Housem8 - LDS

Test design Specification

Version 2.0

12/01/2020

**Histórico de Versões**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** | |
| 1.0 | *Jorge Moreira* | *07/12/2020* | *Jorge Moreira* | *07/12/2020* | Criação do documento e adaptação para o projeto | |
| 1.1 | *Jéssica Coelho*  *Miguel Costa*  *Jorge Moreira* | *08/12/2020* | *Jorge Moreira* | *08/12/2020* | Adição da especificação dos testes de algumas componentes | |
| 1.2 | *Jéssica Coelho*  *Jorge Moreira* | *08/12/2020* | *Jorge Moreira* | *08/12/2020* | Âmbito, finalização dos componentes userDao, EmployerDao, MateDao, JobDao e workDao  Revisão | |
| 1.3 | *Miguel Costa* | *24/12/2020* | *Miguel Costa* | *24/12/2020* | Correções de testes relacionados com a pesquisa de trabalho | |
| 1.4 | *Miguel Costa* | *27/12/2020* | *Miguel Costa* | *27/12/2020* | Correções de testes relacionados com o trabalho. | |
| 1.5 | *Jéssica Coelho* | *05/01/2021* | *Jéssica Coelho* | *05/01/2021* | Adição da especificação de um teste da componente de atualizar palavra passe. | |
| 1.6 | *Jéssica Coelho* | *11/01/2021* | *Jéssica Coelho* | *11/01/2021* | Adição da especificação de testes das componentes de atualizar e remover Work. | |
| 2.0 | *Jorge Moreira* | *12/01/2020* | *Jorge Moreira* | *12/01/2020* | Reformulação dos testes do chat.  Review aos testes de remover e atualizar Work |

Tabela de Conteúdos

[1. Introdução 4](#_Toc60778895)

[1.1. Identificador do documento 4](#_Toc60778896)

[1.2. Âmbito 4](#_Toc60778897)

[1.3. Referências 5](#_Toc60778898)

[1.4 Glossário 5](#_Toc60778899)

[2. Features/Itens a testar 6](#_Toc60778900)

[3. Detalhes da abordagem aos testes 9](#_Toc60778901)

[4. Identificação dos Testes 10](#_Toc60778902)

[**4.1 Casos de teste das features do componente *UserDAO*** 10](#_Toc60778903)

[**4.2 Casos de teste das features do componente *EmployerDAO*** 11](#_Toc60778904)

[**4.3** **Casos de teste das features do componente *MateDAO*** 12](#_Toc60778905)

[**4.4** **Casos de teste das features do componente *JobDAO*** 15](#_Toc60778906)

[**4.5** **Casos de teste das features do componente *WorkDAO*** 18](#_Toc60778907)

[**4.6** **Casos de teste das features do componente *LoginDAO*** 20](#_Toc60778908)

[**4.7** **Casos de teste das features do componente *ReportDAO*** 21](#_Toc60778909)

[**4.8** **Casos de teste das features do componente *ReviewEmployerDAO*** 22](#_Toc60778910)

[**4.9** **Casos de teste das features do componente *ReviewMateDAO*** 23](#_Toc60778911)

[**4.10 Casos de teste das features do componente *DistanceHelper*** 24](#_Toc60778912)

[**4.11** **Casos de teste das features do componente *PaymentDAO*** 25](#_Toc60778913)

[**4.12 Casos de teste das features do componente *ChatDAO*** 27](#_Toc60778914)

[5. Critérios de passagem ou falha das features 29](#_Toc60778915)

[**5.1 ECs para as features do componente *UserDAO*** 29](#_Toc60778916)

[**5.2 ECs para as features do componente *EmployerDAO*** 31](#_Toc60778917)

[**5.3 ECs para as features do componente *MateDAO*** 33](#_Toc60778918)

[**5.4 ECs para as features do componente *JobDAO*** 37](#_Toc60778919)

[**5.5 ECs para as features do componente *WorkDAO*** 42](#_Toc60778920)

[**5.6 ECs para as features do componente *LoginDAO*** 44](#_Toc60778921)

[**5.7 ECs para as features do componente *ReportDAO*** 45](#_Toc60778922)

[**5.8 ECs para as features do componente *ReviewEmployerDAO*** 46](#_Toc60778923)

[**5.9 ECs para as features do componente *ReviewMateDAO*** 47](#_Toc60778924)

[**5.10 ECs para as features do componente *DistanceHelper*** 48](#_Toc60778925)

[**5.11 ECs para as features do componente *PaymentDAO*** 49](#_Toc60778926)

[**5.11 ECs para as features do componente *ChatDAO*** 51](#_Toc60778927)

# Introdução

## 1.1. Identificador do documento

***TDS01122021V2.0***

Descrição do identificador do projeto:

*TDS* - TestDesignSpecification

01 - Número do mês (dois dígitos seguidos dos dois dígitos iniciais)

12 - Dia do mês (dois dígitos iniciais)

2021 - Ano de criação (quatro dígitos finais)

V2.0 - Número de versão do projeto (dois caracteres sendo “V” o identificador de versão através de sigla seguido do número de versão “2” e número de alteração “.0”)

## 1.2. Âmbito

O produto a ser testado é uma aplicação (web e mobile) com o objetivo de fornecer serviços, ou procura de trabalhadores para realizar trabalhos (reparações, etc..) em casas de clientes. Este sistema será composto por uma aplicação para dispositivos móveis desenvolvida em Flutter, irá conter um site desenvolvido em React/Angular (ainda a decidir) e, para a disponibilização dos serviços de negócio, irá ter uma aplicação no backend desenvolvida em ASP.net core.

Nesta versão do documento apenas vão ser testadas as classes de *data access* da API. Os controladores podem ser testados utilizando a documentação swagger.

Nota: Esta versão do documento não representa o seu estado final. Este documento vai ser submetido a futuras atualizações.

Funcionalidades a testar:

* *UserDAO*
  + UpdatePassword – Editar palavra passe.
  + FindUserByEmail – Procurar um utilizador por email.
  + FindById – Procurar um utilizador por Id.
* *EmployerDAO*
  + Create - Criar um *Employer* com os seus dados pessoais.
  + Update - Atualizar os dados pessoais de um *Employer.*
  + FavouriteList - Obter a lista de *Mates* favoritos de um *Employer.*
  + AddFavourite - Adicionar um *Mate* à lista de favoritos de um *Employer.*
  + RemoveFavourite - Remover um *Mate* da lista de favoritos de um *Employer.*
  + FindEmployerById - Obter um *Employer* através do seu Id.
* *MateDAO*
  + Create - criar um *Mate* com os seus dados pessoais.
  + CategoriesList - obter a lista de categorias de trabalho de um *Mate.*
  + AddCategory - adicionar categorias à lista de categorias de trabalho de um *Mate.*
  + RemoveCategory - remover uma categoria da lista de categorias de trabalho de um *Mate.*
  + Update - Atualizar os dados pessoais de um *M*ate.
  + FindMateById - Obter um *Mate* através do seu Id.
  + PendingJobsList - Obter a lista de trabalhos pendentes de um *Mate.*
  + GetMates - Utilização de filtros para encontrar *Mates.*
  + IgnoreJobPost - *Ignorar um JobPost.*
* *JobDAO*
  + AddPayment – Adicionar métodos de pagamento a uma publicação de trabalho.
  + RemovePayment – Remover um método de pagamento a uma publicação de trabalho.
  + GetJobs – Obter lista de trabalhos para o *Mate* com filtros selecionados por ele.
  + MakeOfferOnJob – Realizar oferta de preço num trabalho.
  + Create – Criar uma publicação de trabalho.
  + Delete – Apagar uma publicação de trabalho.
  + FindById – Encontrar uma publicação de trabalho através do seu Id.
  + GetEmployerPosts – Obter todas as publicações de trabalho pertencentes a um *Employer*.
  + UpdatePostDetails – Atualizar os detatlhes de uma publicação de trabalho.
* *WorkDAO*
  + MarkJobAsDone – Marcar um trabalho como realizado tanto para o *Mate* como para o *Employer*.
  + Create – Criar um trabalho associado a um *Employer*, *JobPost* e *Mate*.
  + FindById – Encontrar um trabalho através do seu Id.
* *LoginDAO*
  + Authenticate – Autenticar um utilizador.
* *ReportDAO*
  + ReportUser – Reportar um utilizador.
* *ReviewEmployerDAO*
  + Dar uma *review/rating* ao *Employer.*
* *ReviewMateDAO*
  + Dar uma *review* ao *Mate.*
* *PaymentDAO*
  + GetInvoiceById – Obter o recido de pagamento através do seu Id.
  + MakePayment – Realizar o pagamento de um trabalho.
  + ConfirmPayment – Permitir ao *Mate* confirmar que o pagamento foi realizado.
* *DistanceHelper*
  + GetAddressFromCoordinates – Obter um endereço através de coordenadas geográficas.
* *ChatDAO*
  + AddMessage – Adicionar uma mensagem para permitir a comunicação entre dois utilizadores.
  + CreateChat – Criar *chats* com referências de dois utilizadores para estes poderem enviar mensagens entre si.
  + GetChatIdWithTargetUser – Obtém o Id do chat que um utilizador tem com outro.
  + GetMessageList – Visualizar uma lista de mensagens associadas a um *chat*.
  + GetChats – Visualizar um *array* de chats associados a um utilizador.
  + AddChatHubConnection – Adiciona a BD uma conexão ao ChatHub.
  + UpdateChatHubConnection – Atualiza na BD duma conexão do ChatHub.
  + DeleteChatHubConnection – Apaga uma conexão de ChatHub da BD.
  + GetChatConnectionFromUserId – Obtém a conexão ao chat e userId a partir do userId.
  + GetChatConnectionFromConnectionId – Obtém a conexão ao chat e userId a partir da connectionId.
  + GetLastMessageFromChat – Obtém a última mensagem do chat.
  + IsUserFromChat – Verifica se o utilizador pertence a um chat.

## 1.3. Referências

Lista de referências/outros documentos relevantes para este documento:

* Test Case Outline;
* https://gitlab.estg.ipp.pt/8160297/lds-housem8

## 1.4 Glossário

JM – Jorge Moreira

MC – Marcelo Carvalho

SC – Samuel Cunha

JC – Jéssica Coelho

MMC – Miguel Martins Costa

TC – Test Case (Caso de Test)

ECP - Equivalence Class Partitioning

EC – Equivalence Class (Classe de Equivalência)

