PLANO DE TRABALHO - ORQUIDEA

HISTÓRICO DE REVISÕES

Data	Versão	Descrição	Autor(es)
11/04/2025	0.1	Criação do documento de Plano de Projeto	Todos os participantes.
24/04/2025	0.2	Revisão e aprimoramento da versão anterior com inclusão dos tópicos acerca do gerenciamento de riscos e manutenção do software.	Todos os participantes.
07/05/2025	0.3	Melhora da documentação, incluindo um cronograma mais bem elaborado e melhorando a organização do plano de trabalho.	Todos os participantes.
22/05/2025	0.4	Reavaliação e mudança do cronograma, ajustando o incremento 4.	Todos os participantes.

1. IDENTIFICAÇÃO

A. **Nome do Projeto:** Orquidea
B. **Equipe:** Lattes Mas Não Morde
C. **Data de criação do documento:** 11/04/2025
D. **Membros:**
   a. **Felipe de Castro Azambuja (nº USP: 14675437)** – Estudante de Bacharelado em Ciência da Computação
   b. **João Pedro Viguini T.T. Correa (nº USP: 14675503)** – Estudante de Bacharelado em Ciência da Computação
   c. **Matheus Paiva Angarola (nº USP: 12560982)** – Estudante de Bacharelado em Ciência da Computação
   Pietra Gullo Salgado Chaves (nº USP: 14603822) – Estudante de Bacharelado em Ciência da Computação

2. INTRODUÇÃO:

Descrição dos objetivos do documento, público-alvo e o propósito do projeto a ser desenvolvido. Deve informar como o plano de projeto irá evoluir e qual será o método de desenvolvimento utilizado.

Este documento apresenta o plano de trabalho para o desenvolvimento de um sistema de gerenciamento de publicações científicas. O principal objetivo é oferecer funcionalidades aprimoradas e uma interface mais intuitiva e informativa em comparação aos sistemas atualmente disponíveis, como o ORCID e a plataforma Currículo Lattes.

Dado que nosso projeto é de pequeno porte, com tempo limitado para execução e uma equipe reduzida, de quatro integrantes, optamos por utilizar o modelo incremental de desenvolvimento. Essa abordagem se mostra adequada por ser leve, flexível e prática, sem exigir uma estrutura rígida de papéis ou processos.

O modelo incremental nos permite dividir o sistema em partes menores (incrementos), que serão desenvolvidas de forma iterativa e funcional. A cada incremento, há a entrega de uma nova funcionalidade que pode ser testada e validada individualmente, o que facilita o acompanhamento do progresso e a identificação precoce de ajustes necessários. Isso é particularmente útil em um contexto acadêmico e de curta duração, no qual mudanças de escopo ou prioridades podem ocorrer durante o desenvolvimento.

Além disso, o modelo favorece a colaboração entre todos os membros da equipe, uma vez que não impõe papéis fixos como em outros modelos (ex: Scrum). Cada integrante poderá propor, implementar e evoluir funcionalidades específicas, de forma coordenada, promovendo uma divisão de tarefas natural e equilibrada.

Em resumo, o uso do modelo incremental nos permitirá manter a organização do projeto, garantir entregas contínuas, adaptar o planejamento e aproveitar ao máximo o tempo e recursos disponíveis.

3. ESCOPO DO PROJETO: Descrição em linhas gerais do projeto a ser desenvolvido. Deve deixar claro sua importância para os stakeholders. Incluir uma descrição dos requisitos técnicos, funcionais e não funcionais, além das funcionalidades que não fazem parte do escopo do projeto.

O projeto visa integrar dados acadêmicos provenientes de múltiplas fontes, centralizando perfis, citações e métricas de impacto dos pesquisadores. A plataforma possibilitará buscas unificadas e fornecimento de análises do desempenho científico de cada pesquisador.

Para os Stakeholders, é fundamental evidenciarmos o quanto esse projeto facilitará o acesso de dados consolidados e confiáveis, além de um apoio na avaliação e monitoramento do impacto das pesquisas, bem como uma melhoria na gestão de currículos e publicações acadêmicas.

Ao se tratar dos requisitos funcionais, o sistema deverá integrar dados acadêmicos a partir de APIs. Nesse sentido, a extração de informações dos perfis é simplificada. Já os requisitos não funcionais incluirão considerações sobre desempenho, usabilidade e segurança, enquanto funcionalidades mais avançadas poderão ser trabalhadas em futuras versões e, desta forma, ficarão fora do escopo inicial.

