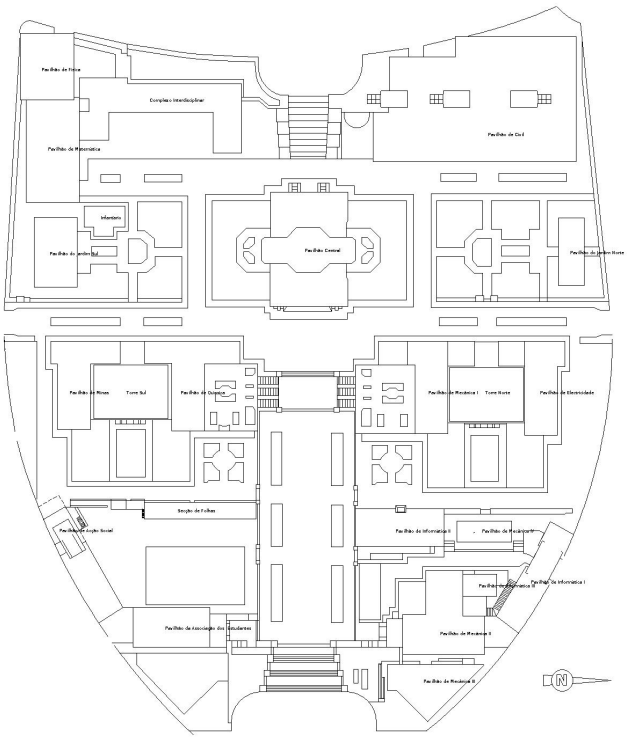
**Mestrado Integrado em Engenharia Eletrotécnica e de Computadores**

TÉCNICO LISBOA

# Arquitectura de Sistemas de Internet

2017/2018 - Primeiro semestre

**Projeto:** Aplicação web de gestão de salas no IST




| | | |
|---|---|---|
| **RODRIGO REIS** | número 78385 | rodrigobscreis@tecnico.ulisboa.pt |
| **JOÃO VIEIRA** | número 79191 | joaomiguelvieira@tecnico.ulisboa.pt |


10 de Março de 2019.

Docente: João Nuno de Oliveira e Silva

*[Página intencionalmente deixada em branco]*

# Introdução

No Técnico Lisboa, existem salas de estudo espalhadas pelos vários *campi*. Por vezes é necessário comunicar com os utilizadores destes espaços, por exemplo em caso de emergência. O projeto sob o qual incide este relatório consiste numa aplicação web para esse efeito. O sistema foi dimensionado para que os alunos que estejam a usar uma sala de estudo num determinado instante possam receber notificações de entidades de gestão e também ter conhecimento da identidade dos outros utilizadores que estão na mesma sala. Por outro lado, a aplicação oferece uma visão de administrador, onde é possível enviar mensagens aos utilizadores comuns e ver relatórios de ocupação em tempo real ou passado. Os utilizadores com permissões de administrador não têm acesso às mesmas funcionalidades que os utilizadores comuns, i. e., não se podem registar em salas nem receber mensagens.

O Técnico Lisboa compreende três *campi*, situados em localizações distintas: Alameda, Taguspark e Oeiras. Os espaços pertencentes a estes três *campi* estão documentados, e esta informação está disponível para utilização por aplicações externas num *endpoint* REST do sistema Fénix do Técnico Lisboa. É através deste *endpoint* que a aplicação obtém os dados necessários para efetuar a pesquisa das salas.

Externamente à aplicação, foi ainda usado o Técnico ID para efetuar a autenticação dos utilizadores comuns. O Técnico ID é baseado no protocolo de autenticação OAuth 2.0. Os administradores autenticam-se apenas localmente (não usam o Técnico ID).

No desenvolvimento deste sistema foram utilizadas diversas tecnologias, desde uma *framework* de desenvolvimento para PHP—Laravel 5.5—a bibliotecas de alto nível para manipulação de dados e realização de pedidos REST—Eloquent e Guzzle, respetivamente. A aplicação foi concebida para executar num servidor web, ter acesso a uma base de dados MySQL e ainda ao Técnico ID e ao sistema Fénix do Técnico Lisboa.

Na sua versão final, o sistema executa na *cloud* Microsoft Azure, num contentor para o efeito, utiliza uma base de dados para *cloud* autogerida igualmente hospedada no Microsoft Azure e compatível com MySQL. A aplicação pode ser visitada em asintproject.azurewebsites.net.

# Capítulo 1

# Desenvolvimento da aplicação

## 1.1 Estrutura global

A aplicação web sob a qual incide este relatório foi desenvolvida em PHP. Como se ilustra na figura 1.1, o sistema é composto essencialmente por quatro partes/serviços, nomeadamente um servidor web, um servidor de base de dados compatível com MySQL, um servidor de autenticação—que corresponde ao Técnico ID—, e um cliente, que pode ser um *browser* ou uma aplicação externa.

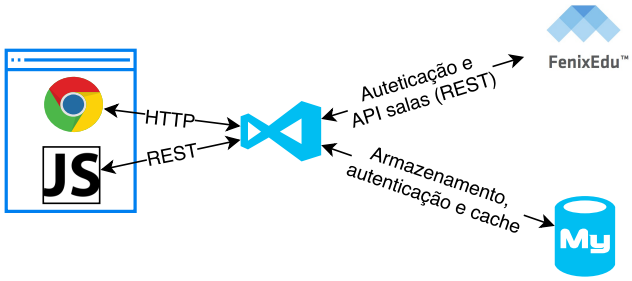



**Figura 1.1:** *Arquitetura genérica do sistema. A aplicação conta com quatro módulos principais: um servidor web, uma base de dados compatível com MySQL, um servidor de autenticação OAuth 2.0 que corresponde ao Técnico ID, e ainda clientes, que podem ser* browsers *ou aplicações externas.*

A aplicação executa num servidor web, Apache ou Nginx, e uma vez que foi desenvolvida em PHP, necessita ainda do PHP 7.0 ou superior para funcionar. Adicionalmente, e em parte devido à *framework* de desenvolvimento utilizada, são necessários alguns módulos de PHP: OpenSSL, PDO, Mbstring, Tokenizer e XML [1].

Dados persistentes e informações de sessão (por exemplo se o utilizador se encontra registado numa sala) são armazenadas numa base de dados compatível com MySQL. Esta base de dados pode ser local (correr localmente no servidor web) ou remota.

Os utilizadores comuns autenticam-se na aplicação através do Técnico ID, pelo que a aplicação está registada no sistema Fénix do Técnico Lisboa. Este recurso só é utilizado no ato de autenticação de um utilizador, sendo que o estado da sessão é depois controlado pela aplicação web sem utilizar os *tokens* obtidos do Técnico ID.

O sistema utiliza também a API REST do Fénix do Técnico Lisboa para obter informações acerca dos espaços contidos nos *campi*. Devido à latência associada aos acessos aos servidores do Fénix do Técnico Lisboa e ao elevado número de acessos necessários (devido ao número de espaços), o acesso aos servidores tem de ser explicitamente ordenado por um administrador. Quando um administrador ordena o acesso aos servidores do Fénix do Técnico Lisboa, a aplicação fica indisponível durante um largo período de tempo durante o qual a informação de todos os espaços é lida e armazenada na base de dados à qual a aplicação está vinculada. Desta forma, quando um utilizador realiza uma pesquisa, apenas a base de dados será utilizada e o resultado será imediato. Caso contrário teria de esperar um elevado período de tempo até que a pesquisa fosse concluída.

O sistema pode ser acedido através do *browser* (com pedidos HTTP a que a aplicação responde com páginas web HTML, ou com pedidos REST utilizando JavaScript) ou através de uma aplicação externa (utilizando a API REST implementada). No caso do sistema ser acedido por um *browser*, o controlo de acesso é efetuado com base nos privilégios do utilizador (que pode ser um utilizador comum, um administrador ou não estar autenticado). Já uma aplicação externa que utilize a API REST não está sujeita a controlo de acesso. A API foi concebida para ser independente de autenticação, pelo que, mesmo que não esteja autenticada, uma aplicação externa tem acesso a todos os *endpoints* da mesma.

