



DEETC – Departamento de Engenharia Eletrónica e Telecomunicações e de
Computadores

MEIM - Mestrado Engenharia informática e multimédia

Engenharia de Software

Projeto

Trabalho realizado por:

Duarte Domingues N°45140

Docente:

Luís Morgado

Data: 23/01/2023

Índice

1. INTRODUÇÃO	1
2. DESENVOLVIMENTO	3
2.1 ANÁLISE DE REQUISITOS.....	3
2.1.1 Documento de visão	3
2.1.2 Modelos dos casos de utilização	3
2.1.3 Especificação suplementar	6
2.1.4 Glossário	6
2.2 ARQUITETURA LÓGICA.....	6
2.2.1 Modelo de domínio.....	7
2.2.2 Padrões de organização de subsistemas	8
2.2.3 Diagramas de interação	9
2.2.4 Realização dos casos de utilização.....	9
2.2.5 Arquitetura de mecanismos	10
2.2.6 Arquitetura geral da solução	11
2.3 ARQUITETURA DETALHADA	12
2.3.1 Modelo de dinâmica	12
2.3.2 Arquitetura de teste.....	13
2.3.3 Tecnologias utilizadas	14
2.3.4 Modelo de implantação da aplicação	15
2.4 IMPLEMENTAÇÃO DO PROTÓTIPO APLICACIONAL	17
2.4.1 Base de dados	18
2.4.2 <i>Forms</i>	19
2.4.3 Páginas realizadas.....	19
ANÁLISE CRÍTICA DO PROJETO REALIZADO	24
CONCLUSÕES.....	25
BIBLIOGRAFIA	26

1. Introdução

Neste projeto foi desenvolvida uma aplicação informática, a partir de diversas técnicas e fundamentos de engenharia de software. A aplicação desenvolvida é uma aplicação Web para pesquisa e partilha de livros com amigos. A aplicação tem um conjunto de funcionalidades de como a realização de críticas e comentários, coleção de livros e um sistema de amizades.

Enquanto a tecnologia continuar a evoluir e novas gerações de computadores permitirem criar sistemas mais densos e complexos, existe uma necessidade exponencial de lidar com complexidade. Esta necessidade foi sentida intensivamente na década de 1960, um período marcado pela falha e impossibilidade de manter aplicações de elevada dimensão e qualidade. De forma a combater este problema surgiu a solução de aplicar princípios de engenharia ao desenvolvimento de software, denotada como engenharia de software.

A engenharia de software baseia-se num conjunto de princípios sólidos, metodologias e técnicas. Estes princípios permitem garantir o desenvolvimento de sistemas de software de qualidade e eficientes mesmo perante complexidade e mudança constante. Neste contexto, um conceito importante é a entropia. A entropia é uma medida de desordem ou incerteza de um sistema, é necessário minimizar a entropia ao máximo. O aumento da entropia leva a sistemas de software complicados, difíceis de entender e modificar.

O projeto realizado utilizou técnicas e ferramentas para combater a entropia. No projeto foram realizadas as seguintes etapas de forma organizada e sistemática:

- Especificação de requisitos: Esta foi a primeira etapa do projeto, permitiu definir o problema e identificar os requisitos e especificações da solução. Esta etapa foi muito importante para combater ambiguidade, e iniciar o projeto com um forte fundamento.
- Arquitetura lógica: A segunda etapa definiu uma representação abstrata do sistema. Foi definido como os diferentes elementos do sistema interagem e se relacionam entre si. Esta etapa permitiu realizar um modelo do funcionamento do sistema e das suas diferentes funcionalidades.
- Arquitetura detalhada: Nesta fase foi definida a implementação da solução e os mecanismos dos diferentes elementos do sistema. Nesta etapa foi tido em conta as plataformas aplicacionais e linguagens de programação que serviram de base da aplicação.

- Implementação: Por fim foi realizada a implementação do sistema. Nesta fase foi primeiro elaborado um protótipo de teste para verificar que o sistema foi projetado corretamente e para encontrar erros e falhas do sistema implementado.

2. Desenvolvimento

2.1 Análise de requisitos

Como mencionado na introdução o desenvolvimento da aplicação começou pela análise de requisitos. Esta etapa foi fundamental para recolher e concretizar informação para a conceção do sistema, **independentemente do modelo computacional**. Começou-se por definir a visão do problema, e acabou-se por especificar a lista de requisitos concretos do problema.

2.1.1 Documento de visão

O **documento de visão** definiu de forma geral a descrição do problema e a sua solução, tendo em conta detalhes e informação importante sobre a aplicação. Alguns destes detalhes foram por exemplo, o posicionamento do produto e o ambiente de utilização em que o problema se enquadra.

O **problema** principal foi a necessidade de uma aplicação do género de rede social exclusivamente para entusiastas de livros, tendo como foco a interação entre utilizadores e pesquisa de livros. A **solução** foi uma aplicação Web com múltiplas opções de interação entre utilizadores que permite pesquisar, criticar e catalogizar livros com uma enorme oferta de conjunto de livros.

2.1.2 Modelos dos casos de utilização

De seguida foram definidos os **modelos dos casos de utilização**. Estes modelos são uma representação da interação entre os atores e o sistema. Foram definidos mais de 10 casos de utilização principais e descrição detalhada de 7 ou mais casos.

Nesta etapa começou-se por definir os atores que vão interagir com o sistema, os atores foram respetivamente:

- Utilizador
- Administrador

Numa fase inicial do projeto também foi projetado um ator que seria o sistema de recomendação de livros. Entretanto este ator e o sistema de recomendação de livros acabou por ser abandonado de modo a ter-se um melhor foco no desenvolvimento das outras funcionalidades da aplicação.

Inicialmente começou-se por definir a **funcionalidade geral da aplicação**, que pode ser observada no seguinte diagrama. O diagrama apresenta os atores principais e as duas funcionalidades mais importantes, a interação entre utilizadores e a interação de livros.

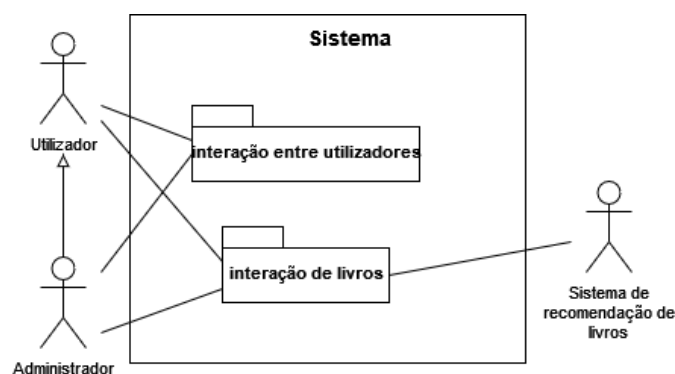


Figura 1 - funcionalidade geral da aplicação

De seguida foi realizado o diagrama das funcionalidades da **interação de livros**. Este diagrama apresenta casos de utilização essenciais como por exemplo: Pesquisar um livro, atualizar a coleção de livros e realizar uma crítica de um livro. O diagrama pode ser observado na figura seguinte:

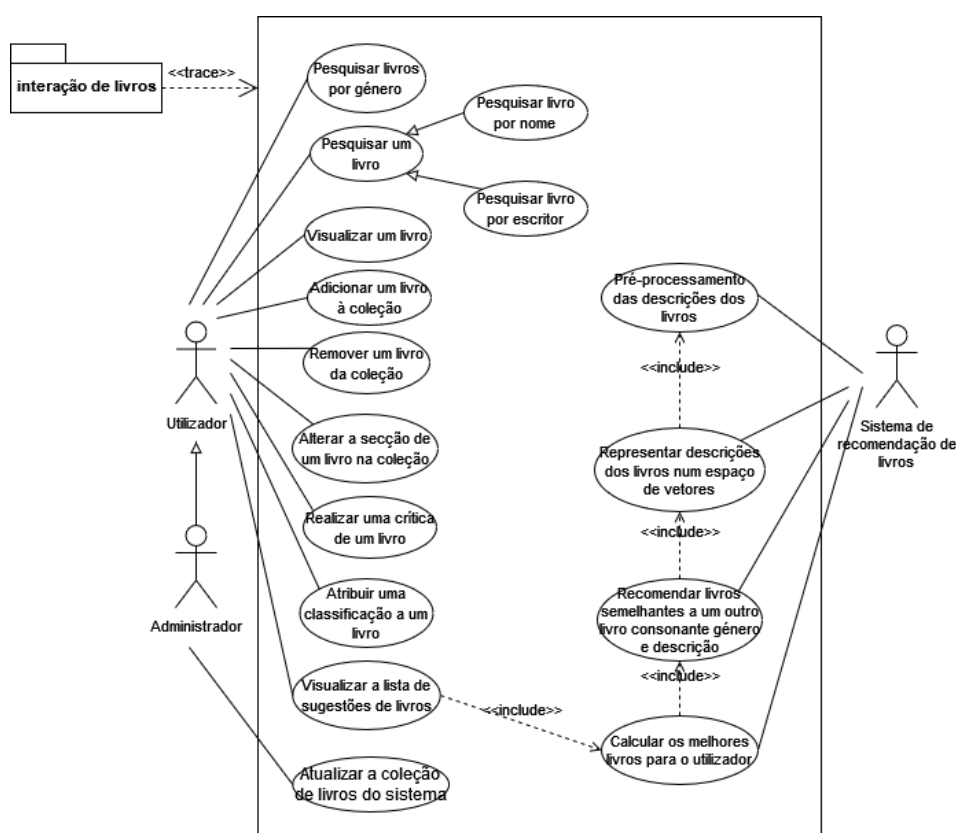


Figura 2 – Caso de utilização interação de livros

A **interação entre utilizadores** define as diferentes funcionalidades que o utilizador pode realizar no sistema, que envolvem ou são afetadas por outros utilizadores. A interação entre utilizadores pode ser observada no diagrama seguinte:



Figura 3 - Caso de utilização interação entre utilizadores

De forma a manter coesão entre diferentes casos de utilização foram definidas diversas relações entre os casos. As relações utilizadas foram:

- **Relação de inclusão:** Utilizada na modelação de um caso de utilização que é um subconjunto de outro caso.
- **Relação de extensão:** Usada na modelação de um caso de utilização que é uma extensão ou variação de outro caso.
- **Relação de generalização:** Utilizada na modelação de um caso de utilização que é uma versão generalizada de outro caso mais específico.

Foi também definida uma descrição detalhada de alguns dos casos de utilização. Foram realizados cenários principais e em alguns casos cenários alternativos dos casos de utilização, sendo também definidos os pré-requisitos necessários.

2.1.3 Especificação suplementar

Em seguida foi realizada uma **especificação suplementar** da aplicação, que consta numa lista de requisitos adicionais importantes. Um exemplo de uma especificação foi a aplicação ter de ter compatibilidade com diferentes aparelhos eletrónicos.

2.1.4 Glossário

Foi também realizado um **glossário**, o glossário é um conjunto de termos e definições que permitem especificar a linguagem utilizada na especificação de requisitos. Um exemplo de uma definição do glossário foi o termo “Classificação de um livro” que tem a descrição “Classificação de um livro entre 1 e 5 estrelas”.

2.2 Arquitetura lógica

Após a análise de requisitos, baixou-se um nível de abstração no desenvolvimento do sistema e passou-se à implementação da arquitetura lógica. A arquitetura lógica permite obter um modelo do sistema que descreve as suas funcionalidades, comportamentos e mecanismos, **independentemente da plataforma de execução**. De forma a poder realizar a descrição do modelo de forma abstrata utiliza-se a linguagem de modelação UML. Uma linguagem utilizada para descrever conhecimento acerca do domínio do problema e da solução do sistema a desenvolver.

No desenvolvimento do sistema foi crucial ter em conta as diferentes métricas de arquitetura. Um bom conhecimento e aplicação destas métricas permite minimizar a entropia e complexidade associada com o desenvolvimento de aplicações. As métricas são nomeadamente:

- **Coesão:** Traduz o nível de coerência funcional de um subsistema/módulo (até que ponto este módulo realiza uma única função).
 - Um nível de coesão baixo traz diversos problemas devido a caso seja necessário alterar um subsistema, o número de módulos afetados é muito elevado.
 - Um nível de coesão alto tem o benefício do número de módulos afetados pela alteração de um subsistema ser baixa. Por estas razões pretende-se que o nível de coesão seja alto.

- **Acoplamento:** Mede o grau de interdependência entre subsistemas. Um nível de acoplamento baixo permite obter maior facilidade de desenvolvimento, instalação, manutenção e expansão. Um grau baixo de acoplamento oferece também:
 - Melhor escalabilidade devido à possibilidade da replicação e distribuição de módulos que prestem serviços.
 - Maior tolerância a falhas devido a uma falha ter um impacto mais isolado no sistema. Pretende-se então ter um grau reduzido de acoplamento.
- **Simplicidade:** Mede o grau de facilidade de compreensão e comunicação da arquitetura. Um nível elevado de simplicidade oferece múltiplas vantagens, como por exemplo:
 - Menores custos de desenvolvimento e maior facilidade de manutenção.
- **Adaptabilidade:** Mede o nível de facilidade da alteração da arquitetura de modo a acomodar a incorporação de novos requisitos ou alterações nos requisitos previamente definidos. Um nível elevado de adaptabilidade tem como vantagens por exemplo:
 - Maior flexibilidade, o sistema é capaz de lidar com mudanças e novos requisitos, sem precisar de um número elevado de alterações.
 - Escalabilidade, o sistema é capaz de se adaptar melhor ao crescimento do número de requisitos necessários.

2.2.1 Modelo de domínio

O primeiro passo da arquitetura lógica foi o desenvolvimento do **modelo de domínio**. O modelo de domínio é uma representação das entidades do sistema a partir de classes e as relações entre elas. Esta etapa foca-se nos conceitos envolvidos no domínio do problema e não na implementação da solução.

Começou-se por definir as diferentes entidades presentes no domínio do problema, algumas destas entidades foram:

- Crítica
- Livro
- Comentário
- Informação utilizador
- Coleção

De seguida foram estabelecidas as diferentes relações das entidades entre si, por exemplo um livro pode possuir zero ou mais críticas, e uma crítica pode possuir 0 ou mais comentários.

O modelo de domínio permitiu ajudar a perceber o problema e enquadrar os conceitos das entidades no domínio do problema.

2.2.2 Padrões de organização de subsistemas

Na arquitetura do software realizado foi utilizada uma arquitetura de **três camadas**, como pode ser observado no seguinte diagrama.

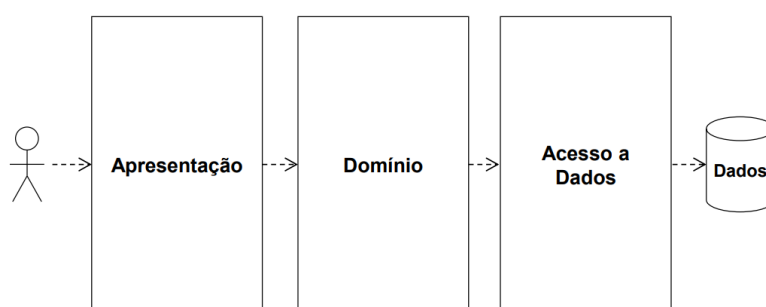


Figura 4 - Diagrama arquitetura três camadas

A camada de apresentação é responsável por gerir a interação do utilizador com o sistema. O domínio é responsável por manter a lógica da aplicação. Acesso a dados define a forma como os dados são acedidos.

Surge o problema de ter de encontrar uma forma sucinta de organizar a relação entre as classes responsáveis pela interação com o utilizador e as classes para processamento interno. A solução é o **padrão MVC (Model View Controller)**. Este padrão permite separar o modelo do domínio, o controlo de interação e apresentação em três classes distintas. Este padrão tem o modelo, vista e o controlador, como pode ser observado no diagrama seguinte, que ilustra o funcionamento do padrão:

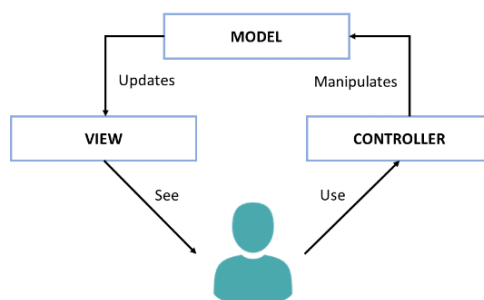


Figura 5 - padrão MVC

O **modelo** gere o comportamento dos dados do domínio da aplicação, responde a pedidos de informação sobre o seu estado e responde às instruções para alterar estado.

A **vista** é responsável por apresentar os dados ao utilizador.

O **Controlador** é responsável por gerir o fluxo dos dados entre o modelo e a vista. Contém as classes que gerem as ações dos utilizadores e que solicitam dados ao modelo.

Existem também outras variantes inspiradas no padrão MVC, como por exemplo o padrão *Model-View-Presenter* (MVP) e *Model-View-ViewModel* (MVVM).

