Universidade do Minho
Escola de Engenharia
Departamento de Sistemas de Informação

Rui Pedro Costa Vieira

Caso de demonstração de uma framework para automatização do teste de APIs de aplicações SaaS

Guimarães, Outubro de 2015

Universidade do Minho

Escola de Engenharia

Departamento de Sistemas de Informação

Rui Pedro Costa Vieira

Caso de demonstração de uma framework para automatização do teste de APIs de aplicações SaaS

Dissertação de Mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação do

Professor Doutor Ricardo Jorge Silvério Magalhães Machado

Guimarães, Outubro de 2015

DECLARAÇÃO

Nome

Rui Pedro Costa Vieira

Endereço Eletrónico

Ruivieira418@gmail.com

Número do Cartão de Cidadão

13719710

Título dissertação

Caso de demonstração de uma framework para automatização do teste de APIs de aplicações SaaS

Orientador

Professor Doutor Ricardo Jorge Silvério Magalhães Machado

Co-orientador

Doutor Carlos Argainha

Ano de conclusão

2015

Designação do Mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

A realização deste trabalho foi um processo bastante exigente que não conseguiria realizar da mesma forma sem o contributo direto e indireto de algumas pessoas, às quais faço questão de agradecer.

Gostaria de começar por agradecer ao meu orientador Professor Doutor Ricardo Machado pela sua disponibilidade para me orientar neste projeto e pela sua ajuda sempre que solicitado.

Em seguida queria fazer um especial agradecimento a três pessoas da PRIMAVERA, que contribuíram para o sucesso deste trabalho, ao meu co-orientador Doutor Carlos Argainha, ao Engenheiro Nuno Vieira e ao meu colega Doutor Braúlio Batista, um muito obrigado por toda a ajuda e apoio que me deram.

Quero deixar um agradecimento especial aos meus pais, aos meus irmãos e cunhados, às minha sobrinhas e principalmente à minha namorada e amiga Sílvia Valente por acreditarem em mim apoiando sempre as minhas decisões nesta fase da minha vida, sem eles não teria metade da motivação para concretizar este objectivo.

Por fim queria também agradecer a todos os meus amigos da Tuna Universitária do Minho todo o apoio que sempre demostraram para que conseguisse ultrapassar esta etapa com o maior sucesso.

Um MUITO OBRIGADO a todos.

Resumo

Atualmente a implementação de serviços na *cloud* é uma prática cada vez mais comum nas empresas ligadas à área dos sistemas de informação. Relativamente ao facto de vivermos num mundo cada vez mais informatizado, os sistemas existentes são cada vez mais complexos devido à sua dimensão, e o mercado por sua vez exige cada vez mais às empresas ligadas ao desenvolvimento de software uma maior qualidade, eficiência e rapidez na apresentação de novas soluções. Com isto aumentou a necessidade das empresas de desenvolvimento de software adotarem metodologias de desenvolvimento ágeis. Sendo o *SCRUM* uma dessas metodologias, esta contém ciclos de *sprints* curtos, com a duração de duas a quatro semanas, fazendo com que no final de cada *sprint* seja complicado a nível de tempo testar manualmente todas as funcionalidades do software. Os testes automáticos têm por isso um papel fundamental pois possibilitam uma maior capacidade de testar com repetição cobrindo assim um maior número de funcionalidades do produto, refletindo-se na qualidade do produto final.

Esta dissertação está enquadrada num contexto real, em colaboração com a PRIMAVERA Software, esta propõe-se começar por realizar uma selecção de ferramentas de teste de APIs, para posteriormente escolher a mais adequada e utilizá-la na automatização de um conjunto de casos de teste sobre o produto *EAM* da empresa PRIMAVERA, inserido na *cloud*. Como resultado final desta dissertação surge a aplicação de uma *framework* constituída por um ambiente de desenvolvimento e um ambiente de execução de testes à API, sendo que a primeira é constituída pelo projecto do *ReadyAPI* e tem como principal objectivo simplificar a manutenção futura e a adição de novos *scripts* de testes. Através da utilização da metodologia *data-driven*, da criação de variáveis globais, da parametrização de pedidos *http* e recorrendo a um projecto comum criado com o objectivo de ser uma biblioteca de dados e funcionalidades, a manutenção e inserção de novos *scripts* poderá ser um processo tão simples como inserir ou alterar os dados do ficheiro *datasource*. Relativamente ao ambiente de execução, este faz uso de uma *cloud* privada garantido que o ambiente de testes não difere nas várias execuções, resultando assim que qualquer alteração nos testes tem origem apenas na aplicação que está a ser testada.

Palavras-chave: *Cloud*; Garantia de qualidade; Testes automáticos; *Cloud*; Framework; *API*; *Data-driven*; *Datasource*; *Scripts*; *Builds*.

Abstract

Currently the implementation of services in the cloud is an increasingly common practice in companies linked to the area of information systems. Regarding the fact that we live in an increasingly computerized world, existing systems are becoming more complex because of its size and the current market require more quality, efficiency and speed to the companies linked to the software development in the presentation of new solutions. With this has increased the need of software development companies adopt agile development methodologies. Being the Scrum one of those methods, it contains short sprints cycles lasting two to four weeks, making the end of each sprint complicated in terms of time to manually test all the features of the software. Therefore, automated tests have an important role because they allow a greater ability to test with repetition covering a larger number of product features that are reflected in the quality of the final product.

This thesis is framed in a real context, in collaboration with PRIMAVERA software, it is proposed to start by performing a selection of API's testing tools, to later choose the most appropriate and use it to automate a set of test cases with the PRIMAVERA product EAM, inserted into the cloud. The end result of this work comes to applying a framework that consists of a development environment and a runtime environment to test the API, being the first the ReadyAPI project and aims to simplify future maintenance and adding new test scripts. Through the use of data-driven methodology, the global variable creation, the http order parameterization and using a common project created to be one data and functionality library, the new maintenance and insert scripts can be a process as simple as insert or change data to the datasource file. For the execution environment, this makes use of a private cloud, ensured that the test environment does not differ in the different executions, thus resulting in that any change in testing is only source in the application being tested.

Keywords: Cloud; Quality assurance; Automated testing; Cloud; Framework; API; Data-driven; datasource; scripts; builds.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

API - Application Programming Interface

DSR - Design Science Research

EAM – Enterprise Asset Management

HTML – HyperText Markup Language

HTTP - Hypertext Transfer Protocol

HTTPS - HyperText Transfer Protocol Secure

IAAS - Infrastructure as a service

JSON - JavaScript Object Notation

PAAS - Platform as a service

REST - Representational State Transfer

SAAS - Software as a service

SOA - Service-oriented architectures

SOAP - Simple Object Access Protocol

TFS – Team Foundation Server

URI - Uniform Resource Identifier

URL - Uniform resource locator

XML - Extensible Markup Language

1 Introdução

Esta primeira fase do documento pretende explicar em que consiste o mesmo, começando por descrever o seu enquadramento e a respetiva motivação fundamentando assim a escolha do tema.

Outro dos pontos mais importantes desta fase é descrever os objectivos desta dissertação, tendo em conta principalmente a importância que este trabalho terá no futuro para a empresa que colabora na realização desta dissertação, que é a PRIMAVERA Software.

No final será descrita a estrutura que o documento irá seguir.

1.1 Motivação e enquadramento

A minha motivação provém devido a trabalhar na área de automatização de testes há cerca de 2 anos, na empresa PRIMAVERA Software, penso que tenho recursos e ferramentas necessárias para a realização deste projecto, além disto a utilização de serviços na *cloud* é das tecnologias cada vez mais utilizada pelas empresas que desenvolvem software de gestão, como tal garantir a qualidade desses serviços através dos testes automáticos, é sem dúvida uma grande motivação.

Atualmente diversas empresas, de diferentes áreas de actuação, têm se direcionado para o desenvolvimento de software como serviço, o *SaaS*, devido às vantagens que este tipo de software contém, sendo bastante mais flexível e reutilizável, permitindo suporte a vários utilizadores numa infraestrutura de fácil configuração. Além disso contém um infraestrutura bastante dinâmica, com uma grande escalabilidade, mais conhecida como computação na nuvem, ou *cloud computing*. (Cambiucci, 2009)

Como tal a implementação de serviços na *cloud* têm sido uma prática comum nas empresas ligadas à área dos sistemas de informação. Identificar os novos paradigmas associados aos serviços disponibilizados neste tipo de sistemas e garantir a sua qualidade, é um factor essencial.

1.2 Objetivos

Atualmente a implementação de serviços na *cloud* são uma prática muito comum nas empresas ligadas à área dos sistemas de informação, é necessário identificar os novos paradigmas associados aos serviços disponibilizados neste tipo de sistemas e garantir a sua qualidade utilizando a automatização de testes. Pretende-se com este projecto realizar uma selecção de ferramentas de teste às APIs, para posteriormente escolher a mais adequada e com esta aplicá-la a um projecto real de automatização de testes à API, mais precisamente sobre o produto EAM pertencente à empresa PRIMAVERA.

Como resultado desta dissertação surge a implementação de uma *framework* constituída por um ambiente de desenvolvimento e um ambiente de execução, sendo que o primeiro tem como objectivo simplificar a manutenção futura e a adição de novos casos de testes, e a segunda tem como principal objectivo garantir que o ambiente de testes não difere nas várias execuções, resultando assim que qualquer alteração nos testes tem origem apenas na aplicação que está a ser testada.

1.3 Abordagem Metodológica

Tendo em consideração a natureza desta investigação, será adotada a metodologia de investigação do *Design Science Research* (DSR).(Vaishnavi & Jr, 2007)

As duas primeiras atividades do modelo dizem respeito ao primeiro capítulo desta dissertação, que são a atividade de identificação, motivação do projecto, e a atividade de definição dos objectivos, ver Figura 1.

No capítulo referente ao "estado de arte" foi utilizada a abordagem de revisão de literatura. Através desta abordagem foi realizada uma revisão da bibliografia, reforçada com uma análise sobre artigos e casos de estudo já realizados sobre temáticas relacionadas com o mesmo, tendo o objetivo de conseguir obter um conhecimento mais profundo e uma melhor compreensão para o desenvolvimento do projeto.

Este estudo terá como resultado final uma *framework* de testes à API, sendo esta constituída por um projecto de automatização de testes utilizando a tecnologia *ReadyAPI*, tecnologia esta que surge do capitulo 3 referente à selecção de tecnologia de suporte ao teste de APIs, tendo

como principal objectivo simplificar a manutenção futura e a adição de novos *scripts* de testes e garantir que o ambiente de testes não difere nas várias execuções.

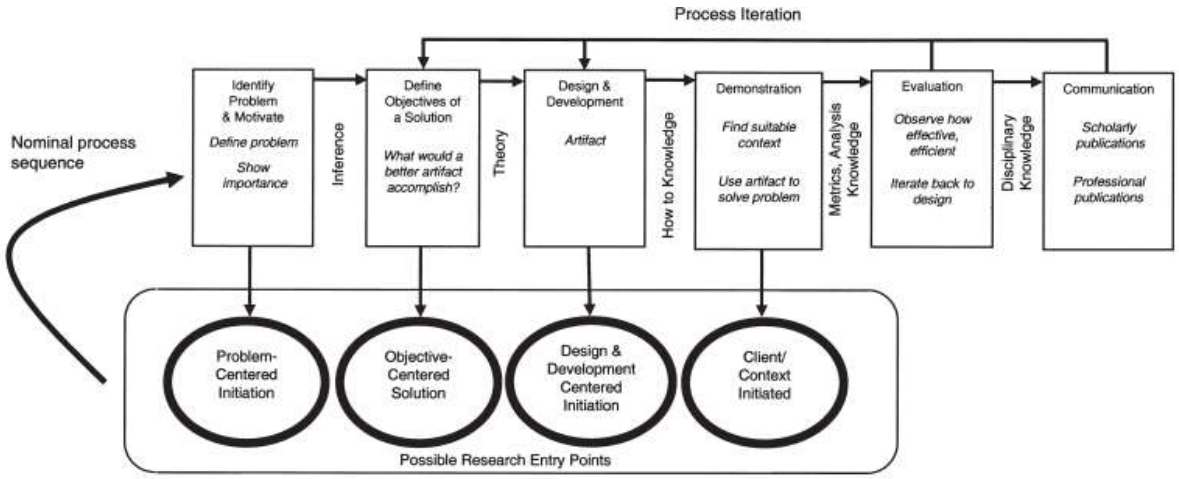


Figura 1 -Modelo de processos da metodologia, retirada de (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007)

A abordagem metodológica *Design Science Research (DSR)* têm como objectivo garantir a melhor orientação para o desenvolvimento de um projecto ligado à área de sistemas de informação, resultando na construção e aplicação de artefactos com o objectivo de resolver o problema proposto.(Hevner, March, & Park, 2004)

Esta é uma metodologia que pretende que sejam respondidas questões relevantes e de cariz científico, resultando na construção de artefactos inovadores. Para isto é necessário adquirir conhecimento para aplicar na interpretação e resolução do problema em questão, promover a aplicação de artefactos, sendo estes novos ou versões já melhoradas.(Peffers et al., 2007)

Como resultado na utilização desta metodologia pretende-se que se realize um estudo para entender o problema em questão recorrendo à utilização e criação de artefactos, com objectivo final de encontrar a solução pretendida.

1.3.1 Dificuldades e riscos

No desenvolvimento de um projecto existem várias dificuldades e riscos que podem surgir, ameaçando o sucesso do mesmo. No âmbito deste projecto destacam-se as seguintes:

1. Elevado custo de aquisição de uma ferramenta de automatização, levando à escolha de ferramentas de menor qualidade;
2. Infraestrutura da PRIMAVERA não suportar a aplicação do sistema resultante desta dissertação;

Durante o processo de selecção de ferramenta de automatização de testes a APIs, é necessário realizar uma análise detalhada de forma a garantir que a ferramenta escolhida vá de encontro aos requisitos do produto que vai ser testado e assim minimizar o risco de uma má escolha.

Para diminuir o risco da Infraestrutura da PRIMAVERA não suportar todos os cenários de execução necessários para a implementação deve-se avaliar a infraestrutura actual da PRIMAVERA e em conjunto com os requisitos do projecto de automatização verificar se existem incompatibilidades a nível de tecnologias.

1.3.2 Estratégia de Investigação

Para o desenvolvimento deste projecto é essencial adquirir conhecimento para compreender o problema e contextualiza-lo, resultando num trabalho de investigação e revisão bibliográfica. Este conhecimento diz respeito à informação e conceitos existentes que estão relacionados com o projecto.

Antes de iniciar o processo de revisão de literatura, foi necessário definir um conjunto de palavras-chave de forma a obter informação com qualidade e objectiva. As palavras-chaves utilizadas foram as seguintes: "Evolução no Desenvolvimento de software","*Cloud Computing*", "Testes de software", "Testes automáticos" e "Testes automáticos a Serviços".

Os motores de pesquisa utilizados para obter informação utilizando estes tópicos foram os seguintes:

- Google académico;
- RepositóriUM;
- Citeseer;
- Microsoft academic;
- I EEE Xplore;

As pesquisas foram realizadas em português e inglês.

De forma a escolher a melhor informação do material encontrado, numa primeira fase foi necessário avaliar a credibilidade da informação e realizar uma leitura do resumo respectivo a cada documento, seguindo de uma leitura mais detalhada retirando a informação necessária e garantir a relação com os objectivos do projecto.

Foi utilizada a ferramenta Mendeley, para que de forma organizada fizesse a gestão das várias referências.

1.4 Estrutura do Documento

O presente documento tem uma estrutura sequencial sob a forma de capítulos. Neste primeiro capítulo de introdução é apresentada uma visão sobre o enquadramento deste projeto, quais os riscos e contributos que este projeto tem e quais os objetivos que se pretendem alcançarem com o desenrolar do trabalho. É também apresentada neste capítulo a abordagem metodológica seguida para atingir os resultados esperados.

No segundo capítulo é apresentado o estado da arte, este capítulo tem como objectivo descrever o que existe actualmente sobre o tema abordado, sendo este uma base de conhecimento para entender melhor o propósito e resultado desta dissertação. Este segundo capítulo é constituído por três subcapítulos: Evolução no desenvolvimento de software, *Cloud Computing* e testes de software. Inicia com uma apresentação à evolução no desenvolvimento de software, passando para uma abordagem relativa ao subcapítulo *cloud computing*, onde é definido o conceito, é feita uma descrição dos tipos e modelos de *cloud* existentes e uma abordagem aos protocolos de comunicação. O terceiro subcapítulo corresponde aos testes de software, apresentando algumas definições do tema em questão, seguindo para a descrição das várias fases e tipos de testes existentes, apresentação de várias definições relativas aos testes automáticos fazendo uma referência às suas vantagens e objectivos, finalizando com uma abordagem aos testes automáticos a serviços, servindo como ponto de ligação às próximas fases do projecto.

No terceiro capítulo é apresentado uma selecção de ferramentas de testes de APIs, onde são descritas três ferramentas de automatização de testes a serviços. Para que a análise entre as ferramentas fosse coerente, foi necessário criar um cenário único de forma a aplica-lo na

experimentação das várias ferramentas. Além disto foram definidos uma serie de parâmetros de comparação para avaliar a capacidade de funcionamento destas.

No quarto capítulo é demonstrado o caso de demonstração na PRIMAVERA Software. Este capítulo contém a descrição da metodologia de desenvolvimento utilizada, plano de teste, apresentação da *framework* desenvolvida que contém o ambiente de execução e desenvolvimento, finalizando com o projecto do *ReadyAPI*.

No quinto e último capítulo são feitas as conclusões sobre todo o trabalho desenvolvido, indicando as limitações encontradas na realização do trabalho e apontar algumas soluções para trabalho futuro.

2 Estado da arte

O presente capítulo pretende apresentar o estado de arte relativo às várias temáticas abordadas neste trabalho. A primeira abordagem neste capítulo começa por apresentar a evolução existente na área de desenvolvimento de software, seguindo para uma abordagem relativa ao *cloud computing*, e por fim uma abordagem aos testes automáticos, tendo um maior enfoque na automatização de testes a serviços.

2.1 Evolução no desenvolvimento de software

Temos vindo a assistir a uma evolução no desenvolvimento de software que inicia na década de 70 até aos dias de hoje.

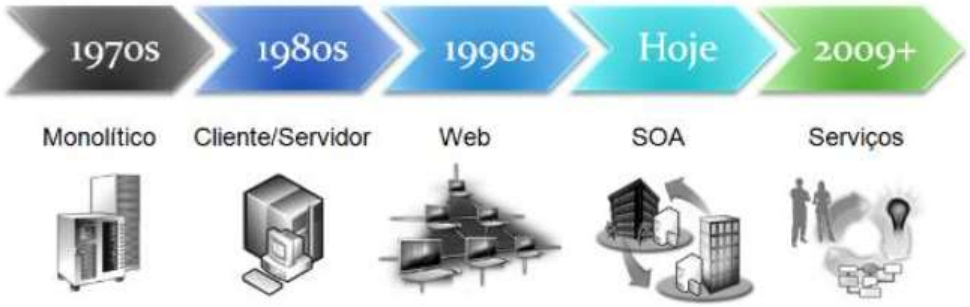


Figura 2 - Evolução do ambiente de TI em década, figura retirada de (Cambiucci, 2009)

Nesta primeira década de desenvolvimento de software temos os sistemas monolíticos, sistemas estes que necessitavam de um enorme conhecimento para poderem ser manuseados, tanto para o programador como para o utilizador final, além do mais eram sistemas bastante dispendiosos tanto no seu manuseamento como na sua manutenção.