## 1.2 Framework e bibliotecas usadas

O sistema foi desenvolvido utilizando uma *framework* de desenvolvimento para PHP. O Laravel 5.5 é uma das *framework* PHP mais comuns. O seu assistente de desenvolvimento—artisan—possibilita a criação de um servidor web de depuração, assim como modelos, controladores, *views*, ficheiros de migração e um sistema de autenticação básico.

A *framework* tem a arquitetura *Model-View-Controller* (MVC). Como se ilustra na figura 1.2, esta arquitetura divide a aplicação em três tipos de componentes e define as interações entre eles.

**Controlador:** Reage a estímulos externos do *browser* ou de uma aplicação e executa comandos sobre um modelo para atualizar o seu estado ou retorna uma *view*.

**Modelo:** Representa um objeto que pode ser uma tabela numa base de dados. É utilizado pelos controladores aquando do armazenamento ou consulta de dados. Pode também ser utilizado para controlar o contexto da aplicação.

***View*:** Uma *view* é modelo de apresentação de informação que é compilado por uma ferramenta—no caso do Laravel o Blade [2]—resultando numa página HTML que pode ser vista num *browser*.

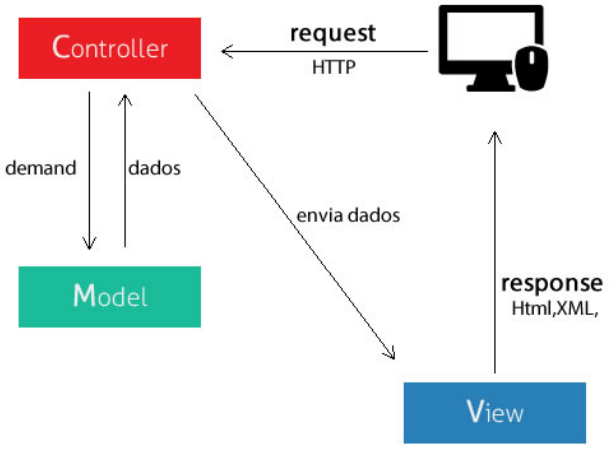



**Figura 1.2:** *Representação da arquitetura* Model-View-Controller *[3]. Os controladores são responsáveis pelo processamento dos dados e são, portanto, o* core *da aplicação. Uma* view *é um protótipo de apresentação de dados num formato legível e agradável para o utilizador. Um modelo é uma representação de um objeto que pode ser uma tabela numa base de dados.*

Tenha-se como exemplo o seguinte cenário de funcionamento de uma aplicação que segue a arquitetura MVC: o sistema recebe um pedido e consequentemente invoca uma função de um controlador. Aquando do tratamento dos dados, é necessário consultar e alterar informações de entidades conhecidas da aplicação, pelo que são utilizados os modelos que definem estas entidades. Cessado o processamento do pedido, o controlador envia os resultados para a ferramenta de compilação das *views* que produz um documento HTML para ser visto no *browser*.

Além dos modelos, das *views* e dos controladores, o Laravel 5.5 introduz o conceito de *middleware* [4]. Um *middleware*, como o próprio nome indica, é algo que é executado "no meio" da chamada ao controlador. Quando uma função de um controlador é invocada, através de uma rota especificada para o efeito, um *middleware* pode ser executado antes da chamada à função do controlador e depois de a mesma retornar. Tenha-se como exemplo o *middleware* de autenticação. Existem funcionalidades que devem estar disponíveis se e só se o utilizador tiver sessão iniciada. Por esse motivo, quando uma rota que chama uma função desse tipo é acionada, antes de a função ser executada é chamada a função *"handle"* do *middleware* de autenticação. Se o utilizador estiver autenticado, a função do controlador é executada, caso contrário a função não é executada e o *middleware* redireciona o utilizador para a página de *login*. Se as ações realizadas pela função do controlador forem suscetíveis de corromper o estado da aplicação, por exemplo, a integridade da mesma pode ser verificada na função *"terminate"* do *middleware* associado à rota que foi acionada.

### 1.2.1 Principais bibliotecas usadas

Em particular para a aplicação desenvolvida, um modelo corresponde a uma tabela numa base de dados. O Laravel 5.5 oferece uma camada de abstração para manipulação de bases de dados—Eloquent [5]—que torna transparente ao programador qual o tipo de base de dados de que se trata.

Como explicado na secção 1.1, a aplicação realiza pedidos REST ao sistema Fénix do Técnico Lisboa. Para esse efeito, foi utilizada uma biblioteca para PHP—Guzzle [6]. Com o Guzzle, é possível criar facilmente um cliente REST e efetuar pedidos a servidores externos. As respostas aos pedidos são depois descodificadas a partir da resposta em JSON e convertidas em *arrays* associativos do PHP.

### 1.2.2 Estrutura básica do Laravel 5.5

Ao instalar o Laravel 5.5 usando o Composer [7], um número elevado de diretórios e ficheiros são criados automaticamente. Os diretórios e ficheiros mais importantes do ponto de vista do programador são:

`app`: Diretório que contem os modelos e os controladores. O ficheiro `app/Http/Kernel.php` destaca-se pois contem o *kernel* da aplicação. Neste ficheiro são, por exemplo, declarados todos os *middlewares*.

`config`: Diretório que contém os principais ficheiros de configuração da aplicação. É nestes ficheiros que são especificadas as *drivers* das bases de dados, dos repositórios locais e remotos de ficheiros, dos serviços de e-mail e autenticação.

`database`: Diretório que contém os ficheiros relacionados com a estrutura da base de dados da aplicação, nomeadamente os ficheiros de migração dos modelos.

`public`: Diretório público da aplicação. Contem os ficheiros que são feitos públicos quando a aplicação é instalada na sua versão final. Todos os ficheiros deste diretório e sub-diretórios, por omissão não estão protegidos, pelo que todo o código da aplicação deve ser colocado nos sítios respetivos, caso contrário podem ser gerados problemas graves de segurança.

`resources`: Diretório que contem os ficheiros de estilo pré-compilados e *scripts* (Sass e JavaScript, respetivamente), ficheiros de linguagem e protótipos de páginas HTML (*views*).

`routes`: Diretório que contém os ficheiros de rotas da aplicação. Destaca-se o ficheiro `web.php` onde devem ser declaradas as novas rotas pelo programador.

`storage`: É neste diretório que constam os suportes de armazenamento locais e seguros que só podem ser acedidos explicitamente do lado do servidor pela biblioteca do Laravel 5.5 para o efeito. Não é possível para o cliente aceder aos ficheiros contidos neste diretório pelos métodos convencionais (*wget*, *curl*, entre outros).

`vendor`: Diretório onde constam as bibliotecas da *framework*. Este conteúdo não deve ser editado diretamente pelo programador. Todas as modificações aos módulos base que são descritos em ficheiros contidos neste diretório devem ser feitas por *override* dos respetivos métodos. Visto que é neste diretório que se encontram os ficheiros de raiz da *framework*, a edição forçada dos mesmos pode levar a que a aplicação deixe de funcionar.

`.env`: Ficheiro que contém as variáveis de ambiente da aplicação. Não deve ser feito público ou exibido, por exemplo, num repositório do GitHub. É neste ficheiro que constam todas as credenciais utilizadas pela aplicação.

`artisan`: Assistente de desenvolvimento do Laravel 5.5. A sua utilização possibilita a criação rápida de funcionalidades e protótipos de ficheiros de código fonte. Alguns comandos suportados pelo artisan são:

- `php artisan serve`: Lança um processo de depuração que simula um servidor web.

- `php artisan make:auth`: Cria *views*, controladores e modelos e respetivos ficheiros de migração necessários para um sistema de registo e autenticação básico que efetua o registo e autenticação dos utilizadores contra a base de dados à qual a aplicação está vinculada.
- `php artisan make:controller`: Cria um protótipo de controlador no diretório destinado ao efeito (`app/Http/Controllers`).
- `php artisan migrate`: Cria na base de dados à qual a aplicação está vinculada tabelas correspondentes aos ficheiros de migração.

`composer.json`: Ficheiro que contem a configuração necessária para a instalação da aplicação.

