Laboratório de Desenvolvimento de Software

Plano de Projeto

Grupo 2

Conteúdo

1. Elementos de Referência

1.1. Objetivo do Projeto

O principal objetivo deste projeto é o de alicerçar a plataforma **SeePaw** através de um projeto de **backend**, uma **aplicação web** e uma **aplicação android**.

O backend trata da lógica de negócio e da interação com a base de dados. Expõe um conjunto de endpoints, que permitem o acesso e manipulação de informação para as aplicações.

A aplicação web é a interface principal dos utilizadores finais, que podem ser pessoas interessadas em apadrinhar ou adotar um animal, ou gestores de centros de acolhimento animal. É através desta aplicação que são inseridos os animais com toda a sua informação relevante, são feitos os pedidos de *fostering* e de *ownership*, e agendados os pedidos de visita ou de recolha de animais.

Por fim a aplicação android, permite que utilizadores que procuram apadrinhar ou adotar um animal, que possam fazer os pedidos de *fostering* e *ownership* e agendar visitas ou recolha de animais.

1.2. Dados Gerais da Equipa de Desenvolvimento:

Ana Sofia Vaz (8230095@estg.ipp.pt) – *Software Engineer e Team Leader*

Cristiana Pinheiro (8230091@estg.ipp.pt) – *Software Engineer*

Joel Pinto (8230098@estg.ipp.pt) – *Software Engineer*

Rui Araújo (8230112@estg.ipp.pt) – *Software Engineer*

Papel do *Team Leader*

- Coordenar o trabalho da equipa e garantir o cumprimento do plano do projeto.
- Atribuir tarefas e apoiar a resolução de constrangimentos.
- Enviar convocatórias para as reuniões e preparar ordem de trabalho das mesmas
- Assegurar a qualidade do código e o cumprimento das boas práticas.

Papel do *Software Engineer*

- Implementar as funcionalidades atribuídas de acordo com os requisitos definidos.
- Escrever código limpo, testável e alinhado com as boas práticas da equipa.
- Colaborar na revisão de código e na resolução de problemas técnicos.
- Participar nas reuniões de equipa e contribuir para a melhoria contínua do processo.

1.3. Regulamento interno

1.3.1 Funcionamento Interno da Equipa

a. **Na ausência do *Team Leader***, por questões pessoais ou profissionais, este cargo será assegurado temporariamente por Cristiana Pinheiro.
b. **Reuniões semanais**: será agendada semanalmente pelo menos uma reunião semanal, em que as convocatórias (ver **modelo A**) para as mesmas serão enviadas com 48h de antecedência. A quantidade de reuniões semanais será de acordo com as necessidades de formalização de decisões da equipa, mas o previsto é o de uma reunião semanal.
c. As reuniões podem ser em formato presencial ou online.
d. As reuniões online serão feitas usando a plataforma Teams.
e. **A convocatória para as reuniões** será elaborada pelo *Team Leader* e posteriormente enviada por email institucional dos restantes membros da equipa
f. As atas de reunião (**modelo B**) são preenchidas pelo *Team Leader* e assinadas por todos os membros da equipa. As atas são homologadas até 72h após a reunião.
g. A avaliação de cada membro da equipa deve ser feita em todos os milestones.
h. A falta a uma reunião ou não cumprimento das datas de entrega das tarefas, sem uma justificação válida, serão penalizadas com uma infração de 5% na nota final dessa milestone.
i. O repositório dos projetos (separador **Repositories**) pertencem à organização SeePaw do Github (https://github.com/see-paw), bem como a gestão do *Product Backlog* (separador **Projects/SeePaw Development 2025**) e armazenamento dos diferentes artefactos. O projeto de backend encontra-se no repositório "**backend**", o projeto da web app encontra-se no repositório "**web-app**", o projeto de android encontrar-se no repositório de "**android-app**" e por fim todos os artefactos do projeto homologados encontram-se em "**project-artifacts**".
j. A plataforma de comunicação entre os membros da equipa será feita no Whatsapp num grupo criado para o efeito.