4. EQUIPE E INFRAESTRUTURA: Descrição da equipe, papéis e responsabilidades, e da infraestrutura, que inclui: ferramentas, materiais, softwares de apoio, equipamentos, etc.

A equipe é composta por quatro integrantes com competências complementares nas áreas envolvidas no projeto, incluindo desenvolvimento web, design de interfaces, uso de APIs, organização de tarefas e uso de ferramentas de

colaboração. Cada membro assumirá responsabilidades específicas, como desenvolvimento front-end, gerenciamento de tarefas, testes e integração de funcionalidades, garantindo uma abordagem colaborativa e eficiente.

Para a organização e acompanhamento das atividades, poderá ser utilizado o Notion, que permitirá o controle de prazos, atribuição de tarefas e centralização de informações relevantes ao projeto. Para a comunicação de atualizações rápidas/pontuais, poderão ser utilizados aplicativos de mensagem.

O desenvolvimento da interface e das funcionalidades do sistema será realizado principalmente com React, framework moderno e eficiente para criação de interfaces web dinâmicas e responsivas. Além disso, serão utilizados outros recursos de apoio, como bibliotecas de componentes, ferramentas de versionamento de código (GitHub).

Para a divisão de tarefas, dois membros serão responsáveis pelo desenvolvimento da interface e dois membros serão responsáveis pela integração das funcionalidades.

A infraestrutura também contará com computadores pessoais dos integrantes, acesso à internet estável e demais softwares de suporte ao desenvolvimento, garantindo as condições técnicas necessárias para o bom andamento do projeto. Como o projeto não necessita de recursos computacionais extensos, o uso de CPUs será suficiente para o desenvolvimento em sua totalidade, não havendo a necessidade de GPUs.

5. ACOMPANHAMENTO DO PROJETO: Descrição dos momentos em que as atividades (reuniões, desenvolvimento assíncrono, etc) serão realizadas e de quem irá realizá-las. Explicar como o projeto será acompanhado pela equipe e pelo cliente.

As atividades do projeto serão organizadas por meio de encontros regulares e desenvolvimento assíncrono. A equipe não realizará reuniões formais com frequência. A maior parte da comunicação será realizada por ferramentas de mensagem ou interação presencial de maneira informal. Reuniões presenciais ocorrerão apenas quando forem estritamente necessárias, como em marcos importantes do projeto ou atividades práticas que exijam interação direta – a fim de tornar o desenvolvimento mais eficiente, sem gastos excessivos de tempo com reuniões ineficientes.

O desenvolvimento assíncrono poderá ser acompanhado por ferramentas de gestão de projetos, como Notion por exemplo, garantindo visibilidade e organização das atividades em andamento. Cada membro da equipe será responsável por atualizar o status de suas tarefas, permitindo o acompanhamento contínuo do progresso.

O cliente será envolvido por meio de reuniões quinzenais (preferencialmente presenciais), podendo também serem realizadas remotamente, nas quais serão

apresentados os avanços, discutidos feedbacks e ajustadas as prioridades conforme necessário. Esse formato garante uma comunicação fluida entre equipe e cliente, promovendo alinhamento constante ao longo do projeto.

6. CRONOGRAMA E MARCOS DO PROJETO: Descrever o cronograma do projeto, com as principais atividades e etapas relevantes. Incluir marcos importantes do projeto (milestones), bem como quais serão os artefatos entregues nesses marcos (exemplo: MVP).

Incremento 1 – Interface da Página Inicial [Entrega: 21 de maio de 2025]

Desenvolvimento da interface visual da página inicial do sistema, com foco em usabilidade, identidade visual e responsividade.

- Artefato entregue: Protótipo funcional da home page.

Incremento 2 – Página de Perfil do Pesquisador [Entrega: 28 de maio de 2025]

Implementação da interface visual da página de perfil do pesquisador, incluindo campos e seções destinadas às informações bibliográficas e acadêmicas.

- Artefato entregue: Página de perfil com layout finalizado.

Marco atingido: protótipo completo da interface visual.

Incremento 3 – Integração com ORCID API [Entrega: 4 de junho de 2025]

Implementação do processo de autenticação de usuários por meio da ORCID API e importação automatizada do perfil e dados bibliográficos (como últimas publicações, palavras-chave das áreas de atuação do pesquisador, histórico acadêmico, etc).

- Artefato entregue: Autenticação funcional e exibição dos dados do perfil ORCID..

Incremento 4 – Métricas e avaliação [Entrega: 11 de junho de 2025]

Adição de funcionalidades para busca de outros perfis, comparação entre pesquisadores, estabelecendo as métricas necessárias.

