

Departamento de Informática

Licenciatura em Engenharia Informática

UDO

Relatório de Estágio Curricular

Setembro 2014

João Luís Fazendeiro Campos (34626)

Orientador: Miguel Monteiro

COordenador Externo: Nuno Bragado

InstituIção: findmore Consulting

Agradecimentos

Agradecimentos. Página impar.

Resumo

Índice Geral

1 Introdução 1

1.1 Contexto Académico 2

1.2 Contexto Tecnológico e Científico 2

1.3 Objectivos do estágio 2

1.4 Descrição do Projecto 2

1.5 Estrutura do documento 2

2 Actividades Desenvolvidas 5

2.1 Integração na empresa 7

2.2 Integração no projecto 7

2.3 Reunião com o orientador e com o coordenador 7

2.4 Recolha de requisitos e formação do problema 7

2.5 Pesquisa e formação autodidacta 8

2.6 Tecnologias utilizadas 9

2.6.1 Node.js 9

2.6.2 MongoDB 9

2.6.3 Phonegap 9

2.7 Análise Funcional 9

2.8 Análise não funcional 9

2.9 Diagrama de casos de uso 9

2.10 Descrição dos casos de uso 9

2.11 Diagrama de actividades para os casos de uso 9

2.12 Implementação do projecto 9

2.12.1 Ambiente de desenvolvimento 9

2.12.2 Ambiente de testes 9

2.13 Competências 9

3 Conclusões 10

3.1 Apreciação Crítica do Trabalho Desenvolvido 11

3.2 Trabalho Futuro 11

3.3 Apreciação do Estágio 11

4 Bibliografia 13

5 Anexos 15

5.1 Guia de instalação do servidor de suporte à aplicação 16

5.1.1 Descrição 16

5.1.2 Pré-requisitos 16

5.1.3 Importação do repositório 16

5.1.4 Instalação dos módulos 17

5.1.5 Execução da API 18

5.1.6 Executar a API em ambiente de produção 19

5.2 Manual de utilizador 21

5.3 Representational State Transfer Application Programming Interface (REST API) 21

Índice de Figuras

Figura 1 **Error! Bookmark not defined.**

Figura 2. **Error! Bookmark not defined.**

Índice de Tabelas

**No table of figures entries found.**

# Introdução

1.1 Contexto Académico 2

1.2 Contexto Tecnológico e Científico 2

1.3 Objectivos do estágio 2

1.4 Descrição do projecto 2

1.5 Estrutura do documento 2

## Contexto Académico

## Contexto Tecnológico e Científico

## Objectivos do estágio

## Descrição do Projecto

## Estrutura do documento

# Actividades Desenvolvidas

1 Introdução 1

1.1 Contexto Académico 2

1.2 Contexto Tecnológico e Científico 2

1.3 Objectivos do estágio 2

1.4 Descrição do Projecto 2

1.5 Estrutura do documento 2

2 Actividades Desenvolvidas 5

2.1 Integração na empresa 7

2.2 Integração no projecto 7

2.3 Reunião com o orientador e com o coordenador 7

2.4 Recolha de requisitos e formação do problema 7

2.5 Pesquisa e formação autodidacta 8

2.6 Escolha das tecnologias a utilizar 8

2.6.1 Node.js 8

2.6.2 MongoDB 8

2.6.3 Phonegap 8

2.7 Análise Funcional 8

2.8 Análise não funcional 8

2.9 Diagrama de casos de uso 8

2.10 Descrição dos casos de uso 8

2.11 Diagrama de actividades para os casos de uso 8

2.12 Implementação do projecto 8

2.12.1 Ambiente de desenvolvimento 9

2.12.2 Ambiente de testes 9

2.13 Competências 9

3 Conclusões 10

3.1 Apreciação Crítica do Trabalho Desenvolvido 11

3.2 Trabalho Futuro 11

3.3 Apreciação do Estágio 11

4 Bibliografia 13

5 Anexos 15

5.1 Guia de instalação do servidor de suporte à aplicação 16

5.2 Manual de utilizador 16

5.3 Representational State Transfer Application Programming Interface (REST API) 16

## Integração na empresa

A integração na empresa deu-se no dia 24 de Março de 2014 onde foram facultados recursos para serem utilizados durante o estágio. Foram alocados um portátil, acesso à rede interna da empresa e um endereço *email* com o domínio da empresa.

## Integração no projecto

Aquando o início do estágio o projecto estava ainda em fase embrionária. Havia apenas uma proposta para a aplicação onde existiam *mockups* para o *design* da mesma e um pouco sobre o objectivo da aplicação. Mas nada de muito completo pois faltava, claro está, o levantamento dos requisitos por parte da equipa de desenvolvimento.

## Reunião com o orientador e com o coordenador

A primeira e única reunião com o coordenador na empresa e com o orientador da FCT deu-se no dia 9 de Maio de 2014. Uma reunião meramente de apresentação quer entre o coordenador e orientador quer com o estagiário e com o orientador.

## Recolha de requisitos e formação do problema

A recolha dos requisitos ocorreu numa reunião informal com o representante da CPDJ[[1]](#footnote-1) Miguel Galhardas. Foi falado sobre o objectivo da aplicação e sobre quais as principais funcionalidades da mesma.

Chegou-se à conclusão o projecto a implementar seria uma aplicação móvel que para uma primeira fase permita a adição de combates de judo (mediante as regras intrínsecas do mesmo no que respeita a técnicas e à pontuação de um combate) e que os dados deste gerem estatísticas gerais - número de vitorias e derrotas, percentagem de vitórias - e estatíticas específicas – percentagem de derrotas e vitórias com uma técnica em específico - a fim de o *judoca* poder fazer uma avaliação a si próprio com o objectivo de melhorar a sua estratégia e até técnica de combate. A aplicação funcionará como uma pequena rede social – no sentido em que um utilizador pode ter amigos – onde, para além de se poder adicionar combates contra outros *judocas*, existe a possibilidade de adicionar combates contra um amigo, desafiá-lo e até comparar estatísticas com ele.

