Arquitetura de Software

Rede de partilha de playlists

Bruno Madureira, Fábio Moura, Nuno Miranda
Departamento de Engenharia Informática, FCTUC
Universidade de Coimbra
Coimbra, Portugal

***Sumário* — Este documento apresenta documentação arquitetural sobre o desenvolvimento de uma aplicação web de criação e partilha de playlist musicais. Apresentaremos os requisitos (funcionais e não funcionais) assim como a arquitetura resultante deste estudo. A nossa arquitetura é uma versão alterada do modelo mvc, para melhor responder aos nossos requisitos. Por fim fizemos uma avaliação da forma que a arquitetura responde aos requisitos.**

I. Introdução

Este documento apresenta um conjunto de drivers arquiteturais de um sistema, uma arquitetura resultante das mesmas e a respetiva avaliação arquitetural.

O sistema será uma plataforma *online* de criação e partilha de playlists de música. O sistema será suportado monetariamente através de publicidade ou de um modelo de subscrição *premium*.

II. Drivers Arquiteturais

TABELA I. Registo

Atores Primários	Utilizador free
Atores Secundários	
Descrição	Utilizador efetua registo na aplicação
Trigger	Desejo de experimentar a aplicação
Pré-condição	Não ter conta na aplicação
Pós-condição	Conta criada com sucesso

TABELA II. Login

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador efetua login na aplicação
Trigger	Desejo de fazer login na aplicação
Pré-condição	Ter conta na aplicação ou conta em plataforma pré-existente
Pós-condição	

TABELA III. Recuperar Password

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador pretende recuperar os seus dados de login
Trigger	Login falhado / Utilizador não se lembra das credenciais de acesso
Pré-condição	Ter registo efetuado na aplicação
Pós-condição	Dados de login recuperados com sucesso

TABELA IV. Utilizar plataformas pré-existentes

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador pretende entrar na aplicação com a sua conta de plataformas pré-existentes (e.g.: facebook, twitter, Google)
Trigger	Utilizador pretende usar conta de plataforma pré-existente poupando assim tempo no registo manual de aplicação
Pré-condição	Ter uma conta numa plataforma
Pós-condição	Login efetuado com sucesso

TABELA V. Criar playlist

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador cria uma playlist
Trigger	Desejo de criar uma playlist para futura referência
Pré-condição	Login efetuado
Pós-condição	Playlist criada

TABELA VI. Partilhar playlist

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador partilha uma playlist com seguidores
Trigger	Vontade de mostrar músicas que gosta aos seus seguidores
Pré-condição	Ter playlist criada
Pós-condição	Seguidores podem visualizar a playlist criada

TABELA VII. Partilhar playlist numa rede social

Atores Primários	Utilizador free e premium
Atores Secundários	

Descrição	Utilizador partilha uma playlist no perfil de uma rede social
Trigger	Vontade de mostrar músicas que gosta a quem está ligado numa dada rede social
Pré-condição	Ter playlist criada e ter conta de uma rede social associada
Pós-condição	Utilizadores da rede têm acesso ao link para a playlist

TABELA VIII. SEGUIR PLAYLIST

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador segue uma playlist
Trigger	Utilizador acha uma playlist interessante e deseja segui-la para receber futuras atualizações
Pré-condição	Seguir playlist existente
Pós-condição	Playlist seguida com sucesso

TABELA IX. AVALIAR PLAYLIST

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador avalia uma playlist
Trigger	Utilizador consulta uma playlist
Pré-condição	Ter playlist criada
Pós-condição	Playlist avaliada com sucesso

TABELA X. PARTILHAR PLAYLIST

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador partilha uma playlist com seguidores
Trigger	Vontade de mostrar músicas que gosta aos seus seguidores
Pré-condição	Ter playlist criada
Pós-condição	Seguidores podem visualizar a playlist criada

TABELA XI. VISUALIZAR PLAYLIST SUGERIDA

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador visualiza playlist sugerida
Trigger	Utilizador deparasse com playlist sugerida
Pré-condição	Ter playlist criada
Pós-condição	Sugestão visualizada