- Artefato entregue: Ferramentas de busca, comparação de perfis, visualização aprimorada e ranking por área.

Incremento 5 – Sistema de ranqueamento [Entrega: 18 de junho de 2025]

Consiste em um sistema de ranqueamento que organiza pesquisadores por áreas de atuação, destacando as áreas mais ativas e os principais pesquisadores em cada uma delas. O sistema facilita a comparação entre pesquisadores e a visualização de tendências na pesquisa, apresentando os dados de forma clara e acessível para apoiar a análise do cenário acadêmico.

- Artefato entregue: Sistema de ranqueamento funcional com visualização dos pesquisadores e áreas mais relevantes.

A tabela abaixo demonstra exatamente o cronograma:

Incremento	Entrega	Descrição
1	21/05/2025	Página inicial (protótipo funcional)
2	28/05/2025	Página de perfil do pesquisador
3	04/06/2025	Integração com ORCID API
4	11/06/2025	Métricas e avaliação
5	18/06/2025	Sistema de ranqueamento
6	22/06/2025	Entrega do projeto finalizado

7. GERÊNCIA DE RISCOS: Identificar e monitorar os riscos do projeto.

Abaixo estão descritos os principais riscos identificados, junto com sua avaliação e os planos de resposta elaborados para cada um.

Escala de probabilidade: 1 (muito baixa) a 5 (muito alta)
Escala de impacto: 1 (baixo impacto) a 5 (impacto crítico)

Atrasos nas APIs externas

- **Probabilidade:** 3
- **Impacto:** 4
- **Plano de resposta:**
    A. Estudar e entender detalhadamente a documentação das APIs com antecedência, para prever limitações ou requisitos complexos.

B. Implementar "mocks" e simulações locais dos dados da API para permitir o avanço do desenvolvimento independentemente da disponibilidade real.
C. Definir prazos de contingência no cronograma para eventuais atrasos externos.
D. Caso a API apresente instabilidades prolongadas, planejar rotas alternativas, como obtenção manual ou integração com fontes secundárias.

Erro na obtenção de determinados dados via API

- **Probabilidade:** 4
- **Impacto:** 5
- **Plano de resposta:**
    A. Revisar toda a documentação da API ORCID e mapear antecipadamente campos obrigatórios, opcionais e potenciais erros de retorno.
    B. Implementar validações automáticas para identificar dados incompletos ou inconsistentes logo após a chamada à API.
    C. Definir fluxos de *fallback*, como solicitações manuais de preenchimento de dados faltantes diretamente pelos usuários.
    D. Monitorar ativamente as mudanças nas versões da API durante o projeto, adaptando o sistema quando necessário.

Falta de domínio em tecnologias novas

- **Probabilidade:** 4
- **Impacto:** 3
- **Plano de resposta:**
    A. Organizar sessões semanais de estudo interno focadas nas principais tecnologias usadas (React, APIs REST, integrações OAuth).
    B. Criar uma base de conhecimento compartilhada com tutoriais, boas práticas e exemplos práticos.
    C. Estimular a divisão de tarefas para que membros mais experientes auxiliem outros em dificuldades específicas.
    D. Priorizar a implementação de funcionalidades mais críticas primeiro, garantindo tempo extra para amadurecimento técnico antes das etapas mais complexas.

Conflitos de merge no Git

- **Probabilidade:** 2
- **Impacto:** 2
- **Plano de resposta:**
    A. Definir claramente a responsabilidade de cada integrante sobre áreas específicas do código, evitando sobreposição de alterações.

B. Criar uma branch separada para cada nova funcionalidade ou correção, seguindo uma nomenclatura padronizada.
C. Exigir revisões obrigatórias de código por pelo menos um membro da equipe antes de qualquer merge.
D. Integrar as alterações frequentemente na branch de desenvolvimento (develop) para evitar divergências acumuladas.
E. Utilizar ferramentas de comparação de branches (Pull Requests) para identificar conflitos antecipadamente.

Sobrecarga de tarefas e prazos não realistas

- **Probabilidade:** 3
- **Impacto:** 3
- **Plano de resposta:**
  A. Fazer revisões periódicas da carga de trabalho de cada membro.
  B. Redistribuir tarefas em caso de sobrecarga identificada, priorizando a conclusão das entregas mais críticas.
  C. Incorporar margens de segurança no cronograma para acomodar imprevistos.

