Uma imagem com desenho

Descrição gerada automaticamente

**Instituto Politécnico de Beja**

**Escola Superior de Tecnologia e Gestão**

**Licenciatura de Engenharia Informática**

**Tópicos de Engenharia Informática**

**ChatBot com React e NPL**

**Alexandre Pereira 17678**

**Mário Miguel 17693**

**Rodrigo Fernandes 17845**

**Beja**

**2020/2021**

**Instituto Politécnico de Beja**

**Escola Superior de Tecnologia e Gestão**

**Licenciatura de Engenharia Informática**

**Tópicos de Engenharia Informática**

**ChatBot com React e NPL**

**Elaborado por:**

Alexandre Pereira 17678

Mário Miguel 17693

Rodrigo Fernandes 17845

**Orientado por:**

**Prof. Gonçalo Fontes**

**Beja**

**2020/2021**

**Conteúdo**

[**Capítulo I – Introdução** 4](#_Toc72095005)

[**Capítulo II – Front-End** 5](#_Toc72095006)

[**Capítulo III – Back-End** 6](#_Toc72095007)

[**III.1** **Wit.ai** 6](#_Toc72095008)

[**III.1.1**  **Intents e Entities do Wit.ai** 6](#_Toc72095009)

[**III.1.2**  **Pedido ao Wit.ai** 7](#_Toc72095010)

[**III.2** **Script responseHandler.js** 8](#_Toc72095011)

[**III.2.1**  **Saudações** 8](#_Toc72095012)

[**III.2.2**  **Reservar Quartos** 9](#_Toc72095013)

[**III.2.3**  **Alterar Reserva** 9](#_Toc72095014)

[**III.2.4**  **Cancelar Reserva** 9](#_Toc72095015)

[**III.2.5**  **Espaço do Ginásio** 10](#_Toc72095016)

[**III.2.6** **Espaço da Piscina** 10](#_Toc72095017)

[**III.3** **Ficheiro Chatdata.json** 11](#_Toc72095018)

[**Capítulo IV – Correr o Projeto** 13](#_Toc72095019)

[**I.1.2**  **Instalar os Pacotes** 13](#_Toc72095020)

[**I.1.3**  **Correr a App e a Api** 13](#_Toc72095021)

[**Capítulo v – Conclusão** 14](#_Toc72095022)

# **Capítulo I – Introdução**

O objetivo deste projeto é criar um *Chatbot* de interface dinâmica na tecnologia React, com ligação ao motor do Wit.ai, potenciando-lhe capacidade de interação e conversão.

O *Chatbot* tem a temática de “Hotel” onde se enquadram as interações com o *Chatbot*. Por exemplo, é possível fazer reservas, informar sobre preços de quartos e serviços, ou ainda, saber informações dos espaços comuns do hotel.

O *Chatbot* dá resposta a todos os pedidos relacionados com a temática, aproximando-se a “conversa” o mais possível, à que o utilizador teria com um ser humano.

# **Capítulo II – Front-End**

Para o Front-End utilizámos todo o trabalho feito anteriormente na primeira parte, apenas alterando a imagem de fundo, o texto e os ícones.

Decidimos utilizar componentes do Material UI, para estarmos a par com os designs modernos.

De forma a mantermos o aspecto de chat, utilizamos o componente Grid, pois permite manter uma proporção igual em todos os tamanhos de ecrã.

Criámos 3 componentes principais: Header, Body e Footer. Dentro do Header colocámos o nome do *bot*, no Body o *chat* e no Footer um pequeno texto de informação.

No chat, para cada a lista de mensagens foi criado um componente e para cada mensagem foi criado um componente também. O *input* do utilizador é também um componente à parte.

Para termos controlo sobre o estado de cada componente, o estado da lista, o estado dos *inputs*, utilizámos os React Hooks, mais concretamente o UseState e os Props.

O website ficou com o aspeto apresentado na Figura 1 - Interface do Utilizador.

