

DEEC
DEPARTAMENTO DE ENGENHARIA
ELECTROTÉCNICA E DE COMPUTADORES

PROJECTO DE ENGENHARIA DE SOFTWARE 2017/2018

Etapa 2 Conceção do Software em UML

AUTORIA:

Diogo M. T. Poço	João C. da C. Barreiros
2014205007	2014196880
uc2014205007@student.uc.pt	uc2014196880@student.uc.pt

João M. P. Agria
2010129833
uc2010129833@student.uc.pt

Grupo: 2

16 de Março de 2018

Conteúdo

1	Introdução	2
2	Consolidação de Requisitos	2
3	Design Concetual	4
3.1	Descrição da Arquitectura do Sistema	4
3.1.1	Solução Proposta	4
3.1.2	Análise Comparativa de Arquitecturas	5
3.2	Interface Gráfica	7
4	Design Técnico	13
4.1	Arquitectura do Sistema Principal	13
4.2	Arquitectura do Sistema de Armazenamento	13
4.3	Descrição de Cenários Principais	14
4.4	Design do Código	17
4.4.1	Estruturas de Dados	17
4.4.2	Implementação de Métodos	20
4.4.3	Organização do Código Fonte	22
5	Conclusão	25

1 Introdução

Durante a primeira etapa do processo de desenvolvimento da aplicação de software o grupo de *developers* produziu um *deliverable* onde identificavam as exigências do *costumer* para o produto final e especificavam, recorrendo à linguagem de notação UML, os requisitos que descreviam o comportamento desejado para o sistema proposto. O próximo passo do processo de desenvolvimento é começar a criar o *design* que mapeará a construção do sistema.

Este documento será um dos dois *deliverables* a serem entregues no fim da segunda etapa do projecto, e o seu conteúdo divide-se em três secções principais: Consolidação de Requisitos, *Design* Concetual e *Design* Técnico.

Num projecto de desenvolvimento de software, a equipa de *developers* deve reunir-se com os *stakeholders* no final de cada etapa para apresentar documentos que registam o progresso do trabalho e obter indicações das alterações e melhorias a realizar antes de se prosseguir para a próxima fase. Assim, apesar de termos concluído a especificação de requisitos na etapa anterior, a consolidação dos requisitos será a primeira parte desta etapa. As alterações mais significativas serão aqui identificadas, e o *deliverable* apresentado no final da etapa 1 será actualizado e entregue em anexo.

O *design* da arquitectura será dividido em duas partes que descrevem uma solução particular para o problema definido pelos requisitos: o *Design* Concetual e o *Design* Técnico .

O primeiro destes é destinado para o cliente e descreve o sistema implementado em termos funcionais. Desta maneira, são concetualmente definidas a organização das estruturas de dados, a comunicação entre elas e a interactividade do utilizador com estas. Por fim, é apresentada uma análise comparativa entre a solução tomada e possíveis soluções alternativas, justificando as opções de implementação.

O segundo será para a equipa de *developers* e descreve pormenorizadamente o funcionamento do sistema. Isto implica uma projecção técnica sobre a solução a implementar que se prende na descrição da implementação, dos tipos e estruturas de dados, métodos e algoritmos utilizados. Descreve ainda a organização dos ficheiros utilizados e das inter-dependências de todos os componentes constituintes do sistema.

2 Consolidação de Requisitos

As alterações e correções feitas no *deliverable* entregue no final da etapa 1 foram as seguintes:

- Erros Ortográficos:
 - No segundo parágrafo da primeira página, 'desconstoí-se' foi alterado para desconstroi-se.
 - Na legenda da figura 1 na página 6, 'kyte' foi alterado para kite.
- Requisitos Funcionais

Foi alterada a apresentação dos requisitos funcionais na tabela da secção 2.1 da página 3 para desdobrar, quando necessário, cada requisito em vários requisitos funcionais mais simples. Desta forma facilitamos a leitura dos requisitos comparativamente com as longas frases que condensavam demasiada informação. Foi definida e adicionada uma categorização das prioridades de cada requisito funcional.

 - O requisito 3 da tabela foi expandido para definir as formas de browsing possíveis.
 - O requisito 8 foi expandido para permitir a criação de slideshows com música e a apresentação de forma organizada das fotos.

- Ainda nos requisitos funcionais foi adicionado o requisito 10, que originalmente tinha sido erradamente categorizado como restrição de design.
- Adicionámos o requisito 11, que diz respeito à apresentação organizada de fotos de uma página quando se navega a árvore de directórios.
- Restrições de Design
 - Na secção 2.3, restrições de design, foi retirada a linha "*A apresentação de fotos em slideshow é feita em ecrã completo*", pois foi-nos apontado que um slideshow é, por definição, realizado sempre em ecrã completo.
 - Foi também retirada a frase "*Quando se faz browsing, pode-se escolher entre miniaturas (thumbnails), apresentar só nome, lista detalhada, entre outros, para a vista.*". Esta frase enquadra-se melhor na categoria de requisitos funcionais.
- Especificação de Requisitos
 - O título da secção 3 foi alterado: era *Diagramas UML* e passou a ser *Especificação de Requisitos*.
 - A figura 1 definia que o caso de uso *Visualizar* estava incluído no caso de uso *Imprimir*, mas na realidade, o caso de uso *Imprimir* é uma extensão de *Visualizar*.
 - Na figura 2 foi incluído o actor *Impressora* e o caso de uso *Imprimir*, uma vez que se trata de uma vista mais detalhada do caso de uso *Visualizar*. Foi também incluído o caso de uso *Browsing*.
 - Na figura 4, 5 e 6 foram eliminadas as ligações entre o actor *Utilizador* e o caso de uso *Confirmar*, uma vez que nos diagramas já estavam implícitas essas ligações.
 - Na figura 4 foi também retirado o caso de uso *Mover*.
 - Na secção 3.2, *Diagrama de Classes*, foi alterado o segundo parágrafo. Onde anteriormente se podia ler "*A classe pessoa tem uma relação de agregação com a classe foto.*", agora está escrito "*A classe foto tem uma relação de agregação com a classe pessoa.*". Esta diferença de sintaxe define a classe foto como a classe agregadora da classe pessoa e não o contrário. Desta forma, a frase não está em contradição com o esquemático da figura 7 e a frase que se seguia: "*Uma foto pode ter várias pessoas associadas.*".
- Glossário Foi adicionada uma definição para álbum e outra para página.

3 Design Concetual

Um problema de *design* de software tem um conjunto de soluções infinito. Nesta secção descreveremos pormenorizadamente a nossa solução a um nível concetual, ou seja, queremos que sejam claras ao *costumer* todas as funcionalidades que correspondem aos requisitos. Cabe à equipa de *developers* fazer uma análise comparativa da solução e das opções tomadas, de forma a justificar a solução final.

Iremos começar pela arquitectura da solução, e depois passar para a interface gráfica de utilização da mesma.

3.1 Descrição da Arquitectura do Sistema

A decomposição do sistema em subsistemas é uma técnica tradicional que ajuda os *developers* a perceber e isolar os problemas-chave que o sistema visa resolver. Esta construção de uma arquitectura particionada tem várias vantagens, nomeadamente a construção independente e em paralelo de cada unidade de software e a possibilidade de realizar manutenção/reconstrução de uma das unidades de software de forma invisível às restantes. Desta forma cria-se uma arquitetura de fácil manutenção.

O método de *design* escolhido é o *feature-oriented design*, que particiona o sistema em módulos com *features*, ou características, atribuídas. Num nível de abstracção superior, o *design* descreve o sistema em termos de serviços e uma colecção de *features*, num nível de abstracção inferior descreve como cada *feature* melhora um serviço e quais as interações entre as *features*.

3.1.1 Solução Proposta

Uma representação ilustrativa da arquitectura lógica do sistema e principais dependências inter-módulos, quando descrita num nível de elevada abstracção, pode ser encontrada no diagrama de pacotes UML da figura 1.

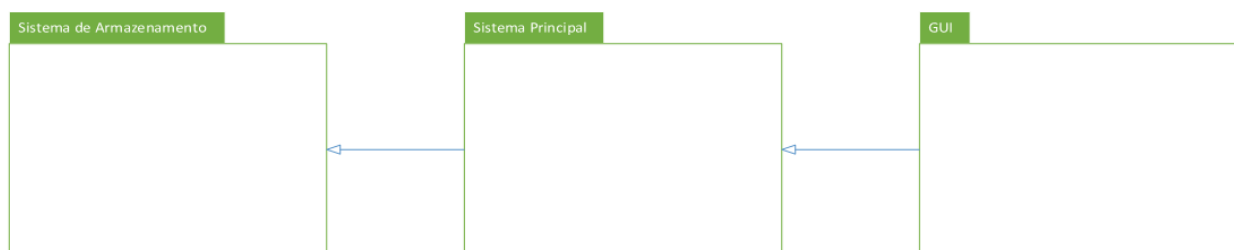


Fig 1: Diagrama de pacotes relativo ao maior nível de abstracção da arquitectura do sistema

Os três subsistemas são a Interface Gráfica de Utilização (GUI), o Sistema Principal e o Sistema de Armazenamento. Procedemos a descrever de forma sucinta cada um deles:

Interface Gráfica de Utilização: Componente responsável pela interacção do utilizador com o Sistema Principal. Permite consultar e modificar informação contida no Sistema Principal.

Sistema Principal: Componente que contém as funcionalidades de resposta aos requisitos funcionais, bem como toda a informação, em memória temporária, referente às classes e estruturas de dados utilizadas.

Sistema de Armazenamento: Módulo responsável pela gestão do conjunto de ficheiros armazenados em disco. Estes ficheiros serão utilizados pelo Sistema Principal para leitura e escrita,

de modo a manter a informação guardada de forma permanente e consistente.

