Engenharia de Serviços

Projeto 2

1. Introdução

Este trabalho teve como objetivo desenvolver uma aplicação web na cloud para gerir playlists de músicas de múltiplos utilizadores. Uma playlist contém a identificação de um conjunto de músicas a serem reproduzidas e a indicação da ordem em que esta reprodução deve ocorrer. Esta aplicação está dividida em três camadas distintas: uma camada de dados que armazena e manipula informações sobre utilizadores, músicas e playlists, uma camada de negócio que fornece uma interface REST para o mundo exterior e uma camada de apresentação para os browsers web desenvolvida em React. Para o desenvolvimento da aplicação utilizámos as seguintes tecnologias: a base de dados MySQL como sistema de gestão de base de dados, o SQLAlchemy como mapeador relacional de objetos que permite associar classes com tabelas de bases de dados e objetos dessas classes com registos nas tabelas correspondentes, o Flask para fornecer os serviços web e JavaScript com React para a interface web.
Este relatório está dividido em três partes, em que cada uma das partes descreve as decisões de projeto tomadas para cada uma das três camadas. Na final do relatório descrevemos as experiências de escalabilidade efetuadas.

2. Camada de dados (SQLAlchemy + CRUD)

A camada de dados está dividida em duas partes distintas, as classes referentes ao modelo de dados da aplicação e a camada CRUD totalmente demarcada que manipula a informação.

2.1 Modelo de dados

O modelo de dados do projeto é constituído por três entidades: Utilizador, Musica e Playlist. Para cada uma destas entidades (ou classes) foram definidos vários atributos e identificadas as relações necessárias para responder às necessidades do problema apresentado. De seguida são referidos os atributos e as relações definidas para cada uma das entidades da camada de dados.

<u>Utilizador</u>

A entidade Utilizador é constituída pelos seguintes atributos: id (gerado automaticamente), nome, email, password, telefone, uma lista de músicas e uma lista de playlists.
Quanto às relações, foram aplicadas relações OneToMany relativamente à classe Musica uma vez que um utilizador pode ter várias músicas, e relativamente à classe Playlist uma vez que um utilizador pode ter várias playlists.
As passwords dos utilizadores guardadas na base de dados são um hash MD5 não invertível da password utilizada no registo. Deste modo caso a base de dados seja comprometida a informação relativa aos utilizadores do sistema não é comprometida.

<u>Musica</u>

A entidade Musica é constituída pelos seguintes atributos: id (gerado automaticamente), título, artista, álbum, ano lançamento, path ficheiro, um utilizador e uma lista de playlists.

Quanto às relações, foi aplicada uma relação ManyToOne relativamente à classe Utilizador, uma vez que cada música tem apenas um utilizador, mas cada utilizador pode ter várias músicas. Uma relação ManyToMany relativamente à classe Playlist, uma vez que uma música pode pertencer a várias playlists e uma playlist pode ter várias músicas.

Playlist

A entidade Playlist é constituída pelos seguintes atributos: id (gerado automaticamente), nome, data criação, um utilizador e uma lista de músicas. Quanto às relações, foi aplicada uma relação ManyToOne relativamente à classe Utilizador, uma vez que cada playlist tem apenas um utilizador, mas cada utilizador pode ter várias playlists. Uma relação ManyToMany relativamente à classe Musica, uma vez que uma playlist pode ter várias músicas e uma música pode pertencer a várias playlists.

Na figura abaixo, encontra-se um esquema sumário das entidades e das suas respetivas relações bidirecionais.