Problemas de compatibilidade entre navegadores

- **Probabilidade:** 2
- **Impacto:** 2
- **Plano de resposta:**
  A. Definir uma lista mínima de navegadores suportados no início do projeto (ex.: Chrome, Firefox, Edge).
  B. Testar a interface em múltiplos navegadores desde os primeiros incrementos.
  C. Corrigir inconsistências utilizando boas práticas de desenvolvimento responsivo e bibliotecas compatíveis

8. TESTES DO PRODUTO: Descrever como serão realizados os testes do produto, incluindo os tipos de teste, estratégias e requisitos de aceitação.

A validação do sistema será conduzida em múltiplos níveis, com o objetivo de garantir a estabilidade, a funcionalidade correta e a boa experiência do usuário final. Para isso, serão realizados diferentes tipos de teste ao longo do desenvolvimento:

Testes unitários: Cada funcionalidade será validada de forma isolada, com foco em componentes independentes — como chamadas a APIs —, assegurando que retornem os resultados esperados com diferentes entradas.

Testes de interface: Será verificada a responsividade da aplicação, avaliando o comportamento da interface em dispositivos com diferentes tamanhos de tela, a fim de garantir uma boa experiência de uso.

Testes de usabilidade: Serão realizados testes informais com usuários da equipe para avaliar a clareza da interface e a intuição dos fluxos, fornecendo feedback qualitativo sobre a experiência de navegação.

Testes de regressão: Sempre que novas funcionalidades forem adicionadas ou alterações forem feitas, testaremos se os recursos existentes continuam funcionando corretamente, evitando que mudanças comprometam partes já implementadas.

Testes de fluxo completo: Serão executados testes ponta-a-ponta para validar processos completos, como autenticação, recuperação e exibição de dados, identificando eventuais falhas de integração entre as partes do sistema.

Testes de desempenho: Ainda que o projeto não exija alta carga, será verificado se o tempo de resposta das principais funcionalidades permanece adequado mesmo com grande volume de dados.

Procedimentos adotados:

A. Antes da implementação de uma funcionalidade, será definido um teste simples que descreva o comportamento esperado.
B. A funcionalidade será considerada pronta apenas quando passar nos testes definidos.
C. Todos os testes serão executados localmente antes do envio da funcionalidade para revisão.
D. O foco de cobertura será maior nas partes mais críticas do sistema, como autenticação, integração de APIs, tratamentos dos dados e suas relações.
E. Sempre que possível, erros identificados em testes serão transformados em casos de teste adicionais, prevenindo regressões futuras.
F. Os testes serão organizados e documentados para facilitar a reexecução durante as entregas parciais (incrementos) do projeto.

9. GERENCIAMENTO DE CONFIGURAÇÃO DE SOFTWARE: Descrição da padronização da nomenclatura dos itens de configuração, mecanismos de versionamento (GitHub, Git flow) e dependências entre os itens.

Usamos o GitHub para armazenar todo o código do projeto e mantemos duas *branches* principais:

A. *main*: versão pronta para uso;
B. *develop*: onde são integradas novas funcionalidades antes de aplicá-las na *main*;

Cada nova funcionalidade ou correção ganha uma branch própria no formato de “feature/<nome-da-feature>”.

Nas mensagens de commit seguiremos o padrão:

A. *novo*: descrição da funcionalidade;
B. *corrige*: descrição do ajuste;

Isso corresponde a algo como por exemplo:

novo: autenticação via ORCID

Antes de juntar qualquer *feature* na *develop*, revisamos o código:

A. Um membro verifica se o código passa nos testes sem erro;
B. Testamos manualmente a funcionalidade para garantir que está funcionando;

Dessa forma conseguimos manter o histórico organizado e evitamos conflitos ao integrar diferentes partes do sistema.

10. PLANO DE MANUTENÇÃO DE SOFTWARE: Descrição do processo de manutenção e evolução do software. Explicar graficamente as etapas e os papéis relacionados

1. Registro de Solicitações

Solicitações de correção, melhorias ou ajustes de documentação devem ser registradas como *issues* na plataforma GitHub, utilizando classificações apropriadas (ex.: *bug*, *enhancement*, *docs*).

Responsável: todos os membros da equipe são autorizados a registrar demandas identificadas durante o ciclo de desenvolvimento.

2. Triagem e Priorização
A cada duas semanas será realizada uma reunião de triagem para revisar as *issues* abertas, avaliar sua relevância e atribuir prioridades de 1 (baixa) a 5 (alta), com base em critérios de impacto e urgência.

