UNIVERSIDADE DA MAIA UMAIA

UNIVERSIDADE DA MAIA

MESTRADO EM INFORMÁTICA

Arquitectura e Desenho de Software

DomusShelf

Sistema de Gestão de Farmácia Doméstica

Relatório de Projecto

Miguel Ângelo Ascensão Real
Aluno n.º 48891
Professor: Alexandre Sousa

14 de Fevereiro de 2026

ÍNDICE

1. INTRODUÇÃO

1.1. Contexto e Motivação

A gestão de medicamentos no contexto doméstico é uma necessidade quotidiana que afecta milhões de famílias. Medicamentos que passam da validade sem que ninguém se aperceba, embalagens repetidas compradas por esquecimento, ou simplesmente a dificuldade em saber o que existe em casa são problemas recorrentes. Segundo dados da Ordem dos Farmacêuticos, estima-se que uma percentagem significativa dos medicamentos adquiridos em Portugal acaba por não ser utilizada dentro do prazo de validade. Esta realidade motivou o desenvolvimento do DomusShelf, uma aplicação web que permite a qualquer pessoa organizar, controlar e ser alertada sobre o estado da sua farmácia caseira.

O nome "DomusShelf" combina a palavra latina "Domus" (casa) com "Shelf" (prateleira), remetendo para a ideia de uma prateleira digital que guarda e organiza os medicamentos de cada agregado familiar.

1.2. Objectivos do Projecto

O objectivo principal deste projecto é desenvolver uma aplicação web funcional utilizando a framework Django, demonstrando a compreensão dos conceitos de arquitectura e desenho de software leccionados na unidade curricular. Conforme enfatizado pelo professor Alexandre Sousa, "o principal é perceber de Django", pelo que o projecto serve simultaneamente como veículo de aprendizagem e como demonstração prática dos padrões de desenho embebidos nesta framework.

Especificamente, a aplicação deve permitir o registo e gestão de medicamentos com as respectivas embalagens, o controlo de stock e datas de validade com alertas configuráveis, o registo de consumos com actualização automática do stock, a gestão de preferências pessoais de cada utilizador, e o registo autónomo de novos utilizadores. Para além da componente funcional, o projecto documenta os padrões de desenho embebidos na framework Django, a arquitectura MTV (Model-Template-View) e a sua relação com o padrão MVC tradicional, e as boas práticas de desenvolvimento web com Python.

1.3. Âmbito e Limitações

O DomusShelf foi desenvolvido como projecto académico para a unidade curricular de Arquitectura e Desenho de Software do Mestrado em Informática da Universidade da Maia. Como tal, existem algumas limitações assumidas: a aplicação funciona em modo de desenvolvimento (não foi feito deployment para produção), a base de dados utilizada é SQLite

(adequada para desenvolvimento mas não para produção com múltiplos utilizadores simultâneos), e não foram implementadas funcionalidades como notificações por email ou integração com serviços externos.

2. Estado da Arte

2.1. Aplicações Existentes

No mercado existem diversas aplicações móveis dedicadas à gestão de medicamentos, tais como o MediSafe, o MyTherapy e o CareZone. Estas aplicações focam-se principalmente no lembrete de tomas e no acompanhamento de tratamentos crónicos. No entanto, poucas oferecem uma funcionalidade robusta de gestão de stock doméstico com controlo de validades e alertas configuráveis, que é o foco principal do DomusShelf.

A maioria destas soluções são aplicações móveis proprietárias com modelos de subscrição paga, ao passo que o DomusShelf é uma aplicação web acessível a partir de qualquer dispositivo com um browser, sem necessidade de instalação.

2.2. Tecnologias Consideradas

2.2.1. Linguagem de Programação: Python

Python foi a linguagem escolhida para o desenvolvimento do projecto, não apenas por ser a linguagem recomendada para a unidade curricular, mas também pelas suas características intrínsecas: sintaxe clara e legível, vasta comunidade de suporte, e um ecossistema rico de bibliotecas. Python segue a filosofia de que o código deve ser legível e explícito, princípios que se reflectem directamente na framework Django. O professor recomendou ainda o Django Girls Tutorial como referência de aprendizagem, o que foi seguido durante o desenvolvimento.

2.2.2. Framework Web: Django

O Django é uma framework web de alto nível para Python que segue o princípio "batteries included", ou seja, inclui de raiz tudo o que é necessário para desenvolver uma aplicação web completa, desde o mapeamento objecto-relacional (ORM) até ao sistema de autenticação, passando pelo painel de administração automático. Foi criado em 2003 e lançado publicamente em 2005, sendo actualmente uma das frameworks web mais utilizadas a nível mundial.

A escolha do Django em detrimento de alternativas como Flask ou FastAPI deveu-se precisamente a esta abordagem integrada. Enquanto o Flask segue uma filosofia minimalista que exige a instalação e configuração manual de cada componente, o Django oferece uma solução coesa onde todos os componentes foram desenhados para funcionar em conjunto. Para um projecto como o DomusShelf, que necessita de autenticação, formulários, administração e ORM, o Django reduz significativamente o esforço de desenvolvimento.

2.2.3. Interface: Bootstrap 5

Para a camada de apresentação foi utilizado o Bootstrap 5, uma das frameworks CSS mais populares do mundo. O Bootstrap oferece um sistema de grelha responsivo, componentes pré-estilizados (botões, cards, formulários, navbar) e utilitários CSS que permitem criar interfaces profissionais com rapidez. A versão 5 removeu a dependência do jQuery, tornando-se mais leve e moderna. Complementarmente, foi utilizado o Flatpickr como widget de selecção de datas, por oferecer uma experiência de utilizador superior aos campos de data nativos dos browsers, e o Bootstrap Icons para iconografia consistente em toda a interface.

2.2.4. Base de Dados: SQLite

O SQLite é o motor de base de dados utilizado por defeito no Django e foi mantido para este projecto. Trata-se de uma base de dados relacional que armazena toda a informação num único ficheiro (db.sqlite3), dispensando a instalação e configuração de um servidor de base de dados separado. Esta simplicidade torna-o ideal para desenvolvimento e para aplicações com um número reduzido de utilizadores simultâneos. Numa eventual passagem a produção, a migração para PostgreSQL seria directa graças à abstracção proporcionada pelo ORM do Django.

2.3. Padrão Arquitectural: MVC e MTV

O padrão Model-View-Controller (MVC) é um dos padrões arquitecturais mais utilizados no desenvolvimento de aplicações web. Propõe a separação da aplicação em três componentes interligados: o Model (dados e lógica de negócio), o View (apresentação ao utilizador) e o Controller (gestão dos pedidos e coordenação entre Model e View).

O Django implementa uma variante deste padrão denominada MTV (Model-Template-View), onde a terminologia difere mas os conceitos são equivalentes. O Model do Django corresponde ao Model do MVC, o Template do Django corresponde ao View do MVC (responsável pela apresentação), e a View do Django corresponde ao Controller do MVC (responsável por processar pedidos e coordenar a resposta). A própria framework actua como um "controlador" adicional através do URL dispatcher, que encaminha cada pedido HTTP para a view correcta. Esta relação será explorada em detalhe no Capítulo 4.

