Documento de Design de Software para Gerenciador de Vídeos

Daiane Macedo, Hugo Carneiro, Thiago Ururay

December 12, 2019

1 Introdução

O objetivo desse documento é propriamente detalhar, organizar e analisar o processo de desenvolvimento de um software com o objetivo de gerenciar e exibir vídeos em um computador ou dispositivo móvel, determinando seus casos de uso e fornecendo uma visão arquitetural do sistema, bem como padrões de projeto utilizados. O escopo do projeto foi feito levando em conta os requisitos funcionais e não funcionais determinados pelo professor, bem como análises de caso de uso.

2 Visão do Sistema

As funcionalidades que procuramos implementar no sistema incluem acesso a um banco de dados para os usuários e para os vídeos, operações CRUD para usuários com propriedades de administrador e operações de consulta a arquivos através de tags tais como título, duração, etc. O projeto foi implementado em Python3 utilizando framework Django para o desenvolvimento em geral, Amazon Simple Storage Service para repositório e Heroku como uma plataforma em nuvem.

2.1 Design Arquitetural

O estilo arquitetural empregado no projeto foi o MVC (Model View Controller), mais especificamente uma variante utilizada na filosofia de implementação do framework Django chamada MTV (Model Template View) [3]. As três partes principais que compõem a arquitetura Django são:

- Um conjunto de ferramentas de trabalho para dados e SGBDs.
- Um sistema de templates que possa ser utilizado por leigos.
- Um sistema que permita comunicação entre a interface de usuário e o banco de dados, automatizando processos envolvidos com a gerência de um site.

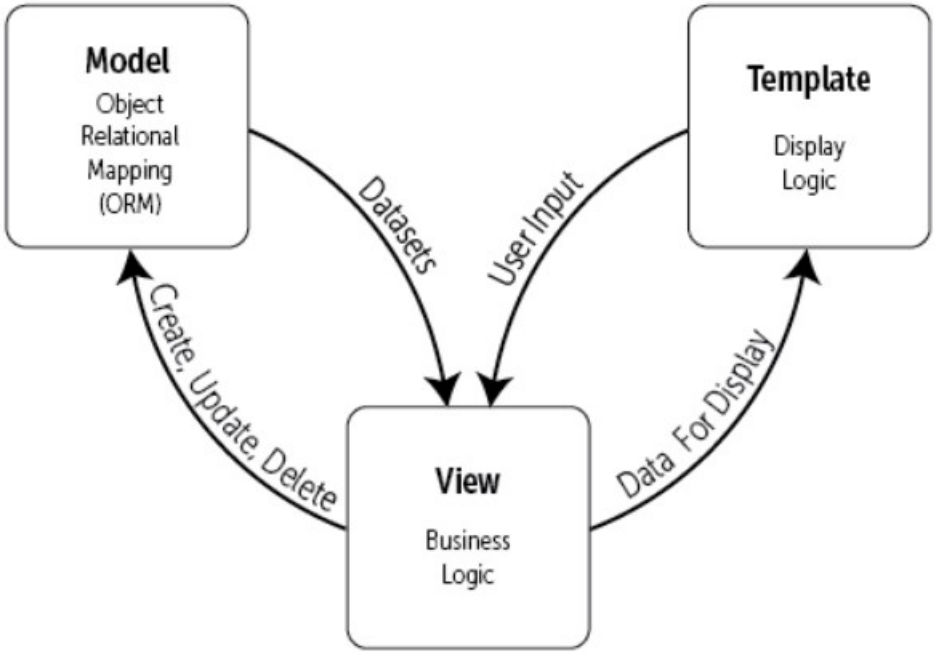

Figure 1: Visão do Modelo MTV.

Esse modelo de implementação permite o baixo acoplamento e organização do projeto, especialmente quando consideramos o objetivo da introdução do componente Template. O propósito é que ele seja o responsável pela comunicação do cliente que utiliza a aplicação, renderizando e recebendo informações da rede. Isso aumenta a facilidade de desenvolvimento dele uma vez que é necessário apenas conhecimentos em HTML e CSS [2].

Alinhados com essa visão, nós oferecemos ao cliente a visão de Template, e as requisições são gerenciadas pelo View que por sua vez se comunica com Model, que acessa ao banco de dados. Abaixo fica demonstrado o diagrama de classes pertinente ao projeto, de forma a esclarecer a composição feita. Embora a filosofia do uso de interfaces tenha sido empregada durante o desenvolvimento, buscando modularização dos métodos para aumentar alta coesão e baixo acoplamento entre classes, a linguagem Python não possui uma interpretação clara do conceito, por isso preferimos não fazer essa representação aqui.

