



Instituto Superior de Engenharia de Lisboa  
Licenciatura em Engenharia Informática e Multimédia

## Sistemas Multimédia para a Internet

Docente: Eng.º Carlos Gonçalves

Semestre de Verão 19/20

### **Trabalho Prático - 3ª Parte**

**Trabalho realizado por:**

Alice Fernandes 45741 LEIM41N Grupo 38

Lisboa, 05 de Junho de 2020



# Índice de matérias

|   |       |
|---|-------|
| Índice de figuras .....                 | - 5 - |
| Índice de Acrónimos .....               | - 6 - |
| 1. Introdução / Objetivos .....         | 7     |
| 1.1. O que é um CMS (ou SGC) .....      | 7     |
| 1.2. Funcionalidades base .....         | 8     |
| 2. Desenvolvimento .....                | 9     |
| 2.1. Wordpress .....                    | 9     |
| 2.2. ZenPhoto .....                     | 10    |
| 2.3. Projeto da cadeira .....           | 11    |
| 2.3.1. Funcionalidades Base .....       | 11    |
| 2.3.2. Funcionalidades opcionais .....  | 14    |
| 2.3.3. Perfis de Utilizadores .....     | 15    |
| 2.3.4. Desenvolvimento do projeto ..... | 16    |
| 3. Conclusão .....                      | 33    |
| Bibliografia .....                      | 37    |



## Índice de figuras

|   |    |
|---|----|
| Figura 1 - Interface de gestão do Wordpress.....                        | 9  |
| Figura 2 - Exemplo de site feito com o ZenPhoto.....                    | 10 |
| Figura 3 - Desenho inicial da base de dados .....                       | 16 |
| Figura 4 - Desenho final da base de dados .....                         | 17 |
| Figura 5 - Estrutura aplicacional do <i>software</i> produzido .....    | 18 |
| Figura 6 - Captura de ecrã da página de configuração .....              | 19 |
| Figura 7 - Página de <i>login</i> .....                                 | 20 |
| Figura 8 - Página de <i>login</i> com mensagem de erro .....            | 21 |
| Figura 9 - Captura de ecrã da página de gestão de utilizadores .....    | 23 |
| Figura 10 - Criação automática de conta.....                            | 24 |
| Figura 11 - Validação de nome de utilizador e endereço eletrónico ..... | 25 |
| Figura 12 - Demonstração da responsividade .....                        | 25 |
| Figura 13 - Gestão de listas de reprodução .....                        | 26 |
| Figura 14 - Demonstração de listas de reprodução criadas .....          | 27 |
| Figura 15 - Painel de criação de episódio .....                         | 28 |
| Figura 16 - Listagem de episódios .....                                 | 29 |

## Índice de Acrónimos

CMS. *Content Managment System*

CTO: Chief Technical Officer, 7

GUI. *Graphical User Interface*

PHP: Hypertext Preprocessor, 20

SEO: Search Engine Optimization, 19

SGC. *Sistema de Gestão de Conteúdos*

SMTP: Simple Mail Transfer Protocol, 22

XSS: Cross-site scripting, 23

## 1. Introdução / Objetivos

Neste trabalho vamos estabelecer o projeto que iremos desenvolver no decorrente da cadeira. Terá de ser um CMS que permita a gestão de sites e dos seus respetivos conteúdos, mas com uma temática livre. No entanto, para chegar ao projeto que irei escrever, é preciso primeiro definir o que é um CMS (Content Managment System) ou SGC (Sistema de Gestão de Conteúdos), dar exemplos de **alguns** CMS já existentes que permitam criar e gerir sites, e, depois apresentar o conceito do meu projeto em particular.

### 1.1. O que é um CMS (ou SGC)

Um CMS é um programa que serve para criar, editar, e (entre outras funcionalidades) gerir o conteúdo de plataformas criadas a partir desse programa, ou seja, é um software que permite criar sites pré-estruturados (a partir de uma estrutura base) onde existem, claro, algumas funcionalidades que são particulares à implementação do site criado pelo CMS, como por exemplo alteração de temas ou extensão de funcionalidades.

A grande vantagem de um site criado por um CMS é que permite o desenvolvimento sem ter de ser desenvolvido código de raiz, sendo que a extensão de funcionalidades ou temas customizados são opcionais. Estes sites vêm já com um conjunto de funcionalidades e GUI (Graphical User Interface) que são "dadas" pelo próprio CMS.

Uma outra vantagem é que o conteúdo do site não fica "preso" ao site em si. Este conteúdo é gerido através do CMS e pode até mesmo ser migrado entre sites, ou seja, a gestão deste conteúdo fica no encargo do CMS e não do site em si.

Este tipo de software permite então que, empresas ou particulares, consigam ter um site, uma presença online, sem terem de ter recursos especializados (programadores de software) e sem terem que alocar um orçamento grande no desenvolvimento de um site de raiz.

## **1.2. Funcionalidades base**

Então podemos definir algumas funcionalidades básicas que um CMS tem de ter:

- Tem de ser personalizável, visto que cada site criado terá, quase que, como uma identidade própria.
- Tem de ser fácil de usar, porque uma característica de um CMS é a gestão de conteúdo multimédia
- No entanto, também têm de ser extensível, porque se cada site terá a sua própria identidade, então o mais certo é que também terá funcionalidades particulares

Alguns exemplos de CMS gratuitos bem conhecidos [2]:

- Wordpress
- Joomla
- Drupal

No próximo capítulo iremos discutir alguns exemplos de CMS, e também como fizemos a implementação do nosso próprio projeto. Iremos também demonstrar as opções que tomámos e discutir implementações alternativas quando relevante.



## 2. Desenvolvimento

Agora que já sabemos o que é um CMS, podemos então olhar para alguns exemplos concretos de CMS já existentes. Para scope deste projeto, vou-me focar nas capacidades multimédia que estes CMS, mesmo que, referindo na generalidade as funcionalidades que têm.

Para base de comparação, irei estabelecer alguns pontos de comparação que os CMS que vou analisar podem ter ou não ter tirado alguma complexidade do CMS em si, e focando-me no essencial e com vista ao que vou implementar. Apresento então os CMS

### 2.1. Wordpress

Como já referido acima, Wordpress é um dos CMS mais usados, e é também um dos mais versáteis. Este CMS permite uma personalização tão ao detalhe, ao ponto de servir apenas como *back office* (como no caso do Observador). No entanto, a sua versão mais básica já permite desenvolver blogs com temas personalizados. É um dos CMS com comunidades mais ativas, e onde existe um repositório de plugins que podem ser adicionados à instalação de WordPress. Números de 2020 indicam que este CMS serve de para 35% da internet [2], sendo o CMS mais usado.

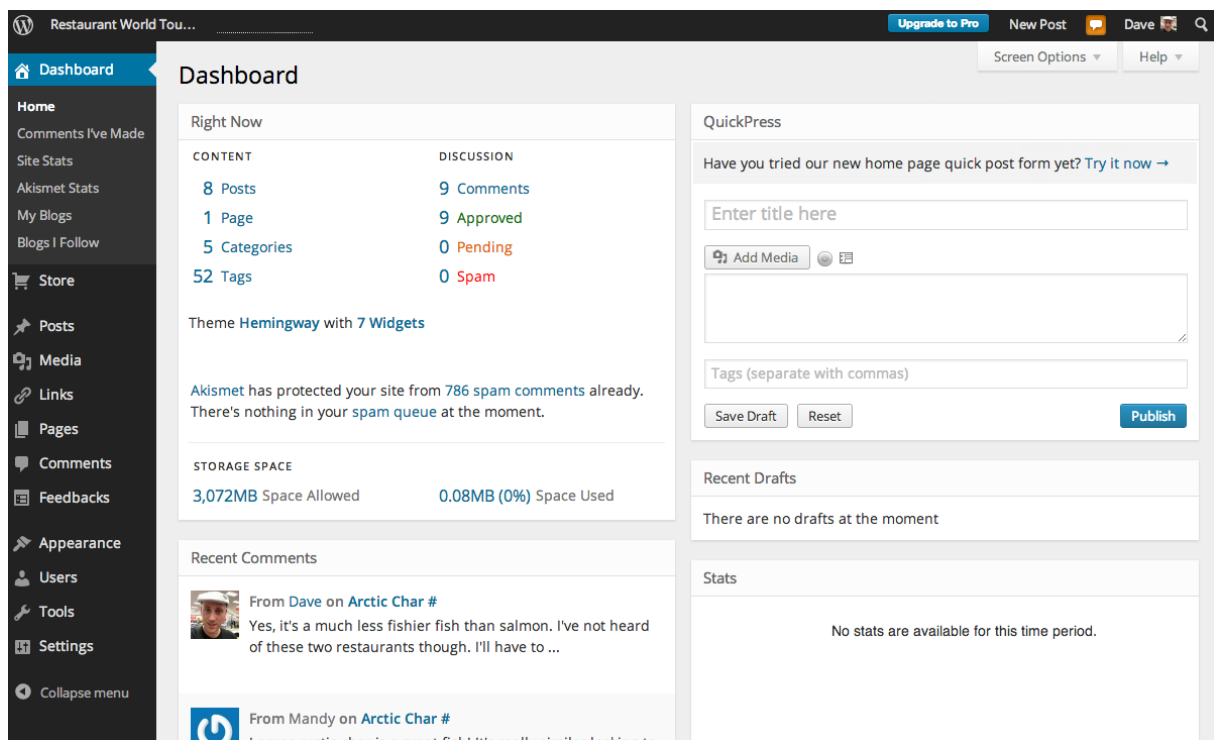


Figura 1 - Interface de gestão do Wordpress

A nível de funcionalidades permite ter galerias de imagem, campos de dados customizados, temas customizados, perfis de utilizadores, comentários, entre outras, portanto diria que qualquer tipo de site é realizável com este CMS, ou seja, a ideia de implementação, que irei

desenvolver mais abaixo, era perfeitamente concretizável (ainda que, era necessário algum desenvolvimento adicional), e foi por essa razão que decidi analisar este CMS, por ser muito "abstrato".

## 2.2. ZenPhoto

Já mais focado em conteúdo temos o ZenPhoto, que é um CMS que permite criar sites de fotos galerias, o que já é mais parecido ao que irei desenvolver. Exemplos de utilização deste podem ser portfólios fotográficos de nome próprio, portfólios gráficos (de *designers* por exemplo), sites de clubes de DJ, ou até mesmo blogs simples.