3. Análise de Requisitos

3.1. Requisitos Funcionais

Os requisitos funcionais definem as funcionalidades que o sistema deve oferecer. Foram identificados os seguintes requisitos principais:

ID	Requisito	Prioridade
RF01	O sistema deve permitir o registo e autenticação de utilizadores	Alta
RF02	O utilizador deve poder criar, consultar, editar e eliminar medicamentos	Alta
RF03	O utilizador deve poder gerir embalagens (stock) com datas de validade	Alta
RF04	O sistema deve alertar o utilizador sobre embalagens expiradas ou a expirar	Alta
RF05	O utilizador deve poder registar consumos/tomas de medicamentos	Média
RF06	O sistema deve actualizar automaticamente o stock ao registar um consumo	Média
RF07	O utilizador deve poder configurar o período de antecedência dos alertas	Média
RF08	O sistema deve apresentar um dashboard com estatísticas gerais	Média
RF09	Deve existir um painel de administração para gestão global dos dados	Baixa
RF10	Cada utilizador só deve ver e gerir os seus próprios dados	Alta
RF11	O sistema deve permitir registo autónomo de novos utilizadores	Média
RF12	O dropdown de consumos deve mostrar o número de lote quando disponível	Baixa

Tabela 1 — Requisitos Funcionais

3.2. Requisitos Não Funcionais

Os requisitos não funcionais descrevem as qualidades que o sistema deve possuir:

ID	Requisito	Categoria
RNF01	A interface deve ser responsiva (adaptável a desktop, tablet e telemóvel)	Usabilidade
RNF02	O sistema deve proteger os dados de cada utilizador (isolamento)	Segurança
RNF03	As palavras-passe devem ser armazenadas de forma segura (hash)	Segurança
RNF04	A protecção CSRF deve estar activa em todos os formulários	Segurança
RNF05	O sistema deve ser desenvolvido em Python com a framework Django	Tecnológico
RNF06	A interface deve utilizar Português de Portugal	Usabilidade
RNF07	O código deve estar sob controlo de versões (Git/GitHub)	Manutenção
RNF08	As datas devem ser apresentadas no formato português (dd/mm/aaaa)	Usabilidade

Tabela 2 — Requisitos Não Funcionais

3.3. Modelo de Domínio

O domínio da aplicação centra-se em quatro entidades principais: o Medicamento (informação genérica sobre um fármaco), a Embalagem (unidade física de stock com data de validade), o

Consumo (registo de cada toma) e as Preferências (configurações do utilizador). O utilizador é representado pelo modelo User do próprio Django.

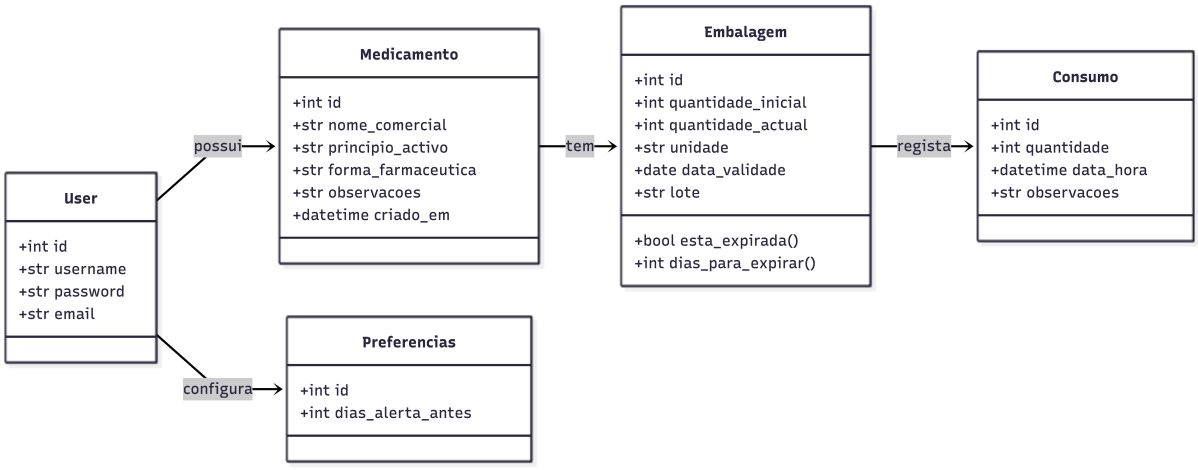

Figura 1 — Modelo de Domínio do DomusShelf

O diagrama acima ilustra as relações entre as entidades: um Utilizador possui zero ou mais Medicamentos, cada Medicamento tem zero ou mais Embalagens, cada Embalagem tem zero ou mais Consumos, e cada Utilizador tem exactamente uma instância de Preferências.

4. Arquitectura e Desenho de Software

Este capítulo constitui o núcleo do relatório, documentando as decisões arquitecturais e os padrões de desenho utilizados no DomusShelf. A estrutura segue parcialmente o template arc42, um standard europeu para documentação de arquitectura de software.

4.1. Contexto e Âmbito

O DomusShelf é uma aplicação web monolítica de uso pessoal ou familiar. O sistema recebe pedidos HTTP do browser do utilizador, processa-os no servidor Django e devolve páginas HTML renderizadas. Não existem integrações com sistemas externos na versão actual.

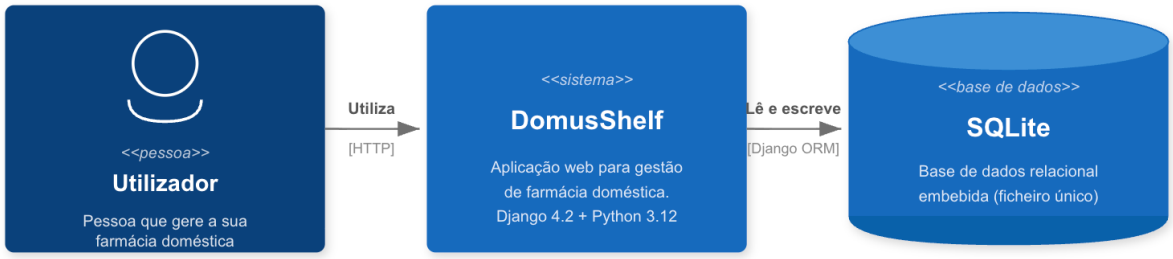

Figura 2 — Diagrama de Contexto do DomusShelf

Conforme ilustrado na Figura 2, o sistema tem um único actor - o utilizador autenticado - que interage com a aplicação através do browser. A aplicação comunica com a base de dados SQLite para persistir e recuperar informação.

4.2. Arquitectura MTV do Django

O Django implementa o padrão MTV (Model-Template-View), uma adaptação do clássico MVC (Model-View-Controller). Para compreender esta relação, é útil pensar numa analogia com um restaurante: o Model é a cozinha (onde os dados são preparados e armazenados), o Template é a mesa posta (a apresentação ao cliente), e a View é o empregado de mesa (que recebe o pedido, vai à cozinha, e traz o prato à mesa). O URL Dispatcher seria o chefe de sala que decide qual empregado atende cada mesa.

Componente MVC	Componente Django MTV	Responsabilidade no DomusShelf
Model	Model (models.py)	Define Medicamento, Embalagem, Consumo e Preferencias
View (apresentação)	Template (ficheiros .html)	Páginas HTML com Bootstrap 5
Controller	View (views.py) + URL Dispatcher	Funções que processam pedidos e devolvem respostas

Tabela 3 — Correspondência entre MVC e MTV no DomusShelf

É importante notar que o próprio Django actua como um "controlador" invisível: o URL Dispatcher (configurado nos ficheiros urls.py) recebe cada pedido HTTP, analisa o URL, e encaminha-o para a view correcta. Este mecanismo segue o padrão Front Controller, onde existe um ponto de entrada único para todos os pedidos.

4.3. Vista de Blocos Construtivos

4.3.1. Estrutura do Projecto

O Django faz uma distinção clara entre projecto e aplicação (app). O projecto (domusshelf_project/) contém as configurações globais, como a lista de middleware, a configuração da base de dados e o mapeamento de URLs principal. A aplicação (pharmacy/) contém a lógica específica do domínio, os modelos de dados, as views, os formulários e os templates. Esta separação permite que uma app seja reutilizada em diferentes projectos.