ECs – Classes de equivalência

BVA - Boundary Value Analysis

## 2. Features/Itens a testar

|  |  |  |  |
| --- | --- | --- | --- |
| **Item a testar** | **Descrição das features** | **Requisitos** | **Responsabilidade** |
| *UserDAO* | ***UpdatePassword*** – permite editar a palavra passe do utilizador | Editar palavra passe | JC |
| **FindUserByEmail** – permite procurar um utilizador pelo seu email | Encontrar utilizador | SC |
| **FindById** – permite procurar um utilizador por Id | Encontrar utilizador | JM |
| *EmployerDAO* | ***Create*** – permite criar um *Employer* com os seus dados pessoais | Criar um *Employer* | JM |
| ***Update*** – permite atualizar os dados pessoais de um *Employer* | Editar *Employer* | JC |
| ***FavoriteList*** – permite obter a lista de *Mates* favoritos de um *Employer* | Obter lista de *Mates* favoritos de um *Employer* | JC |
| ***AddFavorite*** – permite adicionar um *Mate* à lista de favoritos de um *Employer* | Adicionar *Mate* à lista de favoritos de um *Employer* | JC |
| ***RemoveFavorite*** – permite remover um *Mate* da lista de favoritos de um *Employer* | Remover *Mate* da lista de favoritos de um *Employer* | JC |
| ***FindEmployerById*** – permite obter um *Employer* através do seu Id | Obter *Employer* através do Id | JC |
| *MateDAO* | ***Create*** – permite criar um *Mate* com os seus dados pessoais | Criar um *Mate* | JM |
| ***CategoriesList*** – permite obter a lista de categorias de trabalho de um *Mate* | Obter a lista de categorias de um *Mate* | JC |
| ***AddCategory*** – permite adicionar categorias à lista de categorias de trabalho de um *Mate* | Adicionar categorias à lista de categorias de um *Mate* | JC |
| ***RemoveCategory*** – permite remover uma categoria à lista de categorias de trabalho de um *Mate* | Remover categoria à lista de categorias de um *Mate* | JC |
| ***Update*** – permite atualizar os dados pessoais de um *M*ate | Editar *Mate* | JC |
| ***FindMateById*** – permite obter um *Mate* através do seu Id | Obter *Mate* através do Id | JC |
| ***PendingJobsList*** – permite obter a lista de trabalhos pendentes de um *Mate* | Obter a lista de trabalhos pendentes de um *Mate* | JC |
| ***GetMates*** – permite a utilização de filtros para encontrar *Mates* | Obter lista de *Mates* usando filtros. Pesquisar por mates sem autenticação | JM |
| ***IgnoreJobPost*** – permite ao *Mate ignorar um JobPost* | Ignorar Trabalho | MMC |
| *JobDAO* | ***AddPayment*** – permite adicionar métodos de pagamento a uma publicação de trabalho feita pelo *Employer* | Adicionar métodos de pagamento a uma publicação de trabalho | JC |
| ***RemovePayment*** – permite remover um método de pagamento de uma publicação de trabalho feita pelo *Employer* | Remover método de pagamento de uma publicação de trabalho | JC |
| ***GetJobs*** – permite obter uma lista de trabalhos para o *Mate* com filtros selecionados pelo mesmo | Procurar Trabalho, Modificar Filtros | MMC |
| ***MakeOfferOnJob –*** permite realizar uma oferta de preço num trabalho | Realizar oferta de preço ao Trabalho | MMC |
| ***Create –*** permite criar um *JobPost* | Criar um *JobPost* | JM |
| ***Delete*** –permite apagar um *JobPost* | Apagar um *JobPost* | JM |
| ***FindByID –*** permite encontrar um *JobPost* por Id | Encontrar um *JobPost* por Id | JM |
| ***GetEmployerPosts –*** permite obter todos os *JobPosts* pertencentes a um *Employer* | Obter todos os *JobPosts* criados por um *Employer* | JM |
| ***UpdatePostDetails*** – permite atualizar os detalhes de um *JobPost* | Atualizar dados de um *JobPost* | JM |
| *WorkDAO* | ***MarkJobAsDone*** – permite marcar um trabalho como realizado tanto para o *Mate* como para o *Employer* | Marcar trabalho como concluído | MMC, JC |
| ***Create*** – permite criar um trabalho associado a um *Employer*, *JobPost* e *Mate* | Criar um trabalho com *Mate* associado | JM |
| ***FindById*** – permite encontrar um trabalho por Id | Encontrar um trabalho por Id | JM |
| *LoginDAO* | ***Authenticate*** – método que permite autenticar um utilizador | Autenticar utilizador | SC |
| *ReportDAO* | ***ReportUser*** – permite reportar um utilizador | Reportar utilizador | SC |
| *ReviewEmployerDAO* | ***ReviewEmployer*** – permite ao *Mate* dar um *review*/*rating* ao *Employer* | Avaliar *Employer* | SC |
| *ReviewMateDAO* | ***ReviewMate*** – permite ao *Employer* dar um *review* ao *Mate* | Avaliar *Mate* | SC |
| *PaymentDAO* | ***GetInvoiceByID*** – permite obter o recibo de pagamento | Efetuar o Pagamento | MMC |
| ***MakePayment*** – permite realizar o pagamento de um trabalho | Efetuar o Pagamento | MMC |
| ***ConfirmPayment*** – permite que o *Mate confirme que o pagamento foi realizado* | Efetuar o Pagamento | MMC |
| *DistanceHelper* | ***GetAddressFromCoordinates*** – permite obter um endereço através de coordenadas geográficas | Utilização da Localização | MMC |

|  |
| --- |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| *ChatDao* | ***AddMessage***- Permite adicionar uma mensagem para permitir comunicação entre dois utilizadores | Efetuar Negociação através de Chat | MC |
| ***CreateChat***- permite criar chats com referências de dois utilizadores para poderem enviar mensagens entre si. | Efetuar Negociação através de Chat | MC |
| ***GetMessageList***-permite ver uma lista de mensagens associadas a um chat. | Efetuar Negociação através de Chat | MC |
| ***GetChats-*** permite ver um array de chats associados com um utilizador. | Efetuar Negociação através de Chat | MC |

## **3. Detalhes da abordagem aos testes**

A abordagem a ser utilizada para os testes dos itens referidos na tabela da secção anterior é uma abordagem *BlackBox*. Esta estratégia avalia o comportamento externo de software sem considerar o comportamento interno do mesmo, ou seja, são apenas consideradas as entradas e as saídas como uma base para o desenho dos casos de teste. Dentro desta abordagem, usaram-se dois métodos que foram usados para o planeamento dos casos de teste, cujos métodos são: *Equivalence Class Partitioning* (Partição em classes de equivalência / ECP) e *Boundary Value Analysis* (Análise de valor limite / BVA), ambos lecionados em unidades curriculares anteriores.

*Equivalence Class Partitioning*: É uma técnica destinada a reduzir o número de testes necessários e que divide o domínio de entrada (ou saída) em classes de dados em que os casos de teste podem ser derivados.

*Boundary Value Analysis*: É uma técnica focada nos limites do domínio de entrada (ou saída) e imediatamente acima e abaixo (além de ou em vez de valores intermédios).

A inspeção visual através do IDE (*Visual Studio*) foi o método usado para a análise dos resultados de teste. Para o apoio da seleção de casos de teste foram criadas classes de equivalência válidas e inválidas que definiram os critérios de passagem e de falha dos *test cases*, bem como os tipos de *input*. O suporte à seleção de *test cases* também foi influenciado pelas partições do intervalo de valores com a técnica de BVA, com o objetivo de testar os limites desta coleção.

## **4. Identificação dos Testes**

### **4.1 Casos de teste das features do componente *UserDAO***

#### **4.1.1 Método UpdatePassword**

**TC1 - CanUserUpdatePasswordTest:**

O TC1 tem como objetivo testar se o método *UpdatePassword* edita a palavra passe do utilizador com sucesso, verificando se o método retorna *True* quando se efetua a alteração da palavra passe, introduzido a antiga palavra passe corretamente e a nova palavra passe.

**TC2 – ReturnExceptionWrongOld Password:**

O TC2 tem como objetivo testar se o método *UpdatePassword* não edita a palavra passe do utilizador, verificando se o método retorna *False* quando se efetua a alteração da palavra passe, introduzido a antiga palavra passe incorretamente e a nova palavra passe.

#### **4.1.2 Método FindUserByEmail**

**TC1 – FindByEmailTest:**

O TC1 tem como objetivo testar se o método *FindUserByEmail* retorna um utilizador válido com sucesso, retornando *True* se o mesmo existir.

**TC2 - FindByEmailNullTest:**

O TC2 tem como objetivo testar se o método *FindUserByEmail* não retorna um utilizador válido com sucesso, retornando *True* se o mesmo não existir.

#### **4.1.5 Método FindById**

**TC1 – CanFindUserByIdTest:**

O TC1 tem como objetivo testar se o método *FindById* retorna o Utilizador com Id correspondente ao Id que entra como parâmetro no método.

**TC2 – *FindUserByIdReturnNullTest*:**

O TC2 tem como objetivo testar se o *user* retornado é *null* caso o Id não exista na base de dados.

### **4.2 Casos de teste das features do componente *EmployerDAO***

#### **4.2.1 Método Create**

**TC1 – CanRegisterEmployer:**

O TC1 tem como objetivo testar se o método *Create* cria um *Employer* na base de dados e retorna o *Employer* criado.

**TC2 – CanRegisterEmployerWithoutDescription:**

O TC2 tem como objetivo testar se o método *Create* cria um *Employer* na base de dados sem descrição e retorna o E*mployer* criado sem a descrição.

#### **4.2.2 Método Update**

**TC1 – CanEmployerUpdateProfileTest:**

O TC1 tem como objetivo testar se o método *Update* consegue editar as informações do *Employer* com sucesso, adicionando um utilizador, acrescentando-lhe de seguida dados atualizados e fazendo a comparação dos novos dados pessoais enviados para o método com os dados que foram recebidos pelo método. Neste caso, comparou-se o primeiro nome, apelido, nome de utilizador, descrição e morada.

#### **4.2.3 Método FavoriteList**

**TC1 – CanEmployerListFavoriteMatesTest:**

O TC1 tem como objetivo testar se o método *FavoriteList* consegue devolver a lista de *Mates* favoritos de um *Employer* com sucesso. Após serem criados um *Employer* e um *Mate* e após a adição de um *Mate* à lista de favoritos, é comparado se o email do *Mate* retornado pelo método de adicionar corresponde ao email do *Mate* que é retornado no método *FavoriteList*.

#### **4.2.4 Método AddFavorite**

**TC1 – CanEmployerAddFavoriteMateTest:**

O TC1 tem como objetivo testar se o método *AddFavorite* consegue adicionar com sucesso um *Mate* à lista de *Mates* favoritos de um *Employer*. Após serem criados um *Employer* e um *Mate*, é testado se o Id do *Mate* enviado para o método *AddFavorite* é igual ao Id recebido por esse mesmo método.

#### **4.2.5 Método RemoveFavorite**

**TC1 – CanEmployerRemoveFavoriteMateTest:**

O TC1 tem como objetivo testar se o método *RemoveFavorite* consegue remover com sucesso um *Mate* da lista de favoritos de um determinado *Employer*. Após serem criados um *Employer* e um *Mate* e após a adição de um *Mate* à lista de favoritos, é comparado o id do *Mate* enviado para o método *RemoveFavorite* com o que é recebido por esse mesmo método.

#### **4.2.6 Método FindEmployerById**

**TC1 – CanGetEmployerWithDescriptionByIdTest:**

O TC1 tem como objetivo testar se o método *FindEmployerById* consegue retornar um *Employer* através do seu Id. Após ser registado um *Employer*, é comparado se os dados retornados pelo método *FindEmployerById* são iguais aos dados retornados pelo método de registo de utilizadores. Neste caso de teste, ao ser criado o *Employer*, é adicionada uma descrição, que posteriormente é retornada pelo método a ser testado.

**TC2 - CanGetEmployerWithoutDescriptionByIdTest:**

O TC2 tem como objetivo testar se o método *FindEmployerById* consegue retornar um *Employer* através do seu Id. Após ser registado um *Employer*, é comparado se os dados retornados pelo método *FindEmployerById* são iguais aos dados retornados pelo método de registo de utilizadores. Neste caso de teste, ao ser criado o *Employer*, não é adicionada uma descrição, sendo apenas retornados os restantes dados pelo método a ser testado.

### **4.3** **Casos de teste das features do componente *MateDAO***

#### **4.3.1 Método Create**

**TC1 – CanRegisterMate:**

O TC1 tem como objetivo testar se o método *Create* cria um mate na base de dados e retorna o *Mate* criado.

**TC2 – CanRegisterMateWithoutDescription:**

O TC2 tem como objetivo testar se o método *Create* cria um mate na base de dados sem descrição e retorna o *Mate* criado sem a descrição.

#### **4.3.2 Método CategoriesList**

**TC1 – CanMateListWorkCategoriesTest:**

O TC1 tem como objetivo testar se o método *CategoriesList* consegue devolver a lista de categorias de trabalho de um determinado *Mate*. Após ser registado um *Mate* na plataforma, são comparadas as categorias as quais ele adicionou aquando do registo com as existentes na lista de categorias.

#### **4.3.3 Método AddCategory**

**TC1 – CanMateAddCategoriesTest:**

O TC1 tem como objetivo testar se o método *AddCategory* consegue adicionar com sucesso uma ou mais categorias à lista de categorias de trabalho de um determinado *Mate*. Após a criação de um *Mate* na plataforma, é adicionada uma ou mais categorias através deste método, retornando *True* caso exista na lista de categorias do *Mate* uma categoria igual àquela que foi adicionada.

#### **4.3.4 Método RemoveCategory**

**TC1 – CanMateRemoveCategoriesTest:**

O TC1 tem como objetivo testar se o método *RemoveCategory* consegue remover com sucesso uma categoria da lista de categorias de trabalho de um determinado *Mate*. Após a criação de um *Mate* na plataforma, é removida uma das categorias adicionadas ao *Mate* no seu registo através deste método, retornando *False* caso exista na lista de categorias do *Mate* uma categoria igual àquela que foi removida.