A primeira grande evolução foi na década de 80, onde surgem os primeiros computadores pessoais, com custos de processamento mais acessíveis para o utilizador final. O desenvolvimento de software em linguagens como o Basic, utilizando máquinas pessoais, possibilita ao utilizador uma maior facilidade no desenvolvimento de software, bem como uma menor necessidade em formação especializada para o fazer. Já no final da década de 80 surge o modelo cliente/servidor, utilizado principalmente em redes interoperáveis, integradas em

ambientes corporativos, permitindo assim às empresas começarem a integrar sistemas de processamento.

Nos anos 90 assistimos ao início da internet, vantajosa no baixo custo para as universidades, empresas e utilizadores domésticos. Com isto foram surgindo equipamentos com capacidade de ampliar as capacidades de comunicação, enquanto uma série de topologias de rede surgiam. Aplicado ao funcionamento destes equipamentos eram introduzidas funcionalidades de aplicações através de serviços, que respondiam a diversos pedidos através do protocolo TCP/IP, normalmente customizados para cada setor industrial. Estes serviços eram implementados possibilitando o acesso a funcionalidades das várias aplicações fora das empresas.

Protocolos como o *HTTP*, *SOAP*, *HTML* e *XML* surgem no século XXI, passam a permitir a utilização de sistemas que fossem integrados mais rapidamente, sistemas estes, escaláveis no que diz respeito ao número de utilizadores e de aplicações web, onde conceitos como *Web Services* ganharam uma enorme relevância.

Já em 2009 surge um novo paradigma no setor das tecnologias da informação, designada como computação na nuvem. Grandes empresas como a Microsoft, Amazon e Google iniciaram este novo paradigma com o fornecimento de recursos de infraestrutura através de *datacenters* espalhados pelo mundo, iniciando assim o surgimento de uma nova TI, neste caso mais dinâmica, flexível e híbrida, através da combinação de serviços locais, alojados em servidores locais, bem como serviços remotos, hospedados em diversos *datacenters*. Sendo esta mais flexível, relativamente aos custos também o é, isto porque apenas as empresas só pagam pelos recursos que utilizam pelo período temporal em que são utilizados. (Cambiucci, 2009)

2.2 Cloud Computing

Em português é conhecido como computação na nuvem, é uma tecnologia que permite o processamento, armazenamento e a utilização de vários software, que estão armazenados na rede, onde o seu acesso é feito remotamente via internet, ou seja, o acesso remoto não é possível apenas ao software, também é possível aos recursos de hardware, tais como o processamento e armazenamento. (SISNEMA, 2009)

A definição de *cloud computing* do NIST (*National institute os Standards and Technology*) descreve a computação na nuvem como um modelo de computação que permite o acesso a um conjunto de funcionalidades computacionais disponibilizadas em rede, tais como redes de

comunicações, servidores, armazenamento, aplicações e serviços, onde a sua manutenção ou atualização é realizada pelo fornecedor de serviços.(Mell & Grance, 2011)

Figura 3 - Funcionamento Cloud Computing, figura retirada de (exuberantsolutions, s.d.)

2.2.1 Tipos de Cloud

A computação na nuvem divide-se em três tipos de fornecimento de serviços, que são infraestrutura, armazenamento de dados, plataformas e software, nos quais estão associados três tipos de *cloud*: Privada, pública e hibrida, onde partilham o objectivo comum de promover o ambiente de trabalho virtual tendo por base o aspecto colaborativo da tecnologia.

(1) *Cloud* Privada: é utilizada exclusivamente por uma organização, onde a gestão dos recursos disponibilizados pode ser da responsabilidade da mesma, de uma entidade externa, ou em alguns casos, de ambos. Esta pode ser implementada no *datacenter* da organização, ou no *datacenter* do fornecedor. Este tipo de *cloud* promove o aumento da produtividade, flexibilidade e escalabilidade, com a vantagem de ter uma alta capacidade de customização, podendo aumentar a eficiência de servidores e de *datacenters* diminuindo os custos inerentes à implementação e aumentando a produtividade da organização.

Neste tipo de *cloud* é necessário haver uma análise preliminar às necessidades da empresa, pois o custo de operação contínua da nuvem privada pode exceder o custo de uso de uma *cloud* pública.

(2) *Cloud* Pública: a utilização dos seus serviços está direcionada para o público em geral, a gestão dos vários recursos e serviços prestados por este tipo de *cloud* é da responsabilidade de uma determinada organização com fins de negócio, por exemplo de uma universidade, para fins acadêmicos. A organização que disponibiliza estes serviços, cobrando por eles, têm a responsabilidade de alojar, proteger, gerir e fazer a devida manutenção dos dados dos clientes, sendo estes utilizadores comuns ou empresas.

Sendo que os serviços disponibilizados pela *cloud* pública são compartilhados por empresas e utilizadores comuns, para as empresas podem surgir problemas de segurança, devido ao caracter público da *cloud*, ou seja, é um modelo adequado para empresas que tenham um orçamento mais reduzido e com outras prioridades, já não tão aconselhado para empresas com um grande volume de dados confidenciais.

(3) *Cloud* Hibrida: É a combinação de *clouds* privadas com *clouds* públicas, ou seja, permite manter sistemas provenientes da nuvem privada e pública simultaneamente, como por exemplo empresas que lidem com dados confidenciais possam ter esses mesmos dados numa *cloud* privada, enquanto outros sistemas possam ser utilizados numa *cloud* pública.

Devido à sua escalabilidade dinâmica este modelo hibrido permite a uma organização aumentar facilmente a capacidade de um servidor para um determinado período de tempo, sendo que, passando este período de tempo têm a vantagem de voltar à capacidade anterior.

Este tipo de *cloud* aprimora o controlo de aplicações que a empresa necessita, fazendo a análise da melhor opção, sendo dos modelos mais utilizados no mercado corporativo. (Cloudways, 2011);(Lopes, 2015); (IBM, s.d.)

2.2.2 Modelos de serviços Cloud

Software como serviço (SaaS) - Mais conhecido como SaaS (Software *as a service*), é um software que integra na recente tecnologia de *cloud computing*, onde o seu acesso é feito através de protocolos de internet. A tecnologia utilizada pelo *SaaS* possibilita que o utilizador tenha um acesso remoto aos seus dados e arquivos num processo de armazenamento seguro.

Não sendo necessários custos com o hardware, licenciamento e manutenção num sistema *SaaS*, acontece que o seu acesso através da web é pago pelo tempo de utilização dos serviços disponibilizados.

A principal vantagem de uma arquitetura *SaaS* é permitir compartilhar uma infraestrutura entre vários clientes, podendo partilhar assim várias instâncias do mesmo software, de forma simultânea.

O processamento é feito na nuvem, o utilizador não tem de se preocupar com a instalação ou atualização do software, sendo que a capacidade de computação é escalonável consoante as necessidades do utilizador.

O investimento inicial por parte do utilizador num software *SaaS* é mais reduzido que na aquisição de um software nativo, e no que diz respeito à amortização do investimento efetuado o utilizador não se vê obrigado a esperar um longo prazo de tempo. Como tal no *SaaS* o utilizador adquire as licenças de utilização como um serviço.

O modo de acesso dos utilizadores *SaaS* é feito através de um *browser*, possibilitando o acesso ao software em qualquer parte do mundo via computador, *tablet* ou *smartphone.*(Iyer, 2014; Lopes, 2015)

Plataforma como serviço (Paas) - É um modelo que fica entre o modelo *SaaS* e o modelo *Iaas*, é constituída por um ambiente de computação em camadas e soluções como serviço, proporcionando um ambiente mais robusto e flexível na utilização de bastantes recursos de tecnologia, possibilitando o desenvolvimento de software tendo como base uma *framework* de desenvolvimento. Ao contrário do *SaaS* que não permite a implementação de um recurso específico necessário para o negócio, o modelo *Paas* possibilita utilizar a mesma estrutura, tal como se utiliza nos nossos computadores pessoais, com a diferença de ser "*as a service*", e faz com que os utilizadores não tenham de se preocupar com recursos de hardware e software. Este modelo também utiliza o método de pagamento por utilização de recursos.

Infraestrutura como serviço (Iaas) - Têm como principal vantagem possibilitar que o utilizador adicione servidores virtuais e dispositivos de infraestrutura, invés do utilizador ter de comprar este tipo de hardware. O valor do serviço depende do número de servidores virtuais utilizados, tráfego, armazenamento de dados, entre outros.

O modelo *Iaas* é bastante dinâmico, pois o utilizador têm a possibilidade de requisitar servidores por um determinado período de tempo, e após isso pode cancelar a sua utilização. (Ricardo, 2013);(Sharma, Bansal, & Sharma, 2012); (Magalhães, 2012)

2.2.3 Protocolos de Comunicação

A escolha de protocolos de comunicação é essencial para o desenvolvimento de aplicações web, tanto no acesso a bases de dados, como na integração com serviços em *middleware*, a software do cliente. Existe uma grande quantidade enorme de protocolos, neste projecto apenas vamos abordar dois deles, *Soap* e *Rest*. (Rozlog, 2013)

SOAP

O protocolo de comunicação *Soap(Simple Object Access Protocol)* utiliza três entidades, o fornecedor de serviço, o registo de serviço, e o consumidor do serviço (ver Figura 4).

O fornecedor do serviço é o serviço, a entidade de endereçamento de rede aceita e executa o pedido do consumidor, já o consumidor do serviço é uma aplicação, serviço ou outro tipo de software que faz o requerimento deste. O registo de serviço trata-se de uma rede baseada num diretório que contém os vários serviços disponíveis. O consumidor do serviço procura a descrição do serviço no registo que é publicada pelo fornecedor de serviço, a comunicação estabelecida por estas entidades é baseada em *XML* e protocolo *SOAP*. A mensagem *SOAP* é constituída por um envelope, cabeçalho e um corpo. A identificação do *XML* como mensagem *Soap* é estabelecida pelo elemento envelope. O cabeçalho contém informações de chamada e de resposta. As mensagens e chamadas de método são definidas como documentos *XML*, e são enviadas através de um transporte de protocolo *SMTP, FTP, HTTP*, considerada uma das grandes vantagens do protocolo *Soap*, ao contrário do *Rest* que apenas utiliza o HTTP e o *HTTPS*. Embora o protocolo *Rest* seja mais fácil de entender e bem mais acessível, o *Soap* contém protocolos bem definidos e um conjunto de regras bem estabelecidas.(Mumbaikar & Padiya, 2013)

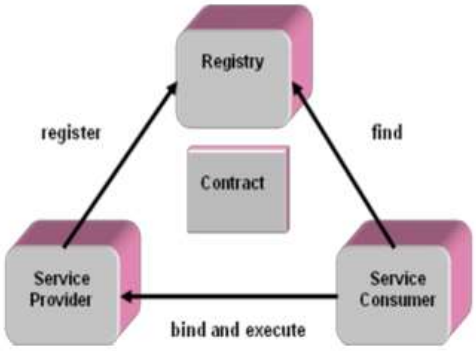


Figura 4 - Arquitectura Web Service, imagem retirada de (Mumbaikar & Padiya, 2013)

REST

Da autoria de *Royal Fielding*, o tipo de arquitectura que o protocolo *Rest* utiliza é cliente/servidor, onde o cliente envia um pedido e o servidor responde, ambos com suporte a *HTTP/HTTPS*. Os pedidos e respostas surgem em torno de representações de recursos, onde estas dizem respeito a um documento que identifica o estado atual ou pretendido de um recurso.

O protocolo *Rest* é um mais simplificado, e menos forte que o *Soap*, pois aproveita recursos da infraestrutura web já existente. A sua linguagem é baseada na utilização de substantivos e verbos, e não requer um formato de mensagem como envelope e cabeçalho tal como é exigido no protocolo *Soap*. Visto que não é necessário haver uma análise de ficheiros *XML*, a exigência deste protocolo relativamente à largura de banda é menor. As aplicações web que utilizam este protocolo designam-no como serviço web *Restful*, onde este utiliza o *GET*, *PUT*, *POST* and *DELETE*, ou seja métodos *HTTP*, para recuperar, criar, atualizar e excluir recursos.(Mumbaikar & Padiya, 2013)

Recursos *REST*

Os recursos são os objetos sobre os quais são feitas operações tais como criar, ler editar e apagar. Estas operações são designadas como *CRUD* (*Create*, *Read*, *Update*, *Delete*). Os clientes, artigos ou faturas são alguns exemplos de recursos.

O formato utilizado para representar os recursos tipicamente utilizada para *APIs REST* é *JSON* ou *XML*. O formato da representação do recurso poderá ser negociado entre o invocador e o serviço invocado.

Códigos de Resposta *HTTP*

O *standard HTTP* define uma lista de códigos de resposta que têm correspondência para o estado de resposta da *API*. Os intervalos de códigos são:

200 a 299
- indicam sucesso.

300 a 399
- indicam redirecção de pedido. Por exemplo, o código 301 indica que o URI do resource foi movido para outro URI.

400 a 499
- indicam erro no pedido ou em consequência do pedido.

500 a 599
- indicam erro no serviço

Como exemplo, o *standard HTTP* define que o código 201 deverá ser utilizado quando o recurso novo foi criado, da mesma maneira o serviço do produto PRIMAVERA responde com este código para um pedido de criação de um recurso com sucesso.

O código *HTTP* 403 define que o recurso acedido é proibido. O produto PRIMAVERA utiliza este código quando é feito um acesso em que o *token* de autenticação é inválido.

Mensagens HTTP

O protocolo HTTP é o mesmo protocolo utilizado para aceder a páginas web. O *standard* HTTP define as mensagens de pedidos e resposta, e os pedidos são compostos por:

- URI;
- Método HTTP;
- Cabeçalhos;
- Opcionalmente um corpo;

E as respostas são compostas por:

- Código de resposta;
- Cabeçalhos;
- Opcionalmente um corpo;

2.3 Testes de Software

Os testes de software são cada vez mais importantes para garantir a qualidade no desenvolvimento de software, normalmente baseiam-se em testar funcionalidades tendo em conta os requisitos definidos. Estes podem ser vistos como um dos requisitos de qualidade num processo de desenvolvimento de software.

Os testes de software não existem apenas na execução do software, sendo esta uma perceção comum. Estes também existem antes da própria execução. As atividades de testes incluem planear e controlar, escolher as condições de testes, desenvolver e executar os casos de teste, verificar os resultados, avaliar critérios de saída, manter os envolvidos informados sobre o decorrer do processo de testes e por fim completar as atividades de encerramento. Os testes também podem incluir a revisão de documentos e a realização de análise estática.

Os principais objectivos dos testes são: encontrar defeitos, ganhar confiança sobre o nível de qualidade, fornecer informação para tomadas de decisão e Prevenir defeitos. (Software & Qualifications, 2011)

A complexidade dos sistemas, o aumento na exigência de qualidade de software, a resposta rápida e eficaz implicam que os testes de software não possam ser executados manualmente de forma abrangente e segura. Por estes motivos a automatização de testes assumiu uma maior importância relativamente aos testes de software.

2.3.1 Fases de Testes

O objectivo de um teste é garantir que o que já foi desenvolvido, ou que está em desenvolvimento funcione corretamente de acordo com os requisitos definidos, como tal existem várias fases ou níveis de testes que incluem diferentes objectivos no que diz respeito ao tipo de inconsistências que cada uma pretende encontrar.

Teste Unitário - Nesta fase pretende-se que sejam testados pequenos excertos de código, ou seja, funcionalidades pequenas, normalmente testadas pelo próprio programador. A base de testes é relativa aos requisitos de componentes, concepção detalhada e o código. Os tipos de testes incidem sobre os componentes, programas, conversão de dados de programas/migrações

e módulos de bases de dados. Se o teste não identificar qualquer tipo de anomalia, a funcionalidade testada passa para a fase de integração.

Teste de Integração - Na fase de integração os testes realizados pretendem encontrar falhas que sejam provenientes da integração de componentes/funcionalidades, como por exemplo interacções entre as partes do sistema operativo, sistema de arquivos, hardware, e as interfaces entre estes sistemas. Têm como bases de testes a concepção do software e do sistema, arquitectura, fluxos de trabalho e casos de uso. Os tipos de testes incidem sobre a implementação de bases de dados, infraestrutura, interfaces e configuração de sistemas e dados de configuração.

Teste de Sistema - Têm como objectivo testar o comportamento do sistema na totalidade. Têm como base de testes a especificação de requisitos de sistema e software, casos de uso, especificação funcional e relatórios de análise de risco. Os tipos de testes incidem sobre os manuais de sistema, utilizador e operação, e na configuração de sistemas e dados de configuração.

Teste de aceitação - Habitualmente são da responsabilidade do cliente ou utilizador do sistema, pretendem validar se o sistema corresponde aos requisitos definidos no início do projeto e se estão em conformidade com as necessidades do mesmo. O objetivo é estabelecer a confiança no sistema, a pesquisa de anomalias nesta fase não é o foco. A base para os testes são os requisitos do utilizador/sistema, casos de uso, processos de negócio e relatórios de análise de risco. O tipo de testes realizados nesta fase incidem nos processos de negócio do sistema integrado, processos operacionais e de manutenção, procedimentos de utilizador, formulários, relatórios e dados de configuração.(Software & Qualifications, 2011)

Teste de regressão - São testes que têm como objectivo garantir que as alterações efetuadas no software não alteram o bom funcionamento do mesmo. Por vezes nesta fase são executados casos de testes antigos, onde os resultados novos são comparados com os antigos, com o objetivo final de serem iguais.

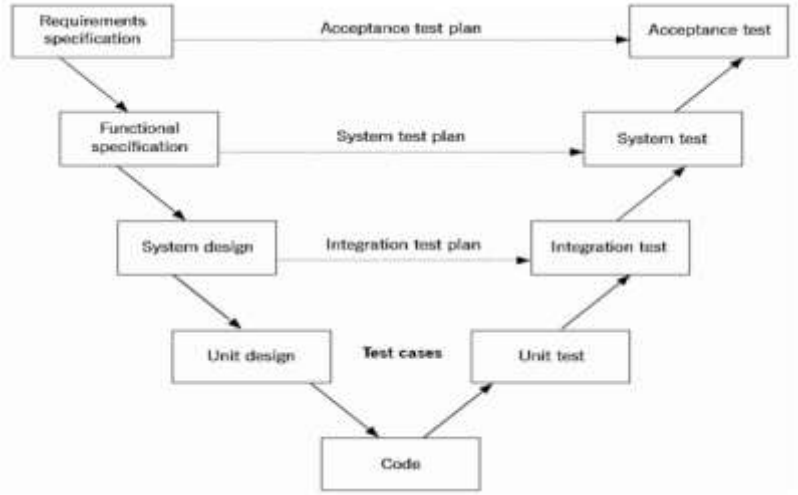


Figura 5 - Teste de software*, V –Model,, imagem retirada de* (Fernandes, 2014)

2.3.2 Tipos de Teste

Existem vários tipos de teste, testes caixa-preta, cobertura de código, testes funcionais, testes de interoperabilidade, testes de carga, testes de manutenibilidade, testes de desempenho, teste de portabilidade, testes de fiabilidade, testes de segurança, testes de stress, testes estruturais, testes de usabilidade e testes de caixa-branca.