2.2.3 Diagramas de interação

Os casos de utilização representam diagramas que descrevem os casos definidos anteriormente na análise de requisitos. Isto foi realizado através de diagramas de interação. Os diagramas de **interação** descrevem a interação entre diferentes elementos pertencentes ao sistema e/ou com o exterior do sistema. Os diagramas realizados foram diagramas de sequência, têm foco na sequência de interação entre as diferentes partes do sistema, com ênfase no tempo. Estes diagramas são compostos por componentes representados por retângulos, e por mensagens trocadas entre eles, simbolizadas por setas.

Os diagramas de **sequência** foram utilizados para representar os casos de utilização definidos anteriormente na análise de requisitos.

2.2.4 Realização dos casos de utilização

Nesta fase os diagramas de sequência foram utilizados para representar os casos de utilização definidos anteriormente na análise de requisitos. Foram realizados múltiplos diagramas de sequência. Os diagramas de sequência permitem estabelecer as relações do padrão MVC entre as múltiplas classes.

Alguns dos padrões do domínio importantes que foram utilizados nesta fase foram o repositório e entidade. Os **repositórios** geram o armazenamento de domínios, possuindo opções de: inserir, obter, atualizar e remover. **Entidades** são identidades próprias, que se mantêm ao longo da operação do sistema, mantendo estado.

O exemplo seguinte, demonstra o caso de utilização da crítica de um livro, que contem o utilizador, vista do livro, controlador a entidade livro e os repositórios necessários.

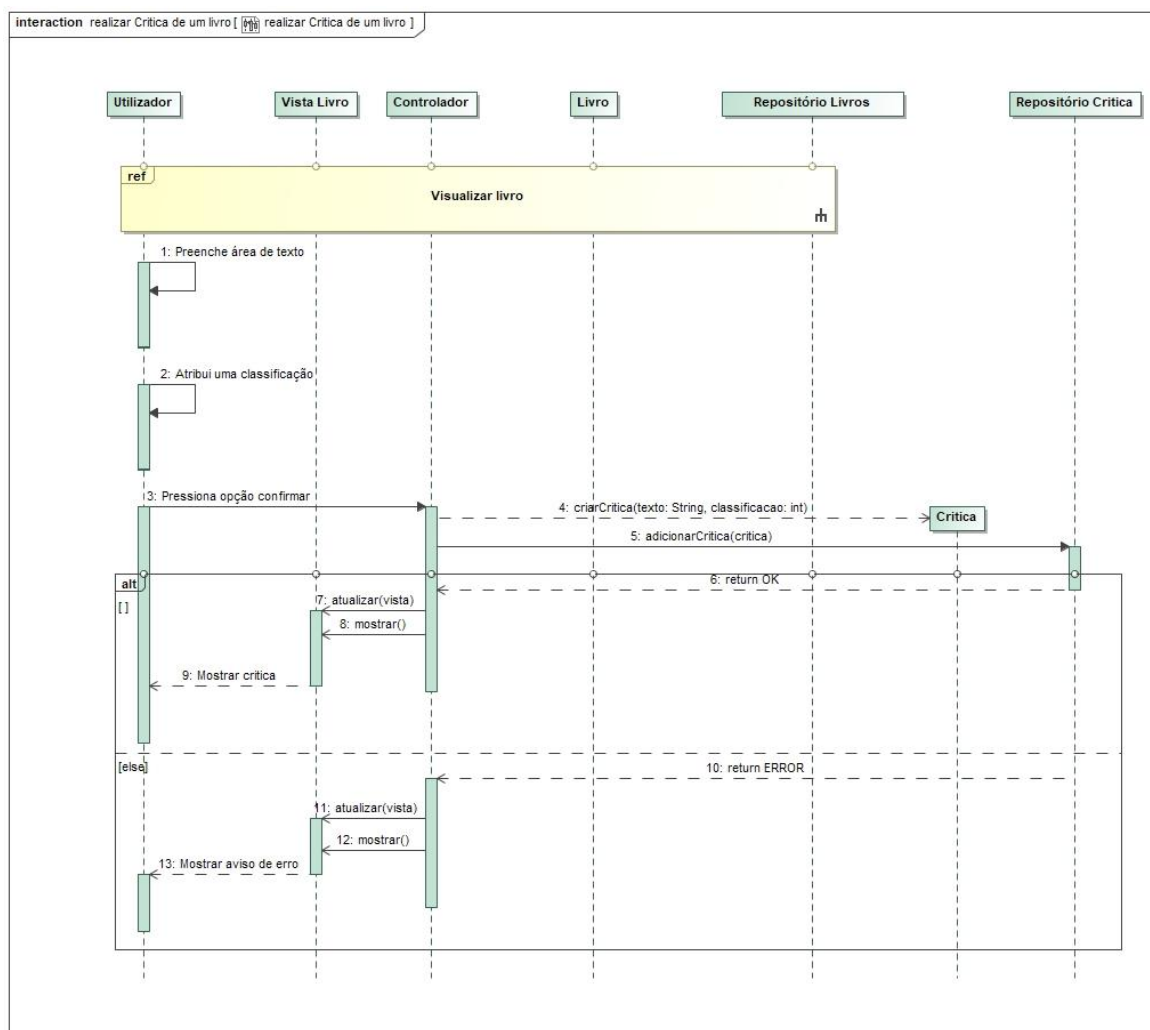


Figura 6 - Caso de utilização realizar crítica de um livro

2.2.5 Arquitetura de mecanismos

Na arquitetura de mecanismos são definidos múltiplos diagramas que descrevem os mecanismos e componentes do sistema. A arquitetura de mecanismos foi representada por diagramas de classes utilizando a linguagem UML. Esta arquitetura é independente da camada de apresentação e da camada de acesso a dados, oferecendo um nível elevado de flexibilidade e adaptabilidade. As diferentes classes e as suas relações foram elaboradas a partir dos casos de utilização realizados.

Na seguinte figura pode ser observada o mecanismo associado à adição de um comentário. Neste mecanismo consta as entidades “Crítica”, “Comentário” e “infoUtilizador” (representa informação de um utilizador) e as vistas, controlador, serviço e repositório associado.