#### **4.3.5 Método Update**

**TC1 – CanMateUpdateProfileTest:**

O TC1 tem como objetivo testar se o método *Update* consegue editar as informações do *Mate* com sucesso, adicionando um utilizador, acrescentando-lhe de seguida dados atualizados e fazendo a comparação dos novos dados pessoais enviados para o método com os dados que foram recebidos pelo método. Neste caso, comparou-se o primeiro nome, apelido, nome de utilizador, descrição, morada e a distância máxima que o *Mate* pretende percorrer para a realização de trabalhos.

#### **4.3.6 Método FindMateById**

**TC1 – CanGetMateWithDescriptionByIdTest:**

O TC1 tem como objetivo testar se o método *FindMateById* consegue retornar um *Mate* através do seu Id. Após ser registado um *Mate*, é comparado se os dados retornados pelo método *FindMateById* são iguais aos dados retornados pelo método de registo de utilizadores. Neste caso de teste, ao ser criado o *Mate*, é adicionada uma descrição, que posteriormente é retornada pelo método a ser testado.

**TC2 - CanGetMateWithoutDescriptionByIdTest:**

O TC2 tem como objetivo testar se o método *FindMateById* consegue retornar um *Mate* através do seu Id. Após ser registado um *Mate*, é comparado se os dados retornados pelo método *FindMateById* são iguais aos dados retornados pelo método de registo de utilizadores. Neste caso de teste, ao ser criado o *Mate*, não é adicionada uma descrição, sendo apenas retornados os restantes dados pelo método a ser testado.

#### **4.3.7 Método PendingJobsList**

**TC1 – CanMateGetPendingJobsListTest:**

O TC1 tem como objetivo testar se o método *PendingJobsList* consegue retornar a lista de trabalhos pendentes de um determinado *Mate* com sucesso. Após ser adicionado um *Employer* e um *Mate*, ser criada uma publicação de trabalho e ser atribuído um *Mate* e uma data a esse trabalho através da classe *WorkDAO*, é comparado se o título do trabalho devolvido pelo método de criação de publicação de trabalho é igual a algum título existente na lista de trabalhos pendentes do *Mate*.

#### **4.3.8 Método GetMates**

**TC1 – CanSearchMateWithAllParams:**

O TC1 tem como objetivo testar se o método *GetMates* retorna uma lista de *Mates* utilizando todos os filtros de procura como parâmetros. Neste caso, foi adicionado um *Mate* e foi verificado se todos os campos do *Mate* que veio na lista eram iguais aos campos do *Mate* instanciado no teste antes de ser adicionado. O teste passa se todos os campos forem iguais.

**TC2 – CanSearchMateWithCategoriesAddressAndRank:**

O TC3 tem como objetivo testar se o método *GetMates* retorna uma lista de *Mates* utilizando somente filtros de categorias, morada e *rank*. Neste caso, foram adicionados dois *Mates* e foi verificado se todos os campos do primeiro *Mate* que veio na lista, eram iguais aos campos do primeiro *Mate* instanciado no teste antes de ser adicionado. O teste passa se todos os campos forem iguais.

**TC3 - CanSearchMateWithCategoriesAddressAndRankAndDistance:**

O TC6 tem como objetivo testar se o método *GetMates* retorna uma lista de *Mates* utilizando somente filtros de categorias, morada, *rank* e distância. Neste caso, foram adicionados dois *Mates* e foi-se verificado se todos os campos do primeiro *Mate* que veio na lista eram iguais aos campos do primeiro *Mate* a ser instanciado no teste antes de ser adicionado. O teste passa se todos os campos forem iguais.

**TC4 – CanSearchMateWithOneCategoryAndAddress:**

O TC4 tem como objetivo testar se o método *GetMates* retorna uma lista de *Mates* utilizando somente filtros de categorias com uma categoria e morada. Neste caso, foram adicionados dois *Mates* e foi verificado se todos os campos do primeiro *Mate* que veio na lista, eram iguais aos campos do primeiro *Mate* instanciado no teste antes de ser adicionado. O teste passa se todos os campos forem iguais.

**TC5 – CanSearchMateWithMultipleCategoriesAndAddress:**

O TC5 tem como objetivo testar se o método *GetMates* retorna uma lista de *Mates* utilizando somente filtros de categorias e morada. Neste caso, foram adicionados dois *Mates* e foi verificado se todos os campos de ambos os *Mates* que vieram na lista, eram iguais aos campos dos *Mates* instanciados no teste antes de serem adicionados. O teste passa se todos os campos forem iguais.

**TC6 – CanSearchWithoutAddress:**

O TC2 tem como objetivo testar se o método *GetMates* retorna uma lista de *Mates* utilizando todos os filtros de procura como parâmetros exceto o filtro da morada (*Address*). Neste caso, foram adicionados dois *Mates,* e foi verificado se todos os campos do primeiro *Mate* que veio na lista, eram iguais aos campos do primeiro *Mate* instanciado no teste antes de ser adicionado. O teste passa se todos os campos forem iguais.

#### **4.3.9 Método IgnoreJobPost**

**TC1 – CanIgnoreValidJob:**

O TC1 tem como objetivo testar se o método *IgnoreJobPost* retorna um booleano com verdadeiro quando é ignorado um *JobPos*t que existe e que o mesmo não aparece no feed. Para a realização do teste é necessário criar um *Employer*,associar um *JobPost* ao mesmo, criar um *Mate* e ignorar o trabalho através da sua conta.

Para o teste ser considerado como sucesso o método deve retornar um booleano de verdadeiro e o trabalho não pode aparece no feed.

**TC2 – ReturnFalselWhenIgnoringAJobThatDoesntExist :**

O TC2 tem como objetivo testar se o método *IgnoreJobPost* retorna um booleano com falso quando é ignorado um *JobPost* que não existe. Para a realização do teste é necessário criar um *Employer*,criar um *Mate* e ignorar o trabalho que não existe através da sua conta.

Para o teste ser considerado como sucesso o método deve retornar um booleano de falso.

### **4.4** **Casos de teste das features do componente *JobDAO***

#### **4.4.1 Método AddPayment**

**TC1 – CanAddPaymentTypeTest:**

O TC1 tem como objetivo testar se o método *AddPayment* consegue adicionar um método de pagamento a uma publicação de trabalho. Após ser registado na plataforma um *Employer* e adicionada uma publicação de trabalho, são adicionados um ou mais métodos de pagamento e a publicação de trabalho é procurada através do seu Id, sendo depois verificado se a publicação pesquisada contém o método de pagamento adicionado.

#### **4.4.2 Método RemovePayment**

**TC1 – CanRemovePaymentTypeTest:**

O TC1 tem como objetivo testar se o método *RemovePayment* consegue remover um método de pagamento a uma publicação de trabalho. Após ser registado na plataforma um *Employer* e adicionada uma publicação de trabalho, é removido um dos métodos de pagamento inseridos aquando da criação da publicação de trabalho e esta é procurada através do seu Id, sendo depois verificado se a publicação pesquisada não contém o método de pagamento removido.

#### **4.4.3 Método GetJobs**

**TC1 – CanSearchJobPostWithOneCategory:**

O TC1 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando é só enviado como filtro uma categoria. Para a realização do teste é necessário criar um *Employer*,associar dois *JobPost* ao mesmo com categorias diferentes, criar um *Mate* e pesquisar por uma categoria que esteja atribuída a um dos *Jobs*.

Para o teste ser considerado como sucesso o método deve retornar uma lista com um dos *JobPosts*, dependendo da categoria selecionada, em que os dados devem estar de acordo com o mesmo.

**TC2 – CanSearchJobPostWithOneCategoryWithAddressAndWithoutDistance:**

O TC2 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando é só enviado como filtro uma categoria e o endereço do *Mate*, sendo que é só enviado o endereço e não a distancia disposta a ser percorrida, o cálculo de distância será ignorado.

Para a realização do teste, é necessário criar um *Employer*,associar um *JobPost* ao mesmo, criar um *Mate* e pesquisar por uma categoria e indicar o seu endereço.

Para o teste ser considerado como sucesso o método deve retornar uma lista com um *JobPost*, mesmo que com distancia grande pois não foi indicada a distância disposta a percorrer pelo *Mate*.

**TC3 – CanSearchJobPostWithOneCategoryWithDefaultAddressAndWithDistance:**

O TC3 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando é só enviado como filtro uma categoria e a distância disposta a ser percorrida pelo *Mate*.

Sendo que é só enviada a distância disposta a ser percorrida, o cálculo de distância será feito utilizando o endereço *default* do *Mate* fornecido durante a sua inscrição na plataforma.

Para a realização do teste é necessário criar um *Employer*,associar vários *JobPosst* ao mesmo, criar um *Mate* e pesquisar por uma categoria e indicar a distância.

Para o teste ser considerado como sucesso o método deve retornar uma lista com *JobPosts* dentro da distância indicada pelo *Mate*.

**TC4 – CanSearchJobPostWithOneCategoryWithDistanceAndAddress:**

O TC4 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando é só enviado como filtro uma categoria, o endereço do *Mate* e a distância disposta a ser percorrida pelo mesmo.

Sendo que é enviada a distância disposta a ser percorrida e o endereço, o cálculo de distância será feito utilizando o endereçoenviado.

Para a realização do teste é necessário criar um *Employer*,associar vários *JobPosts* ao mesmo, criar um *Mate* e pesquisar por uma categoria, indicar a distância e o endereço.

Para o teste ser considerado como sucesso o método deve retornar uma lista com *jobPosts* dentro da distância indicada pelo *Mate*.

**TC5 – CanSearchJobPostWithoutQueries:**

O TC5 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando não enviado nenhum filtro. Para a realização do teste é necessário criar um *Employer*,associar vários *JobPost*, criar um *Mate* e pesquisar por *Jobs*.

Para o teste ser considerado como sucesso o método deve retornar uma lista com *jobPosts*.

**TC6 – CanSearchJobPostWithRatingAndTwoCategory:**

O TC6 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando é enviado como filtro, a média do rating desejado e duas categorias. Para a realização do teste é necessário criar um *Employer* com o rating indicado,associar vários *JobPosts,* alguns com as categorias indicadas, criar um *Mate* e pesquisar por *Jobs* com um certo rating do *Employer*.

Para o teste ser considerado como sucesso o método deve retornar uma lista com *JobPosts* de acordo com os filtros.

**TC7 – CanSearchJobPostWithRatingOneCategory:**

O TC7 tem como objetivo testar se o método *GetJobs* consegue retornar uma lista de trabalhos quando é enviado como filtro, a média do *rating* desejado e uma categoria. Para a realização do teste é necessário criar um *Employer* com o rating indicado,associar vários *JobPosts,* alguns com a categoria indicada, criar um *Mate* e pesquisar por *Jobs* com um certo *rating* do *Employer*.

Para o teste ser considerado como sucesso o método deve retornar uma lista com *JobPosts* de acordo com os filtros.

**TC8 – CanSearchJobPostWithRatingToHighForJob:**

O TC8 tem como objetivo testar se o método *GetJobs* consegue filtrar pela média do *rating* desejada. Para a realização do teste é necessário criar um *Employer* com o rating baixo,associar vários *JobPosts,* criar um *Mate* e pesquisar por *Jobs* com um certo *rating* do *Employer*.

Para o teste ser considerado como sucesso o método deve retornar uma lista vaziapois o *rating* indicado é superior ao do *Employer*.

**TC9 – CanSearchJobPostWithTwoCategory:**

O TC9 tem como objetivo testar se o método *GetJobs* consegue filtrar por duas categorias desejadas. Para a realização do teste é necessário criar um *Employer*,associar vários *JobPosts* com as duas categorias desejadas e outras diferentes, criar um *Mate* e pesquisar por *Jobs* com as duas categorias desejadas.

Para o teste ser considerado como sucesso o método deve retornar uma lista de *JobPosts* de acordo com as categorias pesquisadas.

**TC10 – ReturnEmptyWhenSearchJobWithWrongLocation:**

O TC10 tem como objetivo testar se o método *GetJobs* retorna uma lista vazia quando o endereço enviado não existe. Para a realização do teste é necessário criar um *Employer*,associar vários *JobPosts,* criar um *Mate* e pesquisar por *Jobs*, enviado o endereço e distancia como filtro.

Para o teste ser considerado como sucesso o método deve retornar uma lista vazia.