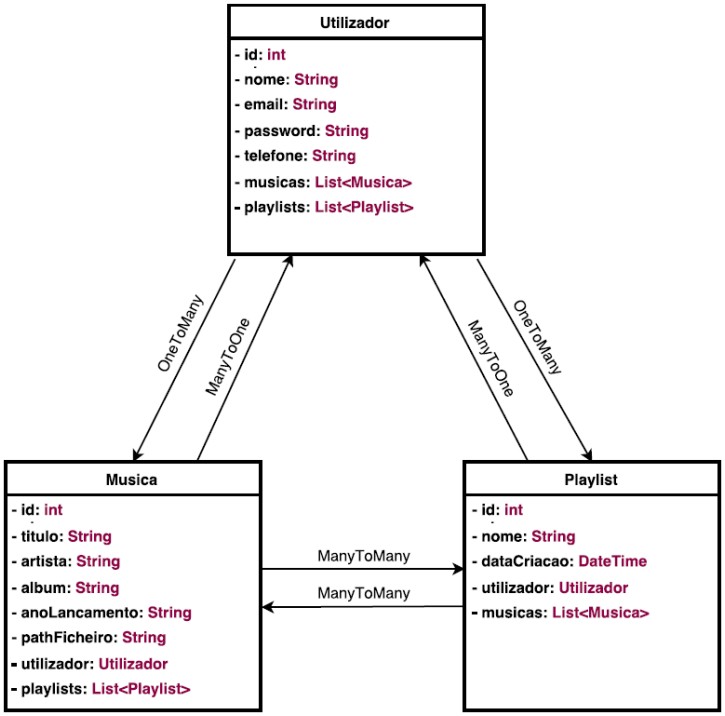


Figura 1 - Diagrama de classes da base de dados

De referir que para fazer a relação ManyToMany entre a classe Musica e Playlist foi necessário criar uma tabela relacional adicional que optámos por omitir no diagrama.

2.2 CRUD

O CRUD (Create, Read, Update e Delete) é uma camada totalmente demarcada que contém o código que responde ao objetivo do pedido, ou seja, contém métodos para armazenar e manipular os dados relativos aos utilizadores, às músicas e às playlists. De forma a evitar o SQL Injection todos os parâmetros foram introduzidos nas queries através das funções do SQlAlchemy como o filter_by e o get. Deste modo asseguramos a segurança da camada de dados ao não permitir a introdução direta de parâmetros nas queries. De seguida apresentamos uma listagem dos métodos definidos no ficheiro do CRUD:

Métodos gerais necessários ao funcionamento do sistema
String hashMD5(password), MusicaDTO[] listarMusicasSistema(), MusicaDTO[] pesquisar MusicaSistema(titulo, artista), UtilizadorDTO converterUtilizadorDTO(utilizador), MusicaDTO converterMusicaDTO(musica), PlaylistDTO converterPlaylistDTO(playlist)
Métodos para lidar com as operações relativas aos utilizadores
int registarConta(nome, email, password, telefone), boolean editarConta(idUtilizador, nome, email, password, telefone), boolean removerConta(idUtilizador), int login(email, password), UtilizadorDTO obterInfoUtilizador(idUtilizador)
Métodos para lidar com as operações relativas às músicas
boolean inserirMusica(titulo, artista, album, anoLancamento, pathFicheiro, idUtilizador), boolean editarMusica(idMusica, titulo, artista, album, anoLancamento), boolean removerMusica(idMusica), MusicaDTO[] listarMusicas(idUtilizador)
Métodos para lidar com as operações relativas às playlists
boolean criarPlaylist(idUtilizador, nome), boolean editarPlaylist(idPlaylist, nome), boolean removerPlaylist(idUtilizador, idPlaylist), boolean adicionarMusicaPlaylist(idPlaylist, idMusica), boolean removerMusicaPlaylist(idPlaylist, idMusica), PlaylistDTO[] listarPlaylists(idUtilizador, tipo), PlaylistDTO[] listarMusicasPlaylist(idPlaylist)

3. Camada de negócio (REST)

A camada de negócio faz a ponte entre a camada de dados e a camada de apresentação, interagindo com o CRUD e fornecendo uma interface REST para o mundo exterior. Os recursos fornecidos pelo REST têm identificadores globais (URIs) que são acedidos através de uma interface padrão (HTTP) e trocam representações desses recursos através de JSON. O servidor REST é stateless, ou seja, cada pedido do cliente traz todas as informações de contexto.

Nesta camada implementámos os mecanismos de segurança relativos à autenticação, à proteção no acesso aos recursos REST, à invalidação da sessão ao fim de um período de tempo de inatividade e à verificação da extensão do ficheiro de música carregado.

1. Autenticação

Quando um utilizador faz login na aplicação, o seu id de utilizador é adicionado à sessão. Sempre que um utilizador faz um pedido o servidor verifica se o seu id existe ou não na sessão. No caso de não existir a informação não é apresentada na página nem o REST processa o pedido e o utilizador é redirecionado para a página de login.

2. Acesso a recursos por URL

Sempre que o servidor recebe um pedido antes de responder verifica se o utilizador está autenticado, e, em caso negativo redireciona-o para a página de login. Desta forma protegemos o acesso a recursos de utilizadores não autenticados.