Outras funcionalidades que a aplicação terá são partilhar os dados nas redes sociais mais conhecidas: *Facebook*, *Google*+ e *Twitter*; filtragem das estatísticas através de vários filtros como por exemplo por oponente.

Assim, no fim da recolha concluiu-se que o problema a implementar seria a criação de uma aplicação móvel que se assemelhe a uma rede social onde se possa enviar dados para um servidor e obter esses mesmo dados sempre que fosse pedido e que esses dados estejam sempre atualizados. No fundo, o desafio é a criação de uma aplicação que funcione como uma aplicação *Web* comum.

## Pesquisa e formação autodidacta

Devido ao meu contacto com a criação de aplicações *Web* ter existido apenas na cadeira de *Desenvolvimento de Aplicações para a Web* (*DaWeb*) e também ao facto de não ter nenhuma experiência com o desenvolvimento de aplicações móveis tive que ter alguma formação nas duas áreas. Formação essa que ocorreu de forma “semi-autodidacta”.

Comecei pelo desenvolvimento de aplicações *web* com *Node.js* – sugestão de Nuno Bragado visto ser uma tecnologia em voga e ser uma mais-valia para mim. Comecei por ver uns tutoriais e a estudar alguns livros sobre Node.js e o gosto por esta tecnologia começou a crescer.

Durante a minha pesquisa sobre Node.js encontrei uma *framework* denominada de MEAN (MongoDB, Express.js, AngularJS and Node.js) - da qual apenas utilizei o MongoDB, o Node.js e o Express.js (que é um pacote para o Node.js) - que me apresentou o MongoDB, uma tecnologia com que nunca me tinha deparado ou sabia que existia algo como, por exemplo, *NoSQL Databases*. Nesta fase adquiri formação necessária para começar a desenvolver um *backend* consistente.

Mas, ainda existia a necessidade de criar uma aplicação móvel que comunica-se com o *backend*. Assim, pesquisei sobre formas de criar aplicações *Web-Based* para dispositivos móveis e deparei-me com uma tecnologia denominada de *Phonegap* que me permitia utilizar linguagens *web*, entre elas *javascript* (que também utilizava em Node.js) para criar uma aplicação nativa para dispositivos móveis. Para esta tecnologia a ambientação deu-se através de tutoriais que foram aparecendo na minha pesquisa.

## Tecnologias utilizadas

Do ponto anterior retém-se que serão utilizadas três tecnologias: Node.js, MongoDB e Phonegap. Nesta secção irá constar uma breve explicação do que cada tecnologia se trata.

### Node.js

O Node.js é uma plataforma que funciona sobre o *runtime* de *Javascript* do *Chrome* e permite que sejam construídas *network applications* de forma rápida e escalável.

O Node utiliza um modelo de programação orientada a eventos e um modelo de *non-blocking I/O* que torna esta plataforma leve, eficiente e perfeita para aplicações em tempo-real que executem através de serviços distribuídos.

Um dos aspectos que torna esta tecnologia tão eficiente é o *non-blocking I/O* que permite que uma aplicação continue a sua execução mesmo que uma chamada de *I/O* seja efectuada. O Node funciona através de funções de *callback*, ou seja, caso uma chamada ao sistema (por exemplo, ler a informação dentro de um ficheiro) seja efectuada o programa prossegue a sua execução (isto é, pode inclusivé receber e executar novos pedidos) sem parar. E assim que a chamada ao sistema terminar o programa “salta” para a função de *callback,* executando-a.

Outro aspecto que torna esta tecnologia tão interessante é o facto de estarmos a utilizar uma linguagem que maioritariamente aparece na parte do cliente, o JavaScript, e estarmos a usá-la no lado do servidor. O Node está a ganhar cada vez mais notoriedade sendo que cada vez mais “gigantes” das tecnologias estão a utilizar esta tecnologia. “Gigantes” como: Paypal, eBay, Yahoo, LinkedIn e até a Microsoft.

Foi esta a tecnologia escolhida para a implementação do *backend* da aplicação pois para além de todos os aspectos mencionados acima mostrou também ser um desafio interessante e que trará mais-valias no futuro.

### MongoDB

O MongoDB é um tipo de base de dados não relacional que se encaixa na categoria de *NoSQL Databases*. O Mongo baseia-se em documentos para guardar a informação, esses documentos tomam um formato semelhante ao formato JSON (ao qual o Mongo chama de BSON – Binary JSON). Este modelo de funcionamento permite que em certas aplicações a integração dos dados seja mais simples e mais rápida.

As principais vantagens do MongoDB são: a alta performance, a alta taxa de disponibilidade e a escalabilidade automática.

Esta tecnologia também tem ganho grande conotação por parte de “gigantes” da tecnologia. Empresas como LinkedIn, Electronic Arts, eBay, entre outras usam o MongoDB nos seus serviços.

A escolha desta tecnologia veio pelo facto de ser um novo desafio e um paradigma diferente daquele que é incutido durante a licenciatura. Pensar em bases de dados onde o conceito de relação e de tabela é inexistente a princípio pode gerar alguma confusão mas por fim já se torna natural o facto desses dois conceitos não existirem no universo do MongoDB.

### Phonegap

O Phonegap é uma *framework* de desenvolvimento para dispositivos móveis que permite a utilização de tecnologias *standard* para a *web* tais como *HTML*, *CSS* e *JavaScript*.

As aplicações criadas com o Phonegap são multiplataforma, isto é, funcionam para Android, iOS, Windows Phone, entre outros sistemas operativos para dispositivos móveis. As aplicações são também consideradas de aplicações híbridas pois não são totalmente aplicações móveis nativas nem aplicações para a *web-based*.