Para além dos pacotes, o diagrama apresenta também as respectivas dependências entre eles:

Interface Gráfica de Utilização - Sistema Principal:

Quando o utilizador está a consultar informações, a informação apresentada é obtida do Sistema Principal. Além do propósito primário do display de informação relativa ao Sistema Principal, a Interface Gráfica de Utilização também responde a mensagens de controlo (como por exemplo recebendo indicativos de erros da parte Sistema Principal). Quando o utilizador modifica algum valor na Interface, esta informação invocará acções por parte do Sistema Principal e comunicar-lhe-á informação a actualizar.

É fácil concluir que a interface gráfica tem uma dependência maior do sistema principal que o contrário. Uma vez que a força das dependências entre os subsistemas é assimétrica, escolhemos representar na figura 1 apenas a mais proeminente.

Sistema Principal - Sistema de Armazenamento:

O Sistema de Armazenamento é dependente do Sistema Principal porque é responsável pelo armazenamento toda a informação do sistema nos ficheiros devidos. Os ficheiros do sistema têm de ser actualizados sempre que uma alteração de informação no Sistema Principal implique perda de consistência de dados armazenados em disco, para assegurar um certo grau de fiabilidade.

Sublinhamos que, apesar disso, o Sistema Principal é extremamente dependente do Sistema de Armazenamento, uma vez que lê dos ficheiros a informação necessária à inicialização da aplicação. Sem a informação do Sistema de Armazenamento, o sistema global não pode funcionar.

A dependência do Sistema Principal do Sistema de armazenamento é superior à dependência inversa, como se pode observar na figura 1.

3.1.2 Análise Comparativa de Architecturas

Quando o grupo idealizou a arquitetura, determinámos logo a necessidade de dividir as tarefas de gerir a memória, gerir a interface gráfica e gerir o sistema principal em módulos distintos. A criação de módulos específico para estes componentes da aplicação permitem que, caso o *costumer* não aprove o *design* que criámos ou indique ajustes, cada componente possa ser alterado sem afetar nenhum outro componente. O grupo irá, nesta secção, analisar e justificar as opções tomadas e comparar com as alternativas possíveis.

O armazenamento e leitura de dados é uma componente essencial do programa com uma estrutura suficientemente distinta para justificar a sua separação. As nossas decisões relativamente a este módulo foram as seguintes:

- **Utilizar uma base de dados vs. Armazenamento em ficheiros:**

Foi discutido, tanto em sessões intergrupais como em sessões intragrupais, a possibilidade de utilizar uma base de dados para implementar a gestão da memória. O nosso grupo decidiu contra devido á nossa inexperiência nessa área, que poderia comprometer a qualidade do produto e o cumprimento de metas temporais. A segunda alternativa foi escolhida dado já conhecermos as bases de como realizar esta tarefa.

- **Leitura e escrita intrínseca a cada módulo vs. Implementação de um módulo isolado:**

Conferir a cada um dos outros módulos a possibilidade de gerir os ficheiros que lhes dizem respeito dificultaria a criação de métodos em cada classe. Optámos por criar um subsistema responsável pela gestão de armazenamento de informação. Esta opção oferece a vantagem de

isolar a operação de entrada e saída, facilitando a evolução e alteração dos mecanismos de entrada e saída sem ter a necessidade de alterar outros módulos. Por outro lado, introduz maior acoplamento no sistema (todos os restantes módulos invocam acções deste), e diminui a coesão dos módulos quando comparada com a alternativa anterior. Para dar alguma flexibilidade à evolução de métodos de entrada e saída, possibilitando a reutilização dos restantes módulos sem necessidade de modificar o código, o grupo optou pela última abordagem.

O Sistema Principal será responsável por manter a coerência interna dos dados, impedindo, por exemplo, casos de remoção de fotos já não existentes no sistema ou a adição de fotos duplicadas na mesma página. Sobre este módulo decidimos:

- **Gestão de informação global do Sistema Principal em listas vs. árvores binárias:**

A informação do sistema será organizada em listas ligadas, nomeadamente listas de Álbuns, Páginas, Fotos e Pessoas. Estas listas foram estruturadas de forma a tornar a navegação e pesquisa de informação do sistema mais rápida e organizada. O grupo optou por uma forma de armazenamento dinâmica, dada a incerteza nas quantidades de informação a armazenar. Os detalhes técnicos da implementação podem ser encontrados na secção de *Design Técnico*. A implementação em árvores binárias aumentaria demasiado a complexidade do Sistema Principal e as vantagens que traria seriam apenas na área da pesquisa.

Outras decisões tomadas sobre a arquitectura foram:

- A decisão da utilização de linguagem de programação C++, face à possibilidade de escolher a linguagem Java. O grupo tomou esta decisão devido à sua proficiência em C++ e inexperiência em Java.
- A decisão de utilizar o IDE Qt Creator devido à possibilidade de desenvolvimento entre diferentes sistemas operativos. A experiência de um dos elementos do grupo em desenvolver interfaces gráficas neste IDE também pesou na nossa decisão.
- A decisão de guardar as informações referentes ao programa em si, ou seja, os álbuns, páginas, fotos e pessoas em ficheiros binários. Desta forma asseguramos que os ficheiros poderão ser apenas alterados pelo sistema aquando qualquer alteração ou adição de informação por parte do utilizador no GUI, e criamos uma barreira contra alterações por parte do utilizador. Outras vantagens da utilização de ficheiros binários são: a rapidez de leitura e escrita destes ficheiros comparativamente aos ficheiros de texto, um ficheiro binário tem um tamanho muito menor que um ficheiro de texto com a mesma informação, a procura de informação num sistema binário é mais simples, bem como a leitura e edição de pedaços de informação dentro do ficheiro.
- Uma vez que ambos os tipos de ficheiros guardados serão utilizados e consultados pelo mesmo utilizador - ou sistema num mesmo computador - serão guardados no disco rígido do computador. Assim, garantimos um rápido e constante acesso às informações

3.2 Interface Gráfica

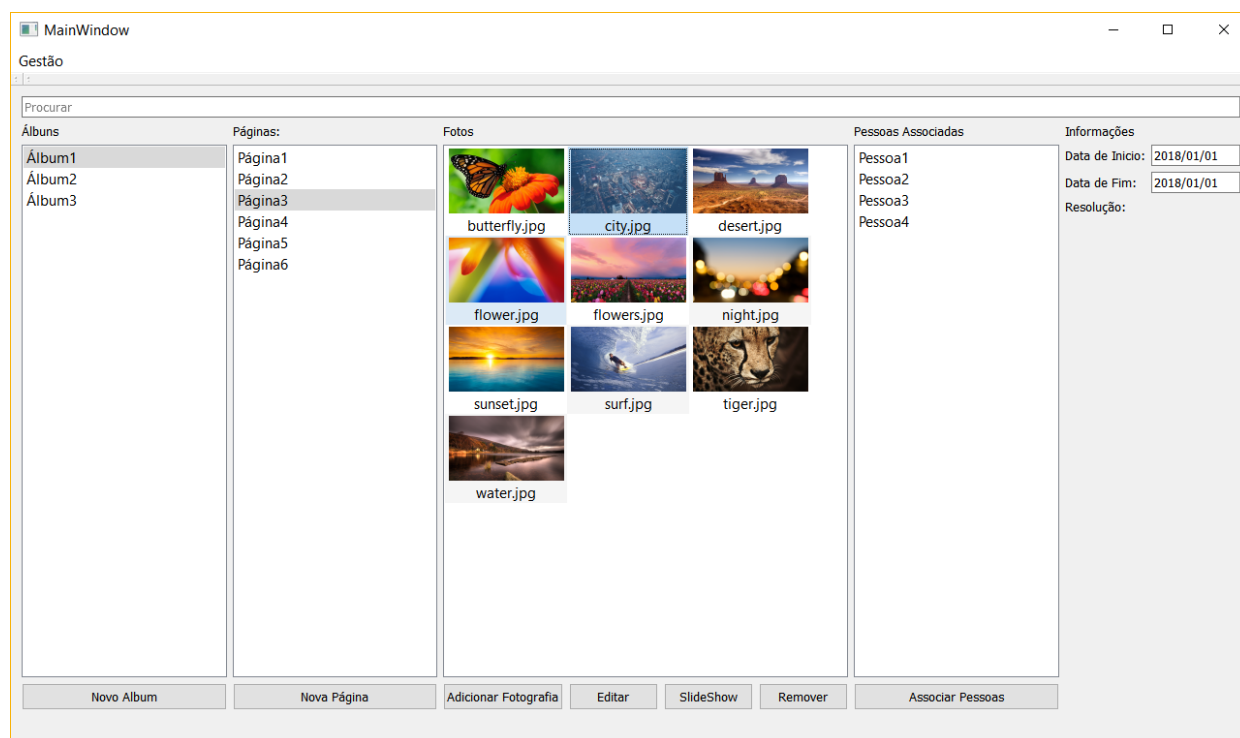


Fig 2: Janela inicial da aplicação

Quando o utilizador inicializa o programa, é esta a janela inicial que aparece. Aqui o utilizador pode ver os conteúdos da árvore de directórios divididos em três aberturas devidamente tituladas e dispostas em ordem decrescente de hierarquia. Pode-se observar que no fundo da janela existem botões que realizam alguns dos requisitos funcionais da aplicação. Nas próximas páginas vamos mostrar algumas janelas auxiliares. Começamos por clicar com o botão esquerdo do rato no menu de Gestão no canto superior esquerdo. A seguinte janela abre-se e podemos observar alguns dos atalhos definidos:



Fig 3: Menu Gestão

Quando o utilizador escolher a opção de adicionar novo álbum, esta será a janela que lhe aparecerá. Devem ser introduzidos os dados como o nome, a descrição o tipo de página e o directório, tal como foi definido no documento de identificação de requisitos.

