TODO

9 MAIO -> IBM Watson a responder minimamente TODO -> Melhorar a conversação, criação do módulo de previsão, ligação entre a java app com a web

16 MAIO -> Para resolver a ligação java -> web foi tentado usar a extensão applet, mas está deprecated desde o java 7. Foi necessário encontrar outra ferramenta para resolver este problema.

A primeira ferramenta encontrada foi o Apache Tomcat. Foram feitos vários testes e várias tentativas de criar o servidor utilizando esta ferramenta. Foi usado um exemplo simples, que funcionava mas quando foi tentada implementar na versão do projeto, onde de se tinha de lidar com todos os imports das classes do projeto e ainda com a configuração do servidor (web.xml). Como estas configurações eram bastante complicadas de se ligar para criar algo simples (é preciso apenas criar um servidor que receba operações HTTP), foi importada a Jersey api e, foi então assim criado um servidor REST. Este servidor apenas tem duas operações a que responde, um GET que devolve a mensagem inicial do bot e um POST em que o cliente envia o input ao servidor e ele responde com a resposta do bot.

Como o servidor já poderia ser acedido pela web com pedidos http, foi criado um ficheiro em javascript (restRequests.js) que faz estes requests por HTTP ao servidor REST e envia as respostas para uma página HTML, onde irá decorrer a conversa entre o bot e o cliente.

Foi ainda apresentada uma ferramenta nova de machine learning (Microsoft Azure Machine Learning) que irá ser usada para desenvolver as revisões do tempo. Até ao momento é uma ferramenta bastante intuitiva, mas só foram vistos apenas demos de como funciona, isto é, ainda não foi mantido o contacto necessário a total compreensão da ferramenta.

TODO -> Melhorar a GUI que neste momento se encontra bastante simples, entender as opções que estão disponíveis pela nova ferramenta de ML, para a realização da última parte base do projeto de estágio (dados para serem analisados em falta!).

17 mai. 18 -> Ontem foi criado um GUI simples que será usado para o utilizador manter contacto com o bot. Foi usado HTML/CSS e Javascript (melhorado o primeiro chat criado a semana passada).

Para hoje será limpar o código e preparar o projeto para receber novas funcionalidades (novas cidades). Se houver tempo descobrir como se acede através da API ao Microsoft Azure Machine Learning Studio. Pois já foi feito um prove of concept, mas ainda não foi feita a tentativa de pedir informações sobre previsões.

Foram adicionados os distritos de Portugal continental à base de dados.

23 MAIO -> Esta semana vai ser dedicada à criação de casos de teste para as operações que ocorrem no chatbot. Irá respeitar o template de exemplo disponibilizado.

Cada caso de teste irá conter o normal funcionamento da operação em questão e ainda as exceções que podem ocorrer na presença de um erro seja este de rede, ou um problema de possível má formatação de dados.

TODO -> Para a semana serão acabados os casos de teste se ainda faltarem. Se esta etapa acabar no final desta semana, para a semana irá ser começado a nova parte do projeto de estágio de machine learning. Esta parte já não irá ser a prevista que consistia em fazer uma máquina de aprendizagem capaz de dar a previsão do tempo. O novo desafio será criar uma máquina capaz de reconhecer nomes de empresas em qualquer disponível que lhe seja entregue. Se for tanto um artigo online como um livro, terá de ser capaz de encontrar o nome das empresas.

30 MAIO -> Na semana passada foi criado um mini programa que usa os serviços de Natural Language Processing do IBM Watson para testar o mesmo. Este serviço consiste em inserir um texto ou um URL de uma página para ser analisado. Quanto este é analisado, o serviço mostra o sentimento que este texto transmite, mostra as entidades envolvidas (pessoas, empresas, etc.) e mostra outras tantas coisas que se podem concluir do texto inserido. O desafio de Machine Learning começou também a semana passada. O objectivo será que através do texto inserido, será gerado um caso de teste de acordo com este. Numa primeira intuição foi pensada uma solução que usasse Natural Language Processing. O serviço usado anteriormente para resolver o problema em questão torna-se inútil porque os resultados obtidos por este não nos ajudam a conseguir caracterizar um caso de teste. Um caso de teste é caracterizado por um nome, uma descrição, passos, requerimentos, resultado esperado, etc. Assim foram feitas algumas pesquisas sobre Natural Language Processing onde foi encontrada uma ferramenta (MonkeyLearn) que classifica o texto inserido como uma categoria. Esta categoria pode ser uma categoria da ferramenta ou pode ser uma categoria customizada. O problema desta ferramenta é que classifica um texto apenas como uma categoria, e como um texto tem várias categorias, esta ferramenta também é inútil.

