Mestrado em Engenharia Informática

Estágio

Relatório Final

Conceção e desenvolvimento de uma plataforma de gestão de serviços SaaS para o setor do alojamento - integração de serviços

Bruno Marcelo Almeida Afonso

afonso@student.dei.uc.pt

Orientador (DEI):

Professor Paulo Rupino da Cunha

Orientadores (IPNlis):

Eng. Carlos Lopes

Resumo

Nos dias de hoje, as Tecnologias da Informação apresentam um papel preponderante na agilização dos processos do dia-a-dia de um hotel. Estes processos têm diversos focos, submetendo o hoteleiro para a utilização de vários serviços. Desta forma, devido à diversidade existente, os utilizadores deparam-se com diversos problemas, como por exemplo, dificuldade de aprendizagem de múltiplas interfaces, ou redundância de informação entre diferentes serviços. Com a finalidade de solucionar estes problemas, surgiu o projeto *Hotelcracy Apps*, que tem como objetivo o desenvolvimento de uma plataforma *Software-as-a-Service* para o setor hoteleiro. Esta permitirá ao utilizador subscrever um conjunto de serviços SaaS atualmente existentes no mercado, e utilizá-los de forma integrada, através de uma interface homogénea entre todos. Assim, a plataforma atuará como intermediária entre o hoteleiro e os serviços subscritos, sendo que esta comunicação será executada através da *application programming interface* destes.

O presente relatório descreve o trabalho realizado para a integração de novos serviços na plataforma *Hotelcracy Apps*. Para tal, foi necessário efetuar um levantamento dos serviços atualmente existentes no mercado, e priorizar quais integrar. Sendo a plataforma responsável pela normalização dos dados provenientes das diferentes APIs, foi necessário evoluir o modelo de dados interno, para que fosse possivel a correta integração dos novos serviços. Na segunda parte do estágio, foram desenvolvidas as diversas componentes da plataforma responsáveis pelo processo de integração.

Assim, o presente estágio contribuiu positivamente para o projeto *Hotelcracy Apps* através da identificação de novos serviços SaaS passíveis de se integrar. As componentes responsáveis pelo processo de integração das novas soluções foram também desenvolvidas, o que incluiu previamente uma evolução do modelo de dados da plataforma. Estas contribuições resultaram positivamente no aumento do leque de serviços integrados na plataforma e de passível subscrição pelos utilizadores.

Palavras-Chave

Integração de Serviços, *Software-as-a-Service*, *SaaS, Property Management Software, Reputation Management, ID-Scanning,* Setor hoteleiro

Abstract

Nowadays, the Information Tecnologies present an important role on streamlining the processes of a hotels day-to-day. These processes have a very broad focus, forcing the innkeeper to use a several services. This way, due to the existent diversity, users come across with many problems. Two examples of those problems could be the learning disability to use various interfaces, or the redundancy of information between different services. With the purpose to solve these problems, emerged Hotelcracy Apps project, which aims to develop a Software-as-a-Service platform for hospitality sector. Through this, the user can subscribe a set of SaaS services available on the market, and use them in an integrated way, through a shared homogeneous interface. This way, the platform acts like an intermediary between the innkeeper and the various subscribed services, being the communication executed through their application programming interface.

This report describes the development done for the integration of new services in the Hotelcracy Apps platform. For such was necessary a knowledge about which services are actually available in the market, and prioritize which ones to integrate. Being the platform responsible for the normalization of the data provided from the APIs, it was necessary to evolve the intern data model, to make the correct integration of the new services possible. In the second part of the internship, the components responsible for the integration process were developed.

This way, the internship presented in this document had a positive contribution for the Hotelcracy Apps project through the identification of new services available to integrate. The components responsible for the integration of the new services were also developed, which included an evolution of the platform intern data model. These contributions resulted positively in an increase of the number of services integrated in the platform and available for subscription.

Keywords

Services Integration, Software-as-a-Service, SaaS, Property Management Software, Reputation Management, ID-Scanning, Hospitality Sector

Índice

Lista de tabelas

Lista de figuras

Tabela de acrónimos

Acrónimo	Descrição
IPNlis	Instituto Pedro Nunes – Laboratório de Informática e Sistemas
TIC	Tecnologias da Informação e Comunicação
OTA	*Online Travel Agency*
ERP	*Enterprise Resource Planning*
PMS	*Property Management Software*
BoB	*Best-of-Breed*
SaaS	*Software-as-a-Service*
API	*Application Programming Interface*
PT2020	Portugal 2020 - Sistema de Incentivos à Inovação Empresarial e Empreendorismo
JSON	*Javascript Object Notation*
XML	*Extensible Markup Language*
ER	Entidade-Relacionamento
TDD	*Test-Driven-Development*
MVC	*Model-View-Controller*
URL	*Uniform Resource Locator*

Capítulo 1 – Introdução

O presente documento tem como objetivo apresentar o trabalho realizado ao longo do estágio "Conceção e desenvolvimento de uma plataforma de gestão de serviços SaaS para o setor do alojamento – integração de serviços".

Este trabalho insere-se no contexto do projeto Hotelcracy Apps, cofinanciado pelo Portugal2020 – Sistema de Incentivos à Inovação Empresarial e Empreendorismo. Decorreu ao longo de 2 semestres, com início em fevereiro de 2018 e término em janeiro de 2019, no Instituto Pedro Nunes – Laboratório de Informática & Sistemas (IPNlis). O estagiário foi inserido na equipa do IPNlis, composta por dois professores universitários, que tinham o cargo de consultores do projeto Hotelcracy Apps, um gestor de projeto, e dois elementos dedicados ao desenvolvimento. O projeto onde o estágio foi inserido, tem a duração de 33 meses, iniciado a 1 de Outubro de 2016, e com término a 30 de Junho de 2019.

Neste primeiro capítulo apresenta-se o estágio, dando a conhecer o projeto enquadrador Hotelcracy Apps e as motivações subjacentes à sua criação. É também feita uma descrição das tarefas que já haviam sido concluídas no âmbito do mesmo, uma vez que já contava com 16 meses, quando foi iniciado o estágio. No final, é feita uma breve descrição da estrutura do relatório.

1.1. Enquadramento e Motivações

Na presente seção é feito um enquadramento do setor hoteleiro e são descritos alguns problemas identificados no mesmo. É ainda apresentado o projeto que os pretende solucionar e onde o presente estágio se insere.

1.1.1. A evolução das tecnologias no setor hoteleiro

Há já mais de duas décadas, as Tecnologias da Informação e Comunicação (TIC) têm ocupado um lugar de destaque no setor hoteleiro, emergindo como um fator crítico de sucesso. Consequentemente, estas contribuíram para que se iniciasse uma revolução no setor, despoletando o interesse das gigantes tecnológicas [1].

Em 1996, a Microsoft fundava a *Expedia*, uma empresa tecnológica de comércio de viagens online (OTA – Online Travel Agency), que em 2012 já recebia o título de maior agência de viagens do mundo [2]. Os números começaram a tornar-se relevantes, e logo atrás da *Expedia*, começaram a surgir outras empresas, como por exemplo, a *Priceline* (proprietária da Booking.com), que também passou a ocupar um lugar neste pódio.

No intervalo de tempo entre 2003 e 2012, a *Priceline* viu os seus lucros crescerem de 10 milhões para 1.1 mil milhões, sendo que em 2013 estimou-se que estes já haviam atingido a marca de 40 mil milhões em reservas [2]. A relevância destes dois *players*, foi de tal ordem, que logo se estimou que em 2014, representariam aproximadamente 5% dos lucros totais de publicidade da Google [2].

A velocidade com que o mercado se reinventa, tem sido a principal causa do surgimento de novas tecnologias. Estas permitem auxiliar o hoteleiro nas suas atividades diárias, bem como, contribuem na ajuda ao cliente que procura as melhores soluções em função das suas necessidades.

Em 2012, o setor das viagens já era caracterizado como um dos maiores mercados a nível de *e-commerce*, pelo que, nesta altura, cerca de 90% das cadeias de hotéis já ofereciam a possibilidade de reserva direta no *website* [3].

Inicialmente, as ferramentas TIC utilizadas resumiam-se a *Enterprise Resource Planning* (ERP) especializados, normalmente denominados de *Property Management Software* (PMS). Estes são sistemas que possibilitam gerir as principais atividades de um hotel, tais como: reservas, *check-in* e *check-out* de hóspedes e gestão de limpezas.

No entanto, começaram a ser identificadas algumas limitações subjacentes a este tipo de ferramentas. A expansão deste tipo de sistemas ficava restringida a opções apresentadas pelo fornecedor, o chamado *vendor lock-in*, impossibilitando assim a integração com outros serviços. Outra limitação prendia-se com a falta de funcionalidades que suportassem todos os processos de negócio inerentes ao setor, como por exemplo, a emissão de faturas. Também o défice no acompanhamento do mercado, de forma rápida e ágil, limitou a utilização destas ferramentas. Devido a estas limitações, tornou-se assim imperativa a busca por outro tipo de soluções que conseguissem satisfazer as necessidades do setor.

1.1.2. As tecnologias adotadas no presente e as suas limitações

O número de hóteis e alojamentos que funcionam de forma autónoma é elevado, estimando-se que representem cerca de 67% da oferta na Europa, e 30% nos Estados Unidos da América [1]. Estes são caracterizados por funcionarem sem qualquer tipo de filiação a cadeias, impondo os seus próprios padrões de serviços e normas. Esta característica tem a vantagem de possibilitar uma maior flexibilidade em termos de preço, posicionamento, dimensão e estrutura organizacional. No entanto, este nível de independência dificulta a aquisição de ferramentas TIC, uma vez que estas entidades com menos escala têm recursos mais limitados. Desta forma, em função dos meios que possuem, estes procuram as soluções que melhor consigam satisfazer as suas necessidades.

Em termos de escolha de ferramentas, são diversas as opções no setor. Estas agrupam-se em diferentes categorias, que são definidas em função das funcionalidades que dispõem. Alguns exemplos, deste tipo de categorias são: gestores de reputação (*Reputation Management Software*), gestores de canais de distribuição (*Channel Manager),* gestores de preços (*Rate Shopper*) ou ferramentas de faturação. Normalmente, as cadeias independentes acabam por adotar um conjunto de soluções *best-of-breed* (BoB) – caracterizadas por serem as melhores de cada categoria [4] – em função das suas necessidades, e que normalmente são comercializadas em modelo *Software-as-a-Service (*SaaS*)* [1]. Para corresponder às diversas necessidades do setor, em alguns casos, é necessária a subscrição de vários destes serviços de diferentes categorias, pelo hoteleiro.

A aquisição destes serviços pode tornar-se extensa e dispendiosa, uma vez que fica do lado do hoteleiro a escolha das soluções a adotar, numa vasta gama disponibilizada. Para além disso, obriga ainda a que exista um novo processo de habituação/aprendizagem de utilização sempre que um novo serviço é adotado. Existe ainda a agravante de não haver qualquer tipo de integração entre os SaaS adotados, obrigando a uma duplicação de informação nas diversas áreas, como por exemplo, a informação dos clientes.

1.1.3. A Hotelcracy Apps como solução

De forma a solucionar esta problemática existente no setor, foi criado o consórcio, entre a *Hotelcracy Software Lda* e o IPNlis, para o desenvolvimento da plataforma *Hotelcracy Apps*.

Este projeto tem como principal objetivo o desenvolvimento de uma plataforma *SaaS* que permitirá ao utilizador, subscrever, integrar, migrar, utilizar e cancelar as diversas soluções SaaS atualmente existentes no mercado. De realçar, que qualquer serviço está passível de integração na plataforma, desde que pertença a uma das categorias de serviços definidas, e que disponibilize uma *Application Programming Interface*.

A escolha dos serviços será feita no *Marketplace*, onde o hoteleiro terá à sua disposição os SaaS respeitantes às diferentes categorias, passíveis de subscrição. A integração entre os diversos serviços ficará também a cargo da plataforma *Hotelcracy Apps*, sendo que esta será responsável pela troca de informação entre as diferentes ferramentas subscritas. Um dos exemplos da integração, pode ser entre um serviço de faturação, um *Property Management Software* e um sistema de pagamentos. Os dados respeitantes a uma reserva estarão armazenados no PMS, e quando o hoteleiro desejar emitir a fatura, a plataforma será capaz de transferir estes mesmos dados para um sistema de faturação, afim de gerar a mesma. No final, os dados podem ser transferidos para o serviço responsável pelo pagamento, com a finalidade de efetuar a cobrança.

Usando a *Hotelcracy Apps*, o hoteleiro poderá ainda trocar de serviços sempre que assim entender. A plataforma será responsável pela migração dos dados entre o serviço utilizado no passado e o novo subscrito. Por exemplo, caso o hoteleiro se sinta insatisfeito com o PMS que possui, e deseje trocar, a plataforma será responsável pela transferência dos dados relativos às reservas efetuadas no PMS anteriormente utilizado para o novo subscrito.

A interação com os diversos serviços subscritos será realizada através de uma interface homogénea, abstraindo o utilizador da heterogeneidade e complexidade das interfaces nativas de cada serviço. Sempre que o utilizador desejar migrar entre SaaS da mesma categoria, a interface permanecerá semelhante, reduzindo, ou até mesmo eliminando, a curva de aprendizagem inerente a cada um dos serviços.

A solução apresentada permitirá mitigar os problemas enunciados anteriormente, quebrando barreiras que impedem a evolução das tecnologias no setor. É solucionada a questão do *vendor lock-in*, uma vez que o utilizador é livre de escolher e mudar serviços em função das suas necessidades. O processo de seleção também ficará bastante agilizado, uma vez que já estarão contemplados no *Marketplace* da plataforma os serviços mais relevantes existentes, eliminando a necessidade de o hoteleiro estar a avaliar uma panóplia de ferramentas atualmente disponibilizadas.

1.2. Estado do projeto Hotelcracy quando iniciado o estágio

Quando o presente estágio foi iniciado, o projeto Portugal2020 em que está inserido, já contava com 16 meses desde o seu início, sendo que já se encontravam terminadas algumas tarefas delineadas. No âmbito de um outro estágio, já haviam sido analisados os serviços SaaS disponibilizados atualmente, e que eram passíveis de serem integrados na plataforma. Já tinha sido criado um documento de "Personas e Cenários", que conduziu à geração do documento de "Macro-Requisitos do Sistema".

A arquitetura também já se encontrava definida, no entanto, encontrava-se ainda na primeira iteração, de três estipuladas, sendo que o trabalho a desenvolver para o presente estágio, teria influência nas restantes.

O início do estágio coincidiu com o começo de duas novas etapas do projeto, nomeadamente o desenvolvimento dos primeiros componentes da plataforma. Sendo que, a implementação do código inicial estava direcionada para serviços das categorias de Faturação, *Channel Manager* e Pagamentos. Acompanhando a execução desta etapa, também se havia iniciado a definição do modelo de dados, de forma a contemplar os serviços a integrar.

1.3. Objetivos do Estágio

O presente estágio contribui para o projeto *Hotelcracy Apps* através da integração de dois novos serviços de três diferentes categorias a designar. Esta integração permitirá assim a disponibilização de seis novas soluções SaaS no *Marketplace* da plataforma, que ficarão de passível subscrição pelos utilizadores.

Para se proceder à integração de serviços, o estagiário deve fazer uma recolha dos serviços atualmente existentes e definir aqueles que vai integrar de acordo com os requisitos previamente levantados. Posteriormente, deve ser realizada uma evolução do modelo de dados da plataforma, de forma a permitir a inserção destes novos serviços.

Na segunda parte do estágio, relativa ao segundo semestre, devem ser desenvolvidas as componentes responsáveis pelo processo de integração, produzindo paralelamente testes às mesmas. No final, o estagiário deve ser responsável pela instalação e validação das componentes desenvolvidas.

1.4. Estrutura do Relatório

Depois de definidos os principais temas a abordar no presente relatório, a estrutura final é a seguinte:

- Capítulo 2: contém uma descrição do planeamento relativo aos 2 semestres, bem como da metodologia de desenvolvimento a ser adotada, de forma a garantir a qualidade no trabalho desenvolvido;
- Capítulo 3: é composto por breves descrições dos principais conceitos subjacentes ao projeto;
- Capítulo 4: possui os resultados do processo de levantamento do estado da arte dos serviços a serem integrados. Nomeadamente das categorias de "*Property Management Systems*", "*ID Scanner*" e "*Reputation Management*";
- Capítulo 5: é composto pela exposição da arquitetura do sistema. É realizada uma explicação das diversas componentes que a integram;
- Capítulo 6: compreende uma explicação do modelo de dados da plataforma, bem como da evolução a que este foi sujeito;
- Capítulo 7: é composto por uma explicação detalhada do processo de desenvolvimento. São especificados os processos e metodologias aplicadas, bem como as ferramentas utilizadas no projeto. É também feita uma explicação das diversas componentes desenvolvidas na plataforma e como estas foram sendo testadas ao longo do desenvolvimento.

Capítulo 2 – Metodologia e Planeamento

No presente capítulo é descrita a metodologia adotada no projeto *Hotelcracy Apps*, bem como a adotada na equipa de desenvolvimento da plataforma. É também apresentado o planeamento delineado para o decorrer dos 2 semestres de estágio, assim como a execução real dos mesmos.

2.1. Metodologia Utilizada

No momento da candidatura ao Portugal2020, o consórcio definiu um planeamento geral para o projeto *Hotelcracy Apps*, que segue uma estrutura baseada no modelo em cascata (*Waterfall Model*). Este é um modelo aplicado na engenharia de software, em que as diferentes etapas do desenvolvimento de um projeto seguem de forma linear e sequencial, como apresentado na Figura 1 [5]. Cada etapa é dividida em diferentes tarefas, sendo que, o *output* é geralmente o *input* da fase seguinte, o que torna este modelo de baixa flexibilidade. Esta característica, dificulta o aproveitamento do conhecimento adquirido ao longo do projeto, uma vez que não se pode voltar atrás de modo a alterar as conclusões das etapas anteriores [6].

Figura 1 - Modelo em Cascata [7].

Tal como referido anteriormente, o desenvolvimento do projeto é assegurado por um consórcio formado por duas entidades (*Hotelcracy Software* e *IPNlis*), que seguem o mesmo planeamento. De forma a manter o sincronismo entre ambas as entidades, são realizadas reuniões com a periodicidade de 3 a 4 semanas, onde estão presentes todos os elementos de ambas as equipas. Nestas é apresentado todo o trabalho desenvolvido, por cada uma das entidades desde a última reunião. São também discutidos aspetos estruturais do projeto (ao nível de lógica de negócio ou arquitetura da plataforma), com consequentes tomadas de decisão. Na segunda parte do estágio (correspondente ao segundo semestre), o estagiário

participou ativamente nas reuniões, demonstrando ao restante consórcio o trabalho desenvolvido, na integração de novos serviços na plataforma.

Para o desenvolvimento da plataforma, foi acordado, entre os parceiros do consórcio, a aplicação de uma abordagem baseada em alguns conceitos da metodologia *Scrum*. Esta obriga a que exista um maior comprometimento da equipa, dado que o trabalho é dividido em curtas iterações, que geralmente não ultrapassam os 30 dias, denominados *sprints*. Todo o fluxo se inicia na etapa de divisão das principais tarefas em tarefas mais pequenas, que posteriormente são incluídas numa lista, o *product backlog*. Antes de iniciar um *sprint,* através da reunião de planeamento, denominada *sprint planning,* são decididas quais as tarefas a concluir durante o mesmo. Estas são retiradas do *product backlog* e passadas para o *sprint backlog*. Durante o *sprint* são efetuadas reuniões diárias, denominadas *daily scrum*, onde todos os elementos apresentam o trabalho realizado, qual o planeado, e caso existam, as condicionantes que impedem o avanço do mesmo. No final de cada *sprint*, é demonstrado todo o trabalho realizado. A metodologia referida pressupõe a existência de 3 papéis base: *product owner*, responsável por definir e priorizar as tarefas que compõem o *product backlog*; a equipa de desenvolvimento, responsável por garantir o desenvolvimento das tarefas delineadas a cada *sprint;* e o *scrum master,* responsável por assegurar que a equipa respeita e segue os valores da metodologia. A abordagem utilizada no *Scrum* permite uma monitorização constante do desenvolvimento, permitindo uma rápida reorganização de prioridades em caso de alterações nos requisitos, ou dificuldades inesperadas.

