioridade no desenvolvimento de projetos deixou de ser a antecipação de mudanças através do seu planeamento e passou a ser a resposta a mudanças à medida que estas vão surgindo ao longo do desenvolvimento [43]. O *Scrum* destaca-se das outras metodologias ágeis como sendo a mais utilizada [44]. É própria para equipas pequenas [45] como a *Devcore*, nome atribuído à equipa da *md3* dedicada a trabalhar nos projetos da *APM*.

São essencialmente 3 componentes que quando integrados compõem o *Scrum* (representado na Figura 8), sendo estes a equipa, os eventos e os artefactos. Seguidamente, procede-se à explicação destes pontos.

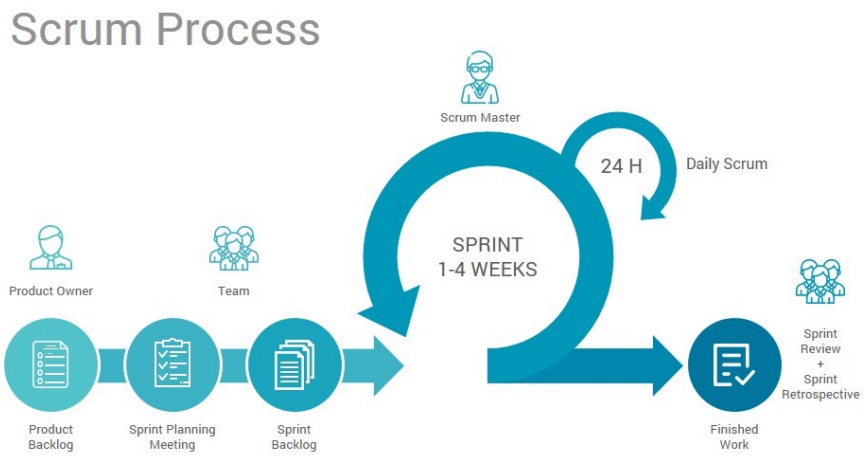


Figura 8 - Processo Scrum[9]

A equipa é importante no sentido em que é crucial para a metodologia o papel de cada indivíduo e as suas interações, assim como a colaboração com o cliente. Existem 3 tipos de papeis que podem ser atribuídos aos membros de uma equipa, sendo estes os de *Scrum Master*, *Product Owner* e *Development Team*. O *Scrum Master* é responsável por assegurar o bom funcionamento da metodologia, trabalhando pela eficácia e produtividade de todos os

[9] https://blog.4geeks.io/scrum-for-digital-product-development/

restantes membros. O *Product Owner* é o elemento responsável pela gestão do *product backlog*, um catálogo de funcionalidades, melhoramentos e *bugs* que se traduzem em tarefas a serem realizadas ao longo do projeto. A *Development Team* é a equipa auto-organizada e não hierárquica composta por todos os restantes elementos que contribuem para o projeto, trabalhando para acrescentar continuamente valor ao projeto. O trabalho realizado para este projeto foi feito sob o papel de *Development Team*, interagindo ativamente com todos os outros membros.

Os eventos são reuniões que acontecem entre os vários elementos da equipa, permitindo a definição e a organização cíclica do processo de desenvolvimento. É devido a este elemento da metodologia que se consegue responder continuamente à mudança e assegurar a constante presença de *software* funcional no projeto, pois é estabelecida uma rotina que permite a inspeção, avaliação e adaptação do processo de desenvolvimento. Esta qualidade acaba por ser vantajosa para um cliente, porque desta forma é possível este ter constante noção do estado do projeto e fazer alterações conforme estas vão sendo necessárias. Os eventos são *Sprint*, *Daily Scrum*, *Sprint Planning*, *Sprint Review* e *Sprint Retrospective*. A *Sprint* é o evento principal da metodologia, contendo todos os outros. O seu objetivo é a produção de um incremento para o produto final, determinado através do planeamento feito no seu início. Uma *sprint* começa assim que outra termina. O *Daily Scrum* é um evento diário onde cada membro da equipa de desenvolvimento comunica o trabalho realizado no dia anterior e o trabalho que pretende realizar nesse mesmo dia. O *Sprint Planning* acontece antes do início de cada *sprint* e reúne todos os membros da equipa de desenvolvimento com o objetivo de escolher as tarefas, denominadas de *stories*, a realizar e a estimar o seu valor para que possam ser incluídas nessa *sprint*. Na *md3* utiliza-se uma técnica de *gamification* denominada de *Planning Poker* para este efeito, onde cada membro escolhe um valor para cada tarefa e no final, cada elemento revela e discute os seus valores até ser atingido um consenso entre todos. A *Sprint Review* acontece no final de cada *sprint* e tem como objetivo a apresentação do trabalho realizado durante a mesma. No final de cada *Sprint Review* acontece a *Sprint Restrospective*, o evento que tem como objetivo a discussão dos aspetos positivos e negativos da *sprint* de modo a formular um plano de melhorias.

Os artefactos em *Scrum* permitem representar o trabalho a ser realizado de forma a ser possível organizá-lo, monitorizá-lo e acompanhá-lo ao longo dos vários eventos. Deste

modo, permite-se obter visibilidade do progresso realizado e do seu valor para o produto final. Os artefactos existentes na metodologia *Scrum* são o *Product Backlog*, *Sprint Backlog*, *Monitoring Chart* e *Product Increment*. O *Product Backlog*, gerido pelo *Product Owner*, corresponde à lista de todos os requisitos funcionais do projeto, denominados de *stories*. O *Sprint Backlog* é a lista de *stories* escolhidas do *Product Backlog* para serem realizadas durante a *sprint*. Os *Monitoring Charts* são gráficos que permitem visualizar o estado do trabalho ao longo das suas fases comparativamente com o trabalho restante. Na *md3* não existe uma forma direta de visualizar este artefacto, sendo utilizado o quadro *Kanban* (explicado na secção 4.4.4) para este efeito. Por fim, existe o *Product Increment* que se trata da soma dos valores de todas as *stories* do *backlog* que foram concluídas durante a *sprint* juntamente com os valores das *stories* das *sprints* anteriores.

O *Scrum* foi essencial para a realização deste projeto, pois permitiu um dinamismo na realização de tarefas e uma constante comunicação entre o cliente e a equipa. Possibilitou a transparência do processo de desenvolvimento que por sua vez se traduziu numa capacidade de adaptabilidade às necessidades do cliente. Foi deste modo possível aprofundar as capacidades de trabalho em equipa e de trabalho com metodologias ágeis ao longo deste projeto. O uso do *Scrum* foi também bastante importante quando este projeto passou a ser realizado remotamente como resultado da pandemia. Devido às características desta metodologia, a transição foi facilitada, estando todos os elementos da equipa em constante comunicação e havendo conhecimento global do estado do projeto.

4. Requisitos, Casos de Uso e Ferramentas

Uma das tarefas mais importantes no desenvolvimento de um projeto é a especificação dos seus requisitos. Determinam-se antes do início do processo de desenvolvimento e representam uma propriedade do *software* que tem de ser implementada de modo a satisfazer as necessidades de um cliente [46]. Em síntese, funcionam como princípios que irão guiar o programador na missão de concluir o seu projeto. A adoção de uma metodologia ágil permite que estes requisitos possam sofrer alterações ao longo do desenvolvimento caso o cliente o deseje. É possível fazer a distinção entre dois principais tipos de requisitos, funcionais e não funcionais. A explicação dos requisitos de um projeto é importante para compreender a lógica por detrás das decisões tomadas durante o desenvolvimento.

É possível aprofundar a compreensão do funcionamento do projeto através dos casos de uso, considerando que estes representam as interações entre agentes e um sistema, de forma a atingir um objetivo. Podem ser utilizados para representar requisitos, sendo o padrão na *md3* para a descrição de *stories*. Encontram-se ilustrados e explicados neste capítulo os casos de uso relativos à funcionalidade de deteção de presenças.

Pretende-se com este capítulo a enumeração dos vários tipos de requisitos estabelecidos assim como explicação dos casos de uso relativos à deteção de presenças. Por fim, encontram-se descritas as ferramentas utilizadas neste projeto, que permitiram a implementação dos requisitos e consequentemente a existência dos casos de uso.

4.1. Requisitos não funcionais

Este tipo de requisito específica atributos de um sistema relacionados com as suas qualidades. Refletem expectativas que existam para com o produto final nomeadamente ao nível do desempenho, segurança, usabilidade e custo. A sua existência é bastante importante devido à sua influência nas escolhas a serem feitas durante o desenvolvimento, sendo os erros relacionados com o incumprimento deste tipo de requisitos os mais dispendiosos e difíceis de resolver em fases avançadas de projetos [47]. É importante destacar que não existe um consenso global relativamente à definição de requisito não funcional [48] e, como tal, a descrição que deu introdução a esta secção representa a definição adotada neste relatório.

Antes do desenvolvimento deste projeto foram estabelecidos alguns requisitos não funcionais, baseados no planeamento realizado inicialmente. Em seguida, estes encontram-se enumerados:

- A nova funcionalidade deverá ser de acesso intuitivo;
- O sistema deverá ter bom desempenho, sendo capaz de responder com rapidez aos pedidos efetuados à *API;*
- O sistema deverá estar constantemente disponível;
- O sistema não deverá recorrer a serviços de terceiros caso estes impliquem acréscimos de custos;
- O sistema deverá funcionar tanto em *Android* como em *iOS;*
- Os dados das presenças registadas devem ser armazenados;
- O código criado deve ser corretamente documentado;
- Caso se aplique, devem existir instruções para a instalação de módulos;
- Devem existir instruções para o *deploy* das aplicações nas lojas;
- Código na aplicação móvel deverá ser escrito com *Ionic* e *Angular;*
- Código na *API* deverá ser escrito de acordo com a *framework Django;*
- Dados de carácter confidencial não deverão ser comprometidos;
- São cumpridas as normas que dizem respeito à proteção de dados.

4.2. Requisitos funcionais

Este tipo de requisito especifica os comportamentos que o sistema deverá ter, alcançáveis através da implementação de novas funcionalidades. Existe uma relação próxima com a metodologia *Scrum*, como foi referido na secção 3.2, os requisitos funcionais existem sobre a forma de *stories*. Na *md3*, para a sua representação é utilizado o formato de *user story*, onde se detalha o agente, o que este quer e o motivo.

Apesar da determinação de requisitos funcionais ter sido realizada de forma contínua ao longo do desenvolvimento, foram estabelecidos alguns na fase inicial com base no planeamento. Em seguida, estes encontram-se enumerados:

- Como utilizador, quero ter a possibilidade de enviar imagens para a *API,* para que estas possam ser sujeitas ao reconhecimento de presenças;

- Como utilizador, quero ter a escolha entre usar a câmara ou selecionar uma imagem da biblioteca, para que possa usufruir das capacidades do meu dispositivo;
- Como utilizador, quero que a informação que resulta da deteção de presenças seja devolvida para o meu dispositivo, para que este esteja constantemente atualizado;
- Como utilizador, quero que a informação que resulta da deteção de presenças seja armazenada no servidor, para que todos os sistemas da *APM* possuam dados atualizados;
- Como utilizador, quero que a aplicação tenha um sistema de notificações *PUSH*, para me manter atualizado e aumentar a minha interação com a aplicação.

4.3. Casos de uso

No contexto de um *software*, um caso de uso representa uma sequência de eventos em que o programa poderá ser utilizado. Estes, são adotados como forma de representação para requisitos funcionais [49], permitindo comunicar de forma simples aquilo que é pretendido. Na Figura 9 encontra-se representado o diagrama de casos de uso referente a este projeto, onde estão ilustradas as várias interações que o utilizador tem com o sistema.

Este tipo de diagrama é composto por vários elementos e pelas interações entre si, representados através de agentes, sistemas, casos de uso e relações. Os agentes são pequenas figuras humanoides, descrevem pessoas ou qualquer elemento que cause uma reação no sistema. No caso deste projeto, os principais agentes são utilizadores da aplicação com privilégios para aceder à funcionalidade de deteção de assinaturas. Seguem-se os sistemas, que representam o *software* em desenvolvimento. São representados por retângulos e contêm em si os casos de uso que ocorrem em si internamente. No caso deste projeto existem dois sistemas diferentes, a *app* e a *API*. Como já mencionando, existem os casos de uso que foram descritos no início desta secção. Representam-se através de elipses. Resta descrever as relações, que detalham como os diferentes componentes interagem entre si. Podem ser representadas de várias formas dependendo do seu tipo. Destacam-se as relações de associação (representa uma comunicação ou interação básica, existe sob a forma de uma linha contínua), inclusão (representa uma dependência entre dois casos de uso, uma seta descontínua com a palavra "*include*" que aponta para o dependente), extensão (representa uma condição e indica que um caso de uso apenas irá ocorrer mediante o cumprimento de

determinadas condições, uma seta descontínua com a palavra "*extends*" que aponta para o condicionante) e generalização (representa uma relação de hierarquia, sendo que os filhos partilham as qualidades do pai mas acrescentam o seu próprio funcionamento, uma seta contínua que aponta para o pai).

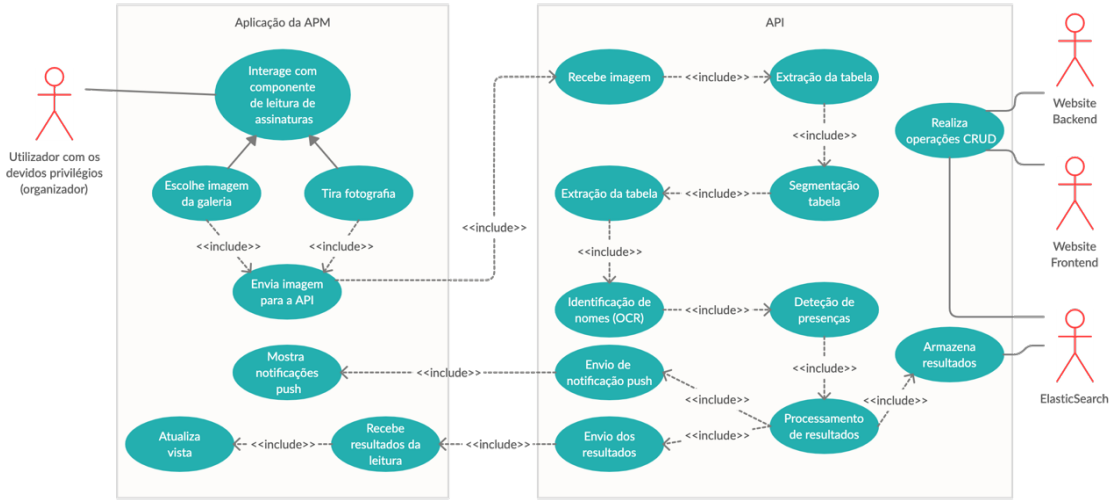


Figura 9 - Diagrama de casos de uso

4.4. Ferramentas utilizadas

Neste capítulo têm sido expostas as várias técnicas utilizadas durante a fase de pré desenvolvimento deste projeto, importantes devido ao seu impacto no processo de tomada de decisões no projeto. Esta secção surge com o intuito de complementar a informação até aqui exposta, apresentando as ferramentas mais relevantes para o desenvolvimento do projeto.

Os vários *softwares*, *frameworks*, serviços e bibliotecas que contribuíram para a construção da solução final são considerados ferramentas do projeto. A sua determinação é tão importante quanto a dos requisitos, pois as suas capacidades e limitações são fatores determinantes do processo de desenvolvimento. Apesar da eleição de algumas ferramentas ter ficado ao critério do programador, existem outras que foram de caracter obrigatório para o projeto, por motivos que serão explicados quando aplicável.

4.4.1. Ferramentas da API

Antes da realização deste projeto, a *API* onde a solução viria a ser construída já estava a ser trabalhada pela *md3*. Este facto justificou a necessidade da adoção das tecnologias que já se encontravam a ser utilizadas para este projeto, como é o caso da linguagem de código e da *framework*.

Python[10] é uma das linguagens mais utilizadas por programadores. Possui várias características que possibilitam que seja uma linguagem fácil de aprender, escalável e versátil. É de alto nível, o que significa que existe uma alta abstração do código máquina. Esta característica verifica-se pela existência de vários elementos que permitem simplificar a programação, como é o caso dos vários tipos de variável e métodos que já se encontram integrados na própria linguagem. Outra principal qualidade desta linguagem é a sua versatilidade, sendo capaz de produzir aplicações de vários domínios. Foi utilizada pela *md3* para a criação de vários *scripts*, *websites* e da *API* em causa. Relativamente à escalabilidade de *python*, esta existe devido à sua capacidade para adotar uma arquitetura modular. Tal é possível devido ao facto de ser uma linguagem orientada a objetos, e também por possuir qualidades como a de possibilitar a criação e importação de módulos e *packages* através da ferramenta *pip*[11]. Esta funcionalidade foi crucial para o processo de desenvolvimento, tendo possibilitado a integração com *OpenCV* (4.4.1.1) e *Tesseract* (4.4.1.2) neste projeto. Em seguimento à explicação da linguagem utilizada, é apresentada em seguida a *framework* utilizada para o desenvolvimento na *API*, o *Django* [3].

Uma *framework* permite simplificar o processo de desenvolvimento, estabelecendo uma base para a criação de programas. No caso do *Django*, são disponibilizados vários recursos que permitem um desenvolvimento rápido e simplificado de aplicações *web* escaláveis. Relativamente à rapidez durante o desenvolvimento, um dos fatores que justifica esta qualidade é a existência de várias funcionalidades integradas na *framework* que permitem a não repetição e a simplificação de tarefas comuns no desenvolvimento de aplicações *web*, sendo possível criar rapidamente protótipos e reduzir o trabalho na implementação de funcionalidades como autenticação através do uso de *packages*. Outro aspeto importante mencionado é a sua escalabilidade, proveniente da sua capacidade para criar aplicações

[10] https://www.python.org/
[11] https://pypi.org/

capazes de funcionar com inúmeros utilizadores. O uso de *Django* foi importante para este projeto porque facilitou o processo de compreensão da lógica da *API* e permitiu a integração de forma descomplicada da solução de leitura de tabelas no projeto existente.

A componente deste projeto que reside na *API* foi desenvolvida em *python* e transformada num *package*, de modo a simplificar o acesso à sua funcionalidade de deteção de presenças. Em seguida, são referidos os dois principais *packages python* que foram utilizados no desenvolvimento deste projeto.

4.4.1.1. OpenCV

A biblioteca *OpenCV* [50] foi a principal ferramenta utilizada neste projeto em operações relacionadas com *CV*. Incorpora mais de 2500 algoritmos otimizados, possuidores de diversas aplicações. Tem o potencial de ser utilizada em trabalhos relacionados com reconhecimento facial, deteção de objetos, edição de fotografias, segmentação de imagens (Figura 10), etc. É *open source*, como o seu nome indica, o que possibilita o seu uso gratuito na construção de soluções comerciais.

Foi escolhida esta biblioteca para a realização deste projeto devido à forte presença em problemas semelhantes [30] [31] [32], eficácia, documentação e natureza *open source*.

Figura 10 - Utilização de OpenCV para segmentação de imagens[12]

São várias as suas aplicações na solução desenvolvida, tendo possibilitado a realização de tarefas complexas como a extração da tabela de presenças e a sua segmentação. A sua importância encontra-se demonstrada no capítulo 6, onde é provada a forma como a ausência das suas técnicas de pré-processamento resulta no insucesso das leituras. É também

[12] https://circuitdigest.com/tutorial/image-segmentation-using-opencv

importante destacar como esta biblioteca vem a complementar técnicas como as de *OCR* (2.2.1) ou *Margin Patrol* (5.5.3.1.1), integrando a sequência de eventos que as definem. As principais técnicas incorporadas no processo de desenvolvimento encontram-se descritas no capítulo 5, conforme vão sendo utilizadas, entre as quais constam operações morfológicas, *Thresholding*, deteção de contornos, etc.

4.4.1.2. Tesseract

Um dos principais requisitos para este projeto era o uso da tecnologia *OCR* no processo de deteção de assinaturas. Ficou ao critério do programador a determinação da melhor abordagem para a implementação desta técnica, pelo que foi realizado uma extensa pesquisa e feito um estudo comparativo com os resultados da mesma, de modo a determinar a melhor solução a aplicar no contexto do problema. Um dos fatores determinantes no processo de escolha deve-se ao requisito estabelecido inicialmente, que não permitia o uso de soluções que implicassem o acréscimo de custos com a solução desenvolvida. Esta condição implicou a exclusão de algumas das propostas mais utilizadas para este tipo de problema, como é o caso da *Google Vision API*[13] e da *Amazon Rekognition*[14]. Para além desta condicionante, foi também crucial considerar as limitações dos próprios serviços, pois alguns destes, mesmo disponibilizando opções gratuitas, como é o caso do *ocr.space*[15], permitiam apenas um número limitado de chamadas à *API* por dia. Deu-se início aos testes com o modulo *pytesseract*[16], que encapsula o motor *Tesseract*[17] da *Google*, considerando que este não possuía as características limitantes mencionadas anteriormente. Na sua essência *Tesseract* é um motor *OCR* de caixa preta, isto é, recebe qualquer imagem que contenha texto, faz o seu processamento internamente e devolve o resultado da análise.

4.4.2. Ionic

A aplicação da *APM* encontra-se desenvolvida com recurso à *framework Ionic*, plataforma que possibilita a criação de aplicações multiplataforma. Recorre a tecnologias *web* como *HTML*, *CSS* e *JavaScript* para realizar o desenvolvimento, resultando numa única base de

[13] https://cloud.google.com/vision
[14] https://aws.amazon.com/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate&blog-cards.sort-order=desc
[15] http://ocr.space/
[16] https://pypi.org/project/pytesseract/
[17] https://github.com/tesseract-ocr/tesseract

código que poderá ser utilizada para a criação de aplicações *Android*, *iOS, web* e até mesmo *PWA*. Foi necessário compreender o seu funcionamento ao iniciar este projeto de modo a possibilitar o trabalho na *app* já existente. Em seguida, de modo a aprofundar o conhecimento relativo à plataforma *Ionic*, encontram-se explicadas algumas das suas principais características.