3. Sessão inválida ao fim de um período de inatividade

O servidor tem a capacidade de invalidar a sessão de um utilizador logado ao final de um determinado período de tempo sem atividade. Assim, um utilizador que não tenha feito o logout ou que tenha fechado o browser, a aplicação elimina a sua sessão.

4. Proteção conta as extensões dos ficheiros carregados

No carregamento de ficheiros de música para o servidor são permitidos apenas ficheiros com as extensões .mp3, .wav ou .flac, de forma a não permitir a execução remota de código no servidor.

Na tabela abaixo apresentamos todas os URIs, todas as páginas HTML e todos os ficheiros JavaScript utilizados.

application.py (backend)	Páginas HTML (frontend)	Ficheiros JS
/login [POST]	login.html	login.js
/logout [GET]	registar.html	registar.js
/utilizador [POST, GET, PUT, DELETE]	musicas.html	musicas.js
/musica [POST, GET]	playlists.html	playlists.js
/musica/<int:idMusica> [PUT, DELETE]	pesquisar.html	pesquisar.js
/playlist [POST]		bootbox.js
/playlist/<int:idPlaylist> [PUT, DELETE]		
/playlist/<string:tipo> [GET]		
/listaMusicasPlaylists/<int:idPlaylist> [GET]		
/adicionaMusicaPlaylist [POST]		
/removeMusicaPlaylist/<int:idPlaylist> [DELETE]		
/musicasSistema [GET]		
/musicasSistema/<string:titulo>/<string:artista> [GET]		
/paginaHtml [GET]		
/js/<path:paginaJs> [GET]		

Tabela 1 - application.py (backend), páginas HTML (frontend) e ficheiros JavaScript

4. Camada de apresentação (React)

A camada web foi desenvolvida recorrendo ao React, uma biblioteca de JavaScript para a construção de interfaces de utilizador baseadas em componentes reutilizáveis. Desta forma o React atualiza e renderiza eficientemente os componentes certos quando os seus dados forem alterados. Para tornar o frontend visualmente apelativo foi utilizada a framework de CSS, Bootstrap. Os métodos do REST são invocados a partir de pedidos AJAX e devolvem a informação através de JSON.

A título de exemplo e de forma muito genérica apresentamos a estrutura geral da página musicas.html dividida em componentes React. As restantes páginas seguem uma estrutura muito semelhante.

Figura 2 - Estrutura geral da página musicas.html dividida em componentes

- O componente 1 fornece as funcionalidades para editar o perfil, excluir a conta e fazer o logout. Este componente possui o método 'componentDidMount' para obter o nome do utilizador do servidor e colocá-lo no canto superior direito quando este é renderizado.
- O componente 2 permite ao utilizador navegar pelo menu da aplicação.

- O componente 3 é constituído pela tabela de músicas e pelo botão inserir música. Este componente possui o método 'componentDidMount' para preencher a tabela inicialmente com as músicas do utilizador e define ainda um timer que invoca um método do servidor ao fim de um período de tempo para atualizar a tabela.
- O componente 4 invocado pelo componente 3 é responsável por renderizar as linhas da tabela, uma a uma para permitir associar eventos para os botões de editar, remover e adicionar a playlist nessa linha (nessa música).

De seguida apresentamos 2 screenshots do frontend da aplicação web.

