IPL escola superior de tecnologia e gestão instituto politécnico de leiria

Desenvolvimento de Aplicações Distribuídas

Eng. Informática	3º Ano	1º Semestre	2017-18	Época Recurso
Projeto	Data Limite Divulgação Resultados: 14 Mar 2018			
Data: 1 Fev 2018	**Data Entrega: 14 Fev 2018**			

Projeto – Época Recurso

OBJECTIVO

O objetivo deste projeto consiste em implementar uma plataforma Web com o Jogo da Sueca em *multiplayer*.

JOGO DA SUECA

O jogo da sueca é um jogo para 4 jogadores em duplas, ou seja, é jogado em equipas de dois jogadores com os parceiros dispostos em lugares opostos (de frente um para o outro). Apesar de existirem várias variantes do jogo, pretende-se com este projeto implementar um jogo da sueca que siga as regras descritas neste enunciado.

O baralho a utilizar deve ter 40 cartas, sendo removidos os 8s, 9s, 10s e jokers existentes nos baralhos padrão. Desta forma, as cartas a utilizar serão: ás, 7s, rei, valete, rainha, 6s, 5s, 4s, 3s e 2s.

O objetivo do jogo é ganhar cartas que valem pontos, de forma a atingir mais de 60 pontos. No total, um baralho tem 120 pontos e o valor das cartas é:

- Ás: 11 pontos;
- 7: 10 pontos;
- rei: 4 pontos;
- valete: 3 pontos;
- rainha: 2 pontos;
- 6, 5, 4, 3, 2: 0 pontos.

Sendo assim, a equipa que contabilize mais do que 60 pontos de acordo com as pontuações anteriormente exibidas, vence o jogo. Caso ambas as equipas tenham 60 pontos, considera-se que há um empate.

O jogo joga-se no sentido horário e cada jogador terá 10 cartas para jogar. As cartas são baralhadas e distribuídas no sentido horário. O primeiro jogador a receber jogo será escolhido aleatoriamente e a primeira carta que recebe deverá ficar visível para todos os jogadores, uma vez que o seu naipe será o trunfo. As restantes 9 cartas só deverão ficar visíveis para o jogador. De

seguida, são distribuídas 10 cartas aos restantes jogadores, no sentido dos ponteiros do relógio, devendo as mesmas ficar visíveis apenas ao respetivo jogador.

O jogo começa a ser jogado pelo jogador que está imediatamente à esquerda do jogador que recebeu primeiro a sua mão. Os outros jogadores devem seguir o naipe jogado pelo primeiro jogador. Caso um jogador não tenha cartas desse naipe, pode jogar uma carta de qualquer outro naipe, incluindo o trunfo. A carta mais alta do naipe em jogo, ou o trunfo mais alto (o trunfo mais alto ganha sempre aos restantes naipes), ganha a rodada. O jogador que ganhou será o primeiro a jogar a próxima jogada, escolhendo o naipe.

Caso um jogador minta sobre a ausência de um naipe na sua mão, e seja detetado, considera-se que está a fazer "renúncia" ao jogo. Neste caso, a dupla oponente ganha o jogo automaticamente. A implementação do jogo deverá incluir uma opção para desconfiar da outra equipa enquanto o jogo não terminar. Caso a outra equipa tenha feito "renúncia" em qualquer momento do jogo (em qualquer jogada desde o inicio até ao momento em que é declarada a desconfiança), o jogo termina com a vitoria da equipa do jogador que desconfiou. Caso contrário, a vitória será atribuída à equipa adversária.