TABELA XII. PROCURAR UTILIZADOR

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador pesquisa outro utilizador
Trigger	Utilizador pretende encontrar um amigo
Pré-condição	Utilizador que procura estar registado
Pós-condição	Apresentação da lista de resultados

TABELA XIII. PROCURAR MÚSICA

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador procura música
Trigger	Desejo de procurar uma música
Pré-condição	Utilizador estar registado
Pós-condição	Apresentação dos resultados da pesquisa

TABELA XIV. ADICIONAR MÚSICA A UMA PLAYLIST

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador adiciona música a uma playlist
Trigger	Utilizador pretende adicionar uma música a uma playlist
Pré-condição	Música tem de existir na base de dados
Pós-condição	Música adicionada com sucesso

TABELA XV. SEGUIR UTILIZADOR

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador segue outro utilizador
Trigger	Utilizador acha interessante os gostos músicas de outro utilizador
Pré-condição	Procurar utilizador / utilizador a adicionar ser dono de uma playlist visualizada
Pós-condição	Utilizador seguido com sucesso

TABELA XVI. VISUALIZAR PERFIL

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador visualiza perfil de outro utilizador
Trigger	Desejo de visualizar perfil de outro utilizador
Pré-condição	Utilizador está deparado com perfil de outro utilizador
Pós-condição	Perfil de outro utilizador visualizado

TABELA XVII. FAZER PAGAMENTO PREMIUM

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador free faz pagamento premium para se tornar utilizador premium ou utilizador premium faz pagamento para manter o seu estatuto
Trigger	Desejo de se tornar utilizador premium ou de manter esse estatuto
Pré-condição	Ser utilizador registado
Pós-condição	Pagamento efetuado

TABELA XVIII. ALTERAR CONFIGURAÇÕES

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador altera configurações
Trigger	Utilizador pretende atualizar os seus dados pessoais
Pré-condição	Utilizador registado
Pós-condição	Configurações alteradas

TABELA XIX. ACEDER AO FÓRUM

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Utilizador acede ao fórum
Trigger	Desejo de partilhar opiniões
Pré-condição	Utilizador registado
Pós-condição	Discussão em curso

TABELA XX. PROCURAR MÚSICA NO SPOTIFY OU YOUTUBE

Atores Primários	Utilizador Premium
Atores Secundários	
Descrição	Utilizador procura uma dada música no spotify ou youtube
Trigger	Utilizador encontra uma música desconhecida numa playlist e pretende ouvi-la
Pré-condição	Utilizador com estatuto *premium*
Pós-condição	Lista de músicas encontradas

TABELA XXI. VER TENDÊNCIAS

Atores Primários	Utilizador Premium
Atores Secundários	
Descrição	Utilizador vê as playlists mais ouvidas
Trigger	Desejo de saber o que é correntemente ouvido
Pré-condição	Utilizador com estatudo *premium*
Pós-condição	Conjunto de listas mais ouvidas

TABELA XXII. VISUALIZAR LISTA DE PRÓXIMOS CONCERTOS DE BANDAS INCLUÍDAS NA PLAYLIST

Atores Primários	Utilizador Premium
Atores Secundários	
Descrição	Utilizador visualiza lista dos próximos concertos de bandas incluídas numa dada playlist
Trigger	Vontade de saber qual o próximo concerto de uma das suas bandas favoritas
Pré-condição	Utilizador registado como *premium*
Pós-condição	Utilizador visualiza lista dos concertos

TABELA XXIII. NOTIFICAÇÃO POR EMAIL

Atores Primários	Utilizador free e premium
Atores Secundários	
Descrição	Receber notificações de partilhas, avaliações de playlists e de novos concertos por e-mail (caso *premium*)
Trigger	Utilizador pretende manter um histórico no seu email de toda a atividade na aplicação
Pré-condição	Utilizador registado
Pós-condição	Utilizador irá receber emails