Listagem 1 — Estrutura de directórios do projecto DomusShelf

```
DomusShelf/
├── domusshelf_project/       # Configuração do projecto Django
│   ├── settings.py           # Configurações globais (localização pt-pt)
│   ├── urls.py               # URLs principais (raiz + auth + registo)
│   └── wsgi.py               # Ponto de entrada WSGI
├── pharmacy/                 # Aplicação principal
│   ├── models.py             # Modelos de dados (4 entidades)
│   ├── views.py              # Views / controladores (13 funções)
│   ├── forms.py              # Formulários (4 ModelForms)
│   ├── urls.py               # URLs da app (com namespace)
│   ├── admin.py              # Configuração do painel admin
│   └── context_processors.py # Processador global de alertas
├── templates/                # Templates HTML
│   ├── base.html             # Template base (herança)
│   ├── pharmacy/             # Templates da app pharmacy (10 ficheiros)
│   └── registration/         # Login e registo de utilizadores
├── db.sqlite3                # Base de dados SQLite
├── manage.py                 # Utilitário de gestão Django
└── requirements.txt          # Dependências Python
```

4.3.2. Diagrama de Componentes

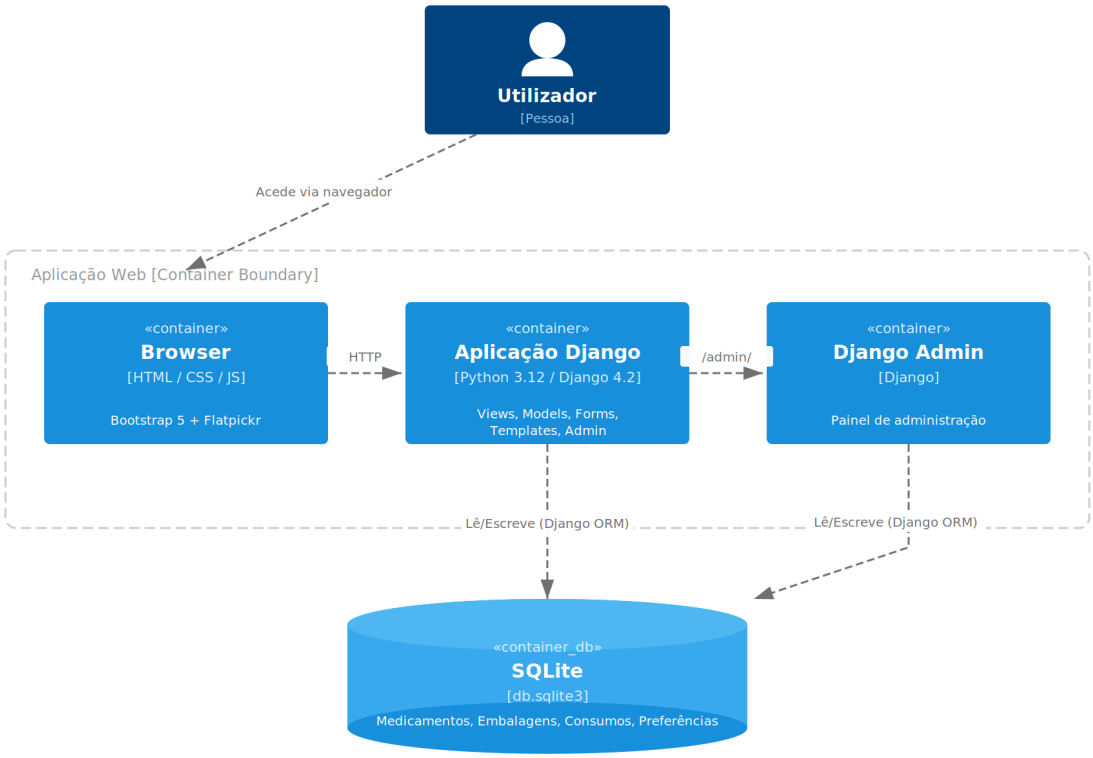

Figura 3 — Diagrama de Componentes do DomusShelf

4.4. Modelo de Dados

O modelo de dados do DomusShelf é composto por quatro entidades principais, todas definidas no ficheiro models.py da aplicação pharmacy. O Django ORM (Object-Relational Mapper) transforma estas classes Python em tabelas na base de dados, sem necessidade de escrever SQL manualmente.

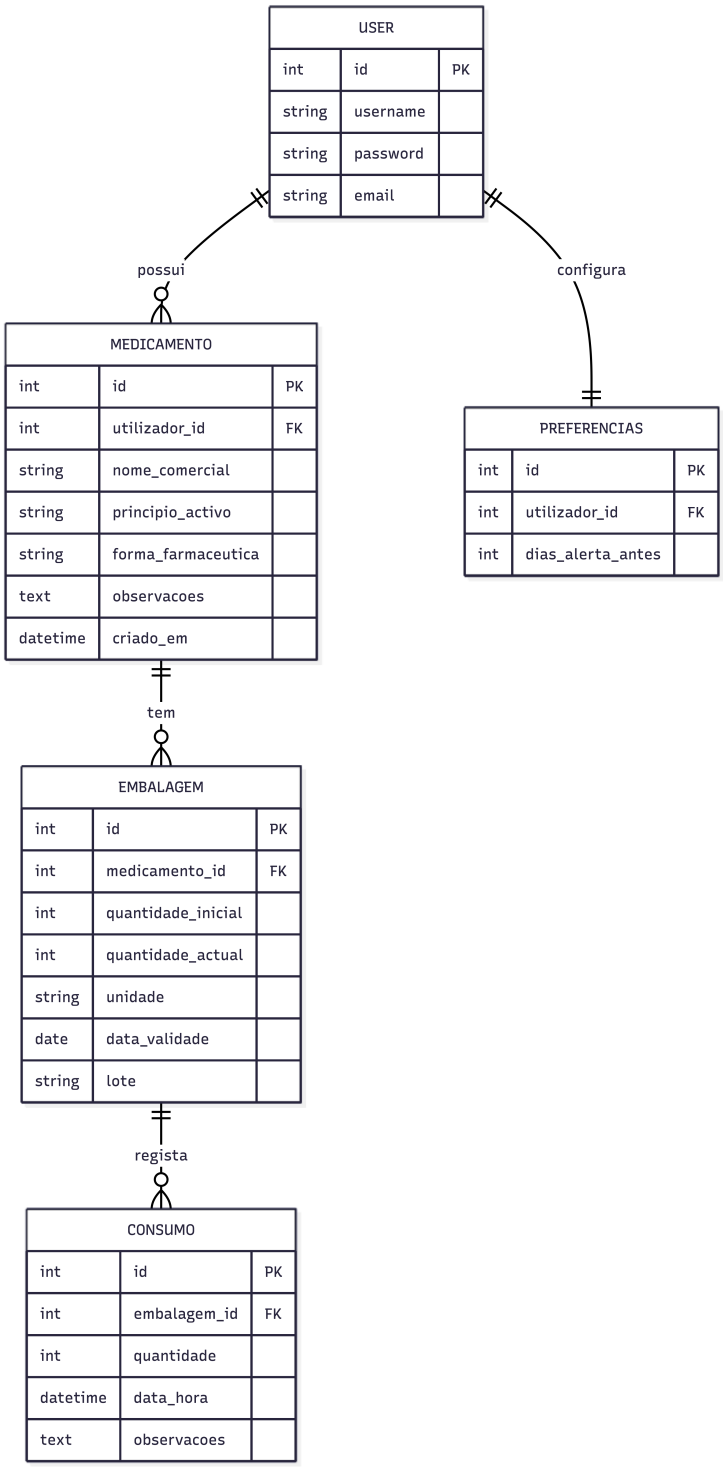

Figura 4 — Diagrama Entidade-Relacionamento

A entidade **Medicamento** representa a informação genérica de um fármaco (nome comercial, princípio activo, forma farmacêutica). Cada medicamento pertence a um utilizador através de uma relação ForeignKey. A entidade **Embalagem** representa uma unidade física de stock com data de validade, quantidade inicial e actual. Utiliza o princípio FEFO (First Expired, First Out) na ordenação, ou seja, as embalagens que expiram mais cedo aparecem primeiro. A entidade

Consumo regista cada toma, ligando-se a uma embalagem específica. A entidade **Preferencias** guarda as configurações de cada utilizador (como o número de dias de antecedência para alertas), utilizando uma relação OneToOneField que garante que cada utilizador tem exactamente um registo de preferências.

4.5. Fluxo de um Pedido HTTP

Para compreender como os componentes interagem, é útil acompanhar o percurso de um pedido HTTP desde o browser até à resposta. A Figura 5 ilustra este fluxo para o caso de um utilizador que acede à lista de medicamentos.