1. É feita a sua integração com *Angular*, resultando na classificação da sua arquitetura como sendo *Model, View ViewModel* (*MVVM*) [51], onde o modelo é responsável pelos dados da aplicação, as *views* pela apresentação de conteúdo aos utilizadores e os *ViewModels* pelo emparelhamento dos dados presentes nas *views* com as suas propriedades.
2. Disponibiliza diversos componentes pré-integrados para a criação de interfaces, possibilitando um foco no desenvolvimento rápido de aplicações [52].
3. Possibilita o acesso a funcionalidades nativas dos dispositivos como a câmara, o calendário e o *bluetooth* através do uso de *plugins*.
4. Disponibiliza de uma interface para a linha de comandos que permite facilmente interagir com projetos. Possibilita o teste de apps no browser, a criação de componentes e a compilação de projetos para a plataforma desejada.

4.4.3. Ferramentas de armazenamento de dados

Este projeto recorre a duas soluções para tratar o armazenamento de dados. Uma destas é *PostGres*[18], um sistema de bases de dados relacionais. A sua função neste projeto é a de armazenar dados correspondentes a utilizadores, permissões e dispositivos. O seu uso é bastante específico e não é considerado como sendo o principal meio de armazenamento, contudo, é de extrema importância para o funcionamento das aplicações da *APM*.

O principal sistema de armazenamento que alimenta os *websites* e a aplicação da *APM* é *ElasticSearch*[19]. Esta ferramenta trata-se de um motor de pesquisa e analítica construído em *Java*, capaz de efetuar pesquisas rapidamente. O fator que permite a velocidade característica do *ElasticSearch* é a sua técnica de indexação que é aplicada sobre os dados recebidos.

[18] https://www.postgresql.org/
[19] https://www.elastic.co/

Essencialmente, os dados existem sob a forma de documentos *JSON*. Os conteúdos destes documentos são utilizados para a construção de um índex invertido, fazendo com que cada termo tenha conhecimento da sua relação com outros documentos. Deste modo, é possível devolver vários documentos relacionado através da busca de um único termo. É, deste modo, possível reconhecer que *ElasticSearch* traz várias vantagens para o projeto, destacando-se como sendo principal a velocidade que consegue obter nas suas pesquisas.

4.4.4. Ferramentas de controlo de versões

Como foi referido na secção 3.2, neste projeto é utilizada a metodologia *Scrum*. Como tal, é crucial a presença de determinadas qualidades no processo de desenvolvimento, destacando-se os três pilares da própria metodologia: transparência, adaptabilidade e inspeção. Para que este objetivo seja possível, ao longo do desenvolvimento toda a equipa tem de ter acesso à informação relevante do projeto, estando este constantemente sujeito a alterações determinadas a partir de eventos *Scrum*.

De modo a assegurar o cumprimento dos princípios da metodologia e a organização do próprio processo de desenvolvimento, a *md3* recorre ao uso de ferramentas de gestão de projetos. A principal ferramenta utilizada é *GitLab*[20], uma plataforma que permite organizar, gerir e observar o progresso do trabalho nos diferentes projetos da empresa. Todas as suas funcionalidades disponibilizam uma interface bastante acessível e de fácil compreensão, como é o caso do seu quadro *Kanban* (representado na Figura 11) que permite organizar e visualizar tarefas que acrescentam valor ao produto final, representadas sob a forma de *stories*. Ao observar o quadro presente na figura, destaca-se o modo como 3 das etapas que constituem o processo se destinam a testes, sendo estas *Code Review*, *Awaiting Review* e *Product Owner*. O funcionamento de cada etapa encontra-se especificado na secção 6.1 deste documento, no capítulo relacionado com testes.

[20] https://about.gitlab.com/

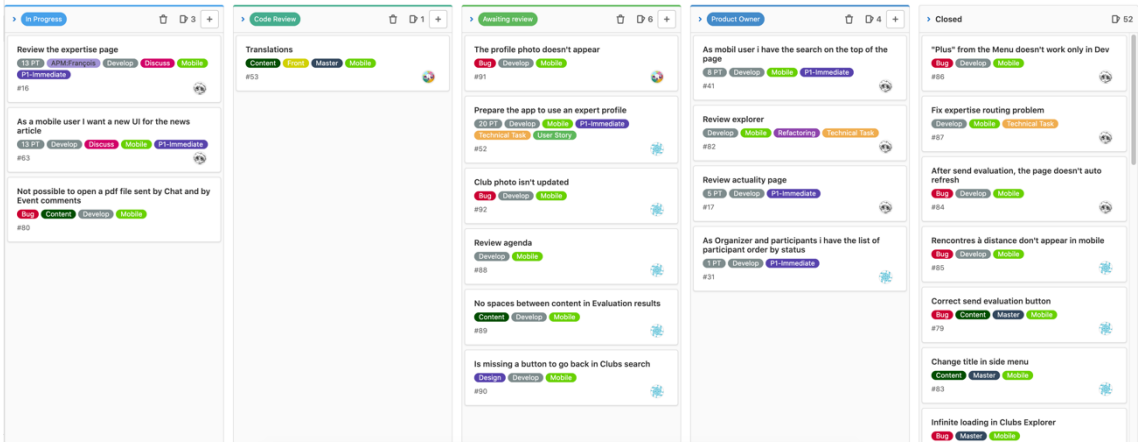


Figura 11 - Imagem do quadro *Kanban* do *GitLab* usado na *md3*

O *GitLab* tem como base o sistema de controlo de versões *Git*[21], complementando-o com as suas ferramentas. Através do *Git* é possível fazer um desenvolvimento em torno de *branches*, cópias que divergiram do programa principal e permitem que o desenvolvimento possa continuar sem interferir com a versão original do código. Internamente, na *md3*, é utilizado *GitFlow* para complementar o processo.

Uma das vantagens do uso de *Git* é que este permite assegurar que o trabalho realizado por um elemento da equipa não interfere com o dos outros, justamente, devido ao conceito de *branches*. No caso deste projeto, o código que se encontra em produção está presente num *branch* denominado de *Master*, onde está presente o código mais estável e seguro. Existe um outro *branch* chamado de *Develop*, utilizado para acrescentar novas funcionalidades ou atualizações, evitando a interferência com o código em *Master*. Deste modo, é assegurado que existe sempre uma função funcional do código. No final de cada *Sprint*, o trabalho realizado em *Develop* poderá ser, após validado, inserido em *Master*. É importante referir que é comum a criação de *branches* a partir de *Develop* e *Master*, de modo a trabalhar *stories* de forma isolada para que o código produzido possa ser posteriormente inserido no respetivo *branch*.

É também importante referir a existência da plataforma *Redmine*, meio utilizado para gestão de conteúdo relacionado com o suporte dos vários projetos da *APM*.

[21] https://git-scm.com

4.4.5. Ferramentas de teste

Existem ferramentas cujo propósito foi auxiliar o processo de testes da *md3*. A sua utilidade está presente na forma como estas permitem simplificar o processo que teria de ser realizado por *testers*. Estas ferramentas têm uma utilidade que depende do contexto onde se pretende que sejam aplicadas, não existindo uma solução universal aplicável a todos os cenários. Em seguida, são explicadas as principais ferramentas de testes utilizadas neste projeto acompanhadas por uma descrição que justifica o seu uso:

- *TestFlight*[22] – Solução padrão para teste de aplicações *iOS*. Esta plataforma encontra-se presente na central de gestão de aplicações da *Apple*. O seu funcionamento é bastante vantajoso em determinados aspetos, pois integra-se naturalmente no processo de distribuição de uma aplicação. Essencialmente, uma aplicação que esteja no *xcode* pode ser arquivada e enviada diretamente para a central de gestão de aplicações, onde será revista por um técnico da *Apple* e após aprovada poderá ser imediatamente distribuída para o *TestFlight*. *Testers* que pretendam testar a aplicação serão adicionados na plataforma e poderão transferi-la através da *app TestFlight*. Esta solução foi utilizada durante bastante tempo, contudo, veio a ser substituída por uma alternativa que permitiu contornar a verificação obrigatória, exigida pela *Apple,* necessária sempre que uma aplicação era submetida na sua plataforma. Este processo implicava a existência de um período de espera, potencialmente demoroso, que quebrava o fluxo de trabalho. Outro aspeto que levou à substituição do *testflight* é o facto de este só lidar com testes *iOS*, algo que se apresenta como uma desvantagem sendo o projeto lida com uma aplicação híbrida, havendo necessidade de testar a versão da aplicação para *Android*.

- *Firebase App Distribution*[23] – Solução integrada na plataforma *firebase* que permite a distribuição de aplicações a *testers*. A forma como funciona é bastante simples, sendo apenas necessário: arrastar as *builds* da aplicação para a plataforma; fornecer os emails de quem se pretende que teste a aplicação; adicionar notas para acompanhar a distribuição; selecionar a opção que irá efetuar a distribuição da aplicação. Os *testers* irão receber um convite, por email, para testar a aplicação. Neste email

[22] https://testflight.apple.com/
[23] https://firebase.google.com/docs/app-distribution

irão constar instruções com a explicação sobre como instalar a aplicação *App Distribution*, que irá servir como meio de acesso a todas as *builds* a que cada *tester* tem acesso. Estas *builds* podem ser transferidas e instaladas no dispositivo. Esta ferramenta é atualmente a principal plataforma de distribuição da aplicação para motivos de teste, servindo como uma alternativa simples e centralizada para as soluções disponibilizadas tanto pela *Google* como pela *Apple*.

- *Protractor*[24] *– Framework* para testes automatizados do tipo *end-to-end* em *Angular*. Testes deste tipo têm como objetivo testar a aplicação desde o seu ponto inicial até ao cumprimento do objetivo pretendido, de modo a simular o comportamento que um utilizador teria com o uso normal da *app*. Deste modo, é possível testar toda a solução no seu fluxo normal, testando a qualidade da integração entre todos os componentes. No caso desta ferramenta, foram escritos vários testes em *JavaScript* onde são especificadas ações a realizar sob o programa como *clicks* em determinados botões, escrita dentro de *inputs* da interface, etc. Os testes escritos com esta ferramenta terminavam com a verificação do resultado esperado. Resultante do seu uso, foi possibilitada a existência de um processo de testes automaticamente reproduzíveis e com uma vasta cobertura da app.

- *Unittest*[25] *– Framework* para automatização de testes unitários em *python*. Testes unitários são testes de baixo nível que tem como objetivo testar métodos ou outros componentes do código base de forma isolada. Esta ferramenta foi utilizada tanto para *testing* como para o *spiking* de funcionalidades na solução de leitura de presenças. Essencialmente, é necessário invocar a biblioteca do *unittest* e criar casos de teste programaticamente, onde será executado o código necessário e serão determinadas condições a serem verificadas quando executados os testes.

Estas foram as principais ferramentas utilizadas na realização dos testes referentes a este projeto. Foi realizado ainda o *spiking* de outras ferramentas de testes como o *TestFairy*, *Selenium* e *Apptim*, contudo nenhuma dessas acabou por ser mais vantajosa para este projeto quando efetuada a comparação com as soluções que acabaram por ser

[24] https://www.protractortest.org/#/
[25] https://docs.python.org/3/library/unittest.html

utilizadas. Em síntese, existem diversas ferramentas que permitem facilitar o processo de testes e que se provaram indispensáveis para o desenvolvimento do projeto.

5. Desenvolvimento

Este capítulo irá descrever o projeto realizado, detalhando também o contexto em que se integra, a arquitetura do sistema, os módulos com que interage, o processo de desenvolvimento e os resultados obtidos.

O objetivo deste trabalho era a criação de um *POC* onde seria analisada uma imagem de uma folha de presenças da *APM* de modo a devolver os seus conteúdos. A informação resultante poderá depois ser utilizada pela empresa, possibilitando o registo de presenças na *API* e consequentemente a disponibilização desses dados a todos os sistemas.

Com este projeto foi possível desenvolver um *package* em *python*, instalável na *API*, capaz de recolher a informação relativa às presenças num evento com base nas assinaturas contidas na imagem da respetiva folha de presenças. Foi também desenvolvida a funcionalidade que permite a integração deste mesmo módulo com a aplicação, sendo esta a fonte da imagem a ser analisada.

Pelo facto de ser necessário o trabalho nos distintos ambientes da aplicação *mobile* e da *API*, foi necessária a familiarização com as diferentes tecnologias que estes integram. Este processo encontra-se descrito nas secções 3.1.3 e 3.1.4. Foram desenvolvidos vários pequenos programas com a intenção de aprender o funcionamento das diversas ferramentas que se encontram expostas na secção 4.4.

Neste capítulo será explicado na secção 5.1 o funcionamento da *APM*. Na secção 5.2 é explicada a arquitetura do sistema onde se insere a solução. Na secção 5.3 é explicado o funcionamento do *Back Office* e na secção 5.4 do *Front Office*. Por fim, na secção 5.5 encontra-se descrito o processo de desenvolvimento da solução para leitura de presenças.

5.1. Funcionamento da APM

Antes de dar início à explicação do desenvolvimento deste projeto, é necessário expor algum conhecimento relativo à *APM*, pois o trabalho realizado foi feito com base nos seus processos internos.

A missão da *APM* é a criação de uma rede de executivos de negócios que pretendem desenvolver os seus conhecimentos e capacidades de empreendedorismo. Ao associarem-se à empresa, os clientes da *APM* poderão desempenhar vários papeis, sendo os relevantes para este projeto os de aderente, organizador e de *expert*. Um aderente é o tipo mais básico de membro. Este, irá ser associado a um clube (um conjunto de membros) compatível com a região onde reside. Cada um destes clubes tem um organizador, cuja principal função é a gestão de eventos. Através do organizador, os membros de um clube têm a possibilidade de ir aos eventos onde poderão desenvolver os seus conhecimentos e capacidades. Sempre que um organizador pretende criar um evento, este terá de escolher uma intervenção, que é essencialmente um tema e um planeamento que poderá ser associado a um evento. O responsável por estas intervenções é o *expert*, que desempenha o papel de orador nos eventos onde irá intervir.

O que foi explicado retrata de forma simplificada aquilo que é o processo de gestão de eventos da *APM*. Encontra-se em constante evolução, estando hoje implementadas bastante mais funcionalidades e papeis de utilizador. Contudo, foi descrito aquilo que é importante para este projeto, sendo que a funcionalidade desenvolvida poderá ser integrada no processo após a conclusão de um evento, estando apenas acessível aos responsáveis pelos eventos de um clube, os organizadores.

5.2. Arquitetura

Há vários anos que a *md3* ajuda a *APM* no desenvolvimento do seu sistema de *software*, criando e evoluindo de forma contínua os vários componentes que o definem, assim, surge esta secção relativa à arquitetura desse mesmo sistema.

A *APM* disponibiliza os seus serviços por meio de vários componentes, incluindo os seus diversos *websites* e até mesmo a sua aplicação *mobile*. Destacam-se dois *websites* principais, denominados de *front office* e *back office*. O primeiro é a principal plataforma da empresa, onde os utilizadores têm acesso aos vários serviços da empresa, sendo que o segundo é uma plataforma de gestão de dados destinada à administração. Ambos *websites*, assim como a *app*, comunicam diretamente com a *API* desenvolvida pela *md3*. As principais características desta *API* são o seu design *REST* e integração com *elasticsearch* e *PostGres* para armazenamento de dados.

O trabalho realizado para este projeto incide na aplicação *mobile* e *API*, sendo possível observar na Figura 12 a forma como ambos componentes se integram com os restantes, de modo a criar o ecossistema de aplicações da *APM*.

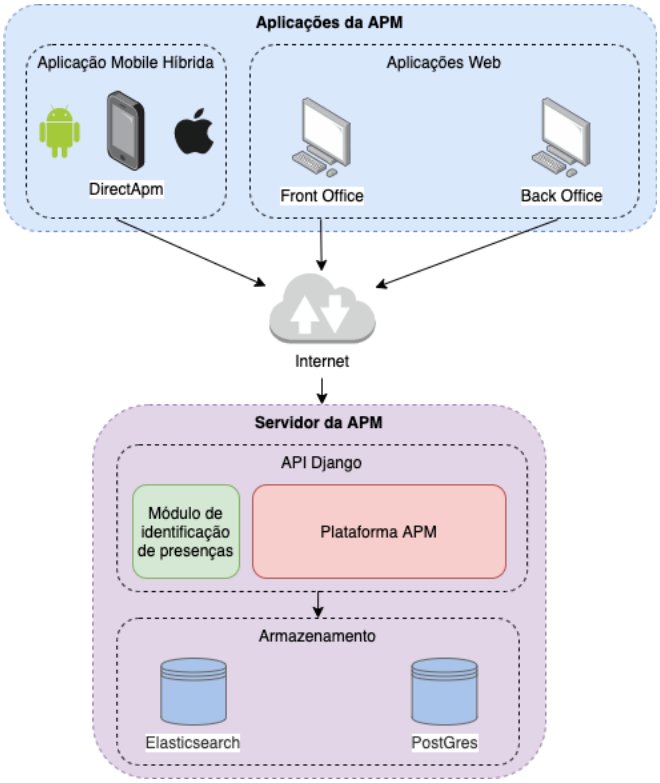


Figura 12 - Arquitetura do sistema APM

5.3. Back Office

Pretende-se com esta secção dar a conhecer as principais funcionalidades da plataforma de administração da *APM*. Por lidar com dados sensíveis de alta importância, o seu acesso é

limitado encontrando-se reservado a utilizadores que possuam o papel de permanente. Contudo, existe ainda outro nível de restrições em que a condicionante é o subtipo de permanente, indicativo da sua função na *APM* ao nível de cargos de gestão, financeiros, etc. Existem várias funcionalidades na plataforma, sendo possível agrupá-las pelo tipo de dados que tratam, descritos em seguida:

- Utilizadores – Existe a possibilidade de pesquisar por utilizadores, sendo disponibilizada uma lista com a capacidade de ser filtrada. É possível editar qualquer perfil nomeadamente ao nível do estado da conta e papel do utilizador.
- Ações – Existe a possibilidade de gerir as ações que ocorrem dentro das várias plataformas. Estes elementos podem ser, por exemplo, a transferência de um documento, um pagamento, a desativação de uma conta, etc. Existe um histórico onde podem ser consultadas todas as ações.
- Alertas – Os alertas das plataformas podem ser geridos a partir do *back office*, podendo ser aceites ou rejeitados. Um alerta é uma situação que se encontra pendente dentro dos processos do sistema e pode ser, por exemplo, o estado pendente da aceitação de um candidato a um evento.
- Conteúdo – O conteúdo mais geral disponibilizado nas outras plataformas é também editável. Dentro deste domínio inserem-se os dados dos clubes, dos eventos, das notícias, das intervenções, do conteúdo apresentado no *front office*, etc. Existe ainda a possibilidade de realizar ações em larga escala, como o envio de emails e a solicitação de intervenções para eventos.
- Relatórios – Existem vários documentos relativos aos vários processos internos da *APM* como, por exemplo, relatórios de eventos e estatísticas globais. É possível o acesso e gestão destes documentos a partir do *back office*.

Apesar do back office não ter sido trabalhado durante este projeto, a compreensão do seu papel no ecossistema da *APM* é essencial. Na possibilidade de uma futura implementação do *POC* de deteção de presenças com esta plataforma, poderia ser implementada a capacidade de permitir a um gestor a análise dos seus resultados, ou até mesmo o ajuste dos parâmetros do algoritmo de forma a otimizar as suas leituras.

5.4. Front Office

Pretende-se com esta secção dar a conhecer as principais funcionalidades da plataforma *front office*, o principal meio de interação entre clientes e sistema da *APM*. Disponível a todos os membros, a aplicação disponibiliza as suas funcionalidades e conteúdo com base no papel de cada utilizador. Contudo, existem páginas acessíveis de forma geral, como por exemplo o perfil, o *blog* e a pesquisa pública, onde é possível procurar por utilizadores, vídeos, intervenções, etc.

A explicação desta plataforma é importante, para auxiliar no entendimento dos objetivos a alcançar com a *app mobile*, pretendendo-se que esta evolua até que passe a incluir as mesmas funcionalidades que existem no *front office*. Apesar das suas imensas características, é especialmente importante para este projeto compreender como é feito atualmente o processo de registo de presenças no evento.

Um evento existe associado a um clube, sendo que serão os membros desse mesmo clube os participantes nesse evento. O acesso a clubes é restrito a aderentes e organizadores, sendo que os primeiros apenas têm acesso a interações básicas com o clube e os segundos têem privilégios de gestão. Por exemplo, um aderente tem a capacidade de realizar determinadas ações como consultar o seu próprio perfil e confirmar a sua presença ou ausência num evento. Contudo, um organizador tem a capacidade de criar os eventos desse clube, solicitar intervenções, convidar membros do clube para eventos, etc.

Após a conclusão de um evento o organizador terá que ir ao *front office,* onde fará o *upload* da folha de presenças e a seleção manual dos convidados que estiveram presentes no evento, ficando esta informação registada internamente. São necessárias várias etapas e intervenções do organizador para confirmar as presenças num dos seus eventos. Espera-se que o *POC* desenvolvido com este projeto tenha o potencial de reduzir o trabalho atualmente exigido, possibilitando o acesso rápido e portátil à *app* onde poderá ser feito o registo de presenças a partir de uma simples foto ao documento das presenças. Na possibilidade de uma futura implementação deste projeto nesta plataforma, poderá também existir a capacidade executar uma leitura de presenças como resposta ao *upload* da folha de presenças a partir do *front office*.