PONTUAÇÃO

Quando os jogos terminam, cada um dos jogadores da equipa vencedora recebe 1, 2 ou 4 pontos que irão somar à sua conta pessoal. Também poderão ser descontados pontos caso a sua equipa perca o jogo devido a uma renuncia confirmada. As regras para a pontuação de cada jogo são as seguintes:

- Os jogadores da equipa vencedora recebem:
    - 1 ponto cada, se o total da pontuação (das cartas) da equipa >=61 e <=90;
    - 2 pontos cada, se o total da pontuação (das cartas) da equipa >= 91 e <=119;
    - 4 ponto cada, se o total da pontuação (das cartas) da equipa = 120;
- Em caso de empate ou derrota (sem renuncia) os jogadores recebem 0 pontos;
- Em caso de derrota devido a uma renuncia confirmada, são descontados 4 pontos aos jogadores da equipa que fez a renuncia e são atribuídos 4 pontos aos jogadores da equipa vencedora (que declarou desconfiança);
- Em caso de derrota devido a uma desconfiança não confirmada, são descontados 4 pontos aos jogadores da equipa que declarou desconfiança (não confirmada) e são atribuídos 4 pontos aos jogadores da equipa vencedora.

A plataforma deverá manter o total de jogos, a pontuação total (soma da pontuação de todos os jogos) e pontuação média (pontuação total a dividir pelo número de jogos) de todos os jogadores inscritos.

CENÁRIO

Neste projeto deverá ser implementada uma plataforma Web com o Jogo da Sueca, em *multiplayer* (para 4 jogadores). Os jogos podem ser criados por qualquer utilizador registado e qualquer outro utilizador registado pode entrar no jogo.

ARQUITETURA E TECNOLOGIAS

A plataforma Web a desenvolver deverá incluir duas aplicações Web separadas, uma para a administração da plataforma e outra que servirá como *frontend* do jogo. As duas aplicações deverão funcionar de forma integrada de acordo com a arquitetura ilustrada na figura 1 e com as restrições tecnológicas enunciadas de seguida.