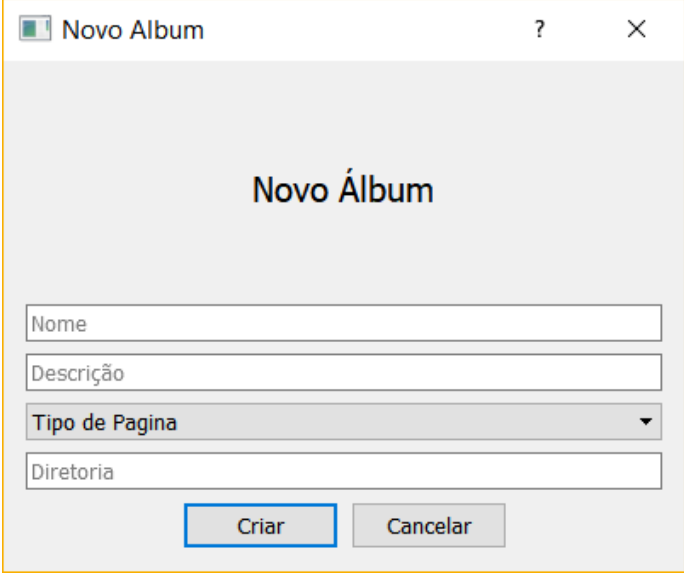
A screenshot of a software dialog box titled "Novo Álbum". The dialog has a title bar with a green icon, a question mark, and a close button. The main area has a light gray background with the title "Novo Álbum" in bold. Below the title are four input fields: "Nome", "Descrição", "Tipo de Pagina" (a dropdown menu), and "Diretoria". At the bottom are two buttons: "Criar" (highlighted with a blue border) and "Cancelar".

Fig 4: Adicionar novo Álbum

Similarmente, esta é a janela que aparece ao utilizador que escolha adicionar uma nova página:

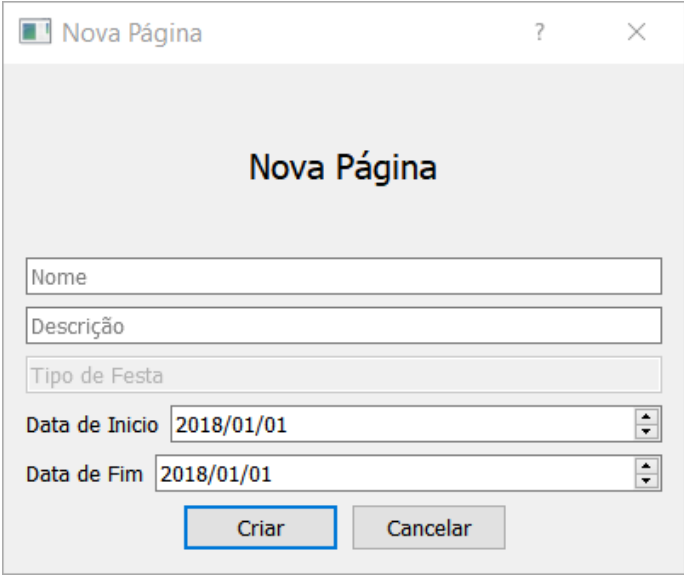
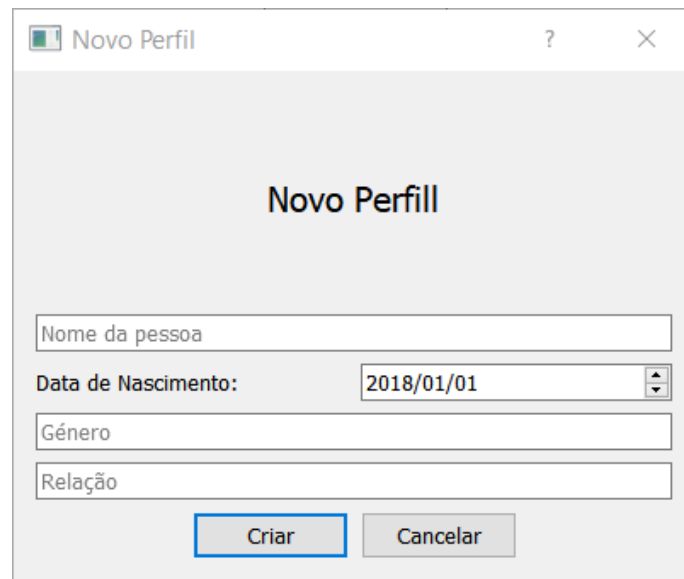
A screenshot of a software dialog box titled "Nova Página". The dialog has a title bar with a green icon, a question mark, and a close button. The main area has a light gray background with the title "Nova Página" in bold. Below the title are five input fields: "Nome", "Descrição", "Tipo de Festa", "Data de Inicio" (with a date picker showing "2018/01/01"), and "Data de Fim" (with a date picker showing "2018/01/01"). At the bottom are two buttons: "Criar" (highlighted with a blue border) and "Cancelar".

Fig 5: Adicionar nova Página

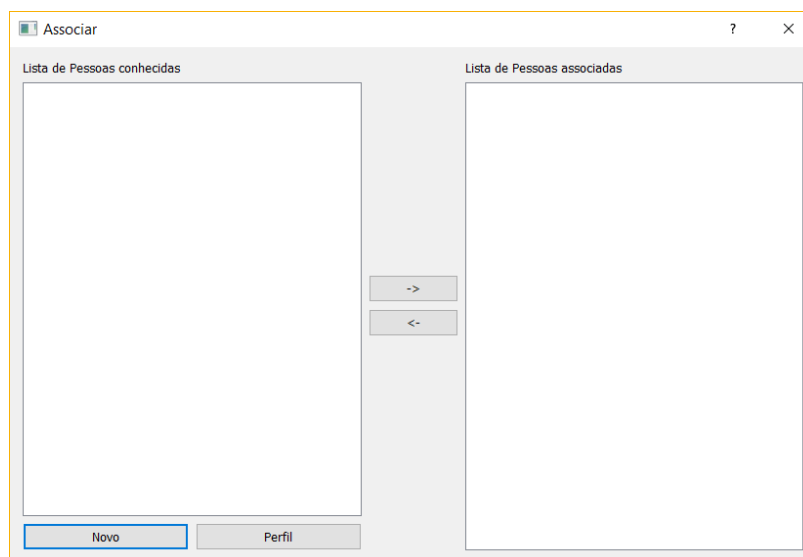
Para adicionar uma nova pessoa, este é o menu que permite ao utilizador introduzir os dados necessários:



A dialog box titled "Novo Perfil" with a standard Windows title bar. The main area contains the title "Novo Perfil" in bold. Below it are four input fields: "Nome da pessoa", "Data de Nascimento:" (with a date picker showing "2018/01/01"), "Género", and "Relação". At the bottom are two buttons: "Criar" (highlighted with a blue border) and "Cancelar".

Fig 6: Adicionar nova Pessoa

Quando o utilizador quiser associar uma ou mais pessoas uma foto ou várias fotos, este será o menu que irá permitir cumprir o requisito:



A dialog box titled "Associar" with a standard Windows title bar. It features two large empty rectangular areas: "Lista de Pessoas conhecidas" on the left and "Lista de Pessoas associadas" on the right. Between these areas are two buttons: "->" and "<-. At the bottom left are two buttons: "Novo" (highlighted with a blue border) and "Perfil".

Fig 7: Menu de Associação

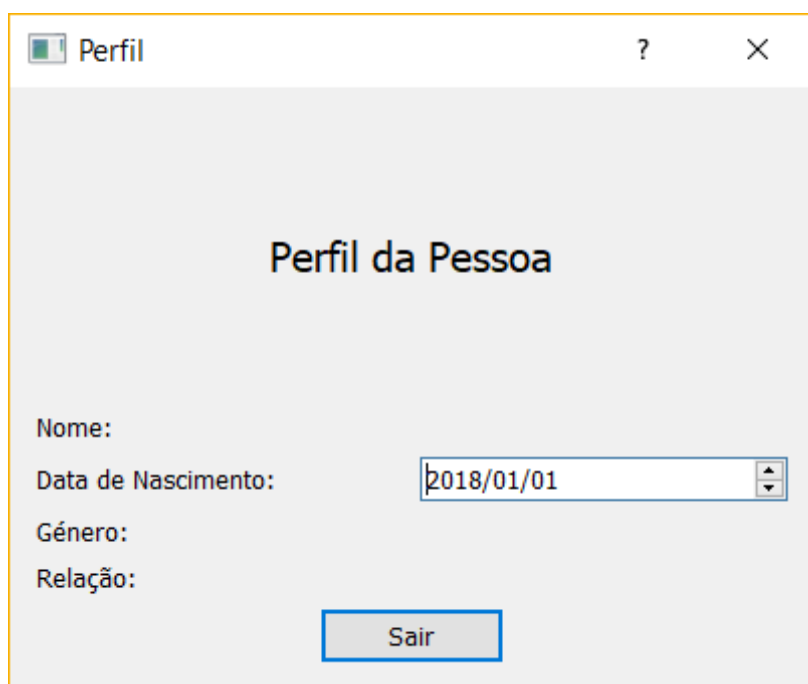


Fig 8: Janela auxiliar que mostra o perfil de uma pessoa

Quando se seleciona uma pessoa, a janela auxiliar da figura acima apresenta os seus dados. A janela que permite criar um Slideshow está em baixo.