Esta tecnologia mostrou ser a indicada para a aplicação que queríamos implementar pois possibilita a utilização do JavaScript. O acesso ao JavaScript permite utilizar a biblioteca jQuery que dispõe de um conjunto de métodos que permitem efectuar pedidos de AJAX (Asyncrhonous JavaScript and XML). Para além disso evitou que tivesse que obter mais formação em relação ao desenvolvimento de aplicações móveis nativas pois já tinha tido contacto com as tecnologias de HTML, CSS e JavaScript na faculdade e este último também na aprendizagem de Node.js.

## Análise Funcional

Existem duas entidades principais que vão interagir com o sistema. São elas o utilizador e o sistema de *backend* que dá suporte à aplicação. Existe também um actor que representa as redes sociais Facebook, Twitter e Google+ que denominaremos de Redes Sociais.

O Sistema de Informação é onde irá constar toda a informação inerente à aplicação: combates, utilizadores, eventos, etc.

Assim o Sistema deve permitir às usas entidades principais:

Utilizador e Sistema de Informação:

* Efectuar Registo;
* Efectuar *Login*;
* *Recuperar a password;*
* Adicionar um combate;
* Adicionar um ídolo;
* Visualizar dados de um combate;
* Remover um combate;
* Editar um combate;
* Alterar o perfil de utilizador;
* Alterar *Password*;
* Alterar *Email*;
* Adicionar um utilizador como amigo;
* Aceitar/Rejeitar um pedido de amizade;
* Aceitar/Rejeitar um desafio;
* Ver estatísticas globais;
* Adicionar um evento;
* Procurar um utilizador;
* Enviar um convite para a aplicação;
* Ver a lista de amigos;
* Ver desafios;
* Adicionar um *Record;*
* Editar um *Record*;
* Remover um *Record;*
* Comparar estatísticas;
* Ver ranking;
* Filtrar os dados;

Utilizador e Redes Sociais:

* Partilhar os dados;

## Análise não funcional

## Diagrama de casos de uso

## Descrição dos casos de uso

## Diagrama de actividades para os casos de uso

## Implementação do projecto

### Ambiente de desenvolvimento

### Ambiente de testes

## Competências

# 

# Conclusões

4.1 Apreciação Crítica do Trabalho Desenvolvido 13

4.2 Trabalho Futuro 13

4.3 Apreciacão do Estágio 13

## Apreciação Crítica do Trabalho Desenvolvido

## Trabalho Futuro

## Apreciação do Estágio

# Bibliografia

# Anexos

5.1 Guia de instalação do servidor de suporte à aplicação 4

5.2 Manual de utilizador 4

5.3 *Representational State Transfer Application Programming Interface (REST API)* 4

## Guia de instalação do servidor de suporte à aplicação

### Descrição

Este manual tem o propósito de mostrar como instalar e correr a API que dá suporte à aplicação UDO. A API foi desenvolvida sobre a plataforma de *NodeJS* e como base de dados foi utilizado o *MongoDB*.

O manual será apoiado com *screenshots* de uma consola em Windows mas para sistemas UNIX será análogo.

### Pré-requisitos

Antes da importação da API é necessário a instalação dos três *softwares* abaixo mencionados e também ter acesso ao respositório que está alojado *online*.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| <http://www.nodejs.org/> | <http://www.mongodb.org/downloads> | <http://git-scm.com/> |

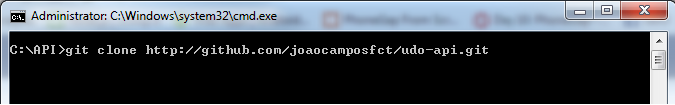


Conta no <http://www.github.com> como colaborador no repositório da *API*.

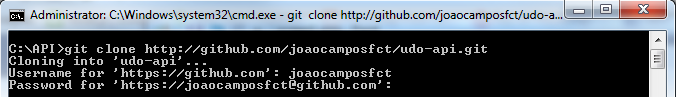
### Importação do repositório

Para importar o repositório que contém a API executar o seguinte comando:

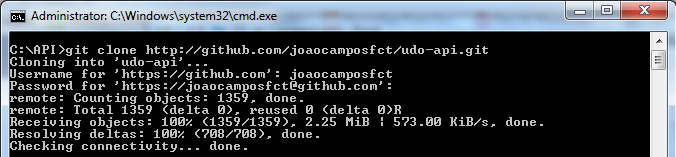
git clone <http://github.com/joaocamposfct/udo-api.git>



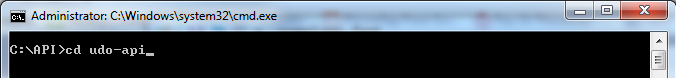
Será requisitado o nome de utilizador e a password do github para conseguir clonar o repositório. Assim quando for necessário inserir essas informações, insira-as.

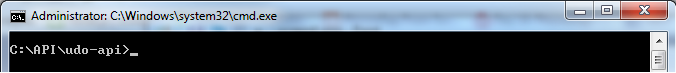


O comando demorará alguns segundos (dependendo da conexão) a importar o repositório, quando completar o output do comando será identico ao que se segue:



Concluído o passo anterior irá reparar que dentro da pasta para onde importou o repositório estará outra pasta denominada de *udo-api*. Deverá direcionar-se para essa pasta (através do comando cd udo-api).

**

**

Agora que já importámos o repositório, na secção a seguir vamos instalar todos os módulos necessários para a execução da API.

### Instalação dos módulos

Nesta secção iremos instalar todos os módulos essenciais à execução do servidor que sustenta a API. Para tal necessitamos de correr o comando:

npm install



Este comando irá buscar uma lista de módulos ao ficheiro package.json – que se encontra na raiz da pasta *udo-api* – e irá importar e instalar cada módulo localmente. Este processo poderá demorar alguns minutos dependendo da conexão que estiver a ser usada.

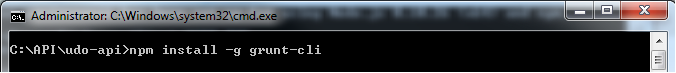
Nota: Poderá ser necessário privilégios de administrador para instalar alguns módulos, assim, certifique-se que tem esses mesmos privilégios antes de executar o comando.