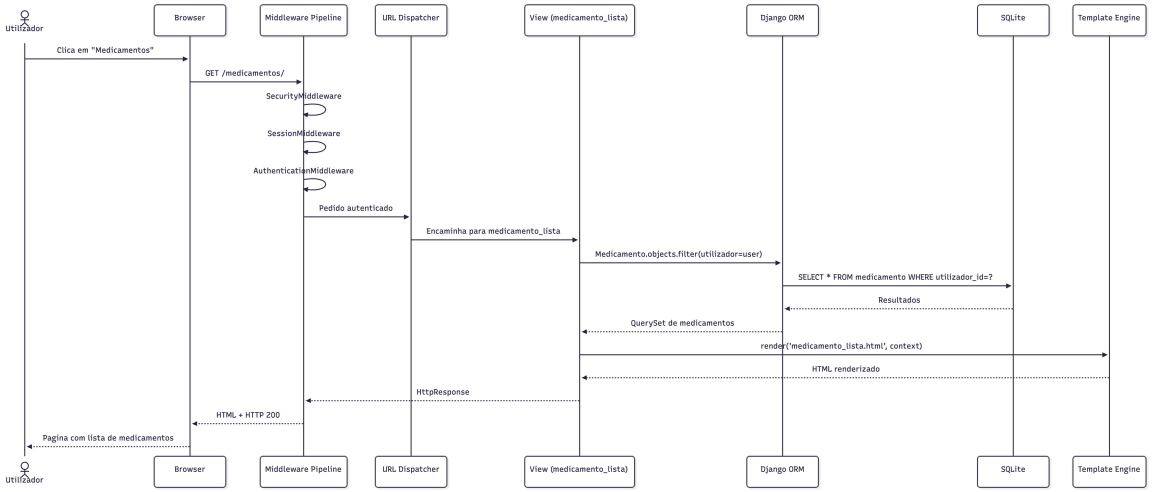

Figura 5 — Diagrama de Sequência: Listar Medicamentos

O pedido passa primeiro pelo pipeline de middleware (segurança, sessão, autenticação, CSRF), depois é encaminhado pelo URL Dispatcher para a view medicamento_lista. Esta view consulta a base de dados através do ORM, filtrando apenas os medicamentos do utilizador autenticado, e passa os resultados ao template para renderização. O HTML final é devolvido ao browser.

4.6. Padrões de Desenho no Django

Uma das riquezas do Django é a quantidade de padrões de desenho (design patterns) que incorpora. Identificar e compreender estes padrões é um exercício valioso para quem estuda arquitectura de software, pois demonstra como conceitos teóricos se materializam em código real.

4.6.1. Active Record (nos Models)

O padrão Active Record, descrito por Martin Fowler, propõe que cada objecto do modelo de dados seja responsável por se guardar, actualizar e eliminar da base de dados. No Django, isto manifesta-se nos métodos save() e delete() disponíveis em cada instância de modelo, e nos

métodos de consulta do objects manager (como filter(), get(), all()). Por exemplo, ao fazer medicamento.save(), o próprio objecto sabe como gerar e executar o comando SQL correspondente.

4.6.2. Front Controller (no URL Dispatcher)

O URL Dispatcher do Django implementa o padrão Front Controller: todos os pedidos HTTP entram por um ponto único e são encaminhados para a view apropriada com base no URL. No DomusShelf, o ficheiro domusshelf_project/urls.py define os caminhos de nível superior, delegando os caminhos específicos da farmácia para pharmacy/urls.py através do include().

4.6.3. Template Method (nas Views)

Embora o DomusShelf utilize function-based views (views baseadas em funções), o Django disponibiliza também class-based views (CBVs) que implementam o padrão Template Method. Neste padrão, uma classe base define o esqueleto de um algoritmo e as subclasses podem sobrescrever passos específicos sem alterar a estrutura geral. Os ModelForms do DomusShelf seguem uma lógica semelhante: o Django define como um formulário é validado e guardado, e o programador apenas especifica quais campos incluir e como os apresentar.

4.6.4. Chain of Responsibility (no Middleware)

O pipeline de middleware do Django é um exemplo clássico do padrão Chain of Responsibility. Cada pedido HTTP passa por uma cadeia de middlewares — SecurityMiddleware, SessionMiddleware, AuthenticationMiddleware, CsrfViewMiddleware, entre outros — onde cada um processa o pedido e decide se o passa ao próximo elo da cadeia ou se devolve uma resposta imediata (por exemplo, um erro 403 se o token CSRF for inválido). Esta cadeia é configurada na lista MIDDLEWARE do settings.py.

4.6.5. Observer (nos Signals)

O Django possui um mecanismo de signals (sinais) que implementa o padrão Observer. Sinais como post_save e pre_delete permitem que determinado código seja executado automaticamente quando um evento ocorre. Embora o DomusShelf não implemente signals explicitamente, o mecanismo está disponível e seria útil para, por exemplo, criar automaticamente um registo de Preferências quando um novo utilizador se regista.

4.6.6. Decorator (nos View Decorators)

O padrão Decorator permite adicionar comportamento a funções existentes sem as modificar. No DomusShelf, o decorador @login_required é usado extensivamente para proteger as views — se um utilizador não autenticado tentar aceder a uma view protegida, é automaticamente redireccionado para a página de login. A view de registo de utilizadores (registo) é,

propositadamente, a única view pública sem este decorador, pois um utilizador não pode estar autenticado antes de se registar.

4.7. Conceitos Transversais

4.7.1. Autenticação e Autorização

A autenticação (verificar quem é o utilizador) é gerida pelo módulo django.contrib.auth, que fornece o modelo User, formulários de login/registo, e middleware de sessão. O DomusShelf implementa registo autónomo de utilizadores através do UserCreationForm do Django, integrado com o messages framework para fornecer feedback após a criação da conta. A autorização (verificar o que o utilizador pode fazer) é implementada ao nível das views, filtrando sempre os dados pelo utilizador autenticado (request.user). Assim, cada utilizador apenas vê e gere os seus próprios medicamentos, embalagens e consumos.

4.7.2. Protecção CSRF

O Django inclui protecção contra ataques Cross-Site Request Forgery (CSRF) activada por defeito. Todos os formulários HTML incluem um token CSRF (gerado pela tag {% csrf_token %}) que é validado pelo CsrfViewMiddleware. Sem este token, o formulário é rejeitado, prevenindo que sites maliciosos submetam formulários em nome do utilizador.

4.7.3. Herança de Templates

O sistema de templates do Django suporta herança, seguindo o princípio DRY (Don't Repeat Yourself). O DomusShelf define um template base (base.html) que contém a estrutura comum a todas as páginas: navbar, footer, carregamento de CSS e JavaScript (incluindo Bootstrap 5, Bootstrap Icons, Google Fonts Inter e Flatpickr), e sistema de mensagens. Cada página específica herda deste template base e define apenas o conteúdo particular no bloco {% block content %}. Isto significa que alterações ao layout geral são feitas num único ficheiro e propagam-se automaticamente a todas as páginas.

4.7.4. Context Processor de Alertas

Para que a contagem de alertas (embalagens expiradas ou a expirar) apareça em todas as páginas no ícone do sino da navbar, foi implementado um context processor personalizado. Um context processor é uma função que recebe o pedido HTTP e devolve um dicionário de variáveis que é automaticamente adicionado ao contexto de todos os templates. O ficheiro context_processors.py define a função alertas_count que calcula o número de embalagens problemáticas do utilizador, considerando as preferências configuradas (número de dias de antecedência), e disponibiliza esta contagem globalmente.

4.7.5. Localização e Internacionalização

A aplicação foi configurada para Português de Portugal (pt-pt) através de várias configurações em settings.py: LANGUAGE_CODE define o idioma, TIME_ZONE define o fuso horário (Europe/Lisbon), USE_L10N activa a formatação local de números e datas, e DATE_INPUT_FORMATS define os formatos de data aceites nos formulários (dd/mm/aaaa como formato preferido). O widget Flatpickr foi integrado com a localização portuguesa para apresentar o calendário de selecção de datas em Português.