Foram feitas pesquisas sobre Deep Learning também, pois este método de aprendizagem é a base do NLP.

TODO -> Acabar as pesquisas e começar a preparar as ferramentas necessárias para começar a criar o novo problema

7/junho -> Continuar o estudo para a realização da criação de casos de teste baseados nos requerimentos. Foi encontrada a técnica de “Predictive Analytics” que consiste em analisar factos correntes e passados para fazer previsões sobre o futuro ou de eventos desconhecidos.

TODO -> Obtenção de conhecimento

8/junho -> Foi encontrada durante o dia de ontem uma possível solução usando não só natural language processing como também diagramas de atividade e de sequência. Irá consistir em transformar a informação classificada pela parte de machine learning nós para criar os diagramas, onde manualmente o user irá ligar estes nós da forma que este pretender para depois, transformar estes diagramas em grafos e depois criar um caso de teste através de uma pesquisa em profundidade. Hoje vai ser começada esta solução com a interpretação dos ficheiros inseridos através de bibliotecas de javascript. O próximo passo será fazer pedidos ao classificador com as frases extraídas dos documentos e com a resposta do mesmo criar uma estrutura de dados que irá guardar todas a classificações.

14/junho -> Durante o dia 8 de junho, para usar as bibliotecas necessárias de javascript, foram feitas pesquisas sobre como o Node.js funciona e também como usar o npm, para poder manipular as bibliotecas descobertas previamente. No entanto, foi descoberto que estas bibliotecas só poderiam ser usadas localmente porque os web browsers não reconhecem o método require, o método que permite usar uma biblioteca do npm num ficheiro Javascript. Foram assim pesquisadas outras formas de “dar a volta” ao problema. A melhor solução encontrada foi criar um servidor REST capaz de extrair dos documentos inseridos no drop, as informações necessárias para que o método de classificação possa classificar. Como os documentos que serão em formato pdf e excel, foram encontradas APIs para tratar da leitura destes. No caso do PDF foi usado o iText. Esta ferramenta permite tanto ler como escrever um documento em formato pdf de uma maneira fácil. Já no caso da interpretação de uma folha excel ainda não há uma ideia fixa em usar uma biblioteca do Java também ou usar a biblioteca encontrada anteriormente para Javascript (SheetJS). Por isso, o problema neste momento poderá funcionar assim (poderá sofrer ainda mudanças, visto que é apenas um “protótipo de pensamento”):

- Numa página web, faz drag and drop de um Requirements Document e de uma folha Excel com as outras informações do caso de teste que queremos (nome, descrição, etc.)

- Destes documentos serão extraídas as informações importantes, através do uso das bibliotecas para ler documentos deste tipo

- As informações sobre os requerimentos serão classificadas através de uma ferramenta de Natural Language Processing (MonkeyLearn). Isto irá classificar o tipo de requerimento que pretendemos.

- As classificações iram ser guardadas numa estrutura de dados para organização.

- Estas informações criaram nós num diagrama de atividades. Depois desta criação, o utilizador irá simplesmente de fazer as ligações entre os nós, para poder criar uma ordem cronológica dos steps. O nome de cada step será a descrição de cada ligação entre dois nós e o resultado esperado será o conteúdo do nó. Cada diagrama de atividade irá ter informações para analisar e criar todos os workflows. Cada workflow vai corresponder a um caso de teste.

- Após a criação de todos os workflows, os diagramas iram ser transformados em grafos. E para criar o caso de teste irá ser feita uma pesquisa em profundidade destes grafos.

- No final, após todos os casos de testes estarem criados, estas informações serão injetadas num ficheiro de Excel e enviadas ao utilizador.

TODO -> Acabar a parte de extração dos requerimentos a partir de um ficheiro pdf. Treinar o algoritmo de classificação com alguns exemplos. Continuar a configuração do servidor e criar pedidos ao servidor do MonkeyLearn para poder classificar as informações extraídas do ficheiro.

Criar a parte web do projeto, que irá envolver as linguagens usadas previamente utilizadas na realização do ChatBot.