Neste documento apenas vamos falar de 3 tipos, que são:

Teste Funcional - Baseiam-se nos testes às funcionalidades relacionadas com os requisitos do sistema. O seu objectivo é garantir que o que foi definido no início do processo corresponde às funcionalidades do sistema.

Teste não funcional - Este tipo de testes têm como objectivo avaliar aspetos não funcionais, tais como usabilidade, portabilidade, eficiência, desempenho, carga, etc.

Teste estrutural - Têm como objectivo avaliar aspectos relativos ao comportamento interno do software, tal como avaliar a qualidade do código.

2.3.3 Testes Automáticos

Um dos principais motivos que levaram os testes automáticos a assumir uma maior importância (Sharma & Mall, 2009), têm a ver com o facto de os software serem cada vez mais complexos, aumentando a necessidade de reexecução dos mesmos, onde os testes manuais por si só também se tornavam numa atividade bastante complexa. Com auxilio de um ferramenta de

automatização, permitem que os testes sejam executados de forma frequente, ou seja, ao contrário dos testes manuais que requerem uma disponibilidade constante de recursos humanos para testar manualmente uma grande quantidade de testes, processo do qual se torna bastante cansativo e de difícil repetição, nos testes automáticos após o processo de automatização, apenas é necessário executar um script ou até mesmo um simples clique num botão. Relativamente ao processo de automatização, ou desenvolvimento dos testes, é um processo que necessita de mais tempo para desenvolver e implementar os testes, do que na execução manual, mas em curto prazo é a melhor forma de economizar tempo e dinheiro.

Antes de iniciar um processo de automatização de testes, é muito importante validar os cenários que será necessários automatizar, pois caso este levantamento não seja bem feito podemos ter *scripts* de testes a validar mais do que uma vez o mesmo cenário. Outro aspecto muito importante é perceber se o produto em questão necessita de testes automáticos, caso o produto altere o seu interface com muita frequência, a automatização poderá não ser o processo mais adequado, devido ao facto que os testes teriam de ser refeitos. Outro caso onde os testes manuais são mais vantajosos é se o produto não tiver dimensão suficiente e que não tenha necessidade de reutilizar os testes.

A aplicação dos testes automáticos deverá iniciar logo na fase inicial de um processo de desenvolvimento, de forma a apoiar a equipa de desenvolvimento na procura de problemas que possam surgir, possibilitando que estes problemas sejam resolvidos mais rapidamente, reduzindo assim o custo da resolução do problema e possibilitando encontrar melhoramentos a introduzir no desenvolvimento. (Kansomkeat & Rivepiboon, 2013)(Fernandes, 2014)

Vantagens e objectivos da automatização de testes

As vantagens para uma organização que detém testes automáticos prende-se pelo facto de que a utilização destes promove a qualidade nos processos críticos, possibilitando assim a rápida resolução dos problemas encontrados, levando ao aumento de qualidade do produto e satisfação do cliente. O processo de testes manuais está diretamente relacionado com o trabalho de um recurso humano, que por vezes está sujeito a falhas, proveniente da imensa quantidade de resultados que necessitam de ser validados. Neste caso os testes automáticos garantem-nos uma análise mais robusta, eficiente e realizada sempre da mesma forma.

Outra grande vantagem é a possibilidade de executar os testes sempre que necessário, ao contrário dos testes manuais que necessitam de uma grande capacidade de recurso humanos para o fazer, tornando assim impossível existir um nível de regularidade como é frequentemente utilizado pelos testes automáticos.

Testes automáticos a serviços

Para poder iniciar um processo de automatização de testes a serviços é fundamental ter uma ferramenta de automatização específica para testar serviços. Existem inúmeras ferramentas para o efeito, a fase seguinte desta dissertação será realizar uma selecção de ferramentas de testes de *APIs*.

Para iniciar o teste de serviços *web*, é necessário criar um projeto com chamadas ao serviço. Será necessário definir e aplicar vários parâmetros dinâmicos para os caminhos do *URL*.
Em seguida, executar as chamadas selecionadas e monitorizar os resultados das execuções, opcionalmente, com o uso de validações de dados de resposta, por exemplo, se os dados de resposta contêm a frase em particular ou se o tamanho de dados de resposta está dentro do intervalo específico.
Em muitos casos, este tipo de ferramentas além de permitirem automatizar testes a serviços, podem ser utilizadas para testar a própria API, dado que têm a capacidade de mostrar e validar os formatos mais comuns de saída *REST*, tais como o *XML*, *JSON*, e *HTML*.

3 Selecção de tecnologia de suporte ao teste de APIs

3.1 Introdução

Neste capítulo será apresentado um estudo de ferramentas de automatização de testes a serviços. As ferramentas estudadas foram as seguintes: *ReadyAPI, Visual Studio – test perfomance* e o *Postman*.

Relativamente às ferramentas escolhidas para a comparação, a principal razão da escolha foi por serem das ferramentas mais referenciadas atualmente, e além disso no caso do *Visual Studio – test perfomance*, a PRIMAVERA já possuía uma licença.

Para que a análise entre as ferramentas fosse coerente, foi necessário criar um cenário único de forma a aplica-lo na experimentação das várias ferramentas. O cenário criado foi: *POST* de um login sobre o produto "*Elevation*" da PRIMAVERA. Além disto foi necessário definir uma serie de parâmetros de comparação para avaliar a capacidade de funcionamento das ferramentas, que são:

Estrutura dos testes - A estrutura dos testes é importante para, primeiro facilitar a leitura do mesmo e segundo facilitar a manutenção dos próprios testes.

Criação/Execução dos testes - Demonstrar de que forma podemos criar os vários testes, fazendo referência ao mecanismo de execução dos mesmos.

Conhecimentos técnicos - Uma das dificuldades no processo de automatização de testes tem sido os conhecimentos técnicos exigidos pelas várias ferramentas de execução existentes no mercado. A falta desses conhecimentos pode levar um *tester* a não se sentir confortável na criação e manutenção dos testes.

Data driven - Verificar a acessibilidade às fontes de dados externas, referindo quais são aquelas que a aplicação suporta.

Integrações (*builds*) - As exigências do mercado e dos clientes obriga a que o processo de disponibilização de novas versões e correções seja o mais célere possível. Para que isso seja possível o processo de testes automáticos deve estar alinhado com todas as alterações de código. As ferramentas de testes automáticos tem por isso que se enquadrar com os sistemas de *builds* utilizados, no caso do PRIMAVERA integrado com o TFS.

Análise de resultados - Perceber de que forma os resultados são apresentados após a execução (gráficos, tabelas, …), e demonstrar a facilidade de leitura dos resultados apresentados.

Estes parâmetros serão utilizados na análise individual de cada ferramenta.

Além destas foram estudadas outras ferramentas de automatização de testes a serviços, como por exemplo a *Robot Framework* e a *Selenium*. As razões pelas quais estas ferramentas não prosseguiram para a fase de comparação foi porque na análise inicial estas ferramentas não conseguiram responder a vários dos pontos que consideramos essenciais.

3.2 Ferramentas de testes de APIs

3.2.1 *ReadyAPI*

A ferramenta *ReadyAPI* é desenvolvida peça *SmartBear* Software, além de versão PRO contém também uma versão *open source.* Esta suporta testes funcionais, de carga, regressão e aceitação a serviços. Além disso suporta a maior parte dos protocolos e é capaz de modificar parâmetros intrínsecos dos mesmos, como por exemplo, *timeouts* ou tamanho dos *requests*.

O *ReadyAPI* é a versão mais recente do *SoapUI* especialmente direcionada para testes sobre uma plataforma API. As suas principais características são:

- Testes funcionais, testes de carga e testes stress (performance).
- Execução de diversos testes em paralelo.
- Capacidade de gerar um número ilimitado de pedidos para cada operação;
- Gerar múltiplos *endpoints* para cada *web-service*.
- O SUI permite testes funcionais, simulação de serviços, testes de segurança e testes de performance e suporta diversas tecnologias como *Soap/WSDL*, *Rest*, *web* e *http(s)* entre outras.
- No final dos testes permite ter relatórios integrados, relatórios personalizados, exportar e importar relatórios em (*PDF*, *Word*, *Excel*, *HTML*, *XML*) entre outros formatos.
- Para além de ter uma IDE própria o SUI permite utilizar outras IDEs como o *IntelliJ*, o *NetBeans* ou o *Eclipse*. (SoapUI - The Home of Functional Testing, s.d.).

As vantagens da versão profissional relativamente à versão gratuita são:

- Acesso direto a fontes de dados externos (*Excel*, *Database, XML, JSON…*).

- Permite gerar vários tipos de relatórios de resultados (*Html, Data Export, Project Report*).
- Permite o mapeamento dinâmico de valores, parâmetros, propriedades, *scripts*, variáveis entre os diversos *test suites* e *test cases* no mesmo projecto.
- Permite integração com diversos *plugins* externos (*GitHub, JIRA, Microsoft Azure API*…).
- Permite agendamento de eventos antes e depois das execuções.
- Melhorias no suporte técnico.

Arquitectura

Relativamente à arquitectura do *ReadyAPI* é uma arquitectura do tipo estrela, permitindo troca de informação com APIs, propriedades, ficheiros de configuração, entre outros, tal como representado na Figura 6.

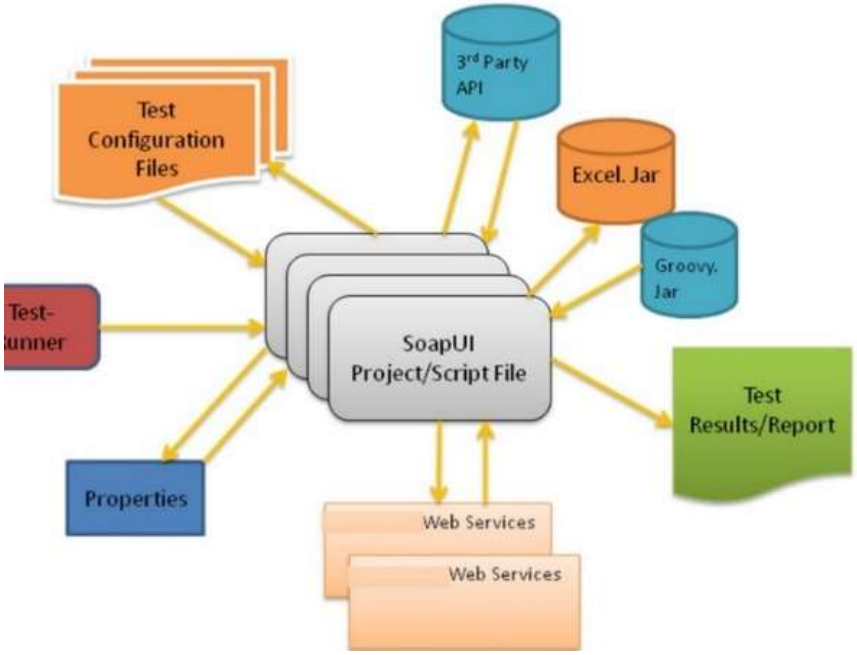


Figura 6 - Arquitectura ReadyAPI, imagem retirada de(Solution, 2013)

Licenças

O *ReadyAPI* tem dois tipos de licenças, *fixed* e *floating*. A diferença é que numa licença *fixed* apenas pode ser utilizada por um utilizador apenas numa máquina, enquanto na licença *floating* a licença poderá estar em várias máquinas, mas apenas um utilizador poderá utilizá-la. O preço

das licenças variam consoante o tipo de licenças e o número de anos que pretendemos adquirir, tal como representado na tabela 1 e tabela 2.

Duração	Preço
1 ano	**552,27 € (c/IVA)**
2 anos	**1.031,97 € (c/IVA)**
3 anos	**1.462,47 € (c/IVA)**

Tabela 1 - Preços licenças Fixed

Duração	Preço
1 ano	**3.811,77 € (c/IVA)**
2 anos	**7.194,27€ (c/IVA)**
3 anos	**10.256,97 € (c/IVA)**

Tabela 2 - Preços licenças Floating

Estrutura dos testes

O *ReadyAPI* permite organização do Projeto em *test suites*, *test cases* e *steps*. Podemos Executar cada bloco individualmente ou o conjunto seguindo uma ordem estabelecida.

Existe também a possibilidade de partilha de informação entre *test suites*, *test cases* e *steps*, aproveitando assim propriedades desenvolvidas anteriormente (Ex: variáveis globais), como representado na Figura 7.

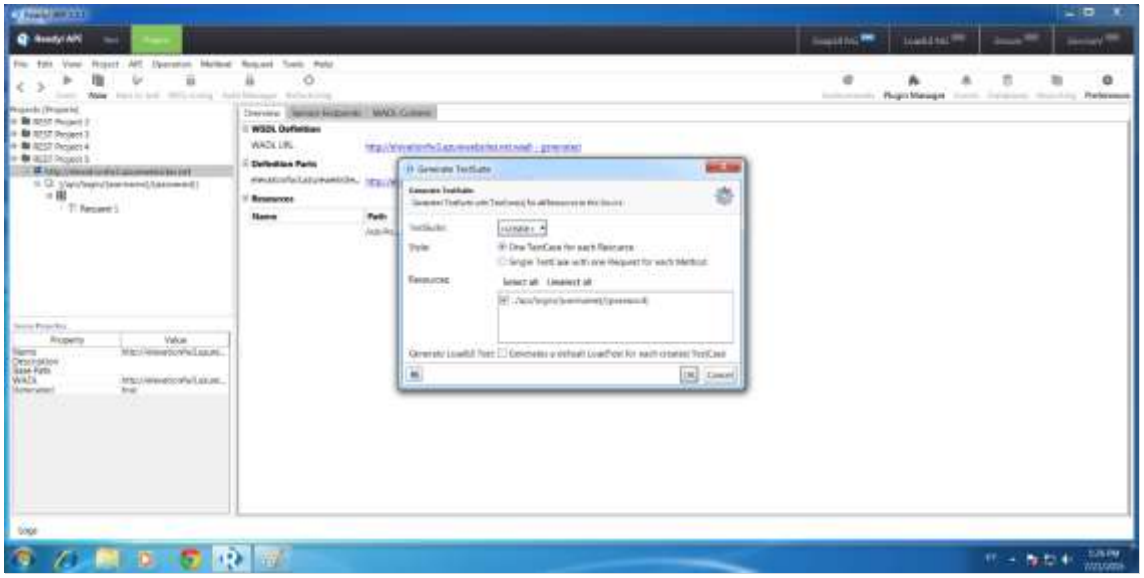


Figura 7 - Criar um Test Suite

Criação/Execução dos testes

Dentro de cada step podemos adicionar *datasource's*, *scripts* (em *Javacript* ou *grovy script*), *requests*, propriedades, entre outros.

Os testes são executados em sequência respeitando a ordem definida com a possibilidade de interrupção em caso de erro, ou execução completa, gerando relatório no final, como representado na Figura 8.

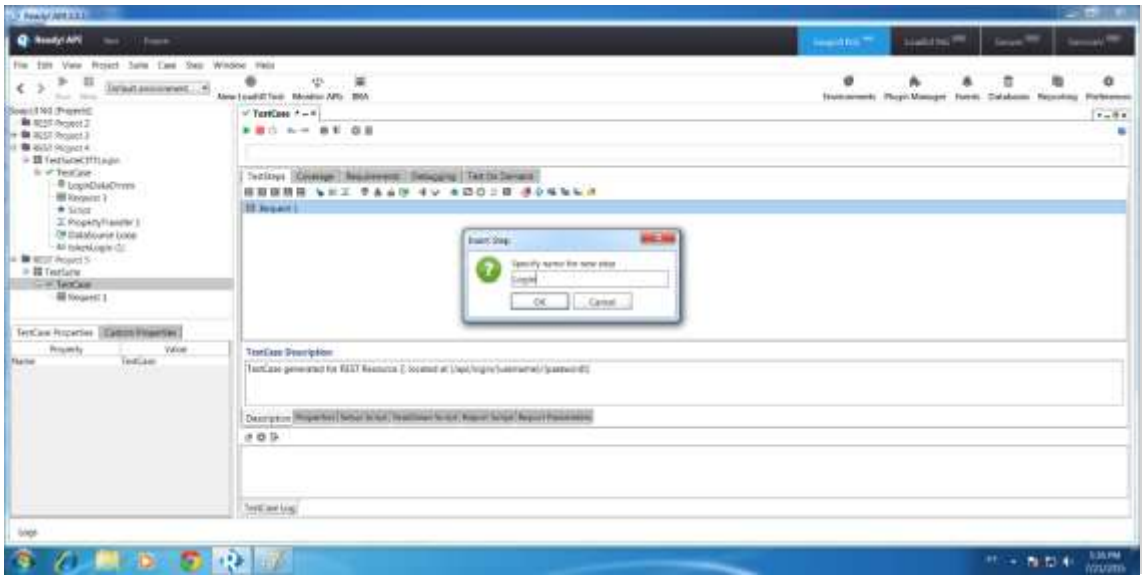


Figura 8 -Criar um novo Data Source

A execução é realizada pela opção *Launch TestRunner* através de um comando constituído pelos seguintes argumentos: Projecto, nome do projecto, nome *test case*, nome *test suite* e formato do relatório a gerar, como representado na Figura 9.

Figura 9 - Exemplo comando TestRunner

Conhecimentos técnicos

O *ReadyAPI* não necessita de um grande conhecimento técnico para criar, executar e analisar resultados, resultando de uma excelente ferramenta para os testers, possibilitando-os fazer manutenção e até criar novos *scripts* de teste sem terem a necessidade de grande conhecimento a nível de programação. É uma ferramenta que possibilita de forma bastante dinâmica adquirir propriedades, através da funcionalidade *Get Data*, como representado na Figura 10.