Tal como referido anteriormente, a metodologia de desenvolvimento foi baseada na *framework Scrum*, mas não aplicada de forma rígida. Foi utilizado o conceito de *sprints* de desenvolvimento*,* e as reuniões que compõem a *framework*. Os *sprints* tinham a duração de duas semanas, e no final de cada um, era agendada uma reunião, onde estavam presentes elementos das equipas de desenvolvimento. Nestas reuniões era discutido o trabalho realizado, e era feito o planeamento para o *sprint* seguinte. Para além destas, eram também realizados *daily scrum,* via *Skype*. Já durante a fase de desenvolvimento, as reuniões de consórcio começaram a estar sincronizadas com os *sprints* de desenvolvimento, realizando-se a cada 2 *sprints*.

Para discussão de outros assuntos técnicos, como por exemplo as tecnologias ou ferramentas a utilizar, eram realizadas reuniões entre as equipas de desenvolvimento, a seguir às reuniões de *sprint planning*. Quando era acordado que um dos pontos de discussão poderia ter impacto na plataforma (por exemplo, decisões arquiteturais), as tomadas de decisão eram agendadas para a posterior reunião de consórcio.

A nível interno, a equipa do *IPNlis* fazia uma reunião semanal, normalmente à quinta-feira, onde estava presente o orientador de estágio, e três *developers* da equipa. Nestas reuniões era demonstrado o trabalho realizado por cada um dos elementos durante a semana e discutidos aspetos técnicos, bem como, aspetos relacionados com o estágio.

Na Figura 2 está esquematizada a metodologia de desenvolvimento descrita anteriormente. Estão exemplificados dois *sprints* de desenvolvimento, o final da semana que os antecede, e o início da semana que os sucede. A semana antecedente tem representada a reunião técnica e de *sprint planning,* que tem impacto no *sprint 1*. Por sua vez, na semana seguinte, está representada a reunião de consórcio, onde viria a ser apresentado o trabalho desenvolvido ao longo de ambos os *sprints*.

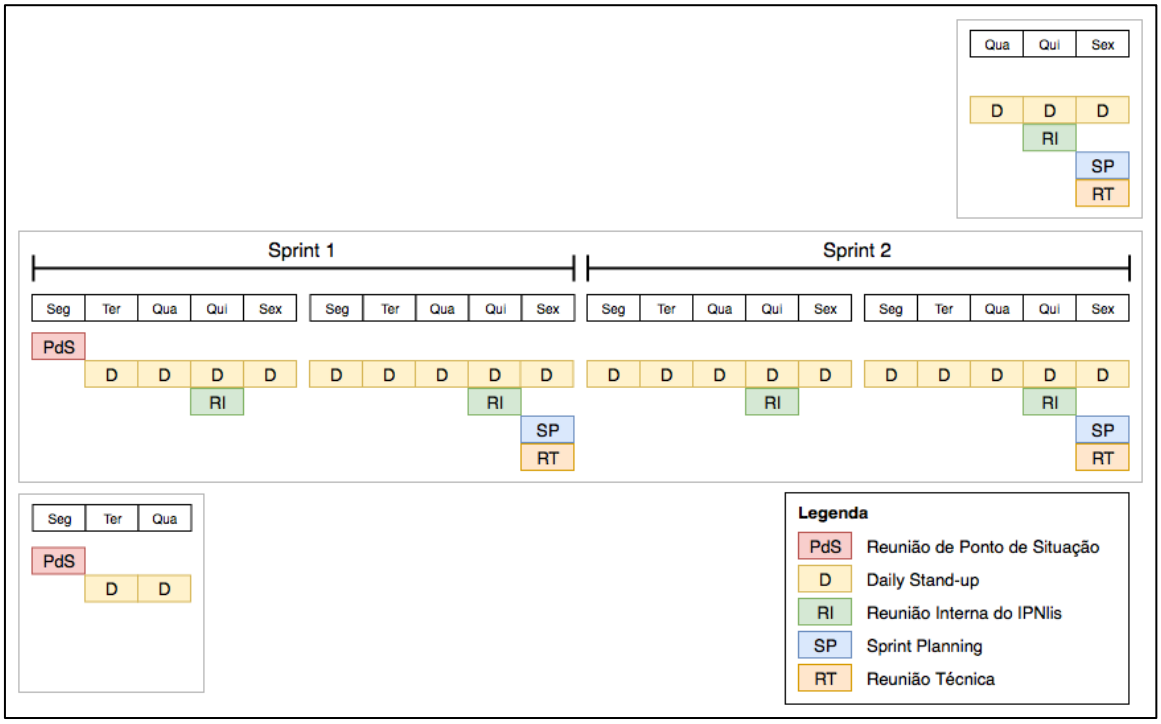

Figura 2 - Exemplo da metodologia utilizada no desenvolvimento do projeto Hotelcracy Apps [8].

2.2. Planeamento

Nesta secção, é apresentado o planeamento da primeira parte do presente estágio, bem como a evolução das diferentes tarefas ao longo dos dois semestres. São também apresentadas justificações para alterações que existiram no planeamento inicial.

2.2.1. Primeiro Semestre

Quando iniciado o estágio, as seguintes tarefas já se encontravam delineadas:

1. Análise do estado da arte e recolha de informações dos serviços SaaS a integrar;
2. Definição detalhada de requisitos e processos de integração de serviços;
3. Evolução do modelo de dados da plataforma.

Uma vez que as tarefas eram apresentadas a um nível muito macro, surgiu a necessidade destas serem decompostas, afim de se conseguir definir melhor o planeamento para o semestre. Para além das tarefas acima referidas, foi também incluída a redação do presente relatório. Para efetuar o planeamento, fez-se uso de um Diagrama de *Gantt*, apresentado na Figura 3.

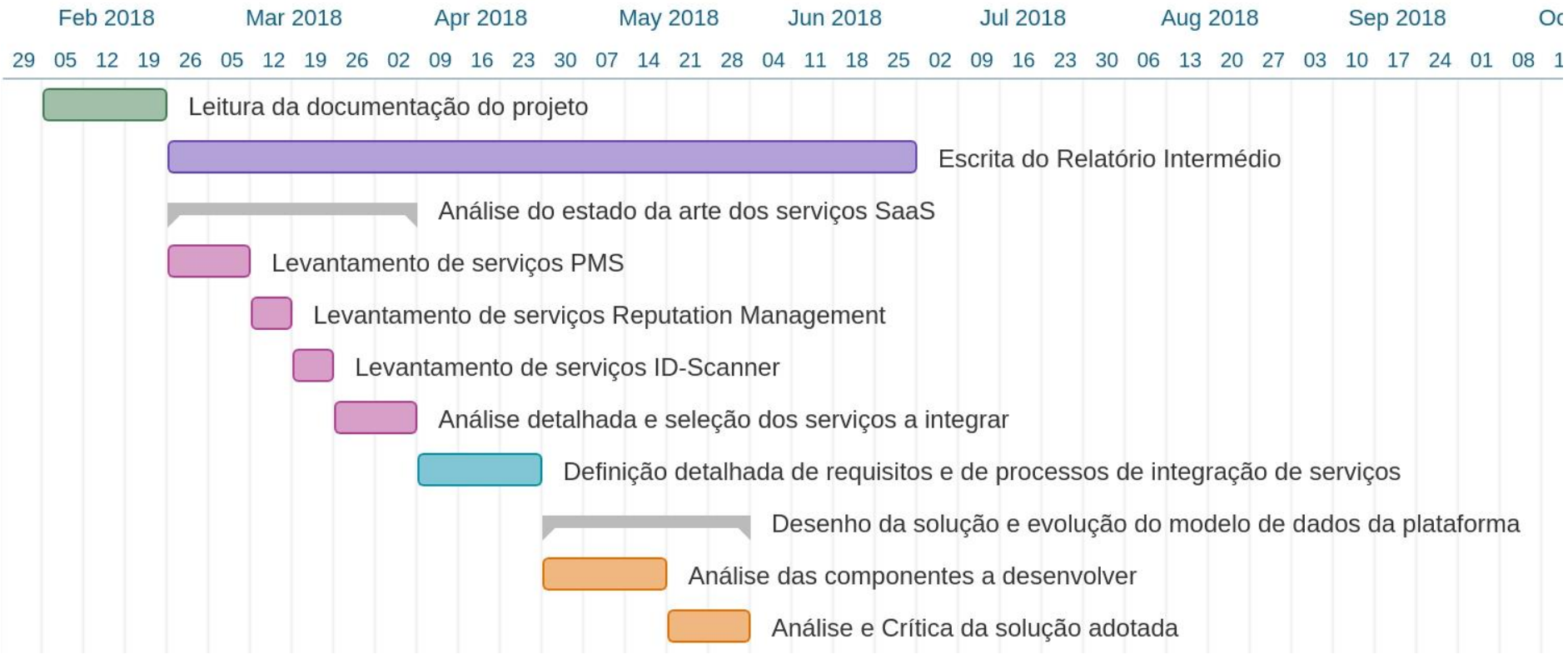

Figura 3 - Diagrama de Gantt relativo ao planeamento do 1º semestre de estágio

A Figura 4 apresenta a execução real das atividades, que haviam sido planeadas para o 1º semestre de estágio. A duração foi diferente do planeamento inicial nas seguintes tarefas:

- **Levantamento de serviços PMS**: quando foi realizado o planeamento para o primeiro semestre, previu-se que esta seria a categoria que necessitava de mais tempo para análise, devido à quantidade de serviços atualmente disponíveis. Ainda assim, o tempo planeado não foi suficiente, sendo necessária a alocação de mais uma semana para a conclusão desta tarefa.
- **Definição detalhada de requisitos e processos de integração de serviços:** esta tarefa envolveu a análise detalhada das APIs dos serviços. Uma vez que o número de serviços passíveis de integração na plataforma, foi superior ao expectável, esta tarefa necessitou de mais tempo para ser concretizada.
- **Análise das componentes a desenvolver:** através da tarefa "Leitura da documentação do Projeto", foi possível realizar uma grande parte do trabalho relativo à análise das componentes a desenvolver. A arquitetura adotada na plataforma foi logo conhecida, facultando assim conhecimento acerca das diversas componentes. Com o conhecimento global já adquirido, foi facilitada a identificação, e o modo como estas viriam a ser desenvolvidas para a integração dos novos serviços, resultando assim na redução de tempo desta tarefa.
- **Evolução do modelo de dados:** esta etapa compreendia a identificação das diferentes entidades que eram possíveis obter nas diversas APIs, por exemplo: cliente, hóteis, quartos, entre outras. Depois de identificadas, foram sendo criadas novas entidades a inserir no modelo de dados, que eram discutidas internamente na equipa do *IPNlis*. Estas discussões levaram a que houvessem sucessivas evoluções do modelo de dados, o que se refletiu no incremento do tempo para concluir a tarefa.

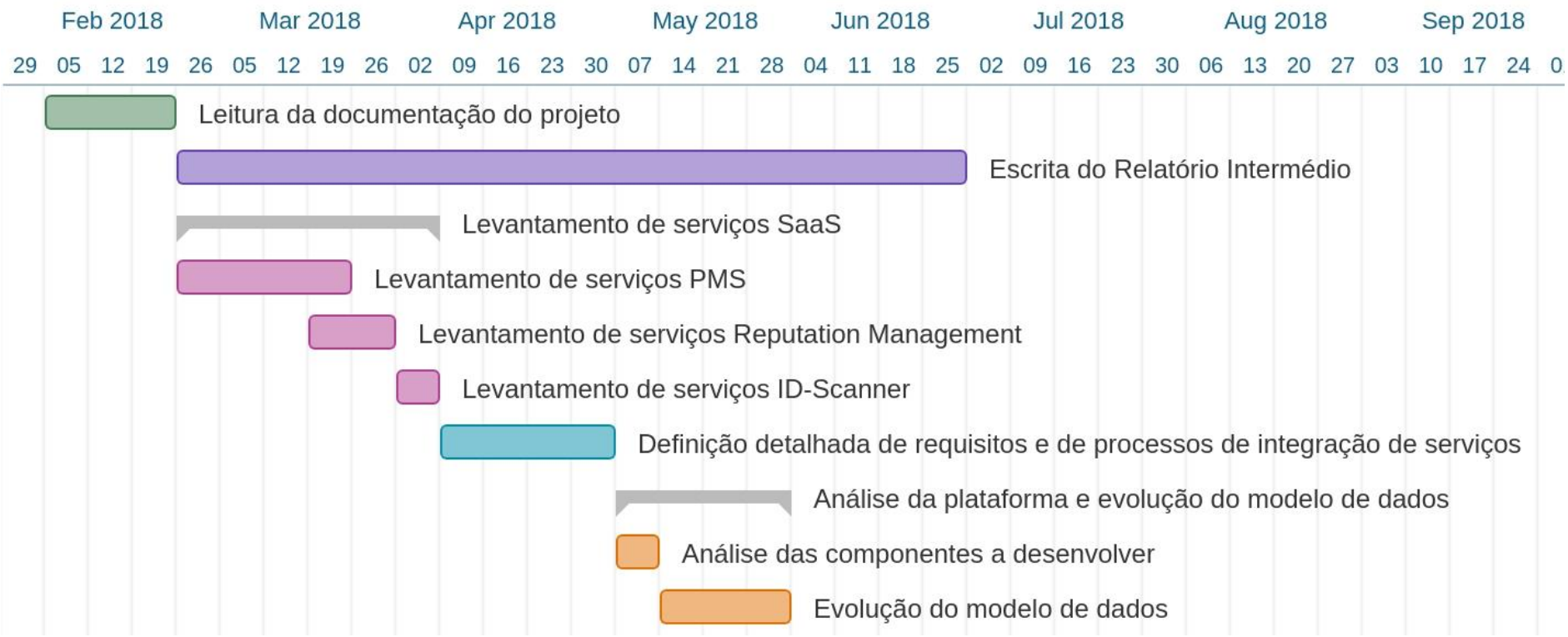

Figura 4 - Diagrama de Gantt relativo à execução do 1º semestre de estágio

2.2.2. Segundo Semestre

Para o segundo semestre as principais tarefas já se encontravam delineadas, sendo as seguintes:

- Desenvolvimento dos componentes de integração de serviços SaaS e evolução de componentes da plataforma existente;
- Definição e desenvolvimento de testes aos componentes de integração de serviços SaaS desenvolvidos;
- Instalação e validação final dos componentes.

O planeamento das tarefas a concretizar foi efetuado posteriormente ao conhecimento da metodologia de desenvolvimento da plataforma. Assim, ficou definido que sempre que um novo serviço era integrado, as 3 tarefas, acima enunciadas, eram executadas sequencialmente. De realçar que a estimativa do tempo total de integração, foi feita com base no número de *endpoints* que eram disponibilizados pelas API dos SaaS envolvidos, sendo a variação relevante. Assim, o planeamento inicial para o decorrer do segundo semestre é o apresentado na Figura 5.

Figura 5 - Diagrama de Gantt relativo ao planeamento do 2º semestre de estágio

A execução real das tarefas planeadas para o 2º semestre é apresentada na Figura 6. A grande mudança que existiu em relação ao planeamento inicial, foi nos serviços integrados.

Antes de se iniciar a integração, era necessário garantir o acesso à API dos serviços a integrar. Desta forma, foi estabelecido um contacto direto com as diversas empresas com a finalidade destas fornecerem as chaves de autenticação. Depois de contactados, alguns serviços não demonstraram interesse na realização da parceria, e outros não forneceram resposta. Este já havia sido identificado como um risco a ser considerado, e, por isso, já existia um plano de mitigação que passava por escolher o serviço imediatamente a seguir na lista de priorização. No entanto, para a categoria de PMS o problema foi além do expectável, e apenas se conseguiu o acesso à API de um serviço, denominado *Beds24*. Nos serviços de *Reputation Management*, a *Review Pro* também não demonstrou interesse, e por isso acabou por ser substituída pela *AboutMyHotel*. No entanto, esta troca não teve grande reflexo no planeamento inicial, uma vez que as APIs de ambos os serviços eram semelhantes e o modelo de dados já contemplava todas as entidades necessárias para a integração.

No entanto, para o serviço *Beds24*, teve que existir uma nova avaliação do modelo de dados da plataforma, de forma a verificar se este se encontrava adaptado para a integração do serviço. Para além do tempo despendido para a conclusão desta tarefa, o necessário para a integração total do serviço foi um pouco superior ao expectável devido a ser o primeiro serviço a ser integrado. Desta forma, o estagiário necessitou de um tempo adicional para conhecer o código das diversas componentes, bem como os diferentes fluxos existentes na plataforma.

Nos restantes serviços, o tempo de integração foi semelhante ao que se havia planeado no início do semestre. Depois de atingidos os objetivos do estágio, foi, adicionalmente, desenvolvida uma interface gráfica de forma a facilitar a apresentação final dos resultados.

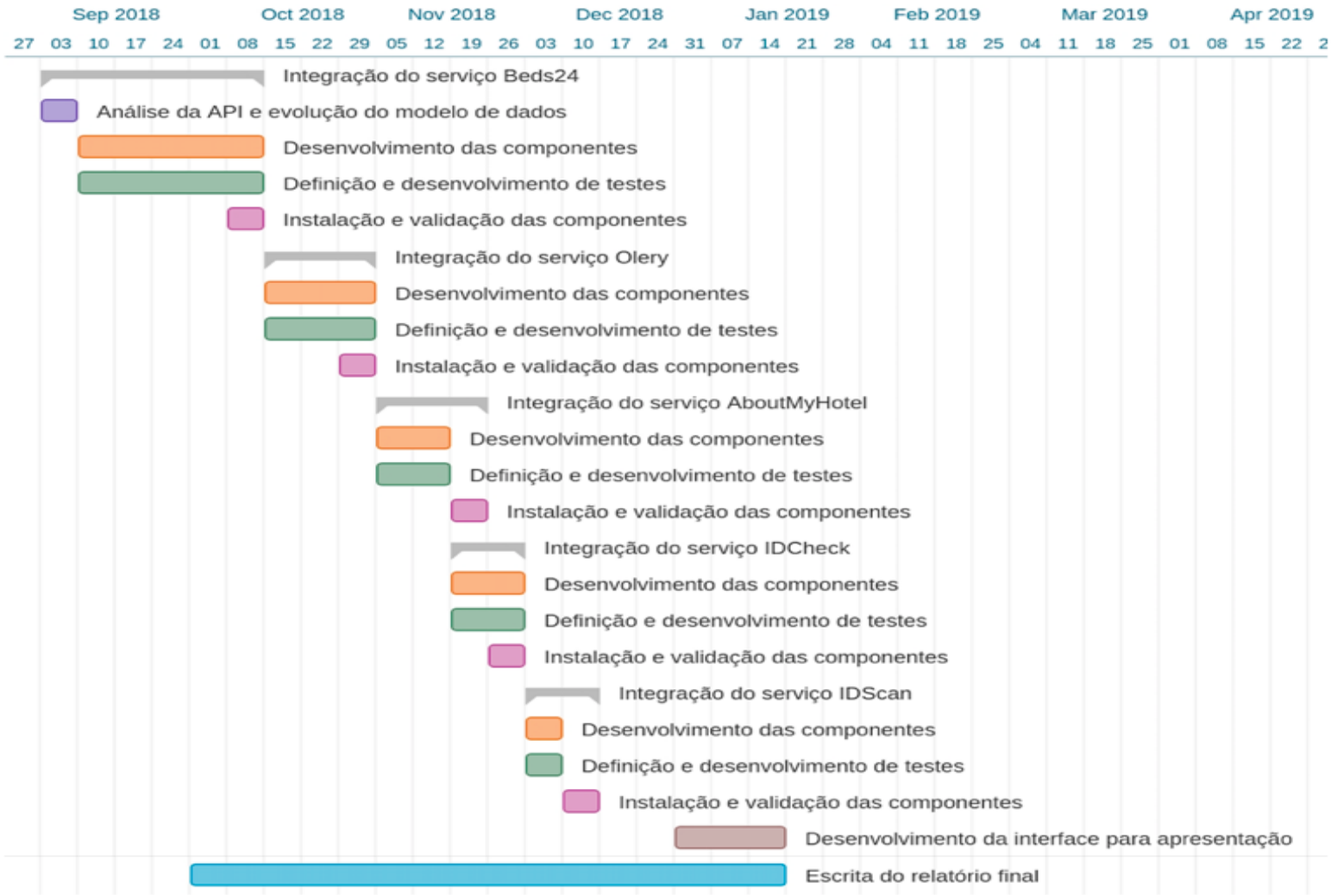

Figura 6 - Diagrama de Gantt relativo à execução do 2º semestre de estágio

2.3. Análise de Riscos

Na Engenharia de *Software*, um risco pode ser encarado como um potencial problema ou circunstância indesejada, que afeta o cumprimento dos objetivos definidos para o projeto. Assim, torna-se fulcral identificá-los e analisá-los, de forma a que se consigam gerir, e desta forma diminuir o impacto negativo no projeto.