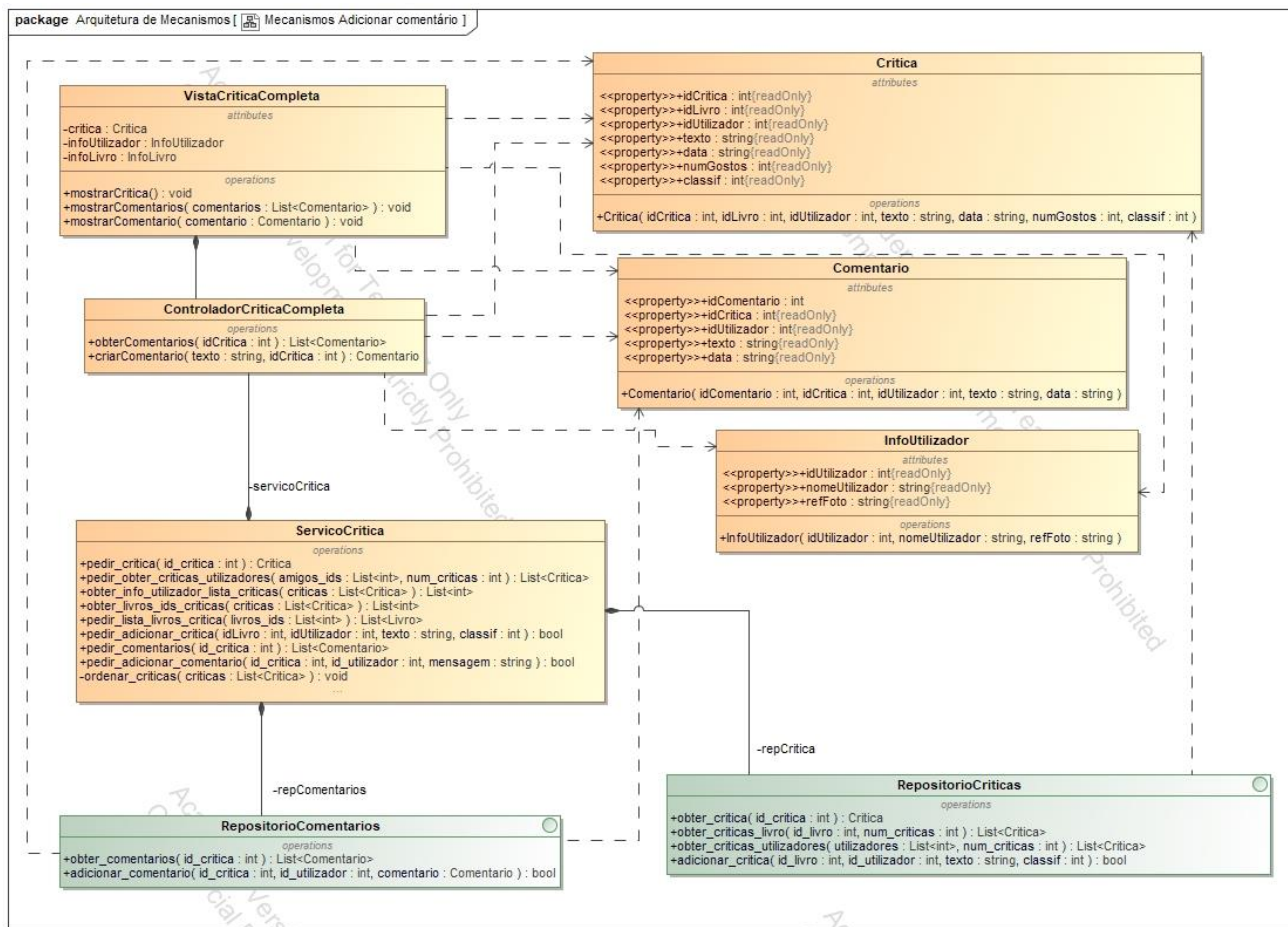


Figura 7 - Mecanismo adicionar comentário

2.2.6 Arquitetura geral da solução

A arquitetura geral da solução descreve como os diferentes subsistemas da aplicação interagem entre si, através de um conjunto de diagramas e como o sistema está estruturado. Nesta arquitetura foi utilizado o esquema de arquitetura de três camadas.

Os diferentes repositórios foram incluídos na camada de acesso a dados, os serviços e entidades na camada de domínio e as vistas e os controladores na camada de apresentação. A arquitetura geral da solução pode ser observada a partir do diagrama seguinte:

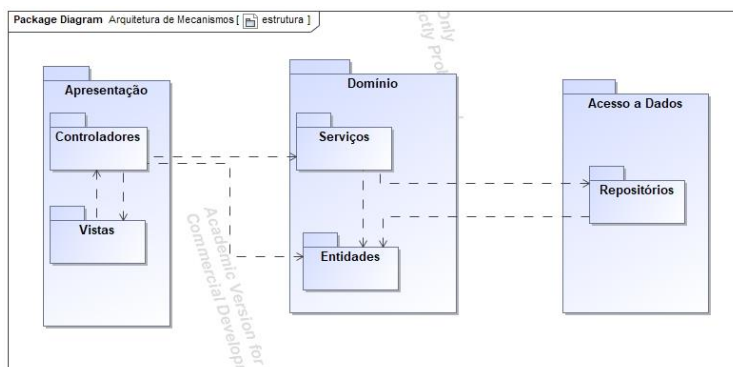


Figura 8 - Estrutura da solução

2.3 Arquitetura detalhada

Nesta etapa foi especificado como a solução será implementada e os mecanismos elaborados. Nesta fase, finalmente são descritas as decisões importantes exclusivas à **plataforma de execução**. Foram agora finalmente definidas as diferentes ferramentas, linguagem de programação e outros aspetos técnicos importantes.

A arquitetura detalhada foi definida consoante a análise de requisitos e a arquitetura geral, descendo para um nível ainda mais baixo de ambiguidade.

2.3.1 Modelo de dinâmica

O **modelo de dinâmica** representa a estrutura do sistema e como as diferentes partes interagem entre si. Denota também a forma de como as partes evoluem no tempo e como reagem a estímulos externos.

Na aplicação utilizou-se **sessões** para manter o estado do utilizador e guardar informação específica sobre a sessão. O armazenamento da sessão ocorre no lado do cliente e a informação da sessão é guardada no *browser* do cliente a partir de *cookies* ou armazenamento local.

Quando o utilizador inicia a aplicação, caso tenha uma sessão iniciada é redirecionado para a página principal. Se não, o utilizador é redirecionado para a página de login, onde pode iniciar sessão com as suas credenciais ou registar uma nova conta caso ainda não tenha feito isso.

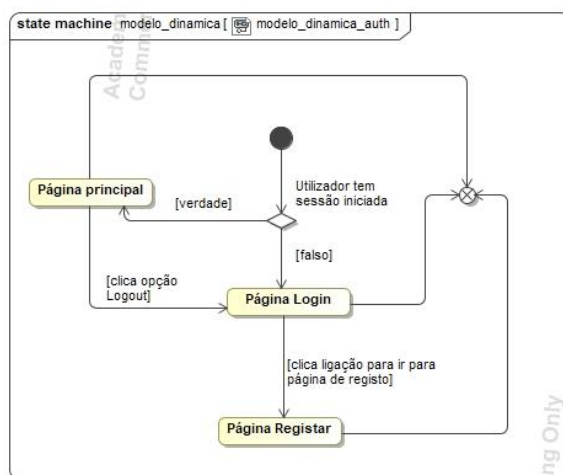


Figura 9 - modelo de dinâmica autenticação

No seguinte diagrama pode-se observar a dinâmica da aplicação, não foi incluída a lógica de início de sessão de forma a simplificar a visualização. O utilizador pode sempre aceder às páginas: principal, livro, pesquisa e utilizador através de uma barra no topo da página.

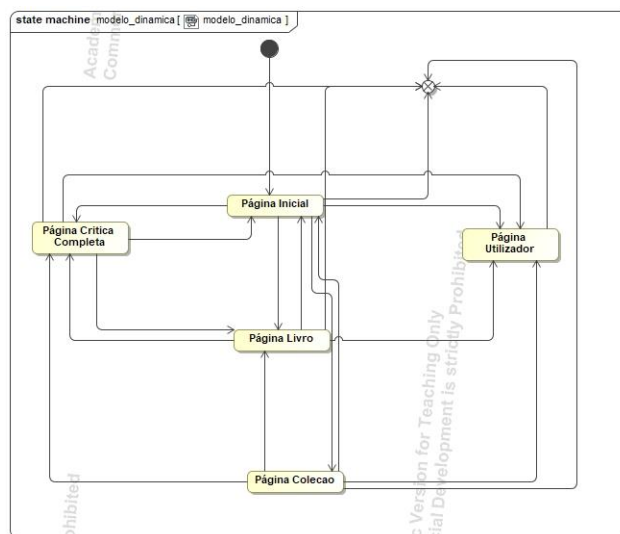


Figura 10 - modelo de dinâmica

2.3.2 Arquitetura de teste

A arquitetura de teste serve para prevenir erros antes da implementação da aplicação. A arquitetura de teste vai testar o correto funcionamento da camada de domínio, testando as classes do domínio. Na camada de domínio o acesso a dados irá ter alguns dados de teste. As tabelas da base de dados foram representadas por listas. A apresentação foi realizada em modo de consola de modo ao utilizador conseguir interagir com a aplicação através da linha de comandos. Foi realizada a

arquitetura para os diferentes casos de utilização. Para testar a aplicação foi apenas utilizada a linguagem de programação Python.

2.3.3 Tecnologias utilizadas

Django

A *framework* utilizada para desenvolvimento da aplicação Web foi o Django. Muitas aplicações populares utilizam esta *framework*, como por exemplo o Instagram, Spotify e o Pinterest.

Django é escrito em Python e foi criado com o intuito de desenvolver rapidamente e facilmente páginas Web, oferecendo um conjunto de ferramentas uteis e uma estrutura robusta. Algumas das razões para a escolha desta *framework* foi por ser escrita em Python e oferecer um conjunto de características úteis, principalmente o Django ORM.

Django ORM

Django tem uma ORM (*Object-Relational Mapping*). O ORM oferece uma camada de abstração entre base de dados relacionais e linguagem de programação orientadas a objetos sem ter de escrever consultas SQL. Isto permite trabalhar com base de dados utilizando objetos Python, em vez de escrever código SQL. Os modelos da base de dados são definidos como classes Python e o Django traduz esses modelos para tabelas de base de dados e gera as consultas SQL necessárias.

Esta funcionalidade tornou o desenvolvimento da aplicação mais rápido devido à *framework* oferecer ferramentas de consulta avançadas e relacionamento entre modelos úteis e simples, não havendo a necessidade de ter de escrever e lidar com códigos SQL.

Google Books API

De forma a conseguir pesquisar e visualizar os livros foi necessário utilizar uma API. Escolheu-se a Google Books API. Esta API permite aceder aos dados do Google Books e obter um conjunto de informação relevante sobre livros. A informação recolhida dos livros foi por exemplo: o título, nome de autor e a referência da imagem do livro.