#### **4.4.4 Método MakeOfferOnJob**

**TC1 – CanMakeOfferOnJob:**

O TC1 tem como objetivo testar se o método *MakeOfferOnJob* consegue retornar uma *Offer* com o preço definido pelo *Mate*. Para a realização do teste é necessário criar um *Employer*,associar dois *JobPost* ao mesmo, criar um *Mate* e realizar uma oferta com preço definido ao *Job*.

Para o teste ser considerado como sucesso o método deve retornar uma *Offer* com o mesmo preço definido pelo *Mate*.

**TC2 – CanMakeOfferOnJobWithoutPrice:**

O TC2 tem como objetivo testar se o método *MakeOfferOnJob* consegue retornar uma *Offer* com o preço *default* do trabalho quando o *Mate* não define nenhum preço. Para a realização do teste é necessário criar um *Employer*,associar dois *JobPost* ao mesmo, criar um *Mate* e realizar uma oferta sem preencher o preço da oferta.

Para o teste ser considerado como sucesso o método deve retornar uma *Offer* com o mesmo preço que está por *default* no trabalho.

#### **4.4.5 Método Create**

**TC1 – CanCreateJobPostWithoutImage:**

O TC1 tem como objetivo testar se o método *Create* consegue adicionar à base de dados um *JobPost* sem o caminho para imagens. Para este teste é necessário instanciar um *Employer* e um *JobPost* sem o atributo *ImagePath e* utilizar o método *Create* do *JobDAO*.

O teste passa se o *JobPost* retornado não for nulo e tiver os campos iguais ao do *JobPost* instanciado no teste.

**TC2 – CanCreateJobPost:**

O TC2 tem como objetivo testar se o método *Create* consegue adicionar á base de dados um *JobPost*. Para este teste é necessário instanciar um *Employer*, um *JobPost* *e* utilizar o método *Create* do *JobDAO*.

O teste passa se o *JobPost* retornado não for nulo e tiver os campos iguais ao do *JobPost* instanciado no teste.

#### **4.4.6 Método Delete**

**TC1 – CanDeleteJobPost:**

O TC1 tem como objetivo testar se o método delete retorna *True* se o *JobPost* for apagado com sucesso da base de dados. Para este teste é necessário instanciar um *Employer*, um *JobPost* *e* utilizar o método *Create* do *JobDAO*, e de seguida usar o método *Delete*. O teste passa se o método *Delete* retornar *True* e se a lista vier vazia, pois só foi criado um teste.

#### **4.4.7 Método FindByID**

**TC1 – CanFindJobPostById:**

O TC1 tem como objetivo testar se o método *FindById* retorna o *Jobpost*, com id correspondente ao id que entre como parâmetro no método. Para este teste é necessário instanciar um *Employer*, um *JobPost* *e* utilizar o método *Create* do *JobDAO*, de seguida é necessário usar o método *FindById*.

O teste passa se o *JobPost* retornado não for nulo, e se o seu Id é igual ao Id do *JobPost* instanciado no teste.

#### **4.4.8 Método GetEmployerPosts**

**TC1 – CanReturnJobPostsFromEmployer:**

O TC1 tem como objetivo testar se o método *GetEmployerPosts* retorna uma lista de *JobPosts* que pertencem ao *Employer* cujo Id é igual ao Id que entre como parâmetro no método. Para este teste é necessário instanciar um *Employer*, dois *JobPosts* *e* utilizar o método *Create* do *JobDAO*.De seguida utiliza-se o método *GetEmployerPosts* e guarda-se o resultado numa lista.

Para este teste passar, os Ids dos *JobPosts* instanciados tem de ser iguais aos Ids dos *JobPosts* da lista. A lista tem de ter exatamente 2 elementos.

#### **4.4.9 Método UpdatePostDetails**

**TC1 – CanUpdatePostDetails:**

O TC1 tem como objetivo testar se o método *UpdatePostDetails* atualiza um *JobPost* na base de dados e se retorna o *JobPost* atualizado. Para este teste é necessário instanciar um *Employer*, um *JobPosts* *e* utilizar o método *Create* do *JobDAO*. De seguida, instancia-se outro *JobPost* com o mesmo id do *JobPost* que foi criado e usa-se o método *UpdatePostDetails*.

O teste passa se os campos do *JobPost* retornados forem iguais ao do novo *JobPost* instanciado.

### **4.5** **Casos de teste das features do componente *WorkDAO***

#### **4.5.1 Método MarkJobAsDone**

**TC1 – CanEmployerMarkJobAsDone:**

O TC1 tem como objetivo testar se o método *MarkJobAsDone* retorna um booleano verdadeiro quando o *Employer* utiliza o método.

Para a realização do teste é necessário criar um *Employer*,associar um *JobPost,* criar um *Mate* e associar o *Mate* ao *Job*.

O teste é considerado como sucesso se o método retornar um booleano com verdadeiro.

**TC2 – CanMateMarkJobAsDone:**

O TC2 tem como objetivo testar se o método *MarkJobAsDone* retorna um booleano verdadeiro quando o *Mate* utiliza o método.

Para a realização do teste é necessário criar um *Employer*,associar um *JobPost,* criar um *Mate* e associar o *Mate* ao *Job*.

O teste é considerado como sucesso se o método retornar um booleano com verdadeiro.

**TC3 – ReturnExceptionWhenJobIDisInvalidOnMarkJobAsDone:**

O TC3 tem como objetivo testar se o método *MarkJobAsDone* retorna uma exceção quando é enviado como argumento um *JobID* inexistente.

Para a realização do teste é necessário criar um *Employer*,associar um *JobPost,* criar um *Mate* e associar o *Mate* ao *Job*.

O teste é considerado como sucesso se o método retornar uma exceção.

**TC4 – ReturnExceptionWhenUserIDisInvalidOnMarkJobAsDone:**

O TC4 tem como objetivo testar se o método *MarkJobAsDone* retorna uma exceção quando é enviado como argumento um *UserID* não está associado ao *Job*.

Para a realização do teste é necessário criar um *Employer*,associar um *JobPost,* criar um *Mate* e associar o *Mate* ao *Job*.

O teste é considerado como sucesso se o método retornar uma exceção.

#### **4.5.2 Método Create**

**TC1 – CanCreateJob:**

O TC1 tem como objetivo testar se o método *Create* adiciona à base de dados, um *Job*, retorna o *Job* adicionado e marca a publicação como terminada. Para este teste é necessário instanciar e adicionar à BD um *Employer*, um *JobPosts* e um *Mate*. De seguida instancia-se um *Job*, associa-se os Ids dos objetos criados anteriormente aos campos corretos, e usa-se o método *Create*.

O teste passa se o *Job* criado for retornado, se toda a informação retornada corresponde ao *Job* instanciado e por final se a publicação de trabalho estiver marcada como concluída.

**TC2 – JobNullWhenSendingNonExistentMateId:**

O TC2tem como objetivo testar se o método *Create* retorna *null* quando se tenta adicionar um *Job* à base de dados com o Id de um mate que não existe. Para este teste basta instanciar um *employer* e um *JobPost* e adicionar à BD. Depois instancia-se um *Job* em que o *MateId* é um número aleatório, por exemplo 100.

O teste passa se o *Job* retornado for *null*.

**TC3 – JobNullWhenSendingNonExistentPostId:**

O TC2tem como objetivo testar se o método *Create* retorna *null* quando se tenta adicionar um *Job* à base de dados com o Id de um *mate* que não existe. Para este teste basta instanciar um *Employer* e um *Mate* e adicionar à BD. Depois instancia-se um *Job* em que o *JobPostId* é um número aleatório, por exemplo 100.

O teste passa se o Job retornado for *null*.

#### **4.5.3 Método FindById**

**TC1 – CanFindWorkById:**

O TC1 tem como objetivo testar se o método *FindById* retorna o *Job* correspondente ao Id que entra como parâmetro. Para este teste é necessário instanciar e adicionar à BD um *Employer*, um *JobPosts* e um *Mate*. De seguida instancia-se um Job e usa-se o método *Create*. Depois de o *Job* estar adicionado, usa-se o método *FindById* e verifica-se se os campos do objeto retornado são iguais ao do Job instanciado anteriormente.

O teste passa se todos os campos forem iguais.

**TC2 – CanFindByIdReturnNullWhenWorkNonExistent:**

O TC2 tem como objetivo testar se o método FindById retorna null caso não exista um Job com o id que entre como parâmetro. Para este teste é necessário instanciar e adicionar à BD um *Employer*, um *JobPosts* e um *Mate*. De seguida instancia-se um Job e usa-se o método *Create*. Depois de o *Job* estar adicionado, usa-se o método *FindById com um id diferente do Job criado.*

*O teste passa se o método retornar null.*

#### **4.5.4 Método Delete**

**TC1 – CanDeleteWork:**

O TC1 tem como objetivo testar se o método Delete remove o Work pretendido da base de dados. Neste teste, é necessário instanciar um Mate, um Employer, um JobPost e um Work. O teste retorna True caso o Work pretendido seja removido com sucesso.

**TC2 – ReturnFalseWhenDeleteNonExistentWork:**

O TC2 tem como objetivo testar se o método Delete devolve False quando se tenta remover um Work que não exista. Neste teste, é necessário instanciar um Mate, um Employer, um JobPost e um Work. O teste retorna False caso o id enviando para o método Delete não corresponda ao de nenhum trabalho existente.

#### **4.5.5 Método Update**

**TC1 – CanUpdateDate:**

O TC1 tem como objetivo testar se o método Update atualiza a data de um Work pretendido. Neste teste, é necessário instanciar um Mate, um Employer, um JobPost e um Work. O teste retorna True caso o Work pretendido tenha a sua data atualizada com sucesso.

**TC2 – ReturnFalseWhenUpdateDate:**

O TC2 tem como objetivo testar se o método Update devolve False caso se tente alterar a data de um trabalho que não exista. Neste teste, é necessário instanciar um Mate, um Employer, um JobPost e um Work. O teste retorna False caso o Id do Work enviado para o método Update de maneira a atualizar a data não corresponda ao de nenhum trabalho existente.

### **4.6** **Casos de teste das features do componente *LoginDAO***

#### **4.7.1 Método Authenticate**

**TC1 – CanEmployerLoginWithCorrectPasswordTest:**

O TC1 tem como objetivo testar se *Employer* consegue fazer o *login* com a *password* correta. É chamado o método *VerifyHash* para verificar se a *password* é a mesma com que o *Employer* se encontra registado. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de verdadeiro.

**TC2 – CanEmployerLoginWithWrongPasswordTest:**

O TC2 tem como objetivo testar se o *Employer* consegue fazer o *login* com a *password* incorreta. É chamado o método *VerifyHash* para verificar se a *password* é a mesma com que o *Employer* se encontra registado. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de falso.

**TC3 – CanMateLoginWithCorrectPasswordTest:**

O TC3 tem como objetivo testar se o *Mate* consegue fazer o *login* com a *password* correta. É chamado o método *VerifyHash* para verificar se a *password* é a mesma com que o *Mate* se encontra registado. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de verdadeiro.

**TC4 – CanMateLoginWithWrongPasswordTest:**

O TC4 tem como objetivo testar se o *Mate* consegue fazer o *login* com a *password* incorreta. É chamado o método *VerifyHash* para verificar se a *password* é a mesma com que o *Mate* se encontra registado. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de falso.

**TC5 – FindByEmailTest:**

O TC5 tem como objetivo testar se o método *FindUserByEmail* retorna um utilizador válido com sucesso. Este método é importante para verificar se um utilizador existe para fazer o *login*. Para o teste ser considerado como bem-sucedido, o método deve retornar não nulo.

**TC6 – FindByEmailNullTest:**

O TC6 tem como objetivo testar se o método *FindUserByEmail* não retorna um utilizador válido com sucesso. Este método é importante para verificar se um é inexistente e consequentemente impossibilitado de fazer *login*. Para o teste ser considerado como bem-sucedido, o método deve retornar nulo.

### **4.7** **Casos de teste das features do componente *ReportDAO***

#### **4.7.1 Método ReportUser**

**TC1 – CanUserAddReportTest:**

O TC1 tem como objetivo testar se um utilizador consegue reportar outro com sucesso. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de verdadeiro.