4.8. Decisões de Arquitectura (ADR)

As decisões arquitecturais mais relevantes tomadas durante o desenvolvimento do DomusShelf são documentadas a seguir no formato ADR (Architecture Decision Record), que regista o contexto, a decisão e as suas consequências.

ADR-01: Django como framework de desenvolvimento

Contexto: Era necessário escolher uma framework web Python para o projecto. **Decisão:** Utilizar Django 4.2 LTS. **Razões:** Filosofia "batteries included" que inclui ORM, autenticação, admin e formulários de raiz; versão LTS com suporte garantido até Abril de 2026; recomendação do professor. **Alternativas consideradas:** Flask (demasiado minimalista para as funcionalidades necessárias) e FastAPI (orientado a APIs, não a renderização de HTML). **Consequências:** Desenvolvimento mais rápido, mas maior curva de aprendizagem inicial.

ADR-02: SQLite como base de dados

Contexto: Era necessário escolher um motor de base de dados. **Decisão:** Manter o SQLite (por defeito no Django). **Razões:** Zero configuração, um único ficheiro, suficiente para desenvolvimento e demonstração. **Alternativas consideradas:** PostgreSQL (mais robusto mas requer instalação e configuração de servidor). **Consequências:** Simplicidade total na instalação, mas sem suporte para acessos concorrentes intensivos. Migração para PostgreSQL possível sem alterar código graças ao ORM.

ADR-03: Renderização no servidor (SSR) com Bootstrap

Contexto: Era necessário decidir entre uma aplicação SPA (Single Page Application) com API REST ou renderização tradicional no servidor. **Decisão:** Renderização no servidor com templates Django e Bootstrap 5. **Razões:** Menor complexidade, sem necessidade de framework JavaScript front-end separado, alinhamento com os objectivos da cadeira (compreender Django). **Alternativas consideradas:** React/Vue.js com Django REST Framework. **Consequências:** Páginas recarregam a cada interacção, mas desenvolvimento mais simples e focado no Django.

ADR-04: Isolamento de dados por utilizador nas views

Contexto: A aplicação suporta múltiplos utilizadores e os dados de cada um devem ser privados. **Decisão:** Filtrar todos os querysets por request.user nas views, em vez de usar permissões por objecto. **Razões:** Simplicidade e eficácia — cada query inclui o filtro do utilizador. **Consequências:** Segurança garantida ao nível da view, sem necessidade de configuração complexa de permissões.

ADR-05: Function-based views em vez de class-based views

Contexto: O Django oferece dois estilos de views: baseadas em funções (FBV) e baseadas em classes (CBV). **Decisão:** Utilizar function-based views para toda a aplicação. **Razões:** Maior clareza e explicitude do fluxo, alinhamento com o Django Girls Tutorial recomendado pelo professor, facilidade de compreensão para quem está a aprender Django. **Consequências:** Alguma repetição de código entre views de CRUD, mas maior controlo e transparência.

5. IMPLEMENTAÇÃO

5.1. Tecnologias e Versões

Componente	Tecnologia	Versão
Linguagem	Python	3.12
Framework Web	Django	4.2.27 (LTS)
Base de Dados	SQLite	3.x (embutido)
Framework CSS	Bootstrap	5.3.2
Ícones	Bootstrap Icons	1.11.1
Date Picker	Flatpickr	4.x
Fonte	Inter (Google Fonts)	—
Controlo de Versões	Git + GitHub	—

Tabela 4 — Stack Tecnológico

5.2. Modelos de Dados

Os modelos de dados foram definidos no ficheiro pharmacy/models.py. Cada classe Python herda de models.Model e os seus atributos definem os campos da tabela correspondente na base de dados. A listagem seguinte mostra o modelo Medicamento como exemplo representativo:

Listagem 2 — Modelo Medicamento (simplificado)

```
class Medicamento(models.Model):
    utilizador = models.ForeignKey(
        User, on_delete=models.CASCADE,
        related_name='medicamentos'
    )
    nome_comercial = models.CharField(max_length=200)
    principio_activo = models.CharField(max_length=200, blank=True)
    forma_farmaceutica = models.CharField(max_length=100, blank=True)
    observacoes = models.TextField(blank=True)
    criado_em = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['nome_comercial']
```

O campo utilizador estabelece a relação com o modelo User do Django, onde on_delete=models.CASCADE garante que, se um utilizador for eliminado, todos os seus medicamentos são também eliminados (integridade referencial). O

related_name='medicamentos' permite aceder a todos os medicamentos de um utilizador através de user.medicamentos.all().

O modelo Embalagem inclui dois atributos calculados implementados como propriedades Python: esta_expirada (que compara a data de validade com a data actual) e dias_para_expirar (que calcula a diferença em dias). Estas propriedades permitem lógica de negócio sem queries adicionais à base de dados. O método __str__() da Embalagem foi refinado na Fase 10 para incluir condicionalmente o número de lote, melhorando a usabilidade do dropdown no registo de consumos.

5.3. Views e Processamento de Pedidos

As views foram implementadas como funções Python (function-based views), cada uma decorada com @login_required para garantir que apenas utilizadores autenticados acedem às funcionalidades. A listagem seguinte mostra a view do dashboard como exemplo:

Listagem 3 — View do Dashboard (simplificada)

```
@login_required
def dashboard(request):
    hoje = date.today()
    prefs, _ = Preferencias.objects.get_or_create(
        utilizador=request.user,
        defaults={'dias_alerta_antes': 30}
    )
    data_limite = hoje + timedelta(days=prefs.dias_alerta_antes)
    context = {
        'total_medicamentos': Medicamento.objects.filter(
            utilizador=request.user).count(),
        'expiradas': Embalagem.objects.filter(
            medicamento__utilizador=request.user,
            data_validade__lt=hoje).count(),
    }
    return render(request, 'pharmacy/dashboard.html', context)
```

Destaca-se o uso da notação de duplo underscore (medicamento__utilizador) para atravessar relações entre modelos nas queries, e o método get_or_create() que obtém as preferências do utilizador ou cria-as com valores por defeito caso ainda não existam.

5.4. Context Processor de Alertas

O context processor de alertas é uma peça fundamental da arquitectura, pois garante que a contagem de alertas está disponível em todas as páginas sem que cada view precise de a calcular individualmente:

Listagem 4 — Context Processor de Alertas (simplificado)

```
def alertas_count(request):
    if not request.user.is_authenticated:
        return {'alertas_count': 0}
    hoje = date.today()
    prefs, _ = Preferencias.objects.get_or_create(
        utilizador=request.user,
        defaults={'dias_alerta_antes': 30}
    )
    data_limite = hoje + timedelta(days=prefs.dias_alerta_antes)
    count = Embalagem.objects.filter(
        medicamento__utilizador=request.user,
        quantidade_actual__gt=0,
        data_validade__lte=data_limite
    ).count()
    return {'alertas_count': count}
```

5.5. Registo de Utilizadores

Na Fase 10 do desenvolvimento, foi implementada a funcionalidade de registo autónomo de utilizadores, permitindo que qualquer pessoa crie a sua própria conta sem intervenção de um administrador. Esta funcionalidade utiliza o UserCreationForm built-in do Django, que inclui validação de palavra-passe (comprimento mínimo, complexidade, etc.). Após o registo com sucesso, o utilizador é redireccionado para a página de login com uma mensagem de sucesso, utilizando o messages framework do Django.

Listagem 5 — View de Registo de Utilizador

```
def registo(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request,
                'Conta criada com sucesso! Pode agora iniciar
sessao.')
            return redirect('login')
```

```
    else:
        form = UserCreationForm()
    return   render(request,   'registration/registo.html',   {'form':
form})
```

5.6. Formulários

Os formulários utilizam a classe ModelForm do Django, que gera automaticamente os campos a partir dos modelos. Um aspecto relevante é a filtragem dinâmica do dropdown de medicamentos no formulário de embalagens: o construtor __init__ recebe o utilizador como parâmetro e filtra o queryset do campo medicamento para mostrar apenas os medicamentos do utilizador actual. Isto garante que um utilizador nunca possa associar uma embalagem a um medicamento de outro utilizador.

O formulário de consumo inclui validação personalizada no método clean() que verifica se a quantidade a consumir não excede a quantidade disponível na embalagem, impedindo stocks negativos.