2.3.4 Modelo de implantação da aplicação

A fase final da arquitetura detalhada é a criação do modelo de implantação da aplicação. O modelo de implantação segue o seguinte diagrama:

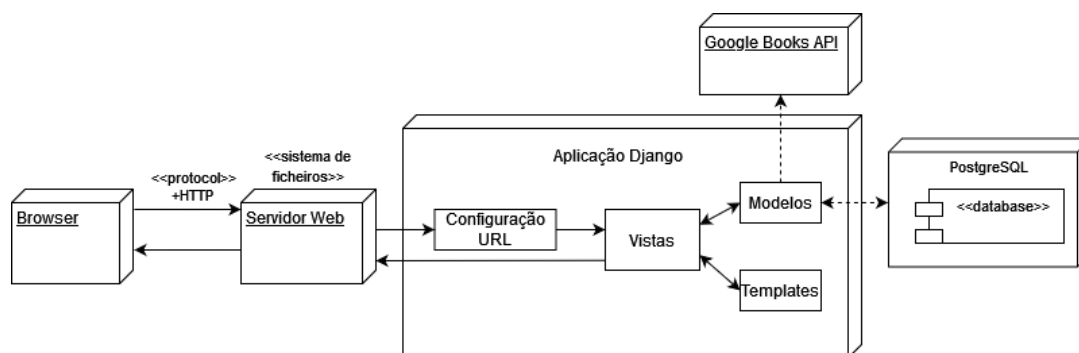


Figura 11 - diagrama de modelo de implantação da aplicação

O **modelo** é a parte de aplicação Web que serve de intermediário entre a base de dados e a Google Books API e a interface do website.

Nas **vistas** são processados os pedidos do utilizador e geradas as respostas corretas. As vistas no Django são implementadas como funções Python.

As **templates** representam a camada de apresentação e a interação do utilizador e são responsáveis por gerar a saída do HTML.

O Django utiliza uma arquitetura cliente / server e utiliza objetos pedido e resposta para comunicar entre cliente e servidor.

Em Django tradicionalmente as vistas obtêm dados da base de dados a partir do modelo, formatam os dados, juntam os dados num objeto resposta HTTP e enviam a resposta para o cliente. A lógica da aplicação fica separada das vistas, como ser observado na seguinte figura:

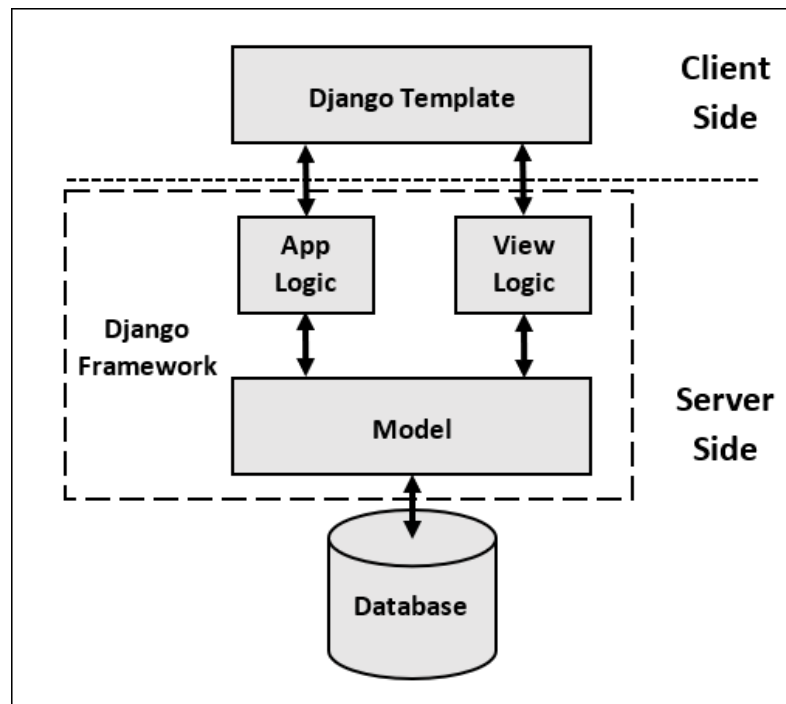


Figura 12 - esquema da lógica da arquitetura Django

As configurações URL do Django são a seta entre a vista e a *template* na figura anterior. Permitem estabelecer um caminho de comunicação entre um pedido do lado do cliente com um recurso do projeto. O Django permite mapear um URL para uma função dentro do projeto.

2.4 Implementação do protótipo aplicativo

Finalmente na implementação do protótipo aplicativo foram concretizados os casos de utilização de acordo com a especificação de requisitos. Foram implementados os seguintes casos de utilização:

- Realizar uma crítica
- Comentar uma crítica um utilizador
- Visualizar e interagir com página principal / livro / coleção / critica / utilizador
- Adicionar / Remover livro da coleção
- Enviar pedido de amizade a utilizadores
- Adicionar amigo à lista de amigos
- Autenticação

De forma a **implementar** a aplicação Web utilizou-se os diagramas de sequência, arquitetura de mecanismo e seguiu-se a estrutura definida na arquitetura geral da solução.

Em relação à arquitetura de mecanismos foram realizadas algumas alterações aos mecanismos de forma a adaptar a arquitetura ao paradigma do Django.

Grande parte do código realizado na implementação de teste foi reutilizado nesta fase, visto que a implementação de teste e implementação da aplicação ambas foram escritas em Python.

Não foi necessário alterar muito o domínio da implementação de teste para a implementação aplicacional. Foram só adicionadas funcionalidades adicionais que não tinham sido implementadas no teste.

A **vista** em Django descreve os dados que são apresentados ao utilizador e não o aspeto da apresentação dos dados. Neste caso a vista é uma função *callback* Python para um URL particular, a função *callback* descreve que dados vão ser apresentados. A vista delega para uma *template*, onde é descrito como os dados vão ser apresentados. Cada vista realiza uma função específica e tem uma associação a uma *template*.

O Django utiliza objetos de pedido e resposta para passar estado pelo sistema. Quando uma página é pedida pelo cliente, a *framework* cria um pedido HTTP com os metadados associados ao pedido. De seguida carrega a vista apropriada passando o pedido HTTP como argumento da função.

Foi necessário adaptar o código da camada de apresentação, pois a camada de apresentação nesta fase é representada por *templates* HTML. As *templates* contêm um conjunto de marcações especiais que possibilitam incluir informações passadas pelas vistas. As *templates* permitem inserir variáveis dinamicamente, condições e instruções de fluxo de controlo. Desta maneira é controlada e gerada a saída da página HTML. As páginas HTML foram formatadas a partir de CSS, tendo um design responsivo centrado no utilizador. O código realizado a partir das marcações especiais das *templates* é basicamente o código das vistas previamente definidas.

Foram criadas classes “ControladorVista” para controlar e auxiliar toda a informação, pedidos HTTP do cliente e pedidos à base de dados, que servem funcionalidades quase idênticas às classes dos controladores definidos anteriormente. Estas classes têm também cada uma um método “vista”. O método da vista serve para carregar a página HTML passando os argumentos necessários para a *template*.

As **configurações URL** foram realizadas no ficheiro “urls.py” da aplicação. Este ficheiro especifica como determinados URLs devem ser mapeados para as vistas. É possível passar argumentos através da configuração URL, por exemplo:

```
path("livro/<str:id_livro>",ViewControladorLivro().view_livro,name="livro"),
```

- O argumento “livro/<str:id_livro>” representa o URL, sendo “id_livro” o argumento passado.
- “ViewControladorLivro().view_livro” representa a vista chamada, onde vai ser chamada a *template* do livro e processada a informação do pedido.
- “Name=livro” é um argumento opcional para fazer referência ao URL de forma reversa.

2.4.1 Base de dados

As bases de dados anteriormente definidas por tabelas foram agora representadas por modelos Django. Cada tabela da base de dados é representada por uma classe, onde são definidos os diferentes atributos, chaves estrangeiras e chave primária. As diferentes classes foram guardadas num ficheiro chamado “models.py”.

O acesso à base de dados foi realizado em código dentro dos repositórios, utilizando ORM (Object-Relational Mapping), realizando consultas à base de dados utilizando objetos Python.

Os serviços foram utilizados para coordenar atividades adicionais de diferentes objetos.

2.4.2 Forms

De forma a coleccionar informação dos utilizadores utilizou-se *forms*. Os *forms* permitem ao utilizador realizar tarefas como inserir texto, seleccionar opções e enviar informação para o servidor. Ao lidar com *forms* utilizou-se os métodos HTTP GET e POST consoante a necessidade. No método **GET** as informações do pedido são enviadas como parâmetro no URL, o que muitas vezes pode não ser indicado por razões de segurança. Utilizou-se então na maior parte dos casos pedidos **POST**, em que os dados são enviados para o servidor de forma codificada.