Figura 3 - Página das músicas do utilizador

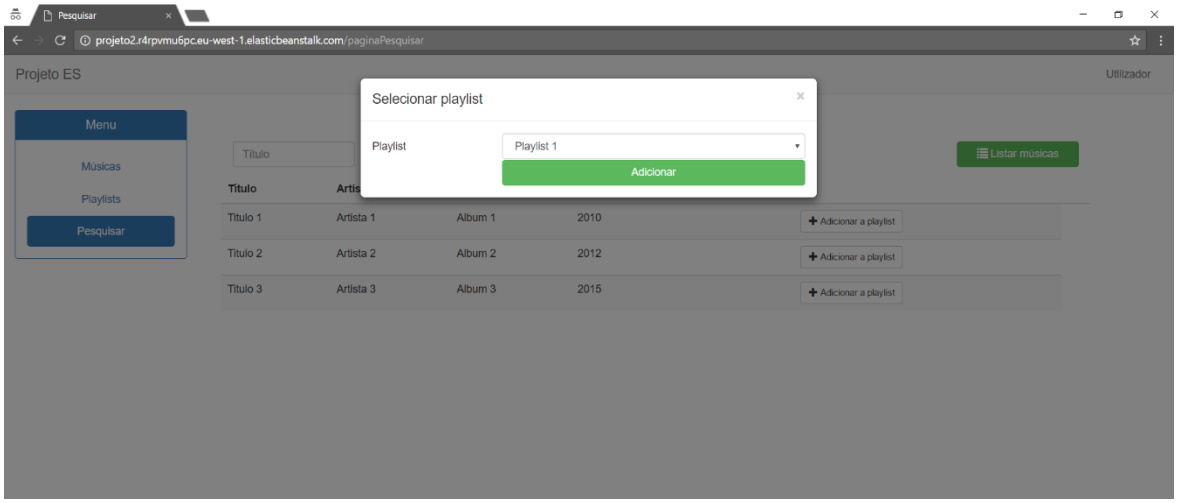


Figura 4 - Página de pesquisar músicas (com a adição de uma música a uma playlist)

5. Amazon AWS

Nesta seção abordamos o deploy da aplicação e os testes de escalabilidade realizados.

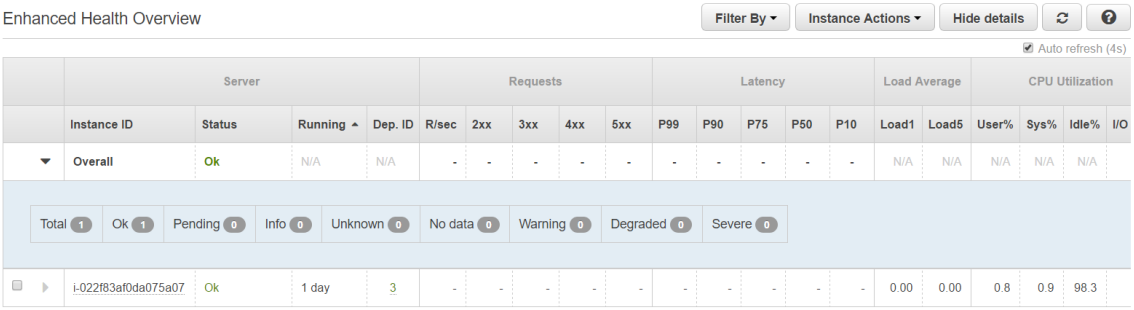
5.1 Deploy da aplicação

A nossa aplicação foi deployed na cloud da Amazon AWS, usando o serviço Elastic Beanstalk. Uma vez que a aplicação tem que suportar escalabilidade, os ficheiros das músicas não poderiam ficar armazenados no sistema de ficheiros de uma máquina. Para solucionar este problema recorremos a um S3 Bucket para armazenar os ficheiros das músicas.