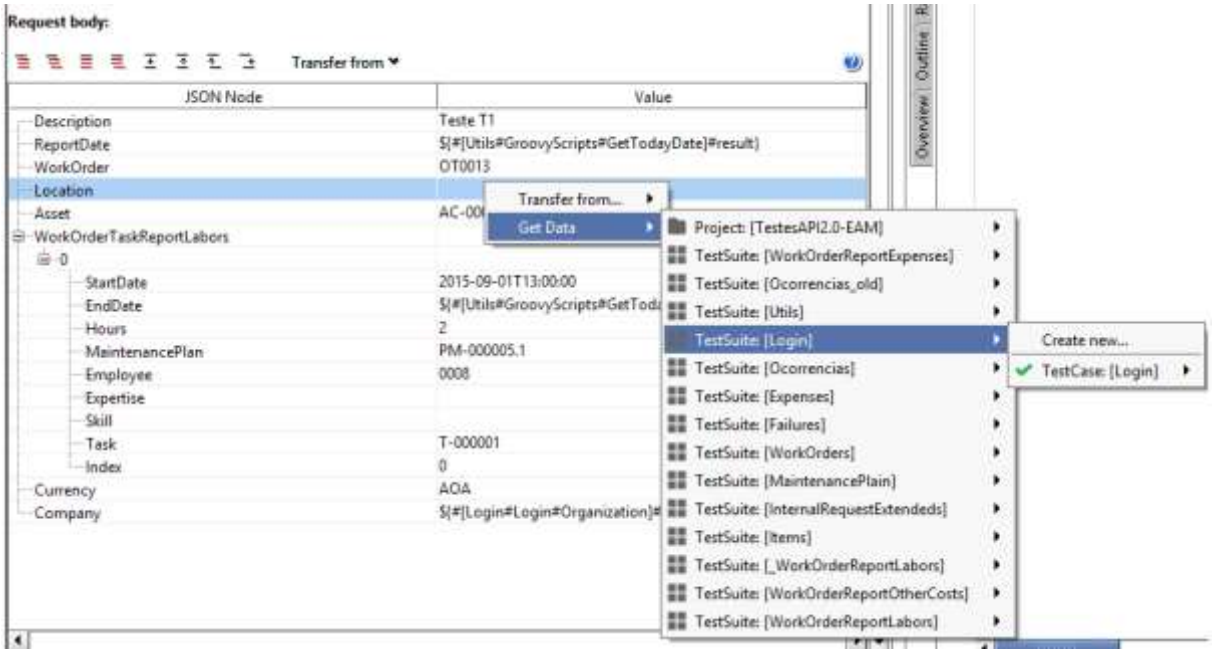


Figura 10 - Exemplo propriedade Get Data

Data driven

Como representado na Figura 11, para criar uma operação data driven (neste caso ligação a um excel), necessitamos de definir propriedades. As fontes de dados que a aplicação permite são: *Excel, database, json, xml, grid, directory, groovy, file e data generator*.

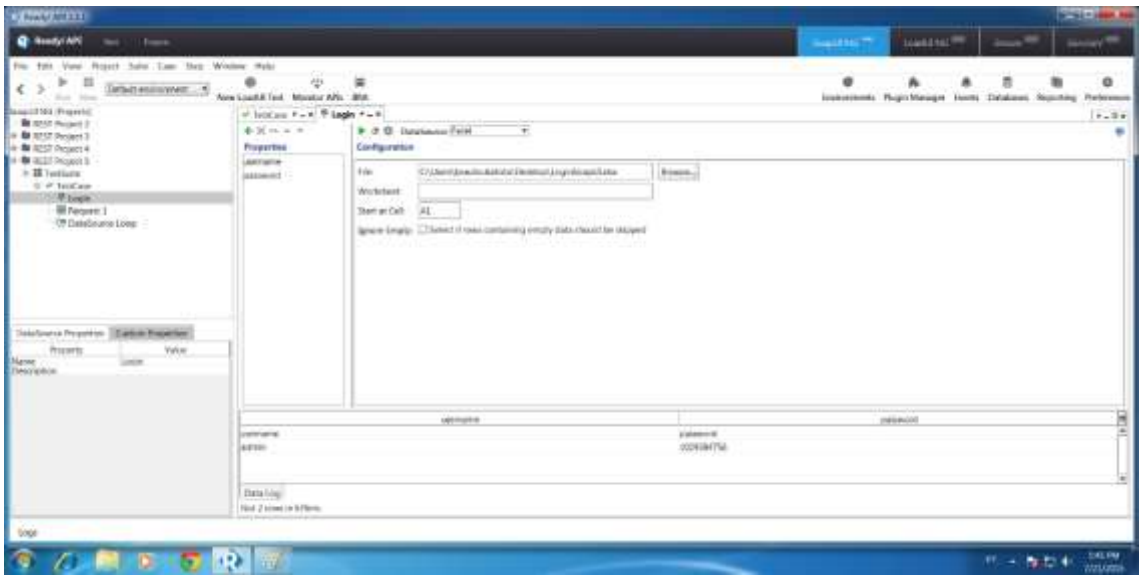


Figura 11 - Exemplo de Data Driven Excel

Integrações (builds)

Relativamente à integração com as builds, apesar de não permitir integrar diretamente com o sistema de builds, podemos desenvolver um ficheiro *.bat que faz esta integração.

Análise de resultados

Podemos ainda acompanhar o desenrolar dos testes e dos cenários durante as execuções e no final consultar os resultados. Os resultados são apresentado em forma de gráficos e tabelas, dando a possibilidade ao utilizador de aplicar filtros para construção de ambos, como representado na Figura 12.

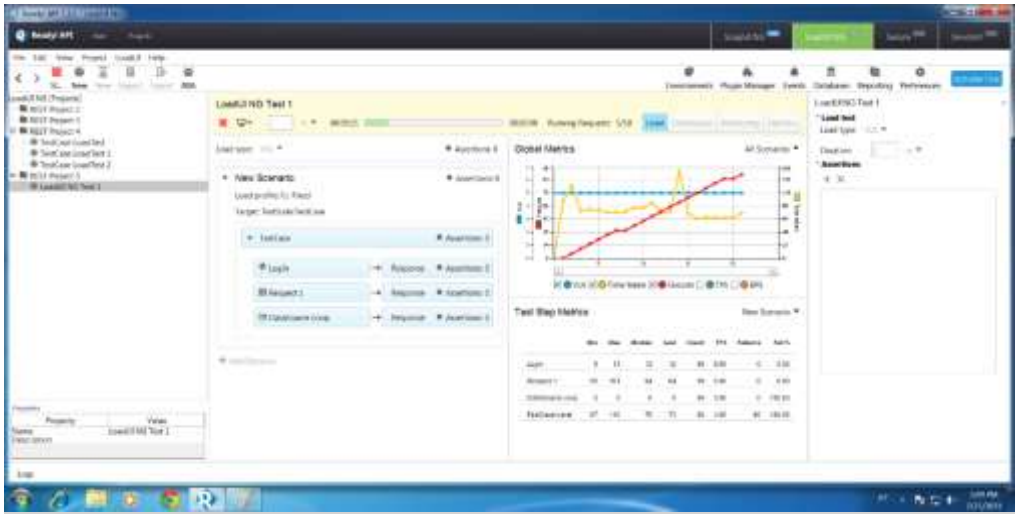


Figura 12 - Análise de resultados

3.2.2 Visual Studio – Test Perfomance

O Visual Test Performance é um projeto disponível no Visual Studio Ultimate 2013 que permite a simulação de pedidos a *web services*. A ferramenta permite fazer testes de performance (carga) e testes aos motores (API). Podemos simular *scripts* individuais ou criar *loading Tests* simulando várias script e vários utilizadores por minuto. (Microsoft, 2015)

As principais caraterísticas são:

- Criar *scripts* de teste individuais;
- Gerar código a partir dos *scripts*;
- Fazer *recording* dos cenários;
- Criar *loading tests* com vários *scripts*, permitindo alterar a carga dos diversos cenários tanto em tempo como em número de utilizador;
- Fácil integração com sistemas Microsoft;
- Integração com o TFS;
- Na análise gráfica dos resultados;
- O visual studio test perfomance não é um IDE de desenvolvimento. Necessita de um IDE instalado.

Licenças:

O preço das licenças do *Visual Studio* varia consoante o produto que se pretende, a licença mais cara é da versão *Ultimate* 2013, seguida da versão *Enterprise* 2015 e por último da versão *Test Professional* 2015, como representado na Tabela 3.

Ferramenta	Preço
Visual Studio Ultimate 2013	**8058 € (c/iva)**
Visual Studio Test Professional 2015	**2322,13€ (c/iva)**
Visual Studio Enterprise 2015	**6,551 € (c/iva)**

Tabela 3 - Preço licenças Visual Studio

Estrutura dos testes

O *visual studio test perfomance* permite organização do Projeto *web perfomance test, loading test* e testes unitários. Quando criamos um *web perfomance test* podemos adicionar *data sources* (*databases, CSV, XML*), podemos também adicionar transações e *requests*, como representado na Figura 13.

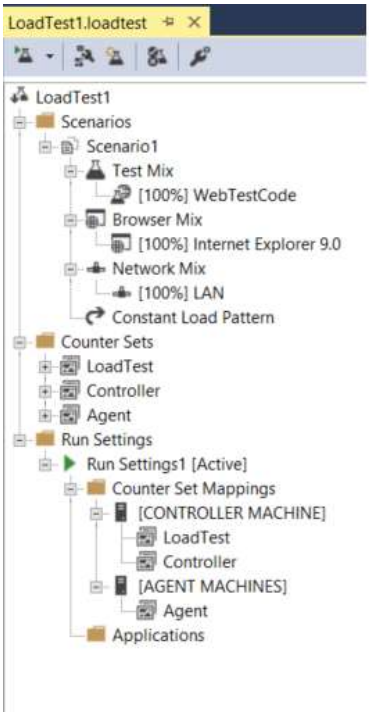


Figura 13 - Load Test Example

Criação/Execução dos testes

O visual studio permite criar, gerar script a partir de *recording*, ou importa-las de ferramentas externas. A funcionalidade de *recording* possibilita criar automaticamente a *script*, sendo possível posteriormente proceder à sua edição.

Quando criamos um *loading test* estamos a configurar o cenário de execução indicando os seguintes parâmetros: Tempo, número de utilizadores, *browser*, tipo de ligação de internet e distribuição dos pedidos.

Podemos adicionar vários *web plugins* de forma a adicionar mais componentes à ferramenta, como representado na Figura 14.

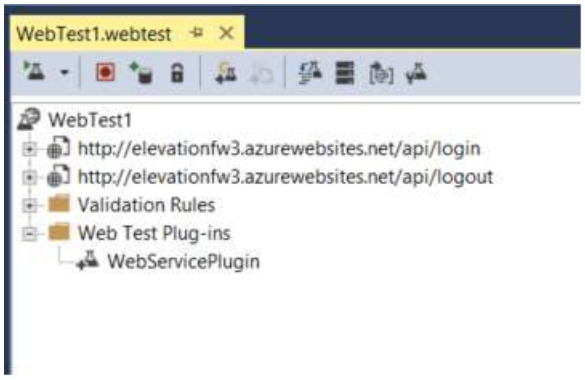


Figura 14 - Exemplo Web Test

Podemos ainda gerar código a partir dos *scripts* existentes, ou construir novas classes de raiz, como representado na Figura 15.

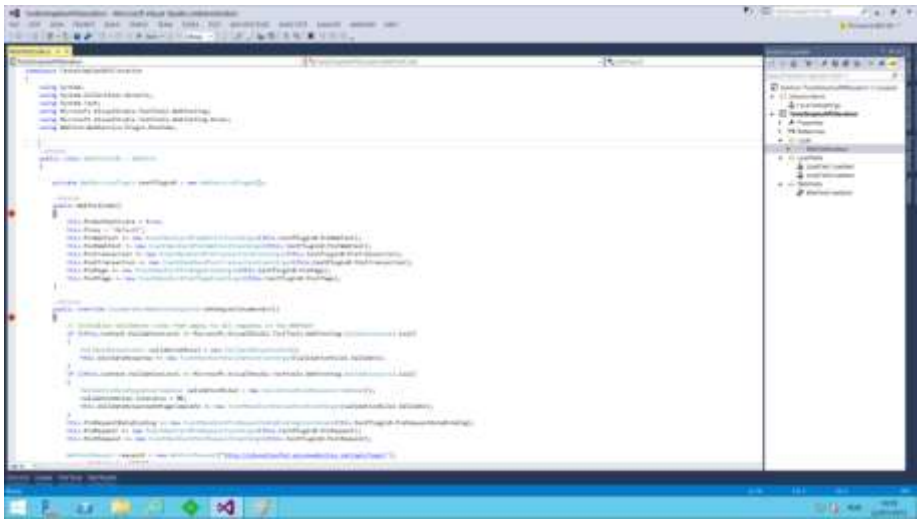


Figura 15 - Exemplo Web Test Code

.

Conhecimentos técnicos

O *visual studio test perfomance* necessita de conhecimento de desenvolvimento de software para criar, executar e analisar resultados.

Embora permita o *record and play* a própria manutenção deste código gerado obriga a ter *skills* de desenvolvimento.

Tendo a PRIMAVERA licença do *Visual Studio Ultimate 2013* e tendo já experiência interna na sua utilização optou-se por também a considerar nesta análise de ferramentas.

Data driven

O *visual studio* é das ferramentas mais completas no que diz respeito ao *data driven*. A aplicação permite conexões diretas a base de dados *SQL Server*, *Oracle*, *Access* e a partir de ligações *ODBC* qualquer tipo de base de dados (Ex:*MySQL*), além disso permite conexões a ficheiros (*CSV* e *XML*), como representado na Figura 16.

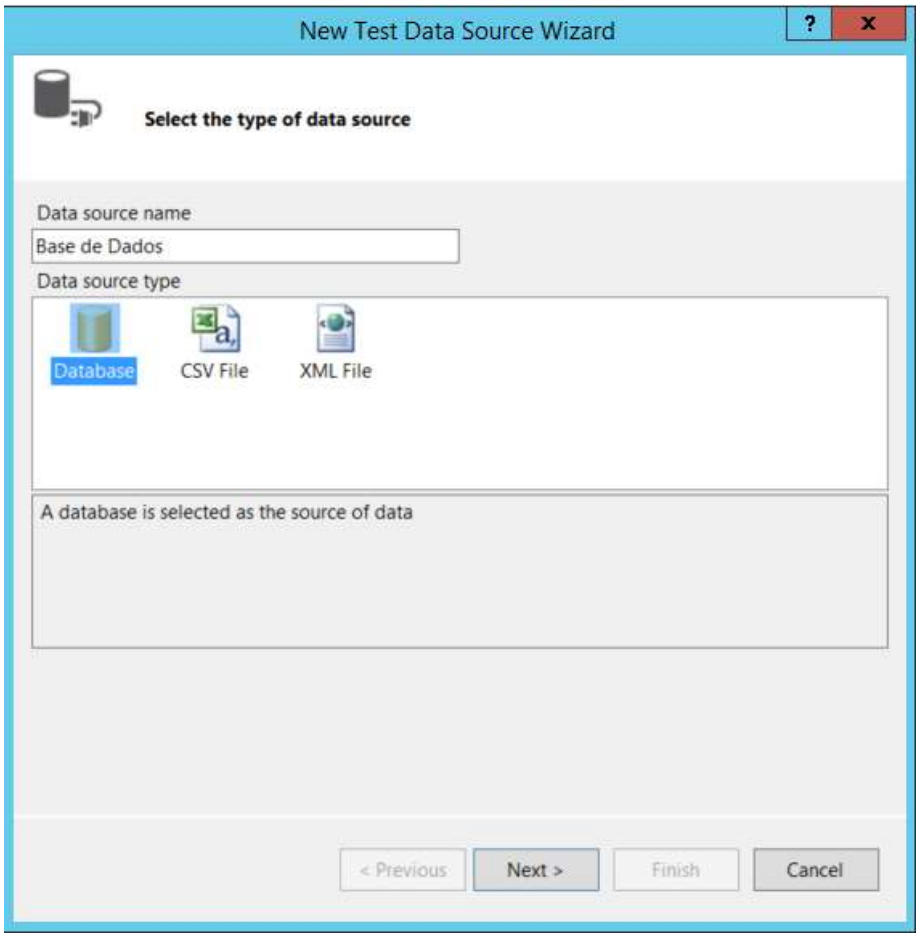


Figura 16 - Data Driven Visual Studio

Integrações (builds)

O *Visual Studio* permite conectar com o TFS (sistema que permite a gestão de código, requisitos e projectos) que a PRIMAVERA utiliza, sendo que este contém as definições e

gestão das de *builds*, logo permite conectar diretamente com o sistema de *builds*, como representado na Figura 17.

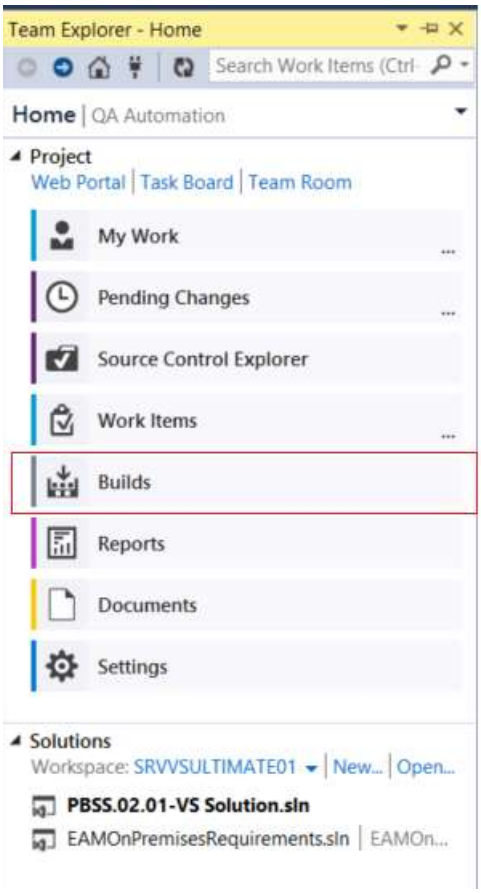


Figura 17 - Integração com builds TFS

Análise de resultados

Relativamente à análise de resultados, o *Visual Studio* regista numa base de dados própria todas as execuções dos *Loading Tests*, permite consultar o detalhe de cada execução (*Load Test*), visualizando assim a informação em tabelas ou gráficos, como representado na Figura 18 e na Figura 19.

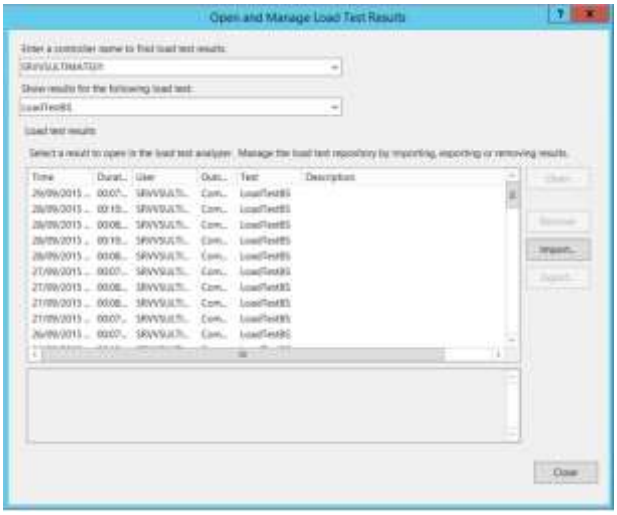


Figura 18 - Análise de Resultados Visual Studio

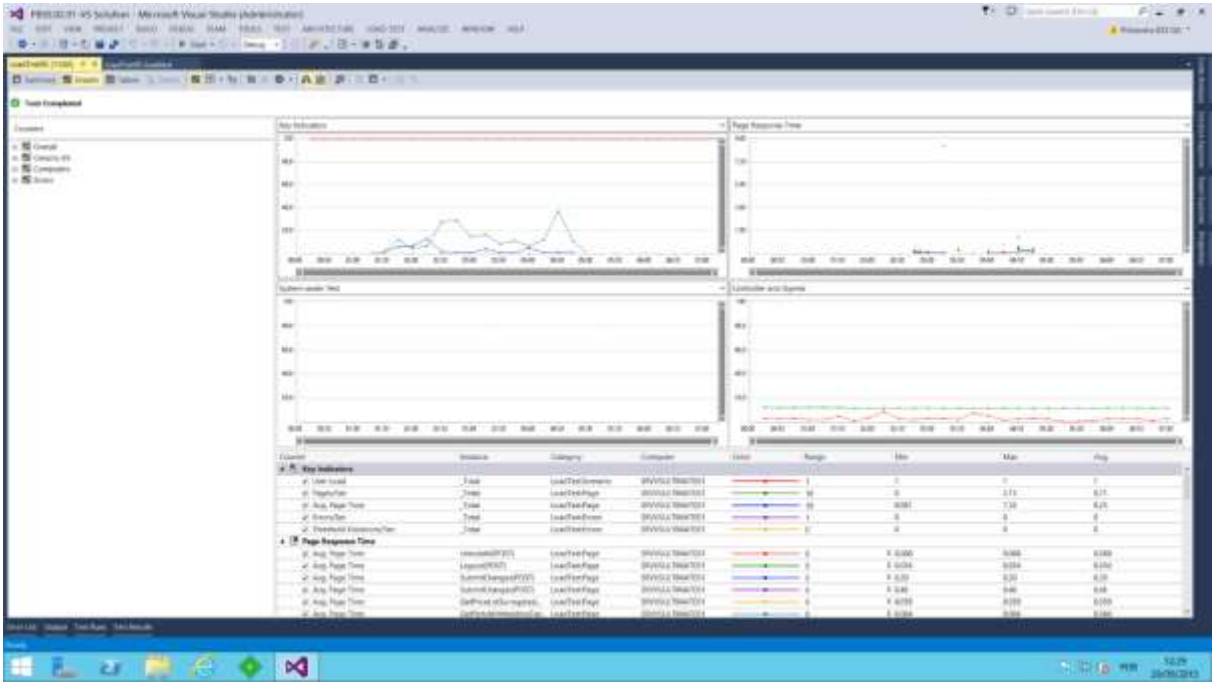


Figura 19 - Consulta detalhada Load Test

3.2.3 *Postman*

O *Postman* é uma aplicação que está disponível na *Web Store* da *Google* e é utilizada para testes de integração de aplicações simulando o envio e retorno de pedidos. O *Postman* contém uma versão *open source*, e uma pro a 9.99 dólares a licença. (Postman | Supercharge your API workflow, s.d.)