### **4.8** **Casos de teste das features do componente *ReviewEmployerDAO***

#### **4.8.1 Método ReviewEmployer**

**TC1 – CanMateAddRatingTest:**

O TC1 tem como objetivo testar se um mate consegue dar um *review*/*rating* a um *Employer*. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de verdadeiro.

### **4.9** **Casos de teste das features do componente *ReviewMateDAO***

#### **4.9.1 Método ReviewMate**

**TC1 – CanEmployerAddReviewTest:**

O TC1 tem como objetivo testar se um *Employer* consegue fazer um *review* a um *Mate*. Para o teste ser considerado como bem-sucedido, o método deve retornar um booleano de verdadeiro.

### **4.10 Casos de teste das features do componente *DistanceHelper***

#### **4.10.1 Método GetAddressFromCoordinates**

**TC1 – ReturnAddressWhenCoordinatesAreValid:**

O TC1 tem como objetivo testar se o método retorna um endereço quando as coordenadas *GPS* são válidas. Para o teste é só necessário enviar um conjunto de coordenadas válidas.

**TC2 – ReturnExceptionWhenCoordinatesAreInvalid**

O TC2 tem como objetivo testar se o método retorna uma exceção quando as coordenadas *GPS* são inválidas. Para o teste é só necessário enviar um conjunto de coordenadas inválidas.

### **4.11** **Casos de teste das features do componente *PaymentDAO***

#### **4.11.1 Método GetInvoiceByID**

**TC1 – CanGetInvoiceByID:**

O TC1 tem como objetivo testar se o método retorna um *Invoice* quando é requisitado por um dos envolvidos no trabalho.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento do mesmo.

O teste é considerado como sucesso quando o método retorna o *Invoice* com as mesmas características que o pagamento.

**TC2 – ReturnExceptionWhenJobIDIsInvalidOnGetInvoiceByID**

O TC2 tem como objetivo testar se o método retorna um*a* exceção quando é enviado como argumento um *ID* de trabalho inválido.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento do mesmo.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC3– ReturnExceptionWhenUserIDIsInvalidOnGetInvoiceByID**

O TC3 tem como objetivo testar se o método retorna um*a* exceção quando é enviado como argumento um *ID* de Utilizador inválido.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento do mesmo.

O teste é considerado como sucesso quando o método retorna uma exceção.

#### **4.11.2 Método MakePayment**

**TC1 – CanMakePayment:**

O TC1 tem como objetivo testar se o método retorna um *Invoice* quando é realizado o pagamento pelo *Employer*.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho e marcar o trabalho como concluído.

O teste é considerado como sucesso quando o método retorna o *Invoice* com os mesmos dados que foram enviados no pagamento.

**TC2 –** **ReturnExceptionWhenEmployerIDIsInvalidOnMakePayment:**

O TC2 tem como objetivo testar se o método retorna uma exceção quando é realizado o pagamento pelo *Employer* utilizando um *EmployerID* inválido.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho e marcar o trabalho como concluído.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC3 –** **ReturnExceptionWhenJobIDIsInvalidOnMakePayment:**

O TC3 tem como objetivo testar se o método retorna uma exceção quando é realizado o pagamento pelo *Employer* utilizando um *JobID* inválido.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate,* associar o mesmo trabalho e marcar o trabalho como concluído.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC4 –** **ReturnExceptionWhenWorkNotConfirmedAsDoneByBothPartiesOnMakePayment:**

O TC4 tem como objetivo testar se o método retorna uma exceção quando é realizado o pagamento pelo *Employer* quando nenhum dos utilizadores marcou o trabalho como concluído.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate* e associar o mesmo trabalho.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC5 –** **ReturnExceptionWhenWorkNotConfirmedAsDoneByEmployerOnMakePayment:**

O TC5 tem como objetivo testar se o método retorna uma exceção quando é realizado o pagamento pelo *Employer* quando o *Employer* não marcou o trabalho como concluído.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate* e associar o mesmo trabalho.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC6 –** **ReturnExceptionWhenWorkNotConfirmedAsDoneByMateOnMakePayment:**

O TC6 tem como objetivo testar se o método retorna uma exceção quando é realizado o pagamento pelo *Employer* quando o *Mate* não marcou o trabalho como concluído.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate* e associar o mesmo trabalho.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC6 –** **ReturnExceptionWhenPriceIsIncorrect:**

O TC6 tem como objetivo testar se o método retorna uma exceção quando é realizado o pagamento pelo *Employer* quando o mesmo não paga o valor acordado.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho e marcar o trabalho como concluído.

O teste é considerado como sucesso quando o método retorna uma exceção.

#### **4.11.2 Método ConfirmPayment**

**TC1 – CanConfirmPayment:**

O TC1 tem como objetivo testar se o método retorna um booleano verdadeiro quando o *Mate* confirma o pagamento.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento.

O teste é considerado como sucesso quando o método retorna um booleano com verdadeiro.

**TC2 – ReturnExceptionWhenJobIDIsInvalidOnConfirmPayment:**

O TC2 tem como objetivo testar se o método retorna uma exceção quando o *Mate* tentaconfirmar o pagamento com o *JobID* inválido.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC3 – ReturnExceptionWhenMateIDIsInvalidOnConfirmPayment:**

O TC3 tem como objetivo testar se o método retorna uma exceção quando o *Mate* tentaconfirmar o pagamento com o *MateID* inválido.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento.

O teste é considerado como sucesso quando o método retorna uma exceção.

**TC4 – ReturnExceptionWhenPaymentWasNotDoneOnConfirmPayment:**

O TC4 tem como objetivo testar se o método retorna uma exceção quando o *Mate* tentaconfirmar o pagamento e o trabalho não foi confirmado como pago.

Para o teste é necessário criar um *Employer*, associar um trabalho, criar um *Mate*, associar o mesmo trabalho, marcar o trabalho como concluído e realizar o pagamento.

O teste é considerado como sucesso quando o método retorna uma exceção.

### **4.12 Casos de teste das features do componente *ChatDAO***

#### **4.12.1 Método AddMessage**

**TC1 – CanAddMessage:**

O TC1 tem como objetivo testar se o método retorna *true* quando um utilizador envia uma mensagem a outro utilizador online, e o servidor tenta guardar a mensagem na BD. Para o teste passar é necessário o utilizador enviar a mensagem.

O teste é considerado com sucesso quando o método retorna *true* e a mensagem é adicionada a BD.

**TC2 – CanAddMessageOfflineUser:**

O TC2 tem como objetivo testar se o método retorna *true* quando um utilizador envia uma mensagem a outro utilizador offline, e o servidor tenta guardar a mensagem na BD. Para o teste passar é necessário o utilizador enviar a mensagem.

O teste é considerado com sucesso quando o método retorna *true* e a mensagem é adicionada a BD.

#### **4.12.2 Método CreateChat e getChatIdWithTargetUser**

**TC1 – CanCreateChatAndGetId:**

O TC1 tem como objetivo testar se o método cria um chat, e testa também se o chat foi criado para dois utilizadores.

O teste passa quando o chat e criado e o id de chat é igual para dois utilizadores.

**TC2 - ReturnExceptionCreatingChatWithId0:**

O TC2 tem como objetivo testar se o método retorna uma exceção ao criar um chat com id igual a 0.

O teste passa quando é retornada uma exceção.

**TC3 - ReturnExceptionCreatingChatWithUserId0:**

O TC3 tem como objetivo testar se o método retorna uma exceção ao criar um chat com id igual a 0 de um utilizador.

O teste passa quando é retornada uma exceção.

#### **4.12.3 Método CreateChatId**

**TC1 – CanCreateChatId:**

O TC1 tem como objetivo testar se o método cria um id para o chat.

O teste passa quando o método retorna o id de chat criado.

#### **4.12.4 Método GetMessageList**

**TC1 – CanGetMessages:**

O TC1 tem como objetivo testar se o método retorna uma lista de mensagens de um chat.

O teste passa quando o utilizador abre um chat ao qual pertence e recebe as mensagens todas.

#### **4.12.5 Método GetChats**

**TC1 – CanGetChats:**

O TC1 tem como objetivo testar se o método retorna todos os chats ao qual o utilizador pertence.

O teste passa quando o utilizador recebe a sua lista de chats.

#### **4.12.6 Método AddChatHubConnection**

**TC1 – CanAddChatHubConnection:**

O TC1 tem como objetivo guardar a *connectionId* do ChatHub na BD, com o ID do utilizador que iniciou a conexão.

O teste passa se o método adicionar à Bd a string de conexão do ChatHub referente ao utilizador que inciou a conexão com o Hub.

#### **4.12.7 Método DeleteChatHubConnection**

**TC1 – CanDeleteChatHubConnections:**

O TC1 tem como objetivo apagar uma *connectionId* da base de dados.

O teste passa quando a *connectionId* enviada é apagada e referente a um utilizador que deu *logout*.

**TC2 – ReturnExceptionDeletingInvalidHubConnection:**

O TC2 tem como objetivo retornar uma exceção caso o método seja usado para apagar uma *connectionId* inválida ou inexistente.

O teste passa se for retornada uma exceção.

#### **4.12.8 Método UpdateChatHubConnection**

**TC1 – CanUpdateChatHubConnection:**

O TC1 tem como objetivo atualizar uma *connectionId* da base de dados.

O teste passa quando a *connectionId* pertencente a um utilizador que inicia uma conexão é atualizada.

#### **4.12.9 Método GetChatConnectionFromConnectionId**

**TC1 – CanGetChatConnectionFromConnectionId:**

O TC1 tem como objetivo retornar um *chatId* a partir de uma connectionId dada.

O teste passa quando o *chatId* retornado pertence ao utilizador com a connectionId dada.

#### **4.12.10 Método GetChatConnectionFromUserId**

**TC1 – CanGetChatConnectionFromUserId:**

O TC1 tem como objetivo retornar um *chatId* a partir de um userId.

O teste passa quando o *chatId* retornado pertence ao utilizador com o userId dado.

#### **4.12.11 Método GetLastMessageFromChat**

**TC1 - CanGetLastMessageFromChat:**

O TC1 tem como objetivo retornar uma lista de mensagens pertencentes a um chat ao qual o utilizador pertence.

O teste passa quando são retornadas as mensagens de um chat ao qual o utilizador pertence.

#### **4.12.12 Método IsUserFromChat**

**TC1 – CanVerifyIfUserIsFromChat:**

O TC1 tem como objetivo retornar true se um utilizador pertencer a um chat.

O teste passa quando é retornado true.

**TC2 – CanVerifyIfUserIsNotFromChat:**

O TC1 tem como objetivo retornar false se um utilizador não pertencer a um chat.

O teste passa quando é retornado false.

## **5. Critérios de passagem ou falha das features**

Os critérios de passagem e de falha das features foram definidos em classes de equivalência e em intervalos de input com a técnica de BVA. As subsecções a seguir pretendem mostrar para cada feature, as classes de equivalência válidas e inválidas e os intervalos de BVA (caso se aplique) de modo a definir se a feature passa ou não.

### **5.1 ECs para as features do componente *UserDAO***

#### **5.1.1 Método UpdatePassword**

**Classes de equivalência**

**EC1** – Válido: Os inputs são a antiga palavra passe, a nova palavra passe e o id do utilizador que se pretende editar a palavra passe. O resultado é True caso a alteração tenha sido realizada com sucesso.

**EC2** – Inválido: Os inputs são a antiga palavra passe, a nova palavra passe e o id do utilizador que se pretende editar a palavra passe. O resultado é False caso a palavra passe antiga introduzida seja diferente da verdadeira palavra passe introduzida.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 3 | 0, >3 ou <3 |
| Tipo de Entrada | Um número inteiro que representa um id de utilizador, e dois objetos de PasswordUpdateModel | Um número inteiro menor que 1 ou nulo. Dois objetos diferentes de PasswordUpdateModel, ou o objeto de PasswordUpdateModel correspondente à antiga palavra passe que seja diferente à antiga palavra passe |
| Valor Específico | EC1 | EC2 |

#### **5.1.2 Método FindUserByEmail**

**Classes de equivalência**

**EC1** – Válido: O input é o email do utilizador que se pretende procurar. O resultado esperado é o user com os seus dados.

**EC2** – Inválido: O input é o email do utilizador que se pretende procurar. O resultado esperado é um user nulo, uma vez que o utilizador não existe.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Uma string com um email de um utilizador | Uma string inválida ou nula |
| Valor Específico | EC1 | EC2 |

#### **5.1.3 Método FindById**

**Classes de equivalência**

**EC1** – Válido: O input é o Id do utilizador pretendido. O resultado é um objeto *User* com a informação relativa ao utilizador com o id de parâmetro.

