

**Pontifícia Universidade Católica do Rio de Janeiro**

**Assistente Virtual para Indicações em Viagens**

**Lucas Hardman Gomes Campos França**

**Primeiro relatório de Projeto Final de Graduação**

**Centro Técnico Científico - CTC**

**Departamento de Informática**

Curso de Graduação em Ciência da Computação

Rio de Janeiro, Setembro de 2017



**Lucas Hardman Gomes Campos França**

**Assistente Virtual para Indicações em Viagens**

Primeiro relatório de Projeto Final, apresentado ao curso de Ciência da Computação da PUC-Rio como requisito parcial para a obtenção de te título de Bacharel em Ciência da Computação.

Orientador: Edmundo Torreão

Rio de Janeiro,

Setembro de 2017.

Índice

[1. Introdução 4](#_Toc498882894)

[2. Situação Atual 5](#_Toc498882895)

[3. Proposta e Objetivo do Trabalho 6](#_Toc498882896)

[4. Plano de Ação 8](#_Toc498882897)

[5. Cronograma 10](#_Toc498882898)

[6. Dicionário de Dados 11](#_Toc498882899)

[7. Especificação de Requisitos 13](#_Toc498882900)

[8. Viabilidade Técnica e Arquitetura de Software 15](#_Toc498882901)

[9. Estudos e Pesquisas 18](#_Toc498882902)

[Parte 1: Preparação do ambiente de desenvolvimento 18](#_Toc498882903)

[Parte 2: Inicialização do projeto e adição das SDK 19](#_Toc498882904)

[Parte 3: Integração com o Facebook 25](#_Toc498882905)

[Parte 4: Conhecendo o Watson Personality Insights 34](#_Toc498882906)

[Parte 4.1: Adicionando o Personality Insights ao projeto 34](#_Toc498882907)

[Parte 4.2: Entendendo o perfil retornado pelo serviço 36](#_Toc498882908)

[Parte 4.3: Interpretando os valores numéricos 41](#_Toc498882909)

[Parte 4.4: Descrição dos modelos de personalidade 41](#_Toc498882910)

[Parte 4.5 Descrição dos modelos de preferência de consumo 41](#_Toc498882911)

[Parte 5: Criação de exemplo de atrações turísticas 41](#_Toc498882912)

[Parte 6: Comparação entre o resultado do Personality Insights e as características das atrações turísticas (Match) 41](#_Toc498882913)

[Parte 7: Conhecendo o Watson Conversation 41](#_Toc498882914)

[10. Definição do Escopo 42](#_Toc498882915)

[11. Modelagem 42](#_Toc498882916)

[12. Casos de Uso 42](#_Toc498882917)

[13. Mockups 42](#_Toc498882918)

[14. Referências Bibliográficas 43](#_Toc498882919)

Introdução

Inicialmente criado pela IBM em 2011 para participar do programa televisivo americano Jeopardy, o Watson é um sistema de computação cognitiva. Desta forma, ele utiliza técnicas de “machine learning” para aprender com experiências anteriores e aplica estes conhecimentos em futuras tomadas de decisão.

O Watson veio para tentar evoluir na solução de um dos grandes problemas da computação: entender e interpretar a linguagem natural. Ele é capaz de entender dados não-estruturados, que representam 80% dos dados encontrados virtualmente em 2017. Estes dados são normalmente informações produzidas por “seres humanos” para “seres humanos”, incluindo textos como artigos, pesquisas e postagens. Ou seja, temos um cenário de um programa computacional tentando interpretar informações com regras como gramática, contexto e cultura, e podem ser ambíguas, implícitas, complexas [1].

A IBM já oferece diversos serviços do Watson que podem ser utilizados por qualquer pessoa com conhecimentos em computação e programação. Dentre eles temos o Conversation e o Personality Insights. O primeiro é um serviço que torna capaz uma conversa por chat entre um usuário “falante” de linguagem natural e um robô que utiliza a inteligência do Watson. O robô deve entender a linguagem natural e responder da forma “mais humana possível”. Já o segundo serviço lê e interpreta os dados retirados de redes sociais como Facebook e Twitter para identificar características de personalidade, necessidades, gostos, valores, preferências e hábitos.

Já podemos contar com aplicações do Watson em diversas áreas, como por exemplo na medicina, onde ele é capaz de recomendar terapias contra o câncer a partir do cruzamento da literatura cientifica com dados clínicos e genéticos dos pacientes [2]. Além dessa área, educação, finanças e “internet das coisas” também são focos da IBM para o Watson.

Dessa forma, o Watson é capaz de ser aplicado em diversos setores, entre eles o turismo. Os negócios desta indústria são normalmente diretos e objetivos, como montagem de pacotes de viagens, reservas de hotéis, aluguéis de carros, compras de passagens aéreas e compra de ingressos. O uso da computação cognitiva pode trazer para estes negócios um entendimento sobre as características e necessidades do cliente e com isso oferecer um serviço mais personalizado para cada cliente.

1. Situação Atual

A IBM possui uma gama de soluções para indústria de turismo e viagens que pode ser encontrada em [IBM.com/Travel](http://ibm.com/travel). A maior parte das companhias deste setor estão implementando soluções em forma de chatbots e robôs, ambas ainda “não maduras” o suficiente para impactar em estratégias significativas dessa indústria ou explorar toda a capacidade da tecnologia. Dessa forma, a maior parte das empresas nesta área estão apenas observando o avanço da computação cognitiva e as poucas que estão investindo lideram projetos pequenos e com baixos riscos[3]. Este é o caso da empresa britânica Thomson, que trabalha no desenvolvimento de um chatbot interage com o usuário providenciando, em tempo real, respostas baseadas em pesquisas sobre destinos e perguntas sobre férias[4]. Por exemplo, um robô conversa com usuário final com o intuito de sugerir as “melhores” férias baseando-se em um maior entendimento sobre o que o usuário está procurando. Outro trabalho similar é o WayBlazer, que implementa inteligência artificial para melhorar e personalizar a experiência do usuário em websites relacionados a turismo e viagens que contratarem o serviço[5]. Neste caso, o serviço pode ser contratado para um website de turismo (por exemplo, Trivago), onde os usuários vão ter o seu comportamento mapeado (cliques, compras, e outros dados relevantes)[6] e assim, com o tempo, o website vai ter um perfil de cada usuário (o que pode ser utilizado para diversos fins, como por exemplo passar a sugerir/facilitar a compra de produtos ou serviços relacionados a este perfil).

Por outro lado, a grande maioria dos serviços disponíveis online para essa indústria funcionam de forma onde o usuário informa o que quer e o sistema responde exclusivamente de acordo com o que foi informado. Ou seja, não há uma análise específica sobre o perfil do usuário que influencie nos resultados. Entre estes sistemas, temos o Hotel Urbano e o Trivago, que possuem a principal funcionalidade de encontrar melhores preços em hotéis. Outro sistema conhecido é o Decolar.com, que permite centralizar no aplicativo todas as informações sobre uma viagem, incluindo tickets de embarques, reservas em hotéis, alugueis de carros e ingressos de atrações turísticas, por exemplo. Nesta mesma linha há várias outras aplicações, bem comuns em forma de websites ou aplicativos para celular.

1. Proposta e Objetivo do Trabalho

Este projeto tem como proposta um estudo sobre computação cognitiva e sobre as APIs do Watson, utilizando um aplicativo mobile que funciona como assistente virtual para indicações em viagens como “prova de conceito” (POC - Proof of Concept). Desta forma, será utilizada uma API de um assistente virtual (“IMB Watson Conversation”) para viabilizar a interação com usuários relativa a seus interesses pessoais sobre viagens. O projeto também utilizará a API "IBM Watson Personality Insights" para obter o perfil do usuário, a partir da leitura e interpretação das suas contas nas redes sociais. Os dados obtidos do "Personality Insights" serão interpretados pelo aplicativo, a ser desenvolvido neste projeto, posicionando os usuários num determinado perfil para que lhes sejam feitas sugestões de acordo com seus gostos, interesses e estilo de vida.

O sistema será implementado em uma plataforma tecnológica ainda a ser definida, podendo ser Android ou iOS, e deverá pedir acesso à conta do Facebook ou do Twitter do usuário. Com as credenciais concedidas, o IBM Watson Personality Insights fará a leitura do perfil do usuário e exportará um arquivo contendo pontuações obtidas em diversas características humanas[7]. Este arquivo deve ser interpretado pelo sistema a ser desenvolvido, que deve guardar informações que identificam as tendências de gostos e interesses do usuário. Estas informações guardadas devem servir como “base” para identificar locais de possível interesse.

Após efetuar o login e garantir o acesso à uma rede social ao sistema, o usuário se encontra em uma sala de chat, onde uma assistente virtual, funcionando através do IBM Watson Conversation[8], inicia uma conversa. Neste dialogo, a assistente procura palavras-chave para identificar quais são os interesses do usuário naquele momento, e com isso começar a fazer sugestões baseadas nas informações armazenadas no sistema. A aceitação ou não do usuário pode alterar as informações persistidas no sistema, fazendo com que o sistema passe a oferecer sugestões melhores no futuro.