A diferença entre a versão paga e a versão gratuita é que a segunda não permite *Loading Test*, ou seja executar um conjunto de testes unitários simultaneamente, além disso não permite acesso a fontes de dados externas (*Data driven*).

Estrutura dos testes

O *Postman* apenas permite organizar os testes por lista de pedidos (*collections*), como representado na Figura 20.

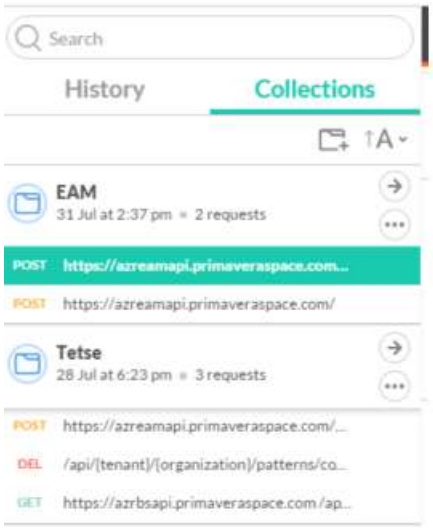


Figura 20 - Estrutura Testes Postman

Criação/Execução dos testes

É uma excelente ferramenta para desenvolver e executar testes unitários de uma forma rápida e intuitiva, como representado na Figura 21, Figura 22 e Figura 23.

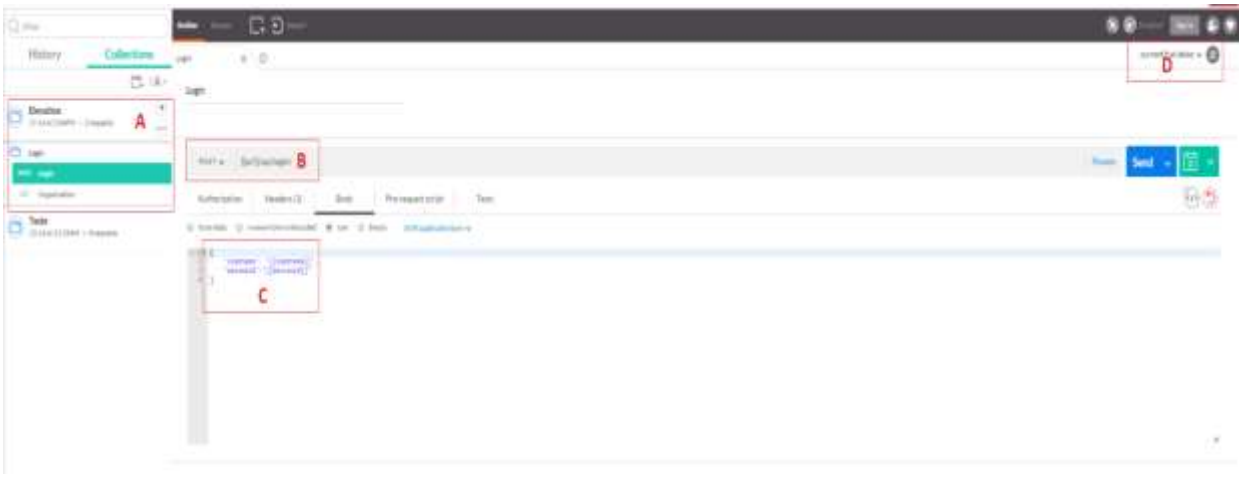


Figura 21 - Postman Layout

A – Temos a *collection* ”Login”, e todos os pedidos dentro da *collection* (*POST* e um *GET*);

B – Escolher o tipo de operação, neste caso *POST*, e escrevemos o *URL* (”{{*url*}}/*api*/*login*/”);

C – Parametrização do “*username*” e da “*password*”;

D – Definição das variáveis ({{*url*}},{{*username*}},{{*passwor*}})

Figura 22 - Gestão de variáveis

E – Exemplo de criação e gestão de variáveis;

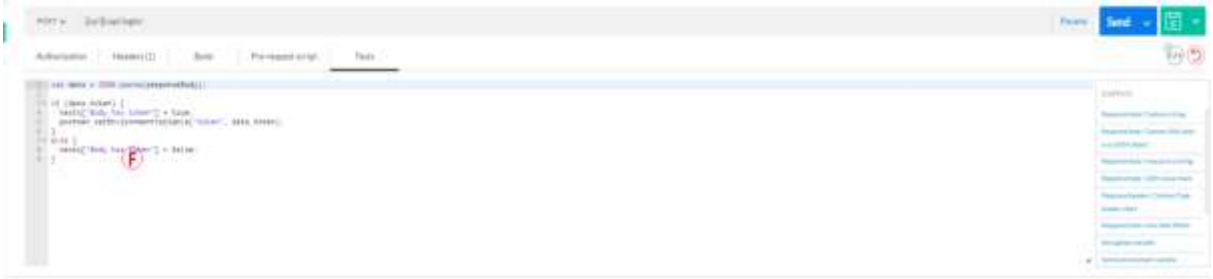


Figura 23 - Editor de Scripts

F – Exemplo do editor de *scripts*;

Conhecimentos técnicos

O *Postman* no que diz respeito à criação, execução e manutenção de testes, necessita que o utilizador apenas tenha conhecimento da API que está a testar, tornando-se uma excelente ferramenta para criar testes de uma forma rápida e intuitiva.

Data driven

Os dados de acesso do utilizador que vai fazer login (*username*, *password*) estão guardados num ficheiro *.CSV, que é previamente carregado, de forma a testar a funcionalidade de d*ata driven*. A grande desvantagem é que no *Load Test* apenas podemos adicionar um único ficheiro *.CSV com os dados, por exemplo se no nosso *Load Test* tivermos um teste para fazer login e outro para criar um cliente e ambos utilizarem *data driven*, as informações do login e do criar cliente terão de estar no mesmo *.CSV, como representado na Figura 24.

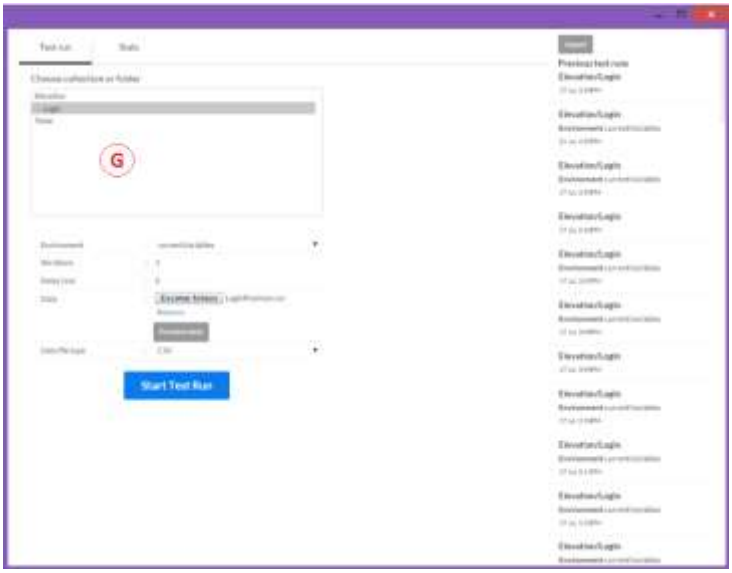


Figura 24 - Exemplo motor de execução

G – Configuração da execução;

Integrações (*builds*)

No *Postman* a integração com *builds* só é possível utilizando a linha de comandos *Newman*. O *Newman* é a linha de comandos desenvolvida para o *Postman*, esta permite integrar as *collections* do *Postman* com o sistema de *builds*.

Análise de resultados

Relativamente à análise de resultados o *Postman* apenas informa se os testes passaram ou falharam, este não cria um histórico com esses dados para serem analisados posteriormente com a utilização de gráficos ou tabelas.

3.3 Conclusão

Após a análise detalhada das três ferramentas *Postman*, *ReadAPI* e o *Visual Studio Test Performance*, com base nos seguintes requisitos: Estrutura dos testes, criação e execução dos testes, conhecimentos técnicos, *data driven*, integrações (*builds*) e análise de resultados. Como resultado, dentro do contexto das aplicações PRIMAVERA e das ferramentas analisadas concluo que o *ReadyAPI* é a melhor solução para testar a API.

O *ReadyAPI* tem um interface interativo e de fácil aprendizagem o que permite a qualquer pessoa sem conhecimentos de desenvolvimento de software aprofundados construir testes sobre APIs. Além disso permite um fácil acesso a fontes de dados externas como por exemplo (*Excel*, *XML* e base de dados). Podemos construir várias validações e comparações em *Javacript* e *Groovy Script*. Não permite integrar diretamente com o *TFS*, mas permite executar os projetos a partir de ficheiros *.bat ou de uma solução criada no *Visual Studio*.

4 Caso de demonstração na PRIMAVERA Software

4.1 Introdução

Neste capítulo será apresentado o caso de demonstração na PRIMAVERA Software. No seguimento do capítulo anterior, a empresa decidiu escolher a tecnologia *ReadyAPI* para utilizar na automatização dos vários casos de teste definidos pela equipa responsável do produto PRIMAVERA designado por *Enterprise Asset Management* (EAM), produto este que é aplicado à área de manutenção de equipamentos.

Para proceder ao desenvolvimento do caso de demonstração, em primeiro lugar foi necessário desenhar um plano de teste, de seguida criar uma *framework* que contempla o ambiente de execução, o ambiente de desenvolvimento, bem como o projecto do *ReadAPI*. Por fim será descrito todo o desenvolvimento relativo ao projecto do *ReadyAPI* desde o processo de automatização dos testes resultantes do plano de teste, bem como o processo de execução e os resultados obtidos.

4.2 Metodologia de Desenvolvimento

A metodologia de desenvolvimento utilizada no desenvolvimento deste caso de demonstração foi a metodologia *Scrum*. Esta é uma metodologia ágil utilizada para gerir e planear projectos de software. Os projectos são divididos em ciclos onde são designados por "*Sprints*". Cada *sprint* é constituída por um conjunto de tarefas normalmente relacionadas com uma *User Story* que estão relacionadas com as tarefas definidas na lista designada como "*Product Backlog*". No início de casa *sprint* é realizado uma cerimónia designada por "*Sprint Planning*", onde o responsável pelo produto "*Product Owner*" prioriza as tarefas do *product backlog* e a respectiva equipa seleciona as actividades que será capaz de desenvolver durante o *sprint* que se segue.

Durante o decorrer da *sprint*, as equipas fazem uma pequena reunião diária, designada por "*Daily Scrum*", esta cerimónia tem como principal objectivo disseminar o conhecimento do que foi feito no dia anterior, identificar os problemas e definir o trabalho do dia que se inicia.

No final de cada *sprint* é realizado uma "*Sprint Review"*, onde a equipa apresenta as funcionalidades implementadas. Por último é realizado uma "*Sprint Retrospective*", onde a

equipa irá inspecionar e criar um plano de melhorias para implementar durante o *sprint* seguinte.(ken Schwaber, Jeff Sutherland, 2014)

Relativamente a este caso de demonstração foram necessários quatro *sprints* com duração de duas semanas cada uma. Cada sprint tem associado 56 horas de trabalho, sendo que o total deste caso de demonstração são 224 horas de trabalho.

Seguindo os princípios do *SCRUM* foi necessário a criação de *User Stories*, ver Figura 25, que podem ser vistas com objectivos, associado a ela temos as tarefas, ver Figura 26.

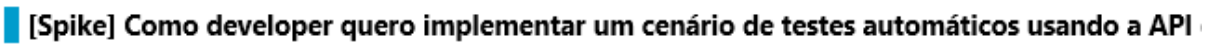


Figura 25 - Exemplo User Story Caso demonstração

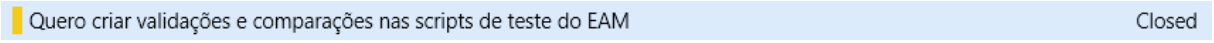


Figura 26 - Exemplo Tarefa Caso demonstração

Todas as tarefas foram realizadas e entregues no final de cada *sprint* respectiva, logo todas as *User Stories* ficaram fechadas, sendo um factor essencial para não atrasar a conclusão deste caso de demonstração.

4.3 Plano de teste

Para a aplicação EAM foram criados três planos de teste, cada plano é referente a uma funcionalidade do produto, neste caso surgiram três planos de teste: Registo de consumos, registo de despesas e registo de mão-de-obra.

A criação dos vários planos de teste tiveram a contribuição da equipa do produto *EAM*, foram eles que definiram quais as funcionalidades prioritárias para aplicação de testes automáticos.

Por motivos de confidencialidade, apenas será apresentado nesta dissertação o plano de teste relativo ao registo de despesa. Este plano foca-se na área de câmbio do produto EAM. Os cenários aplicados a este plano de teste são registar despesas relativas a três ordens de trabalho

(ver Tabela 4). Além destas ordens de trabalho, o cenário também vai variando com utilizadores (ver Tabela 5) que fazem login para proceder ao registo das várias despesas.

Ordem de Trabalho	Tipo de OT	CT	Responsável
OT-0001	T1	CT02	TEC2
OT-0002	T2	CT02	TEC2
OT-0003	T3	CT02	TEC2

Tabela 4 - Ordens de trabalho

Funcionário	Especialidade
TEC1	PINTOR; ELETRICISTA.
TEC2	MECÂNICO.
TEC3	ELETRICISTA,PICHELEIRO.
TEC4	ELETRICISTA.
TEC5	MECÂNICO.
TEC6	ELETRICISTA.
TEC7	MECÂNICO.
TEC8	ELETRICISTA.

Tabela 5 - Funcionários

4.3.1 Cenário 1: Registo de Despesas na OT-0003

No cenário 1 vamos ter registos de despesas na OT-0003, com login do funcionário TEC1, TEC2, TEC3, TEC4 e TEC7. Neste cenário vamos distinguir o resultado do teste como positivo ou negativo. No teste positivo espera-se que aplicação responda positivamente ao pedido (*request)* retornando uma validação *http* (201), verificando igualmente se o campo *ExceptionMessage* existe. No teste negativo espera-se que aplicação responda negativamente ao pedido (*request)* retornando uma validação *http* (400), verificando a mensagem esperada no campo *ExceptionMessage*.

Login como TEC1

Na Tabela 6 estão descritas 7 *scripts* de teste realizadas com o utilizador TEC1.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	201	null
T5	T-0003	TEC6	201	null
T6	T-0003	All	201	null
T7	T-0004	TEC1	400	The employee must belong to the task's work center. (Line #1).

Tabela 6 - Cenário 1 - OT0003, login TEC1

Login como TEC2

Na Tabela 7 estão descritas 8 *scripts* de teste realizadas com o utilizador TEC2.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	201	null
T5	T-0003	TEC6	201	null
T6	T-0004	TEC1	Negativo	The employee must belong to the task's work center. (Line #1).
T7	T-0001	TEC6	Negativo	The employee must belong to the task's work center. (Line #1).
T8	T-0001	TEC7	201	null

Tabela 7 - Cenário 1 - OT0003, login TEC2

Login como TEC3

Na Tabela 8 estão descritas 8 *scripts* de teste realizadas com o utilizador TEC3.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null

T4	T-0002	TEC3	201	null
T5	T-0003	TEC6	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T6	T-0003		400	The Employee field is required. (Line #1).
T7	T-0004	TEC1	400	The employee must belong to the task's work center. (Line #1).”
T8	T-0004		400	The Employee field is required. (Line #1).

Tabela 8 - Cenário 1 - OT0003, login TEC3

Login como TEC4

Na Tabela 9 estão descritas 6 *scripts* de teste realizadas com o utilizador TEC4.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	400	The employee must belong to the task's work center. (Line #1).
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0002	TEC3	400	The specified task is not associated with the current maintenance plan. (Line #1).
T4	T-0004	TEC1	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T5	T-0004		400	The Employee field is required. (Line #1).
T6	T-0001	All	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).

Tabela 9 - Cenário 1 - OT0003, login TEC4

Login como TEC7

Na Tabela 10 estão descritas 5 *scripts* de teste realizadas com o utilizador TEC7.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC7	201	null
T2	T-0001	TEC7	201	null
T3	T-0001	TEC4	400	The employee must belong to the task's work center. (Line #1).
T4	T-0002	TEC7	400	The specified task is not associated with the current maintenance plan. (Line #1).
T5	T-0001	All	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).

Tabela 10 - Cenário 1 - OT0003, login TEC7

4.3.2 Cenário 2: Registo de Despesas na OT-0002

No cenário 2 vamos ter registos de despesas na OT-0002, com login do funcionário TEC1, TEC2, TEC3, TEC7 e TEC8. Tal como no cenário 1 vamos distinguir o resultado do teste como positivo ou negativo. No teste positivo espera-se que aplicação responda positivamente ao pedido (*request)* retornando uma validação *http* (201), verificando igualmente se o campo *ExceptionMessage* existe. No teste negativo espera-se que aplicação responda negativamente ao pedido (*request)* retornando uma validação *http* (400), verificando a mensagem esperada no campo *ExceptionMessage*.