5.5. Desenvolvimento do módulo para registo de presenças

Nesta secção será descrita a componente do desenvolvimento deste projeto relativa à solução principal, o módulo de identificação de presenças. Necessitando apenas de uma imagem da folha de presenças e com recurso a técnicas de visão computacional, incluindo *OCR*, este é capaz de extrair com alta precisão a informação relativa às presenças dos membros da *APM* em eventos da empresa. Existe sob a forma de *package* de *python* para que possa ser utilizado livremente ao longo do código.

Este módulo insere-se no final do ciclo de vida de um evento. Após criado pelo organizador e associado à intervenção de um *expert*, este necessita que lhe sejam atribuídos participantes. Será da responsabilidade do organizador o envio de convites a membros selecionados do respetivo clube. A partir dos convidados selecionados, será possível gerar a folha de presenças no *front office*, que deverá ser impressa e utilizada no evento presencial. Após concluído o evento, a folha deverá encontrar-se assinada por todos os participantes, para que esta possa ser digitalizada ou fotografada, de modo a possibilitar o seu *upload* numa das plataformas da *APM*. O organizador terá ainda que assinalar manualmente os convidados do evento que assinaram a folha, de modo a poder continuar com outras tarefas relacionadas com avaliações, faturação, etc.

Não existe atualmente uma solução que permita a automatização do processo de confirmação das presenças, sendo necessária a realização do processo manual descrito, que obrigatoriamente irá consumir tempo desnecessariamente e será uma tarefa repetitiva para os organizadores. Por este motivo, surge este projeto que possibilita uma alternativa dinâmica, acessível e simples ao processo tradicional descrito, conseguindo os mesmos resultados em menos tempo e a partir de uma simples fotografia à tabela de presenças.

O funcionamento do *package* desenvolvido encontra-se esquematizado na Figura 13. Destacam-se 5 etapas distintas, começando com a aquisição dos dados, o isolamento da tabela presente na figura, a segmentação da tabela em pequenas partes agrupadas, a identificação dos nomes contidos na tabela e a deteção das respetivas presenças.

Nesta secção será descrito o processo mencionado no parágrafo anterior. Cada etapa será abordada individualmente, começando com a sua explicação geral e seguindo-se os detalhes

da sua implementação. Serão descritas também as tecnologias utilizadas e os resultados obtidos em cada etapa.

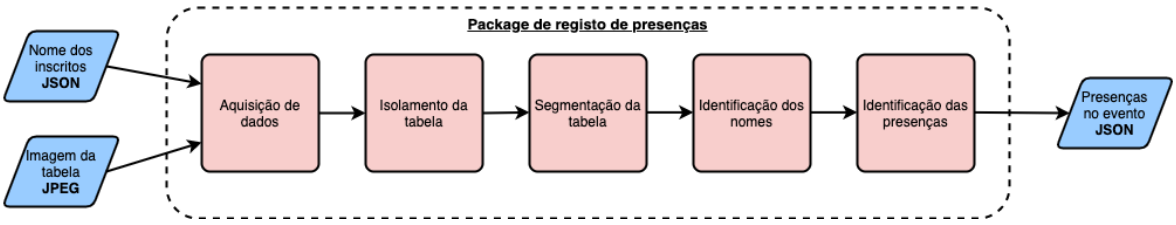


Figura 13 - Esquema do package de registo de presenças

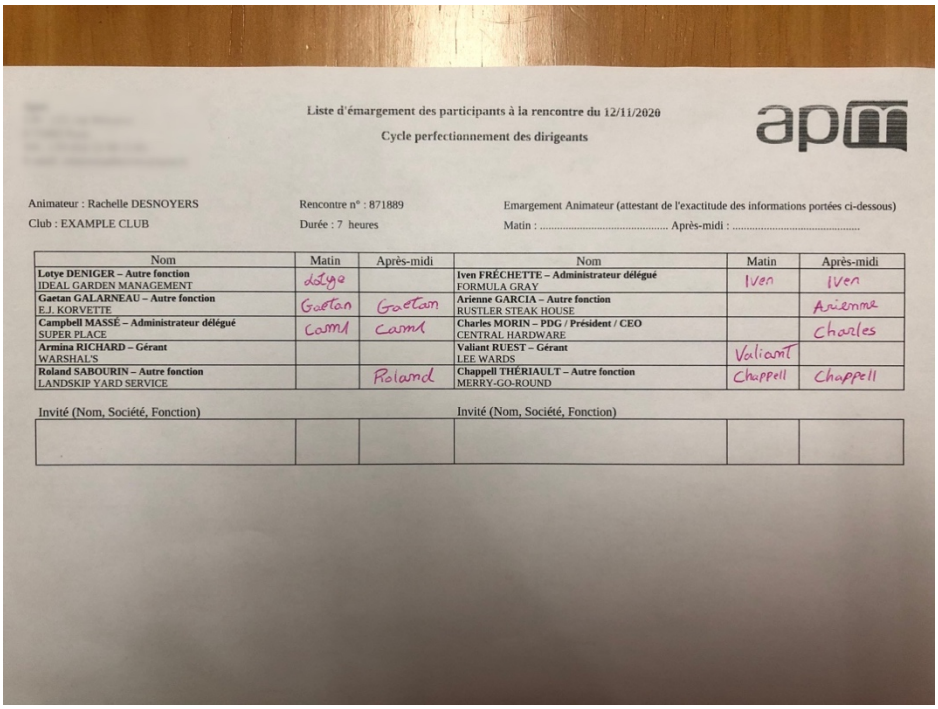
5.5.1. Aquisição de Dados

O *POC* desenvolvido, apesar de ser independente da plataforma *mobile*, foi criado num contexto em que dá continuidade a um processo exclusivo da *app*

Esta funcionalidade destina-se a organizadores, responsáveis pela gestão dos eventos de um clube. Recorrendo à *app*, um utilizador com este papel terá acesso a uma secção onde poderá escolher entre capturar ou submeter uma fotografia da tabela de presenças devidamente assinada pelos participantes. Após a obtenção de uma foto, esta será enviada para a *API* onde o processo terá continuidade.

A imagem é um dos dois parâmetros de caracter obrigatório para o módulo de deteção de presenças, contudo, é o único que recorre à aplicação para ser adquirido. O restante, trata-se de um objeto *json* com os nomes de todos os participantes. Na Figura 14 é possível observar uma fotografia do documento de presenças oficial da *APM*. Este exemplo foi utilizado como base para as demonstrações realizadas ao longo deste capítulo. Os nomes contidos neste documento foram gerados aleatoriamente com recurso ao *Fake Name Generator*[26]. Os dados privados contidos no canto superior esquerdo da imagem foram omitidos após a conclusão da deteção de presenças.

[26] https://www.fakenamegenerator.com/



Liste d'émargement des participants à la rencontre du 12/11/2020

Cycle perfectionnement des dirigeants

apm

Animateur : Rachelle DESNOYERS | Rencontre n° : 871889 | Emargement Animateur (attestant de l'exactitude des informations portées ci-dessous)

Club : EXAMPLE CLUB | Durée : 7 heures | Matin : ..... Après-midi : .....

Nom	Matin	Après-midi	Nom	Matin	Après-midi
Lotye DENIGER – Autre fonction IDEAL GARDEN MANAGEMENT	Lotye		**Iven FRÉCHETTE – Administrateur délégué** FORMULA GRAY	Iven	Iven
Gaetan GALARNEAU – Autre fonction E.J. KORVETTE	Gaetan	Gaetan	**Arienne GARCIA – Autre fonction** RUSTLER STEAK HOUSE		Arienne
Campbell MASSÉ – Administrateur délégué SUPER PLACE	Camd	Camd	**Charles MORIN – PDG / Président / CEO** CENTRAL HARDWARE		Charles
Armina RICHARD – Gérant WARSHAL'S			**Valiant RUEST – Gérant** LEE WARDS	Valiant	
Roland SABOURIN – Autre fonction LANDSKIP YARD SERVICE		Roland	**Chappell THÉRIAULT – Autre fonction** MERRY-GO-ROUND	Chappell	Chappell

Invité (Nom, Société, Fonction)			Invité (Nom, Société, Fonction)		

Figura 14 - Fotografia de uma folha de presenças assinada

5.5.1.1. Implementação da Aquisição de Dados

No início deste projeto foram utilizadas imagens produzidas com recurso a *software* de edição de imagem. O intuito era o de simular tabelas num cenário ideal, sem a presença de ruído, distorção ou outras imperfeições comuns em fotografias capturadas manualmente. Mais tarde, foram realizados testes com fotografias reais, capturadas com a aplicação desenvolvida e com folhas de presenças resultantes de eventos da *APM*. Foram conduzidos vários testes, documentados na secção 6.4.2, que possibilitaram a determinação dos melhores parâmetros a adotar na captura de imagens, de forma a obter os melhores resultados na identificação das presenças.

A tabela contida na imagem tem de respeitar um formato pré-estabelecido pela *APM*. Cada linha da tabela terá de ser composta por itens de três colunas, sendo que cada item diz respeito a um participante no evento. Na primeira estará a identificação da pessoa, composta pelo seu nome, sobrenome, nome da sua empresa e cargo na empresa. A segunda coluna destina-se à sua assinatura correspondente ao período da manhã, enquanto que a terceira corresponde ao período da tarde. Por este formato ser universal, a solução construída foi desenvolvida com o objetivo de tratar tabelas com esta estrutura.

É importante referir que, apesar de estar imposta esta regra estrutural, o módulo de leitura de presenças foi desenvolvido para ser escalável. Ou seja, se cada linha cumprir com as regras referidas, não existe teoricamente um limite para o número de linhas que o módulo consegue interpretar. Também não existe limite teórico para as sequências de itens de utilizadores que se encontrem na mesma linha, desde que estas cumpram com o formato estabelecido. Na Figura 15 é possível observar um exemplo de uma estrutura de tabela de presenças que cumpre com as regras do programa, contendo 5 linhas cada uma com 2 itens em sequência.

Nome	Manhã	Tarde	Nome	Manhã	Tarde
Nome SOBRENOME - Cargo na Empresa NOME DA EMPRESA			**Nome SOBRENOME - Cargo na Empresa** NOME DA EMPRESA		
Nome SOBRENOME - Cargo na Empresa NOME DA EMPRESA			**Nome SOBRENOME - Cargo na Empresa** NOME DA EMPRESA		
Nome SOBRENOME - Cargo na Empresa NOME DA EMPRESA			**Nome SOBRENOME - Cargo na Empresa** NOME DA EMPRESA		
Nome SOBRENOME - Cargo na Empresa NOME DA EMPRESA			**Nome SOBRENOME - Cargo na Empresa** NOME DA EMPRESA		
Nome SOBRENOME - Cargo na Empresa NOME DA EMPRESA			**Nome SOBRENOME - Cargo na Empresa** NOME DA EMPRESA		

Figura 15 - Estrutura de tabelas de presenças

Relativamente ao segundo parâmetro, este também é de caracter obrigatório e deverá ser um objeto do tipo *json* que contenha a identificação de todos os participantes cujo nome esteja presente na folha de presenças. Esta lista é importante, pois permite elevar a precisão do processo de correspondência entre as presenças identificadas e os respetivos utilizadores. É também utilizada como forma de otimização de resultados. No Código 1 encontra-se exemplificado o objeto correspondente à tabela da Figura 15.

```
{
  "0":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "1":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "2":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "3":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "4":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "5":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "6":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "7":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "8":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA",
  "9":"Nome SOBRENOME - Cargo na Empresa\nNOME DA EMPRESA"
}
```

Código 1 - Objeto JSON com participantes

5.5.2. Isolamento da Tabela

Esta etapa tem início após o envido da imagem para a *API*, sendo o seu objetivo a extração da tabela contida na fotografia. Esta preparação é importante pois permite isolar e, consequentemente, padronizar a informação relevante para as restantes etapas do módulo e seus algoritmos.

São imensas as características que podem variar entre imagens. Estas podem ser estruturalmente diferentes, possuindo mais ou menos colunas e linhas, ou até mesmo variar em termos de qualidade. É importante ter em conta que a fonte destas imagens será, na maioria dos casos, a câmara do dispositivo do utilizador. Por este motivo é esperada uma grande variedade entre fotografias, podendo ocorrer casos em que a folha de presenças está demasiado afastada da câmara, inclinada, cortada, etc. Tudo isto impossibilita a existência de uma solução que simplesmente segmenta a imagem em pontos pré-definidos.

Surge esta etapa como resposta a este problema, onde através de uma sequência de técnicas de pré processamento será obtida uma nova imagem, cujo conteúdo será a tabela proveniente da fotografia original, transformada de forma a possuir dimensões pré-definidas. Na Figura 16 é possível observar a nova imagem gerada com esta etapa, resultante da sua aplicação sob a Figura 14.

Nom	Matin	Après-midi	Nom	Matin	Après-midi
Lotye DENIGER – Autre fonction IDEAL GARDEN MANAGEMENT	Lotye		Iven FRÉCHETTE – Administrateur délégué FORMULA GRAY	Iven	Iven
Gaetan GALARNEAU – Autre fonction E.J. KORVETTE	Gaetan	Gaetan	Arienne GARCIA – Autre fonction RUSTLER STEAK HOUSE		Arienne
Campbell MASSÉ – Administrateur délégué SUPER PLACE	Casmd	Casmd	Charles MORIN – PDG / Président / CEO CENTRAL HARDWARE		Charles
Armina RICHARD – Gérant WARSHAL'S			Valiant RUEST – Gérant LEE WARDS	Valiant	
Roland SABOURIN – Autre fonction LANDSKIP YARD SERVICE		Roland	Chappell THÉRIAULT – Autre fonction MERRY-GO-ROUND	Chappell	Chappell

Figura 16 - Tabela isolada

5.5.2.1. Implementação do Isolamento da Tabela

O isolamento da tabela a partir da fotografia é possível através da implementação de técnicas de visão computacional. Para este fim, foi utilizado o *OpenCV*, que disponibiliza métodos

cruciais para esta etapa. Em seguida, são abordadas cada uma das transformações que integram este processo.

Começando com o ajuste das dimensões da imagem, esta é ajustada às dimensões pré-definidas de *2000 píxeis x 800 píxeis* de modo a facilitar o restante processamento. Verificou-se após alguns testes que a escala de cada fotografia utilizada variava bastante dependendo do dispositivo utilizado para fazer a captura. Quanto maior esta escala, ou resolução, mais dados teriam de ser processados e consequentemente mais demoroso seria o tempo de execução. Como tal, foram determinadas estas dimensões com base no desempenho que possibilitavam e de forma a manter a estrutura retangular característica da maioria das tabelas de presenças.

Este ajuste é conseguido através do Código 2, onde se aplica o método *resize* pertencente ao *OpenCV*. Esta função irá receber como parâmetros de entrada a imagem original (Figura 14) juntamente com um *tuple* com as dimensões pretendidas, devolvendo uma nova imagem onde as dimensões já se encontram ajustadas (Figura 17).

```
scaled = cv2.resize(original, (2000, 800))
```

Código 2 - Método para ajuste de dimensões

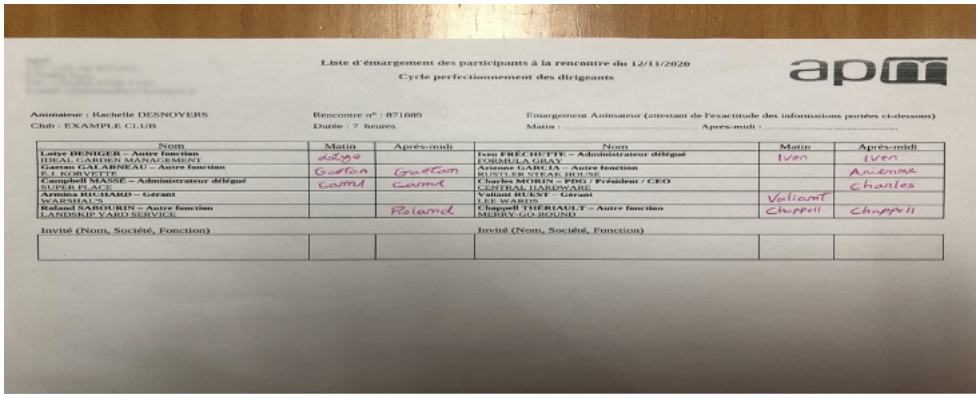


Figura 17 - Fotografia após ajuste de dimensões

Segue-se a segunda transformação, que consiste na conversão de uma imagem a cores para o modelo *grayscale*. Esta alteração facilita o futuro processamento de imagens, exigindo que menos informação tenha de ser processada ao nível dos canais de cores. Em seguida, é explicada de forma breve a teoria por detrás desta conversão.

O elemento mais pequeno de uma imagem são os seus píxeis, e cada um destes tem um significado matemático a si associado de modo a possibilitar a representação da sua cor por uma máquina. O modo como é feita a digitalização de uma cor é determinado pelo modelo de cores adotado [53].

O modelo RGB é utilizado para representar imagens a cores, determinando o valor de cada píxel conforme os valores que lhe tenham sido atribuídos nos espectros vermelho, verde e azul [54]. Imagens *grayscale* são compostas por tons de preto e branco, sendo a cor de cada píxel determinada por um único valor num único canal de cores. Este canal é composto por uma escala com valores entre 0 e 255, sendo 0 a cor preta e 255 a cor branca.

O *OpenCV* disponibiliza o método presente no Código 3, que permite a conversão de uma imagem RGB para *grayscale* através da equação $Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$. Deste modo, os valores atribuídos aos canais de cores RGB de cada píxel, são convertidos para um único valor na escala *grayscale*. O resultado da função mencionada será uma nova imagem (Figura 18) onde é aplicado o modelo *grayscale*.

```
gray = cv2.cvtColor(scaled, cv2.COLOR_BGR2GRAY)
```

Código 3 – Método para conversão para grayscale

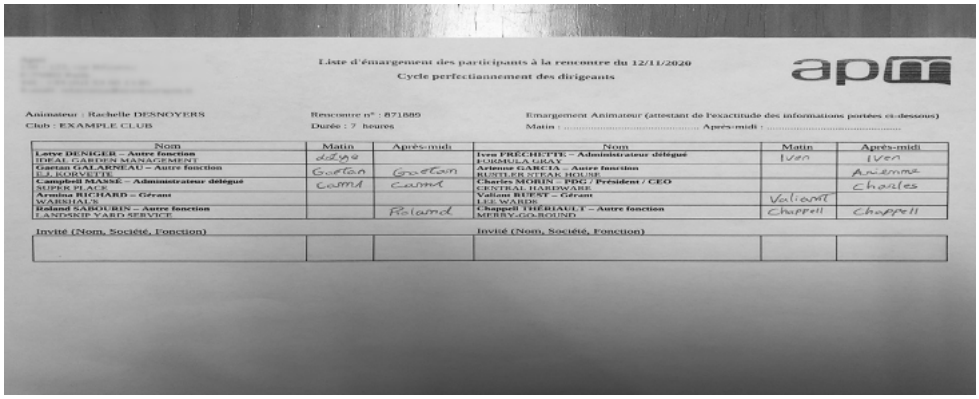


Figura 18 - Fotografia após conversão para grayscale

A terceira transformação realizada nesta etapa é a aplicação de *Gaussian blur* em toda a imagem, resultando num efeito de desfoque. Esta técnica foi aplicada devido à sua utilidade na filtragem de ruído [55]. Dependendo da gravidade, a presença de ruído numa imagem poderá implicar a perda de qualidade nos resultados finais.

Esta técnica recorre a um *kernel* (uma área de píxeis) como base para o desfoque. Essencialmente, o valor do píxel que se encontra no centro do *kernel* será alterado com base nos valores dos restantes píxeis contidos na área.

Para este efeito é utilizado o método presente no Código 4, fornecido pelo *OpenCV*. Este recebe como parâmetros a imagem alvo, o *kernel* (representado por um *tuple* com as suas dimensões) e o desvio padrão, que por estar a 0 é internamente calculado a partir do *kernel*. A imagem devolvida por esta função encontra-se representada na Figura 19.

```
Proc = cv2.GaussianBlur(img.copy(), (9, 9), 0)
```

Código 4 - Método de Gaussian Blur

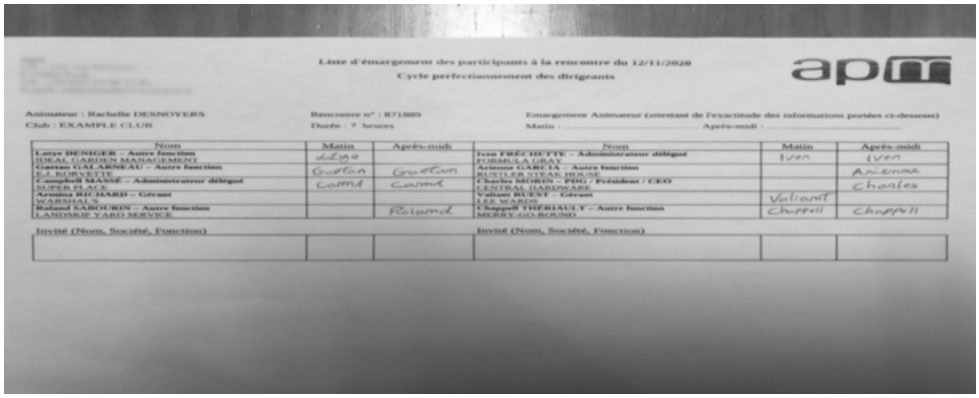


Figura 19 - Fotografia após Gaussian Blur

A quarta transformação aplica uma técnica de *thresholding* sob a imagem, fazendo com que cada píxel da imagem *grayscale* passe a ter um valor binário. Esta alteração é importante para o processo de segmentação, revelando as principais características das imagens [56]. Por este motivo, o uso desta técnica é necessário como meio para dar relevo à tabela, consequentemente facilitando a sua deteção.

Existe um valor, denominado de *Threshold*, que irá representar um ponto na escala de *grayscale*. Se o valor de um píxel da imagem ultrapassar o *Threshold*, ser-lhe-á atribuído uma nova cor. Aos restantes píxeis que se encontrem abaixo do *Threshold*, estes irão ficar com a cor oposta. Na sua forma mais simples, esta técnica assume um *Threshold* global para

todos os píxeis, contudo, existe o conceito de *thresholding* adaptativo que dinamiza este valor ao longo da imagem.