Um último módulo que terá que ser instalado especificamente é o módulo grunt-cli. Este módulo terá que ser instalado globalmente pois o mesmo dá acesso ao comando que nos permitirá executar o servidor.

Para tal, executa-se o comando:

npm install –g grunt-cli

(A opção –g no comando significa que queremos a instalar o módulo globalmente.)



Passemos então à execução da aplicação.

### Execução da API

Para executar a aplicação basta aceder à pasta *udo-api* – caso não tenha acedido previamente – e correr o comando:

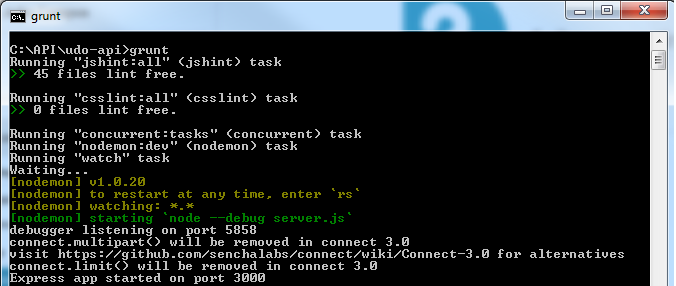
grunt



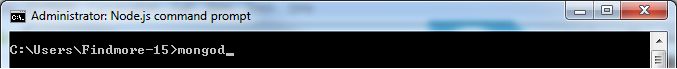
Este comando permitirá executar o servidor. Outra funcionalidade deste comando é que, enquanto está em execução, verifica se existem alterações aos ficheiros da aplicação, caso aconteça, ele reinicializará o servidor.

Nota: Caso não tenha reiniciado a consola onde estivemos a importar e a instalar os módulos poderá ser necessário fazê-lo caso o comando grunt não funcione porque a consola não detecta a presença do mesmo. Teste para ver se o comando é executado, caso contrário reinicie a consola, aceda à pasta onde está a API e execute o comando novamente.

Caso o comando execute com sucesso aparecerão os seguintes outputs (ou algo similar):

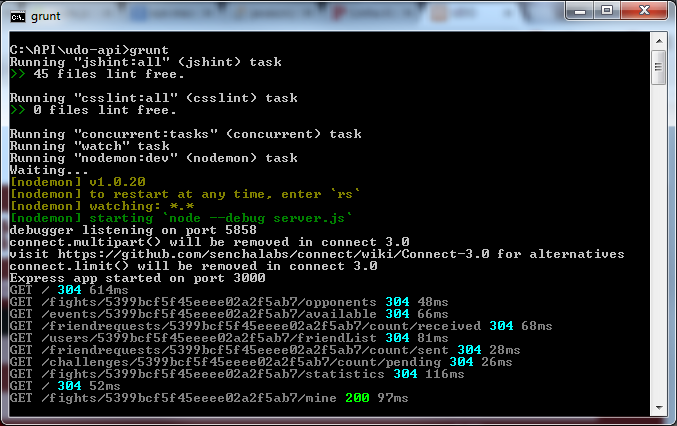


Nota: Para a aplicação estar funcional é necessário que o servidor do MongoDB esteja também em execução. Para tal execute o comando mongod noutra instância da consola ou como serviço.



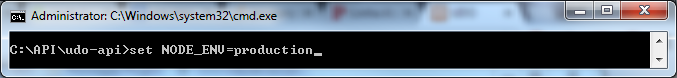
### Executar a API em ambiente de produção

O ambiente de execução por defeito é o de desenvolvimento, isto é, na consola onde a aplicação estiver a correr aparecerão mensagens de *output* com os pedidos e o código HTTP referente à resposta de cada pedido (erro, sucesso ou não modificado). A imagem seguinte representa isso mesmo:



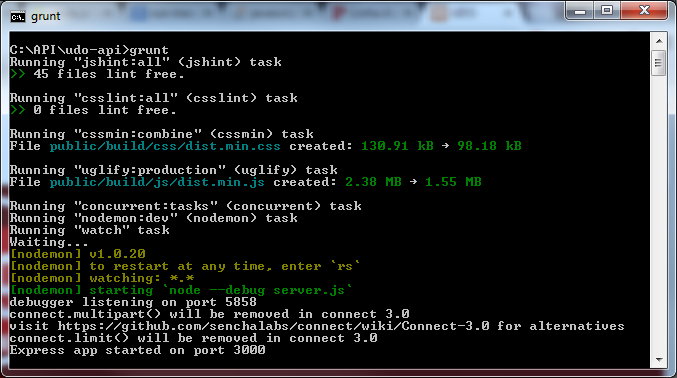
Para correr em ambiente de produção temos que criar e afectar uma variável de ambiente denominada NODE\_ENV, para tal, no *Windows* corremos o seguinte comando:

set NODE\_ENV=production



(Para UNIX o comando seria exports NODE\_ENV=production)

E em seguida correr novamente o comando “grunt” para executar a aplicação.



Sempre que se quiser alterar o ambiente de execução basta afectar a variável NODE\_ENV para ‘development’ para o ambiente de desenvolvimento e para ‘production’ caso queiramos ir para ambiente de produção.

Nota importante: A cada mudança de ambiente de execução a aplicação conecta-se a uma base de dados diferente. A “udo-dev” é a base de dados de desenvolvimento e a “udo” é a de produção.

## Manual de utilizador

Texto do anexo 2

## Representational State Transfer Application Programming Interface (REST API)

A aplicação que é executada no servidor é constituída por sete modelos distintos, cada um com um papel específico na aplicação. São eles os modelos *user*, *challenges*, *events*, *fights*, *friendRequest*, *idols* e *record*.

Para cada modelo da aplicação será feita uma breve descrição e será listado o conjunto de métodos associados a ele. Para cada método será enunciado pedido de *HTTP* inerente ao mesmo, o seu URL (com os parâmetros que o constituem) uma breve descrição e a sua resposta em caso de sucesso e em caso de insucesso e caso seja relevante o corpo do pedido.