2.4.3 Páginas realizadas

Vão ser agora demonstradas as páginas realizadas e também os casos de utilização.

Página principal e página crítica completa

Esta é a página inicial para qual o utilizador é redirecionado, caso tenha uma sessão iniciada. Contém as críticas mais recentes dos amigos do utilizador. Ao clicar na opção de ver todos os comentários o utilizador é redirecionado para a página da crítica completa, onde se pode observar todos os comentários realizados à crítica.

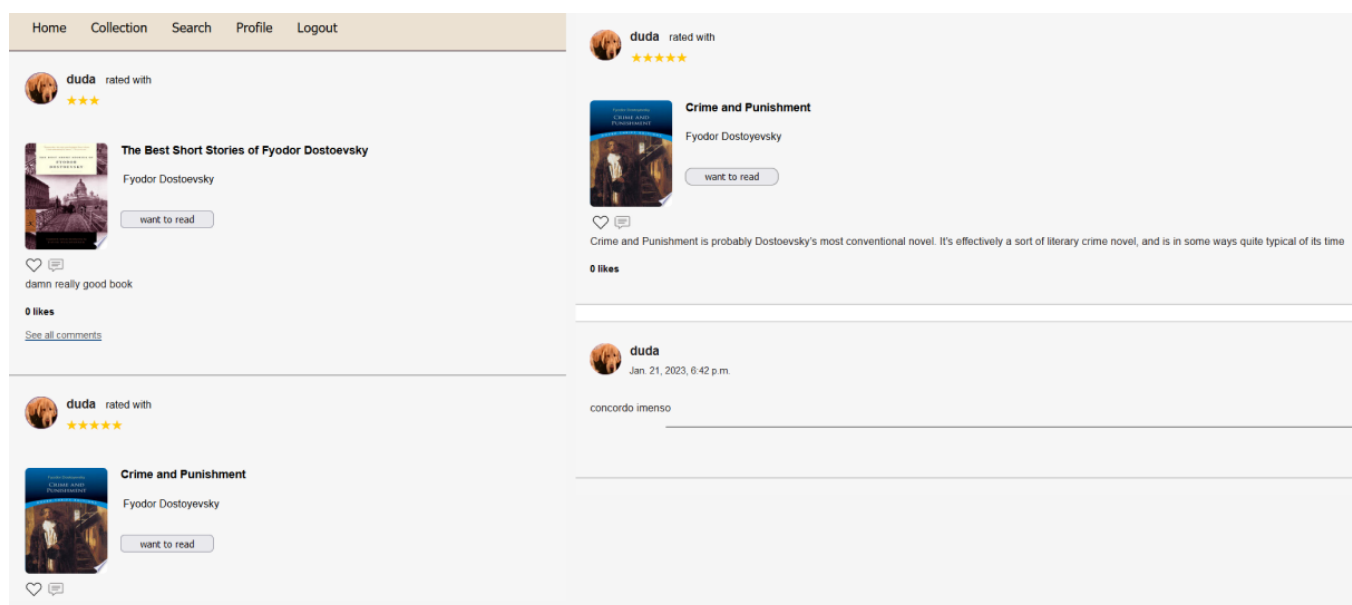


Figura 13 - Página principal e página da crítica

Na seguinte figura, apresenta-se o *form* para o utilizador inserir o comentário na página da crítica completa.

A screenshot of a web form for inserting a comment. It features a large, empty text area with a light beige background and a thin border. Below the text area is a small, rounded button labeled "Comment".

Figura 14 - form para inserir comentário

Página livro

Na seguinte figura está apresentada a página do livro, que contém a informação do livro e uma área para criar uma crítica e classificar o livro e outra área para adicionar o livro à coleção.

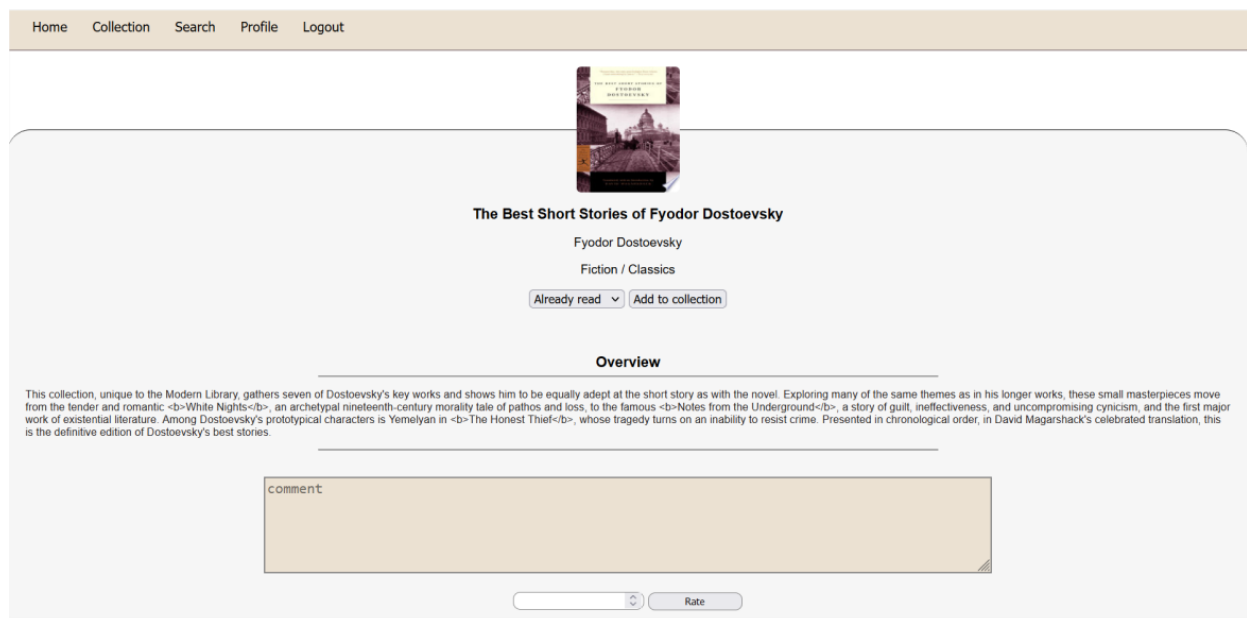
A screenshot of a book page on a website. At the top is a navigation bar with links: Home, Collection, Search, Profile, Logout. Below this is a book cover for "The Best Short Stories of Fyodor Dostoevsky". The title "The Best Short Stories of Fyodor Dostoevsky" is displayed, followed by the author "Fyodor Dostoevsky" and the genre "Fiction / Classics". There are two buttons: "Already read" with a dropdown arrow and "Add to collection". Below this is a section titled "Overview" with a paragraph of text about the collection. At the bottom, there is a large text area for a comment, a small dropdown menu, and a "Rate" button.

Figura 15 - página livro

O caso de utilização de adicionar um livro à coleção foi feito nesta página, o utilizador pode escolher entre três subcategorias de coleção.

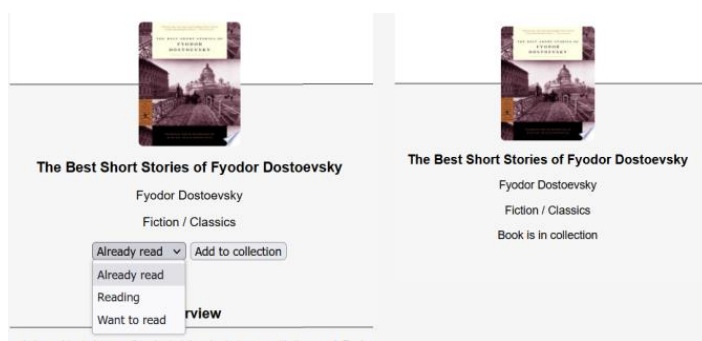


Figura 16 - funcionalidade de adicionar livro à coleção

Página coleção

Na página de coleção pode ser observada a coleção do utilizador. Ao clicar no botão “-” por debaixo de um livro da coleção o utilizador remove o livro da sua coleção. A página da coleção está dividida em três diferentes categorias. Ao clicar na imagem do livro, o utilizador é redirecionado para a página do livro.