Note que toda a lógica que não possui retorno direto para o cliente fica separada do View, responsável por retornar apenas, de acordo com a visão arquitetural, as consultas ou alterações realizadas pelo cliente e principalmente o mapeamento da URL que é o retorno para o Template, atualizando a informação na tela. Portanto, garantimos não apenas que as responsabilidades de cada função sejam claramente delimitadas para cada componente desenvolvido, mas que haja coesão entre as classes utilizadas e um nível de acoplamento adequado.

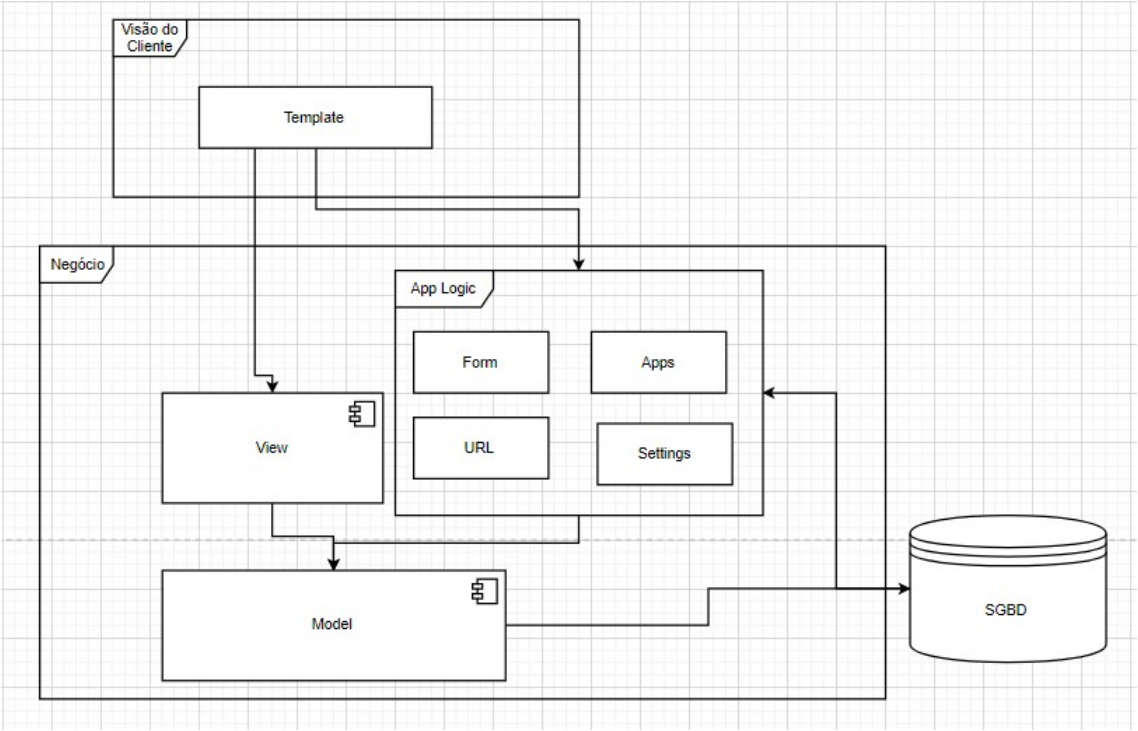

Figure 2: Visão arquitetural do projeto

2.2 Análise de Requisitos e Casos de Uso

Para o desenvolvimento do projeto tomamos os requisitos funcionais e não funcionais descritos na orientação do professor, conforme a tabela abaixo.

Requisitos Funcionais	
RF1	O sistema deve ser capaz de exibir vídeos de um catálogo de vídeos.
RF2	O sistema deve permitir o cadastro, remoção, atualização e consulta de vídeos no catálogo (CRUD). Somente usuários administradores podem cadastrar, remover e atualizar os vídeos do catálogo.
RF3	O sistema deve retornar uma lista de vídeos com links e prévias como resultado da consulta.
RF4	O sistema deve permitir que usuários criem uma conta contendo dados pessoais em seu cadastro.
RF5	O sistema deve permitir que usuários com conta possam manter uma lista de favoritos.
Requisitos Não Funcionais	
RNF1	O sistema deve estar disponível na Web.
RNF2	O catálogo deve ser armazenado em um banco de dados.

Também com o intuito de detalhar claramente nossa visão arquitetural do projeto, detalhamos os casos de uso com relação a ambos os tipos de usuário, o usuário comum, ao qual chamaremos de cliente, e o com privilégios em relação ao banco de dados, denominado administrador.