*User* – Modelo que representa um utilizador.

|  |  |  |
| --- | --- | --- |
| GET | /logout | Logout |
| GET | /users/search | Search |
| GET | /users/:userId/friendList | friendList |
| GET | /users/:userId | Show |
| GET | /users | All |
| GET | /users/addemail | Addemail |
| GET | /reset/:token | middleForgotPassword |
| POST | /register | Create |
| POST | /users/:userId | Update |
| POST | /users/:userId/changepassword | changePassword |
| POST | /users/:userId/changemail | changeEmail |
| POST | /users/rankings | Rankings |
| POST | /forgot | forgotPassword |
| POST | /reset/:token | finalizeForgotPassword |

|  |  |
| --- | --- |
| Logout | |
| HTTP Request | GET |
| URL | /logout |
| Descrição | Efectua o logout de um utilizador |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Logout bem sucedido | | Resposta | N/A | |

|  |  |
| --- | --- |
| Search | |
| HTTP Request | GET |
| URL | /users/search?name=’…’ |
| Parâmetros | Query: name – parâmetro de pesquisa |
| Descrição | Efectua uma pesquisa sobre utilizadores dado um parâmetro de pesquisa. O parâmetro é comparado com o nome e o email do utilizador. |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | A pesquisa não retornou nenhum resultado. | | Resposta | “There are no users with that name or email” | | Código | 200 | | Condição | A pesquisa foi bem sucedida | | Resposta | O resultado da pesquisa em format JSON | |

|  |  |
| --- | --- |
| *FriendList* | |
| HTTP Request | GET |
| URL | /users/:userId/friendList |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém a lista de amigos de um dado utilizador |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção da lista de amigos foi bem sucedida | | Resposta | *Friendlist* do utilizador em formato JSON. | |

|  |  |
| --- | --- |
|  | Show |
| HTTP Request | GET |
| URL | /users/:userId |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém um utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção do utilizador foi bem sucedida | | Resposta | Obtém o utilizador em formato JSON | |

|  |  |
| --- | --- |
|  | All |
| HTTP Request | GET |
| URL | /users |
| Descrição | Obtém todos os utilizadores. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção dos utilizadores foi bem sucedida | | Resposta | Obtém os utilizadores em formato JSON | |

|  |  |
| --- | --- |
| Addemail | |
| HTTP Request | GET |
| URL | /users/addemail?name=’’ |
| Parâmetros | Query: name – email para enviar o convite |
| Descrição | Envia um email para o endereço indicado para convidar o dono do endereço para se registar e utilizar a aplicação |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | O envio do email foi bem sucedido. | | Resposta | ‘An e-mail has been sent to ‘ + Endereço de email submetido + ‘ inviting him to join UDO!’ | | Código | 400 | | Condição | O endereço submetido já se encontra registado na aplicação | | Resposta | ‘That email is already registered in UDO!’ | |

|  |  |
| --- | --- |
|  | middleForgotPassword |
| HTTP Request | GET |
| URL | /reset/:token |
| Parâmetros | :token – Token que representa a validade do pedido de alteração da password. |
| Descrição | Mostra um formulário que permitirá alterar a password do utilizador que iniciou o processo de mudança de password caso o token esteja dentro do prazo de validade. Mostra uma página de erro caso contrário. |
| Responses | N/A |
| Nota | Esta função apenas será executada fora da aplicação móvel. Utilizando um browser, mesmo que seja de um dispositivo móvel. |

|  |  |
| --- | --- |
|  | Create |
| HTTP Request | POST |
| URL | /register |
| Descrição | Efectua o registo de um utilizador. |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | Caso alguns elementos do corpo do pedido não passem nas validações. | | Resposta | Lista com os erros | | Código | 400 | | Condição | O email utilizado no registo já se encontra registado na aplicação | | Resposta | ‘Email is already in use’ | | Código | 400 | | Condição | Erro na adição do utilizador | | Resposta | ‘Error adding user’ | | Código | 200 | | Condição | Registo bem sucedido | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | Update |
| HTTP Request | POST |
| URL | /users/:userId |
| Descrição | Efectua a edição de um utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Edição bem sucedida | | Resposta | JSON com o utilizador já actualizado. | |

|  |  |
| --- | --- |
|  | changePassword |
| HTTP Request | POST |
| URL | /users/:userId/changePassword |
| Descrição | Altera a password do utilizador indicado. |
| Parâmetros | :userId – Identificador de um utilizador |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | Caso a nova password indicada não cumpra o tamanho entre 8 e 20 caractéres. | | Resposta | ‘Password must be between 8-20 characters long’ | | Código | 200 | | Condição | A actualização seja bem sucedida | | Resposta | N/A | | Código | 401 | | Condição | Caso a password corrente não seja igual à submetida para efeitos de autenticação. | | Resposta | ‘Wrong current password’ | |

|  |  |
| --- | --- |
|  | changeEmail |
| HTTP Request | POST |
| URL | /users/:userId/changeEmail |
| Descrição | Altera o email do utilizador indicado. |
| Parâmetros | :userId – Identificador de um utilizador |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A actualização seja bem sucedida | | Resposta | N/A | | Código | 401 | | Condição | Caso a password ou email correntes não sejam iguais aos submetidos para efeitos de autenticação. | | Resposta | ‘Wrong email or password’ | |

|  |  |
| --- | --- |
|  | Rankings |
| HTTP Request | POST |
| URL | /users/rankings |
| Descrição | Obtém os rankings da aplicação e para um dado utilizador. |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | Caso o utilizador em questão não exista. | | Resposta | ‘User does not exist! Something went wrong’ | | Código | 400 | | Condição | Caso a base de dados não contenha nenhum combate. | | Resposta | ‘There are no fights in the database’ | | Código | 200 | | Condição | Em caso de sucesso | | Resposta | JSON com todos os rankings. | |