TABELA XXIV. SINCRONIZAR COM GOOGLE CALENDAR

Atores Primários	Utilizador Premium
Atores Secundários	
Descrição	Sincronização do calendário de concertos com o Google Calendar
Trigger	Utilizador sente que demora muito tempo para visualizar o calendário de concertos na aplicação e pretende ter acesso ao mesmo a partir de outras aplicações
Pré-condição	Utilizador com estatuto *premium*
Pós-condição	Calendário sincronizado

III. DESCRIÇÃO DA ARQUITETURA

A arquitetura proposta baseia-se num típico modelo MVC (*Model-view-controller*) mas com algumas alterações para conseguirem suportar com as necessidades impostas pelas restrições e atributos de qualidade.

O sistema é composto por 3 tipos de entidades: *Load balancer*, *Web Server* e Bases de dados.

As nossas alterações ao modelo MVC foram a inclusão de um *load balancer* que irá permitir ter vários servidores a fazer alojamento do serviço mantendo o mesmo domínio. Isto irá permitir também uma fácil e rápida adição de novos servidores sem ter de alterar grandes quantidades de código.

A. Restrições arquiteturais de negócio

1) Ambientes suportados

Página web apenas inclui suporte para desktop, excluindo suporte para todas as plataformas móveis, visto que o número de utilizadores do nosso público-alvo é bastante superior a nível de computador e o custo de migrar a plataforma para mobile não justifica o investimento.

2) Limitações para utilizadores free

Nos utilizadores free será limitado o número de playlists (para 10 playlists) e o número de músicas por playlist (30 músicas). Nos utilizadores *premium* estes parâmetros serão ilimitados. Desta forma será promovida a adesão ao modelo *premium*.

3) Publicidade na página web

Nos utilizadores free irá ser mostrada publicidade para assegurar a rentabilidade do negócio.

4) Escalabilidade

A arquitetura deve garantir escalabilidade, isto é, deverá ter particionamento de servidores e dados.

5) Dados de armanzenamento de músicas

As músicas não iram estar guardadas na nossa base de dados pois isso iriam implicar contratos e custos extra. Além disso não está no intuito do nosso negócio ser um fornecedor de música mas apenas um serviço de partilha de playlists e discussão musical. No entanto utilizaremos a API do *spotify* e do *youtube* para facilitar o acesso das músicas que o utilizador queira ouvir (nesses serviços). Os dados que iram ser guardados sobre as músicas é apenas o nome das mesmas assim como informação adicional (ano de lançamento, autor, álbum, informação de vendas, etc.).

B. Restrições arquiteturais técnicas

1) Construção da página web

Página web será otimizada apenas para os *browsers Firefox* (a partir da versão 38.0.1) e *Google Chrome* (a partir da versão 43.0).

2) Autenticação com contas externas

O Sistema de autenticação a partir das APIs das redes sociais será feita apenas através do *twitter*, *Google+* e *facebook*.

3) Base de dados de músicas

O Sistema vai usar API do *Spotify* e do *Youtube* para pesquisa de músicas, guardando em base dados apenas o nome das mesmas. Se a música não se encontrar em ambos os serviços o próprio utilizador poderá inseri-la no sistema, enriquecendo assim a base de dados atual.

4) Ouvir música

O sistema vai usar a API do *Spotify* e do *Youtube* para reproduzir a música.

5) Pagamentos

Os pagamentos serão efetuados apenas por *paypal*.

6) Partilha de playlists

A partilha de playlists será assegurada através das redes sociais *Facebook*, *twitter* e *google+* apenas.

C. Atributos de Qualidade

TABELA XXV. SISTEMA FACILMENTE MODIFICÁVEL

Fonte de estímulo	Ambiente de negócio
Estímulo	Mudanças no negócio ditam mudanças arquiteturais
Ambiente	Desenvolvimento e produção
Componente estimulado	Sistema
Resposta	Custo de mudança arquitetural
Medida de resposta	Custo total das mudanças anuais deverá ser no máximo 30% da receita líquida anual.