`server.php`: Ficheiro que contem a descrição do servidor de depuração que pode ser arrancado com o comando `php artisan serve`.

## 1.3 Autenticação

A autenticação na aplicação pode ser feita de duas formas distintas: os utilizadores comuns têm de ser alunos, docentes ou funcionários do Técnico Lisboa, e, por isso, utilizam o sistema de autenticação Técnico ID, baseado em OAuth 2.0. Já aos administradores não é exigido que sejam membros da comunidade académica do Técnico Lisboa, e a sua autenticação é local à aplicação.

### 1.3.1 Integração do sistema de autenticação do Técnico Lisboa

Para utilizar o sistema de autenticação Técnico ID, é necessário efetuar o registo prévio da aplicação no sistema Fénix do Técnico Lisboa [8]. O programador tem de especificar o endereço para o qual o Técnico ID deve retornar o código de autenticação e quais os *scopes* a que a aplicação terá acesso, i. e., a que informações acerca do utilizador a aplicação poderá aceder. Feito o registo, o sistema Fénix do Técnico Lisboa emite um ID de cliente e um segredo que deverão ser utilizados pela aplicação.

Tal como ilustrado na figura 1.3, a autenticação de um utilizador na aplicação usando o Técnico ID consiste num protocolo de três passos.

No primeiro passo, é gerado um endereço de autenticação do Técnico ID, utilizando o ID da aplicação e o endereço de retorno. O utilizador é redirecionado para este endereço e deve iniciar sessão com as suas credenciais do Técnico ID. Depois de a autenticação ser bem sucedida, o Técnico ID retorna à aplicação (utilizando o endereço de retorno) um código de autenticação. A aplicação confirma que a autenticação foi bem sucedida do lado do Técnico ID, verificando que o código de retorno é um HTTP 200 (OK) e faz um pedido REST ao Técnico ID para obter os *tokens* de autenticação e renovação. Neste pedido, a aplicação inclui: o seu ID, o segredo, o endereço de retorno, o código fornecido pelo Técnico ID no passo anterior e o tipo de autorização requerida. O Técnico ID analisa o pedido da aplicação, e se os parâmetros coincidirem retorna os *tokens* de sessão e de renovação que são guardados pela aplicação como variáveis de sessão. Concluído o segundo passo, passa a existir uma sessão ativa no Técnico ID. Por último, a aplicação efetua um pedido REST ao Técnico ID para obter as informações do utilizador, nomeadamente IST-ID, nome e e-mail; guarda estas informações na base de dados local (tabela de utilizadores); desativa a password para que este utilizador não se possa autenticar localmente através do sistema de autenticação nativo da *framework*; e inicia sessão.

### 1.3.2 Sistema de autenticação local

O Laravel 5.5 inclui um sistema de autenticação básico que autentica os utilizadores contra uma base de dados [9]. Um utilizador, como todos os conteúdos da aplicação, é representado por um modelo que por sua vez representa uma tabela na base de dados. Por omissão, um utilizador é caracterizado por um identificador, nome, e-mail, *hash* da sua palavra-passe, *remember token*, data de criação e data de modificação. Uma vez autenticado, o utilizador mantém sessão ativa até fazer explicitamente *logout* ou o *token* de sessão expirar. Este *token* de sessão é um *cookie* que é criado e gerido pela *framework*. O mecanismo de sessão utilizado pelo Laravel 5.5 é equivalente a utilizar *JSON Web Tokens* (JWTs) [10],

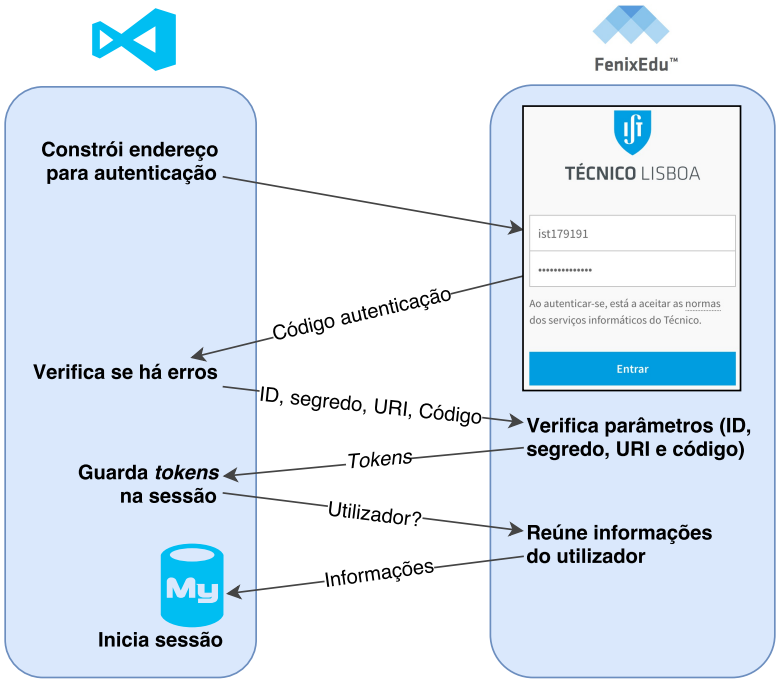



**Figura 1.3:** *Diagrama de comunicação da autenticação na aplicação web utilizando o Técnico ID. Em primeiro lugar é obtido um código para autenticação no Técnico ID. Com esse código, o ID da aplicação e o segredo, são obtidos os* tokens *de sessão e de renovação. Com sessão iniciada no Técnico ID, as informações do utilizador são salvas e uma sessão gerida pela aplicação é iniciada.*

ainda que, ao contrário do que acontece se a aplicação for desenvolvida de raiz em PHP, não seja necessário verificar explicitamente a validade do *token*, pois a *framework* encarrega-se disso.

Ao sistema de autenticação base do Laravel 5.5 foram introduzidas algumas modificações, nomeadamente foram acrescentados campos ao modelo de utilizador assim como foi alterado o método de *login* de forma a utilizar o nome de utilizador em vez do e-mail para iniciar sessão.

## 1.4 REST *endpoints*

O sistema foi preparado para poder ser utilizado por aplicações externas. Por esse motivo, foi implementada uma API REST que permite realizar as operações disponibilizadas pela interface web da aplicação. A tabela 1.1 sumariza os *endpoints* disponibilizados pela API da aplicação.

| Method | *Endpoint* |
|---|---|
| POST | `asintproject.azurewebsites.net/api/messages` |
| GET | `asintproject.azurewebsites.net/api/messages/room_id` |
| GET | `asintproject.azurewebsites.net/api/rooms/room_name` |
| GET | `asintproject.azurewebsites.net/api/rooms/room_id/info` |
| GET | `asintproject.azurewebsites.net/api/rooms/room_id/occupancy` |
| POST | `asintproject.azurewebsites.net/api/rooms/room_id/occupancy` |
| DELETE | `asintproject.azurewebsites.net/api/rooms/room_id/occupancy` |
| GET | `asintproject.azurewebsites.net/api/refresh` |
| GET | `asintproject.azurewebsites.net/api/log` |
| GET | `asintproject.azurewebsites.net/api/users/user_id` |

**Tabela 1.1:** *Lista de* endpoints *disponibilizados pela API da aplicação. A cinzento estão os parâmetros variáveis opcionais das rotas, e a azul estão os parâmetros variáveis mas obrigatórios.*

Em seguida, cada um dos *endpoints* é brevemente explicado.

**POST** `asintproject.azurewebsites.net/api/messages`

Permite enviar mensagens para os utilizadores registados numa determinada sala. Este *endpoint* espera receber um JSON da forma:

```
{
    room_id : ID da sala
    message : Mensagem que se pretende enviar
}
```

Verifica se ambos os campos são válidos e regista uma mensagem para os utilizadores desta sala.

**GET** `asintproject.azurewebsites.net/api/messages/room_id`

Retorna um JSON com as mensagens relativas à sala cujo ID é `room_id`. Caso o campo `room_id` seja omitido no endereço, todas as mensagens registadas são retornadas.

**GET** `asintproject.azurewebsites.net/api/rooms/room_name`

Procura uma sala com um nome semelhante a `room_name`. Caso o campo `room_name` seja omitido, um JSON com as informações de todas as salas é retornado.