5.7. Sistema de URLs e Namespaces

O sistema de URLs do DomusShelf utiliza namespaces para organização. O ficheiro domusshelf_project/urls.py define as rotas de nível superior (incluindo autenticação e registo de utilizadores) e delega para pharmacy/urls.py os caminhos específicos da aplicação. O namespace pharmacy (definido pelo app_name) permite referenciar URLs nos templates de forma não ambígua, por exemplo: {% url 'pharmacy:medicamento_lista' %}. A tabela seguinte resume todas as URLs implementadas:

URL	Função
/	Dashboard (página inicial)
/admin/	Painel de administração Django
/accounts/login/	Página de login
/accounts/logout/	Logout (redireciona para login)
/accounts/registo/	Registo de novo utilizador
/medicamentos/	Lista do catálogo de medicamentos
/medicamentos/novo/	Formulário de criação de medicamento
/medicamentos/<id>/editar/	Formulário de edição
/medicamentos/<id>/eliminar/	Confirmação de eliminação
/medicamentos/stock/	Lista de embalagens (FEFO)
/medicamentos/stock/nova/	Adicionar embalagem
/medicamentos/consumo/novo/	Registar toma/consumo
/medicamentos/alertas/	Lista de alertas de validade
/medicamentos/preferencias/	Configuração de alertas

Tabela 5 — URLs Implementadas

5.8. Painel de Administração

O Django Admin é uma interface administrativa gerada automaticamente a partir dos modelos. No DomusShelf, cada modelo foi registado com configurações personalizadas que definem quais colunas aparecem na listagem (list_display), quais campos são pesquisáveis (search_fields), e quais filtros estão disponíveis (list_filter). No modelo Embalagem, foi adicionado um método personalizado estado_validade que mostra visualmente se cada embalagem está expirada, a expirar, ou em bom estado.

Figura 6 — Painel de Administração do DomusShelf

5.9. Interface Gráfica

A interface do DomusShelf utiliza Bootstrap 5 com uma paleta de cores baseada em vermelho escuro (#8B0000), remetendo para o tema farmacêutico. O template base (base.html) define a estrutura comum: uma navbar responsiva com links de navegação, ícone de sino para alertas com badge dinâmico, menu dropdown do utilizador (com acesso a Preferências, Administração e Sair), e footer com copyright. A fonte Inter (Google Fonts) foi escolhida pela sua legibilidade em ecrã. As datas são apresentadas através do widget Flatpickr com localização portuguesa, que oferece um calendário visual elegante em formato dd/mm/aaaa.

Todas as páginas herdam do template base, redefinindo apenas o bloco de conteúdo. Os formulários utilizam as classes CSS do Bootstrap (form-control, form-select) configuradas directamente nos widgets dos ModelForms, assegurando consistência visual. A interface é responsiva, adaptando-se automaticamente a dispositivos móveis, tablets e desktops graças ao sistema de grelha do Bootstrap e à navbar com menu hamburger.