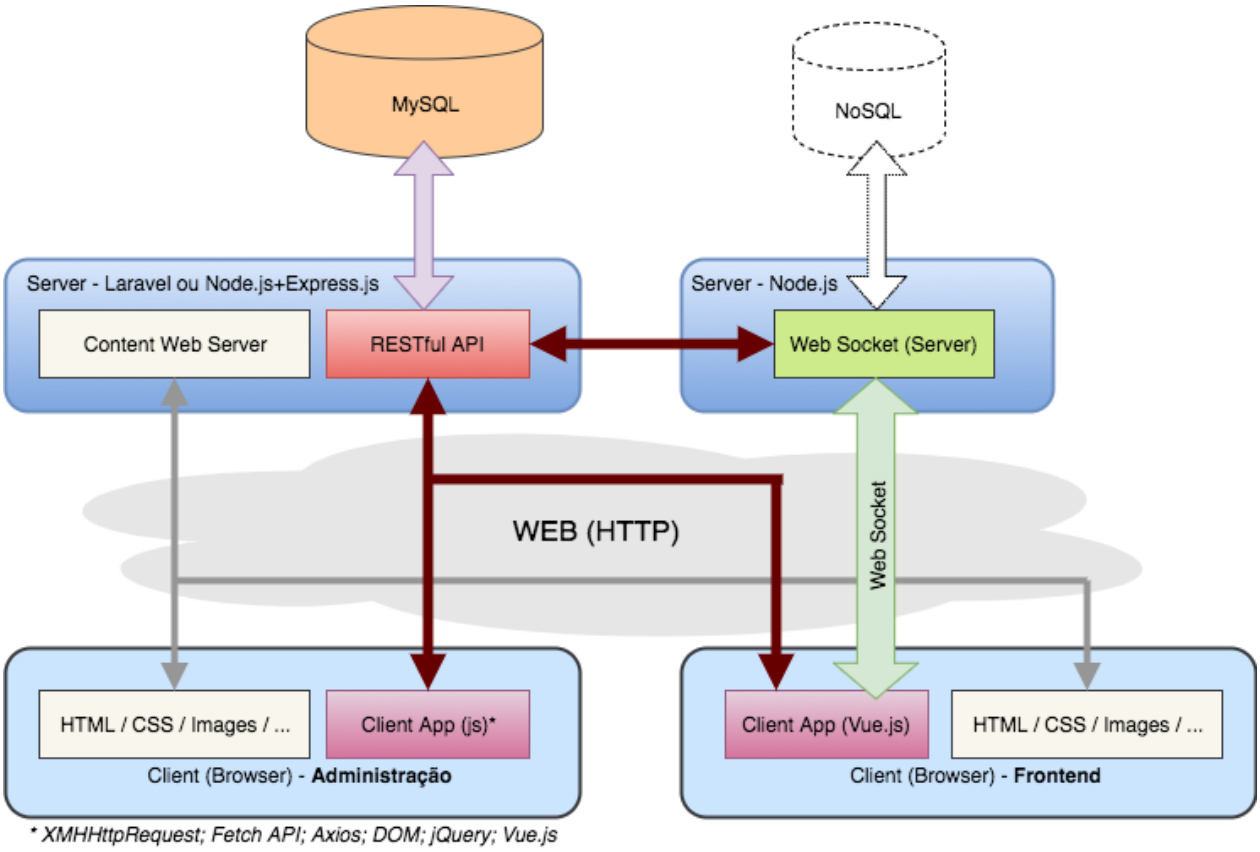

Figura 1: Arquitetura proposta para a solução

- O principal repositório deverá ser implementado numa base de dados relacional MySQL;
- O servidor Web clássico (Content Web Server) e a API RESTful poderão ser implementadas em Laravel (PHP) ou em Node.js com Express.JS (JavaScript);
- A aplicação de administração deverá ser uma aplicação Web SPA (Single-Page Application) ou Web híbrida. Para a programação do cliente desta aplicação pode-se utilizar qualquer das seguintes tecnologias: XMLHttpRequest; Fetch API; Axios; DOM; jQuery (com ou sem *plugins*); Vue.js.

- A aplicação de *frontend* deverá ser uma aplicação Web SPA (Single-Page Application). A programação do cliente desta aplicação deverá utilizar a *framework* Vue.js, complementada com as bibliotecas e *packages* apropriados (para comunicação HTTP; WebSockets; apresentação de dados; etc.);
- A componente reativa da aplicação de *frontend*, que prossupõe o envio de mensagens em tempo real (quase real) entre os vários intervenientes do jogo, terá que ser implementada através de WebSockets. No servidor, os WebSockets terão que ser implementados num servidor Node.js;
- Se for necessário ao servidor de WebSockets utilizar um serviço de dados para facilitar a gestão dos dados dos jogos, jogadas e jogadores, ou para implementar a persistência ou *cache* desses dados, deverá ser utilizada uma das seguintes bases de dados NoSQL: "Redis", "Memcached " ou "MongoDB".

FUNCIONALIDADES

As funcionalidades mínimas a implementar nas duas aplicações (Administração e *Frontend*) serão descritas nas secções respetivas. Os estudantes deverão também implementar outras funcionalidades que considerem relevantes no contexto deste projeto, que serão descritas na secção "Outras funcionalidades".

FRONTEND

A aplicação de *frontend* corresponde à aplicação Web a que os utilizadores da plataforma têm acesso, e na qual poderão jogar. Esta aplicação deverá incluir as seguintes funcionalidades:

- Registar como novo utilizador. O registo implica o preenchimento do e-mail, da senha de entrada, do nome completo e *nickname*. O e-mail e *nickname* deverão ser únicos e ambos poderão ser utilizados como credencial (*login*) para entrar na aplicação. O registo deverá ser confirmado só depois do utilizador receber um e-mail de confirmação e clicar na hiperligação de ativação;
- Gestão da conta do utilizador. Na gestão da sua conta, o utilizador poderá consultar e alterar todos os dados que foram introduzidos no registo (incluindo a senha de entrada). Também deverá ser possível apagar ou anular a sua conta;
- *Lobby*. No *lobby* de entrada dos jogos, deverão aparecer os jogos que estão pendentes, ou seja, que estão à espera de jogadores para começar. Depois de começar um jogo, este já não deverá aparecer no *lobby* (passa do estado "pendente" a "ativo").

