

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ENGENHARIA DE SOFTWARE

MESTRADO ENGENHARIA INFORMÁTICA E COMPUTADORES

**Trabalho Prático**

***Docente:***

Luís Morgado

***Autor:***

Grupo 4

Bernardo Costa nº 42165

Duarte Felício nº 42197

Ivo Pereira nº 42172

Índice

[Introdução 3](#_Toc27776366)

[Arquitetura 4](#_Toc27776367)

[Arquitetura Logica 3 Camadas 4](#_Toc27776368)

[MVP 4](#_Toc27776369)

[Serviços 4](#_Toc27776370)

[Controlador 5](#_Toc27776371)

[Vista 6](#_Toc27776372)

[Base de Dados 7](#_Toc27776373)

[Domínio 8](#_Toc27776374)

[Testes 9](#_Toc27776375)

[Conclusão 10](#_Toc27776376)

# Introdução

Este documento descreve toda a perspetiva do domínio de solução para um sistema automatizado de Gestão de Projetos de Financiamento para apoio à criação de redes de investigação e desenvolvimento (I&D). No entanto, é apresentado um protótipo do objetivo a funcionar, o que significa que é meramente para realizar todos os casos de utilização possíveis no sistema. Começa com a descrição da arquitetura global implementada no sistema, para um panorama geral da solução criada. De seguida, vão ser aprofundados os aspetos mais relevantes implementados neste protótipo para melhor compreensão do mesmo.

Nesta fase, foi usado um padrão de desenho o Model View Presenter (M.V.P), na linguagem C#, para descrever o comportamento do mesmo e ser um interação mais fluida ao utilizar, de forma simplista. Todas as funcionalidades previstas foram desenvolvidas tais como:

* Aceitação de Candidatura;
* Abertura de um projeto;
* Parecer Técnico;
* Despacho da Comissão de Financiamento;
* Suspensão de projeto;
* Reativação de projeto;
* Reforço de projeto;
* Realização de pagamento;
* Alteração de Dados de projeto;
* Relatório de Informação de projeto;
* Relatório de pagamento de projeto;

Todas estes casos de utilização têm um ou mais utilizadores que os conseguem realizar, dependendo do seu tipo (Técnico, Gestor de Financiamento ou Comissão de Financiamento) e do estado em que se encontra o projeto. Foi também adicionado um caso de utilização extra, que é a Autenticação no sistema, para qualquer tipo de utilizador, pois faria sentido para conseguir interpretar o utilizador que estava a interagir com o sistema no momento. Esta fase não está realizada de forma ótima, visto que contem aspetos que poderiam ser melhorados de forma a praticar boas prática em engenharia de software, onde serão evidenciadas neste documento.

# Arquitetura

Neste capítulo será descrito de que forma decidimos desenhar a solução para o problema proposto. Será descrita a arquitetura da nossa solução, ou seja, a forma como decidimos organizar o nosso sistema nos seus diferentes componentes, e de que maneira esse componentes se relacionam.

## Arquitetura Logica 3 Camadas

Podemos estruturar o problema em três partes, que correspondem às três camadas da nossa arquitetura. A camada de apresentação que tem como propósito definir e controlar a interface do sistema com o utilizador; A camada de domínio que define o modelo do domínio no sistema lógico; A camada de acesso a dados que define a ligação da aplicação com a base de dados.

## M.V.P.

Decidimos utilizar o padrão M.V.P. que consiste em três partes, o Modelo, a Vista e os Apresentadores (*presenters*). Neste padrão as Vistas são passivas e definem o que o utilizador vê, pelo que são controladas pelos apresentadores que definem o que irá acontecer em cada evento. De seguida os apresentadores atualizam o modelo e tratam também dos eventos recebidos deste modelo que por sua vez afetam de que forma a vista será atualizada.

O nosso modelo foi criado com o auxílio da ferramenta .NET Entity Framework que é um ORM (object-relational mapper) que tem por objetivo facilitar a adaptação entre o modelo de domínio e o modelo de dados. Este ORM gerou os nossos objetos de domínio que representam o modelo.

## Serviços

Para aceder aos dados da aplicação na base de dados utilizámos uma camada de serviços que define um contrato entre os módulos que pretendem aceder aos serviços e os módulos que irão fornecer esses serviços. Estes serviços estão a representar o acesso a dados, pelo que os módulos que precisem podem fazer pedidos aos serviços e recebem como resposta dados provenientes da base de dados.

Na nossa aplicação os serviços estão divididos em três partes, o gerador de ids de projeto que tem como objetivo a criação de ids numa estrutura definida para projetos. Este serviço de geração de Ids sabe que Ids já foram utilizados e gera um Id adequado e disponível quando lhe é requisitado. O serviço de autenticação cujo objetivo é autenticar um utilizador com a sua password e retornar as informações do utilizador para ser possível efetuar o login. Os serviços CRUD possibilitam ações de criação, leitura, atualização e eliminação sobre a base de dados. Estes serviços definem através de métodos que ações podem ser realizadas sobre a base de dados pela aplicação.