A identificação dos riscos é feita com base na probabilidade de ocorrência, no nível de impacto, e na janela temporal onde se prevê que estes ocorram [8]. Após a identificação, são previstas as consequências que estas possam ter para o normal decorrer do projeto, bem como os planos de mitigação que possam diminuir o impacto negativo.

Um plano de mitigação surge da necessidade de se reduzir o impacto da ocorrência de um risco, previamente identificado. Como tal, é de elevada importância, o planeamento de medidas preventivas que consigam mitigar os possíveis riscos identificados, de forma a aumentar a probabilidade de sucesso do projeto.

A probabilidade de um risco consiste numa estimativa com base numa percentagem de ocorrência [8]. Os graus de probabilidade são assim divididos da seguinte forma:

- Baixa: <40%;
- Média: 40% a 70%;
- Alta: >70%.

O grau de impacto é igualmente uma medida qualitativa, definida com base no esforço que se vai ter que aplicar para atingir objetivos obrigatórios, para que se considerar o projeto bem sucedido [8]. Desta forma, os graus de impacto são os seguintes:

- Insignificante: os critérios de sucesso podem ser atingidos sem grande dificuldade;
- Crítico: os critérios de sucesso podem ser atingidos com grande esforço/custo;
- Catastrófico: os critérios de sucesso não podem ser atingidos.

Por último, a janela temporal consiste no intervalo de tempo estimado, que o risco se vai perpetuar [8]. Esta escala divide-se da seguinte forma:

- Curta: poucas semanas;
- Média: entre 1 e 3 meses;
- Longa: > 3 meses.

2.3.1. Critérios de Sucesso

De forma a quantificar o sucesso do presente estágio, foram definidos os seguintes critérios de sucesso a alcançar:

- Devem ser integrados 2 serviços de cada uma das 3 categorias diferentes;
- A integração deve ser concluída dentro da data limite de entrega;
- O código produzido deve respeitar as boas práticas intrínsecas ao projeto.

2.3.2. Levantamento de Riscos

Depois de efetuada uma análise do trabalho a realizar no decorrer do presente estágio, combinado com as condutas de identificação de riscos descritas anteriormente, definiram-se os seguintes riscos:

ID: R01	Probabilidade: Alta	Impacto: Crítico	Janela Temporal: Curta
Risco: Algumas marcas responsáveis pelos serviços passíveis de integração na plataforma não percecionam benefício na colaboração com a Hotelcracy. Há um risco de não fornecerem as chaves para acesso à API.			
Plano de Mitigação: Procurar o serviço imediatamente seguinte no ranking planeado			

Tabela 1 - Risco 01

ID: R02	Probabilidade: Média	Impacto: Insignificante	Janela Temporal: Curta
Risco: O Modelo de dados da plataforma é de elevada complexidade. Existe o risco de posteriormente à evolução do mesmo, este ainda não permitir a integração completa dos serviços.			
Plano de Mitigação: Reformular modelo de dados de forma a contemplar os serviços escolhidos			

Tabela 2 - Risco 02

ID: R03	Probabilidade: Baixa	Impacto: Crítico	Janela Temporal: Média
Risco: As tecnologias no setor hoteleiro estão num constante ambiente de mudança em função das necessidades. Há um risco dos parâmetros exigidos e devolvidos pelas APIs dos serviços integrados sofrerem alterações.			
Plano de Mitigação: Reformulação dos pedidos efetuados, e leitura das respostas dadas pelos serviços			

Tabela 3 - Risco 03

ID: R04	Probabilidade: Média	Impacto: Crítico	Janela Temporal: Longa
Risco: O estagiário nunca foi integrado numa equipa de desenvolvimento a nível empresarial. Há um risco deste não conseguir seguir o ritmo de desenvolvimento expectável.			
Plano de Mitigação: Alocar mais tempo para o desenvolvimento			

Tabela 4 - Risco 04

ID: R05	Probabilidade: Média	Impacto: Crítico	Janela Temporal: Longa
Risco: O estagiário nunca desenvolveu seguindo exaustivamente uma metodologia de desenvolvimento. Há um risco de adaptação por parte do mesmo.			
Plano de Mitigação: Reuniões regulares com os orientadores de estágio, e developers do projeto			

Tabela 5 - Risco 05

Capítulo 3 – Principais Conceitos Base

Ao longo do presente capítulo, são descritos diferentes conceitos relevantes para a contextualização do leitor, no âmbito do projeto em que o presente estágio foi inserido. Em cada secção será descrito um conceito, de forma breve e sucinta.

3.1. Software-as-a-Service

Software-as-a-Service é definido como uma forma de distribuição e comercialização de *software*. Através deste modelo, o fornecedor é inteiramente responsável pelo fornecimento de toda a estrutura para a disponibilização do *software* (servidores, segurança, conectividade). Do outro lado, o acesso por parte do cliente é realizado via *browser* da *internet* [9].

Assim, este tipo de software define-se principalmente, pelas seguintes características:

- Atualizações são automaticamente aplicadas sem intervenção do cliente;
- O serviço é adquirido num método de subscrição;
- O cliente apenas necessita de um dispositivo com acesso à *internet*, sendo que não é necessário qualquer outro tipo de *hardware* [10].

Podemos concluir, que os benefícios da adoção deste tipo de serviços se prendem essencialmente com os custos inerentes ao suporte do mesmo. Não necessitando de ter custos associados à aquisição de hardware, o cliente fica também ilibado das ações de manutenção e *setup* inicial do mesmo. Para além disto, visto que este tipo de serviço funciona através de um método de subscrição, o cliente apenas paga pelo que efetivamente utiliza. No entanto, ao ficar ilibado de todos os custos enunciados anteriormente, o utilizador vai estar a pagar por uma coisa que nunca vai ser efetivamente sua, à semelhança de um arrendamento. Para além disto, este estará sempre sujeito a ter que se adaptar a pequenas mudanças que existam no *software* subscrito.

3.2. Application Programming Interface (API)

Uma API, é definida como um conjunto de métodos e padrões, estabelecidos por um software, que permitem a utilização dos seus serviços [11]. Tem como objetivo principal a disponibilização de funcionalidades para terceiros, funcionando assim como um intermediário de diálogo entre duas aplicações distintas [12]. Esta comunicação é estabelecida através dos chamados *endpoints*, que funcionam como porta de entrada para as *apps* alheias ao serviço fornecedor da API.

De forma a entender melhor o conceito, tomemos um exemplo prático. Imaginemos que o leitor pretende desenvolver um sistema de *login* através da conta de *Facebook*. Este cenário é possível, devido ao serviço criado por Mark Zuckerberg possuir uma API que disponibiliza *endpoints* para validar os dados de uma conta ali registada. Desta forma, quando um utilizador pretender efetuar o *login* no produto desenvolvido pelo leitor, os dados serão enviados para o *endpoint* da API, que será responsável por verificar a validade dos mesmos. Desta forma, do lado da aplicação apenas se tem que aguardar pela resposta fornecida, sem ter que executar qualquer tipo de ação.

3.3. Categoria de Serviços em Estudo

As categorias de serviços referem-se a grupos de ferramentas que são utilizadas atualmente no setor hoteleiro, e que são definidas e caracterizadas em função das funcionalidades que dispõem. No total, são identificadas 14 categorias, no entanto, para o presente estágio, foram escolhidas 3 categorias a estudar: *Property Management System* (PMS), *Reputation Management e ID-Scanning*. Estas categorias estão descritas nos subcapítulos seguintes.

3.3.1. Property Management System (PMS)

Nos dias de hoje, os PMSs assumem um grau de importância muito relevante para a gestão hoteleira, pois permitem o controlo de todas as ações centrais de um hotel. Algumas das suas funções são: gestão de reservas, gestão de clientes, *check-in* e *check-out*, controlo de limpeza dos quartos, gestão de fólios, geração de relatórios de contas, ou até mesmo controlo de faturação. Exemplos deste tipo ferramentas são: *Cloudbeds* [13] e *Beds24* [14].

3.3.2. *Reputation Management*

A velocidade com que as críticas se expandem nos dias de hoje, obriga a que os serviços hoteleiros aumentem o nível de importância atribuído às mesmas. Assim, com o objetivo de facilitar a procura de críticas, nasceram as ferramentas de *Reputation Management*.

Através do uso destas soluções, é possivel acompanhar, num único local, críticas e avaliações feitas às propriedades do hoteleiro, provenientes de diversas fontes, melhorando assim, a rapidez com que se tem acesso a estas informações. Depois de visualizar as avaliações feitas ao seu serviço, o hoteleiro pode focar-se em melhorar aspetos que estejam com uma menor cotação, ou manter aqueles que se encontram bem cotados.

Existem ainda algumas ferramentas desta categoria, que permitem submeter os clientes a questionários personalizados, indo mais rapidamente ao encontro, das avaliações que os hoteleiros pretendem obter. Exemplos deste tipo de *SaaS* são: *Olery* [15] e *Review Pro* [16].

3.3.3. *ID-Scanning*

Este é um tipo de serviço bastante simples, mas que apresenta uma grande utilidade. Através do uso de uma câmara de um *smartphone*, ou de computador, é realizada uma digitalização do documento de identificação apresentado pelo cliente. Posteriormente, a foto é enviada para a aplicação, e automaticamente são obtidos uma série de dados pessoais referentes ao cliente.

Este tipo de ferramentas apresentam uma grande utilidade, uma vez que permitem, em questão de segundos, a obtenção em formato *text* dos dados disponíveis no documento, evitando que os mesmo tenham que estar a ser inseridos através do teclado. Para além disto, a aplicação faz uma validação do documento fornecido, permitindo assim identificar documentos falsificados. Exemplo deste tipo de *software* são: *IdScan* [17] e *IdCheck* [18].

3.4. Modelo Canónico

Para que todos os serviços integrados na plataforma desenvolvida no projeto *Hotelcracy* consigam "falar" entre si, é necessária a criação de um protocolo de comunicação interno. Este protocolo será o "dicionário" utilizado por todas as ferramentas adotadas, garantindo assim que estas se expressam na mesma "língua".

Este protocolo é concretizado através do desenho de um modelo canónico, resultante do mapeamento dos parâmetros presentes nas APIs dos diversos serviços integrados. Desta forma, é possível criar uma linguagem comum a todos. Assim sendo, podemos entender que o modelo canónico será a linguagem mediadora de comunicação.

Tomemos um exemplo prático para explicar o papel principal do modelo canónico. Estão três pessoas numa sala, de três nacionalidades distintas. A pessoa de nacionalidade inglesa, quer comunicar com a pessoa de nacionalidade alemã, no entanto, não consegue, visto que não falam a mesma língua. Para garantirem a comunicação entre ambas, estas têm que fazer recurso da terceira pessoa, que é um português e que tem em sua posse dicionários de tradução de ambas as línguas para português, e vice-versa. Ao receber a mensagem em Inglês, este irá traduzir a mensagem para português, e de seguida, para conseguir comunicar com o destinatário da mensagem, irá traduzir a mensagem de português para alemão.

Assim, podemos concluir que neste caso, a linguagem portuguesa é assumida como o modelo canónico, e ambos os dicionários utilizados são os resultados da modelagem dos diversos sistemas para a linguagem do modelo canónico.

3.5. Integração de Serviços

No contexto da plataforma, considera-se um serviço integrado se este conseguir estabelecer comunicações com múltiplas soluções disponibilizadas na plataforma *Hotelcracy Apps*. O processo de comunicação entre serviços é iniciado e intermediado pelas diversas componentes da plataforma, sendo que estas serão também responsáveis por fazer a invocação dos métodos nas diferentes APIs. Para que seja possível garantir a correta comunicação para os *endpoints,* a plataforma será também responsável pela tradução dos dados no sentido plataforma-serviço, e serviço-plataforma, fazendo recurso ao modelo canónico definido. Tal como já referido, a existência de API nos diversos serviços, é obrigatória, uma vez que esta é o ponto de conexão entre as diversas soluções e a plataforma.

Capítulo 4 – Análise de serviços das diversas categorias

No presente capítulo é feita uma descrição do processo de levantamento, e análise de serviços SaaS para a indústria hoteleira atualmente existentes no mercado. Para esta avaliação foram contempladas as categorias em estudo, já referidas anteriormente.

Esta etapa iniciou-se a 26 de fevereiro de 2018, com o objetivo de definir os serviços mais prioritários a integrar no decorrer do segundo semestre. Para as categorias de *Reputation Management* e PMS, o método de avaliação foi muito semelhante. No entanto, devido a limitações encontradas nos serviços de *ID-Scanning*, este teve um método diferente, e portanto, será explicado de forma separada.

A análise dos documentos produzidos no estágio "Estudo e desenvolvimento de uma plataforma de gestão de serviços SaaS para o setor do alojamento – Subscrição e Cancelamento de Serviços", do aluno José Carlos Miranda, foram o ponto de partida desta etapa. Durante o seu estágio, este aluno já havia feito um levantamento dos serviços de 2 das 3 categorias em estudo.

4.1. Property Management System & Reputation Management

Dos documentos produzidos pelo aluno José Carlos Miranda, foram considerados para avaliação um total de 24 serviços da categoria de PMS, e 10 serviços da categoria de *Reputation Management*. Posteriormente iniciou-se uma nova etapa, que consistia no levantamento de SaaS até então não considerados.

4.1.1. Levantamento e Análise de novos serviços

Concluída a análise dos documentos, iniciou-se a etapa de levantamento de novos serviços não contemplados nos resultados anteriormente referidos.

A pesquisa de novos serviços desenvolveu-se através do uso de diversas *keywords*, definidas pelo estagiário, e aplicadas no motor de busca *Google*. As *keywords* utilizadas para a categoria de PMS, foram as seguintes:

- *"Property Management Software"* – 01 de março de 2018
- *"Hotel Property Management Software"* – 06 de março de 2018
- *"Hotel Property Manager Software"* – 09 de março de 2018
- "Hotel PMS SaaS" – 09 de março de 2018
- "PMS SaaS"- 11 de março de 2018
- "Property Management SaaS" – 11 de março de 2018
- *"PMS API"* – 15 de março de 2018
- *"Property Management Software API"* – 19 de março de 2018
- "Property Management SaaS API"- 20 de março de 2018

Quando concluído o levantamento de serviços *PMS,* iniciou-se o levantamento de SaaS de *Reputation Management*. Para a recolha de ferramentas desta categoria, foram utilizadas as seguintes *keywords*:

- *"Reputation Manager"* - 21 de março de 2018;
- *"Online Reputation Manager Software"* – 22 de março de 2018;
- *"Online Reputation Manager SaaS"* – 23 de março de 2018;
- *"Reputation Manager SaaS API"* – 26 de março de 2018;
- *"Reputation Manager API"* – 28 de março de 2018;

Ao longo da pesquisa, em ambas as categorias, foram sendo progressivamente feitos refinamentos das *keywords* de pesquisa utilizadas em função dos dados anteriormente conhecidos. Numa fase final, começou-se a fazer uso da palavra "API" de forma a ir ao encontro de serviços que disponibilizassem acesso à mesma.

Depois de encontrados os novos serviços, passou-se à fase de análise. No entanto, para se prosseguir com esta etapa, era necessária a existência de parâmetros que conseguissem caracterizar os diversos SaaS encontrados. Assim, de forma a também efetuar uma verificação da consistência dos dados presentes nos documentos do aluno José Carlos Miranda, foram utilizados os mesmos parâmetros lá presentes. Com efeito, os campos utilizados foram os seguintes:

- API*:* parâmetro que indicava a existência/não-existência de API no serviço em análise;
- Protocolo API*:* protocolo utilizado para desenho da API;
- Formato dos dados: formato dos dados utilizados pela API;
- Acesso à API*:* tipo de utilizadores a que era disponibilizado o acesso à API;
- Documentação da API: parâmetro que indicava a existência/não-existência de documentação da API.

Durante a análise, em quase todos serviços, as informações procuradas eram obtidas no *website* do serviço em análise. No entanto*,* foi também possível obter informações acerca dos serviços avaliados, através dos seguintes *websites*:

- www.capterra.com: Este é um *website* que tem como objetivo ajudar os utilizadores na busca de *software* atualmente existente no mercado. Através de parâmetros de análise, comum a todos os serviços disponiveis na plataforma, é possível ao utilizador fazer uma comparação entre as diversas ferramentas, e decidir qual a melhor em função das suas necessidades. Para além disto, é também possível visualizar *reviews* de outros utilizadores ao serviço em questão. Esta plataforma foi adquirida em 2015 pela Gartner, revelando assim um nível de fiabilidade alto, para utilização dos dados disponibilizados [19];
- www.programmableweb.com: *website* que possui análises detalhadas de diversas *APIs* de serviços existentes.

Na Tabela 6, é possível visualizar os resultados de todo o processo anteriormente descrito, para a categoria de *Property Management Software*.

ID	Nome	API	Protocolo API	Formato dos Dados	Acesso à API	Documentação da API
PMS_01	Alacer- Hotel Manager	Não	N/A	N/A	N/A	N/A
PMS_02	Amadeus PMS	Sim	SOAP	XML	N/D	N/D
PMS_03	Beds24	Sim	N/D	JSON/XML	Subscritores	Sim
PMS_04	Base7Booking	Não	N/A	N/A	N/A	N/A
PMS_05	BookingCenter	Não	N/A	N/A	N/A	N/A
PMS_06	Booking Factory	Sim	REST	JSON	Aberto	SIM
PMS_05	Clock PMS	Sim	REST	XML/JSON/YAML	Subscritores	Sim
PMS_07	CloudBeds	Sim	N/D	JSON	Aberto	Sim
PMS_08	Cubilis	Não	N/D	N/D	N/D	N/D
PMS_09	Elina	Sim	N/D	XML	N/D	N/D
PMS_10	Frontdesk Anywhere	Não	N/A	N/A	N/A	N/A
PMS_11	Guestline – Rezlynx	Não	N/A	N/A	N/A	N/A
PMS_12	HS/Hotelsoftware	Não	N/A	N/A	N/A	N/A
PMS_13	Hetras	Sim	N/D	N/D	Aberto	Sim
PMS_14	HostPMS	Não	N/A	N/A	N/A	N/A
PMS_15	Hotelogix PMS	Sim	REST	XML	N/D	Sim
PMS_16	Hotline Software	Não	N/A	N/A	N/A	N/A
PMS_17	Hoteliga	Sim	REST	JSON/XML	N/D	Sim
PMS_18	Hostaway	Não	N/A	N/A	N/A	N/A
PMS_19	Hotelcinquestelle	Sim	N/D	JSON	Sob-pedido	N/D
PMS_20	IQpms	Sim	REST	N/D	N/D	N/D
PMS_21	InovGuest	Não	N/A	N/A	N/A	N/A
PMS_22	Inkey PMS	Não	N/A	N/A	N/A	N/A
PMS_23	InnQuest	Não	N/A	N/A	N/A	N/A
PMS_24	MSI Cloud PMS	Não	N/A	N/A	N/A	N/A
PMS_25	Micros	Não	N/A	N/A	N/A	N/A
PMS_26	Newhotel PMS	Não	N/A	N/A	N/A	N/A
PMS_27	Oracle - OPERA	Não	N/A	N/A	N/A	N/A
PMS_28	PropertyMe	Não	N/A	N/A	N/A	N/A
PMS_29	Protel PMS	Não	N/A	N/A	N/A	N/A
PMS_30	RoomKey PMS	Sim	REST	N/D	N/D	Sim
PMS_31	RoomMaster	Não	N/A	N/A	N/A	N/A
PMS_32	Roomraccoon	Não	N/A	N/A	N/A	N/A
PMS_33	Resnexus	Não	N/A	N/A	N/A	N/A
PMS_34	Skyware Hospitality Solutions	Não	N/A	N/A	N/A	N/A
PMS_35	SoftPMS	Não	N/A	N/A	N/A	N/A
PMS_36	Seekom iBex	Não	N/A	N/A	N/A	N/A
PMS_37	SkyTouch	Não	N/A	N/A	N/A	N/A
PMS_38	Sirvoy	Não	N/A	N/A	N/A	N/A
PMS_39	VerticalBooking	Sim	N/D	XML	Subscritores	N/D
PMS_40	Webrezpro	Não	N/A	N/A	N/A	N/A
PMS_41	Zak by Wubook	Sim	RPC	XML	N/D	Sim
PMS_42	StayNTouch	Não	N/A	N/A	N/A	N/A
PMS_43	Execu/Suite	Não	N/A	N/A	N/A	N/A
PMS_44	StayNTouch	Não	N/A	N/A	N/A	N/A
PMS_45	LittleHotelier	Não	N/A	N/A	N/A	N/A
PMS_46	Rezlynz PMS	Não	N/A	N/A	N/A	N/A

N/A – Não Aplicável N/D – Não Disponível

Tabela 6 - Resultados do processo de levantamento e análise de PMSs atualmente existentes

Olhando para os resultados, é possível concluir que foram encontrados 22 novos serviços, identificados com a cor verde, sendo que, do total, 5 possuíam acesso via API. Desta forma, no final foram contemplados 11 serviços com acesso à API, uma vez que nos documentos do aluno José Carlos Miranda, estavam identificados 6. Desta forma, foi possível concluir que o número de soluções apresentadas atualmente é considerado elevado, muito provavelmente devido ao forte crescimento tecnológico no setor hoteleiro.