**EC2** – Inválido: O input é o Id do utilizador pretendido. O resultado é um objeto *User* com a informação diferente relativa ao utilizador pretendido.

**EC3** – Válido: O input é o Id de um utilizador inexistente. O resultado é null.

**EC4** – Inválido: O input é o Id de um utilizador inexistente. O resultado é um objeto *User* com a informação de um utilizador.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id de utilizador | Um número inteiro menor que 1 ou nulo. |
| Valor Específico | EC1, EC3 | EC2, EC4 |

### **5.2 ECs para as features do componente *EmployerDAO***

#### **5.2.1 Método Create**

**Classes de equivalência**

**EC1** – Válido: O input é o *Employer* com as informações. O resultado é o Employer retornado e adicionado á base de dados.

**EC2 –** Inválido: O input é o *Employer* com as informações. O resultado é um objeto retornado diferente de Employer e não ser adicionado á base de dados.

**EC3** – Válido: O input é o *Employer* com as informações exceto a descrição. O resultado é o Employer retornado e adicionado á base de dados sem a descrição.

**EC4 –** Inválido: O input é o *Employer* com as informações exceto a descrição. O resultado é um objeto retornado diferente de Employer e não ser adicionado á base de dados sem a descrição.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um objeto de Employer | Um objeto diferente de Employer |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.2.2 Método Update**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o *Employer* com as informações atualizadas e o Id do *Employer* que se pretende atualizar. O resultado é o Employer com as informações atualizadas.

**EC2 –** Inválido: Os inputs são o *Employer* com as informações atualizadas e o Id do *Employer* que se pretende atualizar. O resultado é o Employer com as informações antigas.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id do Employer, e um objeto de Employer | Um número inteiro menor que 1 ou nulo. Um objeto diferente de Employer |
| Valor Específico | EC1 | EC2 |

#### **5.2.3 Método FavoriteList**

**Classes de equivalência**

**EC1** – Válido: O input é Id do *Employer* que pretende aceder à sua lista de *Mates* favoritos. O resultado é uma coleção que representa a lista de *Mates* favoritos desse mesmo *Employer*.

**EC2 –** Inválido: O input é Id do *Employer* que pretende aceder à sua lista de *Mates* favoritos. O resultado é uma coleção que representa a lista de *Mates* favoritos que não corresponde à real.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id do Employer | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.2.4 Método AddFavorite**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o Id do *Employer* que pretende adicionar um *Mate* à sua lista de *Mates* favoritos e o Id do *Mate* a adicionar. O resultado é o Id do *Mate* adicionado.

**EC2 –** Inválido: Os inputs são o Id do *Employer* que pretende adicionar um *Mate* à sua lista de *Mates* favoritos e o Id do *Mate* a adicionar. O resultado é o *Mate* não ter sido adicionado à lista de favoritos.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id do Employer, um número inteiro que representa um id do Mate | Dois números inteiros menores que 1 ou nulos |
| Valor Específico | EC1 | EC2 |

#### **5.2.5 Método RemoveFavorite**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o Id do *Employer* que pretende remover um *Mate* da sua lista de *Mates* favoritos e o Id do *Mate* a remover. O resultado é o Id do *Mate* removido.

**EC2 –** Inválido: Os inputs são o Id do *Employer* que pretende remover um *Mate* da sua lista de *Mates* favoritos e o Id do *Mate* a remover. O resultado é o *Mate* continuar a existir na lista de favoritos.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id do Employer, um número inteiro que representa um id do Mate | Dois números inteiros menores que 1 ou nulos |
| Valor Específico | EC1 | EC2 |

#### **5.2.6 Método FindEmployerById**

**Classes de equivalência**

**EC1** – Válido: O input é Id do *Employer* a pesquisar. O resultado é o *Employer* pesquisado.

**EC2 –** Inválido: O input é Id do *Employer* a pesquisar. O resultado é não retornar nenhum *Employer*.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id do Employer | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

### **5.3 ECs para as features do componente *MateDAO***

#### **5.3.1 Método Create**

**EC1** – Válido: O input é o *Mate* com as informações. O resultado é o Mate retornado e adicionado á base de dados.

**EC2 –** Inválido: O input é o *Mate* com as informações. O resultado é um objeto retornado diferente de Mate e não ser adicionado á base de dados.

**EC3** – Válido: O input é o *Mate* com as informações exceto a descrição. O resultado é o Mate retornado e adicionado á base de dados sem a descrição.

**EC4 –** Inválido: O input é o *Mate* com as informações exceto a descrição. O resultado é um objeto retornado diferente de Mate e não ser adicionado á base de dados sem a descrição.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um objeto de Mate | Um objeto diferente de Mate |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.3.2 Método CategoriesList**

**Classes de equivalência**

**EC1** – Válido: O input é Id do *Mate* que pretende aceder à sua lista de categorias de trabalho. O resultado é uma coleção que representa a lista das categorias de trabalho desse mesmo *Mate*.

**EC2 –** Inválido: O input é Id do *Mate* que pretende aceder à sua lista de categorias de trabalho. O resultado é uma coleção vazia.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id do Mate | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.3.3 Método AddCategory**

**Classes de equivalência**

**EC1** – Válido: Os inputs são as categorias de trabalho que se pretende adicionar e o id do *Mate* que as pretende adicionar. O resultado é ter o *Mate* com as categorias de trabalho adicionadas.

**EC2** – Inválido: Os inputs são as categorias de trabalho que se pretende adicionar e o id do *Mate* que as pretende adicionar. O resultado é ter o *Mate* sem as categorias de trabalho adicionadas.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id do Mate, e um array de objetos de CategoryModel | Um número inteiro menor que 1 ou nulo. Um array de objetos diferentes de CategoryModel |
| Valor Específico | EC1 | EC2 |

#### **5.3.4 Método RemoveCategory**

**Classes de equivalência**

**EC1** – Válido: Os inputs são a categoria de trabalho a remover e o id do *Mate* que a pretende remover. O resultado é ter o *Mate* sem a categoria de trabalho removida.

**EC2** – Inválido: Os inputs são a categoria de trabalho a remover e o id do *Mate* que a pretende remover. O resultado é ter o *Mate* com a categoria de trabalho removida.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id do Mate, e um objeto de CategoryModel | Um número inteiro menor que 1 ou nulo. Um objeto diferente de CategoryModel |
| Valor Específico | EC1 | EC2 |

#### **5.3.5 Método Update**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o *Mate* com as informações atualizadas e o Id do *Mate* que se pretende atualizar. O resultado é o *Mate* com as informações atualizadas.

**EC2 –** Inválido: Os inputs são o *Mate* com as informações atualizadas e o Id do *Mate* que se pretende atualizar. O resultado é o *Mate* com as informações antigas.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id do Mate, e um objeto de Mate | Um número inteiro menor que 1 ou nulo. Um objeto diferente de Mate |
| Valor Específico | EC1 | EC2 |

#### **5.3.6 Método FindMateById**

**Classes de equivalência**

**EC1** – Válido: O input é Id do *Mate* a pesquisar. O resultado é o *Mate* pesquisado.

**EC2 –** Inválido: O input é Id do *Mate* a pesquisar. O resultado é não retornar nenhum *Mate*.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id do Mate | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.3.7 Método PendingJobsList**

**Classes de equivalência**

**EC1** – Válido: O input é Id do *Mate* que pretende aceder à sua lista de trabalhos pendentes. O resultado é uma coleção que representa a lista de trabalhos pendentes desse mesmo *Mate*.

**EC2 –** Inválido: O input é Id do *Mate* que pretende aceder à sua lista de trabalhos pendentes. O resultado é uma coleção diferente daquela que deveria ser apresentada.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id do Mate | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.3.8 Método GetMates**

**Classes de equivalência**

**EC1** – Válido: O input é um *array* de categorias, morada, *rank*, distância e *rating*. O resultado é uma coleção que representa a lista de *mates* resultante da pesquisa com os filtros de input.

**EC2 –** Inválido: O input é um *array* de categorias, morada, *rank*, distancia e *rating*. O resultado é uma coleção que representa a lista de *mates* não correspondente com os filtros de input.

**EC3** – Válido: O input é um *array* de categorias, morada, *rank*. O resultado é uma coleção que representa a lista de *mates* resultante da pesquisa com os filtros de input.

**EC4 –** Inválido: O input é um *array* de categorias, morada, *rank*. O resultado é uma coleção que representa a lista de *mates* não correspondente com os filtros de input.

**EC5** – Válido: O input é um *array* de categorias, morada, *rank* e distancia. O resultado é uma coleção que representa a lista de *mates* resultante da pesquisa com os filtros de input.

**EC6 –** Inválido: O input é um *array* de categorias, morada, *rank* e distancia. O resultado é uma coleção que representa a lista de *mates* não correspondente com os filtros de input.

**EC7** – Válido: O input é um *array* de categorias e morada. O resultado é uma coleção que representa a lista de *mates* resultante da pesquisa com os filtros de input.

**EC8 –** Inválido: O input é um *array* de categorias e morada. O resultado é uma coleção que representa a lista de *mates* não correspondente com os filtros de input.

**EC9** – Válido: O input é um *array* de categorias com uma categoria e morada. O resultado é uma coleção que representa a lista de *mates* resultante da pesquisa com os filtros de input.

**EC10 –** Inválido: O input é um *array* de categorias com uma categoria e morada. O resultado é uma coleção que representa a lista de *mates* não correspondente com os filtros de input.

**EC11** – Válido: O input é um *array* de categorias, *rank*, distancia e *rating*. O resultado é uma coleção que representa a lista de *mates* resultante da pesquisa com os filtros de input.

**EC12 –** Inválido: O input é um *array* de categorias, *rank*, distancia e *rating*. O resultado é uma coleção que representa a lista de *mates* não correspondente com os filtros de input.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 5 | 0, >5 ou <1 |
| Tipo de Entrada | array de categorias, objeto Address, rank, número inteiro de distancia e double de rating | Qualquer parâmetro diferente de array de categorias, objeto Address, rank, número inteiro de distancia e double de rating |
| Valor Específico | EC1, EC3, EC5, EC7, EC9, EC11 | EC2, EC4, EC6, EC8, EC10 |

#### **5.3.9 Método IgnoreJobPost**

**Classes de equivalência**

**EC1** – Válido: O input é um *Job* e um *MateID*. O resultado é um booleano verdadeiro e o trabalho não aparece no feed.

**EC2 –** Inválido: O input é um *Job* e um *MateID*. O resultado é não retornar booleano ou o trabalho aparece no feed.

**EC3** – Válido: O input é um *Job inválido* e um *MateID*. O resultado é um booleano falso.

**EC4 –** Inválido: O input é um *Job inválido* e um *MateID*. O resultado é não retornar é um booleano falso.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 |
| Tipo de Entrada | Job, MateID | Qualquer parâmetro diferente Job ou MateID |
| Valor Específico | EC1, EC3 | EC2, EC4 |

### **5.4 ECs para as features do componente *JobDAO***

#### **5.4.1 Método AddPayment**

**Classes de equivalência**

**EC1** – Válido: Os inputs são métodos de pagamento e o id do JobPost pretendido. O resultado é ter o JobPost com o método de pagamento adicionado.

**EC2** – Inválido: Os inputs são métodos de pagamento e o id do JobPost pretendido. O resultado é ter o JobPost sem o método de pagamento adicionado

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id de JobPost, e um array de objetos de PaymentModel | Um número inteiro menor que 1 ou nulo. Um array de objetos diferentes de PaymentModel |
| Valor Específico | EC1 | EC2 |

#### **5.4.2 Método RemovePayment**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o método de pagamento a remover e o id do JobPost pretendido. O resultado é ter o JobPost sem o método de pagamento removido.