Foi utilizado o método do *OpenCV* descrito no Código 5 para este efeito. É adaptativo, pelo que o valor de *Threshold* é determinado pela área que rodeia cada píxel. Esta função recebe como parâmetros a imagem *grayscale*, o valor atribuído aos píxeis que satisfaçam o *Threshold*, o método adaptativo a utilizar (foi utilizado um baseado no princípio *Gaussian*), o tipo de *thresholding* a utilizar (foi utilizado o mais básico disponível), o valor da área para determinar o *Threshold* e uma constante para subtrair à soma dos píxeis da área. A imagem devolvida pela função encontra-se na Figura 20.

```
proc = cv2.adaptiveThreshold(proc, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 2)
```

Código 5 - Método de Thresholding Adaptativo

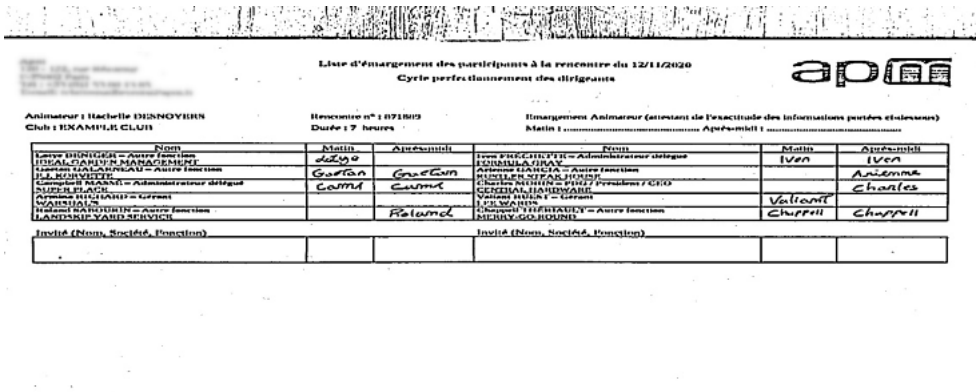


Figura 20 - Fotografia após aplicado Thresholding Adaptativo

A quinta fase de transformação aplica uma técnica de dilatação, resultando no destaque da tabela dentro da imagem. Esta operação classifica-se como morfológica por fazer as suas alterações com base em informação geométrica e através de um *structuring element* [57]. Este elemento pode ser representado por uma matriz binária (*kernel*), sendo o seu tamanho e forma as características que irão determinar a imagem resultante. Optou-se pelo uso de uma técnica morfológica devido à sua utilidade na segmentação de imagens [58]. Contribui para o objetivo desta etapa destacando os contornos da tabela, tornando-a mais detetável.

Pretende-se com a dilatação a expansão de determinadas regiões da imagem com base no *structuring element*. Para aplicar esta transformação foi utilizado o Código 6, que recorre a

dois métodos do *OpenCV*. O primeiro (*cv2.bitwise_not*) apenas inverte as cores da imagem e é necessário pois o segundo método (*cv2.dilate*) aplica a operação morfológica sob os píxeis brancos da imagem. É também criado nesta função o *structuring element*. A imagem resultante desta operação está presenta na Figura 21.

```
proc = cv2.bitwise_not(proc, proc)

kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]], np.uint8)
proc = cv2.dilate(proc, kernel)
```

Código 6 - Métodos para aplicar dilatação à imagem

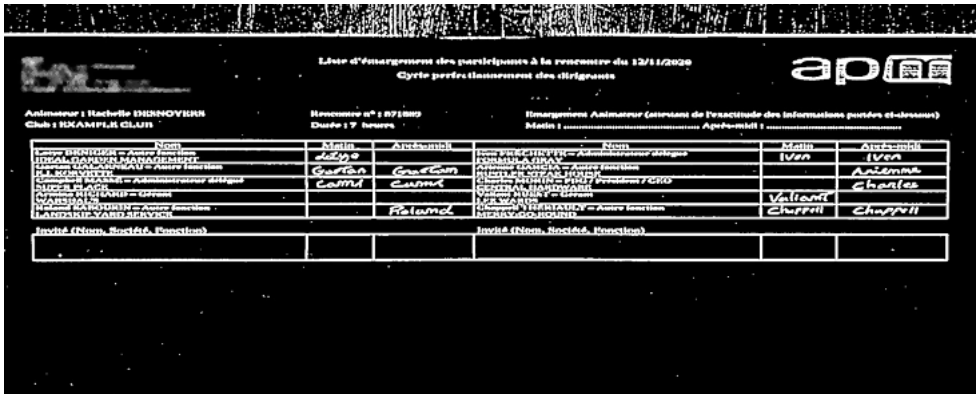


Figura 21 - Fotografia após aplicada inversão de cores e dilatação

A sexta transformação é a penúltima desta etapa e consiste na identificação do contorno da tabela, seguida da extração das suas quatro extremidades. Para este efeito é utilizado o método *findContours* do *OpenCV*, que aplica o algoritmo [59].

De acordo com *Bradsky* e *Kaehler* [60] esta função irá analisar toda a imagem, identificando os contornos nela existentes a partir das áreas brancas. Em *OpenCV* um contorno é definido como sendo uma sequência de pontos de uma curva, em que cada ponto contém informação relativa ao próximo. Destacam-se dois tipos de contorno, o exterior (que traça o perímetro exterior de uma área branca detetada) e o interior (que traça o perímetro do "buraco" preto dentro dessa mesma área branca).

No Código 7 encontra-se representada a forma como foi utilizado este método como princípio para a deteção dos pontos extremos da tabela. Este excerto de código é descrito em seguida.

Relativamente à função *findContours*, esta, primeiramente, este recebe a imagem alvo. Depois, um parâmetro a indicar que apenas deverão ser devolvidos os contornos exteriores. Por último, um valor a indicar que o resultado da função deverá devolver apenas as arestas dos contornos identificados. É devolvido um *array* com todos os contornos detetados na imagem, que será ordenado de modo a ficar o maior contorno na posição 0 do *array*. Em seguida, é guardado o maior contorno encontrado, pois este deverá corresponder à tabela das presenças. São então calculadas e devolvidas as extremidades do polígono. Estas extremidades encontram-se representadas a vermelho na Figura 22, traçadas programaticamente.

Mais tarde foram realizados vários testes com a solução final onde foram analisados documentos de tabelas digitalizadas e fotografias de tabelas. Verificou-se que apesar da existência de uma margem de erro, na maioria das situações o maior correspondia de facto à tabela de presenças.

```
contours, h = cv2.findContours(img.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key=cv2.contourArea, reverse=True
polygon = contours[0]

bottom_right, _ = max(enumerate([pt[0][0] + pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))
top_left, _ = min(enumerate([pt[0][0] + pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))
bottom_left, _ = min(enumerate([pt[0][0] - pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))
top_right, _ = max(enumerate([pt[0][0] - pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))

return [polygon[top_left][0], polygon[top_right][0], polygon[bottom_right][0],
polygon[bottom_left][0]]
```

Código 7 - Extração de extremidades a partir do método findContours

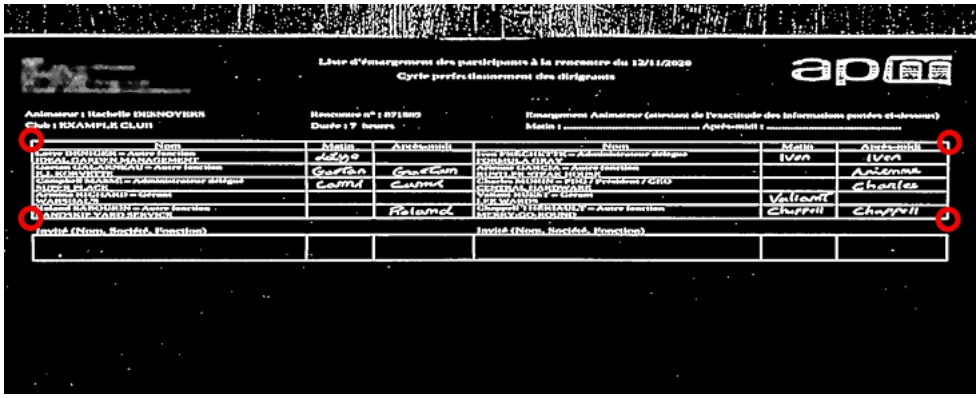


Figura 22 - Contornos da tabela detetados na fotografia

A etapa de isolamento termina com uma operação de transformação de perspetiva, que irá permitir a obtenção uma nova imagem que irá conter apenas a tabela. Este tipo de operação possibilita que uma área da imagem original possa ser enquadrada numa outra área definida. De modo a implementar esta técnica, foi utilizada a função *warpPerspective* do *OpenCV* em conjunto com os seus métodos complementares, retratados no Código 8. Inicialmente, são criados dois *arrays*. O primeiro contém as quatro extremidades previamente detetadas da tabela, enquanto que o segundo contêm as dimensões que se pretendem atribuir à imagem. As dimensões escolhidas para o segundo *array* são representativas de uma imagem de 2000 píxeis *x* 800 píxeis, idênticas às atribuídas na primeira transformação desta etapa. Contudo, é acrescentado um espaçamento extra de 5 píxeis, como medida para prevenir que linhas da tabela sejam cortadas na imagem final. Depois deste processo, é aplicado o método *getPerpectiveTransform*, que permite calcular a matriz que mapeia a transformação do primeiro *array* para o segundo. Por fim, é aplicad o método *warpPerspective* que irá receber como parâmetros a imagem original (representada na Figura 17), a matriz de transformação e o tamanho pretendido para a imagem final (2000 *x* 800). A Figura 16 será a imagem resultante desta etapa, contendo em si apenas a tabela de presenças e dando por concluído este processo.

```
src = np.array([top_left, top_right, bottom_right, bottom_left],
dtype='float32')

dst = np.array([[5, 5], [2000 - 5, 5], [2000 - 5, 800 - 5], [5, 800 - 5]],
dtype='float32')

m = cv2.getPerspectiveTransform(src, dst)

return cv2.warpPerspective(img, m, (2000, 800))
```

Código 8 - Método para transformação de perspetiva

5.5.3. Segmentação da Tabela

Estando concluído o isolamento da tabela, será agora necessária a intervenção de um processo que permita a organização da informação presente na imagem. Surge então esta etapa com o objetivo de segmentar a tabela e agrupar logicamente as partes resultantes, possibilitando a futura análise individual de cada célula.

Apesar de ser imposta uma estrutura sob os elementos da tabela, não existe teoricamente limite para o número de linhas ou colunas que esta possa ter. Por este motivo, foi necessária a elaboração de uma solução dinâmica, capaz de se adaptar aos diferentes tipos de tabela possivelmente utilizados pela *APM*.

Serão descritas nesta secção as várias técnicas que possibilitaram o sucesso desta etapa, começando com a explicação de uma solução original denominada *Margin Partol* (*MP*), que através dos seus métodos internos consegue identificar o esqueleto da tabela. Este, por sua vez, é tratado e utilizado para fazer a divisão da tabela pelas suas células. Estas células serão posteriormente agrupadas, de modo a que cada grupo represente um item com a estrutura referida inicialmente (identificação do participante, assinatura do período da manhã, assinatura do período da tarde).

O resultado desta etapa será um dicionário *python*, composto por vários objetos que representam um utilizador da tabela. Cada um destes objetos contém as três respetivas imagens, encontrando-se alguns representados na Figura 23.

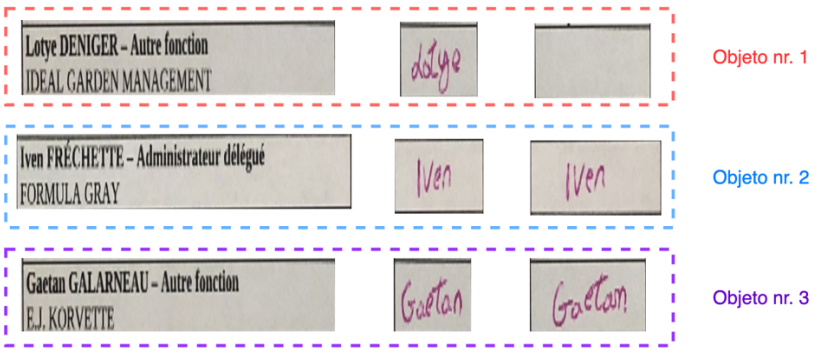


Figura 23 - Exemplos de objetos criados a partir da segmentação da imagem

5.5.3.1. Implementação da Segmentação da Tabela

Pretende-se com esta secção a divisão da fotografia recebida, para que cada uma das células da tabela passe a ser a sua própria imagem. Para que tal seja possível, é primeiro necessária a identificação da estrutura da tabela. Foi feita uma extensa pesquisa de modo a adquirir o conhecimento necessário para realizar a operação pretendida, na qual foram estudadas e testadas várias hipóteses, nomeadamente a técnica *Hough Transform*. Contudo, nenhum dos

métodos mencionados superou em termos de qualidade a solução *MP*, desenvolvida especificamente para este projeto.

Nesta secção é explicado o processo por detrás da etapa de segmentação da imagem, detalhando-se a solução *MP*, o tratamento dos dados devolvidos e o seu uso na divisão da tabela em grupos de pequenas imagens.

5.5.3.1.1. Margin Patrol

A solução *Margin Patrol* (*MP)* é uma classe desenvolvida como resposta à instabilidade dos métodos anteriormente testados, procurando responder de forma específica ao problema em causa. A sua abordagem consiste em percorrer os limites da imagem fornecida, procurando por pontos que correspondam a interseções entre linhas da tabela. É estabelecido um parâmetro denominado *search*, que indica a distância a partir do limite da imagem em que se pode procurar por pontos. Por este motivo, é possível afirmar que o *MP* faz a sua análise sob as margens de uma imagem, justificando o nome da solução.

Inicialmente tinha sido testada uma solução baseada em *Hough Transform* [61]. Aquilo que era inicialmente pretendido era a deteção das linhas que compõem a tabela, de modo a determinar os seus pontos de interseção. Por este motivo, foi escolhida a técnica de Hough Transform, frequentemente utilizada para a identificação de linhas em imagens [62]. Encontra-se disponível no *OpenCV* um método que permite aplicar esta técnica, denominado de *HoughLinesP*. Foram realizados vários testes em que se testaram as capacidades para deteção de linhas deste algoritmo sob imagens feitas a computador e fotografias, contudo, os resultados provaram-se insatisfatórios. Mesmo sob imagens fabricadas, sem presença de ruído e com linhas perfeitamente retas e bem definidas, nunca foi possível detetar corretamente todas as linhas. Não foi encontrada uma combinação de parâmetros para o método do *OpenCV* que fosse capaz de satisfazer os critérios básicos, tendo sido impossível prevenir a ocorrência de erros como: linhas sobrepostas, linhas demasiado curtas, linhas desfasadas, assinaturas detetadas como linhas, etc. Devido à existência de uma enorme margem de erro nas suas leituras, esta hipótese acabou por não ser considerada.

Foi possível observar que a maioria das técnicas testadas falhava devido à presença de assinaturas na tabela, sendo que estas frequentemente eram confundidas por linhas ou pontos de interesse. A identificação deste problema inspirou a criação do *MP* (representado na

Figura 24), tendo-se determinado que seria criada uma solução que, ao contrário das restantes testadas, fizesse a sua análise apenas onde a informação tem a mínima possibilidade de sofrer interferências de assinaturas. Estando implícito o trabalho com uma imagem da tabela isolada, é possível estabelecer uma margem de modo a limitar a análise da imagem, diminuindo a probabilidade de deteção de uma assinatura enquanto se está a definir o esqueleto da tabela.

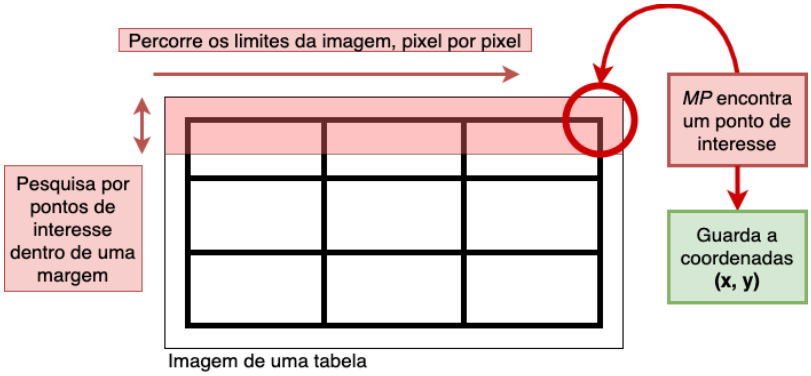


Figura 24 - Funcionamento do Margin Patrol

O primeiro passo para a utilização do *MP* é a sua instanciação no módulo principal, recebendo como único parâmetro de carácter obrigatório a imagem com a tabela. Os restantes parâmetros são *search*, *fuse* e *delete_distance*, que são opcionais por já se encontram com valores pré-estabelecidos, definidos pela sua capacidade geral de obter bons resultados ao longo de várias leituras.

O parâmetro *search* (valor inicial 30 píxeis) representa, como já foi mencionado, o tamanho da margem sob a qual atua esta solução. O parâmetro *fuse* (valor inicial 10 píxeis) indica a distância necessária entre dois pontos para que estes sejam fundidos. O parâmetro *delete_distance* (valor inicial 20 píxeis) tem o propósito de otimizar, as leituras. O funcionamento de cada uma destas variáveis será detalhado consoante o seu uso no código demonstrado.

Após instanciada a classe *MP*, o processo poderá ser iniciado através do método *run*, que contém toda a lógica do código, dividida em 4 fases: pré-processamento, deteção de pontos de interesse, fusão de pontos próximos e limpeza de pontos redundantes.

Começando com a fase de pré-processamento do *MP*, esta tem como principal objetivo a remoção do ruído presente na imagem e o destaque da tabela. Para tal, são utilizadas algumas das mesmas técnicas já explicadas na secção 5.5.2.1, sendo estas *Guassian Blur* e *Thresholding*, resultando na imagem representada na Figura 25. Esta etapa é bastante necessária, pois será determinante da qualidade das leituras efetuadas. Quantos mais elementos, que não sejam a tabela, existirem na imagem, maior será a probabilidade de o *MP* acusar falsos positivos. Foram feitos testes com o intuito de verificar como o pré-processamento impacta as leituras feitas pelo *MP*, estando alguns demonstrados na secção 6.3 deste documento.

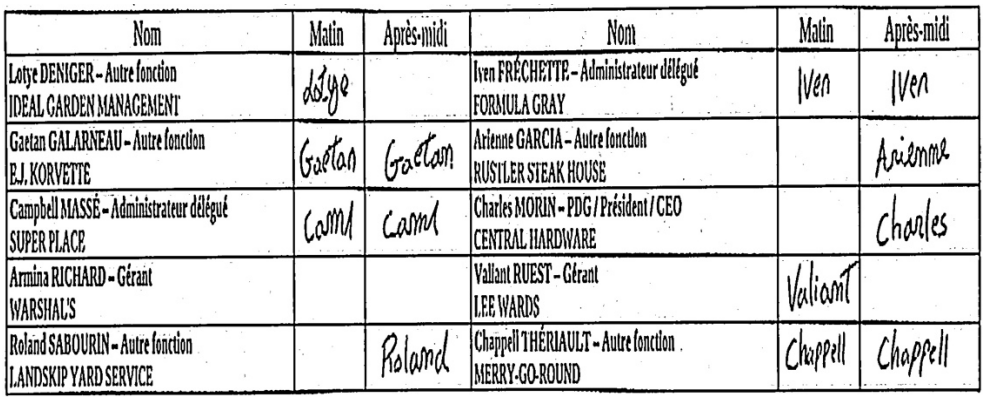


Figura 25 - Pré-processamento aplicado pelo MP

A segunda fase desta solução terá como objetivo a deteção dos pontos de interesse da tabela. Para tal, é chamada a função *__margin_patrol* que executa a lógica necessária para este processo. Uma representação resumida deste método encontra-se no Código 9, onde é feita a análise apenas do limite superior da imagem. Em seguida, é descrito o funcionamento desta solução de modo a possibilitar a compreensão do excerto de código.

Primeiramente, é importante perceber o que são pontos de interesse. Estes, são os píxeis nos limites da tabela que permitem definir as suas células. No contexto do *MP*, que percorre píxel por píxel os limites da tabela, um ponto de interesse será um píxel que dá início a uma linha perpendicular ao limite que está a ser percorrido.

Todos os quatro limites da imagem são analisados, seguindo-se a ordem: topo, fundo, esquerda e direita. Ao píxel que está a ser analisado, dá-se o nome de agente. A função de cada agente será a de procurar por pontos de interesse dentro da margem imposta. Por

exemplo, caso esta margem tenha o valor 30, então o agente irá procurar por uma sequência de pontos pretos que tenha início num dos 30 píxeis diretamente à sua frente.

Existem algumas variáveis que são importantes para este processo: *search*, estabelecido no construtor da classe, que indica o valor da margem a ser analisada; *pixels*, um *array* inicialmente vazio, que será onde os pontos de interesse detetados serão guardados; *height*, que será a altura em píxeis da imagem fornecida; *width*, o comprimento em píxeis da imagem fornecida.

```
def   margin_patrol(self, i):
    """Receives pre-processed image and search distance for agent.
    Agent will search around the image for table characteristics, returning all detected
pixels.

    :param i: image
    :param r: search range

    :return: all detected pixels
    :rtype: array
    """

    r = self.search
    pixels = []
    height = i.shape[0]
    width = i.shape[1]

    # search top
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            # we just want to read the FIRST black pixel we find
            if not found_black_pixel:
                # found a black pixel, let's jump there!
                if i[s, w] == 0:
                    # for now i'll be searching for the same range in front of the pixel
                    found_white_pixel = False
                    range_from_pixel = r
                    for s2 in range(range_from_pixel):
                        y = s + s2 + 1
                        if i[y, w] != 0:
                            found_white_pixel = True
                            break
                    if not found_white_pixel:
                        pixels.append((w, s))
                    found_black_pixel = True
            else:
                break

    ... (here is the analysis of the other limits)

    return pixels
```

Código 9 - Resumo do funcionamento do método __margin_patrol

A imagem atual é composta por píxeis binários devido à aplicação de *Thresholding*, ou seja, cada píxel terá um de dois valores: 0 se for um píxel preto; 1 se for um píxel branco. Este facto serve como base para a deteção de uma linha, sendo que esta será uma sequência de píxeis pretos.