Na Tabela 7 é possível visualizar os resultados do processo de levantamento de serviços da categoria de *Reputation Management*.

ID	Nome	API	Protocolo API	Formato dos Dados	Acesso à API	Documentação da API
ID_1	About my hotel	Sim	REST	XML/JSON	Parceiros/Aberto	Não
ID_2	Barqar	Não	N/A	N/A	N/A	N/A
ID_3	Birdeye	Sim	REST	JSON	Aberto	Sim
ID_4	Brandseye	Sim	N/D	N/D	Subscritores	N/D
ID_5	Brandwatch	Sim	N/D	JSON	Aberto	Sim
ID_6	CrimsonHexagon	Sim	N/D	JSON	Subscritores	Sim
ID_7	Customer-Alliance	Não	N/A	N/A	N/A	N/A
ID_8	ebuzzconnect	Não	N/A	N/A	N/A	N/A
ID_9	Falcon.io	Não	N/A	N/A	N/A	N/A
ID_10	FieldSolutionGroup	Não	N/A	N/A	N/A	N/A
ID_11	FutureSolutionsMedia	Não	N/A	N/A	N/A	N/A
ID_12	Grade.us	Sim	N/D	JSON	Subscritores	Sim
ID_13	Meltwater	Sim	N/D	N/D	Necessita de Registo	Sim
ID_14	Oktopost	Sim	REST	JSON	Aberto	N/D
ID_15	Olery	Sim	N/D	JSON	Mediante pedido	Sim
ID_16	Podium	Não	N/A	N/A	N/A	N/A
ID_17	Rankur	Sim	REST	N/D	Aberto, mas limitado	Sim
ID_18	RavenTools	Sim	REST	XML/JSON	Aberto	Sim
ID_19	RenegadeWorks	Não	N/A	N/A	N/A	N/A
ID_20	Reputation Aegis	Sim	N/D	JSON/XML	Necessita de Credenciais	Sim
ID_21	Reputize	Sim	REST	N/D	N/D	N/D
ID_22	Review Pro	Sim	N/D	JSON	Mediante pedido	Sim
ID_23	Review Wave	Não	N/A	N/A	N/A	N/A
ID_24	ReviewInc	Não	N/A	N/A	N/A	N/A
ID_25	ReviewTrackers	Sim				
ID_26	Revinate	Não	N/A	N/A	N/A	N/A
ID_27	RevuKangaroo	Não	N/A	N/A	N/A	N/A
ID_28	SproutSocial	Não	N/A	N/A	N/A	N/A
ID_29	TrustYou	Sim	N/D	N/D	N/D	Sim

N/D – Não Disponível N/A – Não Aplicável

Tabela 7 - Resultados do processo de levantamento e análise de serviços de Reputation Management atualmente existentes

Através dos resultados, é possível concluir que foram encontrados 20 novos serviços, identificados com a cor verde, sendo que 10 possuíam acesso à API. Depois de realizada a análise desta categoria, concluiu-se que o número de ferramentas atualmente disponiveis é significativo. No entanto, os setores para que estas se vocacionam foi divergindo, não ficando apenas confinado à hotelaria, mas também, por exemplo, à restauração.

Depois de concluída esta etapa, passou-se à seguinte, que compreendia uma análise detalhada da API de cada um dos serviços.

4.1.2. Análise da API dos serviços

Antes de se iniciar a análise da API dos serviços encontrados na fase anterior, foi necessário efetuar uma filtragem de forma a garantir que as soluções a avaliar preenchiam os requisitos mínimos para integração.

Tal como já referido anteriormente, a existência de API é um requisito crítico para integrar um serviço. Deveria igualmente possuir documentação da mesma, de forma a que os *developers* conseguissem obter as informações necessárias para efetuar a integração do serviço. Com efeito, todos os serviços que não possuíssem API, ou documentação da mesma, seriam descartados logo à partida.

Para garantir que os PMS a integrar possuíam versatilidade na API, foi estabelecido que estes deveriam possuir pelo menos um *endpoint* para modificar, criar e listar dados. Por sua vez, devido à natureza das ferramentas de *Reputation Management*, apenas era exigido que estas possuíssem *endpoints* para listar.

Depois de estabelecidos os requisitos de admissão, foram especificados os parâmetros de análise a aplicar a cada uma das APIs. Definiram-se os seguintes campos de avaliação:

- Popularidade: de forma a quantificar a popularidade do serviço em questão, foi utilizado o *website* "www.capterra.com", onde era disponibilizado um valor quantitativo que variava entre 0 e 5.
- Preço: custos monetários associados à utilização da API;
- Número de pedidos: limite máximo de pedidos a realizar à API*;*
- Nível da documentação: uma vez que todos os serviços a analisar teriam que possuir documentação, foram estabelecidos níveis para caracterizar a sua qualidade:
    - Nível 1: Existir documentação, mas sem conteúdo relevante;
    - Nível 2: Documentação apresentar conteúdo, mas insuficiente para esclarecer métodos existentes, e parâmetros de entrada;
    - Nível 3: Documentação apresentar métodos, e parâmetros de entrada e saída;
    - Nível 4: Documentação apresentar métodos, parâmetros de entrada e saída, e ter exemplos de teste;

Os resultados da avaliação da API dos PMSs, através dos parâmetros anteriormente referidos, são os apresentados na Tabela 8.

#	ID	Nome	Preço	Número de Pedidos	Criar Entidades	Obter Entidades	Modificar Entidades	Popularidade	Tem Documentação
1	PMS_02	Amadeus PMS							Não
2	PMS_03	Beds24	7.90€/mês	Sem limite	X	X	X	5*	Sim - 2
3	PMS_05	Clock PMS	S/Resposta	5pedidos/min	X	X	X	5*	Sim - 2
4	PMS_07	CloudBeds	S/Custos	Sem limite	X	X	X	5*	Sim - 3
5	PMS_09	Elina							Não
6	PMS_13	Hetras	S/Custos	Sem limite	X	X	X	S/Cotação	Sim - 3
7	PMS_15	Hotelogix PMS	S/Resposta	S/Resposta	X	X	X	4*	Sim - 3
8	PMS_17	Hoteliga				X			Sim
9	PMS_19	Hotelcinquestelle				X			Sim
10	PMS_20	IQpms							Não
11	PMS_30	RoomKey PMS				X			Sim
12	PMS_39	VerticalBooking							Não
13	PMS_46	*Zak by Wubook	S/Custos	Sem limite	X	X	X		Sim

Tabela 8 - Resultados do processo de análise detalhada das APIs

Identificados com a cor verde, podemos visualizar que no total foram 5 os serviços que passaram em todos os critérios de admissão. Por sua vez, os SaaS identificados com a cor vermelha, não cumpriam os requisitos impostos, e por isso não foram alvo de avaliação. De referir a situação excecional do serviço "*Zak by Wubook*", que apesar de passar em todos os

critérios de admissão, não foi avaliado. Esta situação ocorreu devido a estes assumirem que a sua API se considerava obsoleta, uma vez que já tinham em desenvolvimento uma nova. No entanto, considerou-se deixar o serviço na tabela, com a possibilidade de ser uma solução a integrar no futuro.

Em algumas situações, não foi possível obter as informações relativas aos parâmetros em análise. Estas não se encontravam no *website* respeitante ao serviço, ou na documentação da API. E mesmo após o contacto via *e-mail,* não foi possível conseguir todas as respostas. Nestes casos, o campo ficou assinalado com o valor "S/Resposta".

Como já referido, no total foram analisados 5 serviços. Contudo, para integração seriam apenas considerados 2, de forma a ir ao encontro dos objetivos do presente estágio. Assim, foi estabelecida uma ordem de priorização, de forma a selecionar os serviços que seriam analisados na última etapa, que correspondeu à análise detalhada dos *endpoints* das APIs. Para estabelecer a ordem de priorização, foram contemplados os parâmetros de "Nível da Documentação" e "Popularidade".

No final, os serviços identificados como mais prioritários foram o "*Cloudbeds"* e o "*Hotelogix"*. No entanto, definiu-se mais um serviço de *backup*, para no caso dos serviços selecionados não conseguirem dar a resposta esperada. Desta forma, o serviço "*Hetras"* também entrou nesta lista final. Posteriormente, a proposta de soluções a integrar foi enviada aos principais *stakeholders* do projeto, afim destes verificarem e validarem.

Para a categoria de *Reputation Management*, o procedimento adotado foi igual ao anteriormente descrito. Com efeito, depois de efetuada a análise das diferentes APIs, os resultados foram os apresentados na Tabela 9.

#	ID	Nome	Preço	Número de Pedidos	Obter Entidades	Popularidade	Tem Documentação
1	RM_01	About my Hotel	---	---	---	---	Não
2	RM_04	Brandseye	Mail	Mail		5*	Sim
3	RM_05	Brandwatch	S/Custos	30/10min	X	4*	Sim - 2
4	RM_06	CrimsonHexagon	Mail	Mail	X	5*	Sim – 3
5	RM_12	Grade.us	200$/mês		X	5*	Sim - 2
6	RM_13	Meltwater	Mail	Mail		4*	Sim
7	RM_14	Oktopost	---	---	---	---	Não
8	RM_15	Olery	1500€/ano	S/limite	X	S/Reviews	Sim - 2
9	RM_17	Rankur	98$/mês	1000/24h	X	S/Reviews	Sim - 2
10	RM_20	Reputation Aegis	39$/mês	S/limite	X	5*	Sim - 2
11	RM_21	Reputize	---	---	---	---	Não
12	RM_22	Review Pro	S/Custos	S/limite	X	S/Reviews	Sim - 2
13	RM_29	TrustYou	Mail	Mail	X	S/Reviews	Sim - 1

Tabela 9 - Resultados do processo de análise detalhada das APIs dos serviços de Reputation Management

Olhando para a tabela, é possível visualizar três cores diferentes atribuídas aos serviços. A vermelho, estão identificados os serviços que não passaram nos critérios de admissão. Com a cor verde, estão identificados os que passaram nos critérios de admissão e que foram avaliados. Por fim, os identificados com a cor amarela, necessitavam de credenciais para acesso à documentação da API, e mesmo depois de ser enviado e-mail a explicar o propósito da investigação, não foi fornecido qualquer tipo de resposta. Para além da informação das credenciais, à semelhança do que já havia acontecido na categoria anterior, algumas informações foram impossíveis de obter.

No final, com base nos parâmetros de "Popularidade" e "Nível da Documentação", foi estabelecida uma ordem de priorização, que contemplava os serviços *CrimsonHexagon* e

Grade.us. No entanto, depois da decisão ser enviada aos principais *stakeholders* do projeto, esta lista foi alterada, uma vez que estes sugeriram outros serviços mais vocacionados para o setor hoteleiro. Assim, os *stakeholders* identificaram que os serviços *Olery* e *ReviewPro* eram os melhores cotados dentro dos que correspondiam às necessidades do projeto, ficando assim como os mais prioritários.

4.1.3. Análise dos requisitos dos serviços

Para a análise dos requisitos a cumprir, a metodologia utilizada foi diferente entre as categorias de serviços. Esta diferença deveu-se às informações que estavam disponíveis nos documentos de projeto, "Personas e Cenários" e "Macro-requisitos do sistema". No primeiro documento, são apresentados os resultados da identificação das principais utilidades das ferramentas a integrar na plataforma. Consequência desta tarefa, houve uma definição de personas, e hipotéticos cenários de utilização do sistema Hotelcracy. O segundo documento, vai ao encontro das necessidades expressas no primeiro. Através da concretização de diversas tarefas, foi possível identificar um conjunto de macro-componentes com responsabilidades distintas e complementares, que permitem suprir as necessidades expressas nos diferentes cenários já definidos [20].

Enquanto que para a categoria de *Property Management System*, todos os requisitos a cumprir se encontravam presentes nos documentos, para a categoria de *Reputation Management*, ainda não se encontravam objetivos especificados. Desta forma, para estes últimos serviços foi inicialmente efetuado um estudo acerca das funcionalidades normalmente disponibilizadas, de forma a se conseguir especificar os requisitos a cumprir.

4.1.3.1. Property Management Software

Ao longo dos diferentes cenários expostos no documento de "Personas e Cenários", estava identificada a categoria que dava resposta ao cenário descrito, facilitando a identificação de cenários referentes a PMSs. Por sua vez, o documento de "Macro-requisitos", já pressuponha que o estagiário soubesse a categoria de serviços que deveria cobrir o requisito especificado. Depois de analisados os documentos, foram identificados no total 37 requisitos a cobrir.

Uma vez identificados os requisitos, passou-se à fase seguinte. Esta contemplou um mapeamento das *APIs* dos serviços escolhidos, de forma a listar as funcionalidades passíveis de se implementar. Posteriormente, foi realizado um *match* entre os requisitos levantados e as funcionalidades identificadas, cujo os resultados foram os apresentados na Tabela 10. Por forma a identificar de que documento o requisito foi retirado, criou-se uma notação. Por exemplo, "PC_C01", refere-se a que o requisito foi levantado através do documento de "Personas e Cenários", e do cenário com o ID número 1.

Descrição Requisito	Prioridade	Cenário	Hotelogix	Cloudbeds	Hetras	Hotelcracy
Registo de Hotel, e suas configurações	Must	PC_C01				X
Editar Configurações do Hotel	Must	PC_C08				
Listar Configurações do Hotel	Must	PC_C08	X	X	X	
Adicionar Quarto						X
Remover Quarto						X
Editar Quarto						
Visualizar detalhes de quarto				X	X	
Listar quartos				X	X	
Adicionar serviços/produtos do hotel						
Editar serviços/produtos do hotel						
Produzir relatório de Gestão	Must	PC_C04				
Consultar relatório de Gestão	Must	PC_C04			X	
Adicionar nova reserva	Must	PC_C13	X	X	X	X
Editar reserva	Must	PC_C13	X	X		X
Remover reserva	Must	PC_C13	X		X	X
Reserva – Check-in	Must	MR_T9		X	X	
Reserva – Check-out	Must	MR_T10		X	X	
Listar Detalhes da Reserva				X	X	
Editar Detalhes de Reserva			X	X		
Listar Reservas	Must	PC_C13	X	X		
Adicionar Convidado				X		
Listar Convidados				X		
Criação de Fólio	Must	PC_C09		X	X	X
Edição de Fólio	Must	PC_C09		X	X	X
Adicionar cliente	Must	PC_C14		X	X	X
Editar cliente	Must	PC_C14		X	X	X
Visualizar cliente	Must	PC_C14		X	X	
Listar histórico de cliente	Must	PC_C14				
Consulta de conta corrente do cliente	Must	PC_C14				
Criar Limpeza de Quarto	Should	PC_C20				
Remover Limpeza de Quarto	Should	PC_C20				
Listar Limpezas de Quartos	Should	PC_C20				
Gestão de Mensagens Internas	Should	PC_C21				
Monitorização de Atividade	Should	PC_C11				
Adicionar propriedade	Should	PC_C22				
Listar propriedades	Should	PC_C22		X	X	
Remover propriedade	Should	PC_C22				

Tabela 10 - Resultados do levantamento de requisitos e análise dos endpoints da categoria de PMS

No final da análise, foi possível verificar que o serviço Hotelogix, apesar de se apresentar como dos mais prioritários, cobria uma parte muito pequena dos requisitos especificados, e por isso foi excluído. Assim, foram definidos os restantes dois serviços como mais prioritários para implementação: *Cloudbeds* e *Hetras*.

4.1.3.2. Reputation Management Software

Tal como já referido, de forma a se identificarem os requisitos a cumprir, foi necessário efetuar um estudo acerca das funcionalidades esperadas por serviços desta categoria. Algumas das funcionalidades identificadas provieram essencialmente dos *websites* dos serviços em questão. No entanto, grande parte dos requisitos a cumprir foram resultantes de uma discussão com os principais *stakeholders* do projeto.

Depois de definidos os requisitos, fez-se um mapeamento das APIs dos serviços, de forma a verificar se estas possuíam métodos que cobrissem os requisitos especificados. Quando concluído todo o processo, os resultados foram os apresentados na Tabela 11.

Descrição Requisito	Review Pro	Olery
Obter rating relativo ao serviço	X	X
Obter rating relativo à limpeza	X	X
Obter rating relativo à localização	X	X
Obter rating relativo à gastronomia	X	X
Obter rating relativo aos quartos	X	X
Obter rating relativo à recepção	X	X
Obter citações nas redes sociais		
Exportar informações sobre o rating para ficheiros externos	X	
Quantifica a carga emocional expressa nos comentários		X
Obter a contagem de ratings positivos e negativos	X	X
Obter contagens de ratings filtrados com diversas métricas		
Obter comentários dos utilizadores		
Responder aos ratings de utilizadores		

Tabela 11 - Resultados do levantamento de requisitos e análise dos endpoints da categoria de Reputation Management

A análise dos resultados permitiu concluir que o número de requisitos abrangidos por ambos os serviços era satisfatório, e por isso, ambos foram considerados para integração no decorrer do segundo semestre.

4.2. ID-Scanning Software

Uma vez que a categoria de *ID-Scanning* não era contemplada nos documentos produzidos por o aluno José Carlos Miranda, a análise desta categoria de serviços iniciou-se com o levantamento de novos serviços.

O processo de levantamento de serviços de *ID-Scanning*, foi semelhante ao utilizado anteriormente. Com efeito, esta etapa começou com o uso das seguintes *keywords*, aplicadas no motor de busca *Google*:

- "*ID Scanner Software SaaS*" – 02 de abril de 2018;
- "*Identification Scanner SaaS*" – 02 de abril de 2018;
- "*Online Identification Scanner*" – 04 de abril de 2018;
- "*ID Scanner SaaS API*" – 04 de abril de 2018;
- "*Identification Software SaaS API*" – 05 de abril de 2018;

À semelhança das categorias anteriormente descritas, aqui também foram sendo feitos sucessivos refinamentos em função dos dados anteriormente conhecidos. Numa fase final do levantamento, fez-se igualmente uso da palavra "API" de forma a ir ao encontro de serviços que disponibilizassem acesso à mesma.

Depois de identificados os serviços, fez-se uma análise com base nos seguintes parâmetros, já utilizados anteriormente:

- API*:* parâmetro que indicou a existência/não-existência de *API* no serviço em análise;
- Protocolo API*:* protocolo utilizado para desenho da *API*, caso esta existisse;
- Formato dos dados: formato dos dados utilizados pela *API*, caso esta existisse;

- Acesso à API*:* tipo de utilizadores era disponibilizado o acesso à *API,* caso esta existisse;
- Documentação da API: parâmetro que indicou a existência/não-existência de documentação da API, caso esta existisse.

No final do processo, os resultados foram os apresentados na Tabela 12.