**EC2** – Inválido: Os inputs são o método de pagamento a remover e o id do JobPost pretendido. O resultado é ter o JobPost com o método de pagamento removido.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro que representa um id de jobPost, e um objeto de PaymentModel | Um número inteiro menor que 1 ou nulo. Um objeto diferente de PaymentModel |
| Valor Específico | EC1 | EC2 |

#### **5.4.3 Método** **GetJobs**

**Classes de equivalência**

**EC1** – Válido: O input é um *array* de categorias, morada, distância e rating. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC2 –** Inválido: O input é um *array* de categorias, morada, distância e rating. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC3** – Válido: O input é um *array* de categorias e morada. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC4 –** Inválido: O input é um *array* de categorias e morada. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC5** – Válido: O input é um *array* de categorias, morada e distância. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC6 –** Inválido: O input é um *array* de categorias, morada e distância. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC7** – Válido: O input é um *array* de categorias e rating. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC8 –** Inválido: O input é um *array* de categorias e rating. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC9** – Válido: O input é um *array* de categorias com uma categoria e rating. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC10 –** Inválido: O input é um *array* de categorias com uma categoria e rating. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC11** – Válido: O input é um *array* de categorias. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC12 –** Inválido: O input é um *array* de categorias. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC13** – Válido: O input é um *array* de categorias com uma categoria. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC14 –** Inválido: O input é um *array* de categorias com uma categoria. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC15** – Válido: O input é um *array* de categorias e morada. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC16 –** Inválido: O input é um *array* de categorias e morada. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC17** – Válido: O input é vazio. O resultado é uma coleção que representa a lista de jobs resultante da pesquisa com os filtros de input.

**EC18 –** Inválido: O input é vazio. O resultado é uma coleção que representa a lista de jobs não correspondente com os filtros de input.

**EC19** – Válido: O input é um *array* de categorias e morada inválida. O resultado é uma lista vazia.

**EC20 –** Inválido: O input é um *array* de categorias e morada inválida. O resultado é diferente de uma lista vazia.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | <=4 | >4 |
| Tipo de Entrada | Categorias, Endereço, Distância e Rating | Qualquer parâmetro diferente de Categorias, Endereço, Distancia e Rating |
| Valor Específico | EC1, EC3, EC5, EC7, EC9, EC11, EC13, EC15, EC17, EC19 | EC2, EC4, EC6, EC8, EC10, EC14, EC16, , EC18, , EC20 |

#### **5.4.4 Método MakeOfferOnJob**

**Classes de equivalência**

**EC1** – Válido: O input é uma *Offer* e um *MateID*. O resultado é uma *Offer* válida de acordo com o enviado.

**EC2 –** Inválido: O input é uma *Offer* e um *MateID*. O resultado é uma *Offer* inválida de acordo com o enviado.

**EC3** – Válido: O input é uma *Offer sem preço* e um *MateID*. O resultado é uma *Offer* válida de acordo com o enviado.

**EC4 –** Inválido: O input é uma *Offer sem preço* e um *MateID*. O resultado é uma *Offer* inválida de acordo com o enviado.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 |
| Tipo de Entrada | Offer, MateID | Qualquer parâmetro diferente Offer ou MateID |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.4.5 Método Create**

**Classes de equivalência**

**EC1** – Válido: O input é um objeto do tipo JobPost sem caminho para imagens. O resultado é ter o JobPost adicionado a base de dados sem caminho para imagens, na tabela JobPost, e retornar o JobPost.

**EC2** – Inválido: O input é um objeto do tipo JobPost sem caminho para imagens. O resultado é ter um objeto diferente de JobPost adicionado a base de dados na tabela JobPost, e não retornar o JobPost.

**EC3** – Válido: O input é um objeto do tipo JobPost. O resultado é ter o JobPost adicionado a base de dados na tabela JobPost, e retornar o JobPost.

**EC4** – Inválido: O input é um objeto do tipo JobPost. O resultado é ter um objeto diferente de JobPost adicionado a base de dados na tabela JobPost, e não retornar o JobPost.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um objeto do tipo jobPost | Um objeto do tipo diferente de jobPost |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.4.6 Método Delete**

**Classes de equivalência**

**EC1** – Válido: O input é um número inteiro. O output é verdadeiro e o JobPost foi apagado da base de dados

**EC2** – Inválido: O input é um número inteiro. O output é falso e o JobPost existe na base de dados

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id de jobPost | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.4.7 Método FindById**

**Classes de equivalência**

**EC1** – Válido: O input é um número inteiro. O output é um objeto JobPost com a informação pretendida.

**EC2** – Inválido: O input é um número inteiro. O output é um objeto diferente do tipo JobPost com a informação pretendida.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id de jobPost | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.4.8 Método GetEmployerPosts**

**Classes de equivalência**

**EC1** – Válido: O input é um número inteiro. O output é uma lista de objetos do tipo JobPost com o id do employer igual ao número de input.

**EC2** – Inválido: O input é um número inteiro. O output é uma lista de objetos diferentes do tipo JobPost com o id do employer igual ao número de input.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id de Employer | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.4.9 Método UpdatePostDetails**

**Classes de equivalência**

**EC1** – Válido: O input é um objeto JobPost com id igual ao objeto na Base de dados a atualizar. O output é um objeto do tipo JobPost com os campos atualizados. O objeto na base de dados também é atualizado

**EC2** – Inválido: O input é um objeto JobPost com id igual ao objeto na Base de dados a atualizar. O output é um objeto diferente do tipo JobPost com os campos atualizados. O objeto na base de dados também é atualizado

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um objeto do tipo JobPost | Um objeto do tipo diferente de JobPost |
| Valor Específico | EC1 | EC2 |

### **5.5 ECs para as features do componente *WorkDAO***

#### **5.5.1 Método MarkJobAsDone**

**Classes de equivalência**

**EC1** – Válido: O input é um *JobID* e um *UserID*. O resultado é um booleano verdadeiro.

**EC2 –** Inválido: O input é um *JobID* e um *UserID*. O resultado é não retornar booleano.

**EC3** – Válido: O input é um *JobID inválido* e um *UserID*. O resultado é uma Exception.

**EC4 –** Inválido: O input é um *JobID inválido* e um *UserID*. O resultado é não retornar uma Exception.

**EC5** – Válido: O input é um *JobID* e um *UserID inválido*. O resultado é uma Exception.

**EC6 –** Inválido: O input é um *JobID* e um *UserID inválido*. O resultado é não retornar uma Exception.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 |
| Tipo de Entrada | *JobID*, *UserID* | Qualquer parâmetro diferente *JobID* ou *UserID* |
| Valor Específico | EC1 | EC2 |

#### **5.5.2 Método Create**

**Classes de equivalência**

**EC1** – Válido: O input é um objeto do tipo Job e o ID de employer. O resultado é adicionar o Job á base de dados, retorna o Job adicionado e marcar a publicação como concluída.

**EC2** – Inválido: O input é um objeto do tipo Job e o ID de employer. O resultado é não adicionar o Job á base de dados, retornar um objeto diferente de Job ou não marcar a publicação como concluída.

**EC3** – Válido: O input é um objeto do tipo Job, sem o mateId e o ID de employer. O resultado é retornar null.

**EC4** – Inválido: O input é um objeto do tipo Job, sem o mateId e o ID de employer. O resultado é retornar algo diferente de null.

**EC5** – Válido: O input é um objeto do tipo Job sem o JobPostId e o ID de employer. O resultado é retornar null.

**EC6** – Inválido: O input é um objeto do tipo Job sem o JobPostId e o ID de employer. O resultado é retornar algo diferente de null.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >1 ou <1 |
| Tipo de Entrada | Um objeto do tipo Job e um número inteiro | Um objeto do tipo diferente de Job, e um número inteiro menor que 1 ou igual a zero ou nulo |
| Valor Específico | EC1, EC3, EC5 | EC2, EC4, EC6 |

#### **5.5.3 Método FindById**

**Classes de equivalência**

**EC1** – Válido: O input é um número inteiro. O resultado é um Job com Id correspondente ao número de input.

**EC2** – Inválido: O input é um número inteiro. O resultado é um Job com Id diferente ao número de input.

**EC1** – Válido: O input é um número inteiro. O resultado é null porque o Job com o id igual ao de input não existe.

**EC2** – Inválido: O input é um número inteiro. O resultado é diferente de null.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Número inteiro maior que 0 | Número inteiro menor que 0 ou igual a zero |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.5.4 Método Delete**

**Classes de equivalência**

**EC1** – Válido: O input é um número inteiro. O resultado é True caso o Work seja removido com sucesso.

**EC2** – Inválido: O input é um número inteiro. O resultado é False porque o id do Work enviado não existe.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id de Work | Um número inteiro menor que 1 ou nulo |
| Valor Específico | EC1 | EC2 |

#### **5.5.5 Método Update**

**Classes de equivalência**

**EC1** – Válido: O input é um número inteiro e um objeto do tipo DateTime. O resultado é True caso a data do Work seja alterada com sucesso.

**EC2** – Inválido: O input é um número inteiro e um objeto do tipo DateTime. O resultado é False porque o id do Work enviado não existe.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >1 ou <1 |
| Tipo de Entrada | Um número inteiro que representa um id de Work, um objeto de DateTime | Um número inteiro menor que 1 ou nulo, um objeto diferente de DateTime |
| Valor Específico | EC1 | EC2 |

### **5.6 ECs para as features do componente *LoginDAO***

#### **5.6.1 Método Authenticate**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o email e a password do utilizador que pretende fazer login. O resultado esperado é um user válido com um email e password existentes.

**EC2** – Inválido: O input é o email e a password do utilizador que pretende fazer login. O resultado esperado é um user nulo, uma vez que o utilizador não existe ou a password é nula/incorreta.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Duas strings representado um email e password respetivamente | As duas strings serem inválidas ou nulas |
| Valor Específico | EC1 | EC2 |

### **5.7 ECs para as features do componente *ReportDAO***

#### **5.7.1 Método ReportUser**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o id do utilizador que irá reportar, o id do utilizador a ser reportado e o report com as informações. O resultado esperado é o Report retornado e adicionado á base de dados.

**EC2** – Inválido: Os inputs são o id do utilizador que irá reportar, o id do utilizador a ser reportado e o report com as informações. O resultado esperado é um objeto retornado que seja diferente de Report, sendo assim impossível de ser adicionado à base de dados.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 3 | 0, >3 ou <3 |
| Tipo de Entrada | Dois números inteiros representando os *ids* dos utilizadores e um objeto de Report | Dois números inteiros inválidos ou nulos. Um objeto diferente de Report |
| Valor Específico | EC1 | EC2 |

### **5.8 ECs para as features do componente *ReviewEmployerDAO***

#### **5.8.1 Método ReviewEmployer**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o id do employer que irá ser avaliado e o Review com as informações. O resultado esperado é o Review retornado e adicionado á base de dados.

**EC2** – Inválido: Os inputs são o id do employer que irá ser avaliado e o Review com as informações. O resultado esperado é um objeto retornado que seja diferente de Review, sendo assim impossível de ser adicionado à base de dados.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro representando o id do employer e um objeto de Review | Um número inteiro inválido ou nulo. Um objeto diferente de Review |
| Valor Específico | EC1 | EC2 |

### **5.9 ECs para as features do componente *ReviewMateDAO***

#### **5.9.1 Método ReviewMate**

**Classes de equivalência**

**EC1** – Válido: Os inputs são o id do mate que irá ser avaliado e o MateReview com as informações. O resultado esperado é o MateReview retornado e adicionado á base de dados.