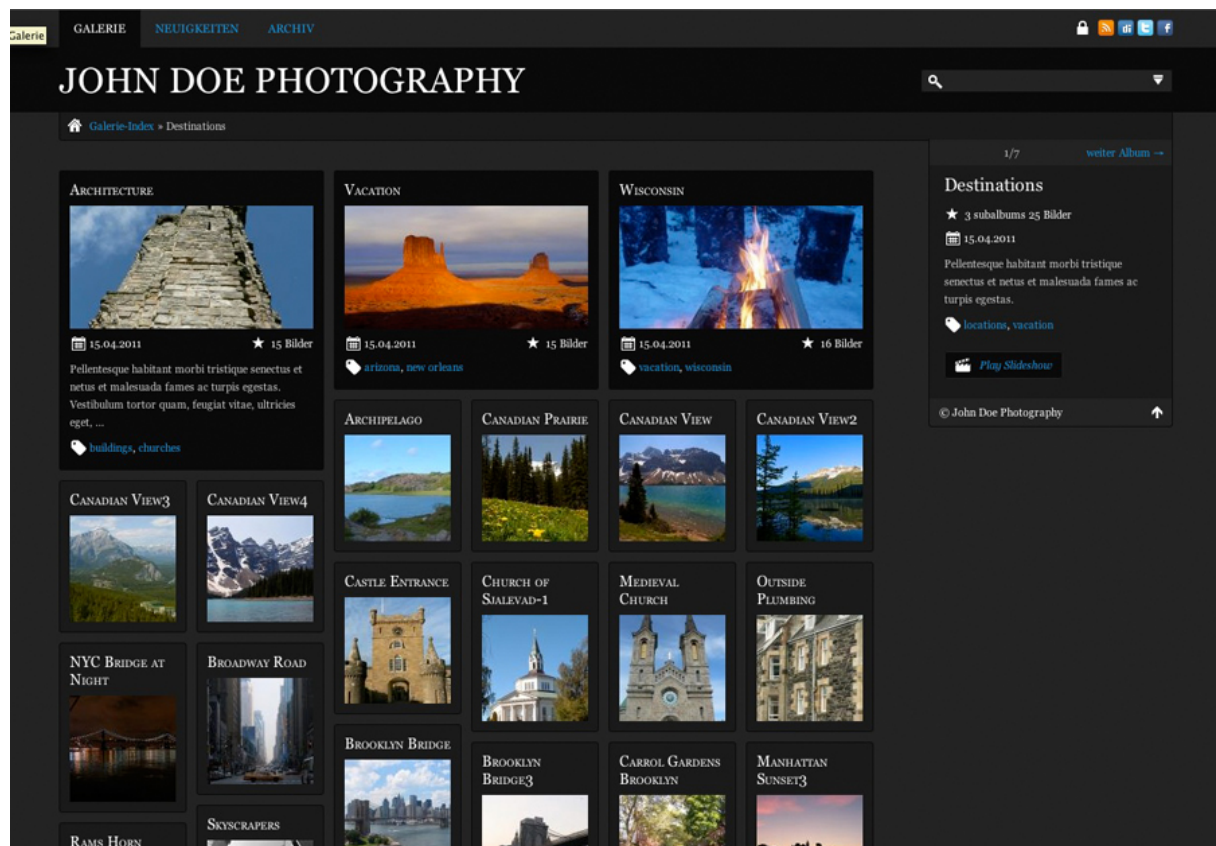


Figura 2 - Exemplo de site feito com o ZenPhoto

Em termos de números, não consegui encontrar o número de instalações que utilizam o CMS. Este CMS é mais focado em sites de galerias de imagens, e, por isso oferece um conjunto de funcionalidades mais direcionado a esse tipo de conteúdo, que incluem Marcas de Água automáticas, galerias de imagens, comentários, permite também ter conteúdos de áudio, perfis de utilizadores, e também é extensível por via de plugins. É escrito em PHP

## 2.3. Projeto da cadeira

O projeto que vou desenvolver vai ser um CMS que possibilite a gestão de sites que permitam a publicação de podcasts, tal como o Wordpress permite a gestão de blogs, e o ZenPhoto permite a gestão de sites, exceto com conteúdo mais focado em fotos galerias. À semelhança do segundo CMS que analisei, vai ter funcionalidades mais focadas no conteúdo que quero trabalhar, este caso, áudio com a duração de pelo menos 5 minutos. O conteúdo base vai ser um Episódio, e um conjunto de episódios vão estar organizados em Programas, e opcionalmente, Playlists. Vou então descrever com algum detalhe o que pretendo implementar:

### 2.3.1. Funcionalidades Base

A nível de *front end* estarei à procura de uma implementação de uma *Single-Page Application*, para que o áudio não seja interrompido entre navegações de página, com carregamento assíncrono de determinado conteúdo. Relativamente à organização do *back office*, poderá ser algo do género:

#### Episódio:

Um episódio é um áudio com uma duração superior a 5 minutos, de formato MP3, WAV, OPUS ou OGG. Um episódio vai estar obrigatoriamente associado a um programa, e pode ser criado por qualquer perfil com capacidades de Editor

Este episódio contém:

- Título
- Descrição *rich text*
- Imagem de capa. Se não tiver imagem de capa, é usada a imagem do programa
- Poderão ter zero ou mais convidados, no caso de haver um convidado a falar no podcast
- Ficheiro de áudio (óbvio)
- Créditos, textuais apenas, sem associação de perfil.
- Haver "Gostos"
- Partilhas para as redes sociais
- Sistema de #hashtags que pode ser usado para pesquisas
- Comentários

Este episódio pode ser adicionado a uma playlist, por via de interações pelo front end, ou então por via de back office, e os uploads irão para um Amazon S3, isto porque estes ficheiros irão ocupar algum espaço em disco, e, portanto, para manter a instalação a ocupar pouco espaço, o upload é feito para um S3.

## Playlists

Playlists são uma coleção de episódios de diferentes programas. Vão ter um:

- Título,
- Uma descrição não *rich text*
- Imagem de capa
- Utilizador que edita a playlist (Pode ser um Utilizador Simples ou Editor)
- Episódios (com referência ao programa onde estão inseridos)
- Haver "Gostos"
- Partilhas para as redes sociais
- Sistema de #hashtags que pode ser usado para pesquisas
- Sistema de "Seguir", que envia para o email notificações quando forem adicionados novos episódios
- Este "seguir" pode ser não autenticado, significa que não é necessário haver conta para subscrever a programas
- Comentários
- Visibilidade (Pode ser pública, privada, invisível)
- Tipo (pode ser um programa ("*program*"), uma playlist automática ("*category*"), ou playlist normal ("*playlist*"))

## Categoria

Um programa pode ter várias categorias, e estas categorias permitem agrupar programas que estão dentro do mesmo gênero. São quase como playlists automáticas obrigatoriamente públicas, do tipo "category", sem editor. Vão ter:

- Título,
- uma descrição não *rich text*
- Imagem de capa
- Gostos

- Programas, e consequentemente, episódios
- Sistema de "Seguir", que envia para o email notificações quando forem adicionados novos episódios. Este "seguir" é não autenticado, significa que não é necessário haver conta para subscrever a programas

### **Programa**

Um Programa é uma coleção de episódios referentes ao mesmo "programa", como episódios de uma série. É uma extensão de uma playlist obrigatoriamente pública (do tipo "program"), exceto que agrupa um conjunto de episódios, que pertencem ao mesmo programa. Vão ter um:

- Título,
- uma descrição não rich text
- Imagem de capa
- Editores que editam este programa, ainda que essa informação não seja apresentada no front end
- Episódios
- Haver "Gostos"
- Partilhas para as redes sociais
- Sistema de #hashtags que pode ser usado para pesquisas
- Sistema de "Seguir", que envia para o email notificações quando forem adicionados novos episódios. Este "seguir" é não autenticado, significa que não é necessário haver conta para subscrever a programas
- Comentários
- Uma ou mais categorias.

#### **2.3.2. Comentário**

Um comentário é um objeto que vai estar associado a um episodio ou *playlist*. Vai conter:

- Nome do comentador
- Email de comentador
- Texto *rich text* sem suporte a imagens e links, mas com suporte a emoji
- Data (tirada automaticamente)
- Respostas (Comentários)
- Imagem (tirada do *gravatar*)

Comentários podem ter comentários como respostas, e pode ser removido durante 5 minutos após a publicação. Se um comentário com respostas for removido, o comentário perde referencia ao utilizador (Passa a ser um comentário cujo utilizador é "Utilizador Removido"), e o texto é trocado por Comentário removido pelo utilizador

### 2.3.2. Funcionalidades opcionais

Numa segunda-fase (diretamente dependente do tempo disponível) poderá haver:

Partilhas nativas para web, isto é, disparar os diálogos de partilha nativa dos dispositivos mobile através do browser. É possível através de uma API, a Web Share API (para episódios, programas e playlists), comentários em (quase) tempo real, com notificação por via de web sockets, playlist de "Ver Mais tarde", mantida em *frontend* com recurso a cookies ou semelhante, e com um botão específico para adicionar a esta lista, implementação da *Media Session API* para permitir o controlo de áudio pelo utilizador através do navegador *Web*, Partilhas com *timestamp* e implementação de serviço de *embeds* (/embed/) para incorporação de conteúdo noutros sites. Podem também ir com *timestamp*

Especificamente para comentários poderá ser desenvolvido a funcionalidade de *upvotes*, e depois destacar os 3/5 mais *upvoted*, e o para os restantes ordenar por data de publicação.

Para episódios poderá haver legendas *WebVTT*, isto por questões de acessibilidade, nos casos de serem podcasts "conversacionais" haver legendas., marcadores temporais, isto é por exemplo, no caso de mixes de música, haver a informação da música que está a tocar a determinado minuto, e haver um marcador visual de quando a música começa, que ao clicar, leva diretamente para aquele minuto. Estes marcadores irão conter o nome da faixa, nome do Autor, referencias para *spotify*, *deezer* e outras plataformas (quando disponível) e imagem de capa. Outras funcionalidades ainda relativas aos episódios que poderão ser implementadas são a importação de meta dados ID3 do ficheiro (para episódios), comentários ao minuto, isto é, comentar ao minuto 3:16., testar anúncios em áudio (com VAST <https://support.google.com/displayvideo/answer/7670534?hl=en>)

Finalmente, para programas e *playlists* poderá ser implementadas *push notifications* em dispositivos *mobile* para o "Seguir", através do *OneSignal*, que é um serviço que permite mandar notificações para dispositivos, e que disponibiliza uma API que recebe pedidos, e que gere essas mesmas notificações.