**GET** `asintproject.azurewebsites.net/api/rooms/room_id/info`

Retorna um JSON com as informações relativas à sala cujo ID é igual a `room_id`.

**GET** `asintproject.azurewebsites.net/api/rooms/room_id/occupancy`

Retorna um JSON com todos os utilizadores registados na sala cujo o ID é igual a `room_id`.

**POST** `asintproject.azurewebsites.net/api/rooms/room_id/occupancy`

Regista um utilizador na sala cujo ID é igual a `room_id`. Este *endpoint* espera receber um JSON da forma:

```
{
    user_id : ID do utilizador a registar
}
```

Verifica se `user_id` corresponde a um utilizador válido e regista-o na sala correspondente a `room_id`.

**DELETE** `asintproject.azurewebsites.net/api/rooms/room_id/occupancy`

Retira um utilizador da sala cujo ID é igual a `room_id`. Este *endpoint* espera receber um JSON da forma:

```
{
    user_id : ID do utilizador a sair da sala
}
```

Verifica se `user_id` corresponde a um utilizador válido, verifica se o utilizador está registado nessa sala e retira-o.

**GET** `asintproject.azurewebsites.net/api/refresh`

Atualiza as informações dos espaços na base de dados da aplicação. Esta operação acede aos servidores do Fénix do Técnico Lisboa e reúne as informações de todos os espaços dos três *campi*.

**GET** `asintproject.azurewebsites.net/api/log`

Retorna um JSON contendo os *logs* de todos os *check-in's* e *check-out's* realizados desde a instalação da aplicação.

**GET** `asintproject.azurewebsites.net/api/users/user_id`

Retorna um JSON com a informação do utilizador cujo ID é igual a `user_id`. Caso o campo `user_id` seja omitido, um JSON com a informação de todos os utilizadores do sistema é retornado.

## 1.5 Manipulação de dados

Em termos de tratamento de dados, existem dois cenários frequentes: quando são manipulados dados internos da aplicação e quando são recebidos dados externos sob a forma de JSON (por exemplo quando a aplicação acede à API REST do Fénix do Técnico Lisboa para descarregar as informações acerca dos espaços [11]). No caso dos dados serem internos da aplicação, é utilizada a biblioteca Eloquent para lidar com a base de dados através dos modelos. Esta biblioteca, como já foi referido, oferece uma camada de abstração em relação aos pedidos diretos à base de dados, e portanto as tabelas são tratadas como objetos de PHP, de forma transparente ao tipo de base de dados que está a ser manipulada. Os modelos de dados utilizados pela aplicação correspondem a tabelas com os seguintes campos na base de dados:

`users`: Representa um utilizador no sistema.

- `id`: ID do utilizador gerado automaticamente pela base de dados no momento de inserção.
- `name`: Nome completo do utilizador.
- `username`: *Nickname* do utilizador. No caso dos administradores, é também o nome de início de sessão.
- `istid`: Identificador de membro da comunidade do Técnico Lisboa. Caso se trate de um administrador, este campo é nulo.
- `email`: E-mail do utilizador.
- `password`: *Hash* da palavra-passe do utilizador.
- `room_id`: Sala em que o utilizador está registado. Caso seja administrador ou não esteja registado em nenhuma sala, este campo é nulo.
- `remember_token`: *Remember token*. Quando definido, a sessão só espira quando o utilizador fizer explicitamente *logout*.
- `created_at`: Data de criação do utilizador.
- `updated_at`: Data de modificação do utilizador.

`rooms`: Representa um espaço no sistema.

- `room_id`: Identificador de espaço importado do sistema Fénix do Técnico Lisboa.
- `name`: Designação do espaço.
- `parent_id`: ID do espaço onde este está contido. Caso se trate de um *campus*, este campo é nulo.
- `created_at`: Data de criação do espaço.
- `updated_at`: Data de modificação do espaço.

`messages`: Representa uma mensagem para um utilizador comum no sistema.

- `id`: ID da mensagem gerado automaticamente pela base de dados no momento de inserção.
- `room_id`: ID da sala a que se destina a mensagem.
- `message`: Corpo da mensagem
- `created_at`: Data de criação da mensagem.
- `updated_at`: Data de modificação da mensagem.

`check_events`: Representa um evento de *check-in* ou *check-out* no sistema.

- `id`: Identificador de evento gerado automaticamente pela base de dados no momento de inserção.
- `action`: *check-in* ou *check-out*.
- `room_id`: Sala a que o evento diz respeito.

`user_id`: Utilizador a que o evento diz respeito.

`created_at`: Data de criação do evento.

`updated_at`: Data de modificação do evento.

Outro cenário em que é necessário lidar com uma grande quantidade de dados é no varrimento dos espaços do *campi* do Técnico Lisboa utilizando a API disponibilizada para o efeito. A API está organizada em árvore, sendo que, na sua raiz, são apresentados os três *campi* do Técnico Lisboa [11]. Para cada um dos *campi*, existe um *endpoint* que contem as informações uma lista dos seus sub-espaços. Para cada sub-espaço, existe também um *endpoint* que contem informações e uma lista de sub-espaços contidos neste. O *endpoint* de cada espaço segue o formato:

`fenix.tecnico.ulisboa.pt/api/fenix/v1/spaces/id-do-espaço`.

Para efetuar o varrimento das salas do Técnico Lisboa e carregar as informações necessárias para a base de dados, a aplicação utiliza uma função recursiva que guarda as informações do espaço e para cada sub-espaço chama-se a si mesma. De notar que na base de dados apenas são armazenados os IDs dos espaços, os IDs dos espaços que os contém e a designação dos mesmos, para efeitos de procura e apresentação hierárquica dos resultados (i. e., apresentar os espaços correspondentes à procura não apenas indicando o nome mas também a sua hierarquia até ao *campus*). As restantes informações acerca dos espaços, quando são necessárias, são extraídas diretamente da API REST do Fénix do Técnico Lisboa (tipo de espaço, lista de sub-espaços e planta).

## 1.6 Interface de utilizador

A base da interface de utilizador web é a original do Laravel 5.5, baseada em Bootstrap [12] com algumas modificações. O Bootstrap é uma *framework* de ficheiros de estilo que permite ao programador conceber páginas web complexas utilizando classes e funções JavaScript pré-feitas. Nesta sequência, a *framework* foi utilizada para desenhar os restantes esqueletos das páginas que constituem a interface de utilizador. De notar que desta forma foi possível desenhar uma interface responsiva [13] que se adapta ao dispositivo onde a página web é visualizada.

Dependendo do tipo de utilizador (utilizador comum ou administrador) as *views* disponíveis são diferentes. A figura 1.4 ilustra as diferentes *views* organizadas tendo em conta o fluxo da aplicação.

A página de apresentação da aplicação e a de *login* são as mesmas para o utilizador comum e para o administrador. Enquanto o utilizador comum, na página de *login*, escolhe a opção "Técnico Lisboa", que o redireciona para a página de autenticação do Técnico ID, o administrador inicia sessão utilizando o formulário de *login* embutido na página.

### 1.6.1 Utilizadores comuns

Depois de autenticado, o utilizador comum é redirecionado para a página de início que contém um formulário para pesquisar salas pelo nome. Ao efetuar efetuar uma pesquisa, uma lista de salas é apresentada. Efetuado o *check-in* numa sala, uma página com a lista de membros da comunidade académica na mesma sala e todas as mensagens enviadas até ao momento são apresentadas. Selecionando o nome no cabeçalho, é ainda possível ver informação detalhada sobre o espaço e navegar na hierarquia clicando no nome de qualquer super-espaço ou sub-espaço do atual.

### 1.6.2 Administradores

Após iniciar sessão, o administrador é redirecionado para a sua página principal que contém uma explicação das funcionalidades disponíveis. A partir do menu, são acessíveis três páginas administrativas: uma para enviar mensagens para salas onde estejam registados utilizadores, outra que lista todos os utilizadores registados em salas e qual a sala, e outra que contém os *logs* de *check-in's* e *check-out's* desde que a aplicação foi instalada.

