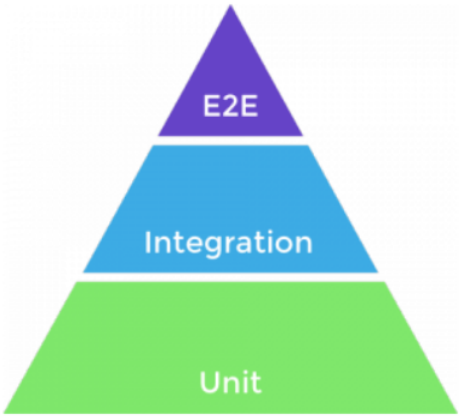
1 Plano de Testes (*draft*)

O presente documento tem como objetivo definir a estratégia global de testes a aplicar no projeto SeePaw.

Sendo um documento em versão *draft*, o plano será progressivamente atualizado ao longo do ciclo de desenvolvimento, acompanhando a evolução da aplicação e o aumento da complexidade dos testes nas próximas milestones.

1.1 Abordagem de Testes

A abordagem de testes adotada no projeto SeePaw baseia-se na pirâmide de testes, ilustrada na figura seguinte, que representa a distribuição equilibrada entre os diferentes tipos de teste ao longo do processo de desenvolvimento.



A pirâmide reflete o princípio de que quanto mais baixo o nível de teste, maior a quantidade e a rapidez de execução, enquanto os níveis superiores são menos numerosos, mas mais abrangentes e próximos da experiência real do utilizador.

A pirâmide de testes é composta por três camadas principais:

- **Testes Unitários:** formam a base da pirâmide e são executados em maior número. Verificam o correto funcionamento de métodos e componentes isolados.
- **Testes de Integração:** situam-se na camada intermédia e garantem que os diferentes módulos do sistema comunicam corretamente entre si (por exemplo, entre o controlador, os serviços e a base de dados).
- **Testes End-to-End (E2E) / Testes de Sistema:** representam o topo da pirâmide e validam o comportamento global da aplicação do ponto de vista do utilizador, simulando interações reais com a interface e verificando o cumprimento dos requisitos funcionais.

1.2 Ferramentas e Tecnologias de Teste

A tabela seguinte apresenta um resumo das tecnologias escolhidas e o respetivo propósito dentro da estratégia de testes.

Tipo de Teste	Objetivo Principal	Ferramentas/ Tecnologias	Observações
Unitário /Backend	Validar métodos e componentes isolados	xUnit (.NET Core)	Utilizado no backend; suporta mocking.
Unitário /Frontend Web	Validar métodos e componentes isolados	Jest (React)	Utilizado no frontend da web App;
Integração	Testar o fluxo entre componentes (controller → serviço → BD)	Postman	Postman para requests
End-to-End (E2E) / Aceitação	Simular o comportamento real do utilizador	Selenium	Automatiza cenários críticos no frontend web.
Mobile (Android)	Testar fluxos ecrã a ecrã na aplicação móvel	JUnit / Espresso	Testes nativos integrados com o Android Studio; executados na Milestone 4.

1.3 Critérios de Cobertura e Prioridades de Teste

Nem todas as funcionalidades da aplicação SeePaw exigem o mesmo nível de profundidade nos testes. Desta forma, as funcionalidades foram classificadas como básicas, intermédias, e críticas, de acordo com a sua importância funcional para o projeto. Esta classificação encontra-se representada na seguinte tabela.

Categoria	Exemplos de Funcionalidades	Prioridade	Cobertura Esperada	Justificação
Básicas	Registo e autenticação de utilizadores, listagem e filtragem de animais, gestão de favoritos	Média	~40 %	Garantem a navegação e interação essencial com o sistema, mas não envolvem dados sensíveis nem operações críticas.
Intermédias	Atualização de dados de	Alta	50–70 %	São funcionalidades com interação frequente e múltiplas

	animais, upload de imagens			dependências, exigindo validação de consistência e desempenho.
Críticas	Processos de *Ownership* e *Fostering*, agendamento de *slots* para visitas ou passeios com o animal	Muito Alta	80–100 %	Representam as funcionalidades centrais e mais relevantes da aplicação. A sua fiabilidade é essencial para o sucesso do sistema.