### 2.3.3. Perfis de Utilizadores

Visto que este projeto poderá ter algumas nuances no que diz respeito a funcionalidades, proponho então os seguintes perfis de utilizadores:

- Administrador de Site

Este utilizador tem controlo total sobre o site, e também sobre todo o conteúdo publicado.

- Editor-Chefe

Este perfil tem permissões de edição avançada, como criar e editar programas, perfis de convidados, playlists e episódios

- Editor

Este perfil tem permissões de edição básicas, e como tal só pode criar e editar episódios e playlists

- Convidado

Este perfil é meramente informativo, serve apenas para sinalizar determinados convidados que possam participar nos podcasts. Este perfil vai ter uma funcionalidade que permite ver uma listagem de episódios em que este convidado participou. Esta listagem será uma playlist automática. Este perfil não permite o login, é apenas informativo.

- Anónimo

Este perfil não necessita de registo, e representa todos os utilizadores não autenticados ou sem conta. Só têm duas interações possível, que serão criar comentários e seguir playlists.

Se for pertinente, poderá também haver um perfil de Utilizador Simples, que permita por exemplo ter uma conta, e ter também algumas funcionalidades extra, como listas de reprodução, episódios gostados.

### 2.3.4 Desenvolvimento do projeto

Para o desenvolvimento está a ser usada a linguagem PHP, com recurso a uma base de dados em MySQL. Para acelerar o desenvolvimento do CMS, recorreremos à utilização de bibliotecas externas, que resolvem uma série de temáticas relacionadas a aspetos particulares do projeto proposto, como por exemplo, a biblioteca PHPMailer que já trata o envio de emails. Cheguei a considerar Laravel ou Lumem, ambas bibliotecas muito potentes, mas devido ao tempo, decidimos largar porque o custo de aprender uma nova *framework* não compensa em comparação ao curto tempo em que tivemos para desenvolver estas funcionalidades.

Em termos de escrever código, estou a ir por uma abordagem mais orientada a objetos, porque acho que é uma maneira mais organizada e sistemática de escrever código.

### Base de dados

A estrutura inicial da base de dados é a que está exemplificada acima. Esta base de dados vai suportar todos os dados produzidos pelos utilizadores durante a utilização do site. Vai precisar de algumas alterações à medida que o CMS vai ser desenvolvido.

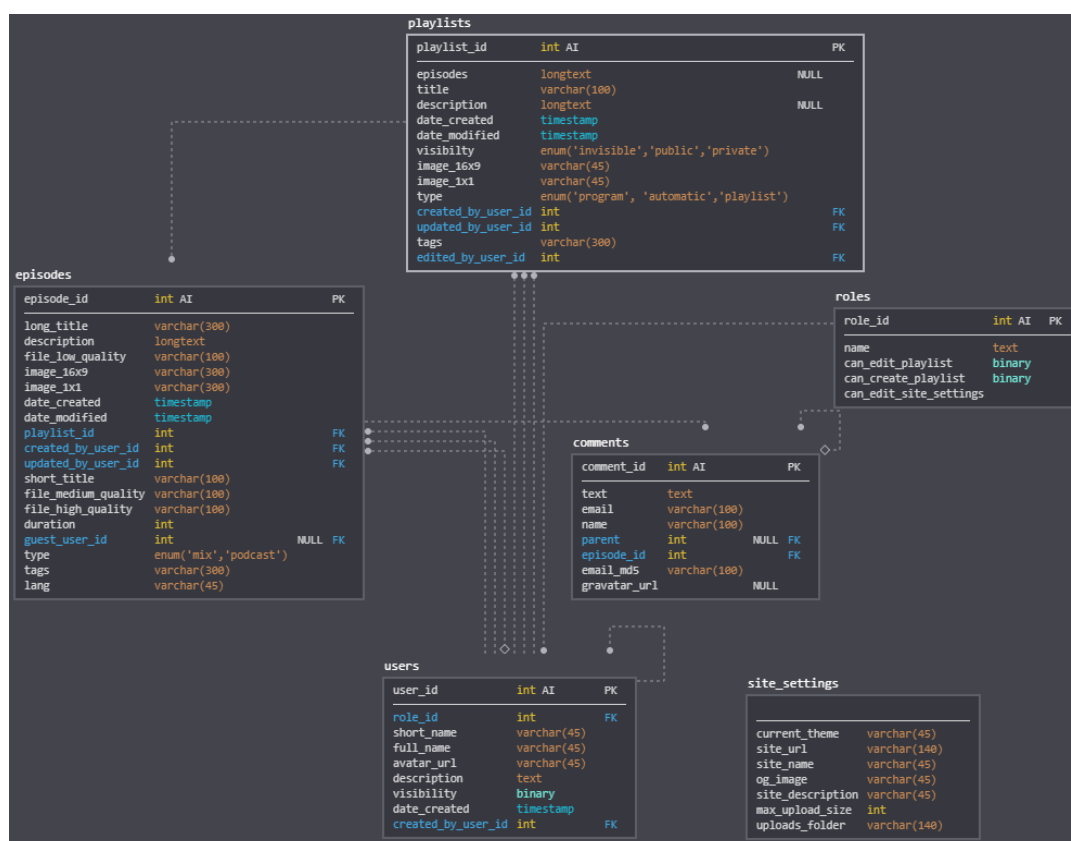
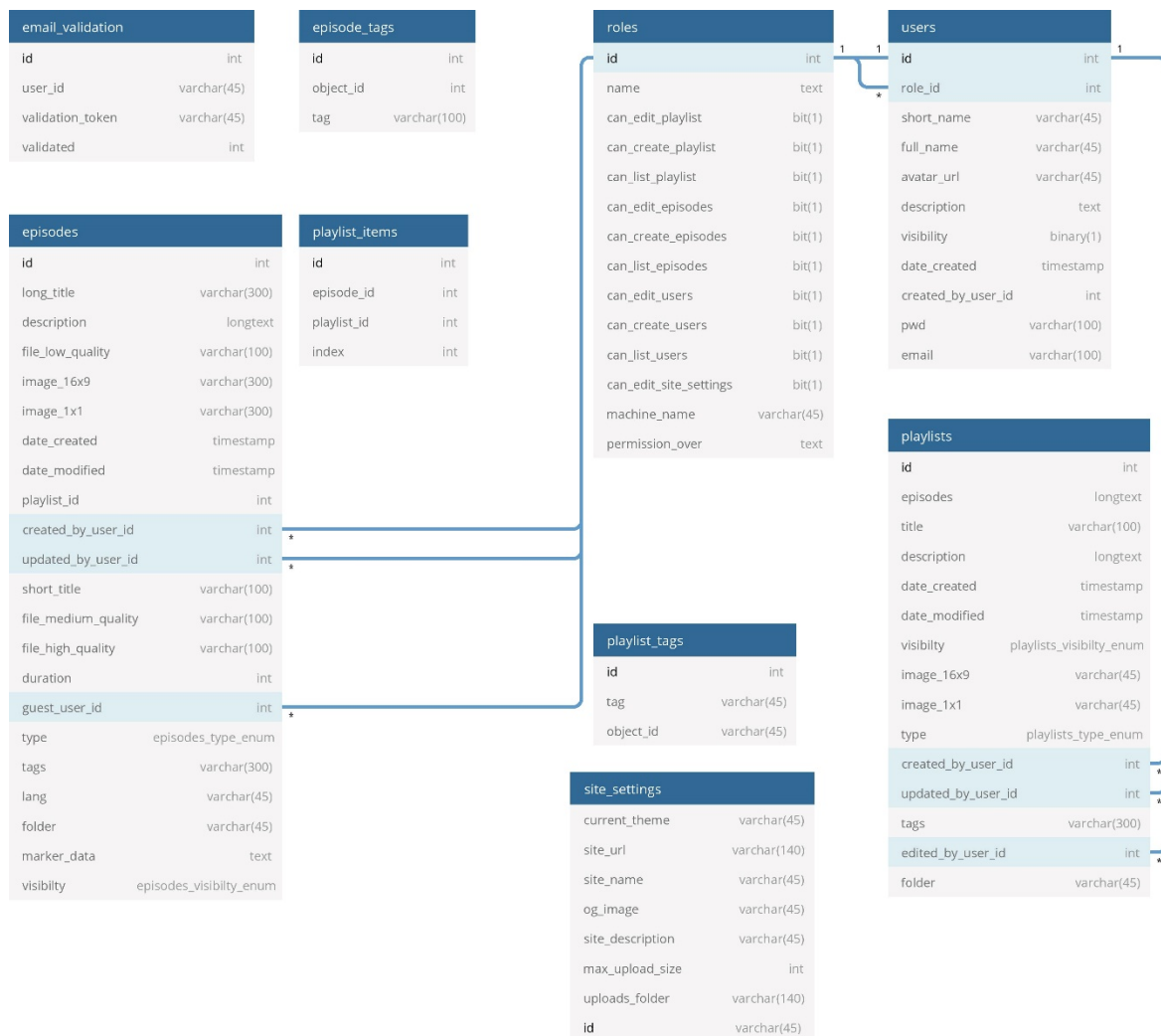


Figura 3 - Desenho inicial da base de dados



O desenho inicial da base de dados não contempla todos o tipo de dados que serão necessários para popular as páginas públicas do CMS, e por isso, foram criadas mais tabelas para fazer a ligação entre os dados, e para ser possível fazer pesquisas mais eficientes à base de dados



**Figura 4 - Desenho final da base de dados**

## Infraestrutura

Este projeto foi escrito inteiramente nas linguagens PHP e *Javascript*. Enquanto que o código Javascript corre no navegador *web*, para correremos código PHP precisamos de ter configurações específicas no lado do servidor. Por isso, durante o desenvolvimento, configuramos um servidor Apache com os módulos de MySQL e PHP. O serviço MySQL é uma linguagem que permite manipular bases de dados relacionais, isto é, com identificadores e referencias entre tabelas, e dependendo da máquina, no porto 3306 ou 3307. Existem também bases de dados não relacionais, as chamadas base de dados NoSQL, como MongoDB ou Redis, que permitem que os registos não obedecem a uma estrutura pré-determinada, como tabelas, e que os dados guardados sejam mais flexíveis. O serviço de PHP permite executar *scripts* desta linguagem no lado do servidor, e também permite manipular bases de dados, através de um *plugin* de base de dados que está disponível na linguagem. Em alternativa ao PHP podíamos escrever em Java ou até mesmo Golang, que ambos também são executadas do lado do servidor.