ID	Nome	API	Protocolo API	Formato dos Dados	Acesso à API	Documentação da API
ID_1	Idcheck	Sim	REST	JSON	Subscrição	Sim
ID_2	Idscan.net	Sim	REST	String	Aberto	Não
ID_3	Zignsec	Sim	REST	JSON	N/D	Sim
ID_4	Mitek	Não	N/A	N/A	N/A	N/A
ID_6	Jumio	Não	N/A	N/A	N/A	N/A
ID_7	icarvision	Não	N/A	N/A	N/A	N/A
ID_9	Idscan.com	Não	N/A	N/A	N/A	N/A
ID_10	Microblink	Sim	N/D	JSON	Aberto	Sim
ID_11	Idscan.com	Não	N/A	N/A	N/A	N/A

N/D – Não Disponível N/A – Não Aplicável

Tabela 12 - Resultados do processo de levantamento e análise de serviços de ID-Scanning atualmente existentes

Através da análise dos resultados, é possível constatar que esta categoria se encontra com um número baixo de serviços, uma vez que apenas 4 permitem acesso à API. A explicação encontrada prende-se, essencialmente, com a sensibilidade dos dados trabalhados, que envolvem uma série de políticas ao nível de partilha dos mesmos, por vezes difíceis de cumprir, limitando assim o número de SaaS desta categoria atualmente disponiveis.

Quando se definiram os parâmetros para analisar as diversas APIs nesta categoria, verificou-se uma grande limitação nos serviços, que se prendia com o número de bases de dados de diferentes países a que estes tinham acesso. Como já referido, os dados trabalhados são bastante sensíveis, sendo que, o acesso e partilha dos mesmos envolve bastantes politicas. Desta forma, um serviço conseguir aceder a bases de dados de um número de países significativo, é algo extremamente exigente.

Verificada esta limitação, foi discutido, com os principais *stakeholders,* um método alternativo de avaliação desta categoria. Uma vez que os 4 serviços a avaliar possuíam APIs semelhantes, ficou definido que a ordem de priorização final seria feita em função do número de países a que cada serviço tinha acesso. Desta forma, os resultados foram os que estão apresentados na Tabela 13.

#	ID	Nome	Nº de Países
1	ID_01	Idcheck	208
2	ID_02	Idscan.net	2
3	ID_03	Zignsec	14
4	ID_10	Microblink	3

Tabela 13 - Resultados da análise do número de serviços abrangidos por cada serviço

Olhando para os resultados finais, é possível concluir que os serviços "*Idcheck*" e "*Zignsec*", são os que possuem um maior número de países abrangidos, e por isso, os mais prioritários para integração.

Capítulo 5 – Arquitetura do Sistema

No presente capítulo será descrita a arquitetura adotada na plataforma. Primeiramente será apresentada uma visão geral, onde é possível visualizar os diversos utilizadores que farão uso da plataforma. Face aos diferentes cenários identificados, foi necessária a definição de diversas componentes que se relacionam entre si, e que serão explicadas de forma detalhada.

5.1. Visão Geral da Arquitetura

Tal como referido anteriormente, quando iniciado o estágio, o projeto já contava com 18 meses desde o seu inicio. Assim sendo, já haviam sido concluídas as tarefas que definiam o desenho inicial da arquitetura do sistema. Desta forma, o presente estágio não contemplou atividades de âmbito arquitetural. No entanto, considerou-se relevante fazer uma descrição das diversas componentes e como estas se relacionam, de forma a contextualizar o leitor para os capítulos seguintes.

No "Documento de arquitetura da plataforma Hotelcracy Apps" [21] , que serviu como base para o desenvolvimento deste capítulo, é possível encontrar as diversas decisões tomadas detalhadamente. Este documento está presente no Anexo 4.

Numa primeira fase, foram criados diversos atores e cenários significativos para o sistema, e, seguidamente, foram levantadas todas as restrições (técnicas, qualidade e sistema). Posteriormente, foram tomadas uma série de decisões que afetaram diretamente o desenho do sistema.

Para apresentação da arquitetura do sistema Hotelcracy Apps foram desenhados diagramas com base na metodologia proposta por Simon Brown, no livro *The Art of Visualising Software Architecture*, e fez-se uso da ferramenta *Structurizr* produzida pelo mesmo [21]. Esta teoria propõe o desenho de arquitetura em diversos níveis de abstracção, sendo que, para este projeto foram escolhidos dois níveis, o de contexto e o de componentes do sistema.

Os diagramas das figuras inseridas no presente capítulo, fazem uso dos objetos gráficos apresentados na Figura 7.

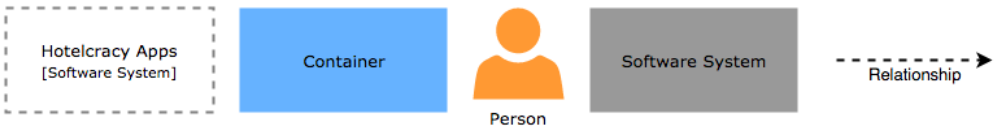

Figura 7 - Objetos gráficos da arquitetura da plataforma

Para o nível de contexto, foi desenhado o diagrama apresentado na Figura 8.

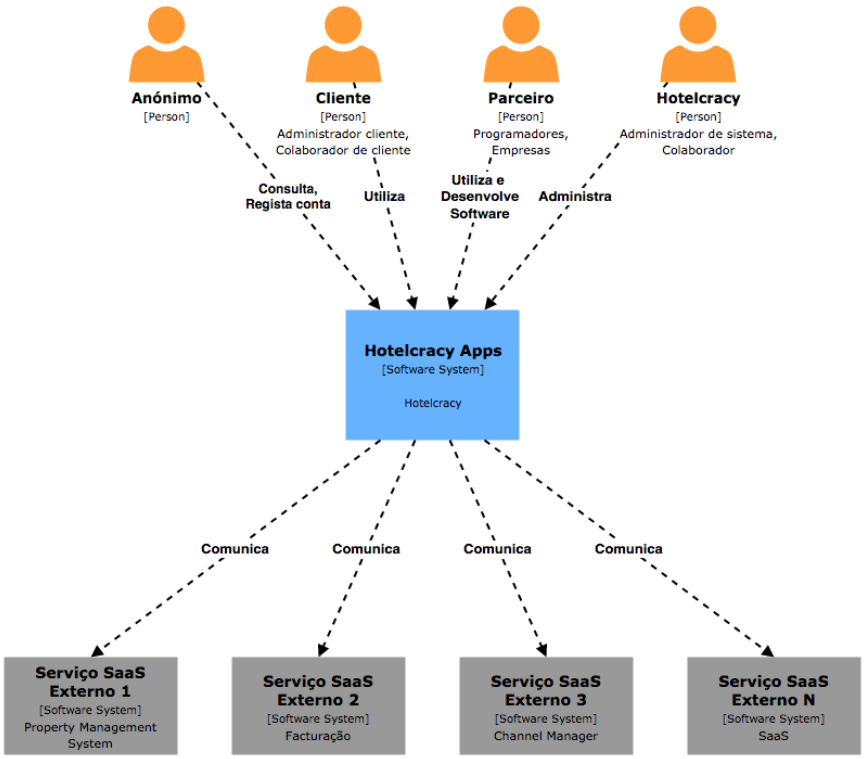

Figura 8 - Diagrama de contexto da plataforma Hotelcracy

Este é um diagrama que permite visualizar o trabalho realizado essencialmente nas primeiras duas etapas de definição de arquitetura. Demonstra assim, os diversos tipos de atores que podem interagir no sistema, bem como, define como estes virão a relacionar-se com o mesmo em função do tipo de privilégios associados.

Num segundo nível de abstração, referente às componentes do sistema, é possível visualizar o trabalho que foi desenvolvido essencialmente na Atividade 1 do projeto. De forma resumida, podemos visualizar assim o *output* deste trabalho através do diagrama representado na Figura 9.

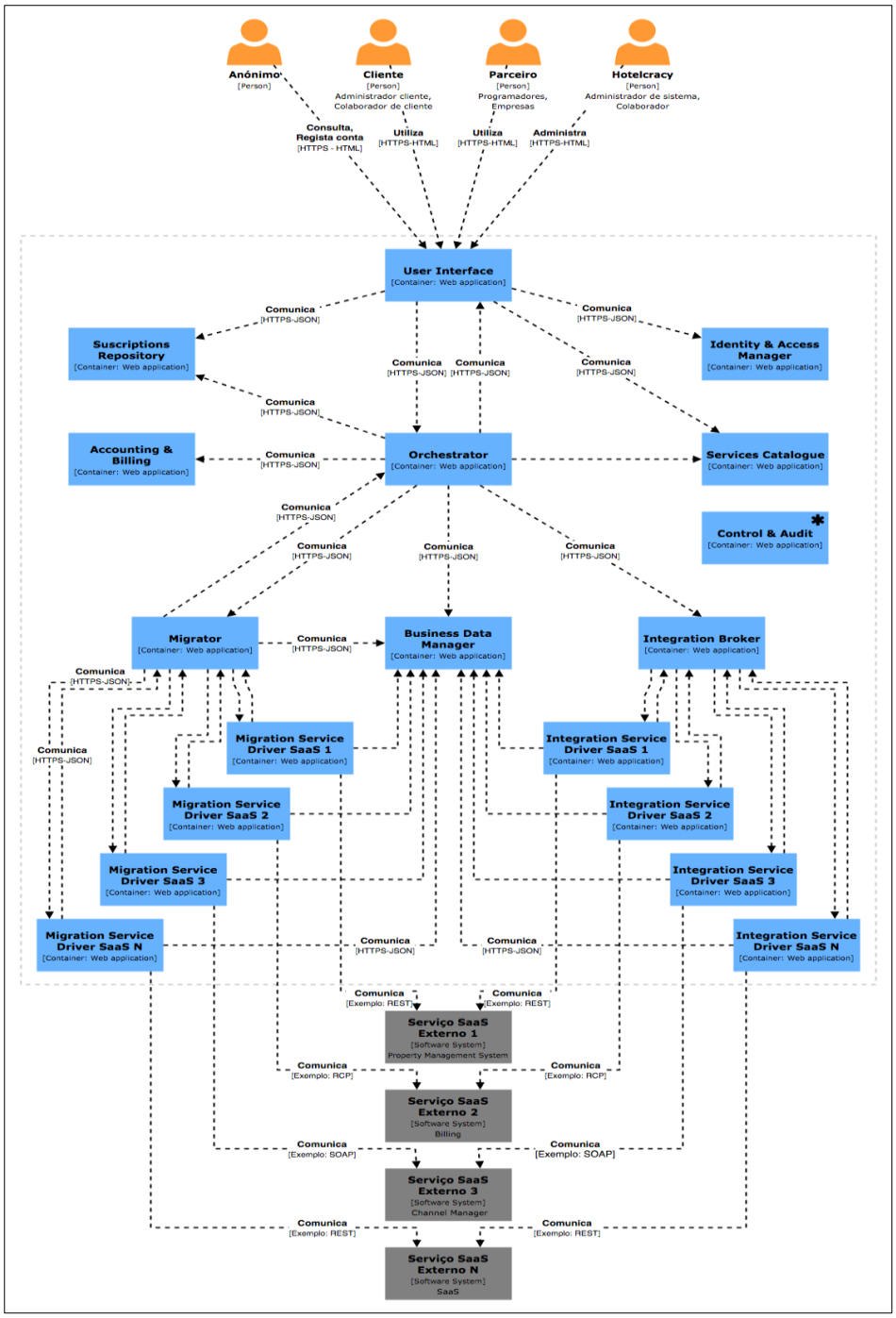

Figura 9 - Diagrama de componentes da plataforma Hotelcracy

Através deste diagrama, podemos visualizar a representação dos principais grupos funcionais do sistema. Numa fase posterior, em função da sua complexidade e importância, estes foram divididos em dois grupos principais de componentes, complexos, e de suporte. No subcapítulo seguinte são explicadas mais detalhadamente as principais funcionalidades de cada um dos componentes identificados.

5.2. Componentes do Sistema

Através de reuniões de equipa, definiu-se que um componente é chamado de complexo, caso este seja responsável por pelo menos um dos processos definidos na Atividade 1 (processos de subscrição, migração, cancelamento e utilização). Para além disto, estes podem também ser responsáveis pela integração de novos serviços na plataforma.

Assim, foram considerados um total de 8 componentes complexos:

- *User Interface*: componente que garante a comunicação com o utilizador. Através de um browser, esta permite aceder às principais funcionalidades do sistema, em função das opções tomadas pelo utilizador;
- *Services Catalogue*: responsável pela implementação funcional do *Marketplace*. Tem como principal função, a disponibilização dos serviços disponiveis para subscrição na plataforma, e que serão apresentados através da *User Interface;*
- *Orchestrator*: coordena a comunicação entre os diversos componentes, nos processos de subscrição, utilização, migração e cancelamento de serviços existentes na plataforma;
- *Integration Service Drivers*: componentes responsáveis por converter o protocolo interno do sistema, no protocolo de comunicação dos serviços SaaS externos;
- *Integration Broker*: responsável pela comunicação entre o *Orchestrator* e as *Integration Service Drivers*. Este funciona como um componente intermédio, à semelhança de um API *Gateway* para as comunicações com os serviços *SaaS* integrados;
- *Migrato*r: componente responsável pela migração de dados entre serviços *SaaS* externos;
- *Migration Service Drivers*: garante a comunicação entre os serviços *SaaS* externos, de forma a que estes consigam migrar os dados entre eles;
- *Business Data Manager*: componente responsável por tratar de pedidos de armazenamento e de leitura de dados de negócio de todo o sistema. Este garante a comunicação com a base de dados, que estará definida segundo o modelo canónico.

Para além destes componentes complexos, existem também os componentes de suporte, responsáveis por processos de segurança e/ou monetização de um sistema comercial:

- *Identity & Access Manager*: componente responsável por controlar o acesso ao sistema autenticado/autorizado e não-repudiável. Este controlo será assegurado através da implementação de um sistema de autenticação e permissões;
- *Subscriptions Repository*: responsável por armazenar as informações relativas às subscrições dos utilizadores;
- *Accounting & Billing*: contabiliza a utilização de serviços SaaS do utilizador, e cria a faturação relativa à relação Cliente-Parceiro;
- *Control & Audit*: componente responsável por armazenar *logs* das operações dos utilizadores, e dos estados das componentes do sistema.

Capítulo 6 – Modelo Canónico de Dados

No presente capítulo é feita uma descrição do Modelo Canónico de dados desenhado para a plataforma. É apresentado o processo de elaboração, que se finalizou com a criação de 3 estruturas de suporte ao mesmo. Estas têm um papel preponderante na plataforma *Hotelcracy Apps,* uma vez que serão responsáveis pela transferência dos dados técnicos e de negócio, entre as diversas componentes.

6.1. Descrição do Modelo Canónico

O Modelo Canónico surgiu da decisão de utilizar um protocolo de comunicação interno entre as componentes da plataforma, proporcionando também um meio de resposta à heterogeneidade existente entre as APIs dos serviços integrados. Entende-se assim, que o modelo descrito será o formato intermédio para o qual os dados de negócio dos hóteis serão convertidos. A conversão dos dados recebidos, para o formato Canónico, possibilitará o seu tratamento, independentemente dos serviços subscritos pelo hoteleiro. Será também possível a partilha de dados entre serviços, dentro do mesmo âmbito (por exemplo hóspedes).

A definição do modelo de dados tem como base as estruturas de dados especificadas nas APIs dos diversos *SaaS* integrados na plataforma. Desta forma, é necessário conhecer as estruturas a respeitar para invocação dos *endpoints,* bem como as que são dadas como resposta. Depois de efetuado o mapeamento das APIs, criou-se os objetos que compõem o modelo canónico, definidos como entidades. Estas descrevem-se como um conjunto de atributos, que servem como suporte dos dados que a plataforma vai tratar. Exemplos de entidades são: clientes, reservas, hotéis, quartos, faturas, entre outros. Sendo que, por exemplo, a entidade cliente, será composta por parâmetros, tais como: nome, número de telemóvel, morada, número de contribuinte, entre outros.

Quando iniciado o presente estágio, o modelo de dados já se encontrava em desenvolvimento. Este já contemplava as diversas entidades necessárias para as categorias de Faturação, bem como, de *Channel Manager*. Desta forma, indo ao encontro dos objetivos delineados, o estagiário contribuiu para a evolução do modelo existente, de forma a que este contemplasse as entidades necessárias para a categoria de *Property Management Software, Reputation Management* e *ID-Scanning*.

Apesar do Modelo Canónico ser único, este divide-se em três vertentes: diagrama Entidade-Relacionamento, Conversores de Dados, e *schemas* de dados para invocação de funcionalidades, que serão explicados nos subcapítulos seguintes.

6.1.1. Diagrama Entidade-Relacionamento

Um diagrama Entidade-Relacionamento é um modelo de dados que ilustra como as entidades (por exemplo, pessoas, objetos, ou conceitos), se relacionam entre si dentro de um sistema [22]. Com efeito, para se iniciar o desenho do diagrama, foi primeiramente necessário identificar as entidades que se iriam relacionar, bem como os parâmetros que iriam compor as mesmas. Para tal, fez-se uso de tabelas de mapeamento, como a ilustrada na Tabela 14.

	InvoiceXpress	Invoice Ocean	Modelo Canónico
Identificação	code		organizational_id
		company	is_company
	name	name	name
		first_name	
		last_name	
		shortcut	
	fiscal_id	tax_no	fiscal_id
Contacto	email	email	email
	website	www	website
	phone	phone	phone
		mobile_phone	mobile_phone
	fax	fax	fax
		person	contact_person
Localização	address	street	street
		street_no	
	postal_code	post_code	postal_code
	city	city	city
	country	country	country

Tabela 14 - Tabela de mapeamento dos parâmetros a inserir na entidade 'Customer' [23]

No exemplo referido, temos o mapeamento de atributos de dois serviços de faturação, 'InvoiceXpress', e 'Invoice Ocean'. Para ambos, está representada a entidade 'Customer', sendo possível visualizar a heterogeneidade existente entre serviços para a mesma referência. Na primeira coluna, é possível visualizar as categorias de parâmetros que eram esperados serem retornados. Seguidamente, nas colunas respeitantes a cada um dos serviços, estão adicionados os parâmetros presentes nas APIs, agrupados pelas categorias definidas na primeira coluna. Na última coluna, estão representados os nomes dos parâmetros a inserir no modelo canónico, em função dos mapeados nas colunas anteriores. Entre serviços, sempre que os parâmetros se encontravam na mesma linha, assumia-se que estes representavam o mesmo. No caso em que os serviços não possuíam os mesmo parâmetros, era feita uma avaliação da importância de inserção desse parâmetro no modelo canónico. Esta ponderação levava a discussões internas de equipa, que por vezes, chegavam até *stakeholders,* afim destes decidirem a inserção/não-inserção no modelo final.

Depois de decididas as entidades a inserir, bem como os parâmetros que compunham as mesmas, estas eram traduzidas para o formato de diagrama, de forma a serem inseridas no modelo. Esta inserção contemplava também o estabelecimento de relações entre as entidades a inserir, com as que já haviam sido inseridas, no caso de se verificar ligações entre ambas. Um excerto do diagrama ER representativo do modelo canónico, com a entidade 'Customer' representada, é possível visualizar na Figura 10.

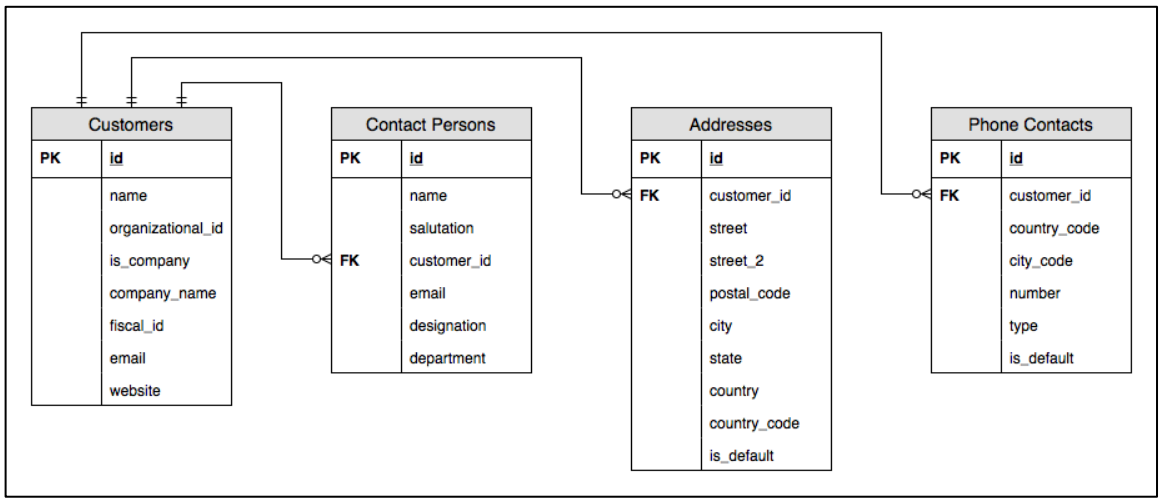

Figura 10 - Excerto do diagrama ER, representativo do Modelo Canónico [23]

Através da figura acima representada, é possível observar como as entidades extraídas das diversas APIs ficam representadas no modelo canónico. As categorias identificadas na primeira coluna da Tabela 14, deram origem a novas tabelas de forma a poderem ser partilhadas por outras entidades também presentes no modelo. Os atributos inseridos em cada uma das tabelas, foram os descritos na última coluna da tabela de mapeamento. De referir, que o excerto do diagrama aqui representado, já contemplava parâmetros identificados no mapeamento de outros serviços.