De forma a resumir o seu funcionamento, será explicado o modo como o método *__margin_patrol* faz a análise do limite superior da imagem, sendo que a análise dos restantes limites é bastante semelhante. Começa-se por iterar o valor de *width*, para que o agente possa percorrer cada píxel no topo da imagem. Para cada posição no eixo *x* (*width)*, será iterado o eixo *y*, nos primeiros *range* píxeis. Deste modo, o agente irá posicionar-se em todos os pontos na margem diretamente à sua frente. Caso um dos pontos percorridos tenha o valor 0, então confirma-se que este é preto e, consequentemente, um possível candidato a ponto de interesse. Para cada ponto candidato detetado, o agente irá procurar saber se os *range* píxeis diretamente à sua frente são também pretos. Caso se confirme uma sequência não interrompida de píxeis pretos, então o *MP* considera que encontrou uma linha da tabela, guardado o ponto que deu início à sequência no *array pixels*. Este processo é então repetido para todos os limites da imagem, resultando no armazenamento de todos os pontos de interesse encontrados dentro do *array pixels*. Este *array* é, por fim, ordenado através do método *sort*, de modo a facilitar a próxima etapa.

Após obtidos os pontos de interesse com a segunda fase, dá-se início à terceira, que consiste na fusão de pontos que se encontrem próximos. Este processo é necessário porque são elevadas as hipóteses de existirem pontos de interesse que correspondam à mesma linha da tabela. Por exemplo, caso uma destas linhas tenha uma grossura de 3 píxeis, isto irá resultar na deteção de 3 pontos de interesse. Mais tarde, para fazer a segmentação da tabela, será necessário indicar as coordenadas onde deverá ser cortada a imagem. Por este motivo surge a necessidade desta fase, onde se pretende a união de ponto próximos, transformando-os num único ponto. O método que possibilita a fusão denomina-se de *__fuse* (representado no Código 10) e será explicado em seguida.

As variáveis importantes para este método são: *p*, o *array* de pontos resultante da função *__margin_patrol*; *fuse*, estabelecido no construtor da classe, que indica a distância necessária entre dois pontos para que estes sejam fundidos; *pixels_fused*, um *array* inicialmente vazio que será onde serão armazenados os píxeis já fundidos; *first*, um *boolean* auxiliar que começa sempre a *True*.

Esta função começa com um ciclo que percorre todos os pontos em *p*. Irá utilizar a variável *first* de modo a armazenar o ponto correspondente à primeira iteração em *pixels_fused*. Nas seguintes iterações, o objetivo será comparar o atual ponto de *p* com todos os pontos

presentes em *pixels_fused*. Esta comparação consiste no cálculo da distância entre ambos os pontos, verificando se essa distância é menor ou igual ao valor de *fuse*. Caso a distância seja superior, então o ponto de *p* é simplesmente adicionado a *pixels_fused*. Contudo, caso a distância seja menor, então confirma-se que esses pontos se encontram demasiado próximos, sendo calculado o ponto médio entre ambos. Este novo ponto será colocado em *pixels_fused*, após a remoção do ponto que foi utilizado para o gerar. Da execução desta função irá resultar um *array* contendo os pontos da imagem já fundidos, sem a presença da redundância previamente presente.

```
def __fuse(self, p):
    """Receives detected points and minimum distance between points to consider a
fusion, will then fuse points that are
     close to each other.

    :param p: list of all detected points
    :param dist: minimum distance to fuse points

    :return: distinct points
    :rtype: array
    """

    dist = self.fuse
    pixels_fused = []
    first = True

    for c1 in p:
        if first:
            pixels_fused.append(c1)
            first = False
        else:
            fused = False
            for c2 in pixels_fused:
                if self.__calculate_distance(c1[0], c1[1], c2[0], c2[1]) <= dist:
                    x = math.ceil(((c1[0] + c2[0]) / 2))
                    y = math.ceil(((c1[1] + c2[1]) / 2))
                    pixels_fused.remove(c2)
                    pixels_fused.append((x, y))
                    fused = True
            if not fused:
                pixels_fused.append((c1[0], c1[1]))

    return pixels_fused
```

Código 10 - Método para fusão de pontos próximos

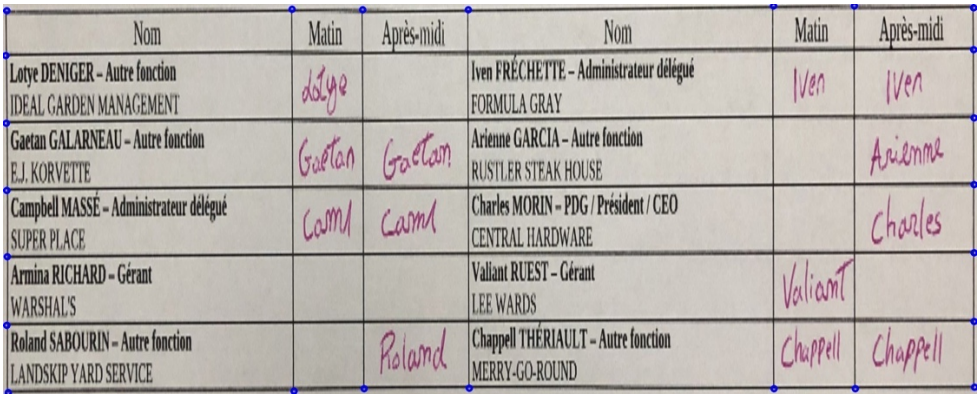
A classe *MP* termina com uma quarta fase de otimização, cuja necessidade surgiu após terem sido realizados vários testes com exemplos de fotografias reais (detalhados na secção 6.4.1). Grande parte desses testes falharam com erros críticos, resultantes da acusação de falsos positivos por parte da função *__margin_patrol*. Após uma análise cuidadosa, foi possível determinar que a causa deste problema era a presença de assinaturas que não cumpriam os limites da tabela.

Uma observação feita que acabou por se tornar a base para a resolução do problema, foi que a maioria dos pontos incorretamente detetados residia no fundo da tabela, na grande maioria

dos exemplos testados. Esta anomalia deve-se ao facto da primeira linha da tabela estar sempre reservada ao cabeçalho, enquanto que a última contém sempre espaços reservados a assinaturas. Idealmente, se existirem 4 pontos de interesse no topo da imagem, devem também existir 4 pontos correspondentes no fundo da imagem. Por este motivo, determinou-se que viria a ser acrescentada esta 4ª fase de otimização ao *MP*, que irá eliminar pontos no fundo da imagem que não tenham correspondência com nenhum ponto no topo da imagem.

Serão criados dois novos *arrays* a partir dos *array* resultantes da fase de fusão. Estes, terão os nomes *bottom_points* e *top_points*, correspondendo respetivamente aos pontos no fundo e no topo da tabela. Os pontos são selecionados para pertencer a estes *arrays* conforme a sua proximidade com o eixo horizontal do ponto de interesse mais acima e mais abaixo da imagem. Caso existam tantos pontos em *bottom_points* como em *top_points*, então a otimização não é feita, sendo devolvido o *array* resultante da fusão. Contudo, caso se verifique diferença, então o processo continua. Existem duas variáveis que serão agora importantes, sendo estas: *delete_distance*, declarado no construtor da classe e com valor que será determinante para a deteção de correspondência entre pontos do topo e do fundo da tabela; *clean_pixels*, *array* com os pontos de interesse resultantes da fusão; *rejected_points*, *array* inicialmente vazio onde serão guardados todos os pontos sem correspondência.

Serão percorridos todos os pontos contidos em *bottom_points*, sendo para cada um destes feita uma comparação com cada um dos pontos em *top_points*. Esta comparação consiste em verificar se a distância entre os dois pontos, no eixo horizontal, é inferior ou igual ao valor de *delete_distance*. Caso um ponto de *bottom_points* não verifique esta condição com pelo menos um ponto de *top_points*, então considera-se que este não tem nenhuma correspondência e o ponto é adicionado ao *array rejected_points*. Após terem sido percorridos todos os pontos de *bottom_points*, resta remover do *array clean_pixels* os pontos que estão também incluídos no *array rejected_points*. Por fim, é devolvido o *array clean_pixels* e dá-se por concluído o processo de identificação da estrutura da tabela. Uma representação dos pontos detetados como resultado deste processo encontra-se na Figura 26, onde os pontos de interesse estão representados a azul.



Nom	Matin	Après-midi	Nom	Matin	Après-midi
Lotye DENIGER – Autre fonction IDEAL GARDEN MANAGEMENT			**Iven FRÉCHETTE – Administrateur délégué** FORMULA GRAY		
Gaetan GALARNEAU – Autre fonction E.J. KORVETTE			**Arienne GARCIA – Autre fonction** RUSTLER STEAK HOUSE		
Campbell MASSÉ – Administrateur délégué SUPER PLACE			**Charles MORIN – PDG / Président / CEO** CENTRAL HARDWARE		
Armina RICHARD – Gérant WARSHAL'S			**Valiant RUEST – Gérant** LEE WARDS		
Roland SABOURIN – Autre fonction LANDSKIP YARD SERVICE			**Chappell THÉRIAULT – Autre fonction** MERRY-GO-ROUND		

Figura 26 - Representação dos pontos detetados com a solução *MP*

5.5.3.1.2. Tratamento dos resultados do *MP* e segmentação

Da análise feita pela classe *MP* irá resultar um *array* contendo os pontos de interesse que definem a estrutura da tabela. Contudo, estes pontos só por si não se encontram no estado ótimo para se realizar a divisão da tabela em partes. Aquilo que é pretendido é a identificação das linhas da tabela, podendo ser representadas por um único valor no eixo *x* ou *y*. Como tal, surge a necessidade deste processo que, através do Código 11, irá extrair dois novos *arrays* a partir da lista de pontos devolvidos pelo *MP*. Estes novos *arrays* serão *x_coordinates* e *y_coordinates*, correspondendo respetivamente às linhas verticais e horizontais da tabela.

Primeiramente, são percorridos todos os pontos devolvidos pelo *MP.* Em cada iteração são guardadas as coordenadas de cada ponto no respetivo *array,* sendo estes organizados através da função *sort* no final de cada ciclo. Desta operação irão resultar as listas ordenadas de todas as linhas identificadas.

Cada um destes *arrays* será posteriormente sujeito à função *delete_close*, que irá realizar uma limpeza em cada lista com a intenção de apagar valores que representem a mesma linha. Esta limpeza é necessária, pois o *MP* irá por natureza detetar tanto o ponto inicial, como ponto final de cada linha. É do mesmo modo importante reconhecer a existência de uma margem de erro entre pontos que deveriam localizar-se no mesmo eixo, o que poderá resultar na existência de pontos que, apesar de representarem a mesma linha, contêm valores diferentes.

Dentro da função *delete_close* serão percorridos todos os valores do *array* em causa e é criado um novo *array* vazio denominado *filtered_array*. Na primeira iteração, é guardado o primeiro valor no novo *array* criado. Contudo, nas seguintes iterações, novos valores apenas serão adicionados se possuírem uma distância dos valores em *filtered_array* superior a um valor pré-determinado, indicando que estão suficientemente afastados de qualquer valor já existente para serem considerados uma linha diferente. Deste modo, são obtidas as listas filtradas com as linhas da tabela, e o processo continua para divisão da imagem.

```
def __get_x_and_y_values(self, pixels):
    """Splits pixel coordinates into lists of x and y values.

    :param pixels: list of pixels
    :type pixels: array

    :return: lists of x and y values
    :rtype: array, array
    """

    x_coordinates = []
    y_coordinates = []

    for point in pixels:
        x_coordinates.append(point[0])
        x_coordinates = list(dict.fromkeys(x_coordinates))
        x_coordinates.sort()
        y_coordinates.append(point[1])
        y_coordinates = list(dict.fromkeys(y_coordinates))
        y_coordinates.sort()

    x_coordinates = self.__delete_close(x_coordinates)
    y_coordinates = self.__delete_close(y_coordinates)

    return x_coordinates, y_coordinates
```

Código 11 - Método para determinar as linhas da tabela

O objetivo de segmentar a tabela é alcançado com esta última fase, onde será feita a divisão da imagem em pequenos segmentos. Pretende-se a criação de um dicionário *python* que contenha grupos com os segmentos correspondentes ao nome, assinatura de manhã e assinatura da tarde de cada utilizador. Para tal, foi construído o método *__split_table*, representado no Código 12.

```
def   split table(self, img, x list, y list):
    """Receives an image and a list of x and y numbers. This function then splits the image
into different segments,
     cutting based on the given points

    :param img: image with table
    :param x_list: x values
    :type x_list: array
    :param y_list: y values
    :type y_list: array

    :return: objects with each user's name, morning and afternoon images
    :rtype: dictionary
    """

    pictures = {}
    nr of columns = len(x list) - 1
    nr_of_rows = len(y_list) - 2
    total_tables = int((len(x_list) - 1) / 3)
    total_people = nr_of_rows * total_tables
    for x in range(total people):
        pictures[str(x)] = []
    count = 0
    table_count = 0

    # Split given image, making an object containing each person's name, morning and
afternoon segments
    for x in range(nr_of_columns):
        if count >= 3:
            table_count += 1
            count = 0
        min_range = nr_of_rows * table_count
        max_range = nr_of_rows + (nr_of_rows * table_count)
        y_count = 0
        for y in range(min range, max range):
            crop_image = img[y_list[y_count + 1]:y_list[y_count + 2], x_list[x]:x_list[x +
1]]
            y_count += 1
            pictures[str(y)].append(crop_image)
        count += 1

    return pictures
```

Código 12 - Método para divisão da tabela em pequenas imagens

Internamente, este lida com diversas variáveis importantes: *img*, a imagem a ser dividida; *x_list*, lista de linha verticais; *y_list*, lista de linhas horizontais; *pictures*, dicionário inicialmente vazio que será preenchido pela função descrita; *nr_of_columns*, correspondente ao número de colunas da tabela; nr_of_rows, correspondente ao número de linhas da tabela; *total_table*, representa o número de "tabelas" lado a lado na imagem; *total_people*, representa o total de participantes contidos na tabela.

O ciclo desenvolvido irá dividir a imagem com base nos parâmetros previamente descritos, preenchendo o dicionário *pictures* com pares chave-valor, em que a chave é um *id* único e o valor é um *array* com pequenas imagens que correspondem ao nome e assinaturas de um utilizador específico (previamente exemplificados na Figura 23). Estas imagens encontram-se propositadamente organizadas, sendo que a primeira posição corresponde à identificação do utilizador, a segunda à assinatura da manhã e a terceira à assinatura da tarde. O dicionário

gerado será então devolvido e utilizado nas próximas fases deste projeto, concluindo deste modo a etapa de segmentação da tabela.

5.5.4. Identificação dos Nomes

Atualmente, o processo de leitura de tabelas possui um objeto com vários grupos de imagens, agrupadas por participante. Este tipo de dados, contudo, não é suficiente para ser possível identificar a pessoa que corresponde a cada objeto. Por este motivo, surgiu a necessidade desta etapa, que terá como objetivo a identificação dos nomes dos participantes correspondentes a cada grupo de imagens.

É utilizada a técnica de *OCR* para este fim, recorrendo à ferramenta *Tesseract*. Em síntese, as imagens em cada objeto que contêm a identificação do participante serão pré-processadas e analisadas com *OCR*, de modo a extrair o texto nelas contido. Este, por sua vez, será comparado com a lista de nomes inicialmente fornecida (parâmetro de entrada do módulo de leitura de presenças, explicado no 5.5.1.1) de modo a identificar o participante correto. No final, foi acrescentada uma fase de otimização, que pretende prevenir casos em que o mesmo participante é identificado incorretamente.

Este processo é alcançado através do Código 13, onde é possível observar a criação do objeto que irá conter os dados extraídos das várias análises que se seguem, denominado de *presences*. O seu funcionamento será detalhado na próxima secção de implementação da identificação dos nomes.

```
presences = {}
nr = NameReader()
for k in user_images.keys():
    presences[k] = {'name': None, 'morning': None, 'afternoon': None}  # start the object

    result = nr.run(user_images[k][0], k, json_path)

    if isinstance(result, int):
        # If an int is return it's because a better match for a name was found
        # the returned int is the key of the name to be replaced
        name = presences[str(result)]["name"]
        presences[str(result)]["name"] = None
        result = name

    presences[k]['name'] = result
```

Código 13 - Processo de identificação dos nomes

5.5.4.1. Implementação da Identificação dos Nomes

Ao analisar o Código 13, é possível observar os principais componentes que integram o processo de reconhecimento de nomes. Para a conclusão deste projeto apenas fica a faltar a implementação da extração da informação contida nas imagens. O objetivo desta etapa em específico é a identificação dos participantes, a partir do dicionário que resultou da etapa de segmentação da imagem. Para tal, foi criado o objeto *presences*, onde será armazenada a coleção de nomes e assinaturas, pós-interpretada, agrupadas por participante.

Inicialmente, são percorridos todos os objetos contidos no dicionário que resultou da segmentação da imagem, denominado de *user_images*. A primeira imagem de cada um destes objetos irá corresponder à identificação do respetivo participante. Cada uma destas primeiras imagens será analisada pela classe *NameReader* (*NR*), onde é feita a interpretação do texto com *OCR*.

A funcionalidade da classe *NR* pode ser acedida através do seu método *run*, que requer 3 parâmetros obrigatórios e 1 opcional, sendo estes: *image*, obrigatório, que será a imagem com a identificação do participante; *key*, obrigatório, correspondente à chave que identifica em *user_images*, o objeto a que pertence o participante; *json_path*, obrigatório, correspondente ao *JSON* com a lista de nomes fornecido inicialmente; *similarity*, opcional por já possuir o valor pré-definido 70, que indica quão semelhantes, numa escala de 0 a 100, o resultado de uma leitura *OCR* tem de ser de um dos nomes do *JSON,* para ser considerada uma correspondência.

Primeiramente, são executadas algumas técnicas simples de pré-processamento sobre *image*, pois, como demonstrado mais à frente na secção 6.2.1, estas poderão ser determinantes da qualidade das leituras. Foram realizados alguns testes de modo a verificar quais a técnicas que permitiam obter os melhores resultados, tendo *Grayscalling* e *Thresholding* sobressaído comparativamente às restantes. Por este motivo, são geradas duas novas imagens a partir de *image*, resultantes da aplicação das técnicas referidas. Em seguida*,* é executado o motor *Tesseract* sobre estas novas imagens, interpretando os seus conteúdos através de *OCR*. Os resultados das leituras realizadas são depois armazenados nas variáveis *g_read* (para a imagem em *grayscale*) e *t_read* (para a imagem com *thresholding*).

Ambos *g_read* e *t_read* são inseridos, juntamente com o *JSON* inicial, como parâmetros numa nova função *__determine_best,* onde será feita a correspondência entre os nomes lidos pelo *Tesseract* e o nome real do utilizador. Neste método, é utilizado o *package fuzzywuzzy*[27], que possui a capacidade de calcular o quão semelhantes duas *strings* são, indicando um valor entre 0 a 100. Essencialmente, são percorridos todos os nomes contidos dentro do *JSON* e, para cada um destes nomes, é feita a comparação com ambos *g_read* e *t_read.* Ao longo das iterações, vai sendo armazenado o nome do *JSON* que conseguiu um maior valor de semelhança com *g_read* ou *t_read,* juntamente com o valor indicado pelo *fuzzywuzzy.* Como tal, o método *__determine_best* irá devolver um *array* que contém na posição 0 o nome do *JSON* mais semelhante aos nomes lidos com *OCR* e, na posição 1, o valor que representa esta semelhança, calculado pelo *fuzzywuzzy*.

Será feita uma verificação, onde é comparada a semelhança entre o valor detetado pelo *fuzzywuzzy* e o parâmetro *similarity*. Caso o primeiro referido seja maior ou igual ao segundo, então o nome do utilizador é extraído a partir do *JSON* e está pronto para ser devolvido pelo *NR*. Contudo, existe ainda uma otimização que é aplicada antes do nome ser devolvido. Os nomes dos participantes detetados vão sendo guardados na classe à medida que são detetados, juntamente com o respetivo valor de semelhança e o seu valor de *key*. Isto é necessário porque, como se encontra demonstrado em 6.4.1, existe o risco do mesmo nome ser detetado duas vezes. Como medida de prevenção, cada nome que é lido terá de ser comparado com os nomes previamente identificados. No caso de ambos os nomes serem o mesmo, será apenas guardado aquele que possuir um maior valor de semelhança. No caso de um nome ser substituído, o *NR* não irá devolver a *string* com o nome detetado, mas sim o valor da *key* do nome que foi substituído. Como é possível observar do Código 13, será verificado se de facto foi devolvido um número inteiro, correspondente à *key*. Caso isto se verifique, o objeto incorretamente identificado será anulado, passando a atribuir-se o seu nome ao objeto correto.

Esta etapa irá resultar no preenchimento dos devidos nomes no dicionário *presences*, representado na Figura 27. Será ainda adicionado um *array* a este objeto contendo os nomes dos participantes cujo nome não está presente no *JSON* inicial, mas ausente em *presences*.