Graphical user interface

Description automatically generated

Figura - Interface do Utilizador

# **Capítulo III – Back-End**

No back-end utilizamos o wit.ai, o Express, o JSON e, claro, o NodeJS.

## **III.1 Wit.ai**

Após uma breve análise realizada por parte do grupo, concordámos que seria melhor começar pelo *Wit.ai*.

### **III.1.1 Intents e Entities do Wit.ai**

No wit.ai escolhemos quais eram os *Intents* para os tópicos de conversa que o utilizador pode ter com o *Chatbot* e quais as *Entities* que estes utilizam. Na Figura 2 - Intents Wit.AI estão os *Intents* e os respetivos *Entities* que colocámos inicialmente.

Table

Description automatically generated

Figura - Intents Wit.AI

### **III.1.2 Pedido ao Wit.ai**

Para enviar o que o utilizador escreve no chat para o wit.ai é utilizado o *endpoint* “/api/sendQuestion” no ficheiro *server.js*, apresentado na Figura 3 - Http request para a API Wit.AI, que faz o *get* com o parâmetro *question* e o valor do texto escrito.

Depois, é recebido um *JSON* com o respetivo *intent* relativo ao texto e a lista das *entities* que nele estão e o script responseHandler.js trata da resposta para ser dada ao utilizador.

Text

Description automatically generated

Figura - Http request para a API Wit.AI

## **III.2 Script responseHandler.js**

Neste Script é onde as respostas ao utilizador são escolhidas e onde está toda a logica correspondente á sequencia do texto apresentado ao utilizador como resposta ao que este diz.

Text

Description automatically generatedNeste script existe um *switch-case* que chama uma função diferente para cada tópico de conversa identificado pelo *Intent* recebido pelo wit.ai como está apresentado na Figura 4 - Switch Case para os Intents.

Figura - Switch Case para os Intents

### **III.2.1 Saudações**

No caso das saudações é utilizado a função *handleGreetings*, apresentada na Figura 5 - Saudaçõesque já sabendo qual é o tópico afirma que a resposta a devolver será uma das respostas, escolhida de forma aleatória, presentes no ficheiro JSON onde estão as respostas possíveis para o utilizar para aquele *Intent*.

Text

Description automatically generated

Figura - Saudações

### **III.2.2 Reservar Quartos**

Para que seja efetuada uma reserva, o utilizador tem de dar 4 informações: data da reserva, tipo de cama, quantidade de camas e quantidade de quartos.

Antes de ser feita uma reserva, normalmente perguntam-se os preços. Para isso, criámos uma condição que verifica se a lista de *Entities* contém certas propriedades como “access\_price:access\_price” e “room\_quantity:room\_quantity”. Depois, sabendo que se trata de um pedido de preços, selecionamos no ficheiro de respostas a resposta correspondente.

Para iniciar uma reserva, o *backend* verifica primeiro se existe alguma reserva no sistema, através de cookies. Caso não exista, é utilizado um *for in loop* para percorrer as propriedades do objecto Reserva e são preenchidas essas propriedades com os inputs que o utilizador fornece. Caso exista algum dado em falta, a resposta do *bot* é obtida dinamicamente a partir do ficheiro de respostas. Quando chega ao fim, como falta o campo “confirmar”, é perguntado ao utilizador se quer confirmar a reserva, mostrando também um resumo da mesma. Ao responder afirmativamente, a reserva é guardada numa cookie. Se a resposta for negativa, a reserva é limpa.

### **III.2.3 Alterar Reserva**

O *backend* ao detetar uma intenção de mudança na reserva, verifica se existe alguma reserva feita nas *cookies*. Quando não existe uma reserva guardada, o utilizador é convidado a efetua uma reserva. Se já existir uma reserva, o utilizador apenas tem de indicar o que pretende alterar.