6.1.2. *Schemas* de dados

Os *schemas* definem-se como um esquema/representação de um formato de dados, seguindo um conjunto de regras aplicadas à estrutura e organização dos mesmos [24]. Desta forma, é possível fazer uma validação das mensagens que fazem uso destas mesmas estruturas, previamente ao tratamento das mesmas. Os *schemas* podem ser criados com recurso a ferramentas como XML *Schema* ou JSON *Schema*, para utilização de mensagens XML ou JSON, sendo que na plataforma as mensagens fazem uso deste último [25].

Para se entender a razão de utilização destas ferramentas, devem-se considerar os seguintes aspetos:

- As funcionalidades disponibilizadas na plataforma, traduzem-se em invocações nos *endpoints* dos diversos serviços adotados;
- Os dados de preenchimento obrigatório nas estruturas de dados utilizadas para invocação dos *endpoints*, variam em função do serviço em utilização.

Devido às circunstâncias enumeradas foi então decidida a utilização de *schemas,* para representar as estruturas de dados a cumprir, em função da funcionalidade e do serviço em utilização. Desta forma, para cada funcionalidade existente em cada um dos serviços disponiveis na plataforma, existe um *schema* que possui a estrutura de dados a cumprir. Os *schemas* têm sempre que respeitar o modelo canónico da plataforma, sendo que, entre serviços, os dados de carácter obrigatório podem variar. No Quadro 1 é possível visualizar um excerto do JSON *schema* referente à criação de um *'Customer'*.

```
{
  "title": "Create customer @ InvoiceXpress",
  "type": "object",
  "properties": {
    "addresses": {
      "items": {
        "$ref": "#/definitions/address"
      },
      "type": "array",
      "minItems": 0,
      "maxItems": 1
    },
    "company_name": {
      "type": "string"
    },
    "contact_persons": {
      "items": {
        "$ref": "#/definitions/contact_person"
      },
      "type": "array",
      "minItems": 0,
      "maxItems": 1
    },
    "email": {
      "format": "email",
      "type": "string"
    },
    "fiscal_id": {
      "type": "string"
    },
    "is_company": {
      "type": "boolean"
    },
    "name": {
      "type": "string"
    },
    "organizational_id": {
      "type": "string"
    },
    "phone_contacts": {
      "items": {
        "$ref": "#/definitions/phone_contact"
      },
      "type": "array",
      "minItems": 0,
      "maxItems": 1
    },
    "website": {
      "format": "uri",
      "type": "string"
    }
  },
  "required": [
    "name",
    "organizational_id"
  ]}
```

Quadro 1 - Excerto do schema de criação de um 'Customer' para o serviço InvoiceXpress [8]

No excerto acima, é possível visualizar uma representação real do *schema* de criação de um *'Customer'* para o serviço InvoiceXpress. É possível constatar os diversos parâmetros disponiveis no modelo canónico, sendo que, no final do excerto, no campo *'required',* encontram-se aqueles que são de preenchimento obrigatório. Os parâmetros que possuem a simbologia *'ref',* indicam que estão a utilizar a mesma estrutura presente numa outra entidade já definida, como por exemplo *'phone_contact'*.

6.1.3. Conversores de dados

Os conversores de dados, são parte integrante do código desenvolvido na plataforma, sendo que estes se inserem na componente dos *Service Drivers*. São responsáveis por realizar a tradução dos pedidos a efetuar aos serviços externos para o formato exigido pelas *APIs* dos mesmos. Depois de efetuados os pedidos, as respostas retornadas são também traduzidas nos *Service Drivers* para o modelo canónico interno à plataforma. No Quadro 2 está presente um excerto do conversor da entidade *'Customer'*.

```
class CustomerConverter
  def self.from_service(client)
    {
      external_id: client[:id],
      name: client[:name],
      organizational_id: client[:code],
      fiscal_id: client[:fiscal_id],
      is_company: false,
      company_name: nil,
      email: client[:email],
      website: client[:website]
    }
  end

  def self.from_hotelcracy(customer, address: nil, contact_person: nil,
phone_contacts: {})
    c = {
      name: customer[:name],
      code: customer[:organizational_id],
      email: customer[:email],
      fiscal_id: customer[:fiscal_id],
      website: customer[:website]
    }

end
```

Quadro 2 - Excerto do conversor de dados para a entidade 'Customer' relativa ao serviço [23]

Olhando para o excerto, é possível concluir, que a função do conversor é efetuar a conversão direta dos dados para os formatos exigidos. No exemplo acima representado, é possível visualizar dois métodos, responsáveis pela conversão dos dados para o formato exigido no serviço (*from_hotelcracy*), e para o modelo canónico adotado na plataforma (*from_service*).

6.2. Evolução do Modelo de Dados

A evolução do modelo de dados, tinha como principal objetivo, a adição das diversas entidades presentes nas APIs dos serviços a integrar, ao modelo canónico existente. Para a categoria de *ID-Scanning*, verificou-se logo à partida que todas as entidades necessárias para a integração destes serviços, já estariam presentes no modelo atual, e por isso não foi alvo de trabalho nesta fase.

A primeira tarefa consistiu no mapeamento das APIs dos serviços, de forma a identificar as principais entidades presentes, e como estas se relacionavam. Resultante desta tarefa, as entidades obtidas foram as apresentadas na Figura 11 para os serviços da categoria de *PMS,* e na Figura 12 para a categoria de *Reputation Management*. De forma a auxiliar o leitor na interpretação dos diagramas, na Tabela 15 e Tabela 16 , é possível ler uma descrição das entidades presentes nos diagramas.

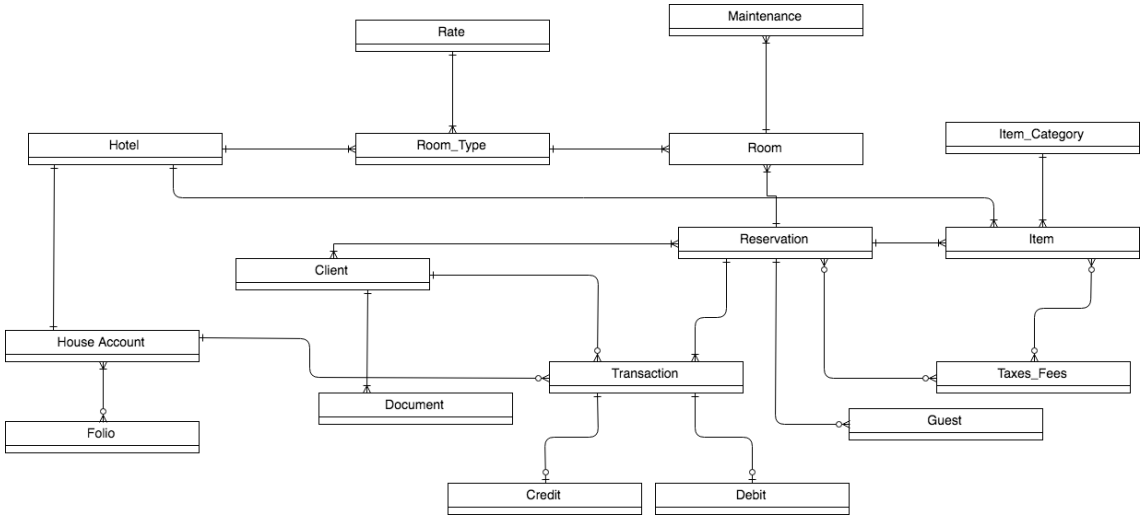

Figura 11 - Diagrama Entidade-Relacionamento dos serviços PMS

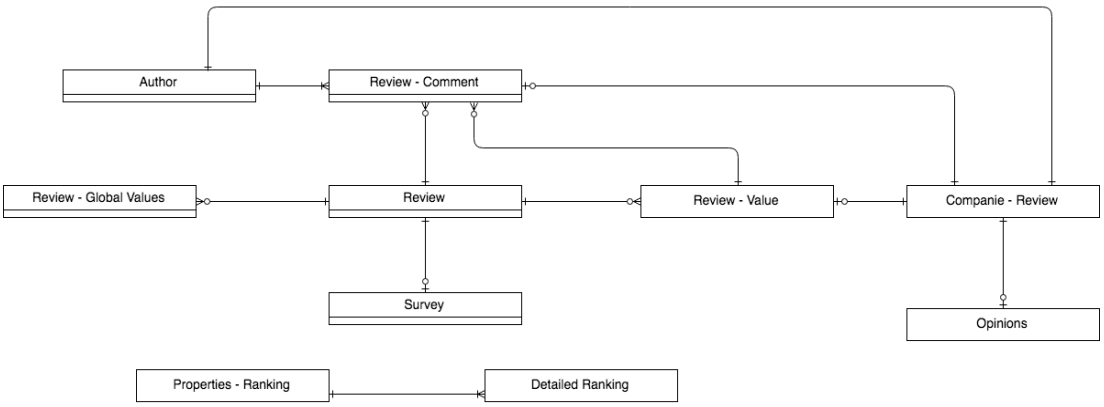

Figura 12 - Diagrama Entidade-Relacionamento dos serviços de Reputation Management

Entidade	Descrição
Document	Entidade relativa aos documentos associados ao cliente
Hotel	Entidade descritiva do Hotel
House Account	Entidade relativa à conta que armazena as transações e movimentos financeiros no hotel
Folio	Entidade que permite o armazenamento de despesas que o cliente faz no hotel
Client	Entidade relativa ao cliente do hotel
Room_Type	Entidade relativa ao tipo de quartos disponiveis
Rate	Entidade relativa ao preço atribuído a um tipo de quartos
Payment	Entidade relativa a um pagamento realizado
Maintenance	Entidade relativa a uma manutenção realizada
Room	Entidade descritiva do quarto
Reservation	Entidade relativa a uma reserva feita
Guest	Entidade relativa a um convidado associado à reserva
Item_Category	Entidade relativa a uma categoria de *items* disponíveis
Item	Entidade relativa a um item disponível
Taxes_Fees	Entidade relativa às taxas a aplicar

Tabela 15 - Tabela de entidades dos serviços de PMS

Entidade	Descrição
Review	Entidade relativa a uma *review* realizada
Review – Global Values	Entidade de que armazena a contagem de *reviews* qualitativas: positivas, negativas e neutras
Review – Comment	Entidade relativa a um comentário realizado
Survey	Entidade relativa a um questionário realizado
Review – Value	Entidade relativa a uma cotação atribuída de forma quantitativa
Companie – Review	Entidade relativa *a review* realizada por uma empresa
Opinions	Entidade relativa a uma opinião dos elementos da empresa
Author	Entidade relativa ao autor da *review* da empresa
Properties - Ranking	Entidade relativa ao ranking de hotéis
Detailed Ranking	Entidade relativa ao ranking detalhado de cada dia

Tabela 16 - Tabela de entidades relativos aos serviços de Reputation Management

Depois de identificadas as entidades, passou-se à fase seguinte, que consistia na identificação dos atributos a inserir nas mesmas. Desta forma, fez-se um mapeamento de todos os parâmetros devolvidos e exigidos pelas diferentes *APIs,* com recurso a tabelas semelhantes às descritas no capítulo 6.1.1. Primeiramente, foram identificados aqueles que eram de carácter obrigatório, uma vez que eram essenciais para conseguir realizar os pedidos. Seguidamente, foram identificados parâmetros que apesar de não serem obrigatórios, possuíam informações relevantes. Por fim, em discussão com alguns dos *stakeholders,* foram também identificados alguns campos que seriam importantes para o modelo de negócio. As tabelas resultantes do mapeamento constam do Anexo 1 para a categoria PMS e Anexo 2 para os serviços de Reputation Management. Depois de decididos os parâmetros a inserir, para concluir o processo foi necessário verificar se as entidades que se pretendiam inserir, já estavam criadas no modelo da plataforma. Nos casos em que estas já existiam, foram adicionados os novos parâmetros. Para os casos em que foi necessário adicionar novas entidades, foi necessário estabelecer as relações com as já existentes. O diagrama ER resultante de todo este processo, é possível visualizar na Figura 13.

Com o diagrama final já definido, de forma a ser feita uma validação, este foi discutido em reuniões internas do *IPNlis.* Aqui estavam presentes os dois responsáveis pelo desenho do modelo canónico utilizado para as categorias já integradas, que tiveram como função avaliar e sugerir possíveis alterações a realizar às propostas apresentadas. Depois de aplicadas as correções sugeridas, o modelo desenhado foi apresentado aos principais *stakeholders* do projeto, e integrado no modelo canónico de toda a plataforma. As restantes estruturas de dados, *schemas* e conversores de dados, foram desenvolvidas no decorrer do segundo semestre, e por isso, serão explicadas num capítulo seguinte.

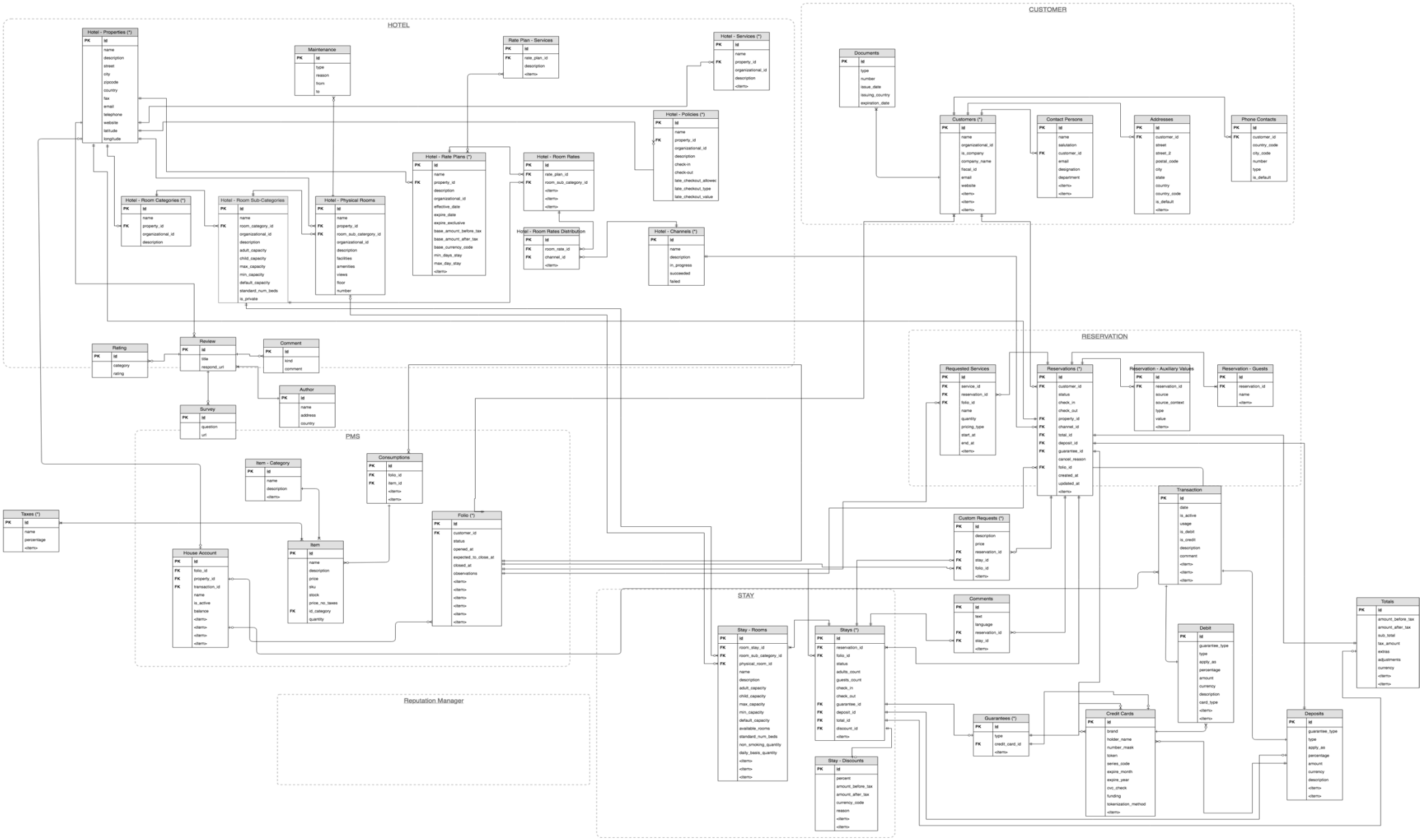

Figura 13 - Diagrama Entidade-Relacionamento resultante do mapeamento da API dos serviços integrados

Capítulo 7 – Desenvolvimento da Plataforma

De forma a evoluir as componentes da plataforma, para cumprir os objetivos propostos para o presente do estágio, foi necessário o entendimento de um conjunto de técnicas e hábitos já adotados na equipa de desenvolvimento. Neste capítulo serão descritos os processos utilizados, bem como será apresentado mais detalhadamente o trabalho desenvolvido nas diversas componentes responsáveis pelo processo de integração de serviços.

7.1. Processos e Técnicas aplicadas

Para o código produzido, foi aplicado um processo de gestão do ciclo de vida apresentado pelo parceiro líder do consórcio *Hotelcracy Apps,* bem como foi utilizada a técnica de desenvolvimento *Test-Driven-Development* [26]. Estes aspetos serão descritos nos subcapítulos seguintes.

7.1.1. Gestão do Ciclo de Vida do código produzido

O desenvolvimento de uma plataforma como a *Hotelcracy* implica que a equipa de desenvolvimento seja composta por vários elementos, de modo a que sejam cumpridos os prazos delineados. Desta forma, para que todo o processo de desenvolvimento siga um fluxo saudável, é necessária uma grande coordenação dentro do grupo. Esta pode tornar-se uma tarefa difícil de alcançar, uma vez que as mesmas linhas de código podem ter que ser acedidas, e por vezes modificadas, por diferentes elementos da equipa. Esta concorrência pode assim originar situações de conflito, que podem atrasar em muito o desenvolvimento do produto final. Assim, para evitar estas potenciais ocorrências, é imperativa a definição de um processo de desenvolvimento que permita uma boa gestão do código produzido entre todos os elementos da equipa.

Uma vez que o desenvolvimento da plataforma era assegurado por duas equipas compostas por vários elementos, foi necessária a existência de um meio comum, que permitisse a gestão de todo o código produzido. Desta forma, fez-se uso de um repositório no *Github* [27], que é uma plataforma de hospedagem de código-fonte com controle de versões através da ferramenta *Git* [28]. Através do *Github*, é possível a aplicação do conceito de *branches* (ramos), onde existe um principal denominado *master*, e outros secundários. Estes herdam do *master,* no entanto, num ambiente diferente. Através desta separação, os diversos intervenientes do desenvolvimento podem desenvolver as suas *features* ou ideias, sem interferir com o trabalho dos outros elementos, através da criação dos seus próprios *branches*. No final, quando está concluído o desenvolvimento, o *developer* cria um *pull request* de forma a enviar o código produzido no seu *branch*, para o principal. Antes de ser transferido para o *master*, todo o trabalho tem que ser revisto e aceite, por pelo menos 2 outros elementos da equipa de desenvolvimento. Durante a revisão, poderiam ser adicionados comentários, ou sugerir alterações a efetuar ao trabalho realizado. Este mecanismo permitia assim que pequenos erros fossem logo detetados numa fase prematura do código produzido, não perpetuando o erro, e tendo assim como consequência uma melhor qualidade do produto final.