Assim como o sistema armazena informações sobre o usuário, ele também deve guardar informações sobre destinos, atrações turísticas, restaurantes, exposições e outros estabelecimentos ou eventos ao redor do mundo. Estas informações devem ser utilizadas pelo sistema a ser implementado comparadas com as informações sobre o usuário para formar as sugestões da assistente.

Desta forma, podemos observar que estamos lidando com um volume muito grande de informações, e ainda não foi especificado em como obter parte delas. Tendo isto em vista, assim como o tempo limitado para a disciplina de Projeto Final, a quantidade de informações no sistema provavelmente será limitada. A princípio a assistente deverá conseguir conversar apenas sobre algumas cidades, e atendendo a determinadas questões.

1. Plano de Ação

Etapa 1: Documentação

* Documentação sobre o estudo realizado e os artefatos do projeto da aplicação
* Atualização do escopo conforme as mudanças necessárias durante a implementação.
* Registro de informações sobre as linhas de código para facilitar futuros retrabalhos e atualizações.
* Atualização da modelagem conforme as necessidades encontradas durante a implementação.

Etapa 2: Estudo sobre as funcionalidades das APIs do Watson

* Leitura e compreensão da documentação das APIs com a finalidade entender o funcionamento da tecnologia que vai ser utilizada.

Etapa 3: Pesquisa

* Pesquisa sobre as tecnologias a serem utilizadas no projeto e como ela se comunicam.
* Pesquisa sobre sistemas similares.

Etapa 4: Especificação de requisitos

* Definição das características que o sistema deve atender.

Etapa 5: Estudo sobre os possíveis ambientes de desenvolvimento

Etapa 6: Definição da arquitetura de software

Etapa 7: Modelagem

* Criação de modelos de dados de representação do sistema.

Etapa 8: Definição do escopo

* Definição clara e objetiva sobre todos os aspectos da aplicação.
* Detalhamento sobre todas as funcionalidades a serem implementadas.

Etapa 9: Inicio da Implementação

* Primeiros passos da implementação do sistema.

Etapa 10: Implementação

* Implementação completa do sistema.

Etapa 11: Testes

* Testes automatizados.
* Testes com usuários.
* Documentação dos testes.
* Elaboração de um documento apresentando os resultados finais.

Etapa 12: Ajustes e finalização

* Ajustes finais.
* Revisão
* Finalização da documentação

1. Cronograma

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Projeto Final** | Ago  17 | Set  17 | Out  17 | Nov  17 | Dez  17 | Jan  18 | Fev  18 | Mar  18 | Mai  18 | Jun  18 | Jul  18 |
| Etapa 1: Documentação |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 2: Estudo sobre as funcionalidades das APIs do Watson |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 3: Pesquisa |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 4: Elaboração da proposta |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 5: Especificação de requisitos |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 6: Viabilidade |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 7: Estudo sobre os possíveis ambientes de desenvolvimento |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 8: Definição da arquitetura de software |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 9: Modelagem |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 10: Definição do escopo |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 11: Inicio da Implementação |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 12: Documentação |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 13: Implementação |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 14: Testes |  |  |  |  |  |  |  |  |  |  |  |
| Etapa 15: Ajustes e finalização |  |  |  |  |  |  |  |  |  |  |  |

1. Dicionário de Dados

* Redes sociais: é uma estrutura social composta por pessoas ou organizações, conectadas por um ou vários tipos de relações, que compartilham valores e objetivos comuns. Estamos lidando apenas com Facebook e Twitter.

**Exemplo:** Facebook, Twitter, LinkedIn.

* Características de personalidade: são indicadores numéricos que representam o quanto um usuário possui de certa característica. Entre várias características, temos: emotividade, ansiedade, imaginatividade, curiosidade....

**Exemplo:** O usuário A tem o indicador de curiosidade igual à 1 (em uma escala de zero à um), o que indica que ele é muito curioso.

* Perfil de usuário: é um conjunto com várias características de personalidade, que juntas podem descrever um usuário.

**Exemplo:** O perfil do usuário A indica que ele é muito criativo e emotivo, porém pouco ansioso.

* Gerente do sistema: é um usuário especial com credenciais de administrador. Este usuário não possui acesso ao código fonte e não pode recompilar o sistema, porém ele tem acesso à partes restritas do programa que podem alterar os dados armazenados no sistema.

**Exemplo:** O gerente A do sistema, adicionou informações sobre um restaurante B da cidade C.

* Atrações: são qualquer tipo de pontos turísticos ou estabelecimentos.

**Exemplo:** Cristo Redentor, Pizza Hut, Praia de Copacabana, Shopping da Gávea, Comuna, mureta da Urca...

* Eventos: são acontecimentos públicos, com dia e hora marcados, que acontecem em locais públicos ou em atrações. Podem ou não ter periodicidade.

**Exemplo:** Festa Treta, fogos de artifício no réveillon de Copacabana, festival de food truck, feiras de artesanato...

* Perfil de atração ou evento: é um conjunto com várias características de personalidade, que juntas podem descrever a maior parte dos usuários que frequentam/gostam da atração ou do evento. Publico alvo.

**Exemplo:** o perfil da atração “trilha da Pedra Bonita” é de pessoas aventureiras, pessoas que gostam de praticar exercício físico...

* Match de compatibilidade: conta o quanto o perfil de atração ou evento é compatível com o perfil do usuário.

**Exemplo:** o usuário possui 75% de match de compatibilidade com o a atração B.

* Assistente virtual: um robô que funciona através da API "IBM Watson Conversation" e conversa de forma mais humana possível com o usuário.

**Exemplo:** o usuário começou uma conversa sobre a sua viagem para o Rio de Janeiro com a assistente virtual a fim de receber dicas sobre o que fazer durante a viagem. A assistente entende o objetivo do usuário e começa a sugerir programas relacionados à conversa atual e ao perfil de usuário.

* Reação do usuário: informação retirada sobre a fala de um usuário sobre uma sugestão da assistente virtual. Essa informação pode ser positiva, neutra ou negativa.

**Exemplo:** Assistente sugere uma festa de música eletrónica e o usuário responde “não gosto de ambiente muito cheios”. Assistente recebe esta informação como uma reação negativa do usuário.

1. Especificação de Requisitos

Requisitos funcionais:

* O sistema deve pedir acesso ao perfil do Twitter ou Facebook do usuário.
* O sistema deve processar as informações obtidas nas redes sociais do usuário toda vez que houver login.
* As informações processadas sobre as redes sociais no momento do login devem ser utilizadas para formar um perfil de usuário.
* O perfil de usuário deve conter contadores (variáveis que podem ser inteiras, float ou booleanas) que medem características de personalidade.
* O sistema deve poder importar de outro sistema informações sobre atrações e eventos.
* O gerente do sistema deve filtrar e selecionar as atrações e eventos importados pelo sistema.
* O gerente do sistema deve preencher características de personalidade mais comuns em um perfil de atração ou evento.
* O sistema deve relacionar o perfil de atração ou evento com o perfil do usuário e criar um match de compatibilidade entre os perfis.
* O sistema deve contar um chat com uma assistente virtual.
* A assistente virtual deve saber sobre qual cidade o usuário quer conversar.
* A assistente virtual deve conversar com o usuário da maneira mais humana possível.
* A assistente virtual deve procurar informações nas falas do usuário para sugerir uma atração ou evento.
* Ao sugerir atração ou evento, a assistente virtual deve interpretar a reação do usuário.
* Quando houver uma reação do usuário, ela deve ser utilizada para efetuar alterações no perfil de atração ou evento, no perfil do usuário e, consequentemente, no match de compatibilidade.

Requisitos não funcionais:

* O sistema deve funcionar em qualquer versão do sistema operacional (iOS/Android) que não foi descontinuada.
* O sistema só deve poder ser acessado após login com senha.
* A assistente virtual deve responder o usuário em menos de 3 segundos.

1. Viabilidade Técnica e Arquitetura de Software

O projeto será implementado para smartphones e utilizará recursos do IBM Watson. Para definir a arquitetura devemos levar em consideração as limitações e necessidades das SDKs do Watson e dos possíveis sistemas operacionais.