  No *lobby*, cada utilizador pode criar um novo jogo ou juntar-se a um jogo pendente. Os jogos começam quando tiverem 4 jogadores.

  A lista de jogos pendentes no *lobby* inclui apenas o id do jogo, o *nickname* do criador do jogo; a data e hora (hora:minutos:segundos) em que o jogo foi criado e o nº de jogadores que já se juntaram ao jogo.

O *lobby* deverá ser o mais reativo possível, ou seja, a lista de jogos pendentes deverá ser atualizada o mais rapidamente possível sempre que forem criados, alterados ou removidos jogos do *lobby*;

- Zona de jogo. Zona da aplicação onde o utilizador poderá jogar um, ou vários jogos em simultâneo (vários jogos distintos, do mesmo utilizador, a decorrer em simultâneo na mesma página). Os jogos deverão ser reativos, de forma a que a visualização do estado do jogo e todas as iterações com o mesmo decorram em "tempo real" e em simultâneo para todos os jogadores envolvidos.

  Os utilizadores poderão ter acesso à zona de jogo através de uma hiperligação, menu ou opção equivalente, ou em alternativa, automaticamente depois do jogo ser criado pelo próprio ou quando o utilizador se junta ao jogo. Note que o processo, ou processos de acesso ao jogo ou jogos, depende da forma como o *lobby* e os jogos são implementados.

  Depois de um jogo terminar, deverá aparecer informação relativa ao fim do jogo (a indicação que o jogo terminou, quem ganhou o jogo, etc.) acrescida de uma hiperligação, botão ou equivalente que permita que o jogo seja removido da zona de jogo;
- Estatísticas. Todos os utilizadores, incluindo o publico em geral (não autenticado), deverão ter acesso às estatísticas gerais de utilização da plataforma, incluindo: o total de jogadores na plataforma, o total de jogos já jogados na plataforma, o *top* com os 5 jogadores com mais jogos, o *top* com os 5 jogadores mais pontuados e o *top* com os 5 jogadores com melhor média.

  Caso as estatísticas sejam visualizadas por um utilizador autenticado, este deverá também ver o total de jogos que ele próprio já jogou, o seu total de vitórias, empates e derrotas, a sua pontuação total e a sua pontuação média.

ADMINISTRAÇÃO

A aplicação de administração permite ao administrador da plataforma gerir alguns parâmetros de utilização e consultar estatísticas de utilização da mesma. O *login* inicial (pré-definido) de acesso a esta aplicação é "**admin**", se for usado o *nickname*, ou "**admin@mail.dad**", se for usado o e-mail. A senha de entrada inicial é "**secret**". Esta aplicação deverá incluir as seguintes funcionalidades:

- Alterar senha de entrada do administrador. O administrador pode alterar a senha de entrada – não esquecer que só deverá ser possível a alteração da senha se o utilizador introduzir também a senha antiga;
- Configuração do e-mail do administrador e da plataforma. Deverá ser possível configurar o e-mail do administrador e o e-mail da plataforma. Este último será utilizado para envio automático de mensagens;

- *Reset* da senha de administrador. Caso o administrador se esqueça da senha de entrada na administração, poderá fazer um *reset* à mesma. Neste caso, será enviado um link para ativar/confirmar a senha por e-mail (para o e-mail do administrador);
- Gestão dos utilizadores. O administrador deverá ter acesso a toda a informação relativa a todos os utilizadores registados na plataforma (exceto a senha de entrada), e deverá poder bloquear, reativar ou remover (ou anular) qualquer utilizador (exceto o próprio) da plataforma. No processo de bloqueio, reativação ou remoção, seria interessante que o utilizador recebesse de forma automática um e-mail de aviso, e que este e-mail incluísse a razão (a preencher pelo administrador) pela qual o utilizador foi bloqueado, reativado ou removido;
- Gestão dos baralhos. A representação de um baralho deverá incluir 40 imagens (não são necessários os 8s, 9s, 10s e Jokers) relativas às cartas e uma imagem relativa à face escondida das cartas. A plataforma deverá suportar vários baralhos que serão utilizados aleatoriamente nos jogos (cada jogo irá utilizar um baralho). O administrador deverá gerir os baralhos, ou seja, criar, alterar ou apagar baralhos. Para cada um dos baralhos, o administrador deverá poder acrescentar (através de *upload*) ou remover imagens e associá-las às cartas e à face escondida. A plataforma deverá garantir que cada baralho é preenchido na totalidade antes de ser utilizado nos jogos.