TABELA XXVI. SEGURANÇA 1

Fonte de estímulo	Utilizador
Estímulo	Utilizador regista-se e o sistema guarda a sua informação crítica na base de dados de forma cifrada
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Percentagem de informação privada que se encontra cifrada
Medida de resposta	100% dos dados críticos devem estar cifrados

TABELA XXVII. SEGURANÇA 2

Fonte de estímulo	Utilizador
Estímulo	Utilizador efetua login
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Percentagem da informação enviada neste pedido que se encontra encriptada
Medida de resposta	100% da informação deve estar encriptada

TABELA XXVIII. DISPONIBILIDADE DO SISTEMA 1

Fonte de estímulo	Utilizador
Estímulo	Utilizador tenta aceder ao sistema
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Percentagem de tempo em que o sistema está ativo durante o ano
Medida de resposta	Certificar que a percentagem de disponibilidade é igual ou superior a 99,9%

TABELA XXIX. DISPONIBILIDADE DO SISTEMA 2

Fonte de estímulo	Utilizador maligno
Estímulo	Utilizador tenta atacar o sistema usando SQL injection
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Número de ataque efetuados com sucesso
Medida de resposta	Número de ataques graves efetuados com sucesso deverá ser zero e o número de ataques pouco graves efetuados com sucesso deverá ser inferior a 5 por ano.

TABELA XXX. DISPONIBILIDADE DO SISTEMA 3

Fonte de estímulo	Máquina onde está alojado o servidor ou a base de dados
Estímulo	Máquina falha
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Tempo que demorará ao serviço passar a ser fornecido por outra máquina
Medida de resposta	O tempo deverá ser inferior a 5 segundos (mudança deve ser automática)

TABELA XXXI. DISPONIBILIDADE DO SISTEMA 4

Fonte de estímulo	Máquina onde está alojado o servidor ou a base de dados
Estímulo	Máquina falha
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Tempo que a máquina demorará a ser substituída ou reparada
Medida de resposta	A máquina deve ser substituída idealmente em menos de uma hora, no entanto é tolerável que o tempo de reparação seja de até 24 horas.

TABELA XXXII. PERFORMANCE 1

Fonte de estímulo	*Browser*
Estímulo	*Browser* recebe um pedido e envia nova página ao utilizador
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Tempo de resposta
Medida de resposta	Sistema deve enviar nova página em menos de 1 segundo

TABELA XXXIII. PERFORMANCE 2

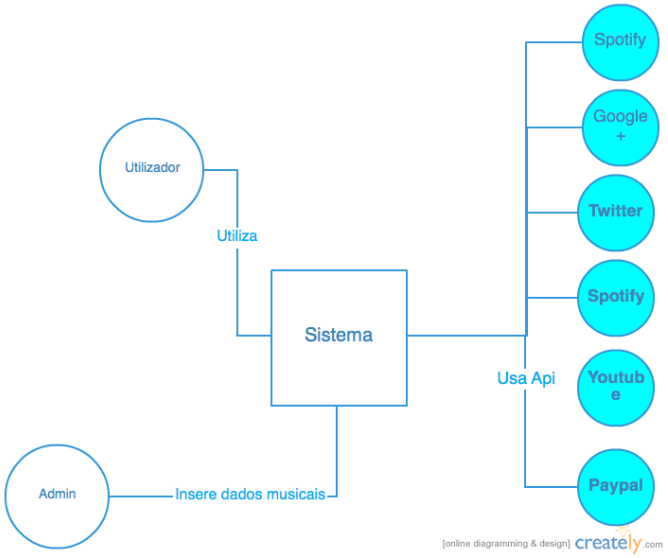
Fonte de estímulo	Utilizador
Estímulo	Utilizador faz pesquisa na base de dados de músicas
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Tempo de resposta
Medida de resposta	Resultados devem ser exibidos em menos de 1.5 segundos

TABELA XXXIV. SISTEMA ESCALÁVEL

Fonte de estímulo	Sistema
Estímulo	Sistema recebe um aumento exponencial de pedidos
Ambiente	Produção
Componente estimulado	Sistema
Resposta	Tempo de resposta a cada um dos pedidos
Medida de resposta	Sistema deve responder em menos de um segundo

D. Vistas

1) Diagrama de Contexto



Este diagrama mostra o contexto do nosso sistema no negócio. O sistema terá ligação a várias entidades externas através das suas Apis.