|  |  |
| --- | --- |
|  | forgotPassword |
| HTTP Request | POST |
| URL | /forgot |
| Descrição | Cria um token e envia um email para alteração da password. |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | Caso o email indicado não esteja registado. | | Resposta | “An account with such email doesn’t exist” | | Código | 200 | | Condição | Caso o email seja enviado com sucesso | | Resposta | ‘An e-mail has been sent to ‘ + email indicado + ‘ with further instructions’ | |
| Corpo do pedido | email: Email indicado para a alteração da password. |
| Nota | Esta função apenas será executada fora da aplicação móvel. Utilizando um browser, mesmo que seja de um dispositivo móvel. |

|  |  |
| --- | --- |
|  | finalizeForgotPassword |
| HTTP Request | POST |
| URL | /reset/:token |
| Descrição | Finaliza a alteração da password. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a alteração ocorra com sucesso. | | Resposta | 'Success! Your password has been changed, you can now return to the application and login with your new password.' | |
| Nota | Esta função apenas será executada fora da aplicação móvel. Utilizando um browser, mesmo que seja de um dispositivo móvel. |

Challenges – Modelo que representa um desafio que um utilizador pode fazer a um amigo.

|  |  |  |
| --- | --- | --- |
| GET | /challenges/all | listChallenges |
| GET | /challenges/:userId/count/pending | countPending |
| GET | /challenges/:userId/get/pending | getPending |
| GET | /challenges/:userId/get/ongoing | getOngoing |
| GET | /challenges/:userId/get/ended | getEnded |
| GET | /challenges/:userId/:challengeId/decline | declineChallenge |
| POST | /challenges/:userId/:challengeId/accept | acceptChallenge |
| POST | /challenges | createChallenge |

|  |  |
| --- | --- |
|  | listChallenges |
| HTTP Request | GET |
| URL | /challenges/all |
| Descrição | Obtém todos os desafios existentes na base de dados. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os desafios na base de dados. | |

|  |  |
| --- | --- |
|  | countPending |
| HTTP Request | GET |
| URL | /challenges/:userId/count/pending |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Conta o número de desafios pendentes que um utilizador tem. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a contagem dos dados seja bem sucedida. | | Resposta | JSON com o número de desafios pendentes | |

|  |  |
| --- | --- |
|  | getPending |
| HTTP Request | GET |
| URL | /challenges/:userId/get/pending |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém todos os desafios pendentes de um dado utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os desafios pendentes de um utilizador. | |

|  |  |
| --- | --- |
|  | getOngoing |
| HTTP Request | GET |
| URL | /challenges/:userId/get/ongoing |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém todos os desafios de um dado utilizador que ainda não terminaram. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os desafios de um utilizador que ainda estão a decorrer. | |

|  |  |
| --- | --- |
|  | getEnded |
| HTTP Request | GET |
| URL | /challenges/:userId/get/ended |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém todos os desafios de um dado utilizador que já terminaram. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os desafios de um utilizador que já terminaram. | |

|  |  |
| --- | --- |
|  | declineChallenge |
| HTTP Request | GET |
| URL | /challenges/:userId/:challengeId/decline |
| Parâmetros | :userId – Identificador de um utilizador.  :challengeId – Identificador de um desafio. |
| Descrição | Rejeita um desafio |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | acceptChallenge |
| HTTP Request | POST |
| URL | /challenges/:userId/:challengeId/accept |
| Parâmetros | :userId – Identificador de um utilizador.  :challengeId – Identificador de um desafio. |
| Descrição | Aceita um desafio |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | createChallenge |
| HTTP Request | POST |
| URL | /challenges |
| Descrição | Cria um desafio |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | N/A | |

Events – Modelo que representa os eventos onde os combates se passaram.

|  |  |  |
| --- | --- | --- |
| GET | /events | showAllEvents |
| GET | /events/:eventId | showEvent |
| GET | /eventsbyuser/:userId | showEventsByUser |
| GET | /events/:eventId/remove | removeEvent |
| POST | /events/add | createEvent |
| POST | /events/:eventId/update | updateEvent |

|  |  |
| --- | --- |
|  | showAllEvents |
| HTTP Request | GET |
| URL | /events |
| Descrição | Obtém todos os eventos na base de dados. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os eventos na base de dados. | |

|  |  |
| --- | --- |
|  | showEvent |
| HTTP Request | GET |
| URL | /events/:eventId |
| Parâmetros | :eventId – Identificador de um evento. |
| Descrição | Obtém um evento através do seu identificador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com o evento pretendido. | |

|  |  |
| --- | --- |
|  | showEventsByUser |
| HTTP Request | GET |
| URL | /eventsbyuser/:userId |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém todos os eventos criados por um utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os eventos de um determinado utilizador. | |

|  |  |
| --- | --- |
|  | showEventsByUser |
| HTTP Request | GET |
| URL | /eventsbyuser/:userId |
| Parâmetros | :userId – Identificador de um utilizador. |
| Descrição | Obtém todos os eventos criados por um utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados seja bem sucedida. | | Resposta | JSON com todos os eventos de um determinado utilizador. | |

|  |  |
| --- | --- |
|  | removeEvent |
| HTTP Request | GET |
| URL | /events/:eventId/remove |
| Parâmetros | :eventId – Identificador de um evento. |
| Descrição | Remove o evento com o identificador indicado. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso o evento seja removido com sucesso. | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | addEvent |
| HTTP Request | POST |
| URL | /events/add |
| Descrição | Cria um evento. |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | Caso exista um evento duplicado | | Resposta | N/A | | Código | **400** | | Condição | Caso existam campos sobre o evento que não foram preenchidos. | | Resposta | ‘Please fill all the required fields’ | | Código | 400 | | Condição | Existem para o mesmo utilizador dois eventos no mesmo dia | | Resposta | ‘You can not have two events on the same day’ | | Código | **400** | | Condição | Existe um evento do utilizador com o mesmo nome | | Resposta | ‘You already have an event with the same name’ | | Código | 200 | | Condição | Caso o evento seja criado com sucesso. | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | updateEvent |
| HTTP Request | POST |
| URL | /events/:eventId/update |
| Descrição | Actualiza um evento |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso o evento seja actualizado com sucesso. | | Resposta | N/A | |