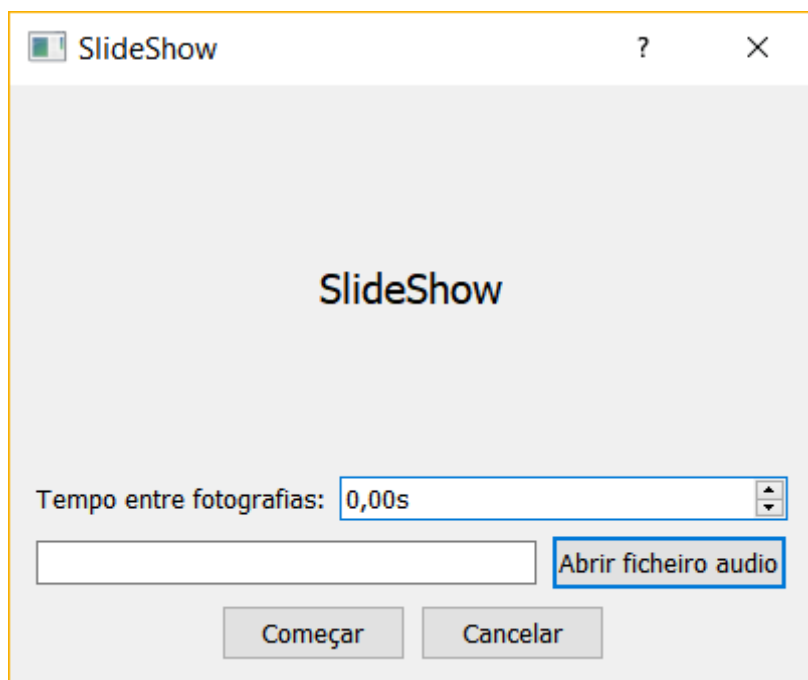


Fig 9: Criar Slideshow

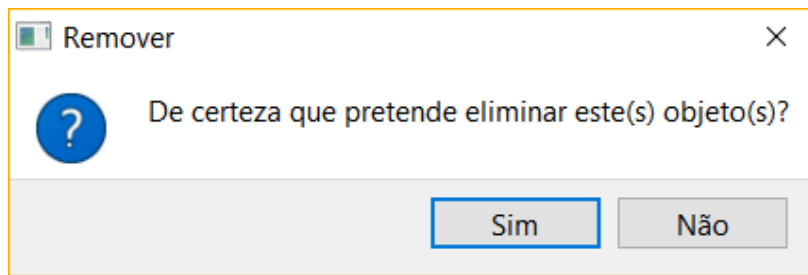


Fig 10: Janela de confirmação para remover um elemento

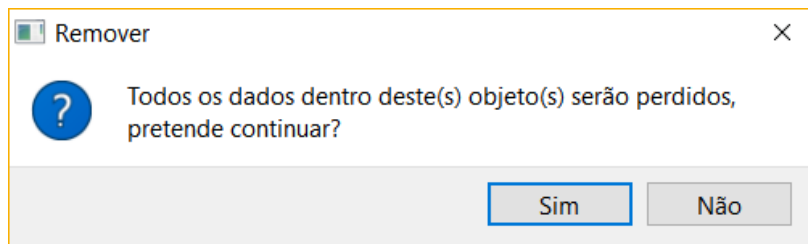


Fig 11: Segunda janela de confirmação para remover um álbum ou página

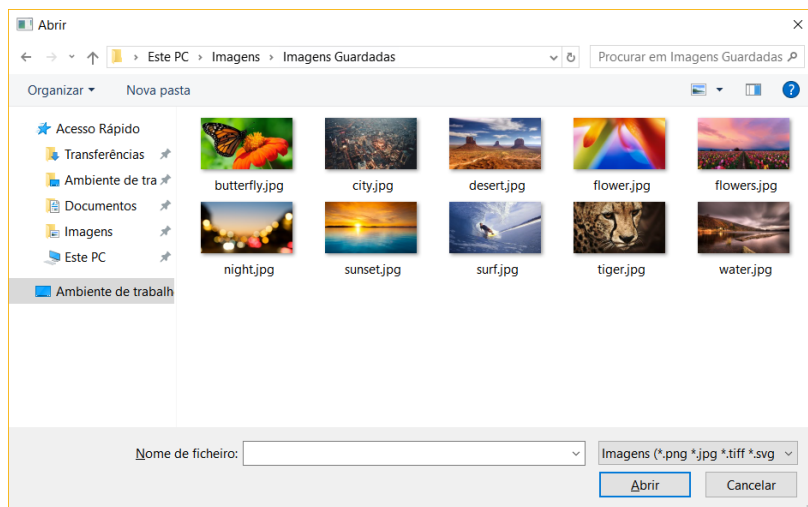


Fig 12: Para adicionar uma imagem

A figura 12 mostra o processo de adicionar uma imagem à aplicação.

Quando se selecciona uma imagem, podemos observá-la em grande escala e é-nos apresentada a opção de impressão.



Fig 13: Janela que permite ver a imagem em tamanho grande

4 Design Técnico

Esta secção pretende descrever detalhadamente a vertente técnica da implementação do projecto. Para os módulos intervenientes na descrição da arquitectura do sistema, representada na figura 1, são especificadas as estruturas de dados a utilizar e o seu modo de relacionamento, procedendo-se posteriormente a uma explicação detalhada dos métodos a implementar.

4.1 Arquitectura do Sistema Principal

A modelação do Sistema Principal, já definido concetualmente, tem como base os métodos e atributos representados na figura 14. Convém referir que os atributos de cada classe são privados. Para ler os atributos Assumimos que cada objecto de uma classe é identificado univocamente pelos seus atributos. Por exemplo, uma foto é identificada não só pelo seu nome data e resolução, como pelo seu directório.

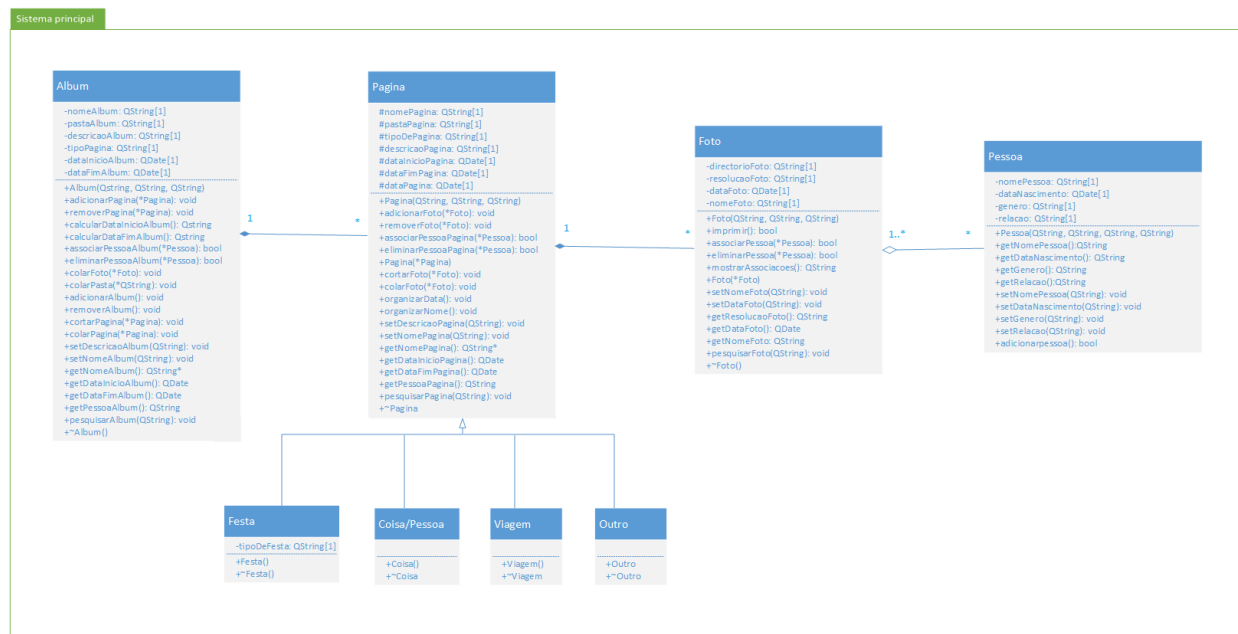


Fig 14: Diagrama referente à classe do Sistema Principal

4.2 Arquitectura do Sistema de Armazenamento

O módulo do Sistema de Armazenamento, além de englobar os ficheiros de armazenamento de informação propriamente ditos, também abrange as funções de gestão de leitura e escrita para estes ficheiros. Na classe concentram-se os atributos e métodos necessários a esta gestão, especificados na figura 14. Esta classe é responsável por todas as transacções do sistema principal com os ficheiros, sejam eles em formato binário ou de texto. Permite então a leitura e escrita de todas as adições ou edições de informação para/do disco rígido.



Fig 15: Diagrama referente à classe Ficheiros

4.3 Descrição de Cenários Principais

Vamos ilustrar o comportamento do software em alguns cenários relevantes, definidos no documento de especificação de requisitos como casos de uso. Para completar esta descrição são utilizados diagramas de sequência UML, que possibilitam a compreensão da interacção inter-classes e a observação, numa perspectiva temporal, do desenrolar de algumas acções escolhidas.

Nalguns diagramas as setas apresentam uma pequena inclinação que deverá ser desprezada. Não existe uma comunicação entre objectos que demore um intervalo de tempo considerável. Esta situação deveu-se a problemas na utilização da ferramenta de edição dos diagramas; de notar também que os diagramas são auto-explicativos. Sendo assim, apresentam-se em baixo dois diagramas de sequência relativos aos casos de uso de maior prioridade: Criar álbuns e Criar páginas.