Login como TEC1

Na Tabela 11 estão descritas 7 *scripts* de teste realizadas com o utilizador TEC1.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	201	null
T5	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).
T6	T-0003	TEC5	201	null
T7	T-0004	TEC1	201	null

Tabela 11 - Cenário 2 - OT0002, login TEC1

Login como TEC2

Na Tabela 12 estão descritas 6 *scripts* de teste realizadas com o utilizador TEC2.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	201	null

T5	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).
T6	T-0004	TEC1	201	null

Tabela 12 - Cenário 2 - OT0002, login TEC2

Login como TEC3

Na Tabela 13 estão descritas 8 *scripts* de teste realizadas com o utilizador TEC3

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	400	The employee must belong to the task's work center. (Line #1).
T5	T-0002		400	The Employee field is required. (Line #1).
T6	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).
T7	T-0004	TEC1	201	null
T8	T-0004		400	The Employee field is required. (Line #1).

Tabela 13 - Cenário 2 - OT0002, login TEC3

.

Login como TEC7

Na Tabela 14 estão descritas 4 *scripts* de teste realizadas com o utilizador TEC7

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC7	201	null
T2	T-0001	TEC4	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T3	T-0002	TEC7	400	The employee must belong to the task's work center. (Line #1).
T4	T-0001		400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).

Tabela 14 - Cenário 2 - OT0002, login TEC7

Login como TEC8

Na Tabela 14 estão descritas 11 *scripts* de teste realizadas com o utilizador TEC8

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).

T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC8	201	null
T4	T-0001		400	The Employee field is required. (Line #1).
T5	T-0002	TEC3	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T6	T-0002		400	The Employee field is required. (Line #1).
T7	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).
T8	T-0003		400	The Employee field is required. (Line #1).
T9	T-0004	TEC1	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T10	T-0004	TEC8	400	The employee must belong to the task's work center. (Line #1).
T11	T-0004		400	The Employee field is required. (Line #1).

Tabela 15 - Cenário 2 - OT0002, login TEC8

4.3.3 Cenário 3: Registo de Despesas na OT-0001

No cenário 3 vamos ter registos de despesas na OT-0001, com login do funcionário TEC1, TEC2, TEC3 e TEC4. Tal como nos outros cenários vamos distinguir o resultado do teste como positivo ou negativo. No teste positivo espera-se que aplicação responda positivamente ao pedido (*request)* retornando uma validação *http* (201), verificando igualmente se o campo *ExceptionMessage* existe. No teste negativo espera-se que aplicação responda negativamente ao pedido (*request)* retornando uma validação *http* (400), verificando a mensagem esperada no campo *ExceptionMessage*.

Login como TEC1

Na Tabela 16 estão descritas 10 *scripts* de teste realizadas com o utilizador TEC1.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0001	TEC4	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T5	T-0002	TEC3	201	null
T6	T-0002	TEC5	400	The employee must belong to the task's work center. (Line #1).
T7	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).

T8	T-0003	TEC5	201	null
T9	T-0004	TEC1	400	The employee must belong to the task's work center. (Line #1).
T10	T-0005	TEC4	201	null

Tabela 16 - Cenário 3 - OT0001, login TEC1

Login como TEC2

Na Tabela 17 estão descritas 10 *scripts* de teste realizadas com o utilizador TEC2.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC2	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	201	null
T5	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).
T6	T-0003	TEC5	201	null
T7	T-0004	TEC1	400	The employee must belong to the task's work center. (Line #1).
T8	T-0001	TEC6	400	The employee must belong to the task's work center. (Line #1).
T9	T-0001	TEC7	400	The employee must belong to the task's work center. (Line #1).

T10	T-0005	TEC4	201	null

Tabela 17 - Cenário 3 - OT0001, login TEC2

Login como TEC3

Na Tabela 18 estão descritas 9 *scripts* de teste realizadas com o utilizador TEC3.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	201	null
T2	T-0001	TEC4	400	The employee must belong to the task's work center. (Line #1).
T3	T-0001	TEC3	201	null
T4	T-0002	TEC3	201	null
T5	T-0002	TEC5	400	The employee must belong to the task's work center. (Line #1).
T6	T-0003	TEC6	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T7	T-0003		400	The Employee field is required. (Line #1).
T8	T-0004	TEC1	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).

T9	T-0004		400	The Employee field is required. (Line #1).

Tabela 18 - Cenário 3 - OT0001, login TEC3

Login como TEC4

Na Tabela 19 estão descritas 11 *scripts* de teste realizadas com o utilizador TEC4.

Nºteste	Tarefa	Funcionário	HTTPCode	ExceptionMessage
T1	T-0001	TEC1	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T2	T-0001	TEC2	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T3	T-0001	TEC4	400	The employee must belong to the task's work center. (Line #1).
T4	T-0002	TEC3	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T5	T-0003	TEC6	400	The employee must belong to the task's work center. (Line #1).

T6	T-0003		400	The Employee field is required. (Line #1).
T7	T-0004	TEC1	400	Only the person in charge of the work order or associated work center is able to report on behalf of an employee other than himself. (Line #1).
T8	T-0004		400	The Employee field is required. (Line #1).
T9	T-0001		400	The Employee field is required. (Line #1).
T10	T-0005		400	The Employee field is required. (Line #1).
T11	T-0005	TEC4	201	null

Tabela 19 - Cenário 3 - OT0001, login TEC4

4.4 Framework

4.4.1 Ambiente de Execução

Relativamente ao ambiente de execução, esta utiliza uma *cloud* privada tendo associada uma tarefa no *windows scheduler* que liga e configura uma máquina virtual através da aplicação Vsphere. O processo de configuração trata de instalar as últimas *builds* da API e copia os ficheiros necessários para a preparação do ambiente de execução. Após a máquina estar configurada, esta executa um ficheiro *.*bat* iniciando a execução do projecto com os *scripts* de teste, estes *scripts* estão alojados na máquina virtual que contém o ambiente de desenvolvimento, tal como representado na Figura 27 - Arquitectura da Framework. Por último a máquina de execução é desligada voltando ao seu estado inicial.

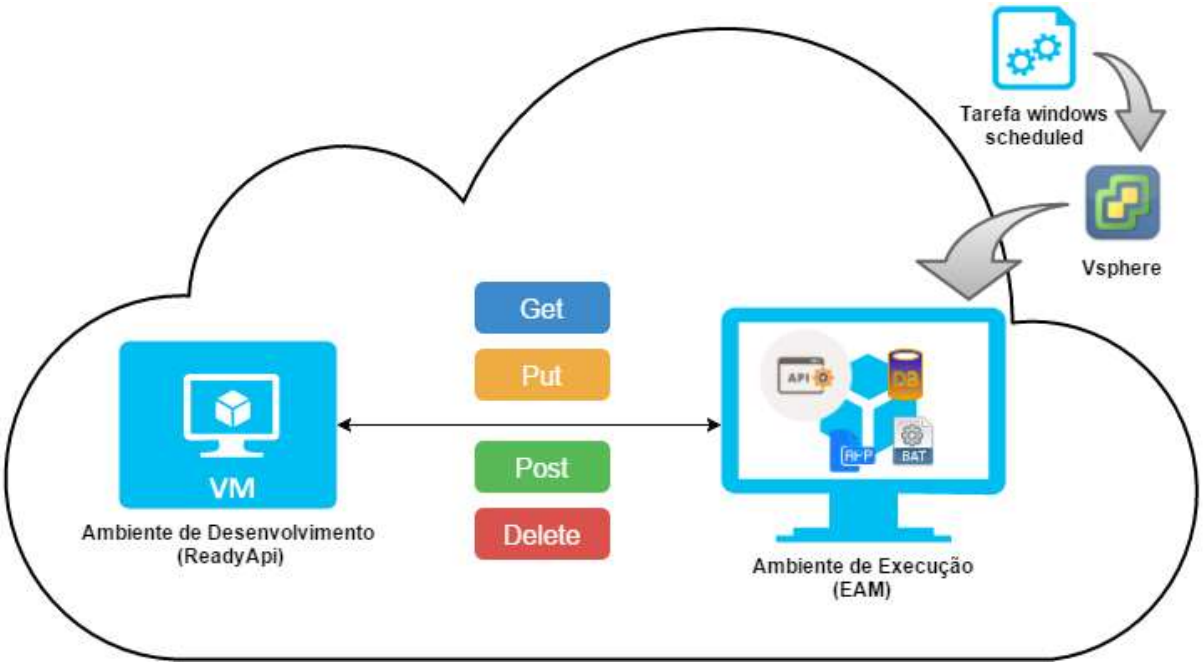


Figura 27 - Arquitectura da Framework

4.4.2 Ambiente de desenvolvimento

Neste ambiente de desenvolvimento é onde se encontra o projecto do *ReadyAPI*, que está detalhado no ponto 4.4.3. A criação deste ambiente de desenvolvimento tem como principal objectivo simplificar a manutenção futura e a adição de novos casos de teste. Através da utilização da metodologia *data-driven*, da criação de variáveis globais, da parametrização de pedidos *http* e recorrendo a um projecto comum criado com o objectivo de ser uma biblioteca de dados e funcionalidades, tal como representado na Figura 28. Posto isto a manutenção e inserção de novos *scripts* poderá ser um processo tão simples como inserir ou alterar os dados no ficheiro *datasource*.

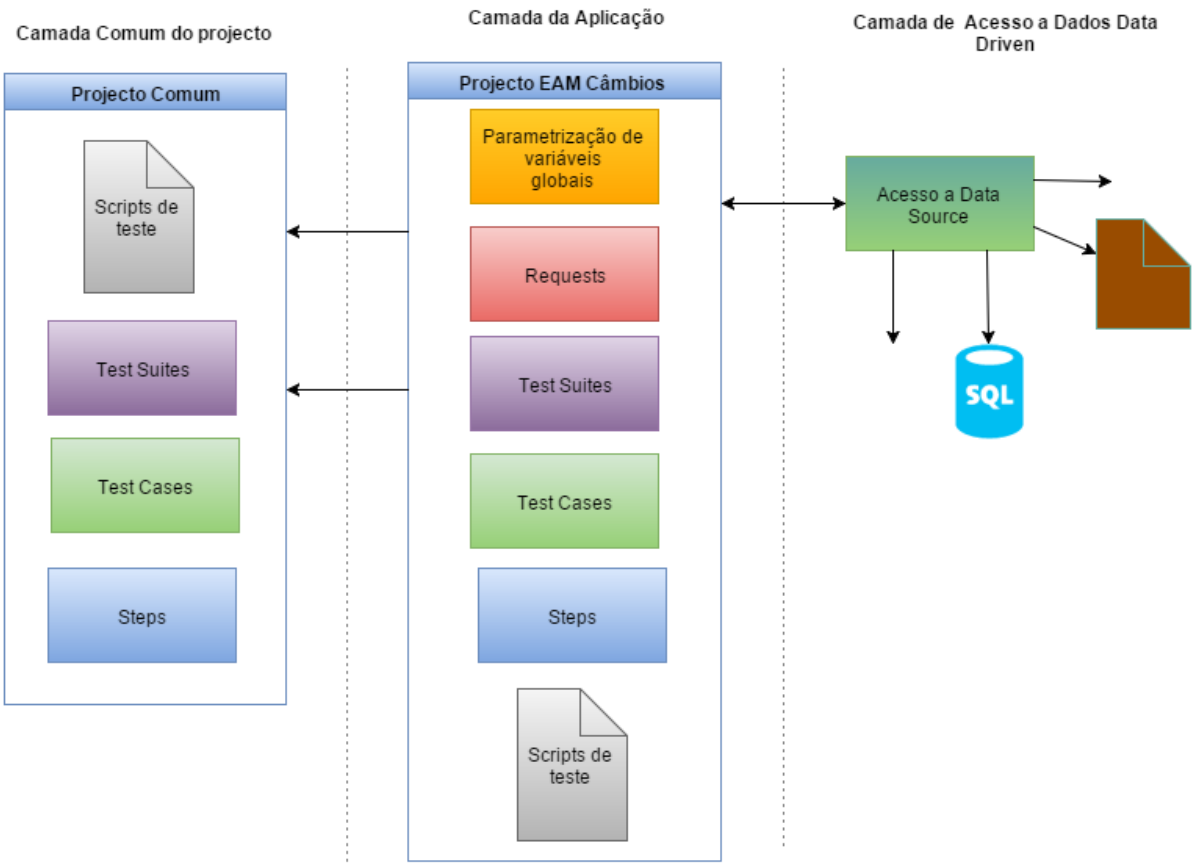


Figura 28 - Arquitectura Projecto do ReadyAPI

4.4.3 Projecto do ReadyAPI

O projecto do *ReadyAPI* está inserido no ambiente de desenvolvimento e é constituído pelo projecto comum e pelo projecto *EAM* câmbios.

Projecto Comum

Relativamente ao desenvolvimento comecei por criar um projecto comum na ferramenta *ReadyAPI* designado de "TestesAPI2.0-Common". Este projecto tem como principais objectivos partilhar funcionalidades comuns entre vários projectos de teste que existam ou que possam surgir no futuro, evitando a duplicação de funcionalidades iguais, tal como representado na Figura 29. O objectivo futuro desta framework é poder evoluir relativamente ao número de funcionalidades comuns, actualmente partilhamos as funcionalidades comuns como por exemplo a criação de datas, valores aleatórios e chaves únicas. Estes *scripts* foram desenvolvidos na linguagem *GroovyScript*, na Figura 30 é apresentada a função criada para

retornar valores dinâmicos desenvolvida em *GroovyScript* inserida no projecto comum e partilhada com o projecto *EAM* câmbios.

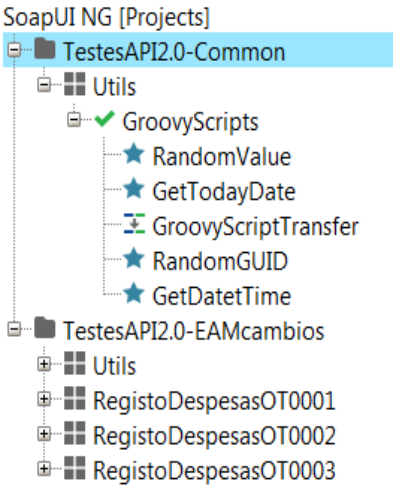


Figura 29 – Organização do Projecto comum

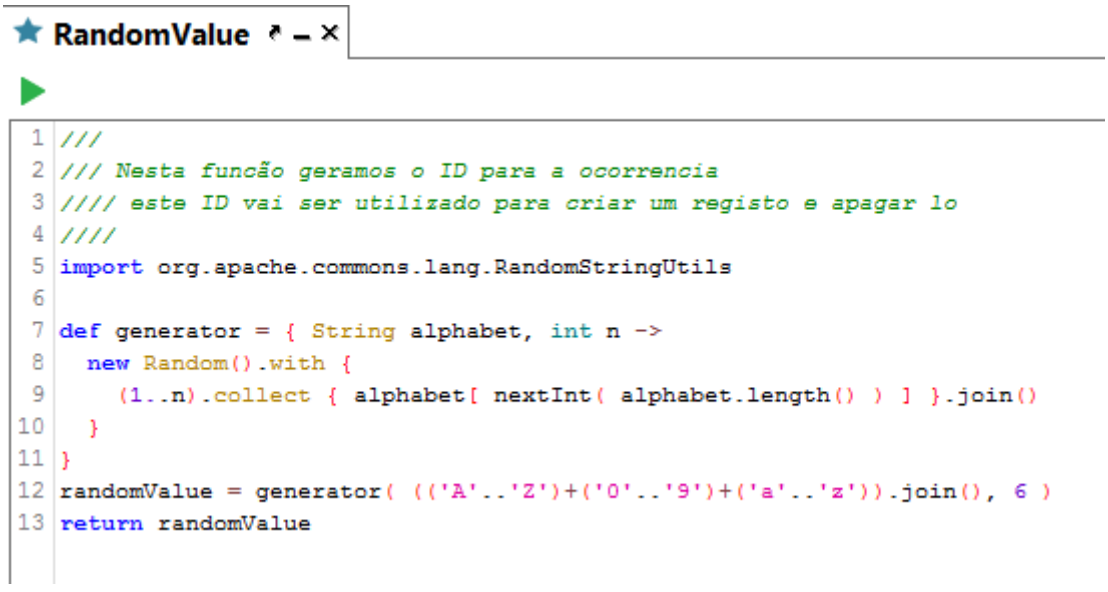


Figura 30 - Exemplo GroovyScript função Random Value

Projecto EAM Câmbios

Relativamente ao projecto "TestesAPI2.0-EAMcambios" no *ReadyAPI*, este projecto é constituído pelos *scripts* referidos no plano de teste (ver ponto 4.3). Este é constituído por dois *TestSuites*: *"Utils"* e "RegistoDespesasOT0001", como representado na Figura 31.

O *TestSuite "Utils"* do projecto *EAM* câmbios é utilizado para efeitos de configuração do projecto. Este comunica com o as funcionalidades do projecto comum, como tal foi necessário criar uma classe dentro do projecto câmbios, que executa o projecto comum obtendo assim os valores pretendidos, a Figura 32 demonstra um exemplo de acesso à classe "*RandomValue*" do projecto comum, a partir do projecto câmbios obtendo o valor pretendido.

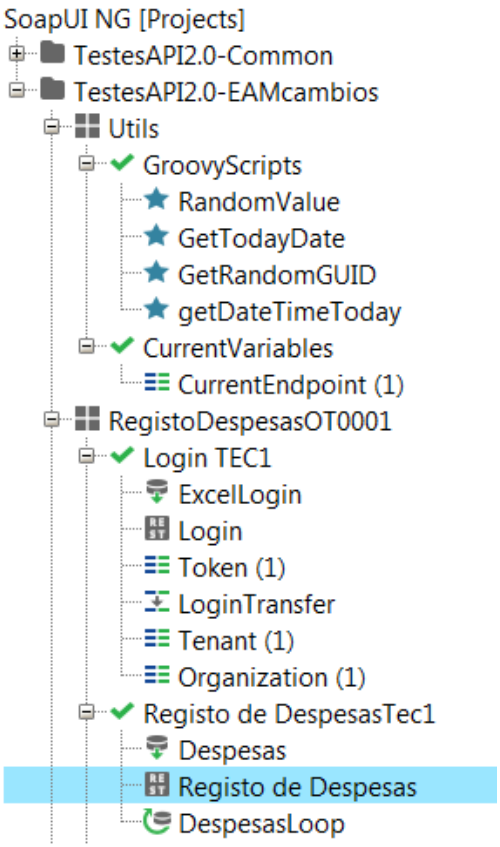


Figura 31 - Estrutura do projecto EAM Câmbios

RandomValue

```
import com.eviware.soapui.model.project.ProjectFactoryRegistry;
import com.eviware.soapui.impl.wsdl.WsdlProjectFactory;
import com.eviware.soapui.support.types.StringToStringMap;
def groovyUtils = new com.eviware.soapui.support.GroovyUtils(context);

print '### Script Workspace Setup ###'
println()
//Indicar qual o projecto que quero criar
log.info(context)
log.info(groovyUtils)
def targetDir = new File(groovyUtils.projectPath).parent

def targetProjectFile = targetDir+'\\'+'Testes-API-2-0---Common-soapui-project.xml'// criar o projecto
def targetWorkspace = targetDir+'\\'+'Testes-API-2-0---Common-soapui-workspace.xml' //criar o workspace
def targetProjectWorkspaceName = 'TestesAPI2.0-Common' // nome do projecto
def workspace = testRunner.testCase.testSuite.project.workspace
if(workspace==null)
{
workspace = new WorkspaceImpl(targetWorkspace, new StringToStringMap());
}
def isProjectInWorkspace = workspace.getProjectByName(targetProjectWorkspaceName)
if (isProjectInWorkspace == null){
   def refProject = workspace.importProject(targetProjectFile.toString())
}

//abrir o workspace
def testProject = (workspace==null) ?
ProjectFactoryRegistry.getProjectFactory(WsdlProjectFactory.WSDL_TYPE).createNew(targetDir+".xml") :
workspace.getProjectByName(targetProjectWorkspaceName)
if(!testProject.open && workspace!=null) {workspace.openProject(testProject)}

print '### End Workspace Setup Script ###'
println()
//test case and Test Suite names
def TargetTestSuiteName = "Utils";
def TargetTestCaseName = "GroovyScripts";

//get test case from other project or from the same one
project = workspace.getProjectByName(targetProjectWorkspaceName);
log.info(project)
testSuite = project.getTestSuiteByName(TargetTestSuiteName);
log.info(testSuite)
testCase = testSuite.getTestCaseByName(TargetTestCaseName);
log.info(testCase)

print '### Get Result data'
runner = testCase.run(new com.eviware.soapui.support.types.StringToObjectMap(), false)
def value= testCase.getPropertyValue("getRandomValue")
return value
log.info(value)

runner = testCase.run(new com.eviware.soapui.support.types.StringToObjectMap(), false)
```

Figura 32 - Script RandomValue

Além do *TestCase "GroovyScrips"*, existe também o *TestCase "Current Variables"*, o seu objectivo é permitir configurar propriedades e variáveis globais utilizadas no *TestSuite* "RegistoDespesasOT0001". A Figura 33 representa a propriedade "*Endpoint*" que é utilizada como URL nos vários pedidos realizados à API no *TestSuite* "RegistoDespesasOT0001".