Fights – Modelo que define os combates de um utilizador. Estes combates podem ser entre utilizadores que sejam amigos, utilizadores contra outro judoca ou de ídolos do utilizador contra outros judocas.

|  |  |  |
| --- | --- | --- |
| GET | /fights/:userId | fightsByUser |
| GET | /fights | allFights |
| GET | /fights/:userId/mine | allUserParticipates |
| GET | /fights/:userId/statistics | retrieveStatistics |
| GET | /fights/:userId/opponents | listOpponents |
| GET | /fights/:userId/stats/byopponent | statsVersusOpponent |
| GET | /fights/:userId/stats/byevent | statsOnEvent |
| GET | /fights/:userId/stats/byfriend | statsVersusFriend |
| GET | /fights/:userId/stats/between | statsBetween |
| GET | /fights/remove/:fightId | removeFight |
| POST | /fights | createFight |
| POST | /fights/update/:fightId | updateFight |

|  |  |
| --- | --- |
|  | fightsByUser |
| HTTP Request | GET |
| URL | /fights/:userId |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém todos os combates inseridos por um utilizador |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com todos os combates que um utilizador inseriu. | |

|  |  |
| --- | --- |
|  | allFights |
| HTTP Request | GET |
| URL | /fights |
| Descrição | Obtém todos os combates na base de dados |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com todos os combates na base de dados. | |

|  |  |
| --- | --- |
|  | allUserParticipates |
| HTTP Request | GET |
| URL | /fights/:userId/mine |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém todos os combates na base de dados em que um determinado utilizador participa. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com os combates em que utilizador participa | |

|  |  |
| --- | --- |
|  | retrieveStatistics |
| HTTP Request | GET |
| URL | /fights/:userId/statistics |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém as estatísticas de um utilizador. Calcula-as através de todos os combates que o utilizador participa. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com as estatísticas | |

|  |  |
| --- | --- |
|  | listOpponents |
| HTTP Request | GET |
| URL | /fights/:userId/opponents |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Devolve todos os oponentes de um determinado utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com todos os oponentes. | |

|  |  |
| --- | --- |
|  | statsVersusOpponent |
| HTTP Request | GET |
| URL | /fights/:userId/stats/byopponent |
| Parâmetros | :userId – Identificador de um utilizador  query – oponente – nome do oponente. |
| Descrição | Devolve as estatísticas de um utilizador contra um determinado oponente. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com as estatísticas contra um determinado oponente. | |

|  |  |
| --- | --- |
|  | statsOnEvent |
| HTTP Request | GET |
| URL | /fights/:userId/stats/byevent |
| Parâmetros | :userId – Identificador de um utilizador  query – event – identificador do evento. |
| Descrição | Devolve as estatísticas de um utilizador contra um determinado oponente. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com as estatísticas num determinado evento. | |

|  |  |
| --- | --- |
|  | statsBetween |
| HTTP Request | GET |
| URL | /fights/:userId/stats/between |
| Parâmetros | :userId – Identificador de um utilizador  query – begin – data de início query - end – data de fim |
| Descrição | Devolve as estatísticas de um utilizador entre duas datas. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção dos dados ocorra com sucesso | | Resposta | JSON com as estatísticas entre duas datas. | |

|  |  |
| --- | --- |
|  | removeFight |
| HTTP Request | GET |
| URL | /fights/remove/:fightId |
| Parâmetros | :fightId – Identificador de um combate |
| Descrição | Remove um combate |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a remoção ocorra com sucesso | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | createFight |
| HTTP Request | POST |
| URL | /fights |
| Descrição | Cria um combate |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a criação ocorra com sucesso | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | updateFight |
| HTTP Request | POST |
| URL | /fights/update/:fightId |
| Parâmetros | :fightId – Identificador de um combate |
| Descrição | Actualiza um combate |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a actualização ocorra com sucesso | | Resposta | N/A | |

FriendRequest – Modelo que define um pedido de amizade entre dois utilizadores.

|  |  |  |
| --- | --- | --- |
| GET | /friendRequests/:userId/all | getAllRequests |
| GET | /friendRequests/:userId/received | getReceivedRequests |
| GET | /friendRequests/:userId/count/received | countReceivedRequests |
| GET | /friendRequests/:userId/count/sent | countSentRequests |
| GET | /friendRequests/:requestId/accept | acceptRequest |
| GET | /friendRequests/:requestId/decline | declineRequest |
| POST | /friendRequests/:userId/addrequest | createRequest |

|  |  |
| --- | --- |
|  | getAllRequests |
| HTTP Request | GET |
| URL | /friendRequests/:userId/all |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém todos os pedidos de amizade de um utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção ocorra com sucesso | | Resposta | JSON com todos os pedidos de amizade. | |

|  |  |
| --- | --- |
|  | getReceivedRequests |
| HTTP Request | GET |
| URL | /friendRequests/:userId/received |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém os pedidos de amizade que um utilizador recebeu e ainda respondeu. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a obtenção ocorra com sucesso | | Resposta | JSON com todos os pedidos de amizade que recebeu e ainda estão pendentes. | |

|  |  |
| --- | --- |
|  | countReceivedRequests |
| HTTP Request | GET |
| URL | /friendRequests/:userId/count/received |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém o número de pedidos de amizade que um utilizador recebeu e ainda respondeu. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a contagem ocorra com sucesso | | Resposta | JSON com o número de pedidos de amizade que o utilizador recebeu e ainda estão pendentes. | |

|  |  |
| --- | --- |
|  | countSentRequests |
| HTTP Request | GET |
| URL | /friendRequests/:userId/count/sent |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém o número de pedidos de amizade que um utilizador enviou. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | Caso a contagem ocorra com sucesso | | Resposta | JSON com o número de pedidos de amizade que o utilizador enviou. | |