15/junho -> No dia anterior foi quase acabada a leitura dos documentos dos requerimentos. Foi acabado. Criada a pré-condição que para ser um documento válido para analisar é necessário que todos os capítulos estejam em páginas diferente no documento, senão puderam se perder alguns requerimentos.

Não existem exemplos suficientes para treinar o classificador de requerimentos, por isso irá ser feito todo o projeto sem a classificação deste para poder obter mais conhecimentos sobre toda a realização do projeto.

TODO -> Ler um documento Excel, criar o servidor REST para atender aos pedidos da aplicação Web. Este servidor irá analisar os documentos PDF e Excel inseridos na página Web e irá enviar os requerimentos necessário para classificar ao MonkeyLearn através de pedidos HTTP. A resposta do servidor REST para a web irá ser em formato JSON para facilitar a leitura dos dados.

Criar a página web e usar o flowchart.js para criar um diagrama de atividade.

20/Junho -> Leitura de um documento Excel acabada e o servidor REST “já liga” mas ainda não foi testado. MonkeyLearn ainda não usado por falta de requerimentos para treinar.

TODO -> Problema no código com a transformação das informações para criar o pedido. O pedido a esta ferramenta é feito através dum JSON quem tem o campo “data” e o seu valor tem de ser obrigatoriamente uma lista.

Começo do front end do projeto.

\*Update\* Problema com o pedido resolvido. Foi usado a classe JsonArray para conseguir criar a lista necessária para o pedido.

Acabar a classe ClassificationsRequest com o mapa quando o NLP estiver treinado.

NPL treinado com alguns exemplos. Não está com muita precisão.

Problema encontrado com não conseguir usar ficheiros importados para o html resolvido. Bastou criar uma pasta chamada public.

21/Junho -> Para receber os ficheiros necessários para a realização dos casos de teste, foi usada uma ferramenta open source designada Dropzone.js. Esta ferramenta torna-se bastante fácil de usar pois basta ter no ficheiro html um nome especifico para cada elemento e esta auto configura-se. Porém, a Dropzone.js para funcionar necessita de um pacote npm designado por express. Este pacote é uma ferramenta que permite de uma maneira fácil de criar um servidor que responde a pedidos HTTP. O único problema deste pacote mencionado anteriormente é a dificuldade inicial de perceber como funciona e não como se implementa. A nova versão desta ferramenta está separada por módulos, o que para tratar o upload de cada ficheiro foi necessário o módulo express-fileupload, que acrescenta novas funcionalidade ao express. Foi usado também outro npm package para conseguir traduzir o conteúdo de cada pedido. Sem este package, todas as vezes que pretendemos aceder ao corpo da mensagem HTTP esta é imprimida no terminal como *undifined*.

TODO -> Verificar o porquê do corpo da mensagem estar vazio. Criar a ligação entre os ficheiros recebidos e o servidor REST, para poder testar se o servidor está a responder corretamente ao que é pedido.

9down vote

*It looks like body-parser did support uploading files in Express 3, but support was dropped for Express 4 when it*[*no longer included Connect as a dependency*](http://expressjs.com/en/guide/migrating-4.html#core-changes)

*After looking through some of the modules in mscdex's answer, I found that*[*express-busboy*](https://www.npmjs.com/package/express-busboy)*was a far better alternative and the closest thing to a drop-in replacement. The only differences I noticed were in the properties of the uploaded file.*

22/Junho -> Novas ferramentas usadas para resolver alguns problemas.

File-api que é o pacote que inclui todas as funcionalidades necessárias para lidar com ficheiros. Anteriormente estava apenas a ser usado o FileReader. Mas para ser mais fácil de ler um ficheiro guardado no disco foi necessário usar também a funcionalidade File deste pacote.

Request é um pacote que permite de uma forma bastante intuitiva criar ficheiro HTTP. Este pacote é necessário pois a maneira standard de criar estes pedidos são feitos apenas com funcionalidades disponíveis nos web browsers, mas como esta parte do projeto está a ser desenvolvida em Node.js, torna-se impossível o uso destas funcionalidades.

O envio do ficheiro em formato PDF para o servidor REST já está operacional. O único problema é a ferramenta de Natural Language Processing já está no limite de pedidos para a classificação de um documento. Provavelmente será necessário procurar outra ferramenta.

O envio do ficheiro Excel não está a funcionar.