Para detalhar ainda mais e ter a perfeita clareza da metodologia empregada na implementação do projeto, também foi desenhado um diagrama de sequência, no

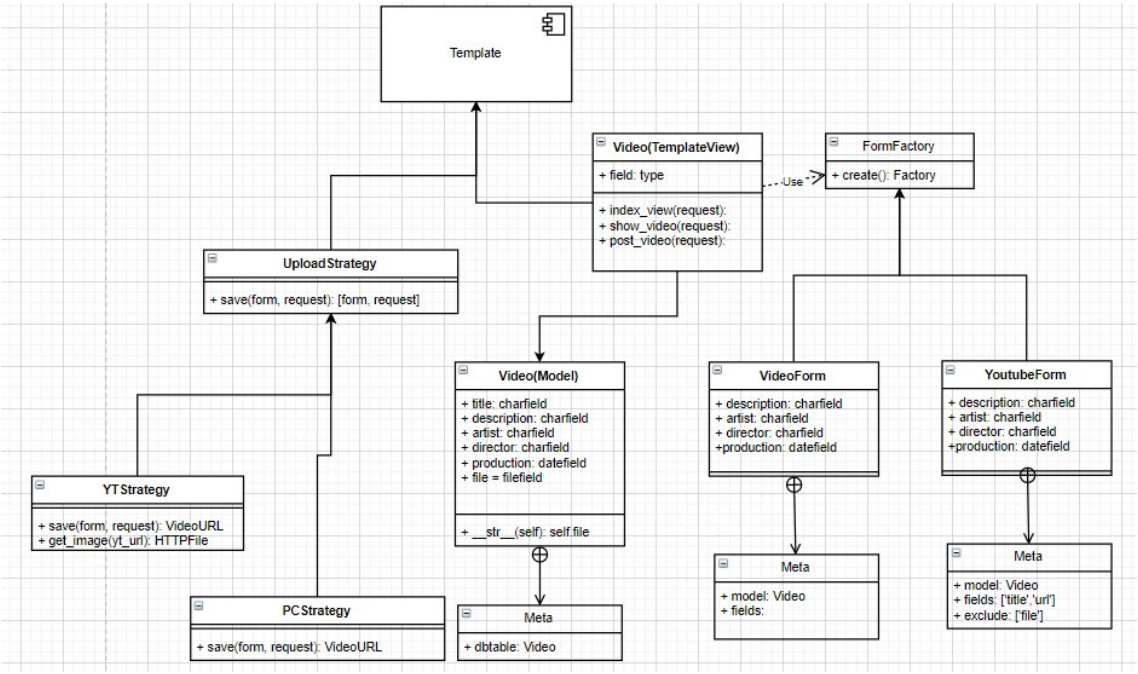

Figure 3: Diagrama de Classes do projeto

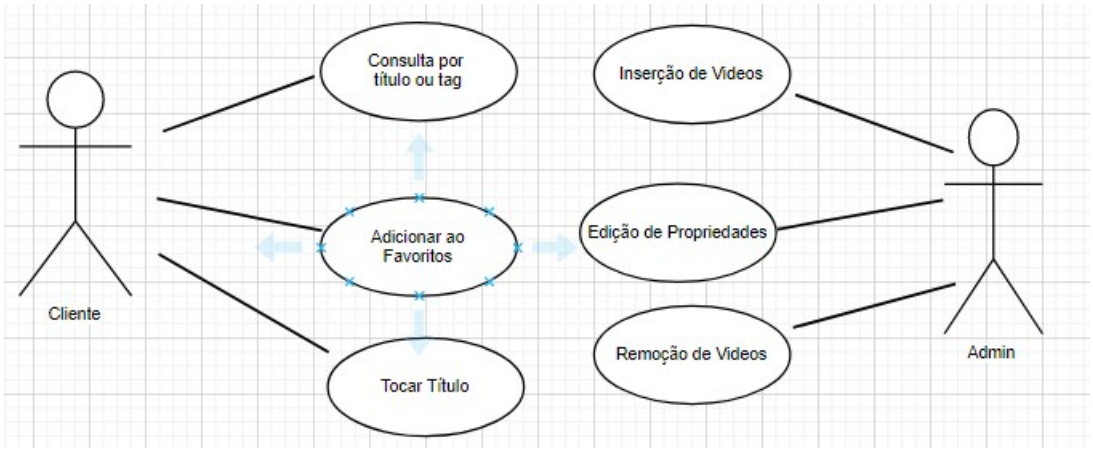

Figure 4: Definição dos Casos de Uso do Software

caso exemplificando o seguinte caso de uso em que um usuário realiza uma consulta, que retorna uma lista [4].

A partir da construção desses modelos foi que partimos para a implementação em si, na seguindo o padrão MTV, também procuramos adotar outros padrões já naturalmente implementados pelos frameworks utilizados.

3 Padrões de Projetos

Procuramos durante toda a implementação já seguir os padrões recomendados pelos frameworks, guiando o desenvolvimento do projeto. Conforme requisitado pelo professor, além do padrão MTV também adotamos dois padrões, que foram encapsulados dentro do MTV visando aumentar escalabilidade e organização do projeto.