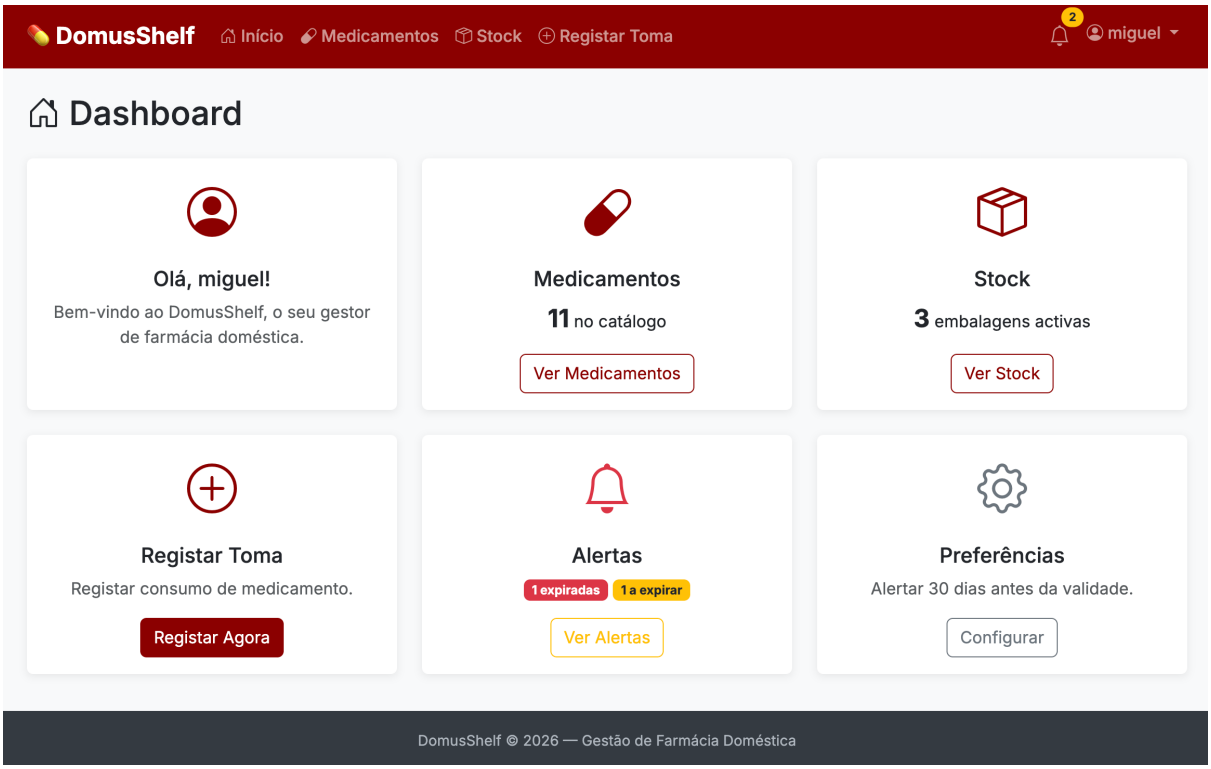

Figura 7 — Dashboard do DomusShelf

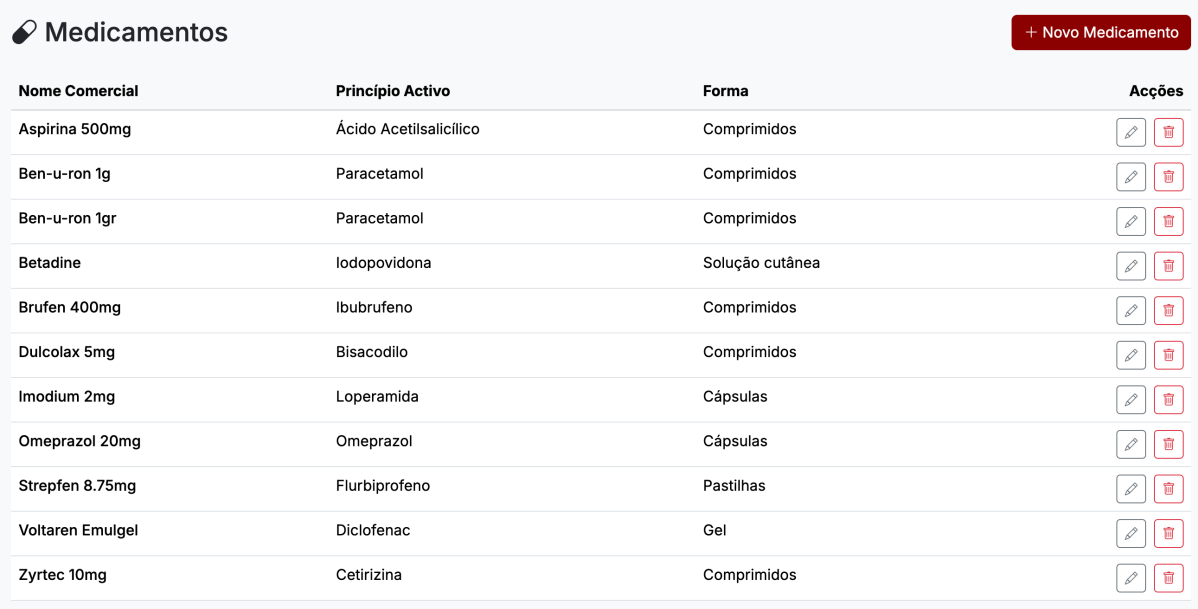

Figura 8 — Lista de Medicamentos

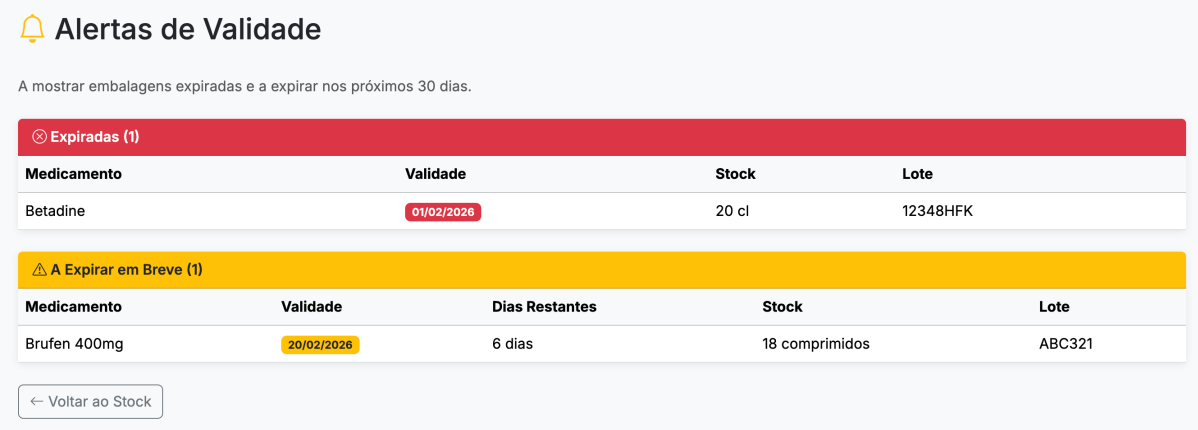

Figura 9 — Página de Alertas

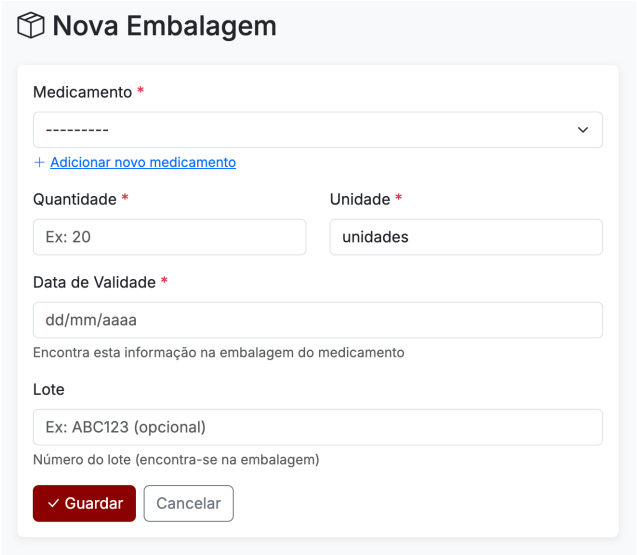

Figura 10 — Formulário de Nova Embalagem

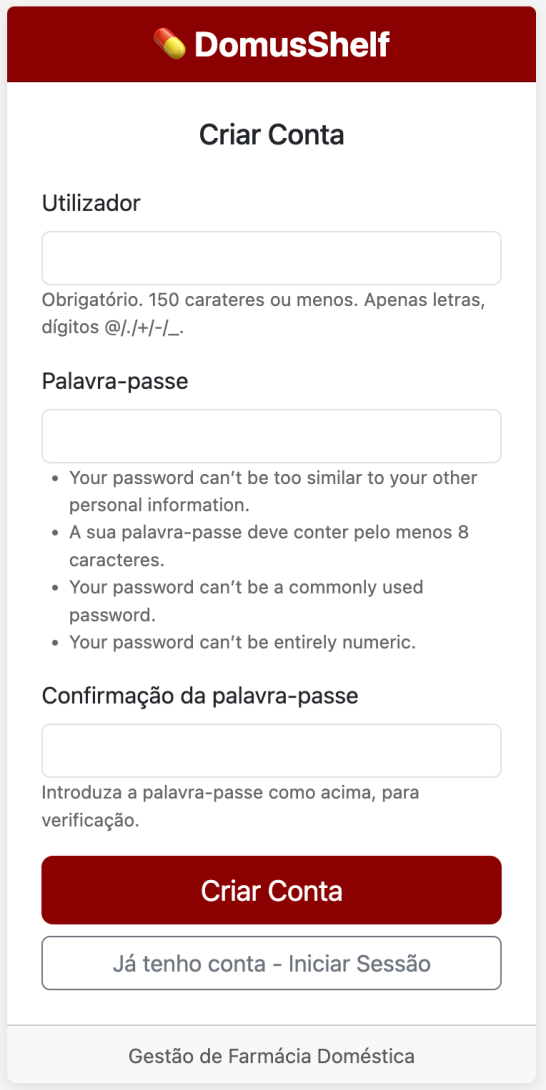

Figura 11 — Página de Registo de Novo Utilizador

6. Processo de Desenvolvimento

6.1. Metodologia Adoptada

O desenvolvimento do DomusShelf seguiu uma abordagem incremental e iterativa, organizada em fases (sprints) que permitiram validar o progresso continuamente. Esta abordagem foi especialmente adequada ao contexto académico, pois cada fase produzia funcionalidade testável e o professor podia acompanhar a evolução através do histórico de commits no GitHub.

A utilização do Claude AI como ferramenta de apoio ao desenvolvimento permitiu esclarecer dúvidas conceptuais, obter orientação sobre boas práticas Django, e receber instruções técnicas detalhadas que foram depois implementadas e testadas pelo autor. Esta abordagem acelerou o processo de aprendizagem, mantendo sempre o controlo e compreensão sobre todas as decisões técnicas.

6.2. Fases de Desenvolvimento

O projecto foi organizado em onze fases distintas, desde a configuração inicial do ambiente até às melhorias finais de usabilidade:

Fase	Descrição	Data
Fase 0	Limpeza do repositório (.gitignore, remoção do venv)	2 Fev 2026
Fase 1	Modelos de dados (4 entidades + migrações + admin)	3 Fev 2026
Fase 2	Sistema de autenticação (login/logout personalizados)	3 Fev 2026
Fase 3	Template base com Bootstrap 5, navbar responsiva	3 Fev 2026
Fase 4	CRUD completo de medicamentos	4 Fev 2026
Fase 5	CRUD de embalagens com FEFO e indicadores visuais	4 Fev 2026
Fase 6	Registo de consumos com desconto automático de stock	4 Fev 2026
Fase 7	Dashboard dinâmico e sistema de alertas	5 Fev 2026
Fase 8	Página de preferências do utilizador	5 Fev 2026
Fase 9	Polimento: localização, Flatpickr, dados demo, README	5 Fev 2026
Fase 10	Registo de utilizadores e lote no dropdown de consumos	13 Fev 2026

Tabela 6 — Fases de Desenvolvimento

6.3. Controlo de Versões

O código fonte foi mantido num repositório Git público, no GitHub (https://github.com/miguelascensaoreal/DomusShelf), para que o professor pudesse acompanhar o progresso. Os commits foram feitos com frequência e com mensagens descritivas em Português, reflectindo a evolução do trabalho. Por exemplo: "Implementa modelos de dados: Medicamento, Embalagem, Consumo, Preferencias", "Adiciona CRUD de

medicamentos com filtro por utilizador", ou "Adiciona Flatpickr para date picker em formato português (dd/mm/aaaa)".

Um problema encontrado na Sessão 1 foi a inclusão acidental da pasta venv/ no repositório. Este problema foi resolvido na Fase 0 através da criação de um ficheiro .gitignore adequado e da remoção dos ficheiros indesejados do histórico do Git usando git rm -r --cached. Esta experiência reforçou a importância de configurar o .gitignore logo no início de qualquer projecto.