  É importante que a plataforma garanta que as imagens das cartas tenham o formato e tamanhos mais adequados aos tabuleiros de jogo, por forma a otimizar o desempenho do jogo e a garantir que o *layout* é sempre o mais adequado.
- Estatísticas. O administrador deverá ter acesso a estatísticas de utilização da plataforma, nomeadamente: o total de jogos já jogados na plataforma; a lista completa e ordenável de jogadores com informação sobre o total de jogos, as vitórias, empates e derrotas de cada um; o histórico dos totais de jogos jogados em cada dia (de preferência com um gráfico que permita ver a evolução da utilização da plataforma).

OUTRAS FUNCIONALIDADES

O projeto deve incluir outras funcionalidades que lhe acrescentem valor e que deverão ser definidas pelos estudantes. A avaliação destas funcionalidades depende da complexidade e quantidade das funcionalidades implementadas. Cada funcionalidade é associada a um nível de complexidade com valores de 1 a 4 (1 – mais simples; 4 – mais complexo). Para obter a nota máxima no critério "Outras Funcionalidades", os estudantes deverão implementar na perfeição (nota=100%) um conjunto de funcionalidades cujo nível total de complexidade seja igual a 4, ou por exemplo, ter uma nota de 50% em todas as funcionalidades de um conjunto com um nível total de complexidade igual a 8.

Exemplo de cálculo:

Funcionalidades implementadas: f1; f2; f3; f 4

Níveis de complexidade:	f1 = 1; f2 = 2; f3 = 1; f4 = 3
Notas individuais:	f1 = 60%; f2 = 30%; f3 = 100%; f4 = 40%
Nota Final:	(60% * 1 + 30% * 2 + 100% * 1 + 40% * 3) / 4 = 85%

De seguida, são apresentadas algumas sugestões de funcionalidades possíveis associadas a níveis de complexidade. Os níveis de complexidade servem apenas de referência e podem variar de acordo com a funcionalidade em concreto.

BOTS (*Nível complexidade = 3*)

Permitir jogar contra jogadores não humanos (*bots*). Os *bots* poderão, ou não, ser inteligentes (com possível aumento do nível de complexidade para 4). Caso esta funcionalidade seja implementada, deverá ser possível ao criador do jogo adicionar *bots* enquanto o jogo está pendente. Para todos os efeitos, um *bot* substituirá um jogador humano - conta para o número total de jogadores e não deverá afetar a experiência de jogo dos restantes jogadores.

SERVIÇOS DE AUTENTICAÇÃO EXTERNOS (*Nível complexidade = 2*)

Permitir que a autenticação seja feita por outros serviços, como por exemplo *Facebook Login for the Web*, *Google Identity Platform*, *Twitter Sign In*, etc. Alguma informação sobre estes serviços:

- https://developers.facebook.com/docs/facebook-login/web
- https://developers.google.com/identity/
- https://developers.google.com/identity/sign-in/web/sign-in
- https://dev.twitter.com/web/sign-in

AVATARES (*Nível complexidade = 1*)

Permitir que os utilizadores possam ser representados não apenas pelo seu nome, mas também por um avatar (imagem ou foto).