[27] https://github.com/seatgeek/fuzzywuzzy

```
{
    0: { 'name': 'Lotye DENIGER', 'morning': None, 'afternoon': None },
    1: { 'name': 'Iven FRÉCHETTE', 'morning': None, 'afternoon': None },
    2: { 'name': 'Gaetan GALARNEAU', 'morning': None, 'afternoon': None },
    3: { 'name': 'Arienne GARCIA', 'morning': None, 'afternoon': None },
    4: { 'name': 'Campbell MASSÉ', 'morning': None, 'afternoon': None },
    5: { 'name': 'Charles MORIN', 'morning': None, 'afternoon': None },
    6: { 'name': 'Armina RICHARD', 'morning': None, 'afternoon': None },
    7: { 'name': 'Valiant RUEST', 'morning': None, 'afternoon': None },
    8: { 'name': 'Roland SABOURIN', 'morning': None, 'afternoon': None },
    9: { 'name': 'chappell  THÉRIAULT', 'morning': None, 'afternoon': None }
}
```

Figura 27 - Objeto presences após leitura dos nomes

5.5.5. Identificação das Presenças

Esta é a última etapa do processo de deteção das presenças dos participantes. O objetivo será verificar se um participante assinou, ou não, a folha de presenças. Neste ponto, o programa já começou a construção do dicionário *presences*, representado na Figura 27, que já contém os nomes dos utilizadores detetados. Resta fazer a análise das imagens correspondentes às assinaturas de cada participante. Por esse motivo, surge esta etapa final, onde serão analisadas as restantes imagens, resultando na versão completa do dicionário *presences* que será devolvido ao utilizador.

A técnica adotada irá verificar se as imagens dedicadas às assinaturas se encontram preenchidas, tendo por base a presença de tinta na imagem. Foi escolhido este método por se ter verificado que, após a aplicação de *grayscalling*, a presença de uma assinatura é facilmente detetável através do cálculo da percentagem de pontos pretos num fundo branco. No final, será retornado *presences* com a informação de cada utilizador identificado, terminando o processo de deteção de presenças.

5.5.5.1. Implementação da Identificação das Presenças

O objetivo desta etapa é o preenchimento dos restantes elementos do dicionário *presences*, relativos às assinaturas. Para tal, será novamente percorrido o objeto com as imagens de cada participante, resultante da etapa de segmentação da tabela. Cada uma das imagens relativas às presenças será processada pelo método *__check_presence*, que irá devolver *True* ou *False* conforme se confirme ou não, respetivamente, a presença de uma assinatura na imagem. O valor resultante da execução deste método será guardado no respetivo lugar dentro de *presences*.

A função *__check_precense* tem a possibilidade de receber vários parâmetros: *image*, de caracter obrigatório, que corresponde à imagem a ser analisada; *sign_thresh*, opcional, que corresponde ao mínimo de preto que uma imagem tem de ter para se considerar uma assinatura; *margin_ignore_h*, opcional, que corresponde à margem de *image* que deve de ser ignorada no topo e fundo; *margin_ignore_w*, opcional, que corresponde à margem de *image* que deve ser ignorada na esquerda e na direita; *thresh*, opcional, que representa o valor na escala de *grayscale* que um píxel precisa ter para ser considerado parte da assinatura. Os parâmetros opcionais encontram-se com valores já predeterminados, resultantes de vários testes. A necessidade dos parâmetros relativos às margens encontra-se explicada na secção 6.4.1.

Primeiramente, a função irá cortar *image* com base nos valores *margin_ignore_w* e *margin_ignore_h*. A necessidade desta transformação surgiu após terem sido realizados vários testes, em que se verificou a frequência com que as assinaturas dos vários participantes excediam os seus limites, invadindo os espaços dos seus vizinhos. Este corte permite que seja analisado apenas o centro de cada imagem onde, em geral, a assinatura do participante correspondente é mais carregada. Isto possibilita a subida do valor *thresh*, fornecendo à máquina uma maior hipótese de ignorar traços alheios e de detetar apenas a assinatura do respetivo participante. Este corte permite também corrigir algumas das imperfeições resultantes da etapa de segmentação, sendo que por vezes as próprias linha da tabela se encontravam presentes na imagem, tornando as leituras imprevisíveis. Um exemplo de como esta alteração impacta as imagens encontra-se representada na Figura 28.

Figura 28 - Remoção de margens às assinaturas

Após cortada, a imagem será convertida a *grayscale*, para que os seus píxeis contenham valores de 0 a 255. Será contado o número de píxeis com valor superior a *thresh,* porque nesse caso consideram-se como fazendo parte da assinatura. Do valor resultante será possível calcular qual a percentagem da imagem que corresponde à assinatura. Caso essa

percentagem seja superior a *sign_thresh*, então considera-se que foram detetados píxeis suficientes para ser considerada a presença de uma assinatura e a função devolve *True*. Caso não existam uma percentagem suficiente de pontos para superar *sign_thresh*, então a função devolve *False*. No fim de terem sido analisadas todas as imagens relativas às assinaturas, o objeto *presences* estará completo e poderá ser devolvido ao utilizador, concluindo o processo de deteção de assinaturas a partir de uma imagem da tabela de presenças de um evento.

5.5.6. Resultados

Dá-se por concluído o processo de deteção de presenças após a execução de todas as etapas referidas, começando com a obtenção dos parâmetros e terminando com a deteção de assinaturas. O resultado final será o dicionário *python presences*, representado no Código 14, que irá conter um objeto para cada participante identificado. Cada um destes objetos contém uma *string* com o nome do participante e dois *booleans* que indicam a presença do participante nos períodos da manhã e da tarde do evento. No final, *presences* é convertido para um objeto *JSON* e devolvido através de *HTTP* à aplicação que o chamou, para que esta se possa atualizar.

Por se tratar de um *POC*, não foi efetuada a integração desta solução na aplicação. Contudo, a solução foi construída para ser facilmente integrada em qualquer parte da *API*, tendo sido provada a sua eficiência com fotografias reais. As funcionalidades restantes, como o envio de notificações *PUSH* ou a atualização da *app* ao receber resultados, foram implementadas no contexto da evolução da aplicação *APM* que sucedeu o desenvolvimento desta solução.

```
{
    0: { 'name': 'Lotye DENIGER', 'morning': True, 'afternoon': False },
    1: { 'name': 'Iven FRÉCHETTE', 'morning': True, 'afternoon': True },
    2: { 'name': 'Gaetan GALARNEAU', 'morning': True, 'afternoon': True },
    3: { 'name': 'Arienne GARCIA', 'morning': False, 'afternoon': True },
    4: { 'name': 'Campbell MASSÉ', 'morning': True, 'afternoon': True },
    5: { 'name': 'Charles MORIN', 'morning': False, 'afternoon': True },
    6: { 'name': 'Armina RICHARD', 'morning': False, 'afternoon': False },
    7: { 'name': 'Valiant RUEST', 'morning': True, 'afternoon': False },
    8: { 'name': 'Roland SABOURIN', 'morning': False, 'afternoon': True },
    9: { 'name': 'chappell  THÉRIAULT', 'morning': True, 'afternoon': True }
}
```

Código 14 - Objeto resultante da deteção de presenças

6. Testes

Para este projeto foi necessário assegurar o cumprimento de diversas práticas ao longo do ciclo de desenvolvimento, de modo a manter constantemente a satisfação do cliente. Destaca-se a influência do *Scrum* no desenvolvimento de *software*, resultando num processo contínuo e transparente, com uma forte presença do cliente. Existe uma constante noção do estado do projeto e do valor do trabalho realizado, devido à estrutura da metodologia referida.

O trabalho que é realizado num projeto existe associado a requisitos e expectativas impostos pelo cliente. Estes são, por este motivo, fatores essenciais na determinação da qualidade do trabalho concluído e da satisfação do cliente. Aquilo que é esperado de uma funcionalidade tem de estar presente no produto final e, tal é algo que muitas vezes não é possível assegurar durante a sua implementação. Esta é uma das razões que justificam a necessidade pela realização de testes: para que seja possível garantir a qualidade do trabalho realizado, assegurando o cumprimento dos requisitos esperados.

Um teste de *software* é um processo realizado por uma máquina, pessoa ou conjunto de pessoas que pretende verificar se o comportamento de uma funcionalidade, sistema ou *software* cumpre com os requisitos estipulados.

Para este projeto, os testes foram obrigatórios ao longo de todo o processo de desenvolvimento, encontrando-se integrados no ciclo de vida de uma *story*. Contudo, a realização de testes não se encontrava restrita à situação referida anteriormente. Existe um canal de comunicação dedicado ao suporte do cliente, onde eram frequentemente publicados problemas encontrados na *app*, o que por sua vez exigia a revisão de funcionalidades com recurso a testes.

Existem diversas ferramentas e métodos que permitem realizar testes, sendo que a técnica utilizada deve ser escolhida conforme o tipo de teste pretendido. Numa fase inicial do desenvolvimento de uma funcionalidade, poderá ser apenas necessário testar se esta demonstra um comportamento que faz o pretendido. Após a integração dessa funcionalidade na aplicação, irá ser também necessário testar todo o sistema e como o componente nele se integra. Uma outra variável entre testes é quem os realiza. Numa fase inicial do

desenvolvimento poderá ser suficiente ter um outro programador a testar a funcionalidade. Contudo, antes de estar disponível ao público, irá ser necessário realizar testes também com possíveis utilizadores reais do sistema.

Destacam-se dois tipos mais importantes de testes, os *white box* e os *black box*. Os primeiros incidem sobre o código, procurando otimizá-lo. Os segundos não têm sequer conhecimento do código, pretendendo simular a perspetiva do utilizador final.

Neste capítulo será explicado o processo de testes adotado para este projeto, detalhando na secção 6.1 a forma como se integra no ciclo de vida de uma *story*, as secções 6.2 e 6.3 os testes realizados relativamente ao modulo de leitura de presenças e a secção 6.4 referente aos testes com exemplos de fotografias reais e à captura de imagens.

6.1. Relação entre testes e *stories*

Esta secção refere-se ao papel que os testes têm no ciclo de vida de uma *story*. Como foi mencionado anteriormente, estes encontram-se integrados no ciclo de desenvolvimento. No quadro *Kanban* que a empresa utiliza, presente no *Gitlab*, o trabalho encontra-se representado sob a forma de *stories*. Sempre que surge a necessidade de implementar uma nova funcionalidade, rever algo ou testar quaisquer outros novos conceitos, é criada uma nova *story*, que é armazenada no *product backlog*, onde se encontrará pronta para ser estimada e atribuída a uma *Sprint* num futuro *Sprint Planning*.

Um programador irá associar-se ao trabalho que pretende realizar, colocando a *story* na coluna *in progress* até o trabalho se encontrar concluído. Após verificada esta conclusão, o programador deverá escrever alguns casos de teste para complementar os requisitos já presentes na *story*. Estes casos de testes servem como um guia para os próximos *testers*, que irão testar a funcionalidade em causa, de modo a verificar se o programa cumpre com o esperado. Um caso de teste está estruturado como *user story*. A *story*, por se encontrar concluída, deverá transitar para a coluna *code review*, onde esta será testada por um outro programador que não tenha intervindo com o desenvolvimento da funcionalidade. Pretende-se deste modo fazer testes do tipo *white box*, onde se tem especial atenção ao código que foi escrito enquanto se revê o funcionamento da funcionalidade. Caso sejam encontradas inconsistências no código, erros imediatamente percetíveis, ou otimizações impactantes,

então, quem está a testar e a rever o código deverá enviar a *story* de novo para a coluna *in progress*, acompanhada por uma justificação do porquê de a *story* não ter sido aprovada nesta etapa. Caso uma *story* que encontre na coluna *code review* seja aprovada, então esta deverá passar para a próxima etapa, denominada de *awaiting review*.

Na etapa de *awaiting review*, é realizado um teste do tipo *Black Box* por um *tester* da *md3*. O teste conduzido é diferente dos anteriores, pois tem tanto o objetivo de testar o trabalho efetuado na respetiva *story*, como o de testar a sua integração com todo o sistema. Por ser do tipo *black box*, pretende-se a simulação da experiência de um utilizador real com a aplicação.

O *tester* da *md3* tem a responsabilidade de testar tanto a aplicação *mobile* como as soluções *web* da *APM*. No caso da aplicação móvel, são distribuídas versões da aplicação próprias para testes, variando o seu conteúdo conforme o *branch* em que foi a *story* trabalhada. Estas versões da aplicação são distribuídas através da plataforma *firebase*, recorrendo à *App Distribuition* para distribuir os *APKs* e *IPAs*. Caso o *tester* rejeite a *story* na fase *awaiting review*, esta regressa à etapa de *in progress*. Contudo, caso esta seja aprovada, então a *story* passa para uma nova etapa denominada *product owner*.

A etapa *product owner* tem como objetivo mostrar o trabalho resultante das *stories* ao *product owner*, o representante do cliente e o elemento mais próximo do utilizador final. Os testes são realizados manualmente pelo *product owner*, que utiliza também uma versão da aplicação disponibilizada através de *firebase*. Caso a *story* seja rejeitada, então esta irá regressar à etapa *in progress*. Contudo, caso esta seja aprovada, então estará pronta para ir para produção.

Após a entrada de uma *story* para produção, esta é arquivada e dá-se por concluído o seu ciclo de vida.

6.2. Testes no contexto da leitura de tabelas

Este projeto exigiu um grande esforço investigativo durante o seu desenvolvimento, tendo sido necessário procurar e estudar várias técnicas e soluções de modo a construir a melhor solução para o problema de leitura de tabelas. Esta pesquisa teve de ser complementada pela realização de testes contínuos, assegurando a qualidade do funcionamento individual e

integrado dos vários componentes do projeto. Deste modo, foi possível identificar os pontos fortes e fracos do trabalho, permitindo o seu melhoramento e consolidação. Nesta secção, são expostos os principais testes realizados ao longo do desenvolvimento deste projeto que contribuíram para o sucesso da solução final.

6.2.1. Impacto da qualidade da imagem com OCR

Foram realizados alguns testes unitários, com recurso à *framework unittest*, que pretendiam estimar o impacto da qualidade de uma imagem em leituras *OCR*. O objetivo destes testes era testar as limitações do motor *Tesseract* e comprovar a eficiência do *OpenCV* na sua resolução.

O *Tesseract* provou-se capaz de obter bons resultados quando tentava detetar texto a partir de imagens limpas, sem qualquer distúrbio na sua qualidade. Contudo, ao serem analisadas imagens com má qualidade, os resultados não eram satisfatórios, podendo até impossibilitar qualquer deteção de texto. Uma solução para este problema é a aplicação de técnicas de pré-processamento.

No exemplo retratado no Código 15, são testadas as imagens na Figura 29. O *Tesseract* consegue extrair perfeitamente o texto contido na imagem limpa, contudo, após ser aplicado algum ruído, verifica-se que o *Tesseract* já não é capaz de sequer detetar texto. De modo a testar a eficiência do *OpenCV* na resolução deste problema, é aplicada a técnica de pré-processamento de *Gaussian Blur* que permitiu eliminar o ruido presente na imagem com sucesso. Verifica-se que após a aplicação da técnica referida o *Tesseract* conseguiu novamente uma leitura perfeita da imagem, provando que de facto o pré-processamento tem impacto na qualidade das leituras *OCR* realizadas com *Tesseract*.

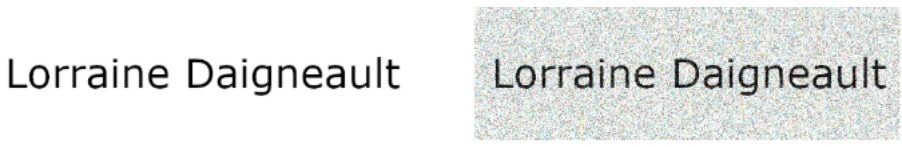


Figura 29 - Imagens para teste de impacto de ruído

```
def test_tesseract_pre_processing_impact(self):
    """Tests some flaws tesseract has and how they can be beaten with pre-processing.
    The quality of an image will severely impact how tesseract reads it. A developer can't expect
users that
    will use this software to provide perfect photos. For this reason, it's attempted to improve
the quality of the
    provided image through pre-processing techniques.

    In this example tesseract perfectly reads a clean image, however after applying some noise to
the same image
    it's verified that tesseract can no longer read it. By applying simple gaussian blurring to the
photo, it was
    possible to remove the noise and prove that pre-processing does have the ability to allow for
the interpretation
    of otherwise unreadable photos.
    """
    example_name = "Lorraine Daigneault"

    # reading a clean image (reads perfectly)
    clean_img = cv2.imread("lorraine_simple.png")
    clean_read = pytesseract.image_to_string(clean_img)
    self.assertEqual(example_name, clean_read, "Names should match.")

    # reading an image with noise (fails)
    noise_img = cv2.imread("lorraine_noise.png")
    noise_read = pytesseract.image_to_string(noise_img)
    self.assertNotEqual(example_name, noise_read, "Names shouldn't match, because image noise broke
tesseract.")

    # applying grayscale and blur
    gray = cv2.cvtColor(noise_img, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (9, 9), 0)
    processed_read = pytesseract.image_to_string(blur)

    self.assertEqual(example_name, processed_read, "Names should match after removing noise with
pre-processing.")
```

Código 15 - Consequências e correção de ruído em leituras

Foram também testadas as limitações ao nível das características do próprio texto, através do Código 16, que age sob a imagem na Figura 30. Neste caso, é fornecida uma imagem onde o texto se encontra em *bold* e ligeiramente aproximado. O teste realizado executa uma leitura *OCR* sobre essa mesma imagem e verifica-se o insucesso da extração do texto. Em seguida, é aplicada a operação morfológica de erosão sobre essa mesma imagem, resultando na segmentação dos caracteres, tornando-os distinguíveis. Isto permitiu que o *Tesseract* fosse capaz de ler perfeitamente a imagem, comprovando mais uma vez a importância do pré-processamento nas leituras realizadas.

close_example.png

Figura 30 - Imagem para teste de impacto de texto *bold* próximo

```
def test_morphological_operations(self):
    """Test how morphological operations can help tesseract better define text characters.
    In this example, it's provided an image where the characters are in bold and too close
to each other.
    Tesseract is NOT able to interpret this image.
    After applying dilation the letters get slimmer (because the whites in the image
dilate!).
    This allows tesseract to interpret the characters and return the expected text.
    """
    example_name = "Johnny Joestar"

    # reading name from original image
    original_img = cv2.imread("close_example.png")
    gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)

    original_read = pytesseract.image_to_string(gray_img)

    self.assertNotEqual(example_name, original_read, "Names shouldn't match since tesseract
can't read properly.")

    # reading after dilation
    kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]], np.uint8)
    dilated_img = cv2.dilate(gray_img, kernel)

    dilated_read = pytesseract.image_to_string(dilated_img)

    self.assertEqual(example_name, dilated_read, "Names should match after applying
erosion.")
```

Código 16 - Consequências e correção de texto *bold* próximo

6.2.2. Deteção de contorno

Inicialmente utilizou-se uma abordagem baseada na função *Hough Lines Transform* do *OpenCV* para detetar tabelas, contudo, verificou-se que existiam falhas neste método nomeadamente quando exposto a imagens reais onde as linhas não são perfeitamente retas e existem vários artefactos nas imagens. Como tal, foi necessário procurar por uma alternativa, tendo-se optado pelo uso do método *findContours* da biblioteca *OpenCV* (ambos conceitos explicados anteriormente na secção 5.5.3.1.1). Decorreram diversos testes de modo a validar o funcionamento desta abordagem, sendo em seguida demonstrado um destes mesmo testes (Código 17), onde se pretende verificar se, ao analisar uma imagem com um contorno (Figura 31) o algoritmo é capaz de detetar as suas extremidades. O resultado foi positivo, tendo sido possível verificar que as leituras são efetuadas com precisão.

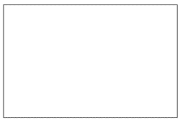


Figura 31 – Imagem de contorno (simulação de tabela)

```
def test_find_contour(self):
    """Test if cv2's find contour function is able to detect a table, and the correct edges of the
table
    Expected coordinates:
        Top Left: (113, 95)
        Top Right: (404, 95)
        Bottom Right: (404, 285)
        Bottom Left: (113, 285)
    """
    img = cv2.imread("simple_contour.png", 0)
    # invert colors
    inv = cv2.bitwise_not(img, img)
    # Find contours
    contours, _ = cv2.findContours(inv, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # Sort by area, descending
    contours = sorted(contours, key=cv2.contourArea, reverse=True)
    # Largest contour
    polygon = contours[0]

    bottom_right, _ = max(enumerate([pt[0][0] + pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))
    top_left, _ = min(enumerate([pt[0][0] + pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))
    bottom_left, _ = min(enumerate([pt[0][0] - pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))
    top_right, _ = max(enumerate([pt[0][0] - pt[0][1] for pt in polygon]),
key=operator.itemgetter(1))

    self.assertEqual(polygon[top_left][0][0], 113, "The top left coordinate should be (113, 95)")
    self.assertEqual(polygon[top_left][0][1], 95, "The top left coordinate should be (113, 95)")

    self.assertEqual(polygon[top_right][0][0], 404, "The top right coordinate should be (404, 95)")
    self.assertEqual(polygon[top_right][0][1], 95, "The top right coordinate should be (404, 95)")

    self.assertEqual(polygon[bottom_left][0][0], 113, "The bottom left coordinate should be (113,
285)")
    self.assertEqual(polygon[bottom_left][0][1], 285, "The bottom left coordinate should be (113,
285)")

    self.assertEqual(polygon[bottom_right][0][0], 404, "The bottom right coordinate should be (404,
285)")
    self.assertEqual(polygon[bottom_right][0][1], 285, "The bottom right coordinate should be (404,
285)")
```

Código 17 - Teste de deteção de contorno

6.3. Testes ao MP

Foram feitos vários testes ao *MP*, desenvolvido para deteção do esqueleto de tabelas de presenças da *APM*. Para além dos extensos testes que foram efetuados de forma manual, foram também realizados testes unitários com recurso à biblioteca *unittest*. Utilizando excertos de código para conduzir estes testes, foi possível demonstrar a qualidade da solução criada. Em seguida são explicados alguns dos testes unitários que foram escritos.