1. IBM Watson Conversation:
2. Línguas: o Conversation permite conversas em diversas linguagens naturais, entre elas estão inglês, português e espanhol. A lista completa com todos os idiomas suportados pode ser encontrada na sessão “[Supported languages](https://console.bluemix.net/docs/services/conversation/lang-support.html#supported-languages)” da documentação [9].
3. SDKs: [Java](https://github.com/watson-developer-cloud/java-sdk), [Node.js](https://github.com/watson-developer-cloud/node-sdk), [Python](https://github.com/watson-developer-cloud/python-sdk), [.NET](https://github.com/watson-developer-cloud/dotnet-standard-sdk), [Swift](https://github.com/watson-developer-cloud/swift-sdk) e [Unity](https://github.com/watson-developer-cloud/unity-sdk).
4. Preço:

– FREE:

* 10.000 chamadas à API
* Até 5 workspaces
* Até 25 intents
* Até 25 entities

– STANDARD: R$0,004838 reais por chamada à API

* Até 20 workspaces
* Até 2.000 intents
* Até 1.000 entities
* Cloud pública compartilhada

1. IBM Watson Personality Insights:
2. Línguas: o Personality Insights permite a importação de diversos textos, incluindo os extraídos de redes sociais. Estes textos devem estar em redigidos em uma das “request languages” suportadas, podendo ser árabe, inglês, japonês, coreano e espanhol. Já a resposta gerada pelo Personality Insights possui outro grupo de possíveis linguagens, o “response language”, que tem mais opções como o português. A lista completa com todos os idiomas suportados pode ser encontrada na sessão “[Language support](https://console.bluemix.net/docs/services/personality-insights/user-overview.html#overviewLanguage)” da documentação [10].
3. SDKs: [Android](https://github.com/watson-developer-cloud/android-sdk), [Java](https://github.com/watson-developer-cloud/java-sdk), [Node.js](https://github.com/watson-developer-cloud/node-sdk), [Python](https://github.com/watson-developer-cloud/python-sdk), [Swift](https://github.com/watson-developer-cloud/swift-sdk) e [Unity](https://github.com/watson-developer-cloud/unity-sdk).
4. Preço:

– FREE:

* 1.000 chamadas à API

– TIERED:

* 1 a 100 chamadas à API: free.
* 100 a 100.000 chamadas à API: R$0,041 reais por chamada.
* 100.001 a 250.000 chamadas à API: R$0,0205 reais por chamada.
* 250.001 ou mais chamadas à API: R$0,0103 reais por chamada.

Levando em conta as necessidades da IBM Watson, vimos que podemos construir um aplicativo compatível com diversas linguagens naturais. As mais importantes são inglês, português e espanhol, pois são as mais faladas mundialmente. A língua ideal para implementar seria o português, pois além de ser a língua do nosso país, ficaria muito mais fácil testar o programa com usuários brasileiros. Porém, o Personality Insights não é compatível com idioma na parte de “request language”, o que impossibilita a leitura de redes sociais (como o Facebook e Twitter) em português. Logo, a língua utilizada será o inglês, que é a língua mais falada pela população mundial.

Outra característica importante do sistema é a sua plataforma e a sua linguagem de progamação. Como vimos anteriormente, as APIs utilizadas possuem em comum as seguintes SDKs: Java, Node.js, Python, Swift e Unity. Dentre estas, as melhores para smartphones são Java, para sistema operacional Android, e Swift, para sistema operacional iOS. Deste modo, devemos comparar os prós e contras de cada linguagem para decidir qual utilizar.

Requisitos Java:

* Computador com sistema operacional Windows, Linux ou MacOS
* Eclipse (IDE)
* Android Studio (SDK)

Requisitos Swift:

* Computador com sistema operacional MacOS
* Xcode (IDE)

Os requisitos de ambas linguagens são atendidos pelo desenvolvedor. Desta forma, por uma questão de preferência a linguagem definida fica como Swift.

Sabemos que o aplicativo utiliza informações sobre diversas atrações e eventos, e que estes dados são importados de outras aplicações e podem ser modificados por gerentes do sistema. Ou seja, estes dados são comuns a todos usuários. Para que isto seja possível, eles devem ficar armazenados em um banco de dados de um servidor (pensando como fazer isso ainda).

|  |  |
| --- | --- |
| Linguagem do aplicativo | Swift 4 |
| Sistema Operacional do aplicativo | iOS 11 |
| Sistema Operacional do ambiente de desenvolvimento | MacOS High Sierra | |
| IDE | Xcode 9 |
| Integração com o Watson | https://github.com/watson-developer-cloud/swift-sdk |
| Integração com o Facebook | https://github.com/facebook/facebook-sdk-swift |

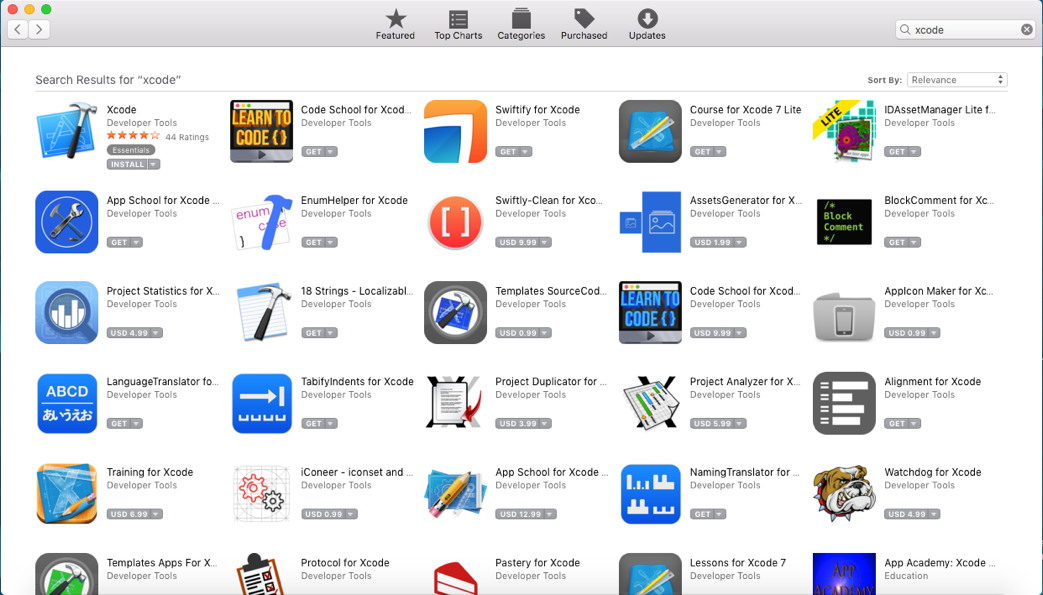
1. Estudos e Pesquisas

Nesta sessão vamos utilizar o passo-a-passo da criação de um protótipo do projeto para aprender as funcionalidades do Watson, a utilizar as SDK do Watson e do Facebook, e a juntar todo esse conhecimento para a criação de um aplicativo iOS.

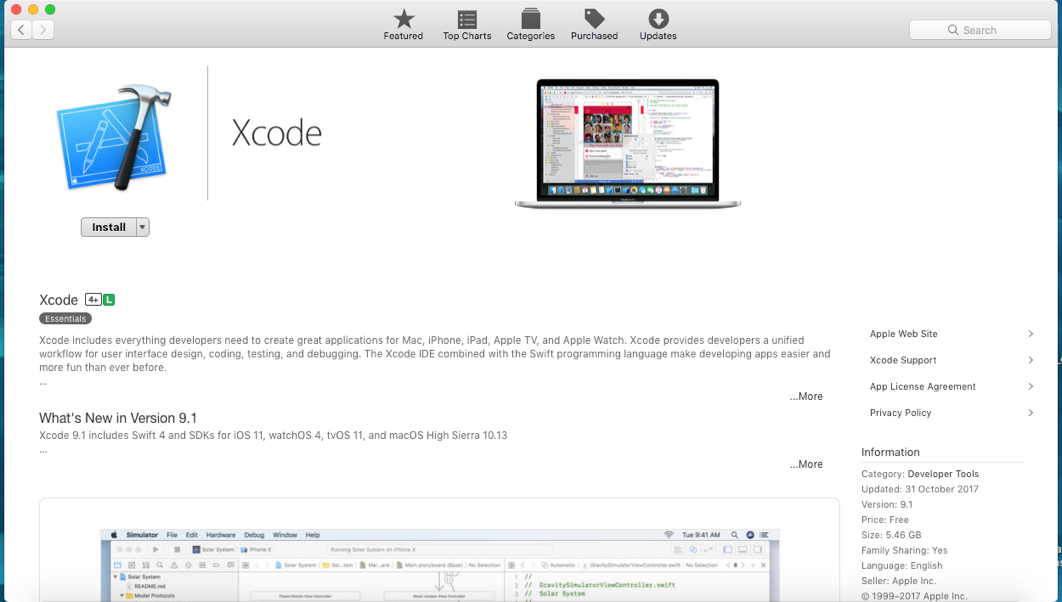
## Parte 1: Preparação do ambiente de desenvolvimento

* Passo 1: Instalação da última versão do Xcode.