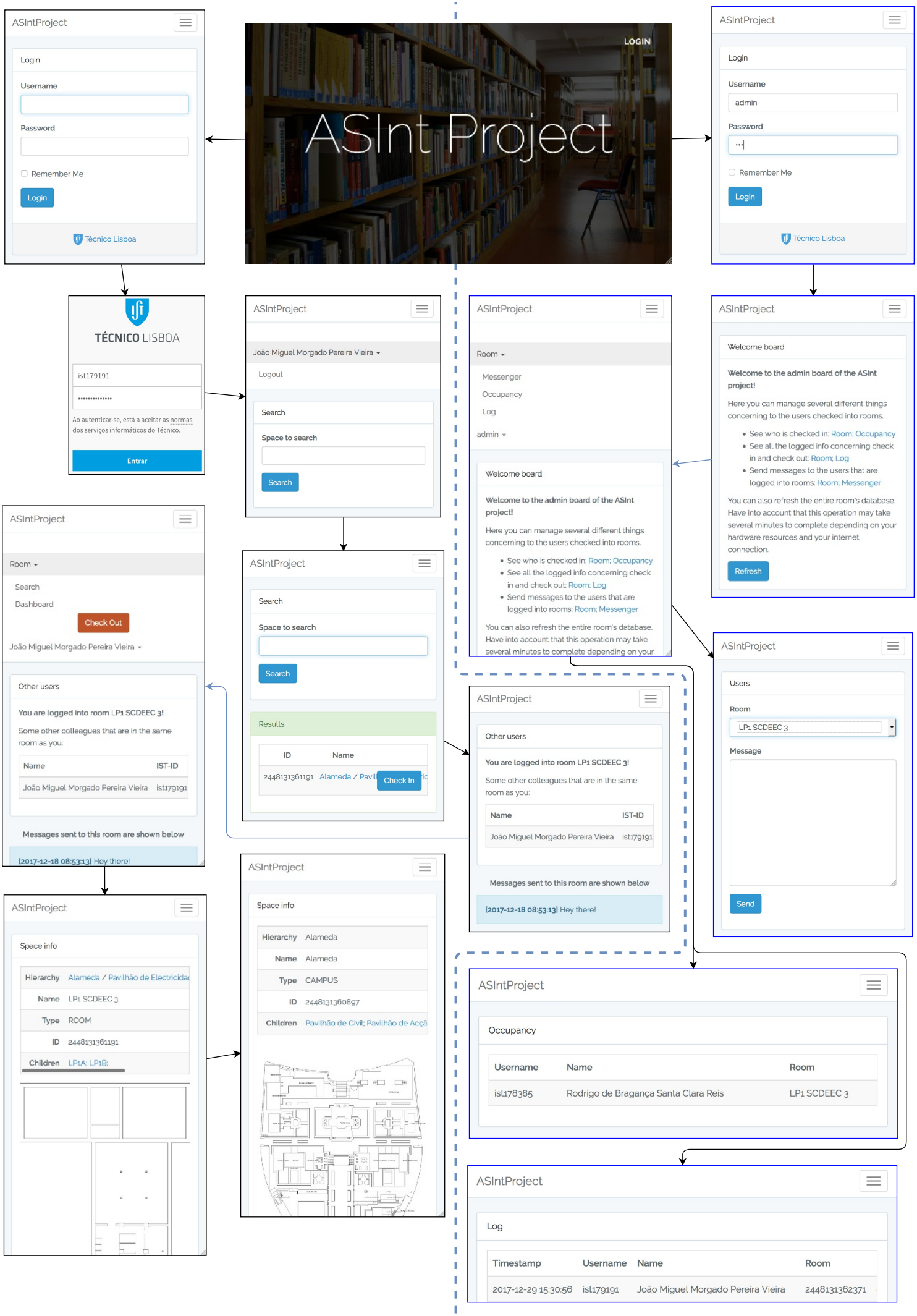



**Figura 1.4:** *Diagrama de fluxo da aplicação. Do lado esquerdo ilustram-se as* views *dos utilizadores comuns, e do lado direito as dos administradores.*

# Capítulo 2

# Implementação na *cloud*

## 2.1 Plataforma Microsoft Azure

A versão final da aplicação foi implementada na *cloud*, nomeadamente no Microsoft Azure [14]. O Microsoft Azure é uma plataforma de computação na nuvem, onde é possível criar e gerir uma diversidade de recursos, incluindo bases de dados e aplicações web. A decisão de hospedar a aplicação no Microsoft Azure prendeu-se com vários motivos, entre os quais:

- O serviço Web App do Microsoft Azure suporta o Laravel 5.5, pois é possível configurar o servidor web manualmente.
- A uma aplicação web hospedada no Azure é associado um endereço sugestivo, neste caso

  asintproject.azurewebsites.net.
- O Microsoft Azure oferece um mês de teste e um *plafond* de 170€, sendo que após um dos dois expirar, o serviço passa a ter um custo de, aproximadamente, 30€/mês.
- O sistema de ficheiros permite escrita, o que significa que a opção de *log* do Laravel 5.5 pode estar ativa e, caso fosse necessário, também poderiam ser utilizados discos locais para armazenar dados.
- É disponibilizada uma consola PowerShell e outras ferramentas de depuração sob o nome de Kudu [15] facilmente acessíveis em

  asintproject.scm.azurewebsites.net

  protegidas por autenticação do administrador da instância. Através da consola é possível efetuar modificações rápidas e com efeito imediato na aplicação.
- É oferecida a possibilidade de ter *continuous deployment*. A aplicação pode estar vinculada a um repositório Git; o Azure deteta quando é feito um *push* e implementa automaticamente a nova versão da aplicação, sem intervenção do programador.
- Existe documentação detalhada sobre como migrar uma aplicação baseada em Laravel 5.5 para o Microsoft Azure [16].
- É possível ter bases de dados auto-geridas compatíveis com MySQL no Microsoft Azure.

No total, foram utilizados dois serviços da plataforma Microsoft Azure: uma instância do serviço Web App para hospedar a aplicação web propriamente dita e uma instância MySQL.

## 2.2 Serviço Web App

O serviço Web App do Microsoft Azure [17] implementa um servidor web e, opcionalmente, uma base de dados local, não escalável. Suporta diversas linguagens de desenvolvimento web, como o PHP, o Python e o Ruby, e oferece suporte direto a *Content Management Systems* (CMSs) como o WordPress e o Drupal.

O serviço Web App permite ainda escolher a plataforma base onde o servidor web é executado (Windows ou Linux), escala automaticamente as instâncias da aplicação consoante os requisitos, faz

balanceamento de carga e é um serviço de alta disponibilidade. Opcionalmente, as aplicações web podem executar em contentores de Docker—esta opção não foi selecionada na implementação desta aplicação.

Como referido na secção 2.1, o serviço Web App oferece *continuous deployment* com o Git. Além desta característica, o Azure faz gestão de versões, sendo possível por a executar qualquer versão anterior num dado momento.

O escalamento de instâncias da aplicação pode ser feito manualmente ou automaticamente seguindo regras estabelecidas por quem gere o serviço. É possível ainda especificar as localizações geográficas dos *datacenters* onde se pretende que as instâncias executem, de forma a mitigar latências de acesso elevadas.

Adicionalmente, a plataforma gera estatísticas de utilização de recursos e relatórios de erros. A figura 2.1 ilustra a página de gestão de uma Web App do Microsoft Azure.