Será feita a comunicação também com um grupo de APIs externas (através dos servidores web):

- *Facebook*, *Google+* e *Twitter*: será utilizado para fazer autenticação no sistema e partilhar playlists na plataforma;
- *Spotify* e *youtube*: será utilizado para aceder às músicas que o utilizador pretenda (que esteja numa playlist no sistema);
- Paypal: será utilizado para realizar pagamentos.

É responsabilidade do administrador adicionar informação de músicas.

2) Diagrama Fisico

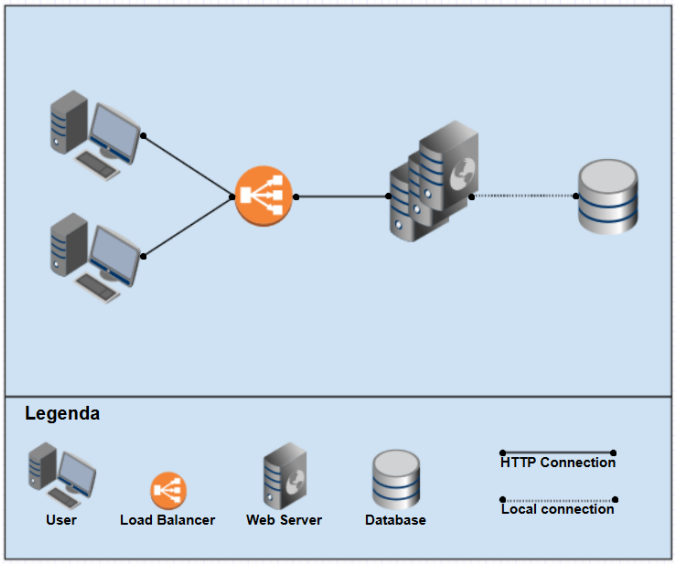


Fig. 1. Diagrama físico

Como foi referido anteriormente, vamos então ter um *Load Balancer* que irá encaminhar os pedidos que estão a chegar para o servidor web que estiver com menor carga, isto é, a processar menos pedidos. É de notar que o *próprio load balancer* irá estar replicado para o caso deste falhar. Depois o servidor web vai comunicar com a base de dados (caso seja necessário) e posteriormente comunica com o *load balancer* para devolver o pedido. De notar que não existe qualquer comunicação entre os servidores web e têm de existir pelo menos dois servidores web para garantir o mínimo de tolerância a falhas.

3) Diagrama de sequência

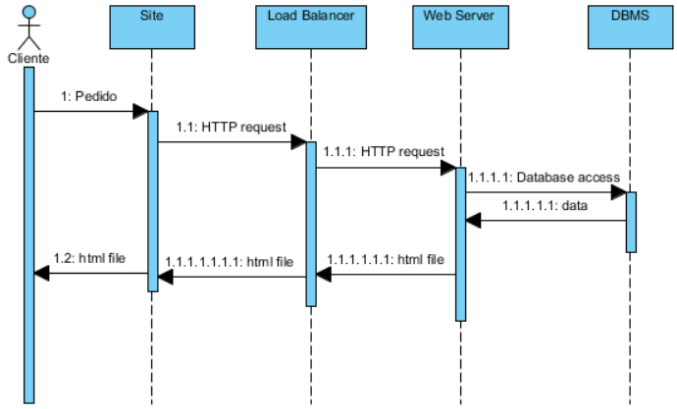


Fig. 2. Diagram de sequência

Este diagrama mostra o fluxo de informação no sistema. Para isso utilizados um cenário genérico em que um utilizador faz um pedido à aplicação. O pedido é transmitido *ao load balancer* que irá redirecionar o pedido ao servidor com menor carga. Caso seja necessário usar informação da base de dados o servidor web ligar-se-á a esta trocando informação. De seguida o servidor web gera uma resposta ao pedido do utilizador no formato de um documento html, este documento é enviado ao *load balancer* que o reencaminhará ao utilizador que fez o pedido.