CHAT ROOMS (*Nível complexidade = 1*)

Permitir que os utilizadores comuniquem entre si através de salas de *chat*. Poderá existir uma sala de *chat* pública no *lobby* (acessível a todos os utilizadores) e uma sala de *chat* privada por cada jogo (acessível apenas aos jogadores desse jogo).

REPETIÇÃO DE JOGO (*Nível complexidade = 4*)

Permite visualizar uma animação que mostre todas as jogadas de um jogo que já decorreu. Deverá ser percetível quais as cartas que cada jogador tem em determinado momento, incluindo as que estão escondidas dos restantes jogadores – poderá ser interessante existir uma representação distinta para as cartas que estão visíveis e as que estão escondidas.

APOSTAS E INTEGRAÇÃO COM SERVIÇOS FINANCEIROS (*Complexidade = 4*)

Permitir aos utilizadores jogar jogos com apostas, em que todos os jogadores entram com um valor fixo (*e.g.*: 4 créditos) e os vencedores recebem (em partes iguais) os créditos de todos. Para evitar trabalhar diretamente com dinheiro real, a plataforma pode implementar um sistema de créditos atribuídos a cada utilizador e manter uma conta corrente por utilizador. A conversão dos créditos em dinheiro real e vice-versa, deverá utilizar serviços como o Stripe ou Braintree através do *package* "Laravel Cashier" ou equivalentes.

ESTRUTURA DE DADOS

Os estudantes são livres para definir a estrutura das bases de dados necessárias ao desenvolvimento da plataforma. No entanto, na figura 2 é apresentada uma sugestão de estrutura para a base de dados relacional. Caso os estudantes optem por utilizar a plataforma Laravel na implementação da API RESTful, podem utilizar o ficheiro disponibilizado com as *migrations* necessárias para a sua criação.

A estrutura de dados inclui as seguintes entidades:

- users: utilizadores registado no sistema (inclui também o administrador);
- games: todos os jogos criados;
- config: parâmetros de configuração da plataforma;
- decks: repositório de baralhos da plataforma (inclui o caminho da imagem para a face escondida das cartas);
- cards: cartas de um baralho (inclui o caminho da imagem da carta);
- password_resets: tabela gerada pelo Laravel para gestão de *passwords*.

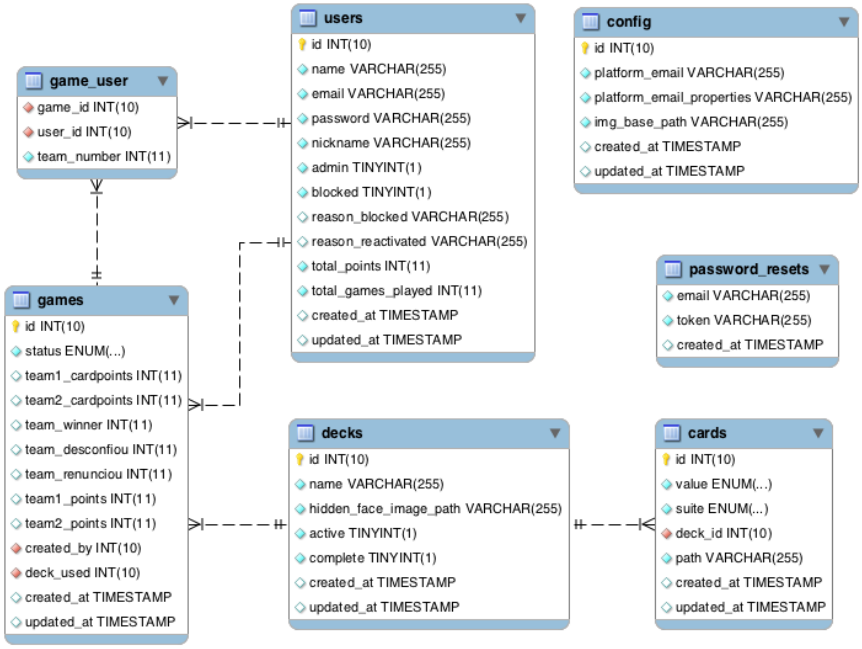

Figura 2: Sugestão para a estrutura de dados

PROJETO

Além dos requisitos funcionais e de arquitetura especificados anteriormente, o projeto deve obedecer a alguns requisitos não-funcionais relacionados com a estrutura da aplicação e do código, assim como com a interação entre o cliente Web e o servidor.