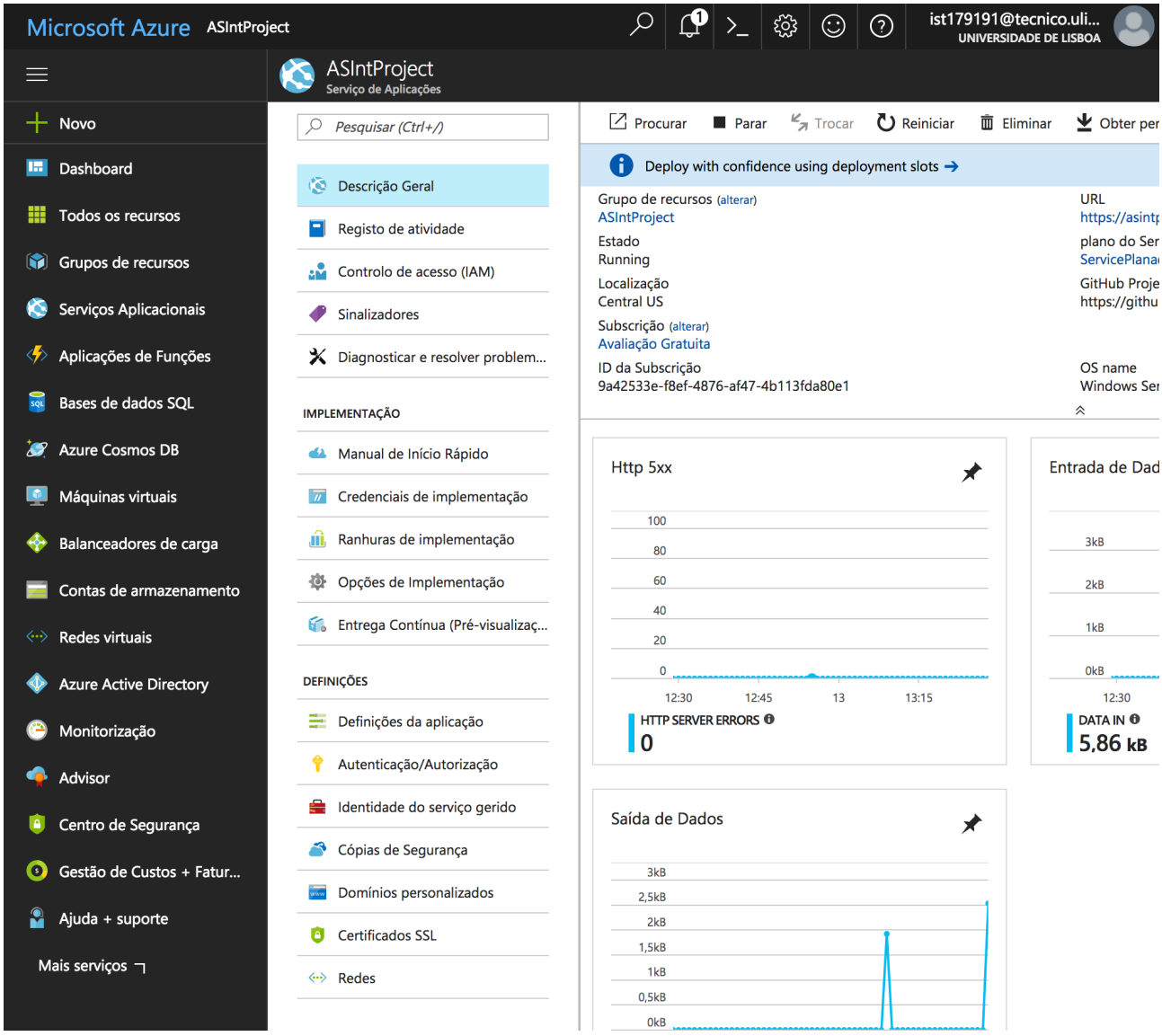



**Figura 2.1:** Dashboard *de gestão de uma Web App do Microsoft Azure. No menu à esquerda são visíveis vários parâmetros configuráveis da instância.*

Para esta aplicação, contudo, o serviço Web App não permite a execução de uma das funcionalidades: a população inicial da base de dados com os espaços do Técnico Lisboa. A função responsável pela população da base de dados é recursiva e, como já foi dito, devido ao elevado número de salas (e também à latência de acesso aos servidores do Fénix do Técnico Lisboa) este processo é demorado. A versão final da aplicação hospedada num *datacenter* nos EUA vinculada a uma base de dados também hospedada nos EUA demora cerca de duas horas para varrer toda a base de dados dos espaços dos ser-

vidores do Fénix do Técnico Lisboa. Acontece que o Azure não permite um *front-end timeout* superior a quatro minutos [18], e por esse motivo não é possível realizar o *caching* das informações relativa às salas através da aplicação final, já que a aplicação não responde a estímulos externos enquanto realiza esta função. Esta operação tem de ser efetuada previamente por uma entidade externa (por exemplo uma instância da aplicação a executar num servidor web que não imponha restrições de tempo de execução).

## 2.3 Serviço MySQL

O Microsoft Azure oferece ainda um serviço de base de dados compatível com MySQL [19–21]. Este serviço representa o recurso pai da base de dados na *cloud*. Semelhante ao serviço Web App, é possível escolher a região dos recursos físicos responsáveis pela hospedagem do serviço de uma lista de recursos distribuídos pelos *datacenters* da Microsoft.

Os recursos associados à instância MySQL são escaláveis, o que significa que o redimensionamento da uma instância não implica que a mesma fique *offline* durante um largo período de tempo. O serviço oferece ainda *backups* automáticos, *database patching*, monitorização, mecanismos de segurança de dados e um modelo de alta disponibilidade. Apesar de todos os recursos, a administração necessária à manutenção destas funcionalidades é mínima, daí se dizer que o serviço é auto-gerido.

De momento, o serviço MySQL do Microsoft Azure está em fase de *public preview*, pelo que ainda tem limitações [22].

A figura 2.2 ilustra conceptualmente o sistema MySQL do Microsoft Azure.

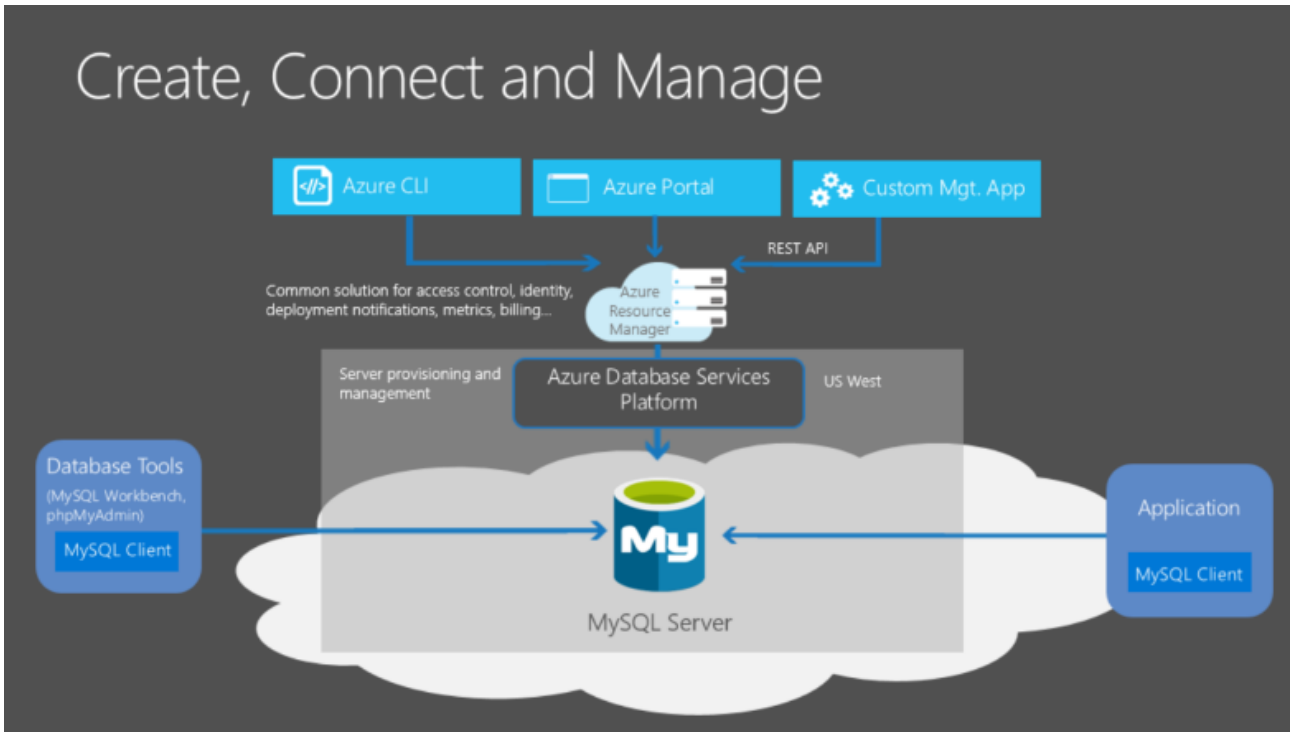



**Figura 2.2:** *Base de dados Azure para MySQL: Diagrama conceptual [20].*

### 2.3.1 *Compute Unit*

Uma *Compute Unit* (CU) ou unidade de computação é uma medida de desempenho de uma base de dados para MySQL do Microsoft Azure [23]. É uma medida abstrata, o que significa que não tem uma equivalência unívoca para recursos físicos. No entanto, no geral, 100 CUs equivalem aproximadamente a um *core* físico.