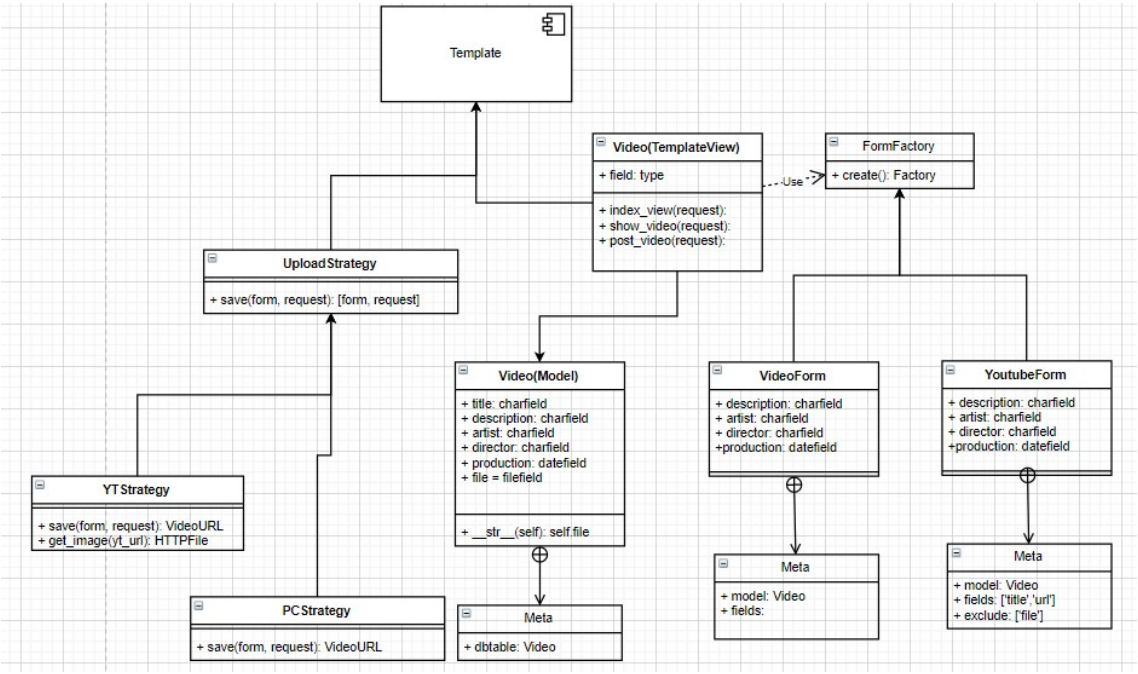

Figure 5: Diagrama de Sequência, demonstrando uma consulta a uma lista de títulos

3.1 Decorator

A filosofia empregada nesse padrão é desenvolver visões genéricas baseadas em classe e através dos decoradores realizar a implementação de outras funcionalidades de forma dinâmica, evitando assim excessiva complexidade no código.

Neste caso, encapsulado no próprio modelo MTV, utilizamos o padrão Decorator em conjunto com a classe View, que necessita de flexibilidade para diferentes tipos de requisições GET e POST permitidas para o usuário. Para isso, utilizamos decoradores, mais especificamente o pacote django.views.decorators para a implementação de funcionalidades na aplicação HTTP, visando abordar principalmente questões de segurança e gerenciamento das requisições enviadas ao banco de dados [1].

3.2 Factory e Strategy

Escolhemos aplicar o padrão Factory na criação de formulários, para lidar com os diferentes formatos de implementação para vídeos de diferentes plataformas. No caso, exemplificamos padrões para upload comum de vídeos da plataforma Youtube e vídeos em formato "padrão".

Como na implementação em linguagem Python não há um conceito bem definido de interface, optamos por encapsular o desenvolvimento dentro do MTV. Neste caso, a responsabilidade da chamada da classe Create é responsabilidade da View, que por sua vez também age chama o controlador por trás da estratégia de upload e escolha do formulário aplicado.

É através do uso do padrão Strategy que também fazemos uma escolha de implementação para diferentes métodos que upload, que também devem mudar de acordo com diferentes plataformas. Assim nós deferimos a instanciação da subclasse Form, para um controlador que por sua vez varia a composição do upload, mantendo to-

dos os diferentes processos separados e evitando alto nível de acoplamento nos níveis mais baixos de arquitetura.

References

[1] J. Forcier, P. Bissex, and W. J. Chun, *Python web development with Django*. Addison-Wesley Professional, 2008.

[2] A. Holovaty and J. Kaplan-Moss, *The definitive guide to Django: Web development done right*. Apress, 2009.

[3] A. Holovaty, J. Kaplan-Moss *et al.*, "The django book," *http://www. djangobook. com/en/1.0*, 2007.

[4] I. Sommerville, *Software engineering*. Addison-Wesley/Pearson, 2011.