TODO -> Resolver o problema com o Excel. Começar a usar o Flowchart.js

27/Junho -> Foi descoberto um erro no envio de ficheiros em formato PDF e tal como o envio do ficheiro Excel, este não está operacional. Foi encontrado em várias pesquisas que para enviar estes ficheiros, existe um “Content-type” específico para cada um deles. Para PDF é *application/pdf* e para Excel (xlsx) é *application/vnd.openxmlformats-officedocument.spreadsheetml.sheet*. Como está a ser usado um servidor REST (Jersey) para processar estes ficheiros e este não contem nenhum dos “*MediaTypes*” referidos anteriormente. Ou seja, a única maneira de enviar estes ficheiros seria através de um Byte array. Mas esta forma de envio estava a corromper os dois tipos de ficheiro.

Por isso, para evitar perda de tempo para a realização do projeto ou pelo menos para permitir uma passagem breve por todas as etapas previamente definidas, irá ser usada a DropBox para fazer a passagem de ficheiros entre a Web e o servidor REST, sendo enviado no pedido HTTP o URL do ficheiro na Dropbox em vez do próprio ficheiro.

TODO -> Implementar as operações da Dropbox no servidor REST.

Como o modelo da ferramenta de NLP não tem treino suficiente, irá ser criada a resposta “à mão” do exemplo que está a ser usado para começar a construir os diagramas UML e começar a construir a página web.

28/Junho -> Com a Dropbox, o problema persiste. Foi descoberta a origem do problema. O problema vem do charset de um ficheiro PDF, que quando transformado num byte array e enviado para o java criar um ficheiro com essa informação, este byte array têm um charset diferente do pré definido, ou seja, cria um ficheiro diferente do que se quer. A solução que se tentará durante o dia de hoje será implementar a manipulação do ficheiro Excel diretamente no Javascript, para evitar o envio do ficheiro codificado em byte arrays, preservando assim a sua codificação. Após feita esta alteração, o procedimento será equivalente para ficheiros Excel

TODO -> Até ao resto da semana é esperado o processamento tanto do ficheiro PDF como do ficheiro Excel para poder avançar para a parte do UML.

29/Junho -> Conseguido o processamento do PDF através de um form-data. É um tipo de conteúdo suportado em HTTP que funciona como caixa e permite enviar mais do que um tipo de ficheiros dentro do mesmo. E foi descoberta também para esta solução as anotações Jersey especificas para este tipo de conteúdo. Existem alguns objetos disponibilizados pelo Jersey para tratar deste tipo de conteúdo, mas nenhum funciona. Foi então usada uma InputStream que devolve o corpo do pedido HTTP vindo do Javascript. Porém, esta única maneira de conseguir aceder aos dados enviados pelo pedido também tinha informações sobre a informação do conteúdo de dentro do form, como o seu “Content-type”, tamanho, etc. Foi necessário a tradução dos arrays de bytes recebidos pela stream para string para se poder analisar o conteúdo que o array trazia a cada iteração da stream.

Este problema de necessária limpeza da informação adicional foi preciso resolver devido aos ficheiros em formato Excel, pois para os ficheiros em formato PDF era totalmente irrelevante. Simplesmente foi feita a diminuição do tamanho do array de bytes para receber simplesmente este cabeçalho com as informações e ignorá-lo.

Foi descoberto que também existe um fim de InputStream personalizado do pacote npm usado, porém não influenciou na construção dos ficheiros.

TODO -> O que será feito o resto do tempo será perceber o FlowCharts.js e pesquisar uma ferramenta nova de classificação de requerimentos.

2/Julho -> Foi acabada a passagem de ficheiros por http para o processamento dos mesmos. O novo desafio será criar com as informações extraídas diagramas UML de atividade e de sequência. Por enquanto será usada uma biblioteca de js. E será criado um mapa com as informações “hard coded” para poder tratar o novo desafio e enquanto não há uma nova ferramenta de classificação.

Antes de trabalhar nos diagramas será preciso melhorar a resposta do servidor para poder haver uma criação mais fácil. Quando a resposta do MonkeyLearn é recebida, esta não será imediatamente enviada. Será primeiro tratada através do particionamento da resposta em três partes. O ID, que representa o ID de um requerimento, o título e a descrição do mesmo. A descrição irá ter uma opção para se construir logo o cenário de um requerimento, quando este está presente na descrição. Com esta melhoria na resposta do servidor, torna-se mais fácil a criação dos nós para os diagramas de atividade e sequência.