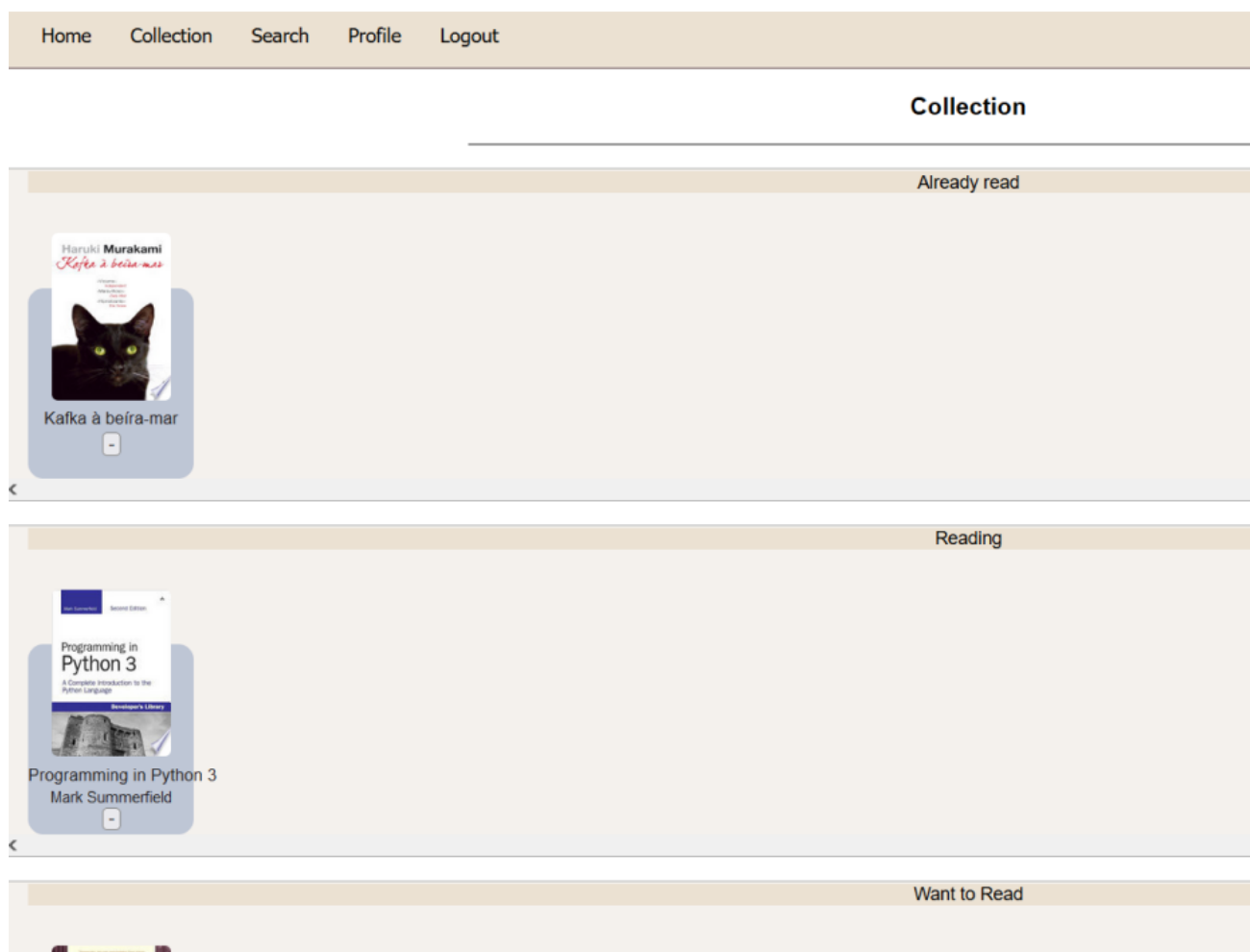


Figura 17 - página coleção

Página pesquisa

Nesta página o utilizador pode fazer uma pesquisa por livro. A pesquisa é realizada a partir de um pedido GET que passa o valor inserido na barra de pesquisa como parâmetro no URL. Os livros são obtidos a partir da Google Books API, retornando os livros mais pertinentes que contêm o valor inserido pelo utilizador.

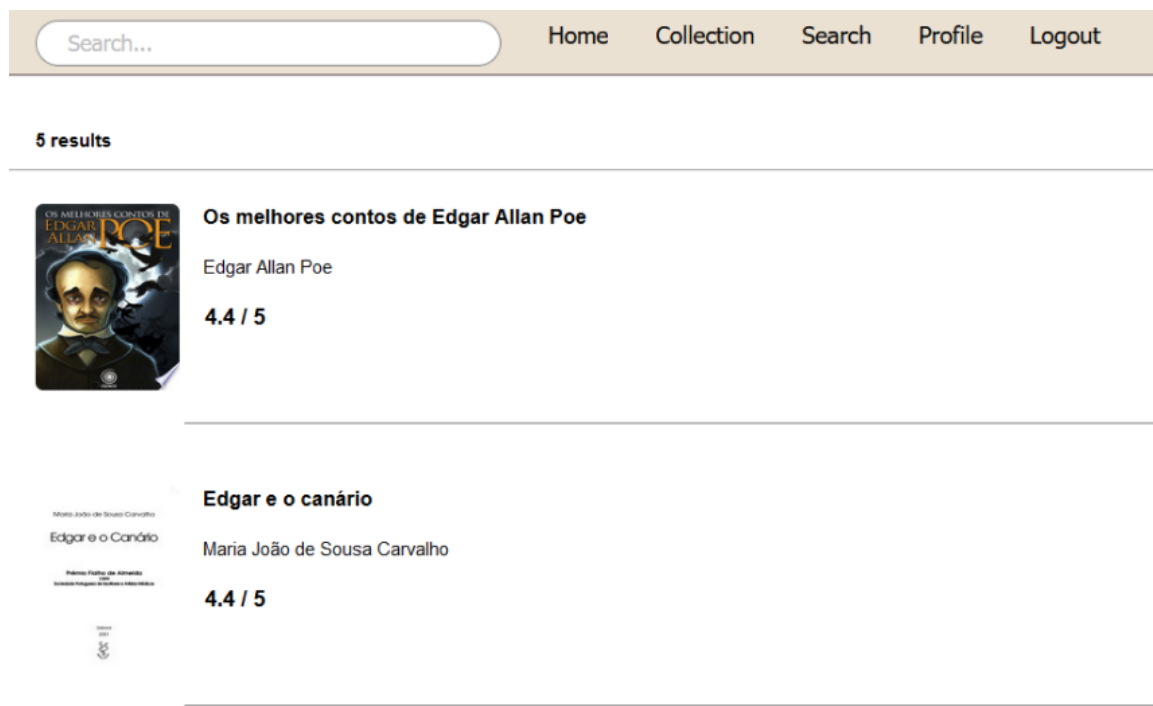


Figura 18 - página pesquisa

Página utilizador

Nesta página é possível visualizar o perfil do próprio utilizador que está a utilizar a aplicação Web e o perfil de outro utilizador. Nesta página é possível observar informação sobre o utilizador e a sua lista de amigos.

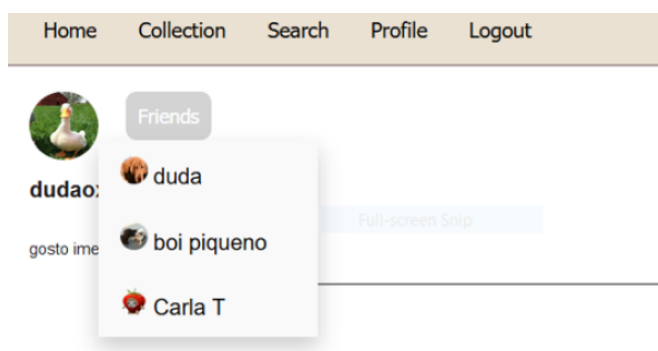


Figura 19 - página de utilizador

Caso o utilizador esteja no perfil de outro utilizador pode enviar um pedido de amizade, na figura abaixo pode se observar um exemplo da opção de enviar pedido de amizade.

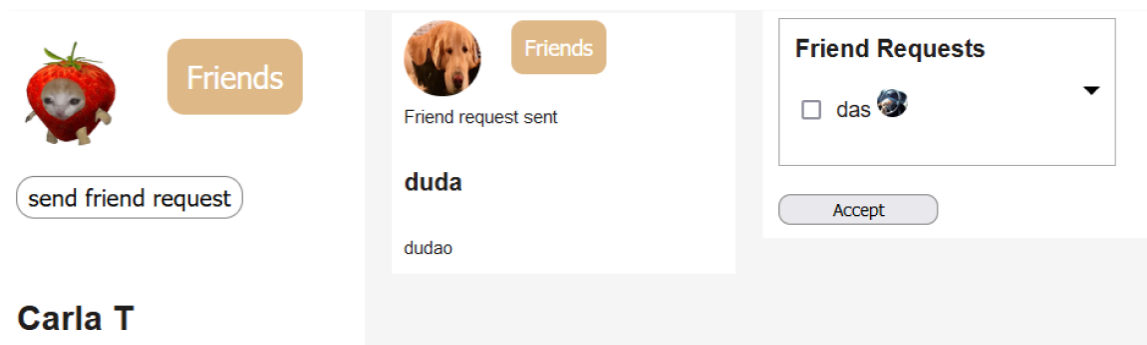


Figura 20 - funcionalidades de sistema de amizades

Páginas de autenticação

Na figura seguinte pode se observar as páginas de autenticação. A autenticação recorreu ao uso do módulo de autenticação do Django. Este módulo oferece uma forma rápida e segura de criação de contas de novos utilizadores, oferecendo encriptação das passwords e proteção da informação sensível dos utilizadores.