CurrentEndpoint

Load from: Save to:

Name	Value
Endpoint	https://eamapi.primaveraspace.com

Figura 33 - CurrentEndpoint

Como podem verificar na Figura 31, relativamente ao *TestSuite* "RegistoDespesasOT0001" este contém dois *TestCases*, o primeiro está relacionado com o processo de login, e o segundo com o pedido à API do EAM.

O *Testcase* "*Login*" está a invocar um pedido *GET* à API do *EAM* onde são passados os seguintes parâmetros: "*Username"* e "*Password",* resultado numa resposta em formato *JSON* constituída pelos dados do utilizador, os seu módulo e a organização respectiva, tal como representado na Figura 34.

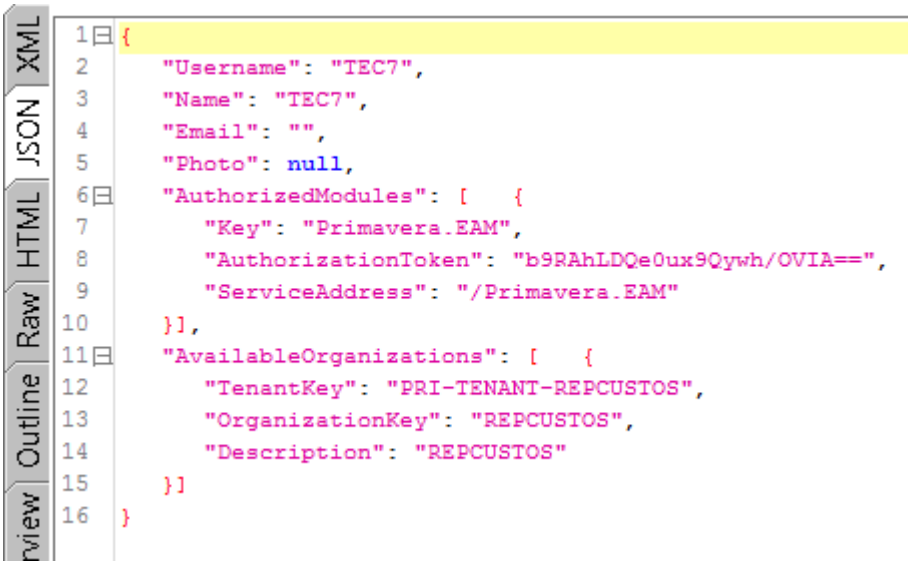


Figura 34 - JSON resposta Login

Recorrendo à funcionalidade "*Property Transfer*" da aplicação *ReadyAPI* esta transfere três propriedades que serão utilizadas para nos próximos pedidos à API. As propriedades são: "TenantKey", "OrganizationKey" e "TokenKey", tal como representado na Figura 35.

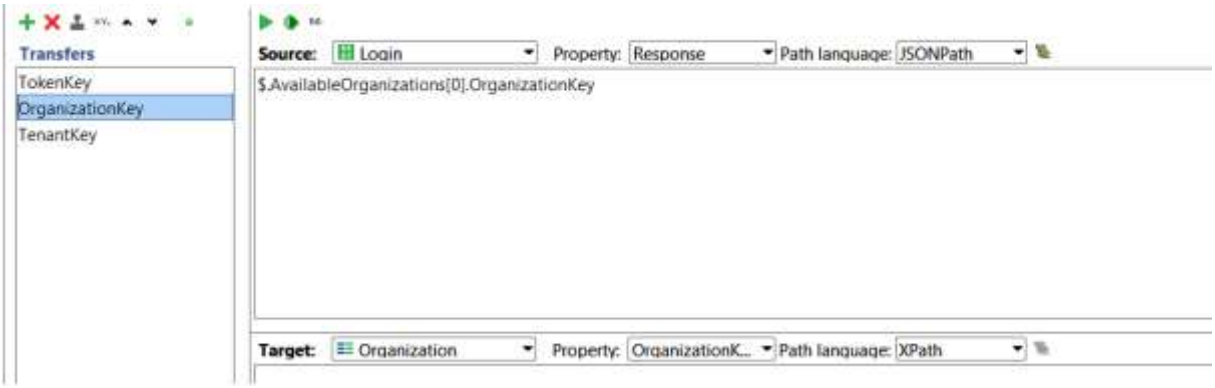


Figura 35 - Property Transfer

O *TestCase* "Registo de DespesasTec1" é constituído pelas seguintes funcionalidades: Criar umnovo *datasource*, criar um novo pedido *REST* e por fim criar um novo ciclo aplicado ao *datasource*.

Na funcionalidade de criação de um novo *datasource* é necessário configurar o tipo de *datasource* como tipo de ficheiro *Excel*, indicando posteriormente a localização do ficheiro *excel* que contém os dados de teste, como representado na Figura 36.

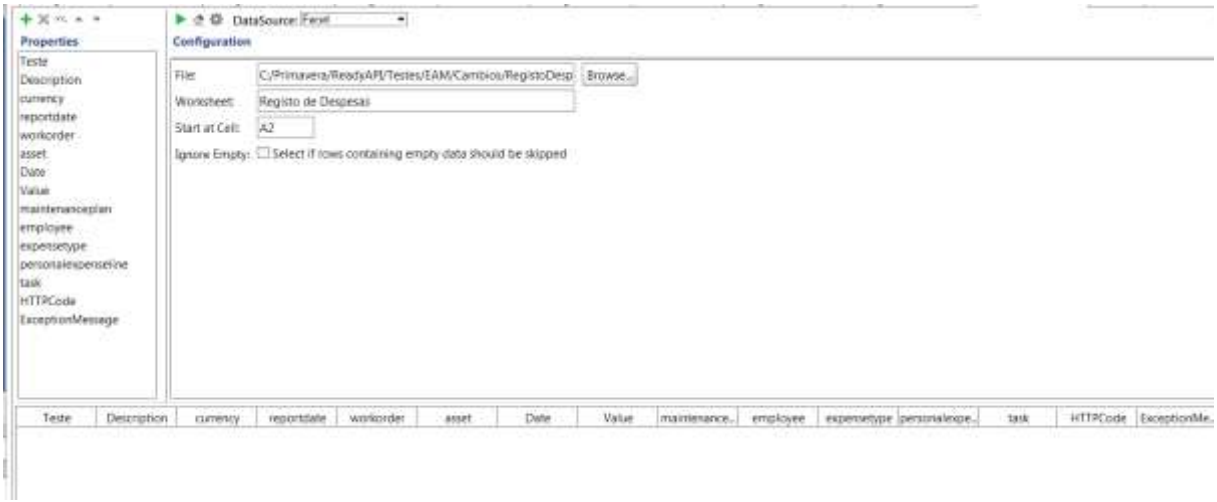
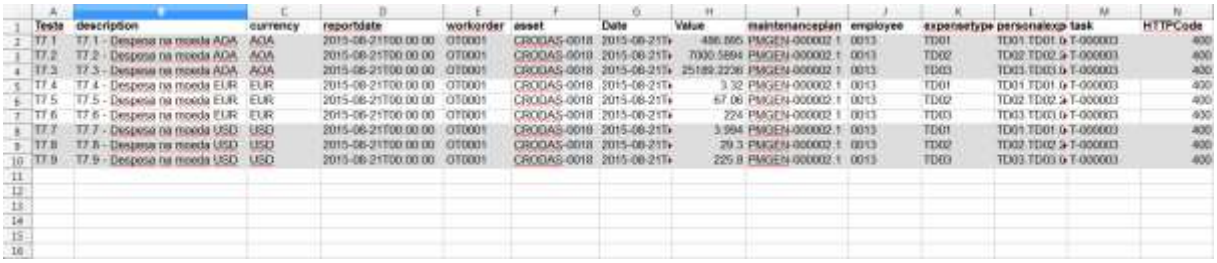


Figura 36 - Configuração datasource

Foi utilizado o tipo de *datasource excel*, ver Figura 37, devido ao facto de proporcionar simplicidade na adição e manutenção de novos casos de teste.



Teste	description	currency	reportdate	workorder	asset	Date	Value	maintenanceplan	employee	expensetype	personalexp	task	HTTPCode
T7.1	T7.1 - Despesa na moeda AOA	AOA	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	486.865	PMGEN-000002.1	0013	TD01	TD01.TD01.0	T-000003	400
T7.2	T7.2 - Despesa na moeda AOA	AOA	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	7000.5894	PMGEN-000002.1	0013	TD02	TD02.TD02.2	T-000003	400
T7.3	T7.3 - Despesa na moeda AOA	AOA	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	25189.2236	PMGEN-000002.1	0013	TD03	TD03.TD03.0	T-000003	400
T7.4	T7.4 - Despesa na moeda EUR	EUR	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	3.32	PMGEN-000002.1	0013	TD01	TD01.TD01.0	T-000003	400
T7.5	T7.5 - Despesa na moeda EUR	EUR	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	67.06	PMGEN-000002.1	0013	TD02	TD02.TD02.2	T-000003	400
T7.6	T7.6 - Despesa na moeda EUR	EUR	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	224	PMGEN-000002.1	0013	TD03	TD03.TD03.0	T-000003	400
T7.7	T7.7 - Despesa na moeda USD	USD	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	3.994	PMGEN-000002.1	0013	TD01	TD01.TD01.0	T-000003	400
T7.8	T7.8 - Despesa na moeda USD	USD	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	29.3	PMGEN-000002.1	0013	TD02	TD02.TD02.2	T-000003	400
T7.9	T7.9 - Despesa na moeda USD	USD	2015-06-21T00:00:00	OT0001	CRODAS-0018	2015-06-21T	225.8	PMGEN-000002.1	0013	TD03	TD03.TD03.0	T-000003	400

Figura 37 - datasource Excel

A funcionalidade de criar um novo pedido *REST* permite a invocação à *API* do *EAM*, sendo esta a funcionalidade base de todo o processo.

Na Figura 38 podemos verificar o exemplo do pedido "Registo de Despesas". Este pedido é um *POST* do método "*workorderreportexpenses*" existente na *API* do *EAM*, como *request* são enviadas as propriedades "TenantKey" e "OrganizationKey" recolhidas no *TestCase* "LoginTEC1".

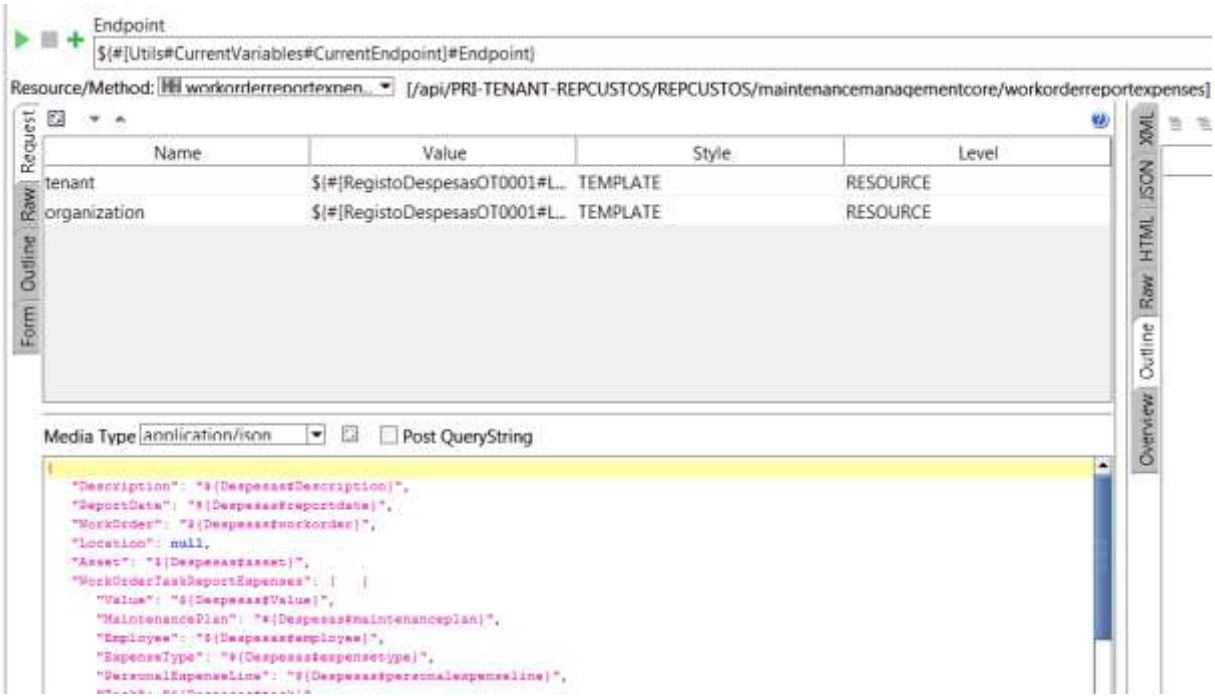


Figura 38 - Exemplo Request Registo de Despesas

No cabeçalho do pedido é enviado um "Authorization" utilizando a propriedade "TokenKey" recolhida no *TestCase* "LoginTEC1", ver Figura 39.

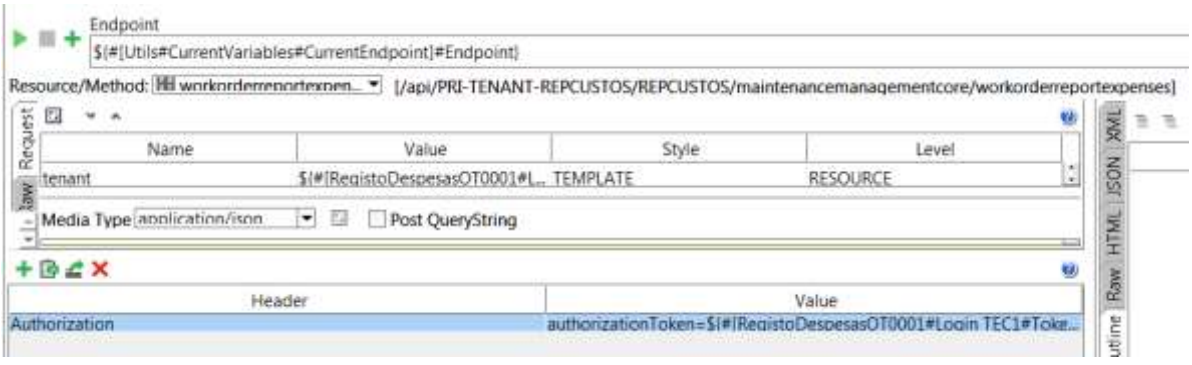


Figura 39 - Cabeçalho Authorization

No *request "Body"* são enviados os dados do pedido, obtidos a partir do *Excel* referenciado anteriormente, ver Figura 40.

Media Type application/json Post QueryString

```
{
   "Description": "${Despesas#Description}",
   "ReportDate": "${Despesas#reportdate}",
   "WorkOrder": "${Despesas#workorder}",
   "Location": null,
   "Asset": "${Despesas#asset}",
   "WorkOrderTaskReportExpenses": [   {
      "Value": "${Despesas#Value}",
      "MaintenancePlan": "${Despesas#maintenanceplan}",
      "Employee": "${Despesas#employee}",
      "ExpenseType": "${Despesas#expensetype}",
      "PersonalExpenseLine": "${Despesas#personalexpenseline}",
      "Task": "${Despesas#task}",
      "Date": "${Despesas#Date}"
   }],
   "Currency": "${Despesas#currency}",
   "Company": "${#[RegistoDespesasOT0001#Login TEC1#Organization]#OrganizationKey}"
}
```

Figura 40 - Request Body

Após o pedido são realizadas validações às respostas. No *excel* existem duas colunas denominadas de "*HTTPCode*" e "*ExceptionMessage*" que são utilizadas para garantir as validações, ver Figura 41.

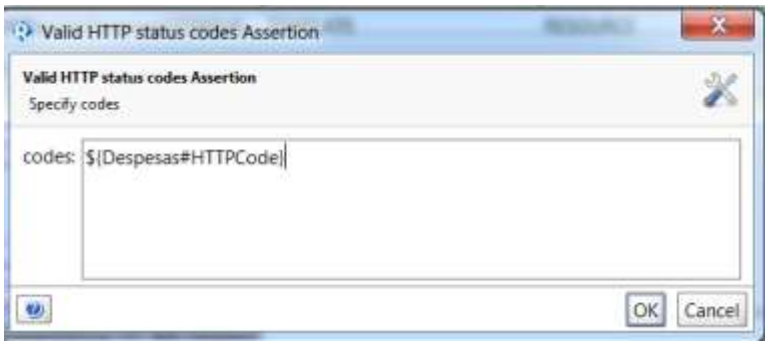


Figura 41 - Validações do pedido

As validações foram definidas no plano de teste, ver ponto 4.3, e são adicionadas na funcionalidade "*Assertions*" da ferramenta *ReadyAPI* como validações "*HTTP"* no caso do "HTTPCode" e validações "JSONPath Expression" para o caso do "*ExceptionMessage*".

Por fim é criado um ciclo aplicado ao *datasource* e ao pedido *REST* de forma a percorrer todas as linhas inseridas no ficheiro *excel*.

4.4.4 Execuções e resultados

Após o desenvolvimento dos testes é necessário iniciar a sua execução automática. A ferramenta *ReadyAPI* permite executar os testes criados através da funcionalidade "*Launch Test Runner*". Para tal é necessário configurar os *TestSuites* e *TestCases* que serão executados, indicar a localização da pasta para guardar os resultados, escolher o formato dos resultados que queremos exportar e definir a possibilidade de parar a execução em caso de erro, ver Figura 42.