7.1.2. Test-Driven-Development

Uma das formas de validar a qualidade do código produzido, é através da aplicação de testes ao mesmo. Estes permitem a definição de um conjunto de regras a validar, que perspetivam uma resposta esperada para os casos em que a execução segue o fluxo desejado. Quando um teste não finaliza com os resultados esperados, podemos estar perante um erro, normalmente denominado de *bug*, que deve ser posteriormente corrigido. A construção e aplicação de testes, possibilita também que seja entendido o impacto de alterações no código anteriormente produzido. Esta medida é possível através da preservação dos diversos testes produzidos em etapas de desenvolvimento anteriores. Desta forma, sempre que são realizadas alterações, todos os testes anteriormente desenvolvidos são executados de forma a validar que as alterações feitas não tiveram impacto negativo em outras partes do código.

Normalmente a criação dos testes é posterior à fase de desenvolvimento, como por exemplo no *Waterfall Model* descrito na Figura 1. Nestes casos, os testes podem ser de vários níveis, como por exemplo, ao sistema ou unitários, e de vários tipos, como por exemplo de aceitação ou regressão [29]. No entanto, na técnica TDD, os testes são escritos previamente ao inicio do desenvolvimento de código para implementar uma nova funcionalidade [26]. Nestes é possível definir uma série de resultados que são esperados, mediante a funcionalidade a implementar, e os *inputs* a receber. Assim, quando concluído o desenvolvimento do código, este pode ser logo submetido a uma bateria de testes, de forma a verificar se está a produzir o fluxo esperado. Na técnica TDD os testes são de baixo nível, como por exemplo unitários ou de integração [30]. Neste caso, os testes de alto nível, ficam para fases posteriores, uma vez que estes já abrangem o *software* num todo.

Kent Beck foi quem popularizou o uso do TDD, e quem enunciou um conjunto de regras a serem seguidas, de forma a que se possam colher vantagens do uso da metodologia. As regras enunciadas, são as seguintes:

- Os testes anteriormente implementados, devem também fornecer *feedback* para novo código produzido;
- O *developer* deve ser responsável pelo desenvolvimento dos seus próprios testes, evitando assim o tempo de espera necessário para que outra pessoa realize o trabalho;
- O ambiente de desenvolvimento deve proporcionar uma resposta rápida para pequenas alterações (por exemplo, um compilador rápido), uma vez que os mesmos testes podem ter que ser executados diversas vezes;
- A arquitetura da plataforma deve possuir um baixo acoplamento entre as diversas componentes, de forma a se conseguir alcançar um maior detalhe nos testes [26].

Aplicando este conjunto de regras, podem ser colhidos uma série de benefícios, sendo que a rapidez e isolamento em todo o processo, são umas das grandes vantagens. Para além disto, o *developer* pode trabalhar com um nível de confiança mais elevado, uma vez que os testes anteriormente produzidos, validam sempre o código em desenvolvimento.

A filosofia do TDD é baseada na ideia que a nossa mente não é capaz de perseguir simultaneamente os dois principais objetivos do desenvolvimento de *software,* desenvolver um produto com o comportamento desejado, e que ao mesmo tempo já esteja otimizado [31]. Desta forma, a metodologia TDD, faz uso do ciclo "*red, green, refactor*", demonstrado na Figura 14. Este tem como objetivo a construção de um conjunto de testes, escrita do código de implementação da funcionalidade a desenvolver, e posteriormente otimização do mesmo em pequenos ciclos de desenvolvimento. A fase *red* é sempre o ponto de partida, e tem como objetivo principal o desenvolvimento dos testes a aplicar ao código a desenvolver. Desta forma, estes são criados com base nas expetativas que se pretendem alcançar com o desenvolvimento. A fase *green* é onde é desenvolvido o código que cumpre os objetivos

desenhados nos testes anteriormente desenvolvidos. Com efeito, o foco desta etapa está apenas em encontrar uma solução, sem quaisquer preocupações ao nível de otimização. Por fim, a fase *refactor*, é onde se pensa nas otimizações a fazer. As sucessivas ideias para otimizar o código desenvolvido são assim aplicadas, garantindo sempre que as expectativas dos testes criados continuam sempre a ser alcançadas.

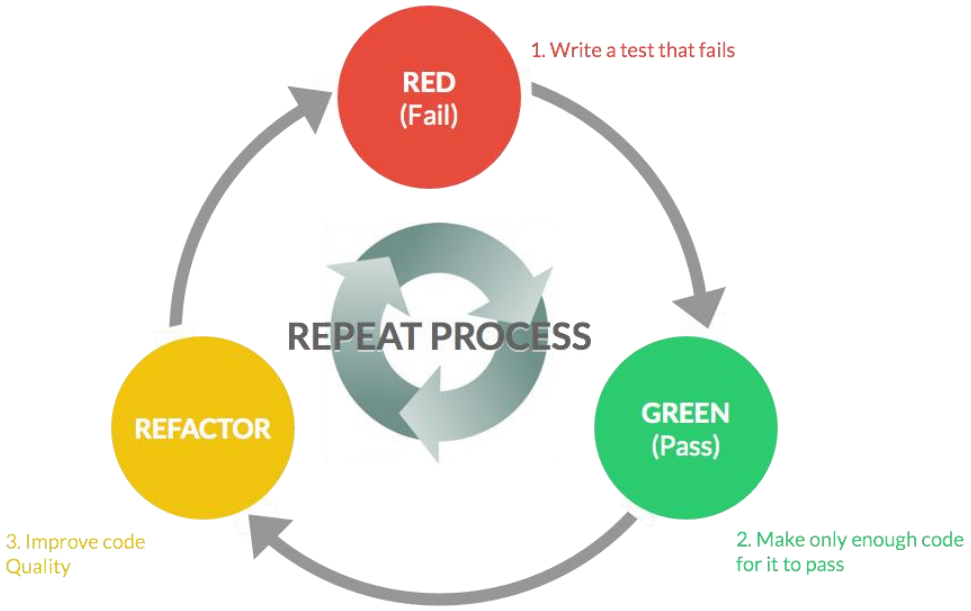

Figura 14 - Ilustração da framework TDD "red, green, refactor" [32]

7.2. Ferramentas e Tecnologias utilizadas

O desenvolvimento necessário para a conclusão dos objetivos do presente estágio era focado essencialmente no *backend* da plataforma *Hotelcracy*. Quando iniciado o estágio, tal como já referido, o desenvolvimento já se havia iniciado, fazendo com que as ferramentas e tecnologias a utilizar já se encontrassem definidas. Desta forma, teve que existir um processo de aprendizagem da *framework* "*Ruby on Rails*", visto que o estagiário nunca havia desenvolvido com recurso à mesma.

O processo de preparação iniciou-se assim com a aprendizagem da linguagem *Ruby*, com a qual o estagiário também nunca havia contactado. Desta forma, para a aquisição de conhecimento, realizaram-se os cursos *online* "*Ruby*" e "*Ruby for Beginners*", dos *websites* *"codeacademy.com"* [33] e *"ruby-for-begginers.rubymonstas"* [34]. Ambos são cursos onde se abordam questões de sintaxe, estruturas de dados, e questões de desenvolvimento orientado a objetos, em muito semelhantes a outras linguagens. Para além dos cursos referidos, a visualização de vídeos na plataforma *Youtube* [35], também permitiu adquirir conhecimento.

Finalizada a aprendizagem da linguagem *Ruby*, passou-se para a *framework Ruby on Rails,* que é vocacionada para o desenvolvimento de *web applications*, de acordo com o modelo *Model-View-Controller* [36]. Para tal, fez-se novamente uso do *website* *"codeacademy.com"*[33], que tinha disponível um curso de aprendizagem da *framework*. Para adquirir conhecimento, fez-se também uso do livro "*The Ruby on Rails Tutorial",* de Michael Hartl [37]. Este aborda vários conceitos relacionados com a *framework*, e ao mesmo tempo vai induzindo o leitor a realizar diversas experiências numa aplicação *web*, inicialmente criada para o efeito.

Para a criação de testes, foi utilizada a ferramenta de testes *RSpec* [38], que é vocacionada para a execução de testes da linguagem *Ruby*. Esta foi a escolha para o desenvolvimento do projeto, uma vez que os conceitos básicos da mesma assentam na metodologia TDD, já descrita no subcapítulo anterior. Para adquirir conhecimento acerca da ferramenta, fez-se uso do livro "*Everyday Rails: Testing with RSpec",* de Aaron Summer [39]. Neste é descrita a sintaxe a utilizar, bem como são exemplificados diversos tipos de testes passíveis de realizar com recurso à ferramenta. Incrementalmente eram também propostos alguns exercícios a resolver, de forma a validar os conhecimentos adquiridos no final de cada capítulo. Através do Quadro 3 é possível visualizar um exemplo de teste, utilizando a ferramenta *RSpec*.

```
RSpec.describe Service, type: :model do
   it "is not valid with empty name" do
      expect(build(:service, name: nil)).not_to be_valid
   end

   it "is not valid without a category" do
   expect(build(:service, category: nil)).not_to be_valid
  end
end
```

Quadro 3 - Exemplo de teste utilizando a ferramenta RSpec

O teste acima demonstrado, faz duas simples validações, relativos a uma instância "*Service*". Como descrito na legenda dos testes, qualquer serviço não será válido, caso não possua nome e categoria.

Foi também necessário aprofundar o conhecimento acerca da ferramenta *Git*, uma vez que apesar de já ter utilizado a mesma, o estagiário apresentava muito pouca experiência. De forma a melhorar o *knowledge* foram visualizados alguns vídeos na plataforma *Youtube* [35], bem como se efetuou a leitura de documentação relativa aos principais comandos desta ferramenta.

7.3. Componentes desenvolvidas

No presente subcapítulo será feita uma explicação detalhada dos componentes abordados durante a etapa de desenvolvimento. Estes são classificados como componentes complexos, uma vez que fazem parte do processo de integração de serviços, definido como um dos principais fluxos de funcionamento da plataforma. Os componentes abordados no desenvolvimento, foram os seguintes: *Services Catalogue, Orchestrator, Integration Broker* e *Integration Service Drivers*.

A descrição a apresentar é um resumo dos documentos técnicos de especificação de componentes, desenvolvidos no âmbito do projeto. Nestes estão presentes informações relativas a conceitos específicos das componentes, requisitos funcionais e tecnológicos, arquitetura e interfaces a implementar.

7.3.1. Services Catalogue

O principal objetivo da componente *Services Catalogue*, é o de armazenar os diversos dados relativos aos serviços SaaS integrados na plataforma. Os dados armazenados são de dois âmbitos: comercial, cujo objetivo é expor as informações necessárias para que os utilizadores tomem as suas decisões, e técnico, que correspondem a dados que outros componentes operantes no sistema, necessitam de conhecer acerca dos serviços integrados.

Relativo a dados comerciais, deverão estar presentes informações relevantes para caracterizar o serviço. Estas serão compostas pelo nome, descrição, custos associados, imagens e funcionalidades acessíveis através da plataforma *Hotelcracy*. As integrações passíveis de se realizar com outros serviços, avaliações submetidas por outros utilizadores, bem como a categoria a que pertence o serviço também serão dados de foro comercial. No âmbito técnico, devem ser disponibilizadas as estruturas de dados para se proceder a uma subscrição, integração, utilização, migração ou cancelamento de um serviço. Deve existir também uma lista das entidades utilizadas, bem como das funcionalidades passíveis de se realizar. Para cada funcionalidade, deverá existir também um *schema* de dados a cumprir, de forma a efetuar a invocação corretamente.

A componente *User Interface* é das mais ativas no acesso à componente *Services Catalogue*, uma vez que esta necessita de todas informações acerca dos serviços, para posteriormente gerar as diversas interfaces. Esta construção visual baseia-se essencialmente no desenho do *Marketplace*, onde todos os serviços disponíveis na plataforma estarão presentes, bem como nas diversas interfaces de gestão e utilização dos serviços.

No âmbito do estágio, o desenvolvimento desta componente incidiu essencialmente na inserção dos dados de âmbito técnico e negócio. Para cada novo serviço integrado, era necessário criar uma nova instância. O estagiário era responsável por fazer a busca dos dados de negócio, que consistiam essencialmente em pequenas descrições dos serviços, imagens, *website*, e um logo do mesmo. De realçar que estes dados seriam posteriormente exibidos no *Marketplace*, tal como já referido anteriormente. A nível técnico, eram especificadas as funcionalidades passíveis de se realizar, bem como os *schemas* para invocar as mesmas. Era também definido o conjunto de chaves necessárias para se efetuar uma subscrição do serviço.

Como exemplo, tomemos a integração do serviço Beds24. Primeiramente o estagiário foi responsável pela pesquisa dos diversos dados de negócio da solução a adicionar, onde se incluía uma pequena descrição, um logo para identificar o serviço no *Marketplace* e o *url* do *website* do serviço. Depois de adicionadas estas informações, era necessário identificar quais as chaves necessárias para efetuar uma subscrição do serviço. Depois de concluída esta pesquisa, os dados de âmbito técnico iam sendo incluídos paralelamente ao desenvolvimento. Cada vez que era adicionada uma nova funcionalidade passível de se efetuar, primeiramente era necessário identificar qual o *schema* de dados a respeitar, e disponibilizar o mesmo no *Services Catalogue*. Desta forma, todas as ações passíveis de se realizar no serviço, teriam sempre identificadas qual a estrutura de dados a respeitar.

Para consultar mais detalhes acerca desta componente, o leitor poderá consultar o Anexo 7.

7.3.2. Orchestrator

A componente *Orchestrator* tem como principal objetivo gerir a execução de processos complexos na plataforma, bem como orquestrar os mesmos. Estes processos designam-se de complexos, uma vez que implicam a comunicação e intervenção de diversos componentes da plataforma, sendo assim essencial a existência de um agente coordenador. Desta forma, o *Orchestrator* surge como o intermediário de comunicação entre os diversos componentes da plataforma.

Como exemplo de orquestrações realizadas por esta componente pode referir-se o processo de subscrição de um novo serviço. Através da *User Interface*, o utilizador seleciona o botão de subscrição de um determinado serviço, e seguidamente insere as chaves necessárias para realizar a subscrição. Este pedido é remetido para o *Orchestrator*, que primeiramente vai fazer uma chamada ao *Services Catalogue* afim de verificar se o serviço existe, e se as chaves enviadas

são as pretendidas. Em caso de resposta afirmativa, o *Orchestrator* vai enviar um pedido para o *Integration Broker*, que fará um pedido ao serviço externo afim de verificar se as chaves inseridas são válidas. Caso as chaves sejam válidas, o *Orchestrator* vai finalmente adicionar a nova subscrição no *Subscriptions Repository*, e o processo termina. Caso algum erro aconteça durante o processo, o *Orchestrator* será sempre o responsável por remeter esse erro para a *User Interface*.

Quando iniciado o presente estágio, os fluxos entre o *Orchestrator* e as restantes componentes já estavam estabelecidos. Desta forma, o desenvolvimento desta componente incidiu essencialmente em indicar as novas funcionalidades relativas aos novos serviços integrados. Com efeito, as comunicações estabelecidas foram entre as componentes *Orchestrator* e *Integration Broker*.

Como exemplo, consideremos uma funcionalidade possível de se realizar através do novo PMS adicionado no estágio: criar uma reserva. Quando o *Orchestrator* recebe o pedido proveniente da *User Interface*, primeiramente verifica se o utilizador autenticado possui uma subscrição para o serviço a utilizar, através de um pedido ao *Subscriptions Repository*. Em caso afirmativo, as chaves de autenticação no serviço são devolvidas pela componente, e prepara-se o pedido para reencaminhar ao *Integration Broker*. No entanto, o *Orchestrator* necessita de saber o *endpoint* e invocar na componente de destino. Desta forma, o trabalho desenvolvido refletiu-se nesta parte do processo, uma vez que para cada funcionalidade adicionada, era necessário definir a rota a seguir até ao método presente no *Integration Broker,* que posteriormente seria o responsável pela invocação do método no serviço externo. O processo de envio de um pedido ao Integration Broker, por parte do Orchestrator, é possível visualizar na Figura 15.

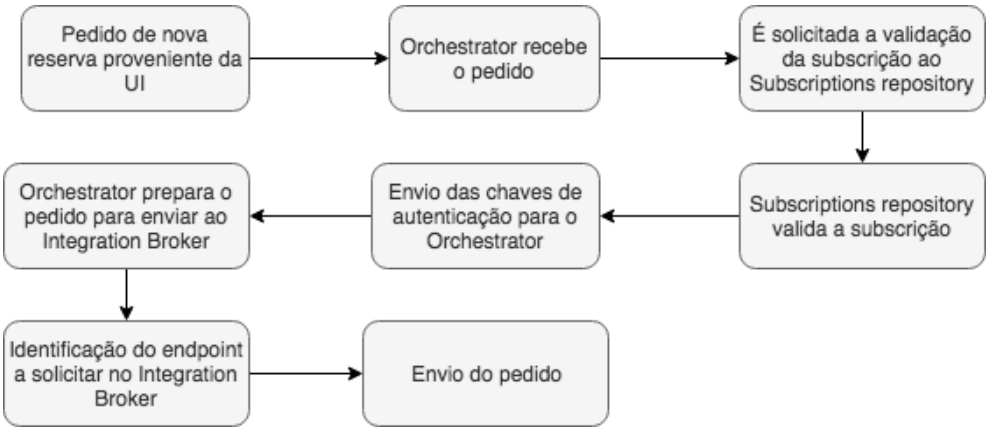

Figura 15 - Diagrama de fluxo de um pedido do Orchestrator ao Integration Broker

Para consultar mais detalhes acerca desta componente, o leitor poderá consultar o Anexo 8.

7.3.3. Integration Broker e Integration Service Drivers

O componente *Integration Broker* tem como objetivo principal estabelecer a ligação entre a plataforma *Hotelcracy*, e os diversos *Integration Service Drivers*. Estes serão associados a cada um dos serviços integrados, e onde será feita a conversão dos dados no sentido plataforma-serviço externo, e vice-versa. Depois de preparado o pedido, serão também responsáveis pela invocação dos *endpoints* dos serviços externos.

Apenas nos *Service Drivers* é que é se faz a conversão de dados, podendo assim destacar-se duas partes distintas, mas que cooperam entre si. No *Driver* propriamente dito, podem encontrar-se todas as funcionalidades a que é possível aceder utilizando determinado serviço. Para cada funcionalidade é criado um método distinto, sendo que é aqui que o pedido vai ser recebido,

para posteriormente ser enviado ao serviço externo. Desta forma, é exigido que em cada método esteja presente o *url* do *endpoint* da API do serviço a que corresponde a funcionalidade a invocar. Para além disto, é também necessária a existência de um conversor de dados, descrito já na secção 6.1.3. Para efetuar o pedido, os *Service Drivers* têm a responsabilidade de ler a estrutura de dados recebida de acordo com o modelo canónico e converter para o formato exigido no *endpoint* da API do SaaS externo. Depois de efetuado o pedido, a resposta proveniente do serviço é lida, e convertida para o modelo canónico, para posteriormente ser reencaminhada para o *Integration Broker*.

Tal como já referido, o *Integration Broker*, tem um papel em muito semelhante a um *gateway*. Com efeito, este apenas terá o papel de realizar a transição de dados entre o *Orchestrator* e os *Service Drivers*. No entanto, este também realiza o controlo das funcionalidades a que cada serviço tem acesso. Dentro da componente *Integration Broker* serão vários os *endpoints* disponíveis, sendo que cada um corresponde a uma funcionalidade. Quando é recebido um pedido, este terá que validar previamente que o serviço, a que se quer aceder efetivamente, permite executar determinada funcionalidade. Em caso afirmativo, o pedido é reencaminhado para o *Service Driver* correspondente, caso contrário, é retornada uma mensagem de erro. Para além do controlo de funcionalidades, este também faz a verificação prévia de que as chaves necessárias para efetuar o pedido são fornecidas. Na Figura 16 é possível visualizar o fluxo seguido por um pedido a um serviço SaaS externo.