The image displays two side-by-side forms for user authentication. The left form is titled 'Log-in' and 'Login', featuring input fields for 'Username' and 'Password', a 'Login' button, and a link to 'Create an account'. The right form is titled 'Register' and includes input fields for 'Username', 'Email', 'Password', and 'Password confirmation'. It also features a 'Register' button and a list of password requirements: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.' Both forms have a footer note: 'If you already have an account, login instead.'

Figura 21 - páginas de autenticação

Análise crítica do projeto realizado

O projeto realizado teve os seus pontos fortes e fracos. O ponto mais forte foi a aplicação funcionar totalmente, todas as páginas e as funcionalidades propostas foram acabadas. A aplicação oferece segurança aos seus utilizadores, através de encriptação de passwords e mecanismos de segurança. Outro ponto forte foi a boa definição das entidades envolvidas no domínio do problema, que permitiu realizar casos de utilização relevantes para o desenvolvimento do sistema.

Os requisitos da aplicação foram bem estruturados, tendo uma base forte devido aos casos de utilização terem um nível alto de detalhe. Apesar disto, algum tempo foi gasto desnecessariamente devido a terem sido definidos demasiados objetivos e funcionalidades para a aplicação na fase inicial. Foi gasto algum tempo a definir a ideia de um possível sistema de recomendação de livros que nunca foi implementado. Porém algumas das ideias deste sistema de recomendação poderiam ser reaproveitadas em trabalho futuro.

O projeto teve o ponto fraco de ter ocorrido alguma perda de informação entre a arquitetura estruturada e a arquitetura implementada na aplicação. Isto deveu-se ao facto do Django seguir um padrão ligeiramente diferente do MVC e devido a ter alguma inexperiência com este *framework*.

Entretanto, a utilização do Django trouxe benefícios como poder fazer consultas à base de dados em Python e ter de realizar apenas código Python e HTML o que tornou a aplicação mais simples.

Algo importante para possível trabalho futuro seria lidar de forma mais eficiente com atividades do utilizador. Um exemplo disto seria apenas carregar um número de críticas inicial na página principal, e caso o utilizador chegue ao final da página, carregar mais críticas para o utilizador visualizar. Outra funcionalidade seria a adição de *auto-complete* na barra de pesquisa, o que iria ajudar significativamente a experiência do utilizador.

Em geral o projeto apesar de ter tido muitas fases de refactorização e ter havido algumas dificuldades em algumas etapas da arquitetura do sistema, foi bem conseguido.

A partir do desenvolvimento do projeto aprendi muitos conceitos e lições importantes sobre como fazer software de qualidade e de baixa entropia. Também foi útil para adquirir capacidades de organização muito úteis para futuros projetos de engenharia informática e para a vida pessoal.

Conclusões

Em suma neste projeto foi realizada uma aplicação Web seguindo padrões e princípios da engenharia de software. Neste projeto foi possível conceptualizar, testar e implementar uma aplicação através da abordagem de começar num nível de elevada abstração e acabar num nível de abstração mínima.

Inicialmente a aplicação Web foi estruturada independente de um modelo computacional, tendo sido inicialmente definida a estrutura e a ideia por de trás da aplicação. Em cada etapa, o sistema foi descrito com o maior nível de detalhe possível de forma a arquitetura desenvolvida ser útil para a implementação.

Enquanto o projeto foi progredindo, a modelação do sistema passou a ser independente da plataforma de execução, em que o sistema foi descrito com um elevado nível de detalhe independentemente de uma linguagem de programação. Este tipo de modulação foi um conceito novo, até agora a maior parte do código produzido por mim tinha sido produzido tendo em conta uma linguagem de programação ou uma plataforma de desenvolvimento. Esta abstração da linguagem de programação, permitiu estar livre de certos conceitos e hábitos associados com programação. Permitindo ter um maior foco no desenho da arquitetura do sistema. Desta forma, analisando as diferentes componentes e metodologias adequadas para alcançar uma aplicação de menor complexidade e de maior confiabilidade.

A realização deste projeto seguiu um processo constante de refactorização, a estrutura do sistema foi analisada e estudada com atenção em cada etapa do projeto. Exemplos de refactorização foi a reorganização de classes e métodos e melhoria da simplicidade do sistema. Isto permitiu evitar que o nível de entropia do sistema chegasse a níveis muito elevados, permitindo que a aplicação seja escalável e de qualidade.

Na implementação do projeto as tecnologias e ferramentas utilizadas contribuíram para o sucesso da aplicação como o Django, ORM, Google Books API e o Python.


No final do projeto foi possível realizar uma aplicação Web funcional, com diferentes utilidades, útil e divertida para pessoas interessadas em leitura e em partilharem as suas opiniões sobre os livros que leram.


Bibliografia



- [1] - Luís Morgado, Engenharia de Software, 02-Introdução à engenharia de software, 2022
- [2] - Luís Morgado, Engenharia de Software, 03-Processos de desenvolvimento de software, 2022
- [3] - Luís Morgado, Engenharia de Software, 04-Análise de requisitos, 2022
- [4] - Luís Morgado, Engenharia de Software, 08-Arquitetura de software, 2022
- [5] - Luís Morgado, Engenharia de Software, 09-Modelo de domínio, 2022
- [6] - Luís Morgado, Engenharia de Software, 10-Modelo de interação, 2022
- [7] - Luís Morgado, Engenharia de Software, 11-Modelo de dinâmica, 2022
- [8] - Luís Morgado, Engenharia de Software, 12-Implementação, 2022
- [9] – Django FAQ, <https://docs.djangoproject.com/en/1.8/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names>
- [10] - Mastering Django: Structure, <https://masteringdjango.com/django-tutorials/mastering-django-structure/>


Anexo


[Home](#) [Collection](#) [Search](#) [Profile](#) [Logout](#)



**duda** rated with
☆☆☆

**The Best Short Stories of Fyodor Dostoevsky**
Fyodor Dostoevsky
[want to read](#)



damn really good book
0 likes
[See all comments](#)

**duda** rated with
☆☆☆☆☆

**Crime and Punishment**
Fyodor Dostoevsky
[want to read](#)


Crime and Punishment is probably Dostoevsky's most conventional novel. It's effectively a sort of literary crime novel, and is in some ways quite typical of its time
0 likes
[See all comments](#)

[Home](#) [Collection](#) [Search](#) [Profile](#) [Logout](#)



The Best Short Stories of Fyodor Dostoevsky
Fyodor Dostoevsky
Fiction / Classics
Book is in collection


Overview

This collection, unique to the Modern Library, gathers seven of Dostoevsky's key works and shows him to be equally adept at the short story as with the novel. Exploring many of the same themes as in his longer works, these small masterpieces move from the tender and romantic White Nights, an archetypal nineteenth-century morality tale of pathos and loss, to the famous Notes from the Underground, a story of guilt, ineffectiveness, and uncompromising cynicism, and the first major work of existential literature. Among Dostoevsky's prototypical characters is Yemelyan in The Honest Thief, whose tragedy turns on an inability to resist crime. Presented in chronological order, in David Magarshack's celebrated translation, this is the definitive edition of Dostoevsky's best stories.

comment

Rate


Reviews

**duda** rated with
☆☆☆


damn really good book

HomeCollectionSearchProfileLogout


5 results




The Rotarian
4.4 / 5




The Crisis
4.4 / 5



The Advocate
4.4 / 5



The Advocate
4.4 / 5




The Crisis
4.4 / 5

HomeCollectionSearchProfileLogout


Collection

Already read



Kafka à beira-mar


Reading



Programming in Python 3
Mark Summerfield

Want to Read

[Home](#) [Collection](#) [Search](#) [Profile](#) [Logout](#)

 [Friends](#)

dudaoxd
gosto imenso de livros históricos

Collection

Already read

< >

Reading

< >

Want to Read

< >

Log-in

Login

Username: Password:

Don't have [Ver credenciais guardadas](#) [punt.](#)

[Register a new account](#)

Register

Username:
Required. Ver credenciais guardadas Letters, digits and @/./+/_ only. Email: Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation:
Enter the same password as before, for verification.

If you already have an account, [login](#) instead.