6.4. Problemas Encontrados e Soluções

Ao longo do desenvolvimento foram encontrados vários problemas que exigiram diagnóstico e resolução:

Problema	Solução Aplicada
Ambiente virtual corrompido após actualização do macOS	Apagar e recriar o venv: rm -rf venv && python3 -m venv venv
Datas interpretadas como mm/dd/aaaa em vez de dd/mm/aaaa	Configurar DATE_INPUT_FORMATS em settings.py e integrar Flatpickr com localização PT
Links da navbar a dar erro 404	Substituir URLs hardcoded por {% url 'pharmacy:nome' %}
Import errado de Preferencias	Corrigir import para usar .models em vez de .forms
Mesma validade em várias embalagens dificulta identificação	Adicionar número de lote ao __str__() da Embalagem (Fase 10)

Tabela 7 — Problemas Encontrados e Soluções

6.5. Ambiente de Desenvolvimento

O desenvolvimento foi realizado num MacBook Air com processador Apple M3, utilizando o Visual Studio Code como editor de código. O Python 3.12 e o Django 4.2.27 foram instalados num ambiente virtual isolado. O servidor de desenvolvimento do Django (python manage.py runserver) foi utilizado para testes locais, acedido através do browser Safari, Orion e Google Chrome. A responsividade móvel foi testada através do Responsive Design Mode do Safari, após dificuldades iniciais com o browser Orion, que não disponibilizava ferramentas de desenvolvimento adequadas.

7. Testes e Validação

7.1. Estratégia de Testes

A validação do DomusShelf foi realizada através de testes manuais sobre todos os fluxos funcionais da aplicação. Cada requisito funcional identificado no Capítulo 3 foi testado individualmente para verificar o seu correcto funcionamento. Foram utilizados dois utilizadores de teste: o utilizador "miguel" (superuser de desenvolvimento) e o utilizador "Professor" (superuser de demonstração), permitindo verificar o isolamento de dados entre utilizadores.

7.2. Testes Funcionais Realizados

Requisito	Teste Realizado	Resultado
RF01	Registo de novo utilizador e login/logout	Conforme
RF02	Criar, listar, editar e eliminar medicamentos	Conforme
RF03	Adicionar, listar, editar e eliminar embalagens com datas de validade	Conforme
RF04	Verificação de alertas no sino e na página de alertas	Conforme
RF05	Registar consumo e verificar desconto no stock	Conforme
RF06	Tentar consumir mais do que o stock disponível	Conforme (erro de validação)
RF07	Alterar dias de antecedência nas preferências	Conforme
RF08	Dashboard com contagens correctas	Conforme
RF09	Acesso e gestão através do Django Admin	Conforme
RF10	Tentar aceder a medicamentos de outro utilizador	Conforme (erro 404)
RF11	Criar conta, verificar redirecionamento e mensagem de sucesso	Conforme
RF12	Verificar que o lote aparece no dropdown de consumos	Conforme

Tabela 8 — Resultados dos Testes Funcionais

7.3. Validação de Segurança

Foram realizados testes básicos de segurança: tentativa de acesso a URLs protegidas sem autenticação (redireccionamento para login confirmado), tentativa de submissão de formulários sem token CSRF (pedido rejeitado pelo middleware), e tentativa de acesso a dados de outro utilizador manipulando o ID na URL (resposta 404 confirmada, graças ao filtro por utilizador nas views).

7.4. Validação de Responsividade

A interface foi testada em múltiplas resoluções através do Responsive Design Mode do Safari, verificando que todos os elementos se adaptam correctamente a dispositivos móveis (navbar colapsa para menu hamburger, tabelas são scrolláveis, formulários ocupam a largura total do ecrã). A abordagem mobile-first do Bootstrap 5 garantiu que a interface funciona adequadamente em ecrãs de diferentes dimensões.

8. CONCLUSÕES

8.1. Objectivos Alcançados

O projecto DomusShelf cumpriu os objectivos estabelecidos: foi desenvolvida uma aplicação web funcional em Django que permite a gestão completa de uma farmácia doméstica, com operações CRUD sobre medicamentos e embalagens, registo de consumos com desconto automático de stock, sistema de alertas configurável, registo autónomo de utilizadores, e interface responsiva. Para além da componente funcional, o projecto permitiu explorar e documentar a arquitectura MTV do Django, identificar múltiplos padrões de desenho embebidos na framework, e compreender como decisões arquitecturais se materializam em código concreto.

Todos os requisitos obrigatórios do enunciado foram cumpridos: a aplicação funciona e tem complexidade adequada, a arquitectura está documentada (este relatório), o manual de utilizador foi produzido e entregue como documento separado, os dados são armazenados em base de dados (SQLite), o código está sob controlo de versões (Git), o repositório está no GitHub, e o README explica como operacionalizar a aplicação.

8.2. Competências Adquiridas

O desenvolvimento deste projecto proporcionou a aquisição de competências em diversas áreas: compreensão profunda da framework Django e do seu ecossistema (ORM, templates, middleware, admin, context processors, messages framework), aplicação prática de padrões de desenho de software em contexto real (Active Record, Front Controller, Template Method, Chain of Responsibility, Observer, Decorator), desenvolvimento de interfaces web responsivas com Bootstrap 5, implementação de mecanismos de autenticação e autorização, configuração de localização e internacionalização, práticas de controlo de versões com Git e GitHub, e documentação de arquitectura de software seguindo standards reconhecidos.

8.3. Limitações e Trabalho Futuro

As principais limitações da versão actual incluem a utilização de SQLite (que não suporta acessos concorrentes intensivos), a ausência de notificações por email para alertas de validade, e a inexistência de testes automatizados. Como trabalho futuro, propõe-se a migração para PostgreSQL para suporte a múltiplos utilizadores em simultâneo, a implementação de notificações por email utilizando o sistema de email do Django, a adição de um módulo de relatórios e estatísticas avançadas com gráficos, a criação de uma API REST com Django REST Framework para suportar uma futura aplicação móvel, e a implementação de testes unitários e de integração com o módulo django.test.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Django Software Foundation (2025). Django documentation — Version 4.2. Disponível em: https://docs.djangoproject.com/en/4.2/

[2] Django Software Foundation (2025). Design philosophies. Disponível em: https://docs.djangoproject.com/en/5.1/misc/design-philosophies/

[3] Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.

[4] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

[5] Starke, G. & Hruschka, P. (2025). arc42 — Template for Software Architecture Documentation. Disponível em: https://arc42.org/

[6] Brown, S. (2025). The C4 model for visualising software architecture. Disponível em: https://c4model.com/

[7] Bootstrap (2024). Bootstrap 5.3 Documentation. Disponível em: https://getbootstrap.com/docs/5.3/

[8] Kruchten, P. (1995). The 4+1 View Model of Architecture. IEEE Software, 12(6), 42-50.

[9] Django Girls (2025). Django Girls Tutorial. Disponível em: https://tutorial.djangogirls.org/pt/

[10] Flatpickr (2024). Flatpickr Documentation. Disponível em: https://flatpickr.js.org/

NOTA SOBRE UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL

No desenvolvimento deste projecto e na elaboração da respectiva documentação foi utilizado o Claude AI da Anthropic como ferramenta de apoio. O Claude auxiliou nas seguintes áreas: esclarecimento de conceitos técnicos sobre Django e padrões de desenho, orientação sobre boas práticas de desenvolvimento web com Python, sugestões de arquitectura e estruturação do código, revisão e estruturação da documentação, e diagnóstico de erros durante o desenvolvimento.

A utilização desta ferramenta permitiu acelerar o processo de aprendizagem e desenvolvimento, mantendo sempre o controlo e compreensão por parte do aluno sobre todas as decisões técnicas e conceptuais. A ideia original, a escolha das tecnologias e linguagens de programação, e as decisões de arquitectura foram da inteira responsabilidade do aluno. A LLM foi utilizada como ferramenta de apoio, não para realização de trabalho completo nem para decisões de engenharia do mesmo.