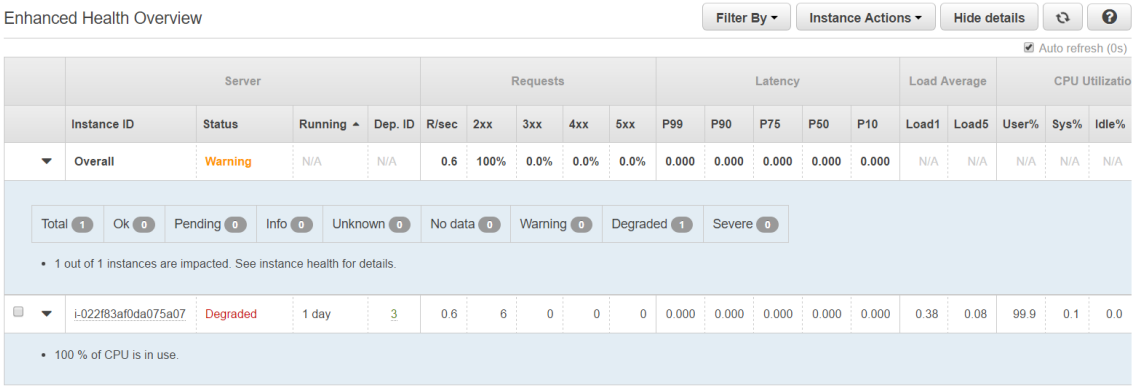
5.2 Testes de escalabilidade

Para as experiências de escalabilidade configurámos uma regra de utilização de CPU com um limite superior de 50% e um limite inferior de 20% para controlar a escalabilidade in e out (scaling in and out). Ou seja, se a ocupação de CPU subir acima de 50%, uma nova máquina é arrancada. Se, entretanto, a ocupação de CPU descer abaixo de 20% o mecanismo de dimensionamento automático remove máquinas, até que todas as máquinas deixam o serviço exceto uma.
Por uma questão de economia de custos fizemos esta experiência de escalabilidade com apenas duas máquinas. De seguida apresentamos a demonstração do descrito acima, passo a passo através de screenshots:

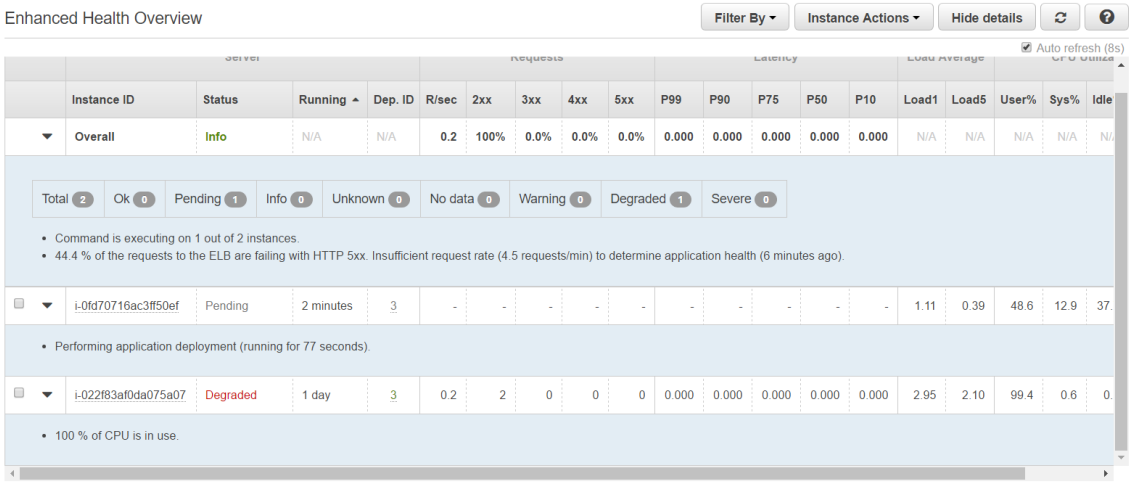
1. Elastic Beanstalk no estado inicial com apenas uma máquina