Neste primeiro teste, indicado no Código 18, é testada a capacidade do *MP* para encontrar linhas pretas numa imagem, representada na Figura 32. Esta deteção é feita com base na propriedade *range (*denominada de *search* na secção *5.5.3.1.1)*, que representa o número de píxeis a serem investigados em frente à posição do agente. A reta que representa o topo da tabela encontra-se no sexto píxel a contar do topo da imagem. Neste teste, o *range* é

inicialmente 5, pelo que se verifica que não são detetados quaisquer píxeis pretos ao correr o teste, contudo, ao aumentar este *range* para 6 já se verifica a deteção de todos os píxeis que compõem a linha do topo da tabela. Com este teste é possível provar a importância do parâmetro *range* para a deteção de uma imagem (tendo sido realizados testes para determinar o melhor valor para este parâmetro com exemplos reais).

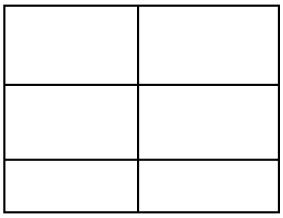


Figura 32 - Imagem para teste do parâmetro *range*

```
def test_min_search_range(self):
    """Test if the expected minimum range matches the image coordinates.
    Black pixels are on x = 5. Chosen range is 5 and then 6.
    NOTE: I'm just testing for the top margin, however this test is valid for all other directions.

    If black pixel is on (0,5), to find it the range needs to be at least 6. This happens because
    when the algorithm starts counting from 1 instead of 0
    """
    img = cv2.imread("table_400_300.png", 0)
    pixels = []
    width = img.shape[1]
    r = 5

    # Range too short
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                # found a black pixel, let's jump there!
                if img[s, w] == 0:
                    pixels.append((w, s))
            else:
                break

    self.assertEqual(len(pixels), 0, "Should be empty when range is 5")

    pixels = []
    width = img.shape[1]
    r = 6

    # Range is good
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                # found a black pixel, let's jump there!
                if img[s, w] == 0:
                    pixels.append((w, s))
            else:
                break

    self.assertGreater(len(pixels), 0, "Should have pixels when range is greater than 5")
```

Código 18 - Teste ao parâmetro *range* do *MP*

No segundo teste, representado no Código 19, verifica-se a forma como o *MP* consegue definir os pontos de interesse da tabela, ou seja, os pontos onde linhas se cruzam. O *MP* percorre a tabela contida na Figura 33 pela horizontal e vertical, procurando por pontos pretos dentro do *range*. Ao encontrar um destes pontos, aplica novamente a *range* a partir do ponto que foi detetado, procurando descobrir se foram encontrados apenas píxeis pretos. Caso isto se verifique, o *MP* irá considerar que encontrou uma linha. O teste teve sucesso, tendo sido detetadas todas as 3 linhas verticais e as 2 linhas horizontais presentes na imagem.

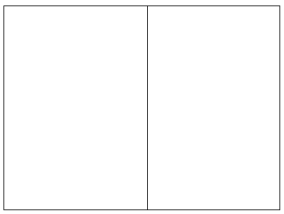


Figura 33 - Imagem para teste de identificação de linhas pelo *MP*

```
def test_finds_lines(self):
    """Check if the vertical and horizontal lines are correctly identified
    Once a black pixel is detected, it's verified if it is followed by a line of black pixels, so
that it can be
    then considered as a table line.

    For vertical lines: Pixels on x = 9 and range = 10.
    For horizontal lines: Pixels on y = 9 and range = 10.
    """
    img = cv2.imread("1pxlength_400_300.png", 0)
    pixels = []
    width = img.shape[1]
    r = 10

    # find vertical lines
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                if img[s, w] == 0:
                    # for now i'll be searching for the same range in front of the pixel
                    found_white_pixel = False
                    range_from_pixel = r
                    for s2 in range(range_from_pixel):
                        y = s + s2 + 1
                        if img[y, w] != 0:
                            found_white_pixel = True
                            break
                    if not found_white_pixel:
                        pixels.append((w, s))
                    found_black_pixel = True
            else:
                break

    self.assertEqual(len(pixels), 3, "Should have found 3 pixels that lead to vertical lines")

    # find horizontal lines
    pixels = []
    height = img.shape[0]

    for h in range(0, height):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                if img[h, s] == 0:
                    found_white_pixel = False
                    range_from_pixel = r
                    for s2 in range(range_from_pixel):
                        x = s + s2 + 1
                        if img[h, x] != 0:
                            found_white_pixel = True
                            break
                    if not found_white_pixel:
                        pixels.append((s, h))
                    found_black_pixel = True
            else:
                break

    self.assertEqual(len(pixels), 2, "Should have found 2 pixels that lead to horizontal lines")
```

Código 19 - Teste para verificar se são detetadas as linhas horizontais e verticais da tabela pelo MP

No seguinte teste, representado no Código 20, pretende-se verificar a forma como um artefacto impacta o funcionamento do *MP* relativamente à deteção de falsos positivos e, como este problema pode ser contornado através de ajustes ao parâmetro *range*. É fornecido ao *MP* uma nova imagem contendo um artefacto no canto superior esquerdo (imagem com o artefacto ampliado na Figura 34). Este artefacto é composto por uma sequência de píxeis que se estende verticalmente, numa tentativa de induzir o *MP* em erro, fazendo-o detetar

uma falsa linha vertical. Sendo que o número de píxeis necessário para considerar uma linha corresponde ao valor de *range*, é possível observar com este teste a forma como este parâmetro possibilita a evasão a problemas relativos à deteção de linhas falsas. Numa primeira condição o *MP* é executadosobre a imagem com uma *range* de 10, que por ser o número exato de píxeis que compõe verticalmente o artefacto, resulta na falsa deteção de uma linha vertical. Contudo, é realizado em seguida o mesmo teste, mas com uma *range* de 15, resultando na identificação correta dos pontos de interesse da tabela. Deste modo, foi comprovado o modo como os ajustes a este parâmetro podem ser determinantes da qualidade da leitura. Este teste foi a base para o processo de determinação para os melhores valores de *range*, tendo sido executados outros testes em exemplos reais.

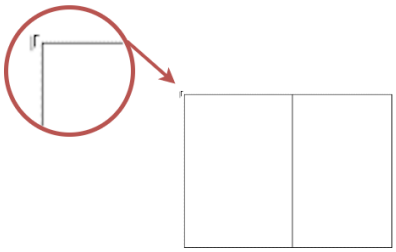


Figura 34 - Zoom de imagem para teste de artefactos no MP

```
def test false vertical lines(self):
    """Check if the an artifact can affect the detection of false vertical lines.
    This is very rare, but it can happen.

    I manage to avoid this through good pre-processing or by adjusting the search range parameter.
    When the search range is 10, the artifacts in the image are falsely interpreted as vertical
lines.
    This is fixed by adjusting the search range to a greater number.
    """
    img = cv2.imread("artifact_1pxlength_400_300.png", 0)
    pixels = []
    width = img.shape[1]

    # read false vertical lines
    r = 10
    for w in range(0, width):
        found black pixel = False
        for s in range(r):
            if not found black pixel:
                if img[s, w] == 0:
                    found white pixel = False
                    range_from_pixel = r
                    for s2 in range(range from pixel):
                        y = s + s2 + 1
                        if img[y, w] != 0:
                            found_white_pixel = True
                            break
                    if not found_white_pixel:
                        pixels.append((w, s))
                    found_black_pixel = True
            else:
                break

    self.assertGreater(len(pixels), 3, "Couldn't detect any artifacts.")

    # ignore false vertical lines
    r = 15
    pixels = []
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                if img[s, w] == 0:
                    found_white_pixel = False
                    range from pixel = r
                    for s2 in range(range_from_pixel):
                        y = s + s2 + 1
                        if img[y, w] != 0:
                            found white pixel = True
                            break
                    if not found white pixel:
                        pixels.append((w, s))
                    found black pixel = True
            else:
                break

    self.assertEqual(len(pixels), 3, "Found more than the expected lines.")
```

Código 20 - Teste para verificar o impacto do parâmetro range/search

Foi realizado um terceiro teste (Código 21) que, à semelhança do anterior, também procura solução para o impacto de artefactos nas leituras do *MP*. A diferença é que no exemplo anterior os artefactos resultavam na leitura de um falso positivo, enquanto que neste caso são reportados falsos negativos. É demonstrado como este problema pode ser resolvido com a aplicação de técnicas de pré processamento.

O *MP* irá analisar uma imagem, representada na Figura 35, que contém uma tabela juntamente com alguns artefactos. Estes artefactos são píxeis que se encontram a pouca distância do topo da figura, exatamente em cima das linhas verticais da tabela. São testadas duas condições dentro deste teste. Na primeira, o *MP* simplesmente analisa a imagem no seu estado original, o que resulta numa falha da deteção das linhas verticais da tabela. Este erro acontece devido ao modo como o *MP* funciona internamente, pois, por existir um espaço em branco entre o artefacto e a própria tabela, não será possível a identificação de uma sequência de píxeis pretos, fator determinante para se considerar a presença de uma linha. Na segunda condição, verifica-se que é possível remover os artefactos recorrendo às técnicas de *Gaussian Blur*, *Thresholding* e dilatação. Deste modo, é possível visualizar a importância do pré processamento para assegurar a qualidade das leituras do *MP*.

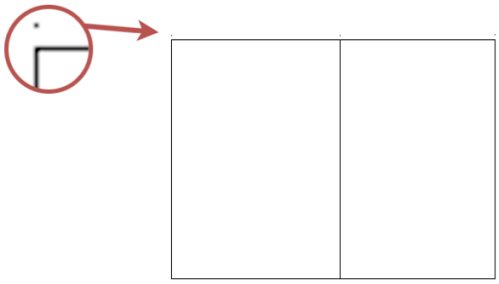


Figura 35 - Zoom em artefacto para teste com *MP*

```
def test_artifacts_interference_line_detection(self):
    """Check if the an artifact can affect the detection of vertical lines.
    Again, this is very rare, but it can happen.
    I manage to avoid this through good pre-processing.

    The provided image has pixels a bit in front of where the lines are located. because of this, no
lines are
    detected, since there is a big gap between the first black pixel and the actual line.

    Using pre-processing techniques on the same image I was able to remove the artifacts and
correctly read the
    lines, detecting the expected 3.
    """
    img = cv2.imread("artifact_alt_1pxlength_400_300.png", 0)
    pixels = []
    width = img.shape[1]

    # pixels blocking the read of vertical lines
    r = 10
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                if img[s, w] == 0:
                    found_white_pixel = False
                    range_from_pixel = r
                    for s2 in range(range_from_pixel):
                        y = s + s2 + 1
                        if img[y, w] != 0:
                            found_white_pixel = True
                            break
                    if not found_white_pixel:
                        pixels.append((w, s))
                    found_black_pixel = True
            else:
                break

    self.assertEqual(len(pixels), 0, "Falsely ignored artifacts.")

    # After applying pre-processing to remove artifacts (blurring, thresholding, dilation)
    blur = cv2.GaussianBlur(img.copy(), (11, 11), 0)
    thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11,
2)
    kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]], np.uint8)
    proc = cv2.dilate(thresh, kernel)

    r = 10
    pixels = []
    for w in range(0, width):
        found_black_pixel = False
        for s in range(r):
            if not found_black_pixel:
                if proc[s, w] == 0:
                    found_white_pixel = False
                    range_from_pixel = r
                    for s2 in range(range_from_pixel):
                        y = s + s2 + 1
                        if proc[y, w] != 0:
                            found_white_pixel = True
                            break
                    if not found_white_pixel:
                        pixels.append((w, s))
                    found_black_pixel = True
            else:
                break

    self.assertEqual(len(pixels), 3, "Found more the 3 lines.")
```

Código 21 - Teste artefactos falsos negativos *MP*

No último teste ao *MP*, representado no Código 22, pretendeu-se demonstrar o funcionamento de um outro parâmetro bastante importante, denominado de *dist* (ou *fuse* no código original). Este valor representa a distância necessária entre dois píxeis para que estes

sejam fundidos. Um exemplo da utilidade do *dist* surge quando a grossura de uma linha tem mais do que um píxel, pois o *MP* irá detetar vários pontos que pretendem representar a mesma linha. Para este teste foram escolhidos três pontos que se encontram próximos, sendo atribuído um valor de *dist* suficiente para considerar a fusão entre estes elementos. Ao executar o teste verifica-se que os três pontos foram transformados num só, que se encontra no centro dos 3 originais. Deste modo, é possível verificar que o uso do *dist* é realmente viável para fundir pontos próximos.

```
def test_minimum_fuse_distance(self):
    """Tests the minimum distance between 2 pixels for them to be fused.

    The distance between all points is 5, however because the algorithm fuses points as it searches
the
    array, the fusion will occur first between (0,0) and (0,5), creating the point (0,3).
    The distance between (0,3) and (5,0) is > 5 and therefore the minimum distance has to be
superior to this.
    """
    dist = 6
    # original pixels
    p = [(0, 0), (0, 5), (5, 0)]
    # pixels after fusion
    pixels_fused = []
    first = True

    for c1 in p:
        if first:
            pixels_fused.append(c1)
            first = False
        else:
            fused = False
            for c2 in pixels_fused:
                if calculate_distance(c1[0], c1[1], c2[0], c2[1]) <= dist:
                    x = math.ceil(((c1[0] + c2[0]) / 2))
                    y = math.ceil(((c1[1] + c2[1]) / 2))
                    pixels_fused.remove(c2)
                    pixels_fused.append((x, y))
                    fused = True
            if not fused:
                pixels_fused.append((c1[0], c1[1]))

    self.assertEqual(len(pixels_fused), 1, "The 3 pixels should have fused into 1.")
```

Código 22 - Teste à fusão de pontos próximos do *MP*

6.4. Em contexto real

Enquanto estava a ser desenvolvida a primeira versão funcional deste projeto, foram utilizadas imagens de tabelas produzidas manualmente com auxílio do *software Gimp*[28]. O uso deste tipo de imagens surgiu da necessidade de conseguir testar o *POC* nas melhores condições possíveis, executando a lógica dos algoritmos sobre exemplos onde não existem os típicos problemas que interferem com a interpretação da imagem. Ao ser concluída a primeira versão (3.5.0) do projeto, obteve-se uma solução capaz de interpretar com sucesso

[28] https://www.gimp.org

os conteúdos da tabela que simulava uma tabela de uma folha de presenças. Foi então que se deu início aos testes com imagens reais, de modo a validar a qualidade da solução no contexto a que esta se destina. Ao longo desta secção serão explicados os testes que foram realizados com imagens reais e a sua importância para o processo de desenvolvimento deste projeto.

6.4.1. Com fotografias de eventos

Existia a necessidade de adaptar a solução inicial, que funcionava bem com imagens simuladas, para uma solução capaz de funcionar com exemplos reais. Para tal, foi essencial a escolha dos melhores valores para os parâmetros que integram o processo, pois apenas deste modo seria possível combater os diversos problemas expostos anteriormente.

Os parâmetros testados foram *range*/*search* do *MP* (distância máxima para encontrar um ponto preto/valor necessário para considerar que uma sequência de píxeis é uma linha da tabela), *dist*/*fuse* do *MP* (distância mínima entre dois pontos para que estes sejam fundidos), *threshold* para assinaturas (valor de píxeis pretos necessários para confirmar a presença de uma assinatura), *delete_distance* (distância mínima para apagar pontos próximos devolvidos pelo *MP*, de modo a obter as linhas da tabela), semelhança entre nomes (percentagem que indica quão similar um nome detetado por *OCR* tem de ser com um dos nomes contidos no *JSON* para existir correspondência). Como tal, a solução original foi executada sobre um conjunto de 6 imagens, fotografias de folhas de presenças já assinadas, fornecidas pelos organizadores da *APM* após a conclusão dos eventos associados à folha em questão. Os fatores que foram utilizados para medir o sucesso destes testes foram o nome dos participantes e as presenças que foram identificadas corretamente. Esta secção surge com o intuito de documentar os vários testes realizados e a forma como estes resultaram em otimizações essenciais para o funcionamento do programa num contexto real.

No primeiro teste realizado a solução foi executada sobre as 6 imagens de eventos, utilizando os mesmos parâmetros que se provaram funcionais nos exemplos feitos através do *Gimp*. Contudo, os resultados não foram bons. Apenas foram interpretadas 2 imagens, tendo sido impossível para o programa a interpretação das restantes 4 fotografias. Uma das leituras feitas conseguiu interpretar com sucesso todos os parâmetros, tendo a outra falhado nas leituras dos nomes, mas acertando todas as presenças. Após obtidos estes resultados, foi realizada uma procura pelos problemas que estavam na origem das falhas ocorridas.

Identificou-se que estes erros provinham de duas fontes distintas, sendo uma delas a distorção da tabela e, a restante, as assinaturas que excediam as margens da tabela. Determinou-se que uma possível solução para o problema da distorção seria um ajuste ao parâmetro das coordenadas próximas, tendo este sido um dos parâmetros trabalhados em seguida. Uma representação dos resultados obtidos durante este primeiro teste encontra-se na Tabela 2.

Tabela 2 - Primeira fase de testes com imagens reais

	Margin Patrol Range	Point Fuse Range	Signature threshold	Close coordinate delete distance	Name reader similarity
	30	**10**	**0.1**	**10**	**70**
	Participants Detected		Correct Presences		
Event 1 - signed.png	19/21 (90%)		38/38 (100%)		
Event 2 - signed.png	**Error** - Table isn't detected because signatures trespass the border				
Event 3 - signed.png	**Error** - Close coordinate distance is too short to work here, because table is skewed				
Event 4 - signed.png	**Error** - Table isn't detected because signatures trespass the border				
Event 5 - signed.png	21/21 (100%)		42/42 (100%)		
Event 6 - signed.png	**Error** - Table isn't detected because signatures trespass the border				

Com o segundo teste pretendia-se descobrir a solução para um dos problemas anteriormente detetados, aquele causado pela distorção da tabela. Foram feitas alterações ao programa para alcançar este objetivo, tendo sido reajustado o parâmetro das *close coordinates* e a forma como eram determinadas as margens durante a identificação de presenças. Ou seja, quando uma secção da tabela referente a uma assinatura é extraída, é agora adicionada uma margem de 15% em todo o seu redor, para que seja apenas considerado o centro da imagem quando é feita a sua leitura. Desta forma é possível reduzir erros comuns como, por exemplo, assinaturas que saem das suas margens. A versão da aplicação resultante destas alterações reproduziu exatamente os mesmos testes executados com a versão anterior, tendo-se observado melhorias nos resultados obtidos: o evento 5 continuou a resultar numa leitura perfeita; o evento 1 não foi capaz de detetar todas as presenças, tendo acertado apenas 97%; o evento 3 conseguiu ser interpretado, tendo sido possível detetar corretamente 90% dos participantes e 97% das presenças; os restantes continuam a não serem lidos. Determinou-se que para o próximo teste seria procurada uma solução para o problema que estava na raiz das falhas restantes, as assinaturas que excediam as margens da tabela. A representação dos resultados obtidos com esta segunda fase de melhorias do *MP* com imagens reais encontra-se na Tabela 3.

Tabela 3 - Segunda fase de testes com imagens reais

	Signature Margin now considers **15%** of the height or width, no longer a fixed value				
	Margin Patrol Range	Point Fuse Range	Signature threshold	Close coordinate delete distance	Name reader similarity
	30	**10**	**0.01**	**20**	**70**
	Participants Detected		Correct Presences		
Event 1 - signed.png	19/21 (90%)		37/38 (97%)		
Event 2 - signed.png	**Error** - Table isn't detected because signatures trespass the border				
Event 3 - signed.png	16/17 (90%)		31/32 (97%)		
Event 4 - signed.png	**Error** - Table isn't detected because signatures trespass the border				
Event 5 - signed.png	21/21 (100%)		42/42 (100%)		
Event 6 - signed.png	**Error** - Table isn't detected because signatures trespass the border				

Com o terceiro teste realizado, o objetivo foi a resolução do problema das assinaturas. Verificou-se ao longo de vários exemplos que as folhas de presenças utilizadas num contexto real tinham, em grande maioria, assinaturas que excediam os limites dos espaços próprios para serem assinados. Verificou-se que este facto fazia com que a solução de leitura de tabelas, muitas vezes, interpretasse mal os resultados, principalmente ao nível do *MP*, que identificava partes destas assinaturas como sendo linhas da tabela. Como tal, foi necessário procurar por uma solução. Uma das observações que foi feita relativamente a estes falsos positivos é que, em todos os casos testados, estes se encontravam no fundo da tabela e não no seu topo. Uma justificação para esta situação poderá ser que a linha inicial da tabela serve sempre como cabeçalho, ao contrário da última, que contém espaços destinados a assinaturas. Como tal, tendo por base esta teoria, foi desenvolvida uma melhoria que pretende eliminar pontos do fundo da tabela caso estes não possuam uma correspondência com pontos do lado oposto. A versão da aplicação resultante destas alterações foi sujeita aos mesmos testes que as anteriores, tendo os resultados apresentado grandes melhorias: Os eventos 1, 3 e 5 continuam a apresentar os mesmos resultados; As restantes imagens, que anteriormente não estavam a ser lidas, conseguiram apresentar resultados, tendo-se detetado 100% dos nomes e 87% das presenças no evento 2, 100% dos nomes e 83% das presenças no evento 4 e, por fim, 90% dos nomes e 84% das presenças no evento 6. Foi apenas detetado um erro crítico nesta fase, devido à imagem 6 ter detetado o mesmo nome duas vezes. Determinou-se que, com os próximos testes, ir-se-ia aumentar a precisão destas leituras e resolver o problema dos nomes duplicados. Uma representação das melhorias e resultados feitos nesta terceira etapa de testes encontra-se na Tabela 4.