É importante salientar que uma elevada taxa de cobertura por si só não garante qualidade nos testes.
Mais relevante do que o número de linhas cobertas é assegurar que todas as ramificações lógicas e cenários possíveis são devidamente verificados. Por isso, os testes devem ser concebidos com base na complexidade ciclomática dos métodos, garantindo que cada caminho relevante é exercitado e validado, sobretudo nas funcionalidades críticas do sistema.

1.4 Integração com o Fluxo de Desenvolvimento e CI/CD

A estratégia de testes da aplicação SeePaw está diretamente integrada no fluxo de desenvolvimento e nas pipelines de CI/CD.

A estratégia de versionamento do projeto SeePaw baseia-se em duas branches principais; develop e main, complementadas por branches temporárias de feature.
A tabela seguinte apresenta a função de cada uma e os testes que nelas são executados.

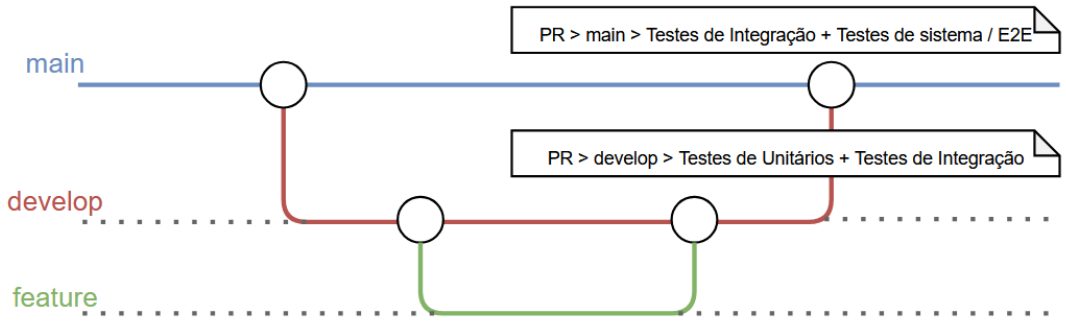
Branch	Função	Tipo de Testes Executados
feature/	Desenvolvimento de novas funcionalidades isoladas	Testes unitários automáticos, garantindo que cada componente funciona corretamente antes de integração.
develop	Integração contínua e validação do código em conjunto	Testes de integração, que verificam o funcionamento entre camadas (controller, serviço, repositório e base de dados).

main	Versão estável, pronta para entrega ou demonstração	Testes finais de sistema e aceitação, executados antes de publicação ou entrega.

1.4.1 Execução dos Testes nas Pipelines

Os testes são executados automaticamente em momentos específicos do fluxo de desenvolvimento, garantindo que cada alteração é devidamente validada antes de ser integrada nas branches principais.

- Pull Request para develop: aciona a pipeline de integração contínua, que executa testes unitários e de integração, assegurando que o código introduzido funciona corretamente de forma isolada e em conjunto com os restantes módulos do sistema.
- Pull Request para main: executa uma pipeline completa de validação, incluindo testes de integração e de sistema, garantindo a estabilidade e consistência da versão final antes da entrega ou publicação.



Os testes não são executados em cada commit de feature para evitar sobrecarga e consumo desnecessário de minutos no GitHub Actions (plano gratuito limitado a 2000 min/mês). As verificações correm localmente via *pre-commit hook*, garantindo cobertura mínima. As pipelines são executadas apenas em *pull requests*: testes unitários e de integração na develop, testes de sistema na main.

1.5 Quality Gates

Os *Quality Gates* são verificações automáticas de qualidade que atuam nas pipelines de integração contínua, assegurando que apenas código devidamente testado e conforme aos critérios definidos é integrado nas branches principais do projeto.

A tabela seguinte apresenta os principais critérios de verificação de qualidade aplicados durante o processo de integração contínua. Estes critérios definem os requisitos mínimos que o código deve cumprir antes de ser integrado.

Tipo de Critério	Condição Mínima	Aplicação
Execução de Testes	Todos os testes devem passar com sucesso	develop, main
Cobertura de Código	Ver tabela da Secção 1.3	develop, main
Inspeção de código	< 10 avisos não críticos Complexidade ciclomática < 14	develop, main
Erros Críticos	Nenhum erro de compilação, ou build	develop, main
Testes E2E	≥ 90% de sucesso (quando aplicável)	main