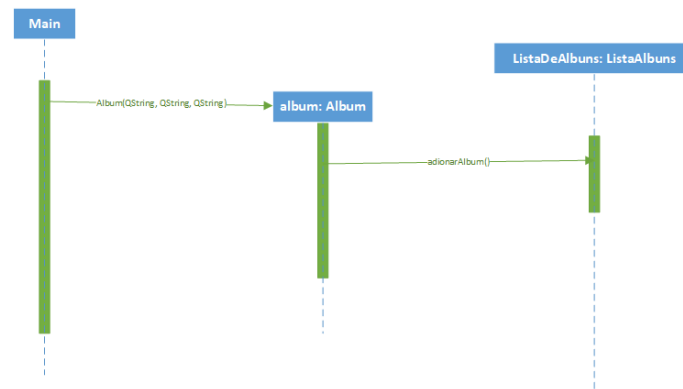


Fig 16: Diagrama de sequência relativo ao caso de uso Adicionar Álbum

Adicionar pessoas novas à lista de pessoas mantida pela aplicação é um processo muito semelhante ao processo de adicionar álbuns novos, pelo que optámos por omitir a inclusão do seu diagrama de sequência. Quanto ao caso de uso adicionar fotos, este teria um diagrama de sequências que se tornaria uma réplica do diagrama de sequências da figura 17, e por isso evitámos representá-lo.

O nosso grupo não acredita que diagramas de sequência sejam vitais para demonstrar a funcionalidade do sistema, por isso escolhemos apresentar o diagrama de actividades, dividido pelas das figura 18, 19, 20, 21, 22 e 23 para facilitar a leitura. Como se verifica pelas figuras, o GUI permite a escolha das operações definidas nos requisitos funcionais.