Como o tema deste projeto é áudio de longa duração, optámos por armazenar os ficheiros de áudio num serviço externo, o Amazon S3. Este serviço permite ter contentores, também chamados de *buckets*, que podem guardar qualquer tipo de dados. O mapeamento é feito através de chaves, onde determinado conjunto de dados está associado a uma chave. No nosso caso em concreto estamos a falar de ficheiros de grande volume de dados, que podem pesar mais do que 100 *Megabytes*, e ter isso no lado do servidor torna este projeto bastante pesado.

Já as imagens e o restante conteúdo é guardado no armazenamento do servidor. Podemos então simplificar este *setup* no seguinte diagrama.

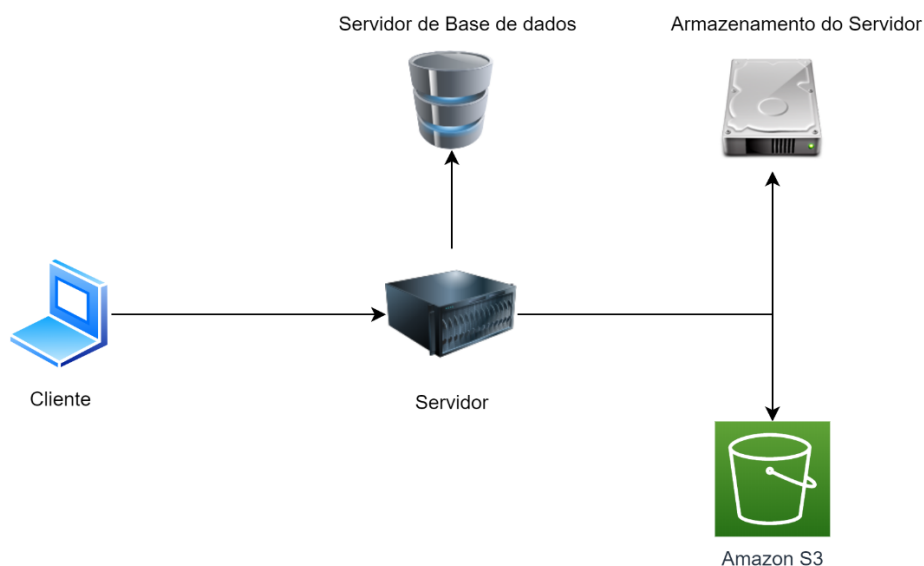


Figura 5 - Estrutura aplicacional do *software* produzido

## Configuração Inicial

Para o configurar o ambiente do CMS no primeiro início, escrevi uma pequena página que vai ajudar o utilizador a configurar a base de dados, tabela e utilizador. O configurador procura pela existência do ficheiro *.htconfig.php*, e este ficheiro define algumas constantes que são utilizadas nos outros ficheiros.

O utilizador define o nome da tabela, servidor de base de dados e *password* da conta. Após a execução do script é criada a base de dados, criação das tabelas que vão suportar os dados.

Com este utilizador criado, o utilizador pode começar a gerir o site. Em termos de navegação, a transição entre páginas é feita através de parâmetros *GET* que estão nas ligações, que leva um parâmetro *p* que representa a página que se quer aceder, e um parâmetro *a* que representa a ação que se quer fazer nessa página.

Para o *front-office* estou a ponderar a utilização de *rewrite rules* do Apache, para ter links mais práticos para SEO e também para serem mais facilmente memorizáveis.

De notar que os ficheiros começados por *.ht* não são servidos pelo Apache, o que significa que não são acessíveis pelos utilizadores. Esta configuração inicial acabou por ficar mais simplificada, para permitir o *setup* mais rápido por quem instala este CMS no servidor



Database Settings

This script handles the creation of the table

Database name:

User name:

User password:

Server:

Server port:

Administrator User settings

Username:

email:

Password:

Repeat password:

Figura 6 - Captura de ecrã da página de configuração

## Autenticação e sessão

No momento da criação de utilizadores, as *passwords* inseridas pelos utilizadores não são guardadas em *plain text*. Ao invés, estes valores foram transformados em *hashes*, através da utilização do método disponibilizado pela linguagem PHP `password_hash`, que por omissão utiliza o algoritmo *bcrypt*. A documentação de PHP não recomenda a utilização do algoritmo MD5 [4] para fazer hash às passwords, devido a já ser facilmente decifradas as palavras geradas

Tendo este valor guardado permite identificar e autenticar utilizadores perante o nosso sistema, e na página de *login*, o utilizador introduz o seu nome de utilizador e password, e com esse nome de utilizador vamos buscar à base de dados a *password* e passamos ambos valores (o valor introduzido pelo utilizador e o valor retirado da base de dados) pela função `password_verify`, que retorna um valor booleano. Após o *login* com sucesso é estabelecida uma sessão PHP com os dados do utilizador autenticado. O *logout* é apenas apagar a sessão que está estabelecida.

Na teoria, é possível escrever sistemas de autenticação utilizado a *Basic Auth* ou *Digest Auth* que funcionam mais ao nível dos *headers* dos pedidos, mas também trazem uma complexidade de implementação. O sistema à base de sessões também tem falhas, mas a implementação é relativamente simples. O ideal será migrar este sistema para *Digest Auth*, uma vez que os navegadores de *web* já sabem reconhecer os *headers* de conteúdos protegidos.

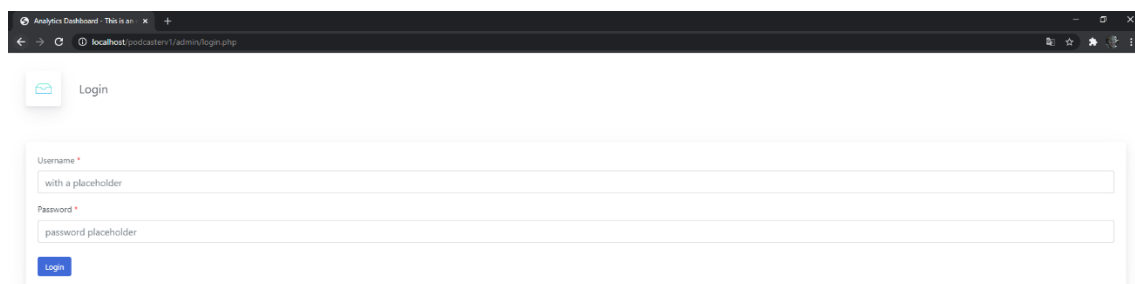
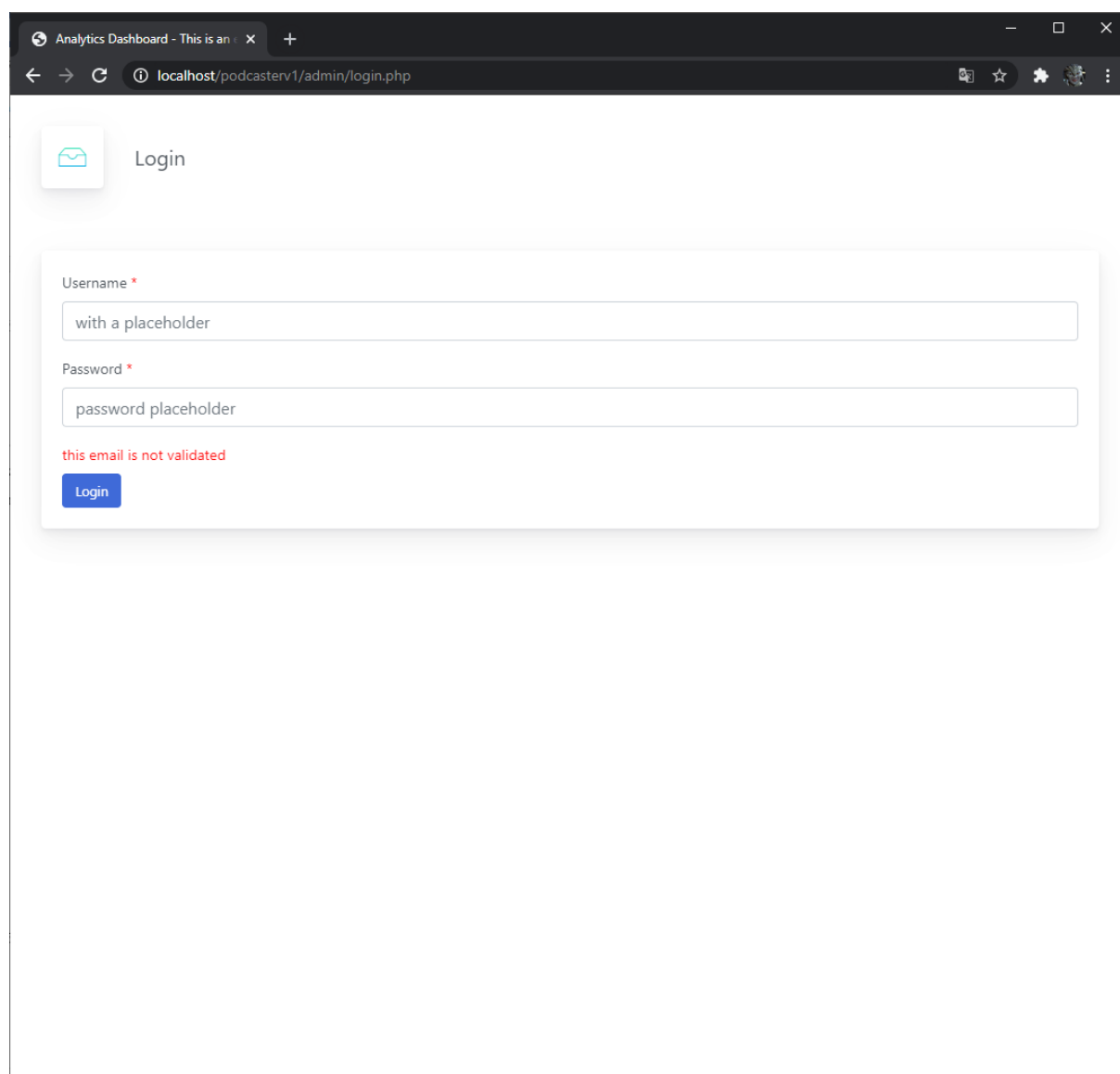


Figura 7 - Página de *login*

Por razões de segurança, só é possível autenticar utilizadores com contas válidas, aparecendo uma mensagem de erro quando essa condição não é verdadeira



**Figura 8 - Página de *login* com mensagem de erro**

## Gestão de utilizadores

Com as tabelas de utilizadores e *roles* criadas é possível começar a criar, apagar e editar utilizadores. Esta funcionalidade está disponível apenas para utilizadores que sejam administradores ou editores-chefe. Após a criação da conta do utilizador, não é permitido trocar o nome de utilizador nem o endereço de correio eletrónico. No entanto, o utilizador pode alterar a sua foto de perfil e restantes dados na sua página de edição de perfil, que está disponível no painel de administração. Inicialmente utilizamos o serviço Gravatar para gerir as fotos de perfil, mas depois optámos por ter essa funcionalidade incorporada no nosso próprio *software*.