Responsável: coordenação da equipe, com apoio de todos os integrantes na definição consensual de prioridades.

3. Planejamento de Correções ou Melhorias
As *issues* priorizadas serão selecionadas para compor o próximo ciclo de desenvolvimento. Cada item será associado a uma nova *branch*, nomeada conforme o padrão feature/<descrição>.

Responsável: membro designado pela coordenação técnica, com base na alocação de tarefas e cronograma vigente.

4. Implementação e Testes
As modificações serão implementadas na *branch* atribuída, seguidas da execução de testes automatizados e manuais, de acordo com o escopo da tarefa. O código deverá passar por revisão técnica antes de ser integrado.

Responsável: desenvolvedor atribuído à tarefa e revisor técnico previamente definido.

5. Integração e Publicação
Após validação da revisão, as alterações serão integradas à *branch* develop e, posteriormente, à *main`, conforme os critérios de estabilidade e versionamento. O *deploy* será realizado nos ambientes de teste e produção conforme aplicável.

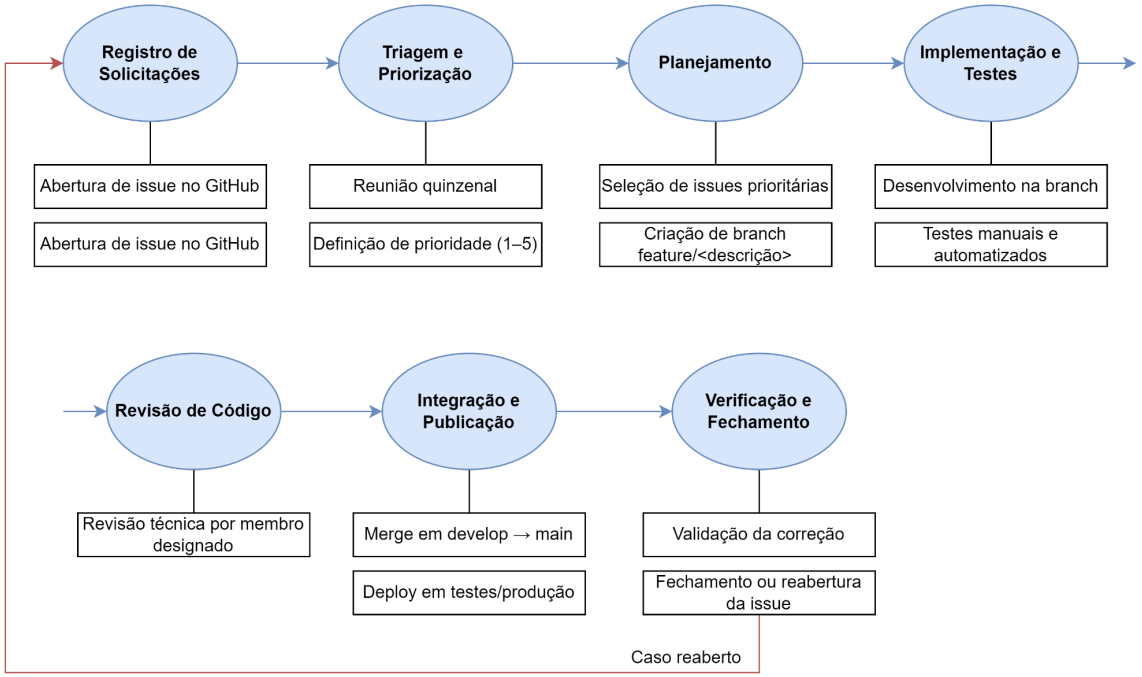
Responsável: responsável técnico pela integração, conforme cronograma e revezamento interno previamente estabelecido.

6. Verificação Pós-Deploy e Fechamento
Será verificado o correto funcionamento da funcionalidade ou correção após publicação. Em caso de sucesso, a *issue* será encerrada. Em caso de falhas ou efeitos colaterais, a *issue* será reaberta e reavaliada.

Responsável: responsável pela tarefa original em conjunto com o revisor técnico.

Fluxo Operacional:

Registro → Triagem → Planejamento → Implementação → Testes → Revisão → Integração → Verificação



Link:
https://drive.google.com/file/d/1keWhXiSbQEkxh30jMhkIQYxGo83XDgWb/view?usp=sharing

Referências

SOMMERVILLE, Ian. *Engenharia de software*. 10. ed. São Paulo: Pearson, 2019.

SSC0130 - Engenharia de Software. Universidade de São Paulo, 2025. Docente: Seiji Isotani.