Uma CU não representa apenas recursos de CPU mas também recursos de memória. É na memória que jaz a grande diferença no significado de CUs de diferentes modelos de implementação. Por exemplo,

um serviço com 800 CUs oferece oito vezes a performance de CPU de um serviço do mesmo modelo apenas com 100 CUs. No entanto, um serviço básico (ver secção 2.3.2) com 100 CUs tem apenas metade da memória de um serviço padrão com o mesmo número de CUs. Neste sentido, apesar de terem o mesmo poder de computação, a performance do serviço básico é inferior à do serviço padrão.

### 2.3.2 Modelos de implementação

A base de dados Azure para MySQL oferece três modalidades de implementação/pagamento: Básico, Padrão e *Premium* [24]. A última modalidade, devido ao serviço se tratar de um *public preview*, ainda não está disponível. Estas modalidades diferem essencialmente em cinco medidas de capacidade/desempenho: número máximo de *Compute Units* (CUs), capacidade total de armazenamento, *Input/output Operations Per Second* (IOPS) garantidas, número máximo de IOPS e período de retenção de *backups* da base de dados. Em [24] é aconselhado um mínimo de 200 a 400 CUs para aplicações que requerem uma disponibilidade razoável. No caso da aplicação sobre a qual incide este relatório, estes parâmetros foram ajustados para o mínimo (50 CUs e 50GB de armazenamento).

Para melhor avaliar a necessidade de recursos para uma instância do serviço, a plataforma Azure fornece estatísticas e métricas de avaliação. Se a uma dada altura os recursos alocados não satisfizerem os requerimentos da aplicação, estes podem ser escalados para satisfazer os novos requisitos.

Se os recursos disponíveis forem utilizados ao máximo, a latência dos pedidos à base de dados podem aumentar, no entanto é pouco provável que aconteçam erros a não ser que a instância não seja capaz de satisfazer os pedidos antes do *timeout*.

### 2.3.3 Modelo de alta disponibilidade

Os serviços fornecidos pelo Microsoft Azure encontram-se sob um *Service Level Agreement* (SLA) de 99.99% de disponibilidade [25]. O modelo adjacente de alta disponibilidade é baseado em mecanismos de redundância e recuperação que são acionados quando ocorre um problema ao nível de um nó na *cloud*. Estas situações podem ser devidas a falhas de *hardware* ou falhas na implementação do serviço.

As alterações de estado nas bases de dados Azure para MySQL são feitas na forma de transações. Se um problema ocorrer enquanto uma transação está em curso, o servidor de base de dados cria um novo nó e migra para lá os dados. As conexões ativas no momento da falha são terminadas e as transações em curso são canceladas. No entanto as futuras conexões da aplicação à base de dados são redirecionada de forma transparente para a nova instância criada. Internamente na plataforma Azure, este redirecionamento é feito por uma *gateway* e todo este processo de recuperação é tipicamente resolvido em alguns segundos.

### 2.3.4 Escalamento de recursos

A qualquer momento, se os requisitos da aplicação mudarem, os recursos associados ao serviço de base de dados Azure para MySQL podem ser ajustados. Este ajustamento é transparente às aplicações que utilizam este recurso, sendo que a única diferença que se faz sentir é que durante o redimensionamento do serviço este fica indisponível. De forma análoga ao que acontece em caso de falha, no redimensionamento do serviço uma nova instância é criada com os recursos especificados. Os dados existentes na instância antiga são transferidos para a nova instância e, por último, a instância antiga é eliminada.

### 2.3.5 Mecanismos de segurança

Este serviço oferece mecanismos de segurança e proteção de dados nativos que limitam o acesso, protegem os dados armazenados e ajudam a monitorizar a atividade respeitante a transações da base de dados. Todos os dados armazenados em disco encontram-se cifrados com a cifra AES 256-bit. Este mecanismo é nativo e não pode ser desativado.

Por pré-definição, a base de dados só aceita conexões seguras que utilizam certificados SSL, e a *firewall* está configurada para recusar todas as conexões exceto as que provêm de IPs especificados [26]. Para facilitar a implementação da aplicação web, a necessidade de certificado SSL foi desativada e foi

adicionada uma regra à *firewall* para aceitar conexões provenientes de todos os IPs. A figura 2.3 ilustra a configuração da *firewall* da instância MySQL à qual a aplicação foi vinculada.

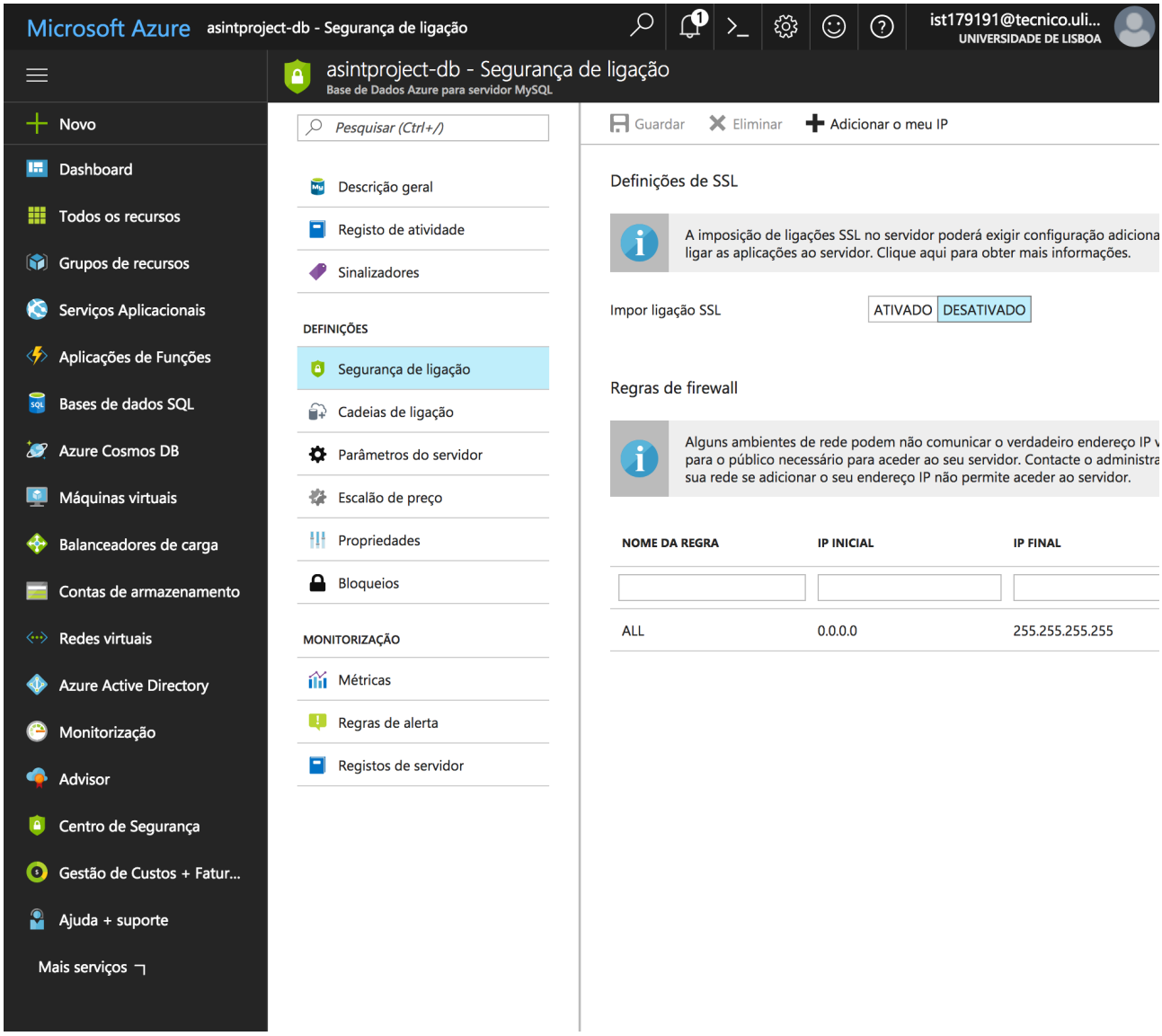



**Figura 2.3:** Dashboard *de gestão da instância MySQL à qual a aplicação web foi vinculada evidenciando as regras de* firewall. *No menu à esquerda são visíveis vários parâmetros configuráveis da instância.*