1. Abrir o App Store
2. Digitar “xcode” no campo de pesquisa
3. Selecionar o aplicativo Xcode



1. Apertar o botão “Install”



* Passo 2: Instalação do Carthage

1. Abrir o Terminal
2. Digitar o seguinte código para instalar o Homebrew

|  |
| --- |
| /usr/bin/ruby -e "$(curl –fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" |

1. Digitar o seguinte código para instalar o Carthage

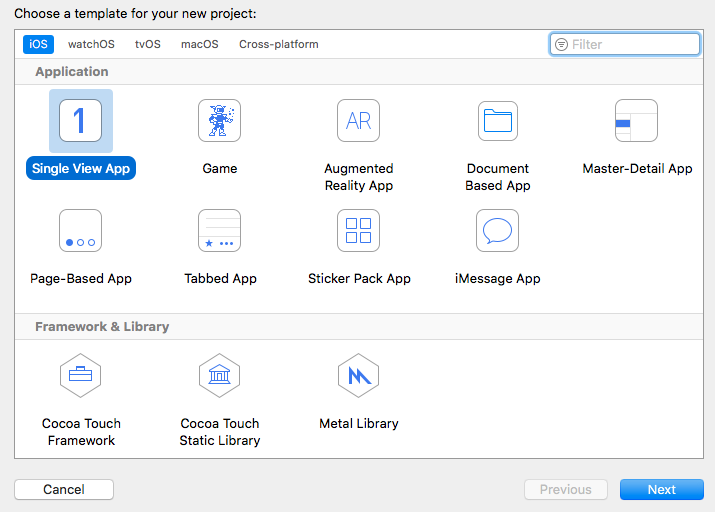
|  |
| --- |
| brew install carthage |

Parte 2: Inicialização do projeto e adição das SDK[11]

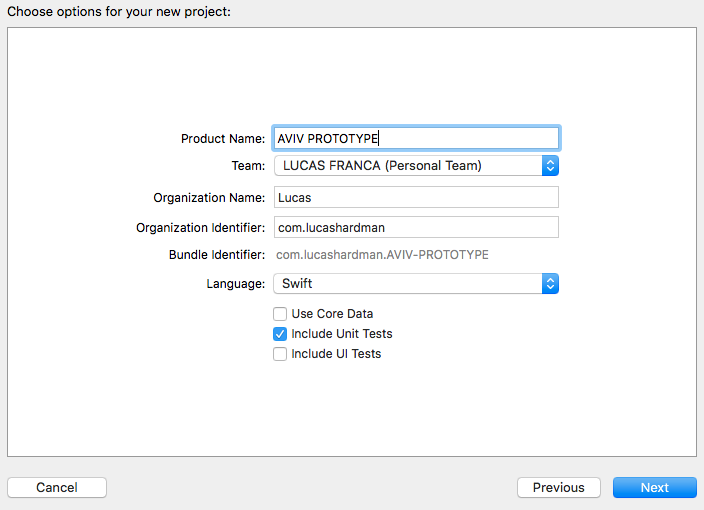
1. Abrir o Xcode
2. Criar um novo projeto selecionando “Create a new Xcode project”.



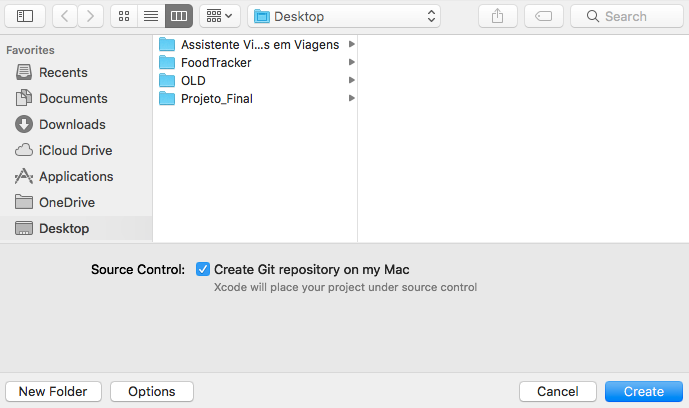
1. Selecionar iOS
2. Selecionar “Single View App”



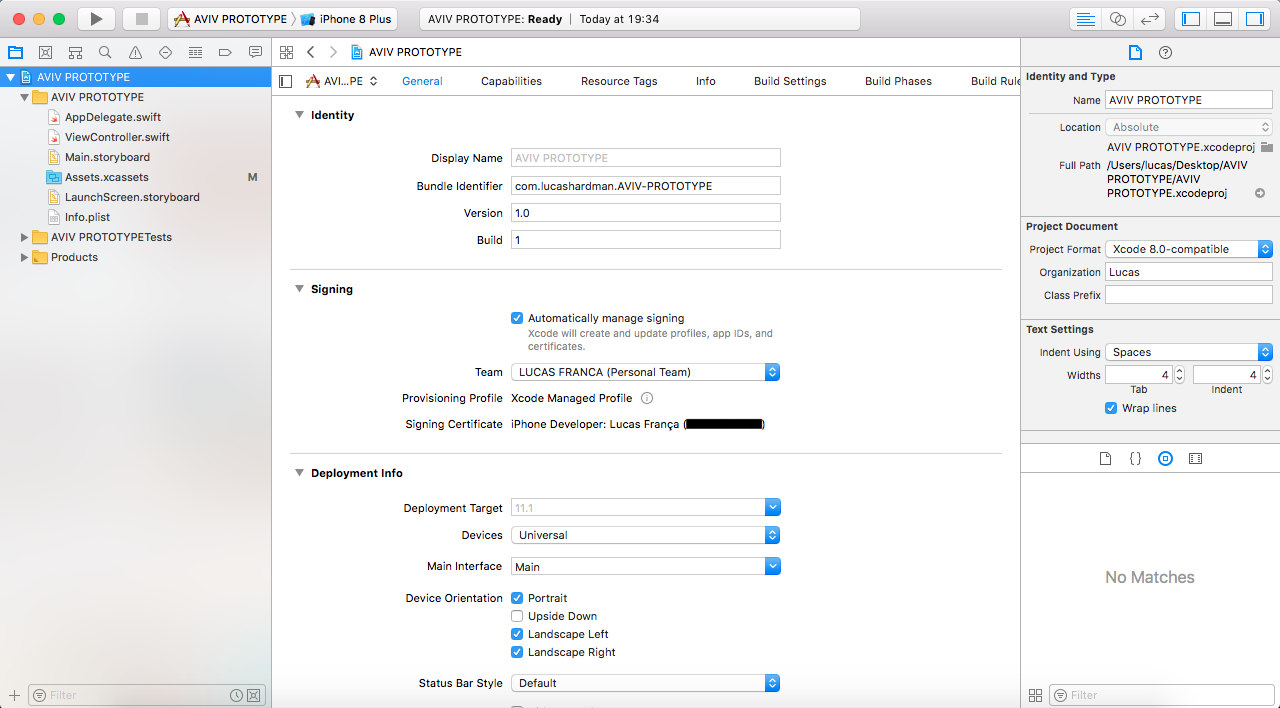
1. Na próxima janela, crie um nome para o projeto e adicione um team (caso tenha)



1. Para finalizar, escolha um diretório para ficar a pasta com o projeto.



O Xcode irá inicializar o projeto com a seguinte configuração:



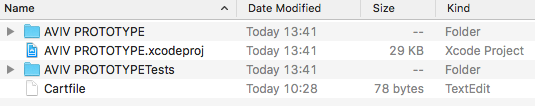
A coluna da esquerda é chamada de Navegator Area e através dela é possível acessar os arquivos do projeto. Conforme necessidade, vamos ver o significado de cada arquivo. A coluna da direita é chamada de Utility Area, e a coluna do meio é chamada de Editor Area.

Agora já temos um projeto criado e precisamos adicionar as frameworks das SDKs do Watson e do Facebook.

1. Abrir uma nova janela do Finder e navegar até a pasta do projeto.
2. Criar um novo arquivo chamado “Cartfile” e adicionar os seguintes comandos.

|  |
| --- |
| github "facebook/facebook-sdk-swift"  github "watson-developer-cloud/swift-sdk" |

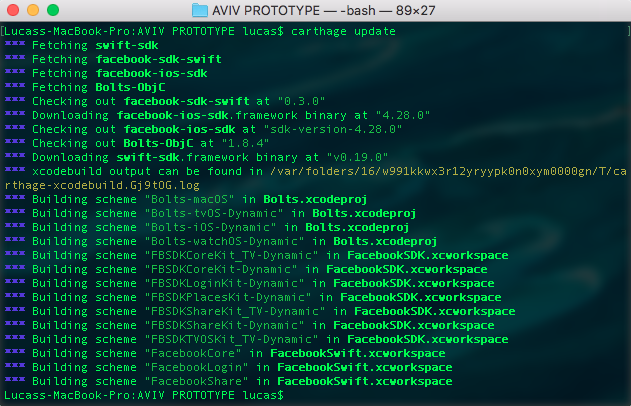
A pasta do projeto deverá ficar assim:



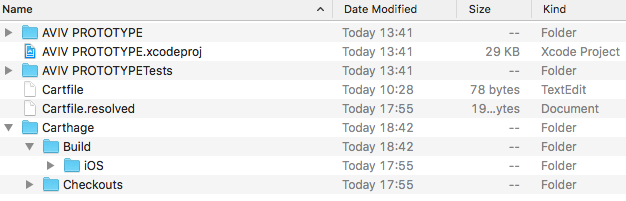
1. Abrir o Terminal e navegar até a pasta do projeto.
2. Digitar o seguinte comando para adicionar as frameworks.

|  |
| --- |
| carthage update |

O Carthage vai começar a baixar e adicionar as frameworks e se não houver erro, no final o terminal deverá estar da seguinte forma.