4) Diagrama de desenvolvimento

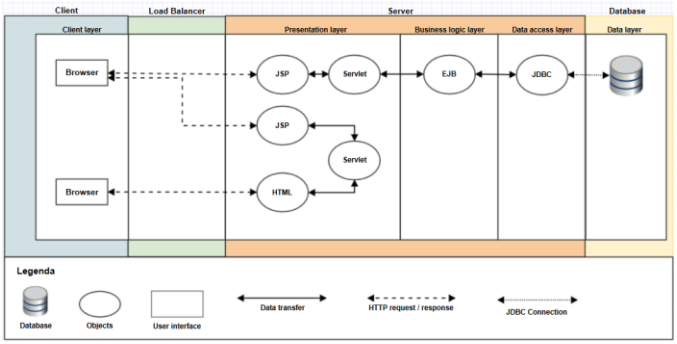


Fig. 3. Diagrama de desenvolvimento

Este diagrama mostra a perspetiva do programador. Nesta vista supomos que o programa será implementado em java, não sendo isto obrigatório nem prescrito pela arquitetura. Podemos ver as 5 camadas principais, a camada do *load balancer*, a *presentation layer*, a *business logic*, a *data acess layer* e da *data layer*. É possível observar o tipo de conexão entre os vários tipos de objetos.

IV. Decisões chave

A. Decisões arquiteturais

1) Bases de dados

- Num estado inicial do projeto estávamos a pensar em utilizar duas bases de dados;
- Será usada uma base de dados SQL clássica garantindo assim consistência e atomicidade, não podendo garantir particionamento. A tolerância a falhas será garantida através de diversos discos ligados através do sistema de RAID-10.

2) Segurança

- As *passwords* não deverão ser gravadas em *plain-text*;
- Deverá ser usado um método baseado em chaves simétricas para cifrar o pedido de login e registo;
- O sistema deverá estar protegido contra ataques de SQL *injection*.
- A segurança do sistema de pagamento pelo *paypal* está garantida pela API do mesmo;
- A disponibilidade e a segurança são muitas vezes atributos de qualidade rivais. No entanto, no nosso caso os procedimentos de segurança implementados não irão comprometer a disponibilidade do sistema.

3) Disponibilidade do sistema

- Para garantir este atributo de qualidade, iremos ter diversos servidores web (como irá ser explicado em detalhe na próxima secção) de preferência em diferentes localidades geográficas para evitar que situações como cortes de eletricidade ou mesmo catástrofes naturais).

4) Interoperalidade

- O tempo de resposta a pedido de login através de contas externas estará sempre sujeito à disponibilidade dos servidores dessas mesmas aplicações.

B. Análise dos atributos de qualidade

1) Sistema facilmente modificável e modular

Como é já sabido, os custos de produção de *software* são bastante elevados. Como tal é importante criar uma arquitetura que minimize ao máximo os custos associados à alteração de componentes de *software*. É nesse sentido que estabelecemos este atributo como prioritário

A arquitetura proposta baseia-se na arquitetura MVC que por si só já modelar e facilmente modificável. A maior modificação que a nossa arquitetura faz é a adição de um *load balancer*. Esta arquitetura permite ter um número arbitrário de servidores e permite ligar e desligar um servidor com custos mínimos. Além disso cada camada é independente podendo alterar cada camada sem afetar as outras evitando adição de novo código.