Para as verificações da alteração, é criado uma reserva temporária com os novos dados fornecidos pelo utilizador. Se as propriedades da reserva temporária forem diferentes da reserva guardada, o sistema deteta que houve uma alteração na reserva e pergunta ao utilizador se quer confirmar a alteração, mostrando um novo resumo da reserva. Depois, dependendo da resposta do utilizador, a reserva guardada nos *cookies* é substituída pela nova reserva, ou não.

### **III.2.4 Cancelar Reserva**

Caso o utilizador queira cancelar a reserva, apenas tem de mostrar a sua intenção ao *bot*. Se não existir uma reserva para ser cancelada, o *bot* avisa o utilizador. Se existir, é pedida uma verificação ao utilizador.

Para ver se existe uma reserva, são verificadas as *cookies* do *request* *http*. Se a confirmação do utilizador for positiva, as cookies são apagadas.

### **III.2.5 Espaço do Ginásio**

Text

Description automatically generated No caso do utilizador fizer perguntas relativas ao ginásio é utilizada a função *handleGymAccess* apresentada naFigura 6 - Ginásio. Nesta função são identificados quais são as *Entities* presentes no texto que o utilizador escreveu e dá uma das respostas que estão no ficheiro *JSON* de acordo com a pergunta. Sendo estas sobre o preço de entrada, sobre as coisas que há no ginásio ou sobre o ginásio em geral.

Figura - Ginásio

### **III.2.6 Espaço da Piscina**

No caso do utilizador fizer perguntas relativas á piscina é utilizada a função *handlePoolAcess* apresentada na Figura 7 - Piscina. Nesta função são identificados quais são as *Entities* presentes no texto que o utilizador escreveu e dá uma das respostas que estão no ficheiro *JSON* de acordo com a pergunta. Sendo estas sobre o preço de entrada, sobre as coisas que há na piscina ou sobre a piscina em geral.

Text

Description automatically generated

Figura - Piscina

## **III.3 Ficheiro Chatdata.json**

Neste ficheiro JSON estão guardadas todas as respostas que são dadas ao utilizador de acordo com a conversa que ele tem com o *Chatbot*. Como está apresentada na Figura 8 - Ficheiro JSON que tem um pequeno trecho do código este documento tem uma resposta pré-definida para quando o *Chatbot* não entende a pergunta e depois está organizado por *Intents em que dentro de cada um tem um objeto com várias respostas para cada Entity.*

Text

Description automatically generated

Figura - Ficheiro JSON

# **Capítulo IV – Correr o Projeto**

Para instalar os pacotes no projeto foi utilizado o npm. Para correr a API foi utilizado o node.

### **I.1.2 Instalar os Pacotes**

Para instalar os pacotes basta fazer “npm install” nas diretorias “…\npl-chatbot\api>” e \npl-chatbot\app> como está exemplificado na Figura 9 - Instalação de Packages.

Text

Description automatically generated

Figura - Instalação de Packages

### **I.1.3 Correr a App e a Api**

Para correr a app basta estar dentro da pasta \app e fazer o comando “npm start”.

Para correr a api basta estar dentro da pasta \api e fazer o comando “nodemon server.js”.

Text

Description automatically generatedPodemos observar na Figura 10 - Como correr a app como fazer o acima referido.

Figura - Como correr a app

# **Capítulo V – Conclusão**

Consideramos que o projeto correu, no geral, bem e que fizemos o requisitado pelo docente.

Através deste projeto ficámos com uma melhor compreensão da diferenciação entre encargos do *backend* e encargos do *frontend*, dividindo assim as responsabilidades para cada lado. Também ficámos com uma compreensão mais profunda em relação à linguagem Javascript, a biblioteca ReactJS para *frontend* e o Express enquanto *Framework* do NodeJS.

Para trabalho futuro, apenas ficou a faltar o *bot* sugerir perguntas em que o utilizador podia escrevê-las ou selecioná-las e fazer *deploy* para uma plataforma de *hosting* como o Heroku, Pusher ou Vercel.