### 2.3.6 Vinculação com a aplicação

O suporte a bases de dados relacionais na *cloud* é uma tecnologia recente, sendo que devido à natureza dos serviços na *cloud*, as operações em bases de dados relacionais são tipicamente ineficientes. Isso acontece porque os dados estão armazenados em memória física distribuída que podem estar em *datacenters* separados por milhares de quilómetros de distância. Além disso a complexidade associada a uma base de dados relacional é desnecessária para a maioria das aplicações web. Por outro lado, o Laravel 5.5 só disponibiliza *drivers* para quatro tipos de bases de dados: MySQL, PostgreSQL, SQL Lite e SQL Server [27]. Para suportar outros tipos de bases de dados, nomeadamente tipos otimizados para *cloud*, é necessária a instalação de drivers *third-party*. Caso não fosse possível alojar uma base de dados compatível com MySQL na *cloud* seria necessário recorrer a uma alternativa deste tipo ou manipular manualmente os dados sem recorrer a *drivers* de alto nível e modelos.

# Conclusão

No desenvolvimento desta aplicação foram utilizadas bibliotecas e *frameworks* de programação com um alto nível de abstração que possibilitam reduzir o tempo de produção. Adicionalmente, e mais importante, estas ferramentas consideram problemas de estrutura que podem levar a falhas de segurança, pelo que a sua utilização conduz a aplicações mais seguras, modulares e robustas.

A aplicação foi estruturada para conciliar dois métodos de autenticação distintos, sendo que os utilizadores autenticados são representados pelo mesmo modelo. Neste sentido, o método de autenticação externo utilizado (Técnico ID) encaixa no sistema de autenticação nativo da *framework*, e apenas a forma como as credenciais do utilizador são verificadas é descrita.

A aplicação resultante disponibiliza ainda um conjunto de *endpoints* REST cujo objetivo é munir uma aplicação externa das ferramentas necessárias para efetuar todas as operações disponibilizadas pela interface web sem ter de recorrer à mesma.

Internamente à aplicação, os dados são tratados recorrendo a modelos, já que o Laravel 5.5 segue a arquitetura MVC e o acesso à base de dados de espaços do Fénix do Técnico Lisboa através da API REST é necessário para obter informações necessárias ao funcionamento do sistema.

A implementação da aplicação na *cloud* foi um processo simples devido à existência de documentação detalhada para o efeito. As vantagens da aplicação correr no Microsoft Azure são inúmeras, desde a alta disponibilidade do serviço à facilidade com que os recursos alocados podem ser escalados. O ambiente de execução na *cloud* permite uma configuração fina da infraestrutura (por exemplo, possibilita a escolha da versão do PHP) que pode ser dimensionada ao detalhe para respeitar os requisitos de um dado sistema.

Foi encontrado apenas um percalço na implementação da aplicação na *cloud*, que desabilita o funcionamento do *caching* da informação dos espaços na base de dados. Isto acontece porque esta função demora cerca de duas horas a concluir e o Microsoft Azure estabelece um *front-end timeout* de quatro minutos que não é ajustável. Este processo tem de ser efetuado previamente à instalação na *cloud* por, por exemplo, uma instância da aplicação a correr num servidor web que não imponha estas restrições. Fosse a aplicação um sistema real e não um exercício e teria de ser reestruturada para mitigar este problema.

As vantagens da *cloud* são notórias, e os preços praticados *versus* a auto-gestão dos recursos tornam esta solução apetecível para empresas e *startups* que não querem/não podem suportar os custos associados a manterem servidores próprios ou cuja gestão seja responsabilidade própria.

# Bibliografia


[1] Laravel installation. https://laravel.com/docs/5.5/installation. Acedido: 2017-12-30.

[2] Blade templates. https://laravel.com/docs/5.5/blade. Acedido: 2017-12-31.

[3] Laravel introdução: Mvc. https://raw.githubusercontent.com/diegoeis/tableless-static-images/master/2015/02/laravel-introducao.jpg. Acedido: 2017-12-31.

[4] Middleware. https://laravel.com/docs/5.5/middleware. Acedido: 2017-12-31.

[5] Eloquent: Getting started. https://laravel.com/docs/5.5/eloquent. Acedido: 2017-12-30.

[6] Guzzle documentation. http://docs.guzzlephp.org/en/stable/. Acedido: 2017-12-30.

[7] Composer: Dependency manager for php. https://getcomposer.org/. Acedido: 2017-12-30.

[8] Use fenixedu™ api in your application. http://fenixedu.org/dev/tutorials/use-fenixedu-api-in-your-application/. Acedido: 2017-12-30.

[9] Laravel authentication. https://laravel.com/docs/5.5/authentication. Acedido: 2017-12-30.

[10] Introduction to json web tokens. https://jwt.io/introduction/. Acedido: 2017-12-30.

[11] Api endpoints. http://fenixedu.org/dev/api/. Acedido: 2017-12-31.

[12] Introduction to bootstrap. https://getbootstrap.com/docs/4.0/getting-started/introduction/. Acedido: 2017-12-30.

[13] Interface responsiva. https://responsividade.wordpress.com/2015/06/01/interface-responsiva/. Acedido: 2017-12-31.

[14] Microsoft azure. https://azure.microsoft.com/pt-pt/. Acedido: 2017-12-31.

[15] What is kudu? - azure web sites deployment with david ebbo. https://azure.microsoft.com/pt-pt/resources/videos/what-is-kudu-with-david-ebbo/. Acedido: 2017-12-31.

[16] Hosting a laravel application on azure web app. https://medium.com/@coderonfleek/hosting-a-laravel-application-on-azure-web-app-b55e12514c46. Acedido: 2017-12-31.

[17] Microsoft azure: Web apps. https://azure.microsoft.com/en-us/services/app-service/web/. Acedido: 2017-12-31.

[18] Azure front-end timeout. http://www.coderanger.com/post/146223478109/azure-front-end-timeout. Acedido: 2017-12-31.

[19] Microsoft azure: Mysql. https://azure.microsoft.com/en-us/services/mysql/. Acedido: 2017-12-31.

[20] Microsoft azure: Mysql overview. https://docs.microsoft.com/en-us/azure/mysql/overview. Acedido: 2017-12-31.

[21] Microsoft azure: Servers. https://docs.microsoft.com/en-us/azure/mysql/concepts-servers. Acedido: 2017-12-31.

[22] Microsoft azure: Mysql limits. https://docs.microsoft.com/en-us/azure/mysql/concepts-limits. Acedido: 2017-12-31.

[23] Microsoft azure: Mysql compute unit and storage. https://docs.microsoft.com/en-us/azure/mysql/concepts-compute-unit-and-storage. Acedido: 2017-12-31.

[24] Microsoft azure: Mysql service tiers. https://docs.microsoft.com/en-us/azure/mysql/concepts-service-tiers. Acedido: 2017-12-31.

[25] Microsoft azure: Mysql high availability. https://docs.microsoft.com/en-us/azure/mysql/concepts-high-availability. Acedido: 2017-12-31.

[26] Microsoft azure: Firewall rules. https://docs.microsoft.com/en-us/azure/mysql/concepts-firewall-rules. Acedido: 2017-12-31.

[27] Database: Getting started. https://laravel.com/docs/5.5/database. Acedido: 2017-12-31.