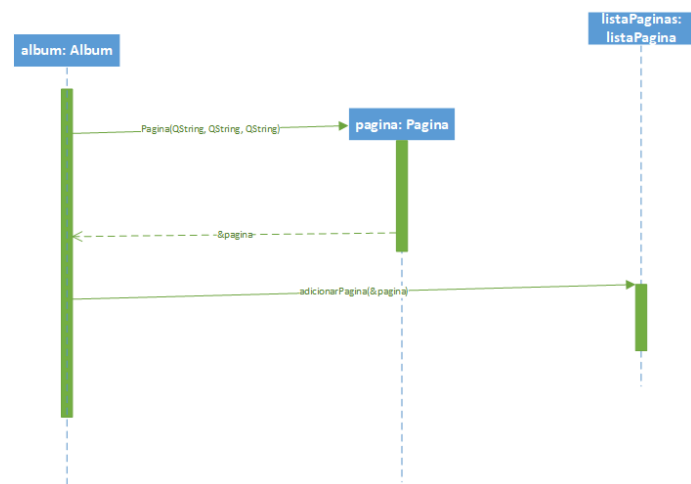


Fig 17: Diagrama de sequência relativo ao caso de uso Adicionar Página

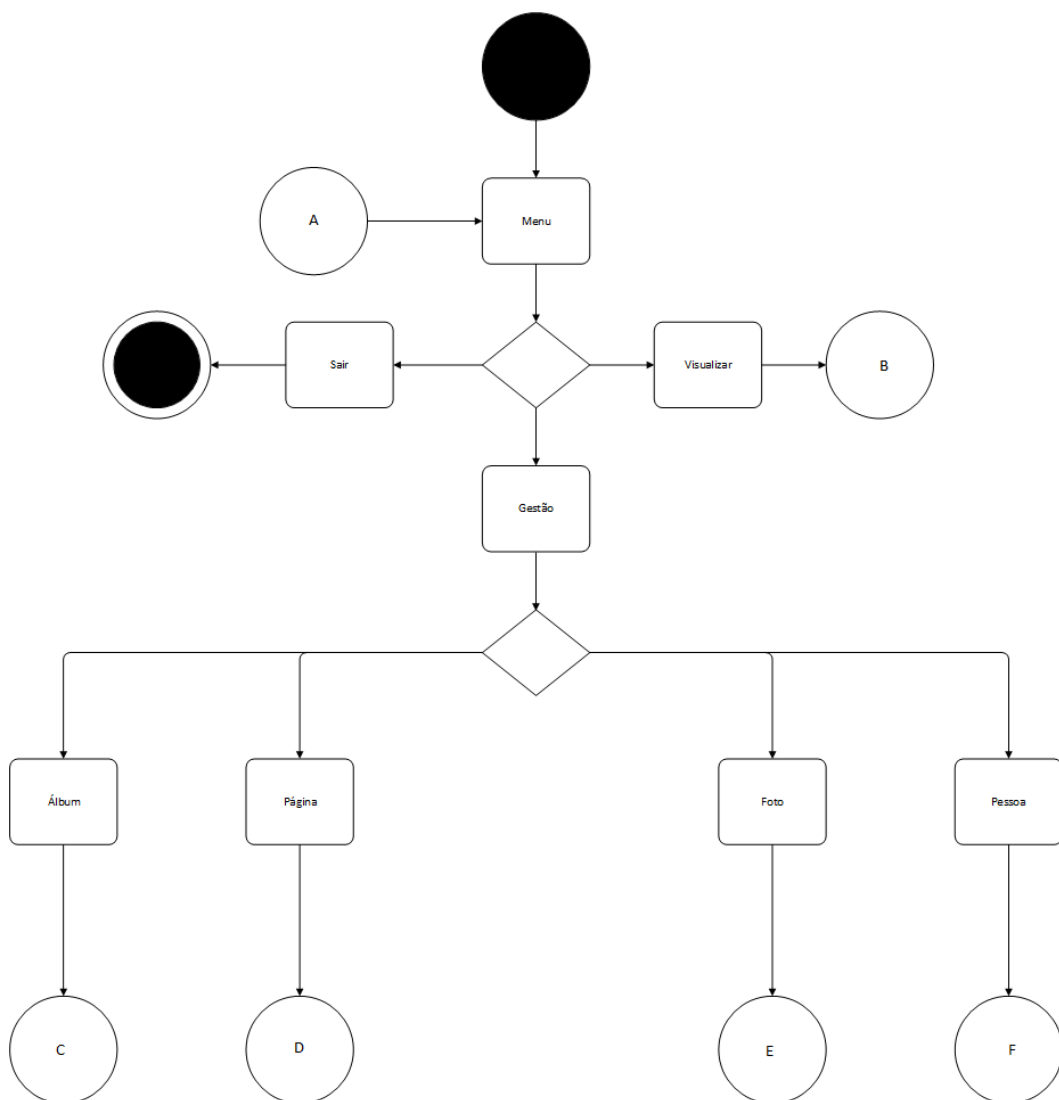


Fig 18: Diagrama de actividades, menu principal

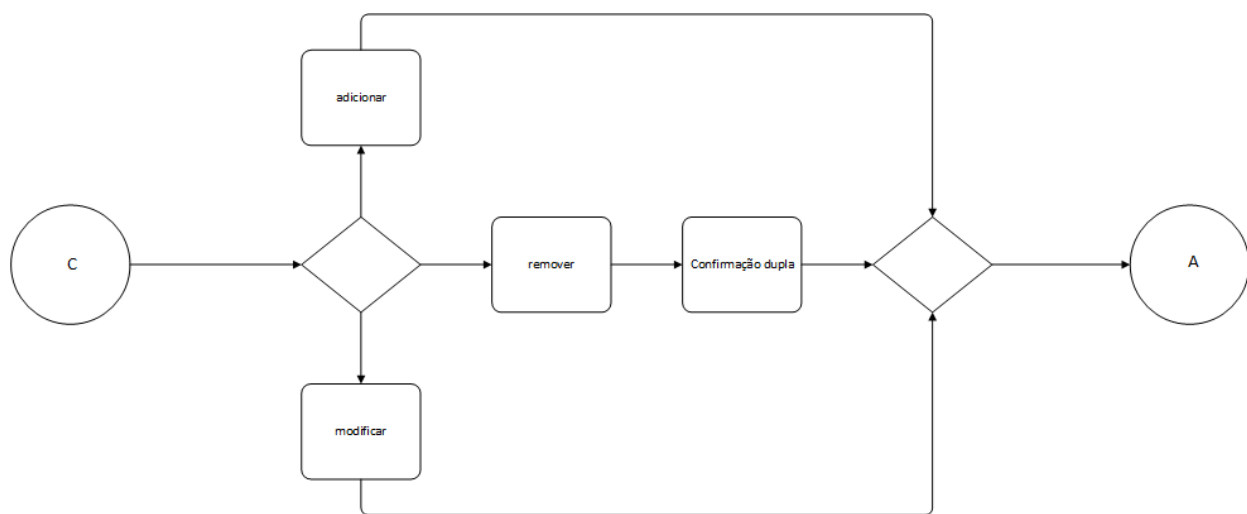


Fig 19: Diagrama de actividades, gestão de álbuns

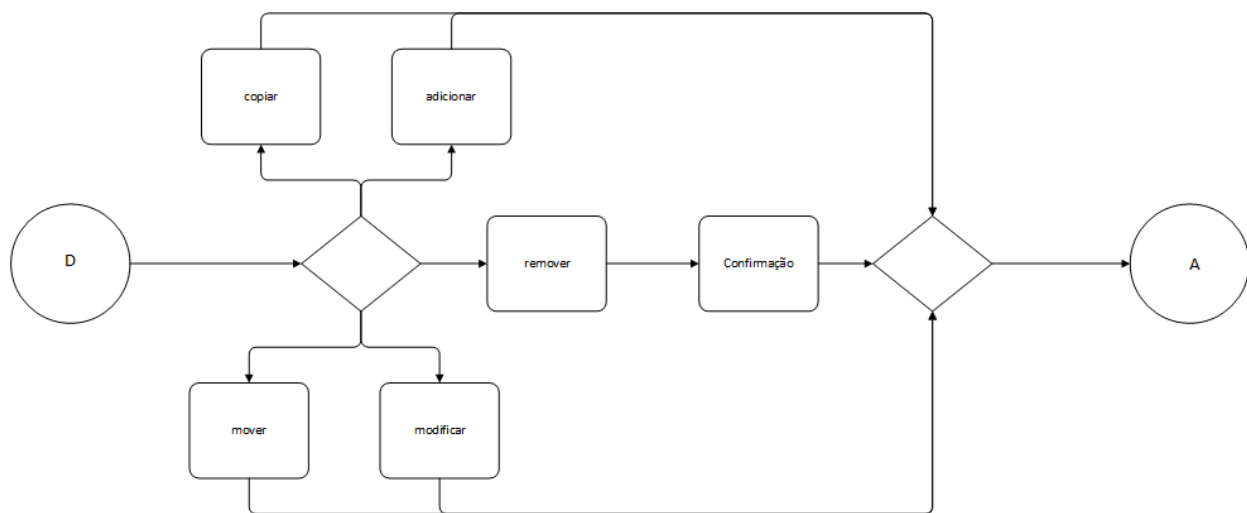


Fig 20: Diagrama de actividades, gestão de páginas

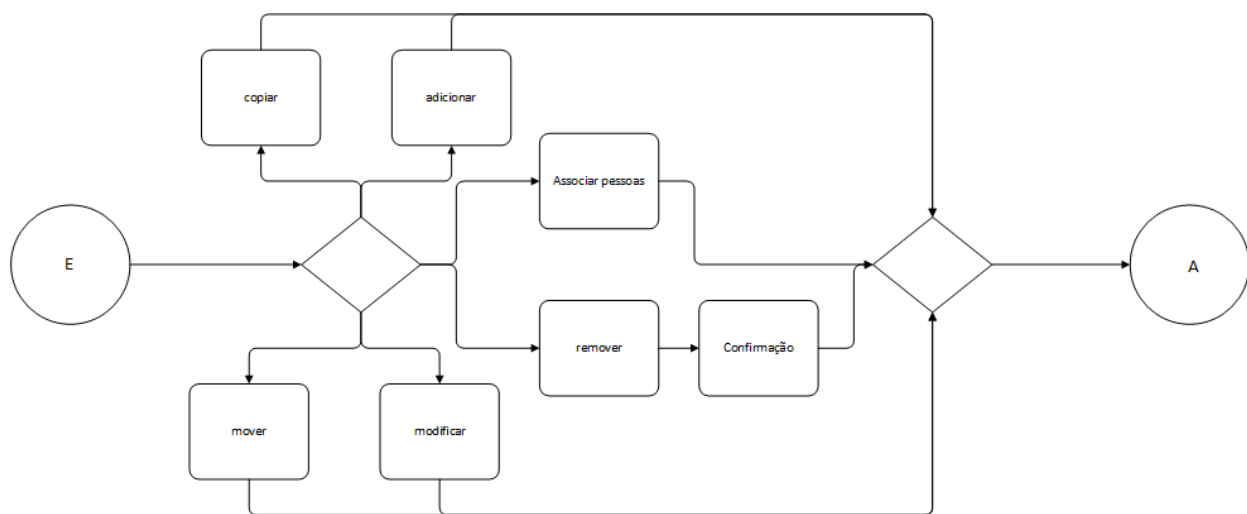


Fig 21: Diagrama de actividades, gestão de fotos

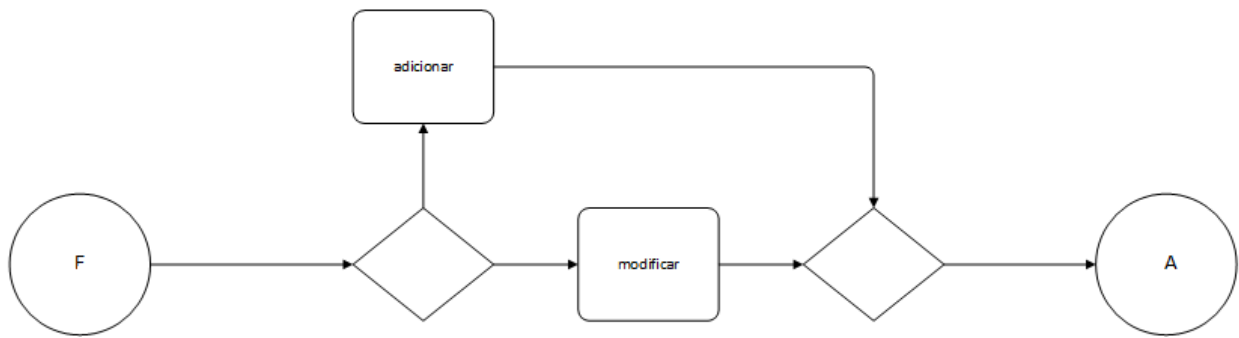


Fig 22: Diagrama de actividades, gestão de pessoas

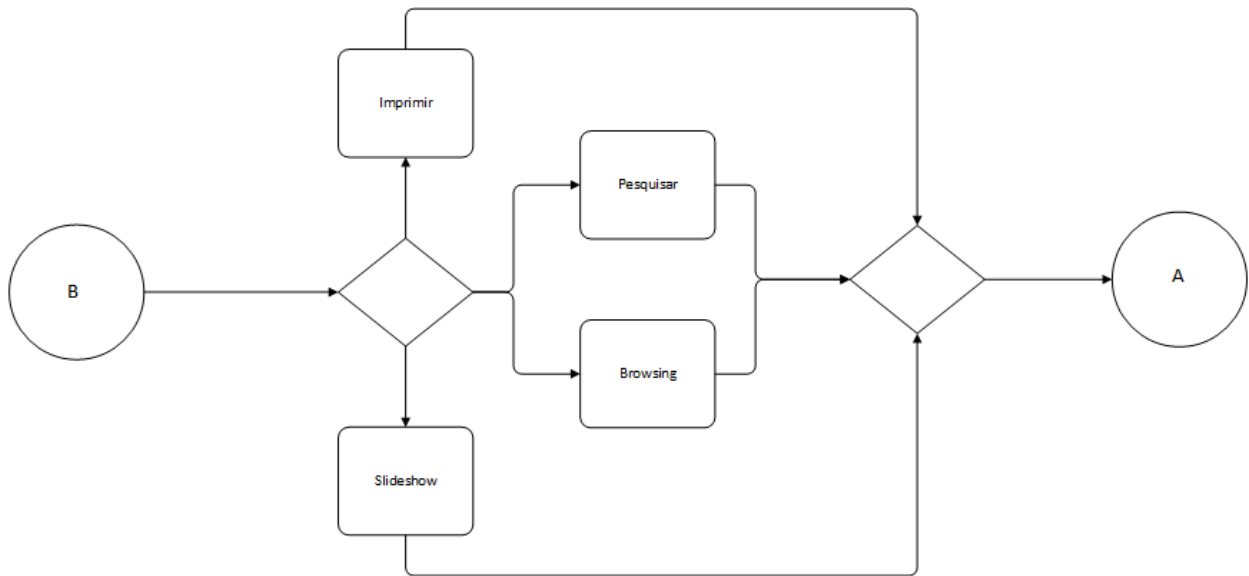


Fig 23: Diagrama de actividades, visualizar

4.4 Design do Código

Nesta subsecção iremos: definir as estruturas de dados a utilizar em cada um dos componentes e a nível global, especificar o funcionamento concreto de funções do sistema e métodos das classes, explicitar a organização do código fonte (o nome dos ficheiros intervenientes, conteúdos e dependências). Começa-se, então, pela definição do tipos de estruturas de dados utilizados no projecto, e do seu propósito no sistema.

4.4.1 Estruturas de Dados

Na parte do *Design Técnico*, referimos que a informação do sistema seria mantida em listas ligadas globais estruturadas para facilitar a adição, remoção e modificação de elementos e a pesquisa de informação. A figura 23 apresenta um esquemático de fácil interpretação que sintetiza a estrutura. Utilizamos setas para simbolizar ponteiros, mas omitimos algumas para simplificar o esquemático. Cada rectângulo é um nó de uma lista ligada.

- **Lista de Álbuns:** Armazena todos os álbuns introduzidos no sistema, um em cada nó. O ponteiro cabeça de lista é uma variável global. Cada nó desta lista possui, para além do ponteiro para o próximo elemento da lista, dois ponteiros: um ponteiro cabeça para uma lista

ligada onde estarão armazenados ponteiros para pessoas associadas ao álbum em questão, e um ponteiro cabeça para a lista de páginas contidas no álbum.

- **Lista de Páginas:** Cada álbum contém um ponteiro cabeça para uma lista deste tipo. Cada nó de uma lista de páginas contém as informações referentes a uma página, e três ponteiros: para o próximo nó da lista, um ponteiro cabeça para uma lista ligada onde estarão armazenados ponteiros para pessoas associadas à página em questão, e um ponteiro cabeça para a lista de fotos contidas na página.
- **Lista de Fotos:** Cada página contém um ponteiro cabeça para uma lista deste tipo. Cada nó de uma lista de fotos contém as informações referentes a uma foto, e dois ponteiros: para o próximo nó da lista e um ponteiro cabeça para uma lista ligada onde estarão armazenados ponteiros para pessoas associadas à foto em questão.
- **Lista de Pessoas:** Armazena todos as pessoas introduzidos no sistema, uma em cada nó. O ponteiro cabeça de lista é uma variável global.
- **Lista de Pessoas Associadas a um Álbum:** Cada álbum contém um ponteiro cabeça para uma lista deste tipo. Cada nó desta lista armazena um ponteiro para um nó da Lista de Pessoas.
- **Lista de Pessoas Associadas a uma Página:** Cada página contém um ponteiro cabeça para uma lista deste tipo. Cada nó desta lista armazena um ponteiro para um nó da Lista de Pessoas.
- **Lista de Pessoas Associadas a uma Foto:** Cada foto contém um ponteiro cabeça para uma lista deste tipo. Cada nó desta lista armazena um ponteiro para um nó da Lista de Pessoas.

Esta organização hierárquica de listas permite uma maior eficiência na edição/consulta dos atributos de cada objecto. As listas de fotos de uma página poderá ser organizada por ordem crescente ou decrescente de datas e por ordem alfabética ou alfabética inversa de nomes. A lista de pessoas e as listas de associação de pessoas a um álbum, página ou foto serão ordenada alfabeticamente. Para tal, é necessário que os métodos das classes que adicionem elementos às listas localizem a posição de inserção ordenada antes de proceder à adição.