# Controlador

Com o uso do padrão M.V.P este componente é chamado de Controlador Supervisor pois, tem como objetivo ter grande parte do controlo do sistema tanto das vistas como modelo. Quando existe um evento (clique num botão, lista, etc…) na vista este é informado dessas ações, para realizar uma alteração no domínio e gerir toda a parte visual. Este componente é usado como ligação entre o acesso a dados, vista e domínio pois, comunica com todos os componente cruciais no sistema.

Os Apresentadores são todos do tipo Apresentador, que por sua vez têm uma vista associada e argumentos (que representam informação da vista). Para além disso, são também do tipo Notificável para poderem subscrever eventos vindos da vista, ou seja, quando acontece um evento na vista este é informado que tal ação aconteceu, para poder reagir a tal e alterar determinados dados do domínio.

Existem Apresentadores genéricos para que, quando existem Apresentadores bastante semelhantes e com vistas bastante similares estes estendem de um Apresentador genérico, pois todos têm características bastante semelhantes. Como por exemplo quando é para listar projetos, existem várias vistas que usam este caso de utilização mas com pequenos pormenores diferentes, então nesse caso é mais viável ter um Controlador genérico em que todos que têm características similares a esse estendem do mesmo.

# Vista

O propósito da vista é de servir de ligação entre o utilizador e o sistema. Como grande maioria das vistas contêm campos que o utilizador tem que preencher obtámos por utilizar a biblioteca Form da linguagem C# de forma a facilitar a interatividade entre as partes.

Esta biblioteca fornece a sua componente gráfica que permite customizar a vista adicionando campos como um *TextBox* que permite escrever texto ou um botão para clicar. Também fornece a sua componento programática que contém os métodos que são chamados após, por exemplo, um click num botão.

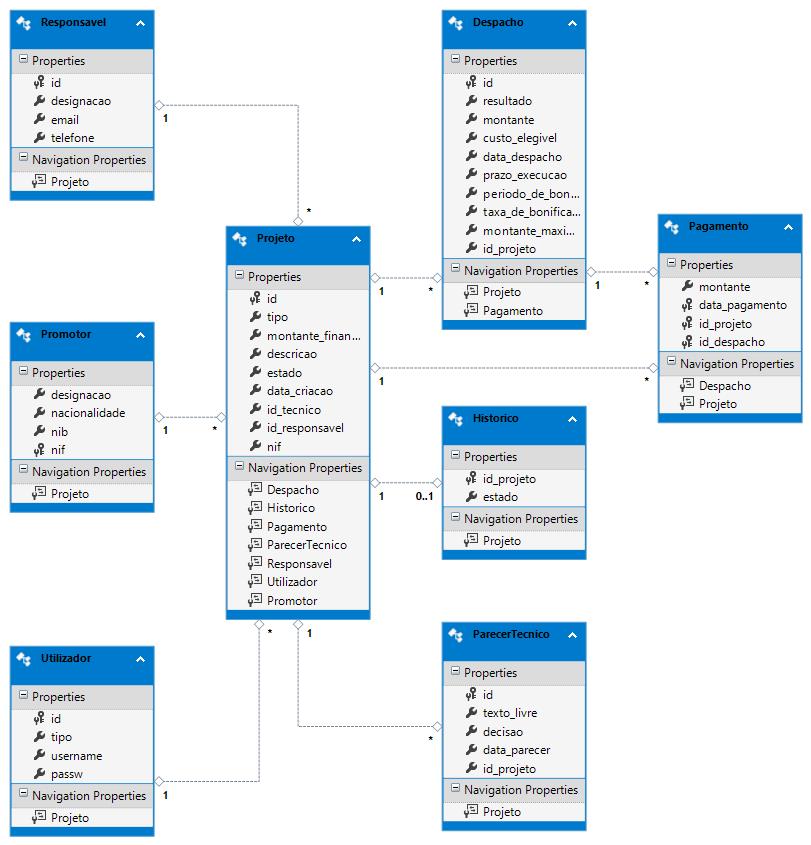
Todas as vistas realizadas (*Forms*) estendem da classe Vista que por sua vez estende da classe Form. Estas vistas contêm um objeto do tipo notificável de forma a notificar todos os possiveis ouvintes (no contexto da aplicação consiste nos apresentadores) que aconteceu certo evento.

As vistas também contêm um tipo genérico que estende de EventArgs, esse tipo é estendido por outras classes de forma a passar os campos da vista para os seus ouvintes. Por exemplo a classe LoginArgs estende de EventArgs e contém os campos username e password para autenticar o utilizador.

Todos os casos de utilização necessitam de no minímo uma vista, uma classe de argumentos e de um apresentador, mas, como vários casos de utilização necessitam de vistas semelhantes foi possível reutilizar a mesma vista em apresentadores diferentes.