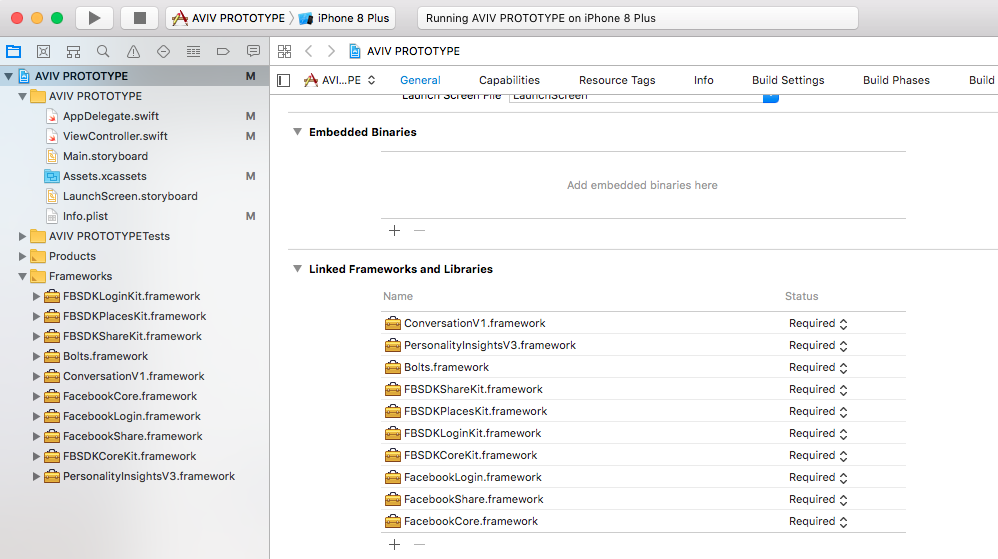


No final do processo, a pasta do projeto deverá ficar assim:



As frameworks se encontram na pasta Carthage/Build/iOS

1. Voltando para o Xcode, selecione o nome do projeto no Navegator Area
2. Selecione a aba General
3. Selecione “Linked Frameworks and Libraries”
4. Selecione “+” e “Add Other”
5. Selecione as seguintes frameworks: PersonalityInsightsV3.framework, ConversationV1.framework, Bolts.framework, FBSDKCoreKit.framework, FBSDKLoginKit.framework, FBSDKPlacesKit.framework, FBSDKShareKit.framework, FacebookCore.framework, FacebookShare.framework e FacebookLogin.framework



1. Selecione a aba Build Phases
2. Selecione “+” e depois “New Run Script Phase”
3. Selecione “Run Script”
4. Preencha o campo Shell com:

|  |
| --- |
| /bin/sh |

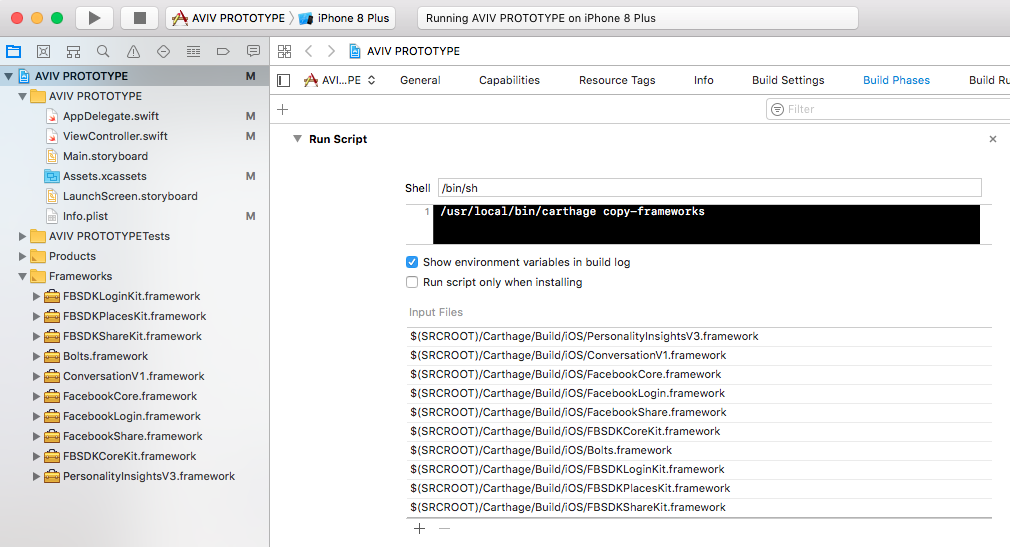
1. Preencha o campo abaixo de Shell com:

|  |
| --- |
| /usr/local/bin/carthage copy-frameworks |

1. Marque a opção “Show environment variables in build log”
2. Desmarque a opção “Run script only when installing”
3. Preencha o campo Input Files com:

|  |
| --- |
| $(SRCROOT)/Carthage/Build/iOS/PersonalityInsightsV3.framework  $(SRCROOT)/Carthage/Build/iOS/ConversationV1.framework  $(SRCROOT)/Carthage/Build/iOS/FacebookCore.framework  $(SRCROOT)/Carthage/Build/iOS/FacebookLogin.framework  $(SRCROOT)/Carthage/Build/iOS/FacebookShare.framework  $(SRCROOT)/Carthage/Build/iOS/FBSDKCoreKit.framework  $(SRCROOT)/Carthage/Build/iOS/FBSDKCLoginKit.framework  $(SRCROOT)/Carthage/Build/iOS/FBSDKPlacesKit.framework  $(SRCROOT)/Carthage/Build/iOS/FBSDKShareKit.framework  $(SRCROOT)/Carthage/Build/iOS/Bolts.framework |

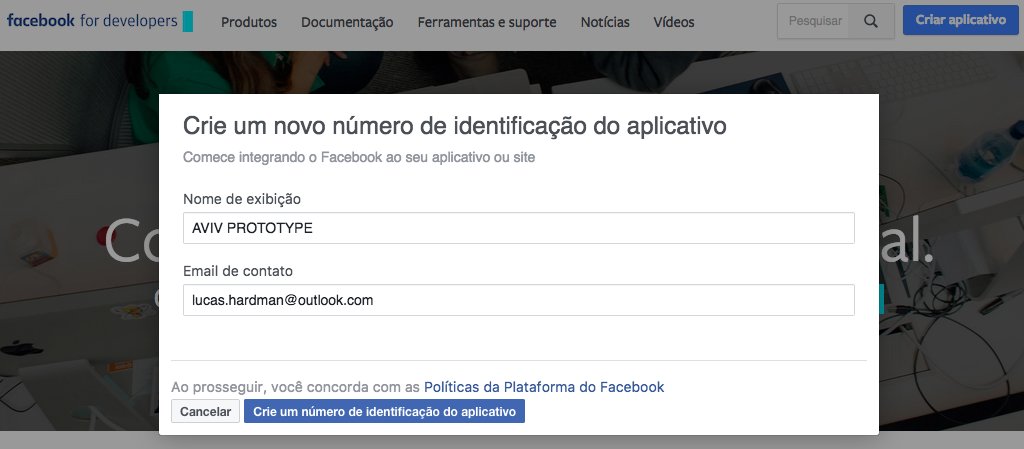
Ao final a sessão “Script Run” deverá ficar assim:



Neste ponto já é possível compilar e rodar o programa sem gerar erros. Para isto selecione o botão de play no canto superior esquerdo do Xcode. O simulador irá iniciar, porém irá apresentar apenas uma tela branca neste momento.

## Parte 3: Integração com o Facebook

1. Acessar o website <https://developers.facebook.com>
2. Clicar em “Criar Aplicativo”
3. Digitar um nome de exibição
4. Digitar um e-mail de contato
5. Clicar em “Crie um número de identificação do aplicativo”



1. Envie a verificação de segurança
2. Será listado vários produtos, clique em “Login do Facebook”
3. Será oferecido um tutorial de início rápido, siga as instruções até o passo 5.

Nesse ponto o seu projeto já está completamente integrado com a SDK do Facebook. No passo 5 do tutorial anterior foi adicionado um trecho de código xml ao arquivo Info.plist, este trecho contém o nome de exibição e o número de identificação do aplicativo criado na conta de desenvolvedor do Facebook.

Nosso objetivo com o Facebook é coletar publicações do usuário e utilizar esses dados para enviar para o Watson Personality Insights. Então o primeiro passo é criar um botão que possibilite o login na conta da rede social.

1. Entre no arquivo AppDelegate.swift. Este é o arquivo responsável por mudanças de estado do aplicativo. Importe a biblioteca FBSDKCoreKit e procure a função application e reescreva-a da seguinte forma:

|  |
| --- |
| func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {  // Override point for customization after application launch.  FBSDKApplicationDelegate.sharedInstance().application(application, didFinishLaunchingWithOptions: launchOptions)  return true  }  func application(\_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey: Any] = [:]) -> Bool {  // Override point for customization after application launch.  let handled = FBSDKApplicationDelegate.sharedInstance().application(app, open: url, sourceApplication: options [UIApplicationOpenURLOptionsKey.sourceApplication] as! String!, annotation: options [UIApplicationOpenURLOptionsKey.annotation])  return handled  } |

Assim temos duas versões da função. Estas funções são responsáveis por manter o funcionamento do aplicativo enquanto o login é efetuado. Isto é necessário pois para efetuar o login é necessário sair e voltar ao aplicativo novamente.