REQUISITOS NÃO-FUNCIONAIS

Devem ser considerados alguns requisitos não-funcionais, nomeadamente:

- A arquitetura da aplicação deve seguir as boas práticas de *design* e as tecnologias utilizadas para cada um dos seus componentes devem ser escolhidas de forma adequada, de acordo com o contexto de utilização e tendo em conta as restrições de arquitetura e tecnologia previamente estabelecidas;
- A estrutura e o código da aplicação devem seguir o princípio da separação de responsabilidades, onde diferentes unidades de software (ficheiros, módulos, objetos, funções, etc.) devem ser o mais independente e desacopladas possível;
- O desempenho da aplicação deve ser otimizado. Por exemplo: o nº de mensagens HTTP e o tamanho das imagens e dos dados enviados nas mensagens deve ser minimizado para evitar latência na rede; as consultas à base de dados devem ser otimizadas e devem devolver apenas os dados estritamente necessários; os movimentos nos tabuleiros de jogo devem ser propagados quase instantaneamente para todas as aplicações cliente;
- A aplicação deve ser o mais segura possível, garantindo não apenas que os recursos e características funcionais estão disponíveis apenas para os utilizadores apropriados, mas também que todos os movimentos no jogo feitos por um jogador são mesmo executados por esse jogador – elementos externos não devem ser capazes de interferir com o jogo.

ENTREGA E PUBLICAÇÃO

A aplicação **tem que ser publicada** num serviço externo acessível a partir de qualquer dispositivo na Internet. A correção funcional da aplicação será feita exclusivamente na versão publicada, pelo que a publicação **não é opcional**. Os estudantes são livres de escolher o serviço a utilizar, no entanto são apresentadas algumas sugestões:

- Amazon Free Tier (https://aws.amazon.com/free/)
- Digital Ocean com o GitHub Education Pack (https://education.github.com/pack)
- Openshift da RedHat (https://www.openshift.com/)

O relatório de projeto cujo modelo será fornecido pelos docentes da UC através da plataforma Moodle, tem entrega obrigatória. Projetos que não sejam acompanhados do respetivo relatório devidamente preenchido serão classificados com 0 (zero) valores.

Para além disso, todo o código fonte do projeto deve ser entregue num **ficheiro zip**, utilizando o *link* disponibilizado na página da Unidade Curricular no Moodle. O ficheiro zip deve conter:

- **info.txt**: ficheiro com informação sobre o elemento ou grupo de projeto, nomeadamente, número de grupo de projeto, nome e número dos estudantes, turno Prático-Laboratorial, URL de acesso à aplicação publicada e credenciais para teste da aplicação.
- **Pasta "code"**: Pasta que contém todos os ficheiros necessários para instalar e executar a plataforma.
- **Pasta "MySQL-bak":** Pasta com um *backup* da base de dados MySQL
- **Pasta "Other-bak":** Caso necessário, uma pasta com um *backup* das restantes bases de dados utilizadas (redis, Memcached ou MongoDB)

AVALIAÇÃO

A tabela seguinte sumaria os critérios de correção para avaliação do projeto. Cada critério será avaliado do ponto de vista de utilização da aplicação (funcionalidades, usabilidade, fiabilidade, desempenho, segurança, etc.) e de implementação interna (arquitetura, padrões de desenho, qualidade do código, desempenho e otimização, etc.).

Nº	Peso	Critério
1	**35%**	Aplicação de administração
2	**50%**	Aplicação *frontend* (jogo)
3	**15%**	Outras Funcionalidades