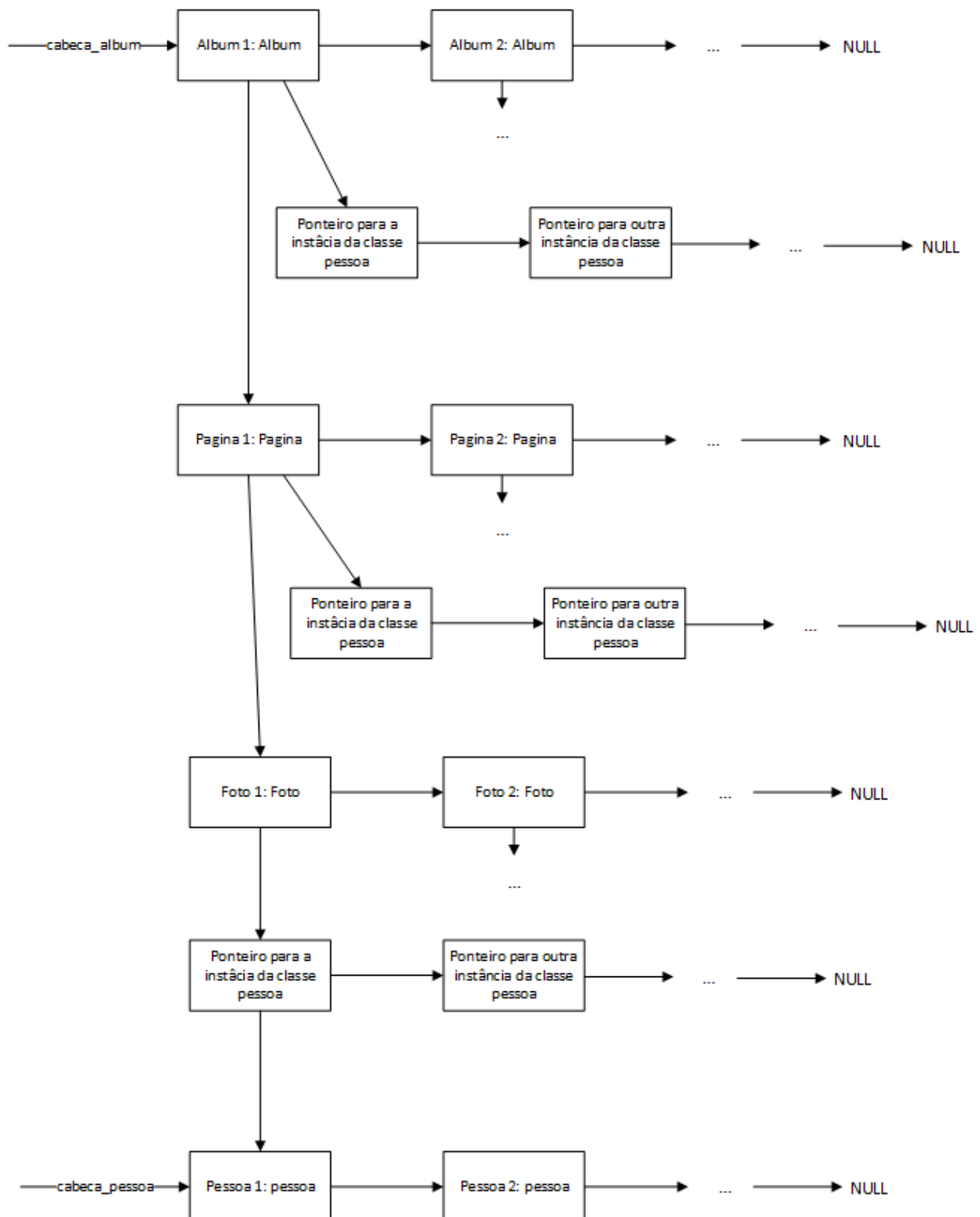


Fig 24: Esquemático da estrutura de dados escolhida

4.4.2 Implementação de Métodos

Iremos agora mostrar os algoritmos de implementação de funções globais e métodos de classes mais prioritários. Primeiramente são abordadas funções globais do sistema, seguindo-se a implementação de métodos específicos de cada classe principal.

Funções Globais

Para pesquisar fotos por palavras-chave, pessoas e por data criámos os seguintes algoritmos. A função pesquisar é uma função global e o algoritmo apresentado tenta descrever a acção de pesquisar fotos recursivamente em toda a árvore de directórios.

Algoritmo 1 Pesquisar

```
1: procedure Pesquisar(QString palavra)
2:   for all album in ListaDeAlbuns
3:     return(album.pesquisarAlbum(palavra))
4: end procedure
```

Algoritmo 2 Album::pesquisarAlbum

```
1: procedure pesquisarAlbum(QString palavra)
2:   for all pagina in ListaDePaginas do
3:     return(pagina.pesquisarPagina(palavra))
4: end procedure
```

Algoritmo 3 Pagina::pesquisarPagina

```
1: procedure pesquisarPagina(QString palavra)
2:   for all foto in ListaDeFotos do
3:     if palavra == foto.nomeFoto then
4:       return(&foto)
5:     if palavra == foto.dataFoto then
6:       return(&foto)
7:     else
8:       return(foto.pesquisarFoto(palavra))
9: end procedure
```

Algoritmo 4 Foto::pesquisarFoto

```
1: procedure pesquisarFoto(QString palavra)
2:   for all pessoa in ListaDePessoasAssociadasFoto do
3:     if palavra == pessoa.nomePessoa then
4:       return(this)
5:     else
6:       return(null)
7: end procedure
```

Algoritmo 5 Ficheiros::inicializar

```
1: procedure inicializar()
2:
3: open ficheiro pessoas
4:   if open succeeds then
5:     for all informacao in ficheiro pessoas do
6:       Pessoa pessoaAux(informação do ficheiro);
7:       pessoaAux.adicionarPessoa();
8:     end for
9:   end if
10:
11: open ficheiro albuns
12:   if open succeeds then
13:     for all informacao in ficheiro albuns do
14:       Album albumAux(informação do ficheiro);
15:       albumAux.adicionarAlbum();
16:       for all pessoa in ListaDePessoas
17:         if informação do ficheiro == pessoa.nomePessoa then
18:           albumAux.associarPessoa.Album(&pessoa)
19:         end for
20:       end for
21:       for all album in ListaDeAlbuns do
22:         open ficheiro paginas
23:         if open succeeds then
24:           for all informacao in ficheiro paginas do
25:             Pagina pag1(informação do ficheiro);
26:             album.adicionarPagina(&pag1);
27:             for all pessoa in ListaDePessoas
28:               if informação == pessoa.nomePessoa then
29:                 pag1.associarPessoaPagina(&pessoa)
30:               end if
31:             end for
32:           end for
33:           for all pagina in ListaDePaginas do
34:             open ficheiro fotos
35:             if open succeeds then
36:               for all informação in ficheiro fotos do
37:                 Foto f1(informação do ficheiro);
38:                 pagina.adicionaFoto(&f1);
39:                 for all pessoa in ListaDePessoas
40:                   if informação ==
41:                     pessoa.nomePessoa then
42:                       f1.associarPessoa(&pessoa)
43:                     end if
44: end procedure
```

O algoritmo 5 é usado para inicializar o sistema criando os sete tipos de listas da figura 24.

Repare-se que estes cinco algoritmos simplificam as tarefas complexas que visam tratar. Por exemplo, no algoritmo de inicialização, são chamados métodos de classes para criar objectos e depois outros métodos para preencher as listas.

Uma vez que escolhemos uma estrutura de dados baseada em listas ligadas decidimos não escrever os algoritmos para todos os métodos porque achamos redundante descrever construtores por cópia, como é o caso do método `Album::colarPágina(*Pagina)`, ou métodos de modificar dados, como por exemplo o método `Foto::setNomeFoto(QString)`. A vantagem da escolha de uma estrutura de dados baseada em listas ligadas é a facilidade de encontrar e definir métodos que adicionem, eliminem, modifiquem ou encontrem nós nas listas.

4.4.3 Organização do Código Fonte

Iremos especificar os tipos, nomes e formatos de ficheiros mantidos pelo sistema global, bem como as dependências entre estes.

Para facilitar a compreensão da organização dos ficheiros, são apresentados diagramas de componentes dos ficheiros referentes aos módulos Sistema Principal, Sistema de Armazenamento e GUI.