**EC2** – Inválido: Os inputs são o id do mate que irá ser avaliado e o MateReview com as informações. O resultado esperado é um objeto retornado que seja diferente de MateReview, sendo assim impossível de ser adicionado à base de dados.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 ou <2 |
| Tipo de Entrada | Um número inteiro representando o id do mate e um objeto de MateReview | Um número inteiro inválido ou nulo. Um objeto diferente de MateReview |
| Valor Específico | EC1 | EC2 |

### **5.10 ECs para as features do componente *DistanceHelper***

#### **5.10.1 Método GetAddressFromCoordinates**

**Classes de equivalência**

**EC1** – Válido: O input é uma *latitude* e uma *longitude*. O resultado é um address.

**EC2 –** Inválido: O input é uma *latitude* e uma *longitude*. O resultado é não retornar address.

**EC3** – Válido: O input é uma *latitude* e uma *longitude inválidos*. O resultado é uma *Exception*.

**EC4 –** Inválido: O input é uma *latitude* e uma *longitude inválidos*. O resultado é não retornar uma *Exception*.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 |
| Tipo de Entrada | *latitude*, *longitude* | Qualquer parâmetro diferente *latitude* ou *longitude* |
| Valor Específico | EC1, EC3 | EC2, EC4 |

### **5.11 ECs para as features do componente *PaymentDAO***

#### **5.10.1 Método GetInvoiceByID**

**Classes de equivalência**

**EC1** – Válido: O input é um *UserID* e um *JobID*. O resultado é um *Invoice* válido.

**EC2 –** Inválido: O input é um *UserID* e um *JobID*. O resultado é um *Invoice* inválido.

**EC3** – Válido: O input é um *UserID* inválido e um *JobID*. O resultado é um*a Exception*.

**EC4 –** Inválido: O input é um *UserID* inválido e um *JobID*. O resultado é diferente um*a Exception*.

**EC5** – Válido: O input é um *UserID* e um *JobID* inválido. O resultado é um*a Exception*.

**EC6 –** Inválido: O input é um *UserID* e um *JobID* inválido. O resultado é diferente um*a Exception*.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 |
| Tipo de Entrada | *UserID, JobID* | Qualquer parâmetro diferente *UserID* ou *JobID* |
| Valor Específico | EC1, EC3, EC5 | EC2, EC4, EC6 |

#### **5.10.2 Método MakePayment**

**Classes de equivalência**

**EC1** – Válido: O input é um *Invoice,* um *JobID e EmployerID*. O resultado é um *Invoice* válido.

**EC2 –** Inválido: O input é um *Invoice,* um *JobID e EmployerID*. O resultado é um *Invoice* inválido.

**EC**3– Válido: O input é um *Invoice,* um *JobID e EmployerID* inválido. O resultado é um*a Exception*.

**EC4 –** Inválido: O input é um *Invoice,* um *JobID e EmployerID* inválido. O resultado é diferente um*a Exception*.

**EC**5– Válido: O input é um *Invoice,* um *JobID* inválido *e EmployerID*. O resultado é um*a Exception*.

**EC6 –** Inválido: O input é um *Invoice,* um *JobID* inválido *e EmployerID*. O resultado é diferente um*a Exception*.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 3 | 0, >3 |
| Tipo de Entrada | *UserID, JobID, EmployerID* | Qualquer parâmetro diferente de *UserID,* *JobID, EmployerID* |
| Valor Específico | EC1 | EC2 |

#### **5.10.3 Método ConfirmPayment**

**Classes de equivalência**

**EC1** – Válido: O input é um *JobID* e um *MateID*. O resultado é um booleano verdadeiro.

**EC2 –** Inválido: O input é um *JobID* e um *MateID*. O resultado é não retornar booleano.

**EC3** – Válido: O input é um *JobID inválido* e um *MateID*. O resultado é um*a Exception*.

**EC4 –** Inválido: O input é um *JobID inválido* e um *MateID*. O resultado é diferente um*a Exception*.

**EC5** – Válido: O input é um *JobID* e um *MateID inválido*. O resultado é um*a Exception*.

**EC6 –** Inválido: O input é um *JobID* e um *MateID inválido*. O resultado é diferente um*a Exception*.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 0, >2 |
| Tipo de Entrada | *JobID*, *MateID* | Qualquer parâmetro diferente de *JobID* ou *MateID* |
| Valor Específico | EC1, EC3, EC5 | EC2, EC4, EC6 |

### **5.11 ECs para as features do componente *ChatDAO***

#### **5.11.1 Método AddMessage**

**Classes de equivalência**

**EC1** – Válido: O user destinatário está online. O input do user remetente é a mensagem, e a app envia o input, as connectionIDs, o Id do destinatário do chat e a hora da mensagem. O resultado é a mensagem aparecer no chat e ser guardada na BD.

**EC2 –** Inválido: O user destinatário está online. O input do user remetente é vazio, e a app envia valores não correspondentes ao input, às connectionIDs, ao Id do destinatário do chat e á hora da mensagem. O resultado é a mensagem aparecer no chat e ser guardada na BD.

**EC3** – Válido: O user destinatário está offline. O input do user remetente é a mensagem, e a app envia o input, as connectionsIDs, o Id do destinatário do chat e a hora da mensagem. O resultado é a mensagem aparecer no chat para o utilizador online e ser guardada na BD.

**EC4 –** Inválido: O user destinatário está offline O input do user remetente é vazio, e a app envia valores não correspondentes ao input, às connectionIDs, o Id do destinatário do chat e á hora da mensagem. O resultado é a mensagem aparecer no chat para o utilizador online e ser guardada na BD.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 5 | 5 |
| Tipo de Entrada | string targetConnectionId, string senderConnectionId, string messageText, DateTime time, int targetId | Tudo diferente de:  string targetConnectionId, string senderConnectionId, string messageText, DateTime time, int targetId |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.11.2 Método CreateChat e getChatIdWithTargetUser**

##### CreateChat

**Classes de equivalência**

**EC1** – Válido: O utilizador inicia um chat. A aplicação envia o Id do utilizador e o id de chat gerado num objeto “Chat”. É adicionado na Bd o chatId associado ao utilizador.

**EC2 –** Inválido: O utilizador inicia um chat. A aplicação envia o Id do utilizador e o id de chat gerado num objeto “Chat”. Não é adicionado na Bd o chatId associado ao utilizador.

**EC3** – Válido: O utilizador inicia um chat. A aplicação envia o Id do utilizador e o id de chat igual a zero num objeto “Chat”. Não é adicionado na Bd o chatId associado ao utilizador e retorna uma exceção.

**EC4 –** Inválido: O utilizador inicia um chat. A aplicação envia o Id do utilizador e o id de chat igual a zero num objeto “Chat”. È adicionado na Bd o chatId associado ao utilizador.

**EC5** – Válido: O utilizador inicia um chat. A aplicação envia o Id do utilizador igual a zero e o id de chat num objeto “Chat”. Não é adicionado na Bd o chatId associado ao utilizador e retorna uma exceção.

**EC6 –** Inválido: O utilizador inicia um chat. A aplicação envia o Id do utilizador igual a zero e o id de chat num objeto “Chat”. É adicionado na Bd o chatId associado ao utilizador.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | Objeto chat | Objeto diferente de Chat |
| Valor Específico | EC1, EC3, EC5 | EC2, EC4, EC6 |

##### GetChatIdWithTragetUser

**Classes de equivalência**

**EC1** – Válido: A aplicação envia os Ids de dois utilizadores e verifica se o dois tem um chat em comum. O id do chat é retornado.

**EC2 –** Inválido: A aplicação envia inputs diferentes dos Ids de dois utilizadores e verifica se o dois tem um chat em comum. O id do chat é retornado.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 2 |
| Tipo de Entrada | 2 números inteiros | 2 inputs diferentes de inteiro |
| Valor Específico | EC1 | EC2 |

#### **5.11.3 Método CreateChatId**

**Classes de equivalência**

**EC1** – Válido: A aplicação cria um chatId diferente de 0 quando o chat é criado. O chat é criado com o Id gerado.

**EC2 –** Inválido: A aplicação cria um chatId diferente de 0 quando o chat é criado. O chat é criado com o Id gerado.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | Invocar o método sem argumentos | Invocar outro método diferente do CreateChatId |
| Valor Específico | EC1 | EC2 |

#### **5.11.4 Método GetMessageList**

**Classes de equivalência**

**EC1** – Válido: O utilizador pede a lista de mensagens de um chat e a aplicação mobile faz um pedido com o id do chat do utilizador. São retornadas as mensagens desse chat.

**EC2 –** Inválido: O utilizador pede a lista de mensagens de um chat que não é dele e a aplicação mobile faz um pedido com o id do chat. São retornadas as mensagens desse chat.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | Int Id do chat, Int Id do utilizador | Dois parâmetros diferentes de número inteiro |
| Valor Específico | EC1 | EC2 |

#### **5.11.5 Método GetChats**

**Classes de equivalência**

**EC1** – Válido: O utilizador pede a sua lista de chats, e a api retorna a lista de chats do utilizador com Id correspondente ao que vem no token de acesso.

**EC2 –** Inválido: O utilizador pede a sua lista de chats, e a api retorna a lista de chats do utilizador com Id diferente do que vem no token de acesso.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | Invocar o método GetChats | Invocar outro método diferente do GetChats |
| Valor Específico | EC1 | EC2 |

#### **5.11.6 Método AddChatHubConnection**

**Classes de equivalência**

**EC1** – Válido: O utilizador conecta-se ao chat e a api adiciona a sua connectionId á Bd.

**EC2 –** Inválido: O utilizador conecta-se ao chat e a api não adiciona aconnectionId á BD

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 2 |
| Tipo de Entrada | Int Id do utilizador, string connectionId | Dois argumentos diferentes de Int e string |
| Valor Específico | EC1 | EC2 |

#### **5.11.7 Método DeleteChatHubConnection**

**Classes de equivalência**

**EC1** – Válido: O user desconecta do chat, a api invoca o método delete para a connectionId desse user. A connectionId é apagada da BD.

**EC2 –** Inválido: O user desconecta do chat, a api invoca o método delete para a connectionId desse user. A connectionId não é apagada da BD.

EC3 – Válido: O user desconecta do chat, a api invoca o método delete para a connectionId não existente ou inválida. É retornada uma exceção.

EC4 – Inválido: : O user desconecta do chat, a api invoca o método delete para a connectionId não existente ou inválida. Não é retornada uma exceção.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | String connectionId | Tudo o que for diferente de string |
| Valor Específico | EC1, EC3 | EC2, EC4 |

#### **5.11.8 Método UpdateChatHubConnection**

**Classes de equivalência**

**EC1** – Válido: O utilizador reconecta ao chat, a api atualiza a sua connectionId. A connectionId é atualizada na BD.

**EC2 –** Inválido: O utilizador reconecta ao chat, a api atualiza a sua connectionId. A connectionId não é atualizada na BD.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 2 |
| Tipo de Entrada | Int userId, string connectionId | Tudo diferente de int e string relativamente aos parâmetros |
| Valor Específico | EC1 | EC2 |

#### **5.11.9 Método GetChatConnectionFromConnectionId**

**Classes de equivalência**

**EC1** – Válido: O utilizador obtém o userId e a connectionID a partir da connectionId. Um obtejo ChatConnection é retornado com a informação.

**EC2 –** Inválido: O utilizador obtém o userId e a connectionID a partir da connectionId. Um obtejo ChatConnection é retornado com informação diferente da pretendida.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | String connectionId | String connectionId |
| Valor Específico | EC1 | EC2 |

#### **5.11.10 Método GetChatConnectionFromUserId**

**Classes de equivalência**

**EC1** – Válido: O utilizador obtém o userId e a connectionID a partir do userId. Um obtejo ChatConnection é retornado com a informação.

**EC2 –** Inválido: O utilizador obtém o userId e a connectionID a partir do userId. Um obtejo ChatConnection é retornado com informação diferente da pretendida.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 1 | 1 |
| Tipo de Entrada | Int userId | Int userId |
| Valor Específico | EC1 | EC2 |

#### **5.11.11 Método GetLastMessageFromChat**

**Classes de equivalência**

**EC1** – Válido: A aplicação mobile solicita a api a última mensagem de um chat do user que usa a app. A última mensagem é retornada.

**EC2 –** Inválido: A aplicação mobile solicita a api a última mensagem de um chat do user que usa a app. A última mensagem é retornada.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 2 |
| Tipo de Entrada | Int chatId, int userId | Parâmetros com tipos diferentes de inteiro |
| Valor Específico | EC1 | EC2 |

#### **5.11.12 Método IsUserFromChat**

**Classes de equivalência**

**EC1** – Válido: A api verifica se o utilizador com id de parâmetro pertence ao chat com id de parâmetro. Retorna true se pertencer.

**EC2 –** Inválido: A api verifica se o utilizador com id de parâmetro pertence ao chat com id de parâmetro. Retorna false se pertencer.

**EC3** – Válido: A api verifica se o utilizador com id de parâmetro pertence ao chat com id de parâmetro. Retorna false se não pertencer.

**EC4 –** Inválido: A api verifica se o utilizador com id de parâmetro pertence ao chat com id de parâmetro. Retorna true se não pertencer.

|  |  |  |
| --- | --- | --- |
| Critério | Classe de equivalência válida | Classe de equivalência inválida |
| Nº de Inputs | 2 | 2 |
| Tipo de Entrada | Int userId, int chatId | Parâmetros diferentes de números inteiros |
| Valor Específico | EC1, EC3 | EC2, EC4 |