Para criar a conta é necessário preencher um *captcha*, para validar se o utilizador não é o *robot*, e o endereço de email terá de ser validado após a criação da conta, portanto o utilizador recebe um *email* no endereço registado, e a conta fica inutilizável enquanto não for validada. A única exceção é a primeira conta de administrador, que não precisa de ser validada. Esta exceção existe para não adicionar mais passos de verificação na instalação do CMS. Ao utilizadores é também atribuído um conjunto de permissões, que definem se pode ou não ver conteúdos privados, editar episódios, editar listas de reprodução e editar utilizadores. Isto depois traduz-se no *frontend* em que a visibilidade determina se determinado bloco de conteúdo é (ou não) renderizado no cliente.

Para gerar este desafio recorreremos a uma biblioteca externa, que trata da lógica de gerar as imagens e as frases. Estes dados ficam guardados na sessão para permitir a validação do desafio em *refresh*. Para o envio dos *emails* utilizamos uma biblioteca chamada PHPMailer e utilizamos o Gmail como *mail broker*. Esta biblioteca trata da lógica de autenticação por SMTP, e também trata do envio dito destes *emails*.

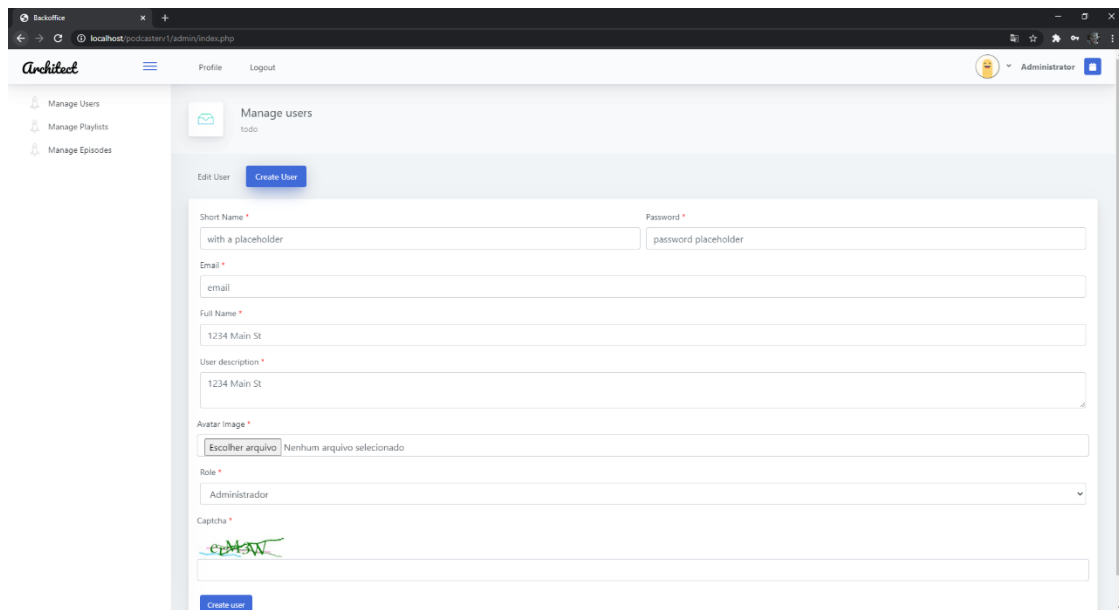


Figura 9 - Captura de ecrã da página de gestão de utilizadores

Este formulário é validado tanto em *frontend* como em *backend*. No lado do PHP estamos a utilizar as funções *filter\_var* com diversos filtros para prevenir ataques *sql injection*, e ataques de XSS, em que se pode introduzir código *javascript* que entra diretamente para a base de dados, e depois nas páginas públicas é executado, e pode causar danos como ler dados dos utilizador e até mesmo manipulá-los.

É também possível que um utilizador se registe no site automaticamente, através da página de registo. Estes utilizadores são autores convidados por omissão, e este *role* só pode ser trocado na página de gestão de utilizadores, por um utilizador com permissões de edição de utilizadores

Analytics Dashboard - This is an x +

localhost/podcasterv1/admin/register.php

### Register Account

todo

Short Name \* Password \*

with a placeholder password placeholder

Email \*

email

Full Name \*

1234 Main St

User description

1234 Main St

Avatar Image \*

Escolher arquivo Nenhum arquivo selecionado

Captcha \*

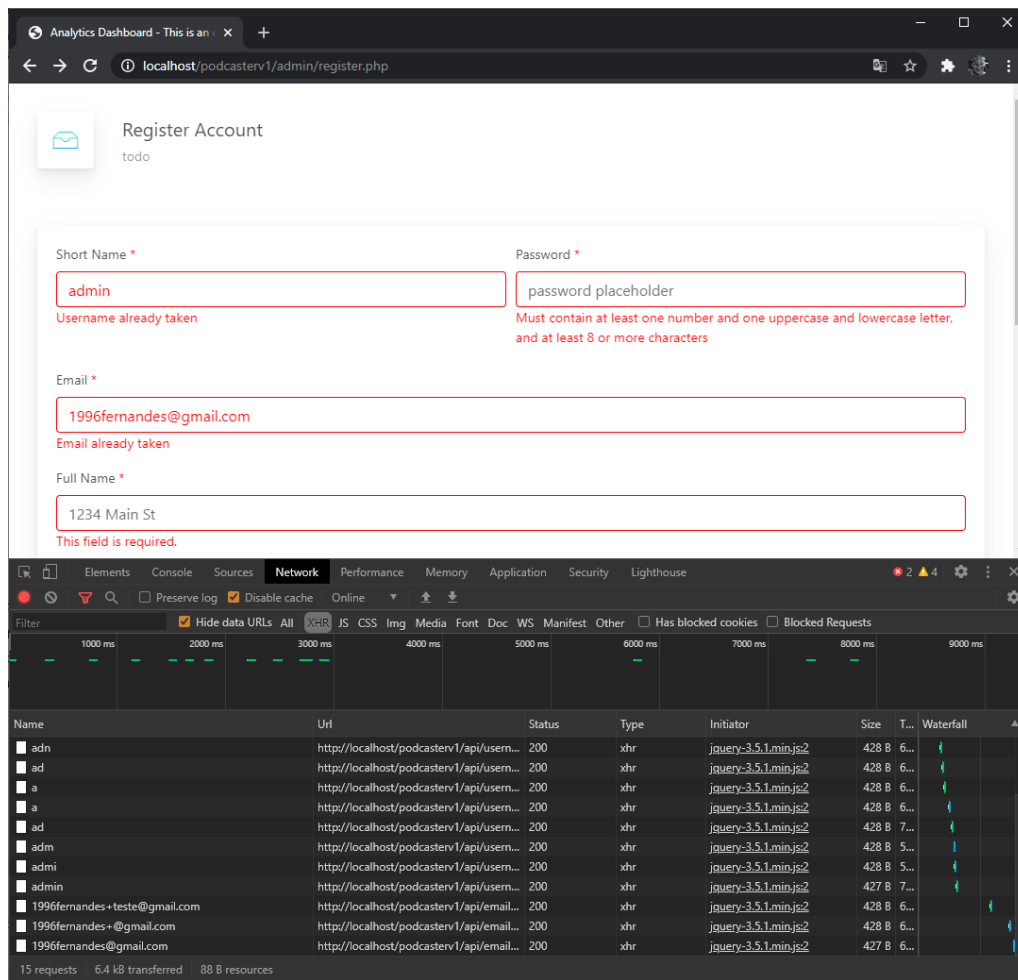
79p1ix

Create user

**Figura 10 - Criação automática de conta**

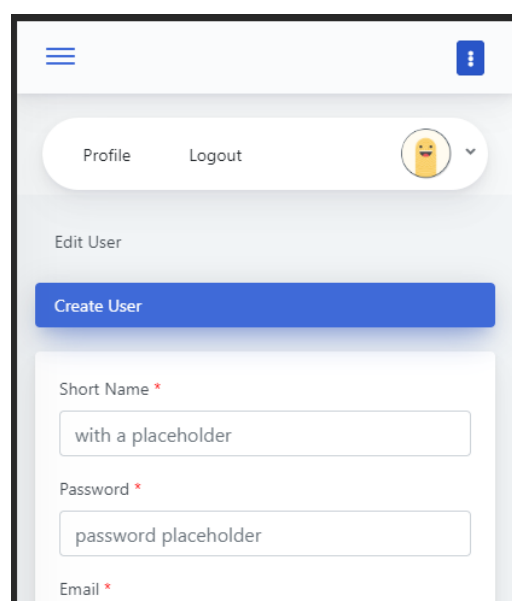
Este formulário, como todos os restantes formulários, é validado em ambos os lados, e também foram escritos dois pequenos serviços para validar endereços de email e nomes de utilizador, para que não seja possível criar uma conta com nomes ou emails já existentes em base de dados. Estes serviços vão ler diretamente à base de dados, e fazem parte das validações, e também são validados em *backend* para evitar ataques maliciosos ao servidor, ficando o formulário desativado quando uma das condições não é verdadeira. Estas validações também estão presentes na página de gestão de utilizadores. A cada *input* do utilizador nos campos de nome de utilizador e *email* é feito um pedido com o valor do campo ao *endpoint* correspondente, e esse *endpoint* retorna uma resposta booleana que indica se valor já está a ser utilizado ou não. Estes pedidos são feitos por via de AJAX ao próprio servidor, que depois tem lógica de validação com descrito acima.