|  |  |
| --- | --- |
|  | acceptRequest |
| HTTP Request | GET |
| URL | /friendRequests/:requestId/accept |
| Parâmetros | :requestId – Identificador de um pedido |
| Descrição | Aceita um pedido de amizade. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A aceitação executou com sucesso. | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | declineRequest |
| HTTP Request | GET |
| URL | /friendRequests/:requestId/decline |
| Parâmetros | :requestId – Identificador de um pedido |
| Descrição | Rejeita um pedido de amizade. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A rejeição executou com sucesso. | | Resposta | N/A | |

|  |  |
| --- | --- |
|  | createRequest |
| HTTP Request | POST |
| URL | /friendRequests/:userId/addrequest |
| Parâmetros | :userId – Identificador de um pedido |
| Corpo do pedido | to – Email do utilizador que se pretende adicionar  fromName – Nome do utilizador que enviou o pedido  fromEmail – Email do utilizador que enviou o pedido. |
| Descrição | Cria um pedido de amizade entre dois utilizadores. |
| Responses | |  |  | | --- | --- | | Código | 400 | | Condição | O utilizador tentou enviar um pedido de amizade a si próprio. | | Resposta | ‘You can't send friend requests to yourself!’ | | Código | 400 | | Condição | Caso o pedido seja para um utilizador que já tenha enviado um pedido igual. | | Resposta | ‘That user already sent you a request, check your notifications!’ | | Código | 400 | | Condição | Caso o pedido seja duplicado. | | Resposta | ‘You've already sent an invite to that user’ | | Código | 200 | | Condição | Caso o envio do pedido execute com sucesso. | | Resposta | JSON com o pedido | | Código | **400** | | Condição | O utilizador para o qual se tentou enviar o pedido não existe. | | Resposta | ‘That user doesn't exist’ | |

Idol – Modelo que define um ídolo. Cada utilizador tem os seus ídolos, assim a aplicação permite que cada utilizador tenha uma lista de ídolos e que para cada ídolo existam (ou não) combates realizados pelos mesmos. O objetivo é que um utilizador consiga comparar as suas estatísticas com as estatísticas do seu ídolo.

|  |  |  |
| --- | --- | --- |
| GET | /idols | getAllIdols |
| GET | /idols/:userId | getUserIdols |
| GET | /idols/:idolId/statistics | getIdolsStatistics |
| POST | /idols | createIdol |

|  |  |
| --- | --- |
|  | getAllIdols |
| HTTP Request | GET |
| URL | /idols |
| Descrição | Obtém todos os ídolos na base de dados |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção executou com sucesso. | | Resposta | JSON com todos os ídolos da base de dados | |

|  |  |
| --- | --- |
|  | getUserIdols |
| HTTP Request | GET |
| URL | /idols/:userId |
| Parâmetros | :userId – Identificador de um utilizador |
| Descrição | Obtém todos os ídolos de um determinado utilizador. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção executou com sucesso. | | Resposta | JSON com os ídolos de um determinado utilizador. | |

|  |  |
| --- | --- |
|  | getIdolStatistics |
| HTTP Request | GET |
| URL | /idols/:idolId/statistics |
| Parâmetros | :idolId – Identificador de um ídolo |
| Descrição | Obtém as estatísticas de um determinado ídolo. |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção executou com sucesso. | | Resposta | JSON com as estatísticas de um determinado ídolo. | |

|  |  |
| --- | --- |
|  | createIdol |
| HTTP Request | POST |
| URL | /idols |
| Descrição | Cria um ídolo |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção executou com sucesso. | | Resposta | JSON com as estatísticas de um determinado ídolo. | |

Records – Modelo que define momento na “história” do utilizador enquanto judoca. Tem o objetivo de funcionar de forma semelhante aos “*Life Events”* do *Facebook* mas aplicado ao Judo. Os *records* na aplicação podem ser inseridos manualmente ou aparecer automaticamente caso o sistema detecte que o utilizador melhorou uma estatística, ou mudou de grau no Judo.

|  |  |  |
| --- | --- | --- |
| GET | /recordsbyuser/:userId | getRecordsUser |
| POST | /records/add | createRecord |
| POST | /records/update | updateRecord |
| POST | /records/delete | deleteRecord |

|  |  |
| --- | --- |
|  | getRecordsUser |
| HTTP Request | GET |
| URL | /records/add |
| Descrição | Cria um record |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A obtenção executou com sucesso. | | Resposta | JSON com todos os records de um utilizador. | |

|  |  |
| --- | --- |
|  | createRecord |
| HTTP Request | POST |
| URL | /idols |
| Descrição | Cria um ídolo |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A criação ocorreu com sucesso. | | Resposta | N/A. | | Código | 400 | | Condição | Caso a inserção na base de dados resulte num erro. | | Resposta | ‘Error’ | | Código | 400 | | Condição | Caso já exista um record igual no mesmo dia. | | Resposta | ‘You can't have equal records on the same day.’ | |

|  |  |
| --- | --- |
|  | updateRecord |
| HTTP Request | POST |
| URL | /records/update |
| Descrição | Actualiza um record |
| Responses | |  |  | | --- | --- | | Código | 200 | | Condição | A actualização ocorreu com sucesso. | | Resposta | N/A | | Código | 400 | | Condição | O record não existe na base de dados. | | Resposta | ‘Such record doesn't exist’ | | Código | 400 | | Condição | Caso o record não seja editável. | | Resposta | ‘Error, you can't edit this record’ | | Código | 400 | | Condição | Caso a actualização resulte num record que é igual a outro que já exista no mesmo dia na base de dados. | | Resposta | ‘You can't have equal records on the same day’. | | Código | 400 | | Condição | Caso a inserção na base de dados result num erro. | | Resposta | ‘Error’ | |

1. CPDJ – Centro para o Desenvolvimento do Judo - http://cpdj.org/ [↑](#footnote-ref-1)