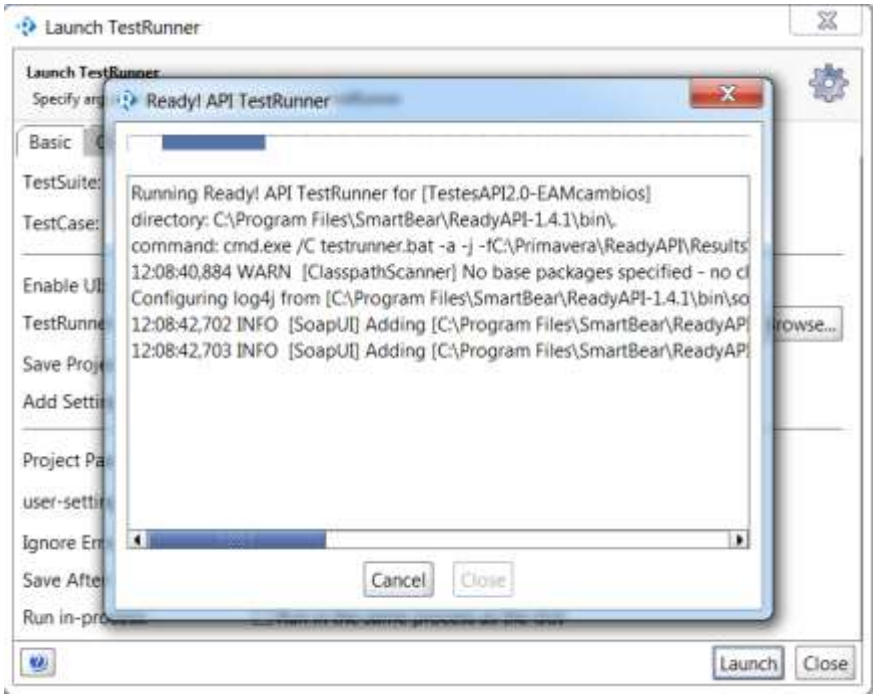


Figura 42 - Launch TestRunner

Através da funcionalidade "*Launch TestRunner*" da ferramenta *ReadyAPI* não é possível agendar execuções automáticas, como tal foi necessário desenvolver um ficheiro *bat* para iniciar a execução. Este ficheiro referencia o comando "*testrunner*", ver Figura 43, localizado na pasta "*bin*" da ferramenta *ReadyAPI* e será inicializado após a instalação e configuração do ambiente de execução.

```
cd C:
cd C:\Program Files\SmartBear\ReadyAPI-1.4.1\bin
  C:
testrunner.bat -a -j -fC:\Primavera\ReadyAPI\Results\EAM\TestesAPI2.0-EAMcambios -R"Project Report" -E"
Default environment" C:\Primavera\ReadyAPI\Testes\EAM\Cambios\TestesAPI2-0-EAMcambios-soapui-project.xml
```

Figura 43 - ficheiro bat iniciar execução

Após a execução são gerados vários ficheiros, em formatos *html,css,xml e txt*, que constituem os resultados obtidos pela execução.

O ficheiro "*overview-summary.html*" proveniente dos relatórios gerados, apresenta os resultados em formato *html* com o sumário de toda a execução, podendo clicar nos vários itens apresentados, podendo ver assim o resultado do item em detalhe, tal como representado na Figura 44.

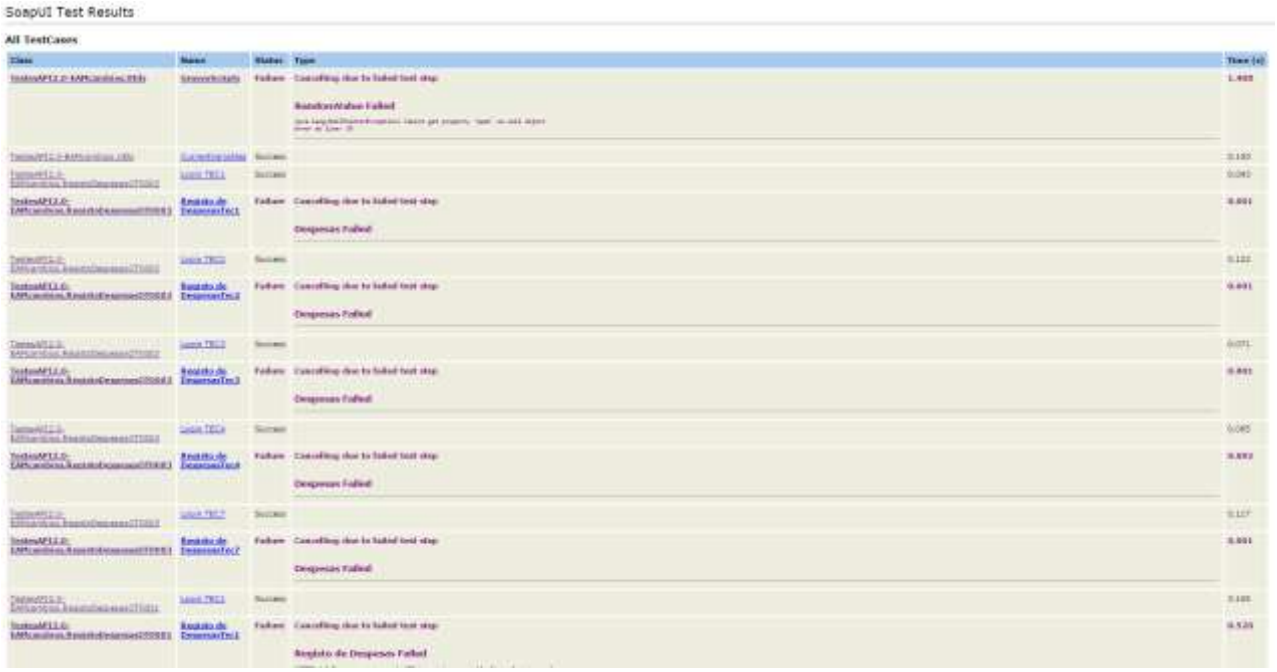
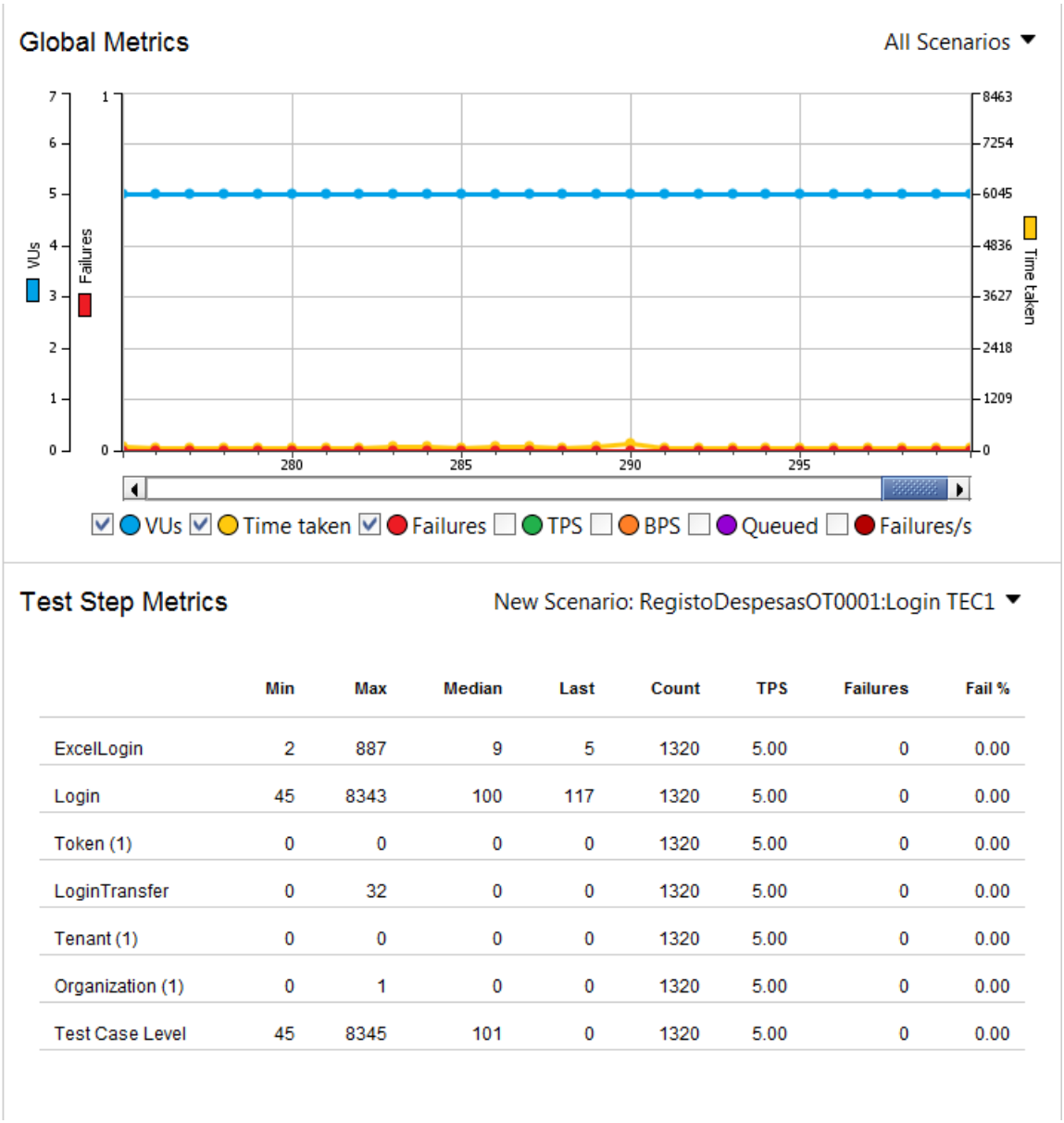


Figura 44 - ficheiro overview-summary.html

Na PRIMAVERA existe um automatismo que vai registar o detalhe dos resultados na base de dados dos testes automáticos, para posteriormente seja possível a consulta do histórico de resultados.

Como complemento à funcionalidade anterior é possível ainda utilizar a funcionalidade "*LoadUI NG*" que permite medir as métricas dos steps que constituem um *testecase*. Esta opção permite visualizar graficamente o desempenho da execução, ver Figura 45.



Test Step Metrics — New Scenario: RegistoDespesasOT0001:Login TEC1

	Min	Max	Median	Last	Count	TPS	Failures	Fail %
ExcelLogin	2	887	9	5	1320	5.00	0	0.00
Login	45	8343	100	117	1320	5.00	0	0.00
Token (1)	0	0	0	0	1320	5.00	0	0.00
LoginTransfer	0	32	0	0	1320	5.00	0	0.00
Tenant (1)	0	0	0	0	1320	5.00	0	0.00
Organization (1)	0	1	0	0	1320	5.00	0	0.00
Test Case Level	45	8345	101	0	1320	5.00	0	0.00

Figura 45 - LoadUI NG

4.5 Conclusão

Concluindo este capítulo obteve-se como resultado a implementação de uma *framework* que é constituída por um ambiente de execução e um ambiente de desenvolvimento de teste à API da aplicação EAM da empresa PRIMAVERA Software.

Relativamente ao ambiente de execução, este tem como objectivo ter um ambiente preparado para execução dos testes. As vantagens deste ambiente de execução são garantir que o ambiente de testes não difere nas várias execuções, resultando assim que qualquer alteração nos testes tem origem apenas na aplicação que está a ser testada.

O ambiente de desenvolvimento constituído pelo projecto do *ReadyAPI* simplifica a manutenção futura e a adição de novos *scripts* de testes, reduzindo assim o tempo de automatização. Sendo isto possível com a utilização da metodologia *data-driven*, da criação de variáveis globais, da parametrização de pedidos *http* e da integração com o projecto comum criado com o objectivo de ser uma biblioteca de dados e funcionalidades. Por fim os resultados obtidos são enviados para uma base de dados, para possibilitar a criação de um histórico de execuções e consequente análise dos dados.

5 Conclusões

Neste último capítulo é encerrada a dissertação de mestrado. Apos a realização e descrição de todos os passos realizados neste trabalho faz-se neste capítulo uma descrição sobre os resultados obtidos, sobre as limitações que a realização deste projeto teve. Faz-se ainda neste capítulo uma pequena observação, sobre alternativas que poderiam ser realizadas e sobre o trabalho futuro que esta dissertação pode suportar.

5.1 Conclusões finais

Os objectivos propostos relativamente ao desenvolvimento do presente documento designado de dissertação foram atingidos. Atualmente o tipo de desenvolvimento de software tende para o desenvolvimento de software como serviço, a implementação de serviços na *cloud* é uma prática cada vez mais comum nas empresas ligadas à área dos sistemas de informação. Como tal é necessário garantir a qualidade deste tipo de software utilizando os testes automáticos. Uma organização que detém testes automáticos promove a qualidade nos processos críticos, possibilitando-a a um aumento na rapidez relativa à resolução de problemas, levando ao aumento de qualidade do produto e satisfação do cliente.

Este trabalho apresenta uma contribuição para a área de automatização de testes, com a aplicação de uma *framework* que é constituída por um ambiente de desenvolvimento e um ambiente de execução de teste à API.

Em suma a utilização da framework vai facilitar a criação, manutenção e execução de testes sobre a API. Com esta estrutura a manutenção e inserção de novos scripts poderá ser um processo tão simples como inserir ou alterar os dados de um *datasource*, resultando assim na diminuição do tempo de automatização e respectiva manutenção.

5.2 Limitações

Uma das principais limitações desta dissertação prende-se pelo facto da base de dados do produto preparada com os dados necessários para executar os vários cenários de teste, como por exemplo dados relativos aos funcionários, tarefas, ordens de trabalho e algumas regras de relação entre estes, caso seja necessário criar novos dados ou até mesmo alterar alguma relação entre estes dados será necessário aplicar estas alterações manualmente, fazendo posteriormente um novo *backup* e colocar a base de dados no servidor de forma a ser copiada e restaurada nas execuções seguintes.

5.3 Trabalho futuro

No âmbito desta dissertação podemos concluir que a utilização do projecto comum, designado por "TestesAPI2.0-Common", permite partilhar funcionalidades comuns entre vários projectos de teste que existam ou que possam surgir no futuro, evitando a duplicação de funcionalidades iguais.

Relativamente ao trabalho futuro podem ser adicionadas novas funcionalidades e utilitários ao projecto comum de forma a promover ainda mais a facilidade de criação e manutenção de novos casos de testes, permitindo a evolução continua e estruturada da *framework* resultante desta dissertação.

Outro aspecto a ser trabalhado seria a possibilidade de partilhar os resultados obtidos pelas execuções numa aplicação *web*, facilitando assim a consulta e consequente análise de resultados pelas equipas, permitindo que estas tomem conhecimento do estado atual das execuções de uma forma mais rápida, e no caso de falha, promova a rapidez na resolução do problema. Associar ainda a este mecanismo uma funcionalidade de alertas automáticas via *e-mail*.

Referências

(s.d.). Obtido de http://renttesters.com/wp-content/uploads/2014/12/SoapUI-Demo-Framework.jpg

Cambiucci, W. (Maio de 2009). Microsoft. Obtido em 20 de Janeiro de 2015, de Uma introdução ao Software + Serviços, SaaS e SOA: https://msdn.microsoft.com/pt-br/library/dd875466.aspx

Cloudways. (28 de Novembro de 2011). What is Public Cloud. Obtido de Cloudways: http://www.cloudways.com/blog/what-is-public-cloud/

exuberantsolutions. (s.d.). exuberantsolutions. Obtido de Cloud Computing: http://www.exuberantsolutions.com/cloud_course.htm

IBM. (s.d.). IBM developerWorks: About cloud computing. Obtido de IBM developerWorks: http://www.ibm.com/developerworks/cloud/about.html

Kansomkeat, & Rivepiboon. (2013). Automated-generating test case using UML statechart diagrams. pp. 296-300.

Magalhães, G. (19 de Março de 2012). Protocolo TI:SaaS, PaaS e IaaS - As camadas do Cloud Computing. Obtido de Protocolo TI: http://protocoloti.blogspot.pt/2012/03/saas-paas-e-iaas-as-camadas-do-cloud.html

Microsoft. (2015). Testing Performance and Stress Using Visual Studio Web Performance and Load Tests. Obtido de https://msdn.microsoft.com/en-us/library/vstudio/dd293540(v=vs.110).aspx

Postman | Supercharge your API workflow. (s.d.). Obtido de https://www.getpostman.com/

Reis, L. P. (2014). Pré-Dissertações e Pré-Projetos Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação. Obtido em 3 de Outubro de 2014

Ricardo, A. (28 de Março de 2013). O que é Saas,Iaas e Pass em Cloud Computing? (Conceitos básicos). Obtido de AntonioRicardo: http://antonioricardo.org/2013/03/28/o-que-e-saas-iaas-e-paas-em-cloud-computing-conceitos-basicos/

Rozlog, M. (2 de Outubro de 2013). InfoQ. Obtido de REST e SOAP: Usar um dos dois ou ambos? : http://www.infoq.com/br/articles/rest-soap-when-to-use-each

Sharma, M., & Mall, R. (2009). Automatic generating of test specifications for coverage of system state transitions. Information and software Technology. pp. 418-432.

SISNEMA. (22 de 01 de 2009). SISNEMA Informática. Obtido de Cloud Computing - novo modelo de computação: http://sisnema.com.br/Materias/idmat019433.htm

SoapUI - The Home of Functional Testing. (s.d.). Obtido de http://www.soapui.org

Solution, S. (2013). renttesters. Obtido de renttesters: http://renttesters.com/

Fernandes, G. (2014). Geração Automática de Casos de Teste a partir de Requisitos.

Hevner, a. R., March, S. T., & Park, J. (2004). Design Science in Information Systems Research. MIS Quarterly, 28(1), 75–105. doi:10.2307/25148625

Iyer, G. N. (n.d.). Test Challenges and Approaches With SaaS and PaaS.

ken Schwaber, Jeff Sutherland, C. (2014). O guia do SEO.

Lopes, M. A. (2015). Universidade do Minho Escola de Engenharia Manuel António Lopes Pereira Conceção de Arquiteturas para Cloud Computing : Casos de Demonstração da Utilização do Modelo de Referência.

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. Nist Special Publication, 145, 7. doi:10.1136/emj.2010.096966

Mumbaikar, S., & Padiya, P. (2013). Web Services Based On SOAP and REST Principles, 3(5), 3–6.

Peffers, K. E. N., Tuunanen, T., Rothenberger, M. a, & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems, 24(3), 45–77.

Sharma, M., Bansal, H., & Sharma, A. K. (2012). Cloud Computing : Different Approach & Security Challenge. International Journal of Soft Computing and Engineering, 2(1), 421–424.

Software, I., & Qualifications, T. (2011). Programa de Certificação de Testador (Tester) de Nível Foundation.

Vaishnavi, V., & Jr, W. K. (2007). Design science research methods and patterns: innovating information and communication technology. Vasa. Retrieved from

http://medcontent.metapress.com/index/A65RM03P4874243N.pdf\nhttp://books.google.com/books?hl=en&lr=&id=sI2Y9Jh8tq8C&oi=fnd&pg=PP1&dq=design+science+research+methods+and+patterns+innovating+information+and+communication+technology&ots=k6bC04QyYN&sig=QX4lD6