Tabela 4 - Terceira fase de testes com imagens reais

	Margin patrol now tries to delete extra points if the number of points in the bottom is different from the number of points on the top				
	Signature Margin now considers **15%** of the height or width, no longer a fixed value				
	Margin Patrol Range	Point Fuse Range	Signature threshold	Close coordinate delete distance	Name reader similarity
	30	**10**	**0.01**	**20**	**70**
	Participants Detected		Correct Presences		
Event 1 - signed.png	19/21 (90%)		37/38 (97%)		
Event 2 - signed.png	24/24 (100%)		42/48 (87%)		
Event 3 - signed.png	16/17 (90%)		31/32 (97%)		
Event 4 - signed.png	18/18 (100%)		30/36 (83%)		
Event 5 - signed.png	21/21 (100%)		42/42 (100%)		
Event 6 - signed.png	18/20 (90%)		32/38 (84%)		

No último dos testes deste género, os objetivos foram resolver o problema de nomes duplicados e melhorar a qualidade das leituras realizadas. De modo a melhorar as leituras, foi ajustada a percentagem de margens relativa aos segmentos das assinaturas. De modo a resolver o problema dos duplicados, foi adicionada uma nova otimização em que se comparam os nomes iguais que foram lidos e se assume apenas aquele que tenha uma maior semelhança com o nome original. Desta forma assegura-se que, caso hajam duplicados, apenas é considerado aquele que é provavelmente o original. O resultado destas otimizações foi testado e foram possíveis obter melhorias em todos os resultados, sendo que os eventos 2, 4 e 5 conseguiram acertas 100% dos nomes enquanto que os restantes detetaram 90%. Os eventos 1, 3 e 5 detetaram 100% das presenças enquanto que os eventos 2 e 6 detetaram 94% e o evento 4 de detetou 92% de presenças, respetivamente. Uma representação destes resultados encontra-se na Tabela 5.

Tabela 5 - Quarta fase de testes com imagens reais

	Name reader now checks if it's the second time reading the same name and returns "None" if it's not as similar as the previous detection				
	Margin patrol now tries to delete extra points if the number of points in the bottom is different from the number of points on the top				
	Signature Margin now considers **35%** of the height and **30%** width				
	Margin Patrol Range	Point Fuse Range	Signature threshold	Close coordinate delete distance	Name reader similarity
	30	**10**	**0.01**	**20**	**70**
	Participants Detected		Correct Presences		
Event 1 - signed.png	19/21 (90%)		38/38 (100%)		
Event 2 - signed.png	24/24 (100%)		45/48 (94%)		
Event 3 - signed.png	16/17 (90%)		32/32 (100%)		
Event 4 - signed.png	18/18 (100%)		33/36 (92%)		
Event 5 - signed.png	21/21 (100%)		42/42 (100%)		
Event 6 - signed.png	18/20 (90%)		34/36 (94%)		

6.4.2. Determinação do melhor input

Foram ainda realizados vários testes manuais de modo a determinar as melhores definições a adotar na captura e processamento da imagem (Anexo A – Determinação de parâmetros).

Após ser concluída uma versão estável da aplicação, foram fornecidas várias versões da mesma imagem que variavam de alguma forma entre si. Pretendia-se com isto testar qual o tipo de imagem que permitia obter os melhores resultados. Para uma imagem ser considerada boa, esta teria de ter um tamanho leve, um bom tempo de execução, acertar as presenças totais, lendo corretamente os nomes e detetando corretamente as assinaturas nos períodos da manhã e da tarde.

Os testes realizados recorrem ao *plugin cordova-plugin-camera*, sendo ajustadas várias das suas configurações, nomeadamente, ao nível da qualidade de fotos do tipo *jpeg*. Outra característica é que imagens lidas por este *plugin* terão dimensões alteradas por vezes. Estes testes foram realizados cinco vezes para cada imagem, sendo os resultados uma média dessas cinco vezes. Na tabela referente a estes testes encontram-se especificadas duas versões da aplicação, sendo a 3.9.0. a última versão desenvolvida até à data (21/11/2020).

Concluiu-se com este teste que, compensa muito mais em termos de tamanho utilizar imagens *jpeg*, pois possuem muito menos dados devido ao facto de não serem *lossless*, ao contrário das imagens *png* utilizadas inicialmente. Isto é benéfico porque torna mais rápido o processo de deteção de assinaturas e evita sobrecarregar os pedidos que são feitos através de *HTML* para enviar a imagens da *app* para a *API*. As dimensões da imagem também impactam bastante o tamanho desta, sendo possível optar por resoluções menores e obtendo a mesma qualidade das leituras. No final, determinou-se que os melhores parâmetros a utilizar são as dimensões 1440 *x* 1080 com o valor 25 para a qualidade da imagem obtida pelo *plugin*.

7. Conclusão

Após o desenvolvimento deste projeto, foi possível criar, com sucesso, a solução para leitura de tabelas de presenças e respetivas assinaturas, cumprindo com os requisitos esperados do *POC*. Neste capítulo encontra-se exposta uma análise do trabalho desenvolvido, assim como uma perspetiva relativa ao seu desenvolvimento futuro.

7.1. Análise Crítica

Com este projeto foi possível o desenvolvimento de uma solução para o *POC* pretendido, capaz de identificar os participantes num evento que assinalaram a sua presença no respetivo documento da *APM*. Esta solução recorreu a uma abordagem baseada em diversas técnicas de *CV*, exigindo a criação de abordagens como o *MP*, e recorrendo a técnicas como *OCR*. Foram realizados vários testes de modo a comprovar o funcionamento do trabalho, incluindo exemplos reais, que obtiveram altas taxas de precisão. No seu estado final, a solução de leitura de tabelas existe como um *package python* instalável e documentado, preparado para possivelmente integrar os sistemas já existentes da *APM*.

Para este projeto foi necessário o estudo das diversas soluções já desenvolvidos pela *md3* para a *APM*, assim como a integração com a equipa responsável por esses mesmos projetos. Parte deste processo exigiu a aprendizagem de novas tecnologias (por exemplo, *Ionic* e *Django)* e a adoção da metodologia *Scrum*, utilizada internamente na empresa. Considera-se que o projeto foi concluído com sucesso, tendo sido comprovado que é de facto possível interpretar os conteúdos de uma folha de presenças da *APM* com recurso a técnicas de *CV*.

7.2. Trabalho Futuro

Futuramente, a solução criada poderá ser integrada nos sistemas da *APM*, juntamente com as funções pretendidas com os seus requisitos. Poderá também incorporar um sistema de validação das assinaturas, de modo a comprovar a identidade de quem assinou a folha de presenças. Atualmente, é assumido que se um espaço destinado a uma assinatura se encontra preenchido, é porque deve ser contada uma presença nesse mesmo lugar. Contudo, isto poderá não corresponder à realidade, levando à necessidade por uma camada extra de segurança sobre o processo.

Bibliografia

[1] "apm," [Online]. Available: https://www.apm.fr/. [Accessed 13 October 2020].

[2] "Ionic Framework," [Online]. Available: https://ionicframework.com/#. [Accessed 17 October 2020].

[3] "django," [Online]. Available: https://www.djangoproject.com. [Accessed 17 October 2020].

[4] "md3," [Online]. Available: https://www.md3.pt/. [Accessed 13 October 2020].

[5] A. Turner, "bankmycell," [Online]. Available: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world. [Accessed 21 11 2020].

[6] J. Clement, "Statista," 17 1 2020. [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed 21 11 2020].

[7] J. Zhang, C. Calabrese, J. Ding, M. Liu and B. Zhang, "Advantages and challenges in using mobile apps for field experiments: A systematic review and a case study," *Mobile Media & Communication,* vol. 6, no. 2, pp. 179-196, 2018.

[8] S. Chan, "Global App Revenue Reached $50 Billion in the First Half of 2020, Up 23% Year-Over-Year," SensorTower, 30 6 2020. [Online]. Available: https://sensortower.com/blog/app-revenue-and-downloads-1h-2020. [Accessed 21 11 2020].

[9] S. Bellman, R. F. Potter, S. Treleaven-Hassard, J. A. Robinson and D. Varan, "The Effectiveness of Branded Mobile Phone Apps," *Journal of Interactive Marketing,* vol. 25, no. 4, pp. 191-200, 2011.

[10] H. v. Heerde, I. Dinner and S. A. Neslin, "Creating Customer Engagement Via Mobile Apps: How App Usage Drives Purchase Behavior," *Tuck School of Business Working Paper No. 2669817, Available at SSRN: https://ssrn.com/abstract=2669817 or http://dx.doi.org/10.2139/ssrn.2669817,* 2015.

[11] "Mobile Operating System Market Share Worldwide," StatCounter, [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide. [Accessed 21 11 2020].

[12] M. Huynh, P. Ghimire and D. Truong, "Hybrid App Approach: Could It Mark The End Of Native App Domination?," *Issues in Informing Science and Information Technology,* vol. 14, pp. 49-65, 2017.

[13] S. Sasidaran, "Survey on Native and Hybrid Mobile Application Development Tools," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. 6, no. 9, pp. 1389-1393, 2017.

[14] S. Al-Fedaghi, "Developing Web Applications," *International Journal of Software Engineering and its Applications,* vol. 5, no. 2, pp. 57-68, 2011.

[15] IBM Software, *Thought Leadership White Paper: Native, web or hybrid mobile-app development,* 2012.

[16] A. Biørn-Hansen, T. A. Majchrzak and T.-M. Grønli, "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development," *Proceedings of the 13th*

International Conference on Web Information Systems and Technologies , pp. 344-351, 2017.

[17] T. Ater, Building Progressive Web Apps, O'Reilly, 2017.

[18] "Can I use Service Workers?," [Online]. Available: https://caniuse.com/serviceworkers. [Accessed 12 11 2019].

[19] C. Griffith, "What is Hybrid App Development?," [Online]. Available: https://ionicframework.com/resources/articles/what-is-hybrid-app-development. [Accessed 21 11 2020].

[20] S. Liu, "Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020," statista, 2 7 2020. [Online]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. [Accessed 21 11 2020].

[21] M. Kremer, "Comparing Cross-Platform Frameworks," Ionic, [Online]. Available: https://ionicframework.com/resources/articles/ionic-vs-react-native-a-comparison-guide. [Accessed 22 11 2020].

[22] VisualContext Digital, "Different Approaches To Mobile App Development," 10 2018. [Online]. Available: https://visualcontext.net/wp-content/uploads/2019/07/White-Paper-Native-Apps-Hybrid-PWA.pdf. [Accessed 22 11 2020].

[23] "Native APIs," Ionic, [Online]. Available: https://ionicframework.com/docs/native. [Accessed 22 11 2020].

[24] "Core Components and Native Components," Facebook, 29 10 2020. [Online]. Available: https://reactnative.dev/docs/intro-react-native-components. [Accessed 22 11 2020].

[25] "Trusted Web Activity," Google, [Online]. Available: https://developers.google.com/web/android/trusted-web-activity. [Accessed 22 11 2020].

[26] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.

[27] K. Sage and S. Young, "Security applications of computer vision," *IEEE Aerospace and Electronic Systems Magazine,* vol. 14, no. 4, pp. 19-29, 1999.

[28] G. J, Y. Y, L. P and P. DS, "Computer Vision in Healthcare Applications," *Journal of Helthcare Engineering,* vol. 2018, 2018.

[29] ADDEPTO, "Computer Vision White Paper 2020," 2020. [Online]. Available: https://addepto.com/wp-content/uploads/2020/07/Computer-Vision-White-Paper-2020.pdf. [Accessed 22 11 2020].

[30] J. E. Solem, Programming Computer Vision with Python, O'Reilly Media, Inc., 2012.

[31] U. Sinha, "SuDoKu Grabber in OpenCV," [Online]. Available: https://aishack.in/tutorials/sudoku-grabber-opencv-plot/. [Accessed 22 11 2020].

[32] "SnapSudoku," [Online]. Available: https://github.com/prajwalkr/SnapSudoku. [Accessed 22 11 2020].

[33] S. Mori, C. Suen and K. Yamamoto, "Historical Review of OCR Research and Development," *Proceedings of the IEEE,* vol. 80, no. 7, pp. 1029-1058, 1992.

[34] MICROSCAN, "Technology White Paper: Understanding Optical Character Recognition," 2012. [Online]. Available: https://files.microscan.com/whitepapers/wp_ocr.pdf. [Accessed 23 11 2020].

[35] K.Karthick, K.B.Ravindrakumar, R.Francis and S.Ilankannan, "Steps Involved in Text Recognition and Recent Research in OCR; A Study," *International Journal of Recent Technology and Engineering (IJRTE),* vol. 8, no. 1, pp. 3095-3100, 2019.

[36] S. Singh, "Optical Character Recognition Techniques: A Survey," *Journal of Emerging Trends in Computing and Information Sciences,* vol. 4, no. 6, pp. 545-550, 2013.

[37] R. Smith, "An Overview of the Tesseract OCR Engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2007.

[38] R. Hercik, R. Slaby, Z. Machacek and J. Koziorek, "Correlation Methods of OCR Algorithm for Traffic Sign Detection Implementable in Microcontrollers," in *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions (pp.381-389)*, 2013.

[39] N. Hazim, M. Abomaali and M. Al-Mayyahi, "An Efficient Character Recognition Technique Using K-Nearest Neighbor Classifier," *International Journal of Engineering & Technology,* vol. 7, no. 4, pp. 3148-3153, 2018.

[40] M. Sabourin and A. Mitiche, "Optical character recognition by a neural network," *Neural Networks,* vol. 5, no. 5, pp. 843-852, 1992.

[41] PayLane, "RESTful API Whitepaper," [Online]. Available: https://static.paylane.com/docs/whitepapers/en/restful-api-whitepaper.pdf. [Accessed 24 11 2020].

[42] "Manifesto for Agile Software Development," 2001. [Online]. Available: https://agilemanifesto.org. [Accessed 24 October 2020].

[43] J. Highsmith and A. Cockburn, "Agile software development: the business of innovation," *Computer,* vol. 34, no. 9, pp. 120-127, 2001.

[44] C. Silva, A. Souza and B. Tavares, "Risk management analysis in Scrum software projects," *International Transactions in Operational Research,* vol. 26, no. 5, pp. 1884-1905, 2017.

[45] L. Rising and N. S. Janoff, "The Scrum software development process for small teams," *IEEE Software,* vol. 17, no. 4, pp. 26-32, 2000.

[46] D. Pandey, U. Suman and A. Ramani, "An Effective Requirement Engineering Process Model for Software Development and Requirements Management," in *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, Kottayam, India, 2010.

[47] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, Non-Functional Requirements in Software Engineering, Springer US , 2000.

[48] M. Glinz, "On Non-Functional Requirements," in *5th IEEE International Requirements Engineering Conference (RE 2007)*, Delhi, India, 2007.

[49] R. Malan and B. D., *Functional Requirements and Use Cases,* 1999.

[50] "OpenCV," OpenCV team, [Online]. Available: https://opencv.org/. [Accessed 23 11 2020].

[51] B. D. Dunka, D. E. A. Emmanuel and D. O. Oyeyinka, "HYBRID MOBILE APPLICATION BASED ON IONIC FRAMEWORK TECHNOLOGIES," *International Journal of Recent Advances in Multidisciplinary Research,* vol. 4, no. 12, pp. 3121-3130, 2017.

[52] C. Rieger and T. A. Majchrzak, "Towards the definitive evaluation framework for cross-platform app development approaches," *The Journal of Systems and Software,* vol. 153, pp. 175-199, 2019.

[53] N. Ibraheem, M. Hasan, R. Khan and P. Mishra, "Understanding Color Models: A Review," *ARPN Journal of Science and Technology,* vol. 2, no. 3, pp. 265-275, 2012.

[54] T. Kumar and K. Verma, "A Theory Based on Conversion of RGB image to Gray image," *International Journal of Computer Applications,* vol. 7, no. 2, pp. 7-10, 2010.

[55] E. Gedraite and M. Hadad, "Investigation on the Effect of a Gaussian Blur in Image Filtering and Segmentation," in *ELMAR*, 2011.

[56] P. Roy, S. Dutta, N. Dey, G. Dey, S. Chakraborty and R. Ray, "Adaptive thresholding: A comparative study," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kanyakumari, 2014.

[57] M. Comer and E. Delp, "Morphological operations for color image processing," *Journal of Electronic Imaging,* vol. 8, no. 3, p. 279–289, 1999.

[58] R. Srisha and A. Khan, "Morphological Operations for Image Processing : Understanding and its Applications," in *National Conference on VLSI, Signal processing & Communications*, 2013.

[59] S. Suzuki and K. A. Be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing,* vol. 30, no. 1, pp. 32-46, 1985.

[60] G. Bradsky and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, Sebastopol, CA: O'Reilly, 2008.

[61] "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM,* vol. 15, no. 1, pp. 11-15, 1972.

[62] A. S. Hassanein, S. Mohammad, M. Sameer and M. E. Ragab, " Survey on Hough Transform, Theory, Techniques and Applications," *IJCSI International Journal of Computer Science Issues,* vol. 12, no. 1, pp. 139-156, 2015.

Anexo A – Determinação de parâmetros

					Resultados Acertados					
	Nome	Qualidade da Imagem	Dimensoes da imagem	Tamanho da imagem	Presencas identificadas	Nomes	Manhas	Tardes	Tempo de execução (media x/5)	
	Versão 3.5.0									
	original.png	100	4640 × 2610	2,7 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	11,39s	
	sent_from_ionic.png	100	4640 × 2610	11,5 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	11,54s	
	sent_from_ionic_1.png	100	1422x799 (2000x800 no ionic)	1,3 MB	**21 / 22 (95%)**	21 / 21 (100%)	**10 / 21 (48%)**	**16 / 21 (76%)**	10,73s	
	sent_from_ionic_2.png	100	1024x576 (1024x768 no ionic)	725 KB	22 / 22 (100%)	22/22 (100%)	**4 / 22 (18%)**	**13 / 22 (59%)**	10,78s	
	sent_from_ionic_3.png	**90**	4640 × 2610	11,5 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	11,65s	A qualidade não é considerada
	sent_from_ionic_4.png	**0**	4640 × 2610	11,5 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	11,56s	
	sent_from_ionic_5.png	100	1280x720	1,1 MB	22 / 22 (100%)	22 / 22 (100%)	**11/22 (50%)**	**17/22 (77%)**	10,70s	
	sent_from_ionic_6.png	100	1920x1080	2,5 MB	22 / 22 (100%)	22 / 22 (100%)	**18/22 (82%)**	**20/22 (91%)**	11,39s	
	sent_from_ionic_7.png	100	2560x1440	4,2 MB	22 / 22 (100%)	22 / 22 (100%)	**17/22 (77%)**	**16/22 (73%)**	11,17s	
	sent_from_ionic.**jpeg**	100	4640 × 2610	**5 MB**	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	12,51s	Comparado com o ".png" é mais leve e tão bom, mas mais lento
	iphone_photo.jpeg	100	3264x2448	5,7 MB	22 / 22 (100%)	22 / 22 (100%)	**19/22 (86%)**	**21 /22 (95%)**	11,47s	
	iphone_photo_1.jpeg	100	1066x800 (2000x800 no ionic)	885 KB	22 / 22 (100%)	22 / 22 (100%)	**16/22 (73%)**	**20/22 (91%)**	11,60s	
	iphone_photo_2.jpeg	100	960x720 (1280x720 no ionic)	725 KB	22 / 22 (100%)	22 / 22 (100%)	**14/22 (64%)**	**20/22 (91%)**	11,09s	
	iphone_photo_q.jpeg	80	3264x2448	1,6 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	12,00s	
	iphone_photo_k.jpeg	80	1080x810 (1920x1080 no ionic)	204 KB	22 / 22 (100%)	22 / 22 (100%)	**16/22 (73%)**	22 / 22 (100%)	11,60s	Tive de alterar ranges no programa
	iphone_photo_t.jpeg	50	3264x2448	682 KB	22 / 22 (100%)	22 / 22 (100%)	**21 /22 (95%)**	22 / 22 (100%)	11,10s	
	iphone_photo_p.jpeg	25	3264x2448	356 KB	22 / 22 (100%)	22 / 22 (100%)	**21 /22 (95%)**	22 / 22 (100%)	11,76s	
	iphone_photo_i.jpeg	0	3264x2448	305 KB	22 / 22 (100%)	22 / 22 (100%)	**19/22 (86%)**	22 / 22 (100%)	11,09s	
	iphone_photo_i_1.jpeg	0	1440x1080 (1920x1080 no ionic)	85 KB	**21 / 22 (95%)**	21 / 21 (100%)	**14/21 (67%)**	21 / 21 (100%)	11,25s	
Melhor desta versão	iphone_photo_p_1.jpeg	25	1440x1080 (1920x1080 no ionic)	84 KB	22 / 22 (100%)	22 / 22 (100%)	**20/22 (91%)**	22 / 22 (100%)	11,64s	
	iphone_photo_p_2.jpeg	25	960x720 (1280x720 no ionic)	54 KB	**21 / 22 (95%)**	21 / 21 (100%)	**16/21 (76%)**	21 / 21 (100%)	11,13s	
	Versão 3.9.0									
	original.png	100	4640 × 2610	2,7 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	10,50s	
	sent_from_ionic_1.png	100	1422x799 (2000x800 no ionic)	1,3 MB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	10,18s	
	sent_from_ionic_2.png	100	1024x576 (1024x768 no ionic)	725 KB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	10s	
	iphone_photo_2.jpeg	100	960x720 (1280x720 no ionic)	725 KB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	10,18s	
Melhor desta versão	iphone_photo_p_1.jpeg	25	1440x1080 (1920x1080 no ionic)	84 KB	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	22 / 22 (100%)	10,32s	
	iphone_photo_p_2.jpeg	25	960x720 (1280x720 no ionic)	54 KB	**20/22 (91%)**	20/20 (100%)	20/20 (100%)	20/20 (100%)	10,63s	