1.3.2 Normas de Desenvolvimento e de Controlo de Versões

k. Branches:

- Cada repositório contará com duas branches principais, a branch *main/master* que guarda o histórico oficial das releases, enquanto que a branch *develop* serve como branch de integração para novas funcionalidades; a branch develop é inicialmente criada a partir da main.
- Cada nova funcionalidade deve ser criada na sua própria branch, ramificada a partir da branch *develop*.
- A convenção adotada para a criação de branches é a de *conventional branching*. Deve ser estruturada por: **<tipo>/<descrição>** (ex.: **feature**/jwt-login, **feature**/animal-crud, **bugfix**/animal-not-found, **chore**/update-dependencies, **refactor**/user-utils, etc). Uso de descrições em minúsculas, números 0-9, separadas por hífen. Para informações detalhadas: https://conventional-branch.github.io/
- Quando a funcionalidade estiver concluída, deve ser integrada na branch *develop* através de um Pull Request.
- As correções de erros também são tratadas em branches dedicadas para o efeito e podem ser criadas a partir de *develop* ou de *main* (no caso de ser um problema crítico).

l. Commits:

- Os membros da equipa devem fazer *commits* regularmente com mensagens significativas.
  Cada *commit* deve tratar de um único objetivo, para melhorar a rastreabilidade e permitir um *rollback* fácil, se necessário.
- A convenção adotada para *commits* é a de *conventional commits* (https://www.conventionalcommits.org/en/v1.0.0/). Podem ser do tipo **feat** (feature), **refactor**, **fix** (bug fix), **chore** (atualização de dependências ou outras tarefas), **revert** (reversão de código), **docs** (inclusão ou ajustes de documentação), **ci** (mudanças de CI/CD). Pode ser usado o ponto de exclamação **"!"** para frisar a alteração de mudanças que **quebram código existente**. Devem ser estruturados da seguinte forma:
  <tipo>(âmbito opcional): <descrição>
  [corpo opcional]
  [footer opcional]

  Exemplos:
  revert: let us never again speak of the noodle incident,
  feat(auth): add new user authentication system,
  fix!: change API response format,

docs: correct spelling of CHANGELOG,

Com body e footer:

feat: add user authentication system

Implement JWT 0Auth2 auth flow with Google and GitHub providers.

Closes Issue#255

m. Pull Request:

- Os programadores submetem *merge/pull requests* para as branches *develop* ou *main*. Cada pedido deve incluir uma descrição detalhada de todas das alterações realizadas.
- **O pull request deve ser aberto tão cedo quanto possível**, isto é, logo que a *issue* correspondente esteja ***in progress*** na ***project board***. Deve ser criado em modo **draft pull request**, para que seja possível seguir as mudanças feitas ao código com maior facilidade, pelos commits enviados para a branch em que está a ser desenvolvida a nova funcionalidade.
- O **título do pull request (PR)** deve seguir o formato dos coventional commits, **e referenciar a issue (user story) que deu origem a esse PR**. O PR mantém todos os commits individuais da branch em que está a ser desenvolvida a nova feature, ao qual é adicionado um commit de merge adicional, cuja mensagem desse commit é o **título** desse PR.
  Exemplo: feat: implement user authentication system (#45).
  É importante que a issue correspondente seja mencionada no título do PR, para que seja mais fácil fazer a associação/tracking no github.
- Os *merge requests* para as branches *develop* ou *main* exigem revisão e aceitação por, pelo menos, **dois outros membros da equipa**, excluindo o autor do pedido. Os *reviewers* em questão têm que ser adicionados ao pedido de PR. O *assignee* é o autor do pull request.
- **A *label*** do PR deve ser de acordo com o tipo de pull request em questão (feature, bug, etc).
- Só depois de passarem com sucesso **todas as *pipelines*** na branch de funcionalidade ou na *develop* é que o *merge* pode ser aprovado pelo reviewer para ser incorporada na *develop* ou *main*, respetivamente.
  Fica a cargo do autor do PR resolver quaisquer conflitos/problemas que falhem na pipeline de CI/CD.
- O código deve passar nos **testes unitários** dentro da branch de funcionalidade antes de ser integrado na branch *develop*.
- O código deve passar nos **testes de integração** na branch *develop* antes de ser integrado na *main*.

n. Continuous Integration:

- A *pipeline* de **CI/CD do Github** compila e testa o código em cada *pull request* para as branches *develop* ou *main*, de forma a garantir que as novas alterações não quebram funcionalidades existentes. Quaisquer erros ou conflitos devem ser resolvidos por quem inicia o pull request.
- Durante cada integração são executados testes automatizados incluindo testes unitários, testes de integração (se for o caso) e análise de qualidade do código.

o. Issues/User Stories:

- A US tem um número único no repositório/projeto sobre o qual é criada e um nome descritivo. Quando é criada, em primeiro lugar deve ser escolhido o repositório/projeto ao qual a *issue* se refere. Posteriormente, a *User Story* é detalhada, estimada e priorizada. Nesta fase a issue fica na board "**Backlog**".
- Deve ser descrita da seguinte forma "Como [utilizador]...Quero [objetivo/função]...Para [valor/benefício esperado]
- A US deve ter critérios de aceitação que definem quando a mesma está concluída
- Apenas quando a *User Story* avança para a fase **"Ready"** é que está realmente pronta para ser atribuída à **iteração atual** (sprint) e ao **Developer**.
- **É da responsabilidade do Developer** a quem foi atribuída a tarefa de a fazer transitar para as *boards* seguintes. Este processo facilita o planeamento semanal nas reuniões de equipa.
- De seguida, a *User Story* passa para "**In Progress**", o que significa que está a ser trabalhada.
- Quando todas as tarefas forem concluídas, a *Issue* transita para a board de **"Done"**, o que corresponde a uma Issue que está associada a um Pull Request bem sucedido do sprint decorrente.
- No início de cada sprint as Issues do sprint anterior passam para a board de "**Archive**", onde são agrupadas todas as **Issues** anteriores de sprints passados.

p. Epics:

- Uma Epic tem um número único e um título descritivo que representa uma funcionalidade ou objetivo de alto nível. Cada issue que representa uma Epic deve ter obrigatoriamente a tag "Epic".
- As **Epics** estão numa board específica (Epics), que por defeito não aparecem na **Development Board** da organização. Do lado esquerdo da Development Board é possível ter uma planificação visual da percentagem de conclusão de uma determinada Epic.

- O estado de uma Epic é automaticamente calculado com base no progresso das User Stories associadas (ex: 3/5 User Stories concluídas = 60%)
- Cada Epic é uma *parent issue*, e todas as *issues* (*user stories*) associadas a esse Epic são *sub-issues* dessa *parent issue*.
- As Epics devem ter critérios de aceitação ao nível macro que definem quando a funcionalidade completa está pronta para produção.
- O ciclo de vida das Epics na Development Board é de: Backlog -> Epics -> Done -> Archive
- No planeamento de sprint, as User Stories são selecionadas tendo em consideração o progresso e prioridade das Epics parentais

q. Priorização das US/Issues:

- **Burndown charts**
- A priorização é definida durante as reuniões de planeamento de sprint. Mediante a pertinência da funcionalidade, ou da importância para o avanço do projeto, é atribuída um grau de importância à issue que simboliza a prioridade com que esta tem que ser tratada.
- As de prioridade **critical** são as que têm que ser resolvidas com o máximo de urgência, pois são problemas que afetam ou que bloqueiam o desenvolvimento e/ou progresso de outras Issues.
- Prioridade 1 – Low (baixa prioridade)
- Prioridade 2 – Medium (prioridade média)
- Prioridade 3 – High (alta prioridade)
- Prioridade 4 – Critical (crítica/bloqueante)

r. Estimativa das US:

- São acordadas em cada reunião de sprint. Cada membro da equipa vota (XS, S, M, L) e revelam as estimativas ao mesmo tempo. Discutem discrepâncias na votação e fazem uma nova votação até haver um consenso. Os critérios de estimativa podem ser de complexidade técnica, quantidade de testes a produzir, documentação requerida, tipo de dependências externas ou tecnologias a adotar (especialmente no caso de ser uma tecnologia que ainda não foi devidamente explorada).
- XS: (cerca de 1 hora)
- S: (até 1 dia)
- M: (até 3 dias)
- L: (até 1 semana)

s. Definição de tarefas de cada US/Issue:

- Cada tarefa deve ser **pequena** (idealmente < 1 dia de trabalho). Salvo exceções que o justifiquem.
- Deve ter um propósito claro e critérios de conclusão bem definidos (ver secção **o**).

t. Atribuição de US/Issues:

Após a atribuição, deve ser registado na respetiva Issue: - *Assignee* (membro responsável) - Prazo de conclusão esperado - Comentário com contexto adicional, se necessário. **Mudanças de atribuição** só podem ser feitas mediante discussão em reunião ou aprovação do *Team Leader*, sendo sempre documentadas na Issue correspondente.

u. Semantic Versioning

Cada repositório do github tem configurado um sistema automático de *semantic versioning*. É automatizado através de um github workflow (.github/workflows/release.yml), que gere este sistema automaticamente com base nos commits realizados, daí a importância do uso do sistema de **conventional commits**.

O sistema adotado é um dos padrões mais utilizados na indústria de incremento de versões de software: **MAJOR.MINOR.PATCH**. (https://semver.org/)

- MAJOR são alterações que mudam/quebram a compatibilidade do código (feat!:, fix!:, etc);

- **MINOR** representam novas funcionalidades compatíveis (ex.: feat:);

- **PATCH** está associado a correções e atualizações menores (ex.: fix:, docs:, chore:);

Ex.:

MINOR increment: 2.**1**.4 → 2.**2**.0, neste caso faz-se um reset ao número correspondente ao PATCH, que fica a 0.

MAJOR increment: 2.1.4 → **3**.0.0, neste caso faz-se um reste aos números correspondentes ao de **PATCH** e de **MINOR**, ambos ficam a 0.

PATCH increment: 1.2.**3** → 1.2.**4**, apenas é feito o incremento no último número, por exemplo um *bugfix* ou uma atualização de documentação.

1.4. Regras de avaliação interna

A equipa realizará uma reunião oficial no final de cada *milestone* dedicada à avaliação interna. Nesta reunião, cada elemento será avaliado pelos restantes membros do grupo, sem possibilidade de autoavaliação, sendo o processo registado em ata e incluído no Plano de Projeto. A avaliação terá em conta os critérios enunciados na Figura 1.1:

Figura 1.1 – Critério para avaliação interna, e o respetivo peso.

Critério	Descrição	Peso (%)	Escala (1–5)
Assiduidade e Pontualidade	Presença nas reuniões e cumprimento dos horários definidos	10 %	1 = Faltas frequentes / atrasos, 5 = Sempre presente e pontual
Participação Ativa	Contributos em discussões, propostas de soluções, envolvimento nas tarefas	20 %	1 = Passivo, 5 = Participa de forma consistente e relevante
Cumprimento de Prazos	Respeito pelas datas-limite acordadas para tarefas e entregas	15 %	1 = Nunca cumpre, 5 = Cumpre sempre sem atrasos
Qualidade do Trabalho	Clareza, rigor técnico e alinhamento com os requisitos	25 %	1 = Trabalho incompleto/deficiente, 5 = Trabalho de elevada qualidade
Quantidade de Trabalho	*Story points*/tarefas concluídas e aceites pelo grupo	15 %	1 = Pouco contributo, 5 = Contributo elevado e consistente
Colaboração e Trabalho em Equipa	Cooperação, entreajuda, comunicação e ambiente positivo no grupo	15 %	1 = Individualista / não colabora, 5 = Forte espírito de equipa

1.5. Modelos

Modelo de cronograma

O cronograma é planeado através do microsoft planner, onde os membros da equipa estão incluídos, que são onde estão definidas as tarefas com as respetivas datas e durações. De seguida é exportado um ficheiro excel, que é posteriormente convertido para o formato "*.csv*".

Por fim, é usada uma ferramenta para gerar um gráfico de Gantt (https://www.onlinegantt.com/#/gantt), com o objetivo de obter-se uma configuração visual do cronograma.

Modelo A (Modelo de convocatórias)

O modelo de atas pode ser consultado no ficheiro Concovatória_template.pdf

Modelo B (Modelo de atas)

O modelo de atas pode ser consultado no ficheiro Ata_template.pdf

2. Registos de Execução

Todos os artefactos homologados e assinados encontram-se no repositório "**project-artifacts**" da organização SeePaw.