**Figura 11 - Validação de nome de utilizador e endereço eletrónico**

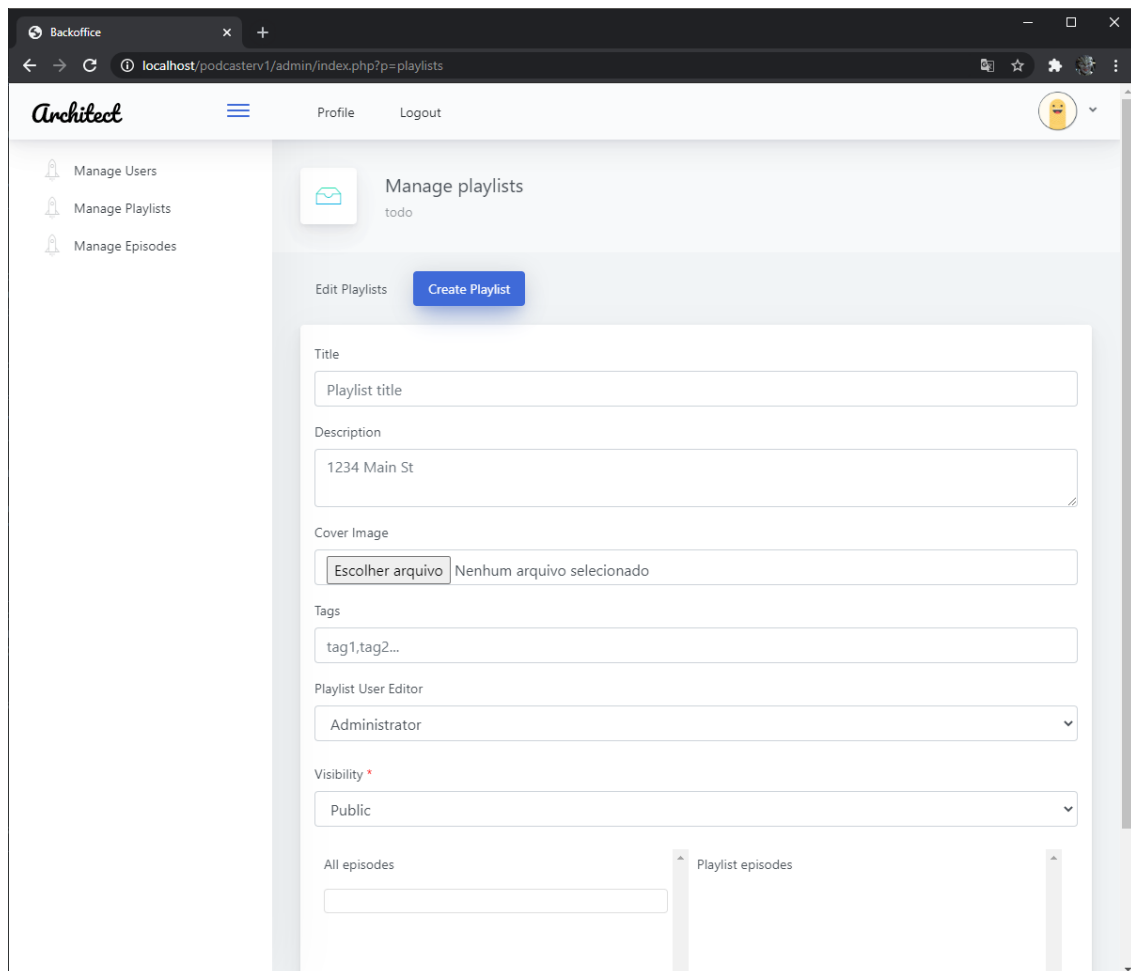
Gostaríamos também de demonstrar que este painel de gestão é responsivo, sendo que foi escrito com recurso à biblioteca *bootstrap* e *jQuery*



**Figura 12 - Demonstração da responsividade**

## Gestão de Listas de reprodução

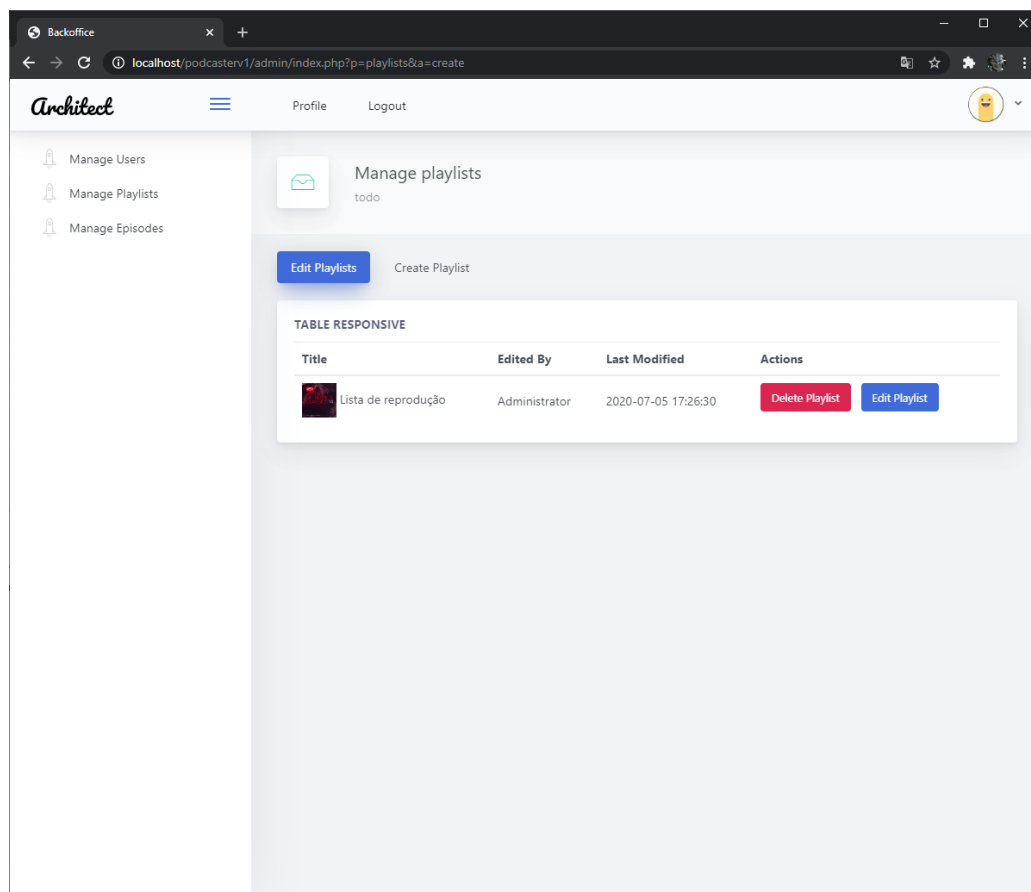
Cada episódio está obrigatoriamente associado a uma lista de reprodução, e por isso, é necessário criar uma lista antes de se criar um episódio. Esta lista tem alguns atributos, como imagem, descrição e visibilidade, que determinam como é apresentada ao cliente.



The screenshot shows a web browser window with the URL `localhost/podcasterv1/admin/index.php?p=playlists`. The page is titled 'Architect' and has a sidebar with links: 'Manage Users', 'Manage Playlists', and 'Manage Episodes'. The main content area is titled 'Manage playlists' and includes a 'Create Playlist' button. Below this, there is a form with the following fields: 'Title' (with placeholder 'Playlist title'), 'Description' (with placeholder '1234 Main St'), 'Cover Image' (with a button 'Escolher arquivo' and text 'Nenhum arquivo selecionado'), 'Tags' (with placeholder 'tag1,tag2...'), 'Playlist User Editor' (a dropdown menu with 'Administrator' selected), 'Visibility' (a dropdown menu with 'Public' selected), and two empty input fields at the bottom labeled 'All episodes' and 'Playlist episodes'.

**Figura 13 - Gestão de listas de reprodução**

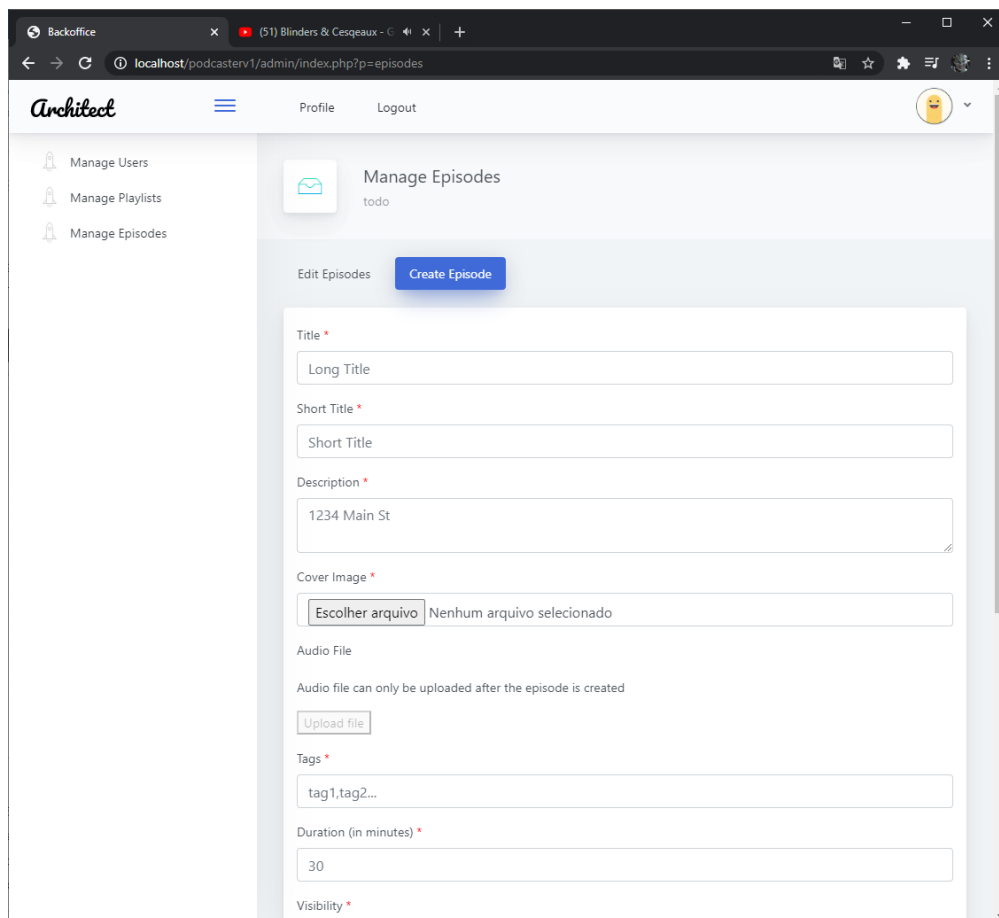
Após a criação da lista, é possível criar episódios e adicioná-los a listas de reprodução. Este elemento é o mais simples deste CMS, uma vez que serve apenas para agrupar episódios num único sítio, e têm poucas coisas que lhe tornam mais personalizáveis. Estas listas, como todos os restantes elementos deste CMS, são editáveis, e essas funcionalidades podem ser acedidas na página de gestão.