1. Entre no arquivo Main.storyboard. Aqui é possível “desenhar” o seu aplicativo, colocando cada peça no local desejado de forma visual.
2. Na parte inferior da Utility Area selecione “Show the Object Library”
3. Procure por “Container View” e arraste para a Editor Area dentro de “View Controller Scene > View Controller > View”
4. Selecione o Container View adicionado e selecione “Add New Constraints”
5. Adicione 5 no topo e 250 na direita, 16 na esquerda e 615 em baixo, depois selecione “Add 4 Constraints”
6. Selecione “Show the Identity inspecot”
7. No campo Class digite “LoginButton” e selecione “Inherite Module From Target”
8. Selecione “Show the Assistant editor”
9. Pressione a tecla “control” e segure-a enquanto arrasta o Container View para o ViewController.swift, exatamente acima do método viewDidLoad()
10. Digite o nome “facebookLoginButton”, deixe o resto das opções como está e clique em “Connect”. Isso irá adicionar a seguinte linha de código no ViewController.Swift:

|  |
| --- |
| @IBOutlet weak var facebookLoginButton: LoginButton! |

Agora que adicionamos uma Container View, vamos adicionar o botão de login do Facebook nela.

1. Entre no arquivo ViewController.swift. Aqui é possível alterar as views do aplicativo.
2. Importe a biblioteca FacebookLogin
3. A classe ViewController possui o método viewDidLoad, que possibilita adicionar novas funcionalidades imediatamente após a view ser carregada. Adicione o botão utilizando o método LoginButton.
4. Adicione o botão criado ao Container View criado utilizando o método addSubview(). Observe o código a seguir para ver os detalhes.

|  |
| --- |
| import UIKit  import FacebookLogin  class ViewController: UIViewController {    // MARK: Properties  @IBOutlet weak var facebookLoginButton: LoginButton!    // MARK: Override functions  override func viewDidLoad() {  super.viewDidLoad()    let temp = LoginButton(readPermissions: [ .publicProfile, .userPosts, .email ])  temp.frame = facebookLoginButton.bounds  facebookLoginButton.addSubview(temp)  }  } |
|  |

Agora nosso usuário já pode efetuar o login no Facebook. Com as permissões de leitura cedidas, temos acesso ao perfil público (o que contém informações como nome e aniversário), ao e-mail às postagens.

O próximo passo é acessar as postagens do usuário (é importante enfatizar que precisamos acessar os posts, e não timeline ou o feed). Isso se deve ao fato de que o feed e a timeline contém postagens de outras pessoas, como amigos e empresas por exemplo, e não queremos confundir o Personality Insights com textos de terceiros.

Para acessar as postagens vamos utilizar o método FBSDKGraphRequest que retorna um objeto do tipo Any, que vamos chamar de result. Este objeto pode ser convertido para string utilizando result.debugDescription. O objeto result contém as postagens do usuário, e são elas que precisamos enviar para o Personality Insights.

Porém, o Personality Insights só aceita os seguintes tipos de texto: plain text, HTML, JSON e CSV, e o nosso objeto result não é de nenhum destes tipos. Precisamos então efetuar uma conversão [10]. Os tipos de texto mais acessíveis em Swift são o plain text e o JSON, mas qual é a melhor opção?

O formato JSON é o melhor formato para o conteúdo de redes sociais que consistem de várias postagens e textos, como o Facebook e o Twitter. Outra opção seria concatenar todas as postagens, criando uma grande string no formato plain text. Mas o formato JSON possui a vantagem de ter os textos separados, e assim o Personality Insights pode saber quais textos estão relacionados.

Então é necessário criar um método para converter a string result.debugDescription para JSON. Para isto, criei uma classe AVIVUtilities com o método convertFacebookResultToJSON. Este método recebe uma string result (result.debugDescription) e retorna uma string inputJSON, que é a string no formato JSON pronta para ser utilizada pelo Personality Insights. Não irei entrar nos detalhes deste método neste tutorial. Mas é importante saber como é o fomato das strings que estamos trabalhando.

A seguir temos um exemplo com duas postagens da string retornada pelo FBSDKGraphRequest (result.debugDescription):

|  |
| --- |
| Optional({  data = (  {  "created\_time" = "2017-11-14T22:05:13+0000";  id = "10155060941237546\_10155057821802546";  message = "Pedro Gomes e Clara Malizia combinando de ir pro teatro sabado";  story = "Lucas Hardman shared Desiludindo S/A's post.";  },  {  "created\_time" = "2017-11-06T17:28:13+0000";  id = "10155060941237546\_10155038260237546";  story = "Lucas Hardman shared Nintendo Switch's video.";  }  );  paging = {  next = "https://graph.facebook.com/v2.11/10155060941237546/posts?limit=5&format=json&access\_token=EAAOxWzzApBUBAFApxII2bFF6VTGu7ZCg3qq5PUg62v05RuY5FdtDUWqZChwHmn0gHU8vseZCKpWJOtdUSfMFahFL6rtj783uZAP2V1XRexZAZA8k6o7CZB4B3hmGW9u2oZA2T5heZADplu05yyyZBnBhsVZCsYKfRVWpuUULGo7NPTSVtYk5c05tFyE0epjIROIM8TQanjAsKFAwzSSL1hLxOW0jHPx1uFtGpfsllhuPvLD0QZDZD&until=1509989293&\_\_paging\_token=enc\_AdDUFB19DZCwi49Qpf9tmAKllalZAhcgP4Y2pn9h1bBkgeK9EbZAKH8eP7ByKUfwBoP0jtD4gDfS2zQKDlAx58pgr4f";  previous = "https://graph.facebook.com/v2.11/10155060941237546/posts?limit=5&format=json&since=1510697113&access\_token=EAAOxWzzApBUBAFApxII2bFF6VTGu7ZCg3qq5PUg62v05RuY5FdtDUWqZChwHmn0gHU8vseZCKpWJOtdUSfMFahFL6rtj783uZAP2V1XRexZAZA8k6o7CZB4B3hmGW9u2oZA2T5heZADplu05yyyZBnBhsVZCsYKfRVWpuUULGo7NPTSVtYk5c05tFyE0epjIROIM8TQanjAsKFAwzSSL1hLxOW0jHPx1uFtGpfsllhuPvLD0QZDZD&\_\_paging\_token=enc\_AdDWZCrRwlP0FhuiRwWAynoo2A97aq4zkPIipxG7EzfZB77yCbL8Rysf2ncaFw8PlxYDqmZAEo9LL4FU3KINkM3efve&\_\_previous=1";  };  }) |

A seguir temos um exemplo de JSON aceito pelo Personality Insights:

|  |
| --- |
| {  "contentItems": [  {  "content": "Wow, I liked @TheRock before, now I really SEE how special he is. The daughter story was IT for me. So great! #MasterClass",  "contenttype": "text/plain",  "created": 1447639154000,  "id": "666073008692314113",  "language": "en"  },  {  "content": ".@TheRock how did you Know to listen to your gut and Not go back to football? #Masterclass",  "contenttype": "text/plain",  "created": 1447638226000,  "id": "666069114889179136",  "language": "en"  },  ]  } |

A estrutura desse JSON precisa necessariamente seguir o modelo definido pelo objeto Content, que é um array de objetos do tipo ContentItem. O único atributo obrigatório de ContentItem é o “content” que é o texto que será analizado pelo Watson.

Vamos criar uma classe InputMessages para armazenar um array de strings que será utilizado no input do Personality Insights. Porque fazer uma classe e não uma variável global? Este input será modificado diversas vezes, por diversos módulos do programa. Criar esta classe possibilita adicionar um observer e facilitar o acesso em tempo real.

|  |
| --- |
| import Foundation  class InputMessages: NSObject {    @objc dynamic var input: [String]?    init() {  super.init()  self.input = []  }  } |

Voltando para o nosso ViewController.

1. Vamos adicionar a classe InputMessages às propriedades do nosso ViewController.

|  |
| --- |
| // MARK: Properties  @IBOutlet weak var facebookLoginButton: LoginButton!  @objc var inputMessages = InputMessages.init(text: "inputMessages is empty") |

1. Na função viewDidLoad() vamos adicionar um observer ao inputMessages.

|  |
| --- |
| // Add an Observer to inputMessages  self.inputMessages.addObserver(self, forKeyPath: #keyPath(InputMessages.input), options: .new , context: &inputContext) |

1. Reescrever o método observeValue na classe ViewController para printar no console a string inputMessages.input e assim podermos observar os resultados.

|  |
| --- |
| override func observeValue(forKeyPath keyPath: String?, of object: Any?, change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {    if context == &inputContext{  print("-> InputMessage has changed...\nNew value:\n")  if let newInput = change?[.newKey] as? [String] {  print(newInput)  }  }  } |

1. Voltando para o método viewDidLoad(), finalmente vamos chamar o FBSDKGraphRequest para obter as postagens do usuário.

|  |
| --- |
| FBSDKGraphRequest(graphPath: "/me/posts", parameters: ["fields": "", "limit": 250]).start {  (connection, result, err) in    if err != nil{  print("Failed to start graph request: ", err!);  return  }  self.inputMessages.input = utilities.convertFacebookResultToJSON(result.debugDescription)  } |

Em graphPath, mandamos o endereço das postagens do usuário. Nos parâmetros fields, devemos informar quais informações das postagens queremos (por exemplo: id, created\_time...). Mas no nosso caso, desejamos apenas as mensagens, então não precisamos enviar nenhum parâmetro deste tipo. Já o parâmetro limit (que é opcional), devemos informar a quantidade de postagens que desejamos ler. Por default o limit é um número baixo, então temos que enviar um numero bem alto. Isso se deve ao fato de que o Personality Insights precisa de pelo menos 600 palavras para fornecer dados mais precisos.