3/Julho -> Não foi encontrada uma nova ferramenta de classificação. Por enquanto vai continuar a ser usada a que já está implementada. Processar do lado do javascript o ficheiro JSON recebido através do servidor REST.

TODO -> Descobrir como é que se atualiza o HTML a partir do node, visto que as operações todas Document não funcionam em node. Após isto será criar o diagrama para ser preenchido apenas com as setas. Após isto serão usados algoritmos para fazer a criação os casos de teste.

**Realizando esta fase, a primeira versão do projeto estará concluída.**

4/Julho -> Para poder criar os diagramas é preciso uma forma de através do Express js conseguir mostrar as mudanças na páginas html, pois quando inseri-mos um ficheiro, este é tratado e de seguida ele cria um diagrama, este diagrama precisa de ser exibido nesta página html.

O problema neste momento não é criar os elementos no ficheiro html, mas sim passar estas alterações para o ficheiro que é inicialmente mostrado pelo Express.

TODO -> Hoje o objetivo é descobrir uma forma de poder resolver este problema e a partir daqui será criar o diagrama através do mxGraph e fazer upload para a página html.

5/Julho -> O tratamento dos dados foi simplificado. Estava a ser construído um servidor que iria ser usado para receber e processar todas as informações vindas do utilizador.

Para este servidor seria preciso usar Node.js. O grande problema desta solução é não conter as funcionalidades dos browsers, como o document (document.getElementById(…)) e window (window.onload…). Para isso, foram pesquisadas outras alternativas para tratar esta falta de funcionalidades. Foi encontrado um pacote npm (JSDOM) que tratava substituía por completo estas funcionalidades do browser. No entanto, o express (pacote que foi usado previamente para criar o servidor) só suporta ficheiros html estáticos e como iria ser preciso inserir dinamicamente informações na página, as alterações feitas com o JSDOM no ficheiro html não influencia a página previamente definida, o que tornava este inútil para o que era necessário.

Foi então descoberta uma funcionalidade do express que permitia trabalhar com ficheiros dinâmicos. Então surgiu vários conceitos novos da parte do express como “engine”, “template” e de para que servem. Para trabalhar com estes ficheiros dinâmicos existem as chamadas “engines” que renderizam ficheiros html e transformam-nos em outros tipos de ficheiros, onde podemos criar campos de dados e à medida que se vão recebendo novos dados para serem mostrados ao utilizador, estes serão passados para os campos previamente criados. Existem várias engines suportadas pelo express como EJS (Embedded JavaScript templating), Jade e Pug. Estas preparam os seus ficheiros gerados para a qualquer momento receber a informação pretendida, passando um parâmetro comum entre o ficheiro de javascript e os ficheiros gerados.

Esta solução provavelmente seria bastante complexa de ser implementada, visto que ainda se tinha de aprender como todos estes elementos se conjugam.

Assim, para facilitar, o servidor já não vai processar as informações recebidas do servidor REST. Servirá apenas de intermediário entre o cliente e o servidor JAVA. Ou seja, está só a servir o DropzoneJS, o que se torna inútil ter um servidor apenas para esta funcionalidade (**Se sobrar tempo, este “servidor a mais” será tratado**).

TODO -> Processar os requerimentos recebidos do servidor REST e criar o primeiro diagrama e tentar imprimi-lo.

9/Julho -> Foi conseguida na última semana a criação de um diagrama de atividade simples apenas para poder entender como funciona a ferramenta de desenho de esquemas. A ferramenta designa-se por GOJS. Nas diversas pesquisas feitas, esta ferramenta foi a escolhida porque permite de uma forma muito simples, criar os diagramas que são necessários. Nesta ferramenta, cada nó representa um ficheiro JSON onde cada característica deste nó é designada por uma keyword pré-definida pela ferramenta ou pode ser customizada. Assim, a utilização desta ferramenta torna-se bastante fácil porque as características são escritas neste formato e a parte estética do diagrama também é totalmente alterável.

TODO -> Para hoje será criado o primeiro ficheiro Excel com o nome, descrição e passos dos requisitos. Para isso iram ser usados os botões criados na página web para selecionar os diagramas que estão certos. Após esta seleção, todas as informações irão ser enviadas ao servidor REST de novo. É no servidor REST que irá ser criado o ficheiro Excel com os casos de teste.

Não será ainda usada a transformação do diagrama para grafo, pois, os únicos requisitos que estão a ser usados são os que descrevem cenários (hasScenarios). Sendo assim, com esta criação já será possível criar um ficheiro simples.