**Figura 14 - Demonstração de listas de reprodução criadas**

## Gestão de Episódios

Para gerir episódios foi criada uma área específica para a criação destes. À semelhança das outras áreas de gestão, este formulário também é validado em *frontend* e *backend*. Os episódios têm dois atributos que modificam a visibilidade do mesmo no lado do cliente, a visibilidade e o ficheiro de áudio. Por omissão, os episódios são criados sem áudio, e depois o áudio é acrescentado após o episódio criado. Enquanto um episódio não tiver áudio, o episódio não é mostrado no *frontoffice*. Do mesmo modo, se a visibilidade do episódio for “invisível”, o mesmo também não é mostrado no cliente, se o utilizador não estiver autenticado.



The screenshot shows a web browser window with the URL `localhost/podcasterv1/admin/index.php?p=episodes`. The page is titled "Architect" and has a sidebar with links for "Manage Users", "Manage Playlists", and "Manage Episodes". The main content area is titled "Manage Episodes" and includes a "Create Episode" button. Below this, there is a form with the following fields:

- Title \***: A text input field containing "Long Title".
- Short Title \***: A text input field containing "Short Title".
- Description \***: A text area containing "1234 Main St".
- Cover Image \***: A file upload button labeled "Escolher arquivo" and a status message "Nenhum arquivo selecionado".
- Audio File**: A section with the text "Audio file can only be uploaded after the episode is created" and an "Upload file" button.
- Tags \***: A text input field containing "tag1,tag2...".
- Duration (in minutes) \***: A text input field containing "30".
- Visibility \***: A text input field (partially visible).

Figura 15 - Pannel de criação de episódio

Para editar episódios podemos ir a esta página, onde também aparece um botão adicional para fazer *upload* de um ficheiro de áudio.

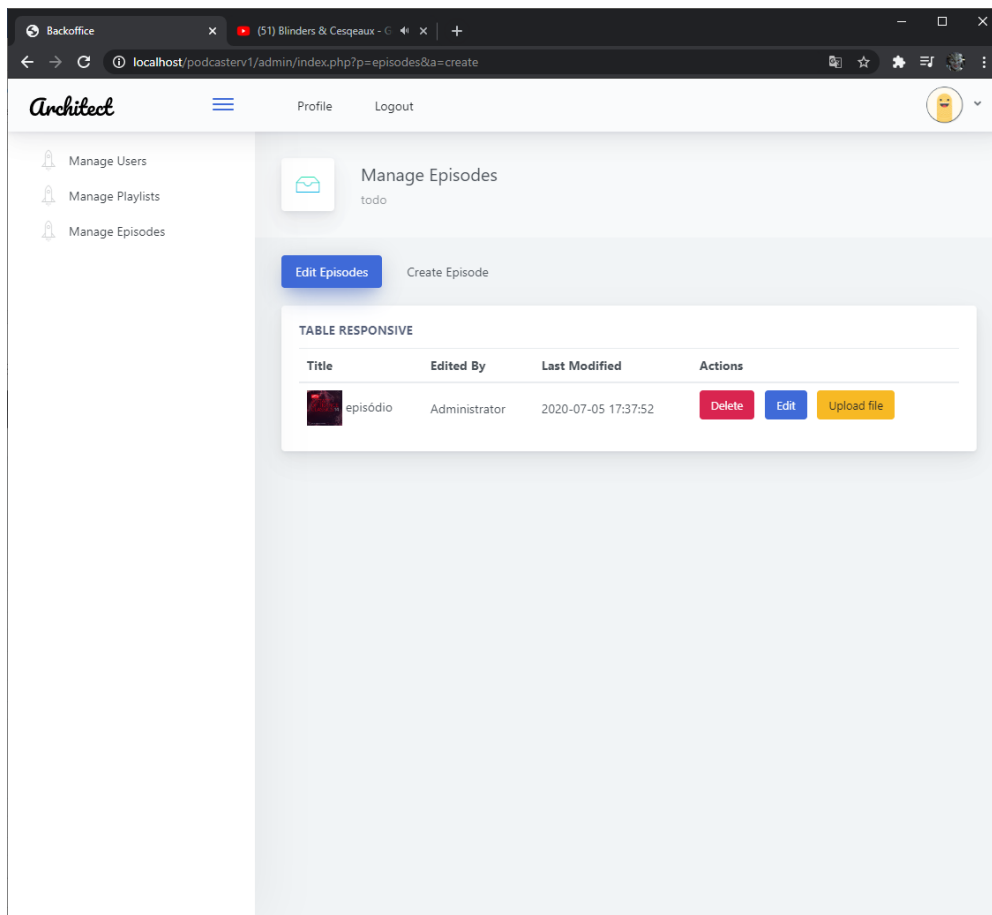
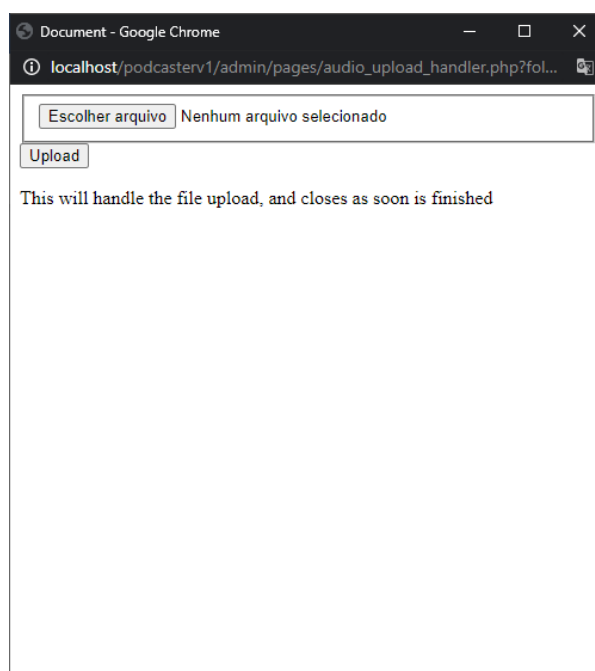


Figura 16 - Listagem de episódios

Ao clicar nesse botão, é disparada uma *popup* onde podemos fazer o *upload* do ficheiro.



Como já explicado acima, todos os ficheiros de áudio são carregados para um serviço externo, o Amazon S3, e por isso, optamos por colocar esta funcionalidade numa janela à parte, porque permite ao utilizador continuar a navegar pelo site enquanto o *upload* é tratado à parte.

## Frontoffice

Para este projeto, foi desenhado de raiz a parte visual e implementada com recurso à biblioteca UIKit e jQuery. Consiste em 3 páginas, a página inicial, que apresenta as categorias e os últimos episódios criados, a página de categoria, que lista episódios de uma determinada categoria, e a página de lista, que apresenta os episódios de uma determinada lista de reprodução.

No lado do *backend*, recorreremos a duas bibliotecas para construir esta parte: uma biblioteca de roteamento (de hiperligações) chamada Klein, que funciona em conjunto com uma configuração que permite obter estes links “limpos” e um sistema de *templates* baseado num componente de Laravel, chamada Blade One. Esta última permite com muita facilidade separar a *markup* de lógica PHP, e com isso, o HTML e CSS associado passa a estar desacoplado, possibilitando a criação de temas para este CMS.