Ao compilar e rodar o aplicativo no simulador, podemos observar os resultados no console do Xcode. Esses resultados foram enviados através do print(inputMessages.input) que escrevemos no observer.

A integração do Facebook com o aplicativo já terminou. Agora precisamos criar usuários de teste. Para isto, vamos criar:

* 3 usuários com perfis diferentes.
* Cada usuário será um conhecido. É importante que sejam pessoas diferentes, pois o Personality Insights reconhece as características das pessoas a partir do texto.
* Cada usuário deverá possui várias postagens, para prover informação suficiente para o Watson.

Para isto siga os seguintes passos:

1. Acesse <https://developers.facebook.com/>
2. Selecione “Meus aplicativos”
3. Selecione “AVIV PROTOTYPE”
4. Selecione “Funções”
5. Selecione “Usuários de Teste”
6. Já vai ter um usuário criado. Vamos adicionar mais dois.
7. Selecione “Adicionar”
8. Selecione “2” em “Número de usuários a serem criados”
9. Selecione “Criar usuários de teste”
10. Altere os nomes conforme a sua vontade



## Parte 4: Conhecendo o Watson Personality Insights

### Parte 4.1: Adicionando o Personality Insights ao projeto

Para continuar precisamos ter uma conta no IBM Cloud para utilizar os diversos serviços fornecidos pelos seus servidores. Após criar sua conta, siga os seguintes passos:

1. Acesse o seu dashboard: <https://console.bluemix.net/dashboard>
2. Acesse o menu no canto superior esquerdo
3. Selecione “Watson” no final da lista
4. Selecione “Wason Services > Browse Services”
5. Selecione “Personality Insights”
6. Selecione “Add Services”
7. Preencha os campos e selecione “Create Project”
8. Selecione “Service credentials”
9. Selecione “View credentials ” e anote o seu usuário (username) e a sua senha (password)

Voltando para o Xcode, acesse a sua classe ViewController. Vamos adicionar a ela o serviço do Personality Insights.

1. Importe a framework PersonalityInsightsV3

|  |
| --- |
| import PersonalityInsightsV3 |

1. Adicione o Personality Insights às propriedades da classe ViewController

|  |
| --- |
| private let personalityInsights = PersonalityInsights(username: "digite o usuário aqui", password: "digite a senha aqui", version: "digite a data de hoje aqui") |

1. Crie uma função printInput que recebe um array de strings, converte ele para array de ContentItem e passa esse array como parâmetro para o método getProfile do Personality Insights.

|  |
| --- |
| // MARK: Functions  func printInput (text: [String]){    let error = { (error: Error) in print(error) }    var content: [ContentItem] = []    // Passa os valores de text para content  for t in text{  print ("\n",t,"\n")  content.append(ContentItem.init(content: t))  }    personalityInsights.getProfile(fromContentItems: content, failure: error){ profile in  for big\_five in profile.personality{  print("\n-> ",big\_five.name,": ",big\_five.percentile)  for facet in big\_five.children!{  print(facet.name,": ",facet.percentile)  }  }  }  } |

1. Chame a função criada no observer.

|  |
| --- |
| override func observeValue(forKeyPath keyPath: String?, of object: Any?, change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {    if context == &inputContext{  print("-> InputMessages has changed...\nNew value:\n")  if let newInput = change?[.newKey] as? [String] {  printInput(text: newInput)  }  }  } |

Na prática o input do método getProfile difere da documentação. Não precisamos nos preocupar com tratamentos especiais de JSON, pois o Watson já interpreta o vetor de ContentItem como um.

### Parte 4.2: Entendendo o perfil retornado pelo serviço

A função printInput por enquanto está apenas imprimindo no console o valor de profile convertido para string. O objeto Profile é o objeto de maior nível do JSON retornado pelo serviço. Este objeto possui as seguintes propriedades:

* **word\_coun**t**:** inteiro que representa a quantidade de palavras lidas do arquivo de entrada. Este inteiro pode conter um valor menor do que o valor real de palavras do arquivo, caso este arquivo seja muito grande.
* **processed\_language:** string que representa o nome da linguagem natural que o serviço utilizou para processar o arquivo de entrada. Por exemplo: en (inglês) ou es (espanhol).
* **personality:** array recursivo de objetos Trait que descreve cada característica (Big Five dimensions e facets) obtida do arquivo de entrada.
* **needs:** array de objetos Trait que descreve cada necessidade (Needs) obtida do arquivo de entrada.
* **values:** objeto de Trait que descreve cada valor (Values) obtido do arquivo de entrada.
* **behavor:** array de objetos Behavor que descreve a distribuição da informação gerada pelos dias das semanas e horas dos dias. Este campo só é retornado caso o arquivo JSON de entrada possua um campo de horário de data.
* **consumption\_preferences:** array de objetos ConsumptionPreferencesCategory que fornecem resultados para cada categoria de preferência de consumo. O retorno deste campo é opcional e para acontecer o parâmetro consumption\_preferences deve ser setado como true.
* **warnings:** array de objetos Warning que geram mensagens de alerta associadas ao arquivo de entrada. Caso não haja alertas o array é vazio.

|  |
| --- |
| {  "word\_count": 15223,  "processed\_language": "en",  "personality": [  . . .  ],  "needs": [  . . .  ],  "values": [  . . .  ],  "behavior": [  . . .  ],  "consumption\_preferences": [  . . .  ],  "warnings": []  } |

O objeto Profile sempre inclui arrays de objetos Trait que representam Personality, Needs e Values. Para Needs e Values, estes arrays possuem apenas um nível, contendo em cada posição informações sobre uma necessidade ou valor. Já para o Personality, o array é duplo e representa as características Big Five. Cada característica Big Five possui um campo children que representa um novo array de objetos Trait, e em cada posição possui uma característica específica de cada Big Five. Cada posição do array de objetos Trai possui os seguintes campos:

* **trait\_id**: id único, em forma de string, para cada característica. Este id possui os formatos abaixo, trocando “characteristic” pelo nome da característica.
  + **big5\_characteristic:** dimensões Big Five. Ex: big5\_openess.
  + **facet\_characteristic:** características da dimensão Big Five. Ex: facet\_altruism.
  + **need\_characteristic:** necessidades Need. Ex: need\_challenge.
  + **value\_characteristic:** valores Value. Ex: value\_conservation.
* **name**: uma string legível pelo usuário contendo o nome da característica.
* **category**: categoria da característica. Pode ser uma das três opções abaixo:
  + **personality:** para dimensões Big Five ou facets.
  + **needs:** para necessidades.
  + **values:** para valores.
* **percentile:** um número do tipo double que contém a pontuação da característica em percentil.
* **raw\_score:** é um número do tipo double que contem a pontuação bruta da característica. Este campo é opcional, só retorna caso o parâmetro raw\_scores seja true na chamada do método getProfile().
* **siginificant:** é um booleano que indica se a característica tem valor para o arquivo de entrada. Sempre é true para inglês, espanhol e japonês. Mas pode ser false para as línguas árabe e coreano, que ainda possuem certa limitação no serviço.
* **children:** é um array de objetos Trait que fornece resultados detalhados para cada característica (facet) de cada dimensão Big Five. Só é retornado quando o trait\_id for uma big5\_characteristic

|  |
| --- |
| {  . . .  "personality": [  {  "trait\_id": "big5\_openness",  "name": "Openness",  "category": "personality",  "percentile": 0.8011555009553,  "raw\_score": 0.77565404255038,  "significant": true,  "children": [  {  "trait\_id": "facet\_adventurousness",  "name": "Adventurousness",  "category": "personality",  "percentile": 0.89755869047319,  "raw\_score": 0.54990704031219,  "significant": true  },  . . .  ]  },  {  "trait\_id": "big5\_conscientiousness",  . . .  },  {  "trait\_id": "big5\_extraversion",  . . .  },  {  "trait\_id": "big5\_agreeableness",  . . .  },  {  "trait\_id": "big5\_neuroticism",  . . .  }  ],  "needs": [  {  "trait\_id": "need\_challenge",  "name": "Challenge",  "category": "needs",  "percentile": 0.67362332054511,  "raw\_score": 0.75196348037675,  "significant": true  },  . . .  ],  "values": [  {  "trait\_id": "value\_conservation",  "name": "Conservation",  "category": "values",  "percentile": 0.89268222856139,  "raw\_score": 0.72135308187423,  "significant": true  },  . . .  ],  . . .  } |

Caso o JSON de entrada contenha marcadores de tempo (dia e hora), o objeto Profile irá conter um campo com um objeto do tipo Behavior para cada dia da semana e cada hora do dia. Este objeto inclui:

* **trait\_id:** id único, em forma de string, para cada característica. Este id possui os formatos abaixo, trocando “day” pelo dia da semana e “hour” pelo horário do dia.
  + **behavior\_day:** para dias da semana. Ex: behavior\_sunday.
  + **behavior\_hour:** para horas do dia: Ex: behavior\_0100, 1hora da manhã.
* **name:** string legível pelo usuário contendo o dia da semana ou horário do dia. Ex: “Sunday”, ou “0:00 am”
* **category:** string contendo acategoria da característica, sempre será “behavior”
* **percentage:** um número double que representa a percentagem de objetos ContentItems que ocorreram durante certa hora do dia ou dia da semana.