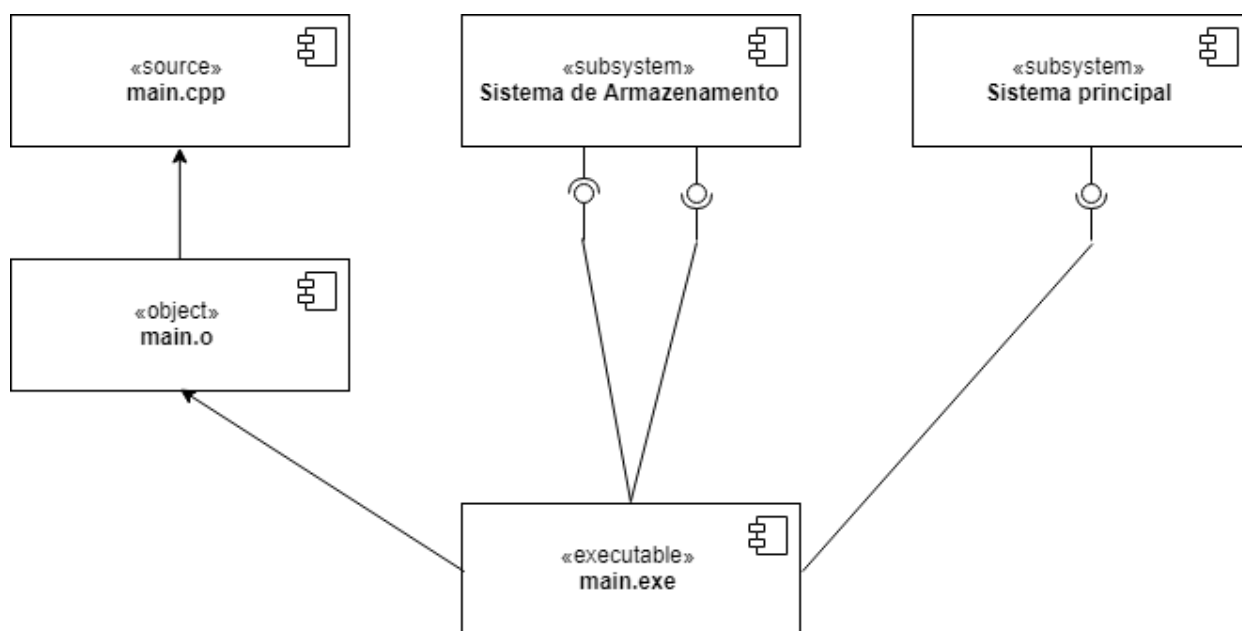


Fig 25: Diagrama de componentes que representa a organização dos ficheiros em Sistema Principal e Sistema de Armazenamento

Os conjuntos de ficheiros principais são:

- **No Sistema de Armazenamento**

Para gerir o sistema principal, serão mantidos quatro tipos de ficheiros essenciais em formato binário, um tipo diferente para cada classe. Cada um destes ficheiros é mantido num directório pertinente para reflectir a estrutura dos componentes cujas informações guarda. Por exemplo, só existirá um ficheiro para todos os álbuns, e dentro de cada pasta de álbum existirá um ficheiro para todas as páginas desse álbum. Dentro de cada pasta que corresponde a uma

página existe um ficheiro que guarda as informações de todas as fotos dessa página. Para a lista de pessoas, só é necessário existir um ficheiro.

Todos estes ficheiros armazenarão a informação relativa a todas as instâncias das estruturas de dados homónimas. A figura 25 apresenta uma perspectiva mais detalhada da organização dos ficheiros de armazenamento.

Sempre que um novo objecto for criado e armazenado numa das listas ligadas das estruturas de dados do programa, sempre que os dados de um objecto forem alterados, e sempre que um objecto for eliminado da estrutura de dados, o programa irá atualizar os ficheiros binários pertinentes.

- **Ficheiros de código fonte, header files e ficheiros objecto**

O código fonte principal é o "main.cpp", o ficheiro objecto correspondente é o "main.o". Os ficheiros código fonte e header files respeitantes à definição de classes e métodos recebem os seus nomes dessas mesmas classes.

- **Ficheiro Executável**

O ficheiro "main.exe" correrá a aplicação final.

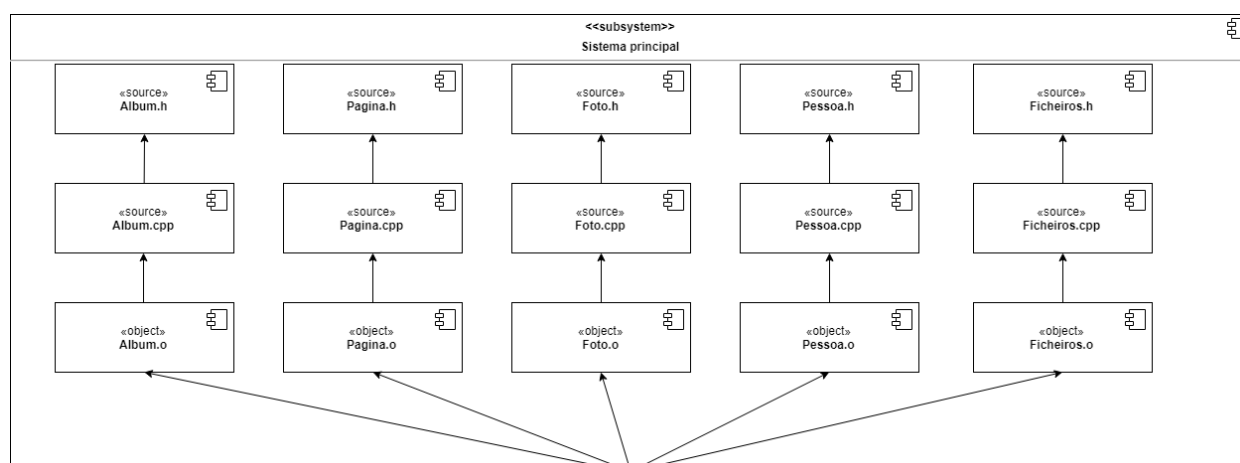


Fig 26: Diagrama de componentes que representa a organização dos ficheiros em Sistema Principal

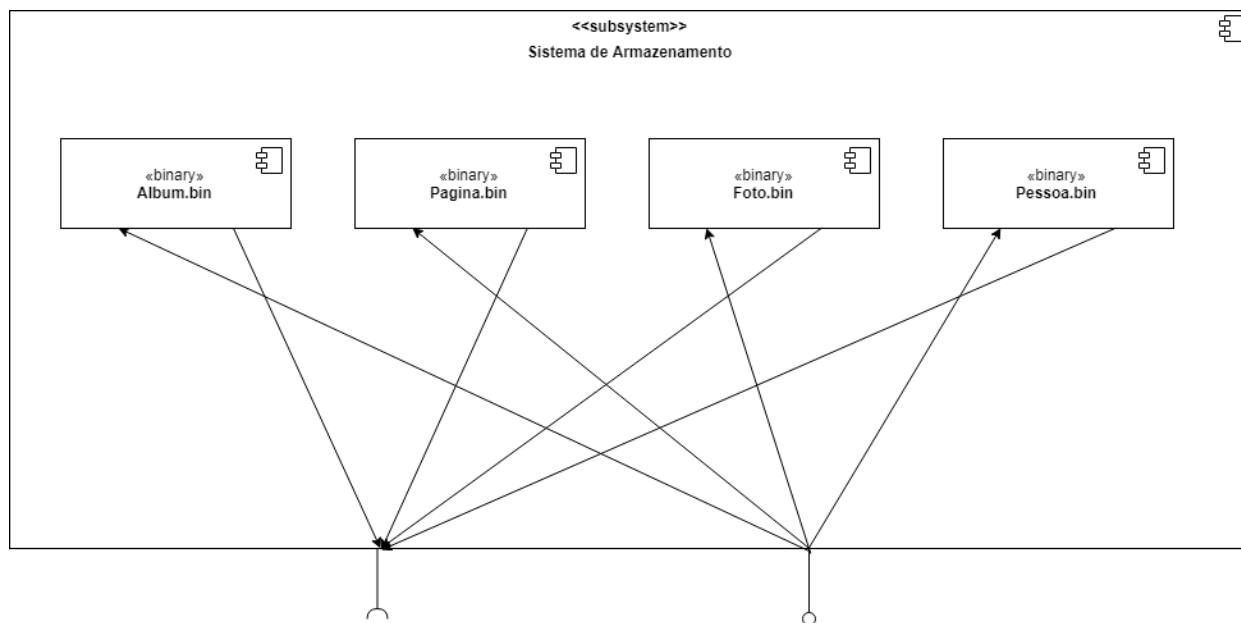


Fig 27: Diagrama de componentes que representa a organização dos ficheiros em Sistema de Armazenamento

5 Conclusão

O *design* da arquitectura permite-nos concetualizar todo o processo de desenvolvimento. Com a consolidação de requisitos confirmámos que a equipa e o *costumer* estão sintonizados em termos das funcionalidades que o programa deverá apresentar. O *design* concetual permite detalhar as funcionalidades de uma maneira mais ilustrativa. Pretende, mais precisamente, mostrar ao cliente como serão implementadas as funcionalidades desejadas, comparando-as com outro tipo de soluções abordadas pela equipa de desenvolvimento. Demonstra também o tipo de navegabilidade que o programa terá, de maneira a explicitar as opções de caminhos possíveis dentro na interface gráfica. O *design* técnico contém, já, a formulação do problema a um nível técnico mais detalhado. Deste modo, desenvolve a solução adoptada ao nível das estruturas usadas e delinea os mais importantes procedimentos que permitem as funcionalidades de maior prioridade. Esta última parte tem também que ser clara e informativa o suficiente para, caso este documento seja fornecido a uma outra equipa de desenvolvimento, esta seja capaz de criar um programa que vá de encontro ao que o cliente necessita, sem ser preciso explicações adicionais.

Uma boa projecção do *design* da arquitectura facilita o desenvolvimento do projecto nas fases seguintes. Na verdade, este documento deverá ser seguido, com os devidos ajustes, durante a implementação da arquitectura.

Referências

- [1] R. P. Rocha, *Projecto de Engenharia de Software 2017/2018 - Descrição Informal de Requisitos*, Disciplina de Engenharia de Software, Departamento de Engenharia Eletrotécnica e de Computadores, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 2018
- [2] R. P. Rocha, *Projecto de Engenharia de Software 2017/2018 - Etapa 1 - Análise de Requisitos e Especificação em UML*, Disciplina de Engenharia de Software, Departamento de Engenharia Eletrotécnica e de Computadores, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 2018
- [3] Shari L. Pfleeger & Joanne M. Atlee, *Software Engineering: Theory and Practice, 4th edition*, Pearson Prentice Hall, ISBN10: 0136061699, ISBN13: 978-0136061694, 2010
- [4] Martin Fowler, Addison-Wesley, *UML Distilled Third Edition: A Brief Guide to the Standard Object Modeling Language*, 3rd ed./5th printing, Pearson Education, 2003 (reimp. 2004). ISBN 0-321-19368-7.