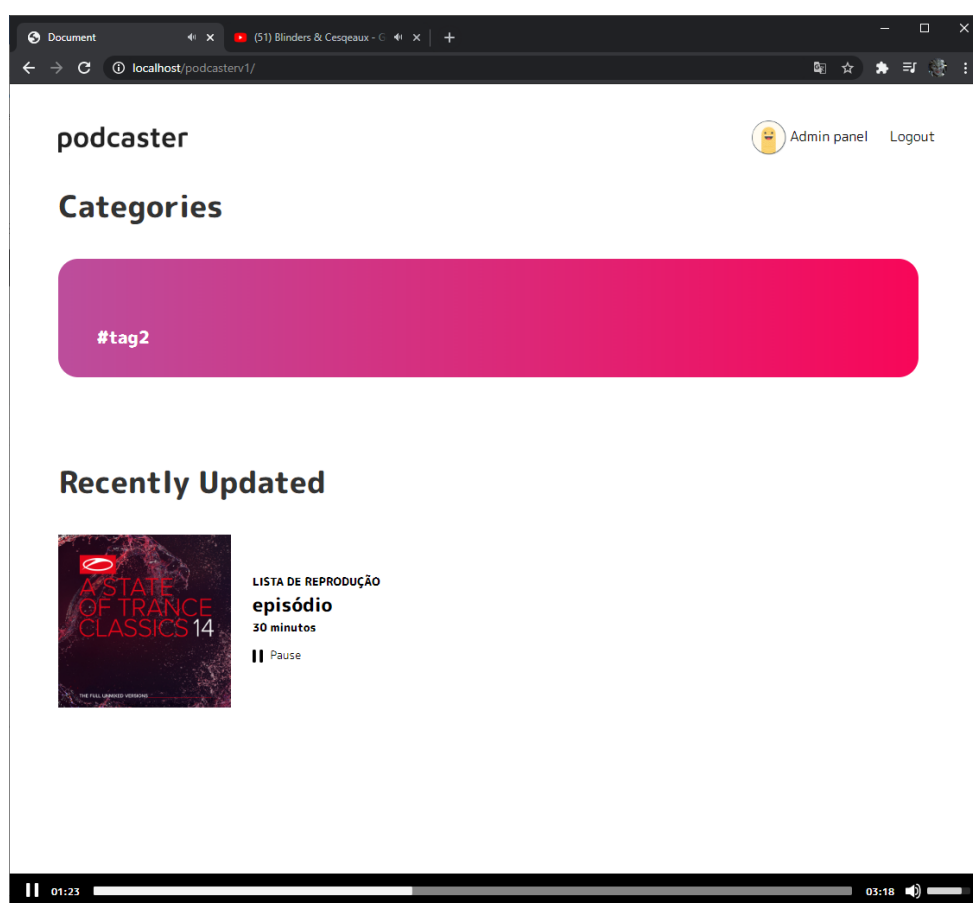


Figura 17 - Página inicial

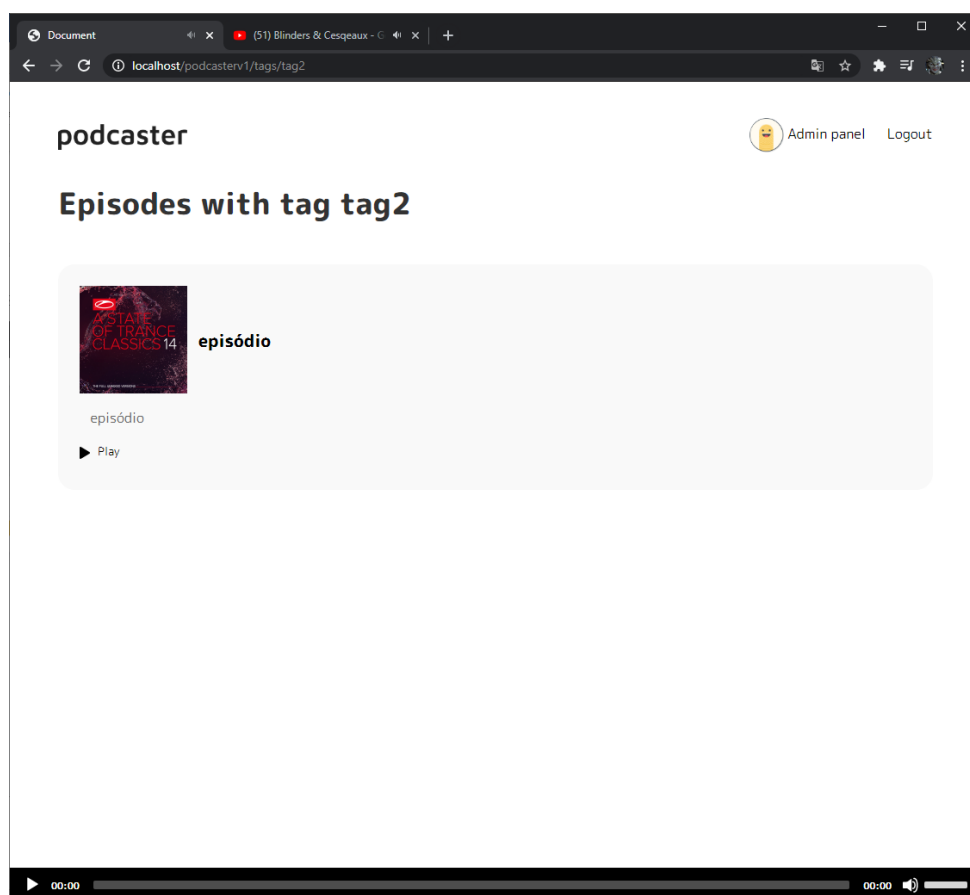


Figura 18 - Página de categoria

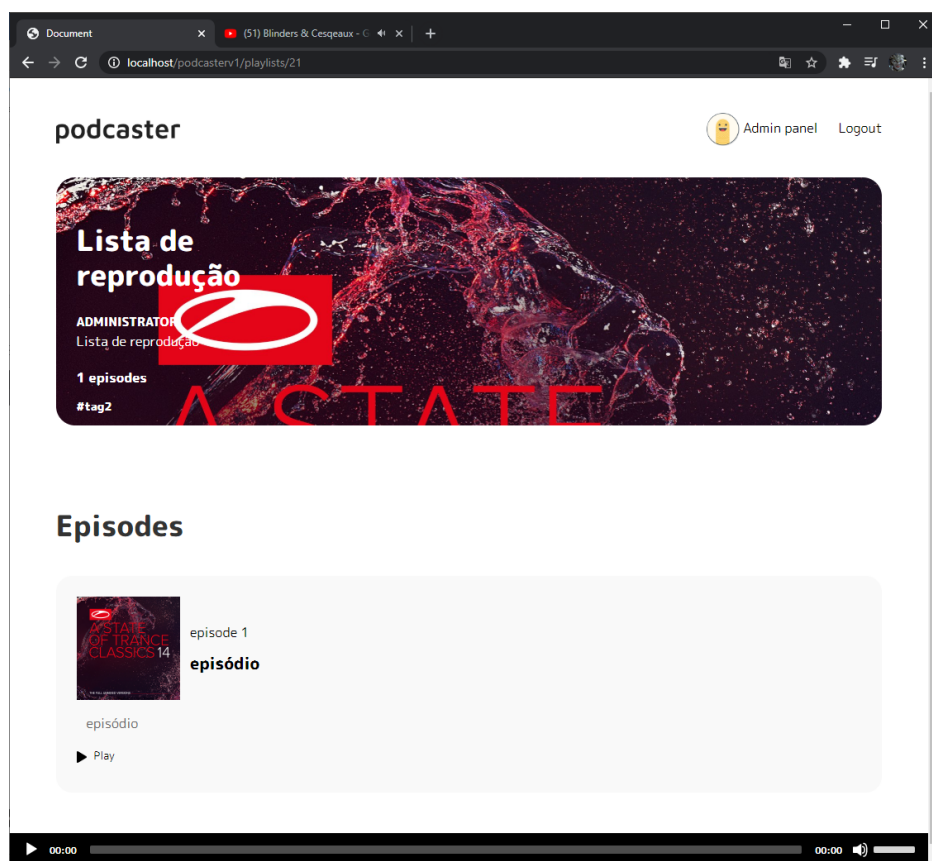


Figura 19 - Página de lista

No *frontoffice* usámos também a biblioteca do Amazon S3, que trata de criar links com a duração de uma hora para os ficheiros de áudio. Para tocar estes ficheiros usamos a biblioteca MediaElements, que torna muito fácil a implementação de um *plauer* de áudio. É também possível aceder ao painel de administração a qualquer momento a partir do *frontoffice*.

### **Bibliotecas externas usadas**

Como este projeto foi desenvolvido por uma única pessoa, a utilização de bibliotecas cortou muito tempo de desenvolvimento que seria necessário para desenvolver estas funcionalidades que as bibliotecas trazem. Para *backend* utilizamos as bibliotecas BladeOne, Klein, Amazon S3, PHPMailer e Grewgar/Captcha, Para a instalação destas bibliotecas utilizamos o Composer, que gere as dependências de cada biblioteca, e permite descarregar e instalar estas bibliotecas de uma forma muito fácil.

No lado do *frontend* utilizamos Bootstrap, UIKit, jQuery, MediaElements, Popper e mais umas bibliotecas que acrescentam funcionalidades ao jQuery. A utilização de Bootstrap permitiu construir sistemas responsivos com muita facilidade e já traz a gestão de modais e de pequenas outras funcionalidades que tornam o site mais responsivo ao toque.



### 3. Conclusão

Dado à dimensão deste projeto, um bom ponto de partida seria mesmo por começar com um sistema de gestão de conteúdos que já traz todas as funcionalidades necessárias à implementação de um projeto deste tipo, e também já trata alguns cenários de erro que possam ocorrer. Se desenvolvêssemos do CMS Wordpress já teríamos toda a parte de gestão de conteúdos acautelada, e bastaria apenas desenvolver as funcionalidades que permitissem ter o conteúdo mais específico desta implementação, mas como era pedido que este sistema fosse criado de raiz, tivemos de fazer um levantamento daquilo que eram as especificações do projeto, para que no final da implementação, tivéssemos cumprido todos os requisitos obrigatórios que eram dados.

No entanto, como já dissemos acima, foi no recurso a bibliotecas que tivemos muita implementação já feita, e só isso cortou imenso o tempo do desenvolvimento deste CMS. Um outro caminho de desenvolvimento seria utilizar uma *framework* como por exemplo Laravel, que já trata dos *captchas*, e que tem uma arquitetura MVC, que separa a lógica da apresentação.

Uma dificuldade que encontrámos foi no *upload* de ficheiros de grande tamanho. Inicialmente estávamos a carregá-los em memória, e depois a escrever para ficheiro, mas para ficheiros de 500 *Megabytes* esta solução causava um erro de memória, em que todo o *buffer* de memória ficava cheio e o servidor desligava. Resolvemos esse problema com a própria biblioteca, que permite o *upload* em partes, que depois são compostas no servidor da Amazon.







## Bibliografia

- [1] “Pseudocódigo - Leo Xavier - Jornalismo Digital, Microserviços, Elon Musk,” 31 01 2020. [Online]. Available: <https://pseudocodigo.transistor.fm/episodes/7-leo-xavier-wordpress-microservicos-media-elon-musk>.
- [2] “Content Managment System,” Wikipedia, [Online]. Available: [https://en.wikipedia.org/wiki/Content\\_management\\_system](https://en.wikipedia.org/wiki/Content_management_system). [Acedido em 04 Abril 2020].
- [3] “Wordpress Statistics,” Kinsta, [Online]. Available: <https://kinsta.com/blog/wordpress-statistics/>. [Acedido em 04 Abril 2020].
- [4] “MD5 PHP,” [Online]. Available: <https://www.php.net/manual/en/function.md5.php>.
- [5] “Wordpress,” Wikipedia, [Online]. Available: <https://pt.wikipedia.org/wiki/WordPress>. [Acedido em 2020 Abril 04].
- [6] “Sistema de gerenciamento de conteúdo,” [Online]. Available: [https://pt.wikipedia.org/wiki/Sistema\\_de\\_gerenciamento\\_de\\_conte%C3%BAdo#Principais\\_sistemas\\_CMS\\_gratuitos\\_dispon.C3.ADveis\\_no\\_mercado](https://pt.wikipedia.org/wiki/Sistema_de_gerenciamento_de_conte%C3%BAdo#Principais_sistemas_CMS_gratuitos_dispon.C3.ADveis_no_mercado).
- [7] “Password verify - php,” [Online]. Available: <https://www.php.net/manual/en/function.password-hash.php>.