Para o projeto em questão o objeto Behavior não é importante, então não há preocupação em adicionar marcadores de dias e horas no JSON de entrada.

|  |
| --- |
| {  . . .  "behavior": [  {  "trait\_id": "behavior\_sunday",  "name": "Sunday",  "category": "behavior",  "percentage": 0.21392532795156  },  . . .  ],  . . .  } |

Como vimos anteriormente, ao fazer o requerimento de um Profile, podemos passar como parâmetro true no campo consumption\_preferences, e assim receber um objeto ConsumptionPreferencesCategory que detalha as preferências de consumo do usuário. Este objeto contém:

* **consumption\_preference\_category\_id:** string contendo um id único para cada categoria de preferência de consumo. Ex: “consumption\_preferences\_shopping”
* **name:** string contendo o nome da categoria de forma legível para o usuário.
* **consumption\_preferences:** um array de objetos do tipo ConsumptionPreferences, que contém resultados detalhados de cada preferência de consumo do usuário.

Algumas categorias de preferência de consumo possuem apenas uma preferência, outras possuem várias. Cada preferência é detalhada a partir de um objeto ConsumptionPreferences, que contém os seguintes campos:

* **consumption\_preference\_id:** string contendo um id único para cada preferência de consumo. Ex: "consumption\_preferences\_automobile\_ownership\_cost".
* **name:** string contendo o nome da categoria de forma legível para o usuário. Ex: Likely to be sensitive to ownership cost when buying automobiles".
* **score:** número double que contem a pontuação que indica a probabilidade do usuário preferir tal item.

|  |
| --- |
| {  . . .  "consumption\_preferences": [  {  "consumption\_preference\_category\_id": "consumption\_preferences\_shopping",  "name": "Purchasing Preferences",  "consumption\_preferences": [  {  "consumption\_preference\_id": "consumption\_preferences\_automobile\_ownership\_cost",  "name": "Likely to be sensitive to ownership cost when buying automobiles",  "score": 0  },  . . .  ]  },  . . .  } |

### Parte 4.3: Interpretando os valores numéricos

Neste ponto já estudamos o objeto Profile completo e sabemos que ele detalha resultados em objetos do tipo Trait (personalidade: Big Five e facet), Behavior (comportamento) e ConsumptionPreferences (preferências de consumo). Todos estes objetos detalham características da personalidade do usuário com indicadores numéricos do tipo double.

1. **Percentil de características de personalidade (*percentile*):**

Para cada requerimento, o serviço fornece uma pontuação através do atribulo *percentile* para cada característica de personalidade Big Five, Needs e Values. Esta pontuação representa um ranking de percentagem (variando de 0 a 1) para cada característica baseado no texto de entrada, e é normalizada através de uma comparação com dados obtidos de uma amostra da população.

Por exemplo, o *percentile* 0.64980796071382 da característica big5\_extraversion indica que o autor pontuou esta característica mais do 64% população e menos do que 34%.

1. **Valores brutos de características de personalidade (*raw\_score*):**

Os valores brutos *raw\_score* são os mesmos que os *percentile*, porém obtidos sem a normalização com uma amostra da população. Eles podem ser interpretados como resultados obtidos em uma prova de personalidade.

Estes valores são fornecidos para desenvolvedores que querem implementar uma normalização personalizada. Por exemplo, criando uma normalização diferente levando em conta a região e cultura. Por isto estes valores não são retornados por default, e para retornanar é necessário passar *true* como parâmetro *raw\_scores* na chamada do método.

1. **Percentagens de características de comportamento: (*percentage*):**

Caso haja indicadores de tempo no JSON de entrada, o serviço retorna uma percentagem para cada característica de comportamento. Isto serve para identificar uma distribuição temporal dos dados analizados, ou seja, o valor indica quanto porcento dos textos de entrada está distribuído em cada hora do dia ou dia da semana.

Por exemplo, a pontuação da característica behavior\_sunday foi 0.4561049445005. Isto significa que 46% dos textos analizados (*ContentItems*) coram criados no domingo.

1. **Pontuação por preferência de consumo (*scores*):**

Conforme interesse, o Personality Insights pode retornar uma análise sobre os interesses de consumo do autor do texto analizado. Para isto basta especificar como *true* o parâmetro *consumption\_preferences* na hora de fazer o requerimento de um objeto Profile.

Para representar os interesses de consumo, cada objeto do tipo ConsumptionPreferences possui um atributo *score*, que é um indicador numérico double. Este atributo é um indicador de preferência e representa a probabilidade de o autor preferir tal item.

O atributo *score* pode possuir um dos três valores a seguir:

* **0.0:** Interesse muito baixo em certo item.
* **0.5:** O autor é neutro quanto ao item.
* **1.0:** Alto nível de interesse no item.

### Parte 4.4: Descrição dos modelos de personalidade

1. **Big Five Agreeableness:**
2. **Big Five Conscientiousness:**
3. **Big Five Extraversion:**
4. **Big Five Emotional Range:**
5. **Big Five Openess:**
6. **Needs:**
   1. **Excitement:**
   2. **Harmony:**
   3. **Curiosity:**
   4. **Ideal:**
   5. **Closeness:**
   6. **Self-expression:**
   7. **Liberty:**
   8. **Love:**
   9. **Practicality:**
   10. **Stability:**
   11. **Challenge:**
   12. **Structure:**
7. **Values:**
   1. **Self-transcedence / Helping others:**
   2. **Conservation / Tradition:**
   3. **Hedoism / Taking pleasure in life**
   4. **Self-enhancement / Achieving success:**
   5. **Open to change / Excitement:**

### Parte 4.5: Descrição dos modelos de preferência de consumo

Baseado nos modelos de caracteristicas de personalidade Big Five, Needs e Values, o Watson é capaz de produzir um relatório sobre as preferencias de consumo do autor dos dados de entrada. Para isto, é necessário que o parâmetro *consumption\_preferences* seja enviado como *true* na hora da chamada ao método *getProfile*.

1. **Compras:**
2. **Filmes:**
3. **Música:**
4. **Leitura e aprendizado:**
5. **Saúde e atividade física:**
6. **Empreendedorismo:**
7. **Meio ambiente:**
8. **Voluntariado:**

## Parte 5: Criação de exemplo de atrações turísticas

## Parte 6: Comparação entre o resultado do Personality Insights e as características das atrações turísticas (Match)

## Parte 7: Conhecendo o Watson Conversation

1. Definição do Escopo
2. Modelagem
   1. Arquitetura
   2. Diagrama Relacional de Dados
   3. Diagrama de Classes
   4. Diagrama BPMN
3. Casos de Uso
4. Mockups
5. Referências Bibliográficas

[1] – YouTube: IBM Watson: How it Works. Acesso em 20/09/2017. Disponível em:

<https://www.youtube.com/watch?v=_Xcmh1LQB9I&t=3s>

[2] – IBM: Conheça o Watson e seu uso na saúde. Acesso em 20/09/2017. Disponível em:

<https://www.ibm.com/blogs/robertoa/2017/03/conheca-o-watson-e-seu-uso-na-saude/>

[3] – IBM: Beyond bots and robots: Exploring the unrealized potential of cognitive computing in the travel industry. Acesso em 20/09/2017. Disponível em:

<https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=GBE03776USEN&>

[4] – Forbes: Bringwater, Adrian. Come Fly With AI, IBM Cloud Builds ’Chatbot’ Virutal Travel Agent. Acesso em 20/09/2017. Disponível em:

<https://www.forbes.com/sites/adrianbridgwater/2016/11/22/come-fly-with-ai-ibm-cloud-builds-chatbot-virtual-travel-agent/#1088409b4813>

[5] – WayBlazer: Acesso em 20/09/2017. Disponível em:

<https://www.wayblazer.ai/>

[6] – Baseline: Greengard, Samuel. WayBlazer’s Journey Leads to Cognitive Computing. Acesso em 24/09/2017. Disponível em:

<http://www.baselinemag.com/cloud-computing/wayblazers-journey-leads-to-cognitive-computing.html>

[7] – Watson Documentação: Acesso em 20/09/2017. Disponível em:

<https://console.bluemix.net/docs/services/personality-insights/models.html#models>

[8] – Watson Documentação: Acesso em 20/09/2017. Disponível em:

<https://console.bluemix.net/docs/services/conversation/index.html#about>

[9] – Watson Documentação: Acesso em 10/10/2017. Disponível em:

<https://console.bluemix.net/docs/services/conversation/lang-support.html#supported-languages>

[10] – Watson Documentação: Acesso em 10/10/2017. Disponível em:

<https://console.bluemix.net/docs/services/personality-insights/user-overview.html#overviewLanguage>

[11] – Watson GitHub: Acesso em 15/11/2017. Disponível em:

<https://github.com/watson-developer-cloud/swift-sdk/blob/master/docs/quickstart.md>