Quando esta etapa for acabada, será começado os melhoramentos.

Nestes melhoramentos consta:

- Criação de diagramas de sequência e não só de atividade.

- Melhorar os diagramas de atividade já criados com nós de decisão e mais sofisticados, se necessário.

- Usar mais requisitos e não apenas os que têm cenários.

- Utilização de algoritmos que nos permitem passar os diagramas resultantes para grafos.

- Utilizar o algoritmo de pesquisa em profundidade para criar todas as etapas de um caso de teste.

- Melhorar performance e estética do projeto.

10/Julho -> Foi conseguido o envio do JSON com os diagramas selecionados para o servidor REST. O problema agora é adaptar o formato recebido na mensagem.

11/Julho -> Corrigir o JSON para o gson poder transformá-lo em objecto.

TODO -> Após completar a conversão do JSON será usadas as informações deste ficheiro para criar uma nova worksheet de Excel. Esta folha de Excel terá o nome do requisito, a descrição e os passos deste. Quando a criação do ficheiro for conseguida, este será enviado ao servidor JS possibilitando assim o download deste.

12/Julho -> Não houve grande adiantamento, mas tem a ver com o encode do ficheiro Excel.

TODO -> Adicionar o botão de download

16/Julho -> Foi conseguido ainda na semana passada a conclusão desta fase do projeto. O problema que estava a existir consistia em fazer um passo manual no envio do ficheiro Excel criado. O problema para ser resolvido era o seguinte: no momento em que temos todas as informações dos cenários para poder transformar em diagramas de atividade, estes serão escolhidos pelo utilizador, pressionando o botão “Add diagram”. Ou também podem ser rejeitados pressionando o botão “Reject diagram”. No momento em que clicamos no botão para adicionar o diagrama, a informação deste diagrama é inserida numa form-data. Quando já não existirem mais diagramas para serem escolhidos, esta form-data é enviada para o servidor REST com o “Content-type” de “multipart/form-data”. A informação guardada nesta form data está em formato JSON. A form é recebida através de uma inputstream. Esta inputstream é processada de forma a extrair o ficheiro construído previamente com a informação dos diagramas selecionados. Para processar esta informação, foi usada a ferramenta GSON para transformar o ficheiro recebido em vários objectos para tornar mais fácil o uso dos diferentes passos de cada cenário.

Com esta conversão do formato JSON para objeto, foi encontrado um problema que provocou um erro. Quando eram adicionados mais do que um diagrama ao ficheiro JSON, não ocorria qualquer problema, pois o GSON estava a usar um objecto que lê um vetor de diagramas. O problema apareceu no momento em que foi enviado para o servidor apenas um diagrama. O GSON não consegue assumir que um objeto apenas é equivalente a um vetor apenas com uma posição. Por isso, para resolver este problema foi criada uma classe bastante semelhante à usada anteriormente, a única diferença é que suporta um objeto e não um vetor de objetos.

Depois de já estarem todos os objetos criados, foi usada a ferramenta Apache POI para criar uma folha Excel com as informações de cada objeto. No final, esta folha foi guardada no sistema e posteriormente enviada como resposta ao pedido HTTP feito pelo servidor JS.

No inicio foi pensado em receber o ficheiro Excel do servidor JAVA no servidor JS, escrevê-lo na pasta dedicada ao servidor JS manualmente, voltar a carregá-lo para uma variável e enviá-lo para o utilizador do browser. No inicio, foi tentada, mas sem sucesso esta solução porque quando era feita a escrita, o default enconding usado pelo pacote npm file-system do JS não era o mesmo do ficheiro Excel, o que acaba por corromper o ficheiro. Para resolver então este problema da corrupção do ficheiro, foi usada uma funcionalidade disponibilizada pelo pacote npm request, (.pipe()). Esta funcionalidade “cria um flow” de transferências de dados. Funciona da seguinte maneira:

request.get(url).pipe(fs.createWriteStream(path)), é feito um pedido GET via http ao url inserido e os dados recebidos deste pedido são diretamente escritos para o path inserido, tornando a escrita do ficheiro automática e sem problemas de encoding.

Após a escrita do ficheiro, é enviado o url de onde este foi escrito para o utilizador do browser. Quando o ficheiro javascript que trata do utilizador, recebe este url recebido pelo servidor JS, é criado um botão na página html com a tag download, e assim o download é feito quando é pressionado o botão.