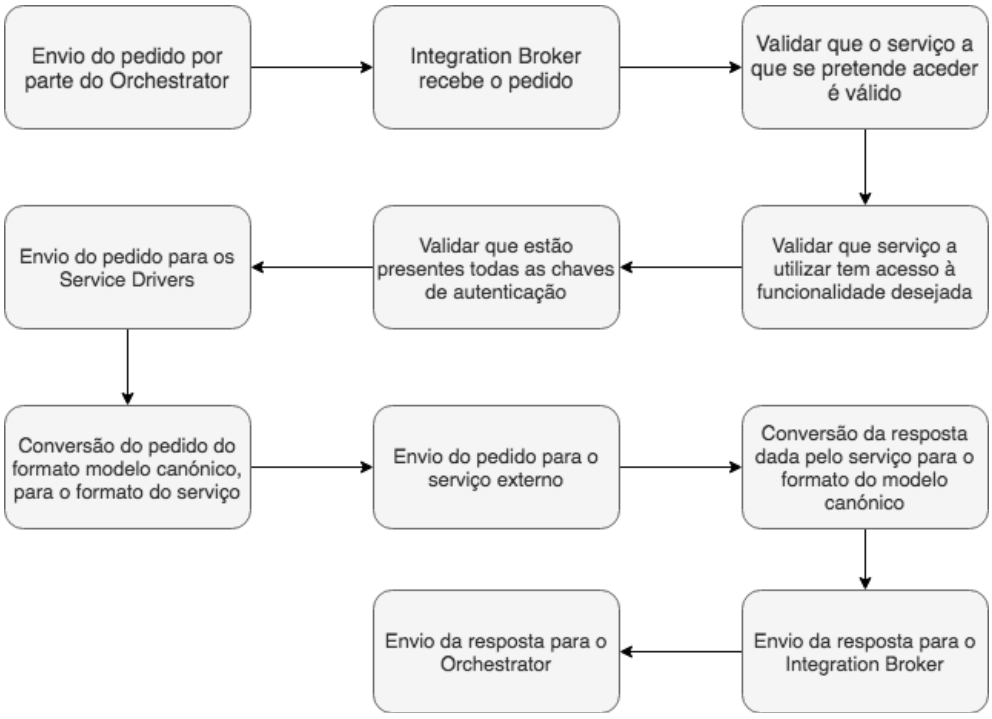

Figura 16 - Diagrama do fluxo de um pedido a um serviço externo

Concluindo, o trabalho desenvolvido nesta componente foi focado nas funcionalidades anteriormente descritas. Ao nível do *Integration Broker*, era necessário "construir o caminho" para um novo serviço, indicar as funcionalidades possíveis de se realizar no mesmo, bem como indicar as chaves que eram necessárias. Quaisquer restrições necessárias para aceder ao serviço, eram também descritas nesta componente. Posteriormente, foi criado o *Service Driver* respeitante a cada um dos serviços integrados, sendo que deveria ser igualmente criado um método para cada funcionalidade passível de ser acedida. Dentro dos *Service Drivers*, eram também criados os conversores de dados, de forma a converter pedidos e respostas, nos formatos desejados.

Para consultar mais detalhes acerca desta componente, o leitor poderá consultar o Anexo 9.

7.4. Ambientes

No presente estágio foram considerados 3 tipos de ambientes diferentes: desenvolvimento, teste e *staging*. O ambiente de produção não foi considerado, uma vez que este só viria a ser implementado numa fase mais avançada do projeto.

O ambiente de desenvolvimento consiste num conjunto de processos e ferramentas que são utilizados para desenvolver o código [40]. No contexto do presente estágio, este ambiente refere-se essencialmente ao computador pessoal do estagiário, uma vez que foi a ferramenta utilizada para desenvolver e testar localmente todo o código produzido.

Para testar o código produzido, existiram dois ambientes diferentes. Primeiramente, logo após ser produzido, o código foi testado localmente no computador do estagiário. Posteriormente à execução dos testes locais, estes eram submetidos à ferramenta *CircleCI* [41]. Esta permite uma abordagem em *continuos integration*, permitindo realizar a *build* e submeter os testes a todo o código desenvolvido armazenado num único local, por exemplo num repositório *Github* [42]. Desta forma, sempre que o código era enviado para o repositório do projeto, era automaticamente submetido a um conjunto de testes guardados na ferramenta *CircleCI*. Antes de ser transferido para o *branch* principal, o código submetido teria que obrigatoriamente passar em todos os testes.

O ambiente de *staging* serve como uma réplica do ambiente de produção. Estes ambientes servem para realizar testes ao código, *builds* e *updates*, de forma a garantir que a qualidade do produto se preserva no ambiente de produção [43]. Este requere assim uma cópia das configurações de hardware, servidores, bases de dados e caches, a utilizar no ambiente de produção.

7.5. Testes

A fase de testes tem como objetivo a avaliação do sistema num todo ou de uma componente em específico, de forma a verificar se os requisitos delineados estão a ser cumpridos. A identificação de erros é um dos grandes objetivos desta fase, que deve ser aplicada em qualquer projeto de *software* [44].

Existem diferentes níveis para efetuar testes, sendo que no âmbito do estágio, e do projeto em que este foi inserido, foram utilizados três níveis distintos [29]. Os testes unitários são responsáveis pela verificação de partes especificas do código, sendo que, quando a programação é orientada a objetos, usualmente testam métodos ou classes. Ao nível de integração, os testes aplicados têm como finalidade o teste de diferentes módulos, a funcionar de forma combinada, com o intuito de verificar se estes cooperam da forma esperada. Por fim, os testes ao sistema correspondem a testes realizados ao sistema num todo, funcionando de forma completamente integrada. Estes servem também para validar que os requisitos delineados estão a ser devidamente cumpridos [45].

Quando se faz uso de TDD no desenvolvimento de um projeto, os testes a aplicar são geralmente do tipo unitários e de integração, pois esta metodologia foca-se na implementação de testes a pequenas partes de código que vão sendo desenvolvidas. Dada a arquitetura MVC que a *framework Ruby on Rails* oferece, os testes implementados foram essencialmente aos modelos e controladores. As vistas não foram testadas, uma vez que estas são implementadas

por uma componente cujo desenvolvimento não estava ao encargo da equipa em que o estagiário foi inserido.

Na Tabela 17 é possível visualizar estatisticamente o trabalho que foi desenvolvido nesta etapa de desenvolvimento. Na primeira coluna está identificada a componente para que os testes foram desenvolvidos. Na coluna seguinte, está presente o número de testes que foram desenhados para essa componente, seguido logo do tempo de execução do mesmo. No final é demonstrada a percentagem de cobertura de código que os testes alcançaram, sendo que um dos objetivos era manter sempre perto dos 100%. Para a componente *Services Catalogue* não foram desenvolvidos testes, uma vez que para esta só era necessário testar os modelos existentes, e esse trabalho já havia sido desenvolvido.

Componente	Número de Testes	Tempo de Execução	Cobertura Total
Orchestrator	96	2,823seg	98,87%
Integration Broker	103	5,51seg	100,00%
Service Drivers	91	1,484seg	100,00%

Tabela 17 - Tabela com análise estatística dos testes desenvolvidos

Para além dos testes automatizados anteriormente referidos, ocasionalmente também foram efetuados alguns testes com recurso à ferramenta *Postman* [46]. Esta é uma ferramenta *desktop* que permite ao utilizador efetuar pedidos a qualquer *API*. No âmbito do estágio, os pedidos efetuados eram com o principal objetivo de validar fluxos que começavam no Orchestrator, e terminavam no serviço *SaaS* externo, através da visualização dos resultados na *interface* nativa do mesmo.

Capítulo 8 – Conclusão

No seguinte capítulo são apresentadas conclusões sobre o trabalho realizado durante o presente estágio, através de uma síntese das componentes desenvolvidas, e o que acrescentaram ao projeto *Hotelcracy Apps*. Visto que o desenvolvimento do projeto ainda vai continuar depois do término do estágio, será também apresentado o trabalho a continuar no futuro.

8.1. Trabalho desenvolvido

O trabalho desenvolvido durante o presente estágio contribuiu positivamente para o avanço do projeto *Hotelcracy Apps*. Numa primeira fase, através da análise do estado de arte, atualizaram-se os documentos que dispunham dos serviços atualmente existentes e passíveis de serem integrados na plataforma. Posteriormente, com base nos que foram considerados mais prioritários, o estagiário evoluiu o modelo canónico de dados da plataforma, passando assim a contemplar as três categorias de serviços estudadas. Apesar do modelo ter sido atualizado com base nos serviços que o estagiário viria a integrar, as entidades inseridas possibilitam também a integração de outros serviços no futuro.

Para conseguir a integração dos serviços escolhidos, o estagiário foi também responsável pelo desenvolvimento, e consequente evolução das componentes complexas responsáveis pelo processo de adição de novos serviços à plataforma. À semelhança do modelo de dados, apesar destas terem sido desenvolvidas com base nos SaaS integrados, ficaram também preparadas para receber outros novos. Desta forma, qualquer marca que deseje integrar o seu serviço na plataforma, e que já tenha contempladas todas as entidades no modelo de dados, apenas terá que desenvolver o seu *Service Driver*.

Ao nível de equipa, o estagiário validou trabalho realizado por outros elementos responsáveis pelo desenvolvimento, através da visualização e aprovação de código que era submetido para a plataforma *Github*. Nas reuniões de consórcio, o estagiário manteve também um papel ativo, apresentando o trabalho desenvolvido entre reuniões.

8.2. Principais Obstáculos

A principal adversidade encontrada ao longo do presente estágio, prendeu-se essencialmente com a situação de não-fornecimento das chaves de acesso à API dos serviços definidos como mais prioritários de integração. Esta situação levou a que o estagiário aplicasse o plano de mitigação enunciado na Tabela 1, tendo assim como consequência uma quebra no fluxo planeado para o início do semestre. Parte dos processos aplicados no decorrer do primeiro semestre, tiveram que ser novamente utilizados, havendo assim um incremento no tempo necessário para a integração do serviço de PMS.

A utilização do repositório para partilha de código desenvolvido, também foi um dos obstáculos encontrados ao longo do presente estágio. Prendeu-se essencialmente com o uso dos comandos disponibilizados na ferramenta *Git*, que por vezes geraram situações de conflito no código desenvolvido pelo estagiário. No entanto, esta situação foi ultrapassada através da alocação de tempo adicional para dominar a ferramenta.

8.3. Trabalho Futuro

O trabalho futuro passa por estender o leque de serviços disponíveis na plataforma *Hotelcracy Apps*. Estes poderão pertencer a categorias já integradas na plataforma, ou até mesmo outras novas. De forma a se conseguir esta evolução, terá assim que haver uma repetição de algumas das etapas que compuseram o presente estágio, nomeadamente a evolução do modelo de dados, e o desenvolvimento das componentes.

Visto que um dos grandes objetivos do projeto é permitir a autónoma integração de serviços, por parte das marcas que assim o desejem, será também desenvolvida documentação neste sentido. Esta documentação fornecerá todos os passos necessários para desenvolver um *Service Driver*.

8.4. Considerações Finais

Os objetivos propostos para o presente estágio podem considerar-se cumpridos, visto que no total foram inseridos 5 novos serviços, de 3 categorias diferentes. Tal como já enunciado no subcapítulo 2.2.2 a integração do sexto serviço, não foi possível devido a circunstâncias externas ao estagiário. Como consequência desta situação, o modelo de dados sofreu uma evolução superior ao expectável, permitindo assim abrir o leque de serviços passíveis de se integrar no futuro.

De forma a se conseguir alcançar o sucesso final, o estagiário teve que dominar e utilizar diversos conceitos aprendidos ao longo do curso, como por exemplo, programação orientada a objetos. Para além disto, foi permitido um primeiro contacto com metodologias e técnicas de gestão do *software*, que permitiu ao estagiário presenciar o que até ao momento só havia estudado em teoria.

Os objetivos propostos permitiram arrecadar experiência de desenvolvimento ao nível empresarial, bem como uma série de boas práticas de programação. Através do trabalho em equipa (IPNlis), e entre equipas, foi possível experienciar e consolidar um conjunto de normas importantes para o sucesso de qualquer equipa num projeto de *software*.

Pode assim concluir-se, que toda a experiência vivida ao longo do presente estágio evoluiu o estagiário como engenheiro informático, sendo o ponto de partida da definição da sua personalidade a nível de profissional. O suporte dado pela equipa do IPNlis e pelos orientadores de estágio, permitiu que situações adversas fossem ultrapassadas de forma breve, permitindo o contínuo progresso do estágio que se dá agora como concluído.

Tabela de Anexos

Anexo 1: HC_Tabelas_PMS	**Confidencial: Não**
Título: Tabelas de mapeamento de entidades da categoria de PMS	
Ficheiro: Modelo_Canonico_PMS.pdf	
Anexo 2: HC_Tabelas_RM	**Confidencial: Não**
Título: Tabelas de mapeamento de entidades da categoria de Reputation Management	
Ficheiro: Modelo_Canonico_RM.pdf	
Anexo 3: HC_Candidatura_PT2020	**Confidencial: Não**
Título: Documento de candidatura ao programa PT2020	
Ficheiro: Anexo_Tecnico_Hotelcracy_VFINAL.pdf	
Anexo 4: HC_Arquitetura	**Confidencial: Sim**
Título: Documento arquitetural da plataforma	
Ficheiro: E2_2_arquitectura_hotelcracy_v1.0.pdf	
Anexo 5: HC_Macro_Requisitos	**Confidencial: Sim**
Título: Documento de especificação de macro-requisitos	
Ficheiro: E2_1_Doc_especific_macro_requisitos_sistema_v1.0.pdf	
Anexo 6: HC_Personas_Cenarios	**Confidencial: Sim**
Título: Documento de especificação de personas e cenários	
Ficheiro: E1.1_Def_Personas_e_Cenarios_v2.0.pdf	
Anexo 7: HC_Services_Catalogue	**Confidencial: Sim**
Título: Documento de especificação detalhada da componente *Services Catalogue*	
Ficheiro: E3_1_especificacao_detalhada_Services_Catalogue_v1.2.pdf	
Anexo 8: HC_Orchestrator	**Confidencial: Sim**
Título: Documento de especificação detalhada da componente *Orchestrator*	
Ficheiro: E3_1_especificacao_detalhada_Orchestrator_v1.2.pdf	

Anexo 9: HC_Integration_Broker	**Confidencial: Sim**
Título: Documento de especificação detalhada da componente *Integration Broker*	
Ficheiro: E3_1_especificacao_detalhada_IntBroker_v1.2.pdf	

Referências

[1] Hotelcracy Apps, "Proposta de Candidatura - Parte B (Anexo Técnico) - Sistema De Incentivos À Investigação E Desenvolvimento Tecnológico (SI I&DT)," 2015.

[2] The Economist, "The market for booking travel online is rapidly consolidating," 2014. [Online]. Available: http://www.economist.com/news/business/21604598-market-booking-travel-online-rapidly-consolidating-sun-sea-and-surfing. [Accessed: 12-Mar-2018].

[3] D. Gursoy, "Chaotic changes in distribution channels : implications for hospitality companies," vol. 1, pp. 126–137, 2010.

[4] Techopedia, "What is a Best of Breed System? - Definition from Techopedia." [Online]. Available: https://www.techopedia.com/definition/23200/best-of-breed-system. [Accessed: 16-Mar-2018].

[5] "Modelo Cascata: O que é e como funciona?" [Online]. Available: https://casadaconsultoria.com.br/modelo-cascata/.

[6] "Descrição do modelo." [Online]. Available: http://modelocascata.blogspot.com/.

[7] "Challenges With Waterfall Model," 2017. [Online]. Available: https://www.genorainfotech.com/article7.

[8] M. Vieira, "Module 3: Planning and Tracking - Risk Management." Coimbra.

[9] "Software-as-a-Service." [Online]. Available: https://en.wikipedia.org/wiki/Software_as_a_service. [Accessed: 16-Jun-2018].

[10] "Software as a Service (SaaS)." [Online]. Available: https://www.techopedia.com/definition/155/software-as-a-service-saas. [Accessed: 16-Jun-2018].

[11] "Application programming interface." [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface. [Accessed: 17-Jun-2018].

[12] "What is an API?" [Online]. Available: https://www.mulesoft.com/resources/api/what-is-an-api. [Accessed: 14-Jan-2019].

[13] "Cloudbeds." [Online]. Available: https://www.cloudbeds.com/pt-br/. [Accessed: 14-Jan-2019].

[14] "Beds24." [Online] Available: "www.beds24.com" . .

[15] "Olery." [Online]. Available: https://www.olery.com/. [Accessed: 14-Jan-2019].

[16] "Review Pro." [Online]. Available: https://www.reviewpro.com/pt-pt/. [Accessed: 14-Jan-2019].

[17] "IDScan." [Online] Available: "www.idscan.com" .

[18] "IDCheck." [Online]. Available: https://www.idcheck.io/. [Accessed: 14-Jan-2019].

[19] "Capterra." [Online]. Available: https://en.wikipedia.org/wiki/Capterra. [Accessed: 19-Jun-2018].

[20] M. Silva, P. Cioga, and T. Gonçalves, "Documento de especificação de macro-requisitos do sistema," 2017.

[21] Bruno Almeida, "Documento de arquitectura da plataforma Hotelcracy Apps."

[22] "Conceito de diagrama entidade relacionamento." [Online]. Available: https://www.lucidchart.com/pages/pt/o-que-e-diagrama-entidade-relacionamento. [Accessed: 16-Dec-2018].

[23] R. Amaro, "Concepção e desenvolvimento de uma plataforma de gestão de serviços SaaS para o sector do alojamento - integração e migração de serviços cloud," Universidade de Coimbra, 2018.

[24] "Database Schema." [Online]. Available: https://www.lucidchart.com/pages/database-diagram/database-schema. [Accessed: 26-Dec-2018].

[25] J. Maenpa, "XML, JSON and Schemas." [Online]. Available: https://www.idug.org/p/bl/et/blogid=278&blogaid=644. [Accessed: 26-Dec-2018].

[26] "TDD." [Online]. Available: http://agiledata.org/essays/tdd.html. [Accessed: 26-Dec-2018].

[27] "Github." [Online]. Available: www.github.com. [Accessed: 20-Jan-2019].

[28] "Github." [Online]. Available: https://pt.wikipedia.org/wiki/GitHub. [Accessed: 26-Dec-2018].

[29] "Software Testing Basics." [Online]. Available: https://usersnap.com/blog/software-testing-basics/. [Accessed: 26-Dec-2018].

[30] "TDD & Unit, Integration & Functional Tests." .

[31] "The Cycles of TDD." [Online]. Available: https://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html. [Accessed: 16-Jan-2019].

[32] "Why Do We Use Red-Green-Refactoring and How is it Related to Test Driven Development?" [Online]. Available: http://ontheroadtocode.blogspot.com/2015/09/why-do-we-use-red-green-refactoring-and.html. [Accessed: 26-Dec-2018].

[33] "Code Academy." [Online] Available: "www.codeacademy.com" .

[34] "Ruby for Beginners." [Online] Available: "http://ruby-for-beginners.rubymonstas.org/" .

[35] "Youtube." [Online] Available: "www.youtube.com" .

[36] "Ruby on Rails." [Online]. Available: https://rubyonrails.org/. [Accessed: 27-Dec-2018].

[37] M. Hartl, *The Ruby on Rails Tutorial*, 5th ed. .

[38] "RSpec." [Online]. Available: http://rspec.info/. [Accessed: 20-Jan-2019].

[39] A. Summer, *Everyday Rails: Testing with RSpec*. 2018.

[40] "Development Environment." [Online]. Available: https://www.techopedia.com/definition/16376/development-environment. [Accessed: 28-Dec-2018].

[41] "CircleCI." [Online] Available: "www.circleci.com" . .

[42] "Continuous Integration e Delivery com CircleCI 2.0." [Online]. Available: https://medium.com/reactbrasil/continuous-integration-e-delivery-com-circleci-2-0-86d137ba1826. [Accessed: 28-Dec-2018].

[43] "Staging Environment." [Online]. Available: https://searchsoftwarequality.techtarget.com/definition/staging-environment. [Accessed: 28-Dec-2018].

[44] "Software Testing Overview." [Online]. Available: https://www.tutorialspoint.com/software_testing/software_testing_overview.htm. [Accessed: 29-Dec-2018].

[45] "Software Testing Levels." [Online]. Available: https://www.tutorialspoint.com/software_testing/software_testing_levels.htm. [Accessed: 29-Dec-2018].

[46] "Postman." [Online]. Available: https://www.getpostman.com/. [Accessed: 20-Jan-2019].