A nossa arquitetura permite separar as responsabilidades: as camadas *controller* e *view* vão estar alojadas nos vários servidores web. A camada *model* é alojada na base de dados. Além disso um *load balancer* que permite redirecionar os pedidos HTTP. A alteração de cada um destes componentes não implica a modificação de outro desde que sejam respeitadas as interfaces de ligação.

2) Segurança

De acordo com os nossos atributos de qualidade, a segurança não é o atributo mais crítico a nível de prioridades. No entanto, o componente mais crítico é o sistema de pagamentos. Porém toda essa componente é garantida pela API do *paypal*, e esta já garante toda a segurança necessária de modo a realizar o pagamento com sucesso.

Outra componente que exige mais segurança é o sistema de autenticação na aplicação que é garantido pela arquitetura usando uma cifra assimétrica não quebrável, cumprindo assim este requisito.

Por último, o sistema irá estar preparado contra ataques de SQL *injection* através de boas práticas de programação.

3) Disponibilidade do sistema

Para este atributo ser garantido temos de garantir disponibilidade tanto do servidor como da base de dados. A arquitetura prescreve uma DBMS relacional com raid 10, isto é, uma base de dados que garanta consistência e atomicidade. Com isto garantimos elevada disponibilidade para o módulo "Base de dados".

Em relação a camada de servidor a disponibilidade é garantida através de replicação. A arquitetura pede no mínimo dois servidores e um *load balancer* (estando este replicado para o caso de falha), assim a carga será distribuída o melhor possível. Na eventualidade de falha de um servidor web o outro está a fazer *backup*. À medida que o negócio escala é necessário adicionar novos servidores para impedir que estes fiquem congestionados. Além disso aumentará a tolerância a falhas no sistema.

Anteriormente tínhamos pensado que cada servidor web iria ter uma réplica desligada e pronta a ficar ativa em caso de falha. No entanto estaríamos a desperdiçar recursos. É mais vantajoso adicionar essas réplicas ao conjunto de servidores, garantindo também melhor gestão de carga.

No caso de uma falha de *hardware* o sistema irá reagir de forma a garantir a qualidade de serviço graças à repartição de carga. A reparação de um servidor será rápida já que a instalação do servidor de uma nova máquina será automatizada (será só correr o programa).

No caso das falhas mais habituais de base de dados (falha de disco), os dados irão estar replicados através do RAID-10 e como tal o sistema não irá ficar comprometido. Será apenas necessário fazer a reparação o mais cedo possível.

4) Sistema escalável e performance

A arquitetura foi desenhada para escalar *on-demand*. À medida que "o negócio" crescer e se notarem padrões de acesso superiores será só adicionar mais servidores. Obviamente que existem limites, no entanto é suficiente para garantir um bom serviço de pequena/média dimensão.

Para além disso, o *load balancer* irá ajudar no atributo de performance ao repartir a carga pelos servidores. Isto é, a performance não se irá deteriorar com o aumento de carga.