2. Ao executar o fibonacci para n = 45 a CPU ficou a 100% e o estado da máquina degradado

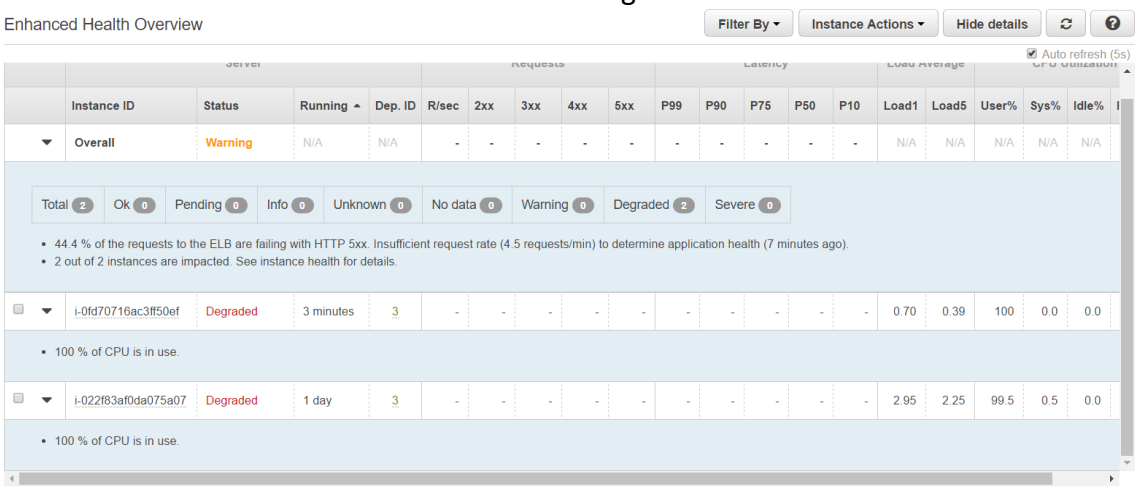


3. Ao final de uns minutos uma nova máquina foi lançada

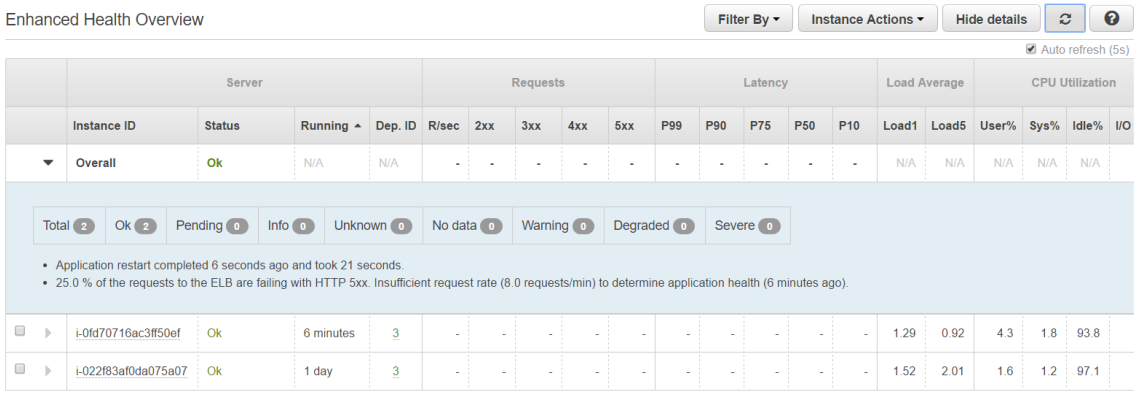


2017-04-13 23:38:35 UTC+0100	INFO	Added instance [i-0fd70716ac3ff50ef] to your environment.

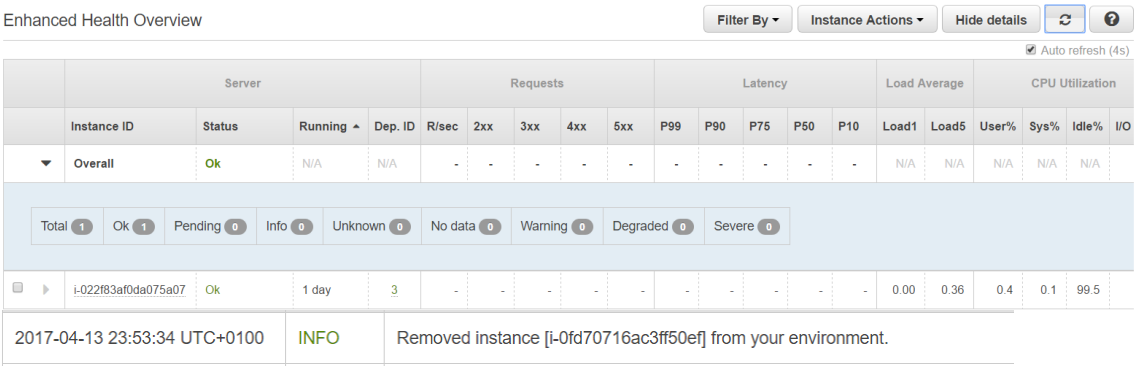
4. Ao executar o fibonacci para n = 45 com duas abas no brownser, ambas as máquinas ficaram com a CPU a 100% e os seus estados degradados



5. Após cancelar os pedidos ambas as máquinas ficaram com o estado OK



6. Ao final de uns minutos uma das máquinas desligou-se ficando só a máquina inicial



7. Gráfico da utilização de CPU durante a experiência de escalabilidade (23:30 até às 23:45)

6. Bibliografia

[1] http://docs.sqlalchemy.org/en/latest/orm/tutorial.html
[2] http://flask.pocoo.org/docs/0.12/quickstart/
[3] https://facebook.github.io/react/
[4] http://getbootstrap.com/
[5] http://bootboxjs.com/
[6] https://github.com/OAI/OpenAPI-Specification
[7] Slides - Setting Up a Beanstalk Environment