# Base de Dados

Criámos o seguinte modelo lógico da base de dados:



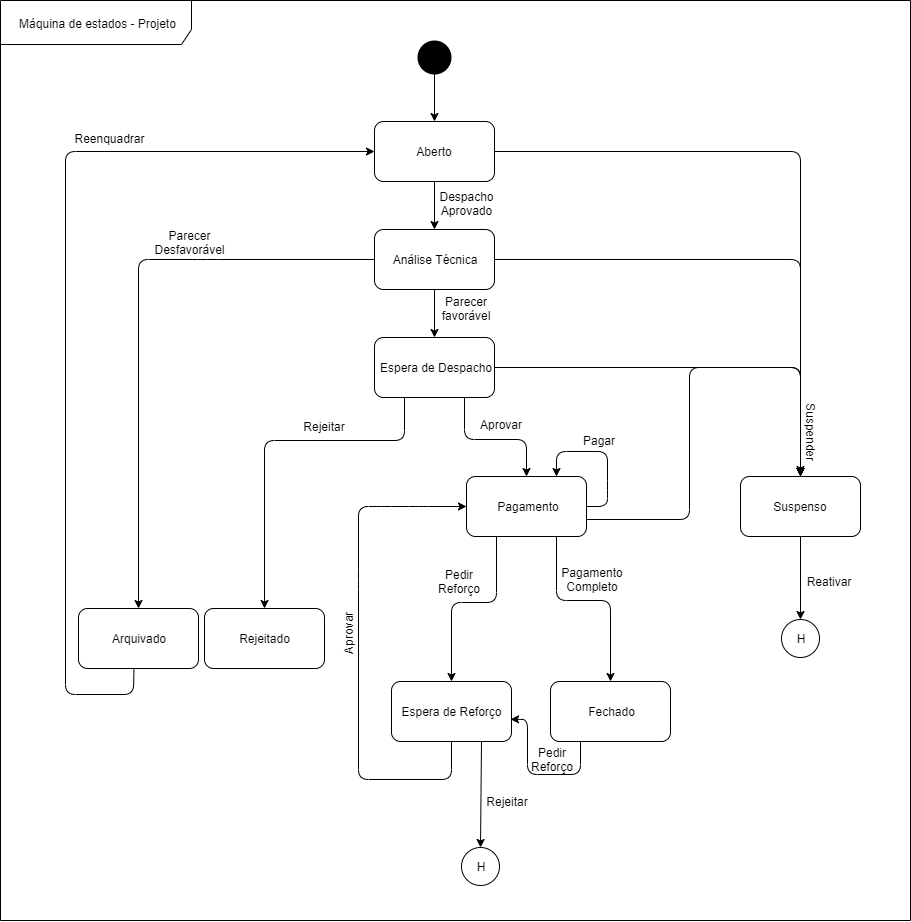
A tabela historíco serve para termos o registo de que estado o projeto se encontrava antes de ele ser ou suspenso ou pedido um reforço de financiamento de forma a ser possível o seu estado anterior após reativação ou despacho.

Este modelo lógico pode no entanto ser melhorado tirando o campo tipo do projeto e adicionando duas novas tabelas: Incentivo e Bonificação em que ambas estendem de Projeto fazendo com a tabela Pagamento apenas estivesse ligada a um projeto do tipo Incentivo.

A mesma lógica também pode ser aplicada na tabela utilizador de forma a separar cada tipo de utilizador podendo depois adicionar a cada um os estados de projetos que cada utilizador tem acesso.

# Domínio

De forma a realizar as transições de estado de um Projeto sem qualquer entropia criámos uma máquina de estados com o seguinte modelo de dinâmica:

****

Esta máquina de estados pode ser melhorada adicionando uma transição do estado Espera de Despacho diretamente para Fechado no caso de o projeto ser do tipo Bonificação.

# Testes

Os testes realizados para confirmação do bom funcionamento do sistema não foram os mais indicados, pois foram realizados manualmente. Ou seja, estes foram testados ao correr toda aplicação e fazendo se passar por um utilizador e confirmar se o que aconteceu no sistema era o suposto. No entanto, numa fase final deste projeto serão realizados testes unitários para confirmar o bom funcionamento de todos os componentes do sistema.

As frameworks mais úteis para conseguir realmente testar, bastante parte do comportamento do sistema seriam:

* TestTools.UnitTesting, usada para os serviços e regras de negócio
* Code UI test, para testar as componentes da vista de cada utilizador.

Assim toda a implementação do sistema, poderia ser publicada sem tanta preocupação a bugs, que poderiam acontecer no sistema, pois da forma como está realizado não há essa garantia.

# Conclusão

Em conclusão, realizámos um sistema cumprindo uma arquitetura definida, mas com módulos que poderão ainda ser alvo de melhorias, nomeadamente o modelo lógico de dados, e algumas incertezas quanto à arquitetura de certos módulos.

Estas falhas resultam de uma dificuldade em interpretar corretamente os requisitos do cliente devido a uma entropia lá presente. Resultaram também problemas devido a falta de conhecimento de qual a melhor solução para alguns problemas na altura em que nos foram colocados.