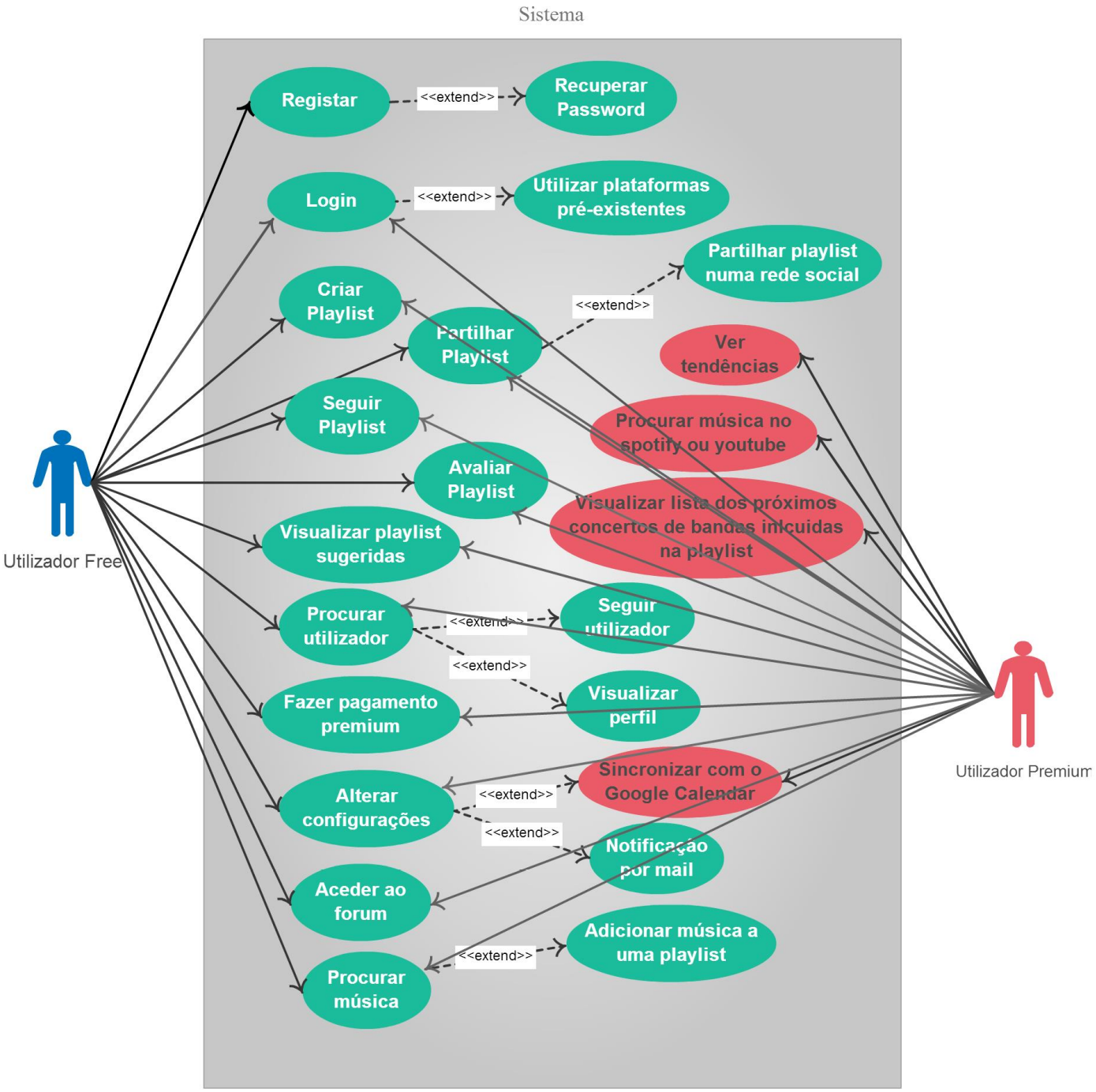
REFERÊNCIAS

[1] Software Architecture in Practice, Third Edition, by Len Bass, Paul Clements, and Rick Kazman. AddisonWesley 2013.

[2] Documenting Software Architectures: Views and Beyond, Second Edition, by Clements, et al.. AddisonWesley 2011.

[3] Design Patterns: Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley, 1988.

ANEXOS (DIAGRAMA DE CASOS DE USO)


Sistema
Registar
<<extend>>
Recuperar Password
Login
<<extend>>
Utilizar plataformas pré-existentes
Partilhar playlist numa rede social
Criar Playlist
<<extend>>
Partilhar Playlist
Ver tendências
Seguir Playlist
Procurar música no spotify ou youtube
Avaliar Playlist
Visualizar lista dos próximos concertos de bandas inlcuidas na playlist
Visualizar playlist sugeridas
Procurar utilizador
<<extend>>
Seguir utilizador
<<extend>>
Fazer pagamento premium
Visualizar perfil
Sincronizar com o Google Calendar
Alterar configurações
<<extend>>
<<extend>>
Notificação por mail
Aceder ao forum
Adicionar música a uma playlist
<<extend>>
Procurar música
Utilizador Free
Utilizador Premium