Nome da aplicação: AchaPro || Integrantes: Filipe B; Lucas M; Joao C; Cleiton V; Daniela L.

1. Definição de Papéis (TSP - Team Software Process)

Papel Principal	Papel Secundário (TSP)	Membro da Equipe	Responsabilidades Principais
Product Owner (PO)	Líder de Equipe	**Filipe**	- Definir a visão e a estratégia do produto. - Gerar todos os documentos e entregáveis necessários. - Gerenciar e priorizar o Backlog do Produto (a lista de funcionalidades). - Ser a "voz do cliente", garantindo que o time construa o produto certo. – Garantir que tudo está sendo construído conforme acordado e definido previamente. - Aceitar as funcionalidades concluídas ao final de cada sprint. - Facilitar a comunicação entre a equipe e stakeholders (neste caso, o professor).
Desenvolvedor Sênior	Gerente de Desenvolvimento	**Lucas**	- **Codificar** as funcionalidades mais complexas do aplicativo. - Liderar a definição da arquitetura do software e tomar decisões técnicas. - Supervisionar e mentorar o outro desenvolvedor. - Garantir a qualidade do código através de revisões (Code Reviews). - Responsável pelo **Documento de Arquitetura**.
Desenvolvedor Pleno	Analista de sistemas	**Daniela**	- **Codificar** as funcionalidades do aplicativo em colaboração com a equipe. - Desenvolver testes unitários para o código que escreve. - Participar ativamente das revisões de código para aprender e contribuir. - Ajudar a corrigir bugs identificados pela equipe de QA.
Desenvolvedor Pleno	Analista de sistemas	**João**	- **Codificar** as funcionalidades do aplicativo em colaboração com a equipe. - Desenvolver testes unitários para o código que escreve. - Participar ativamente das revisões de código para aprender e contribuir. - Ajudar a corrigir bugs identificados pela equipe de QA.
Analista de Qualidade (QA)	Gerente de testes	**Cleiton**	- **Planejar, criar e executar** toda a estratégia de testes. - Responsável por todos os artefatos de teste: **Plano de Testes, Casos de Teste, Relatório de Execução e Relatório de Defeitos**. - Trabalhar em estreita colaboração com os desenvolvedores para identificar e documentar bugs. - Garantir que os requisitos funcionais e não funcionais sejam atendidos antes da entrega.

2. Planejamento da Missão e Metas

Missão do Projeto:

Desenvolver um protótipo funcional (MVP) de um aplicativo de marketplace de serviços, aplicando as melhores práticas de engenharia de software, e entregar todos os artefatos de processo (documentação, código e relatórios de teste) que demonstrem a qualidade e a organização do trabalho da equipe dentro do prazo final de término da matéria.

a. Metas SMART e GQM

Meta 1 (G): Assegurar a Qualidade e a Rastreabilidade do Processo de Desenvolvimento.

- **Pergunta 1.1 (Q):** A equipe está seguindo o processo de desenvolvimento definido (Scrum com revisões)?
    - **Métrica 1.1.1 (M): 100% das novas funcionalidades** (Pull Requests) devem ser submetidas a uma revisão de código (Code Review) e aprovadas por pelo menos um outro membro da equipe antes de serem integradas ao código principal.
    - **Métrica 1.1.2 (M):** Realizar **100% das cerimônias do Scrum** (Planning, Daily, Review, Retrospective) em todos os sprints do projeto, com atas ou registros para comprovar.
- **Pergunta 1.2 (Q):** Os requisitos definidos estão sendo corretamente implementados e testados?
    - **Métrica 1.2.1 (M): 100% dos requisitos funcionais** definidos para o MVP devem ter casos de teste correspondentes no Plano de Testes.
    - **Métrica 1.2.2 (M):** Atingir uma cobertura de **pelo menos 95% de aprovação** nos casos de teste planejados para a entrega final do MVP.

Meta 2 (G): Entregar um Produto de Software de Alta Qualidade Técnica.

- **Pergunta 2.1 (Q):** O código-fonte do projeto é manutenível e segue os padrões definidos?
    - **Métrica 2.1.1 (M):** O código final do projeto não deve conter **nenhum erro crítico** ou "code smell" grave apontado por uma ferramenta de análise de código estático (como SonarLint ou similar).
    - **Métrica 2.1.2 (M): 100% do código** deve estar em conformidade com o guia de estilo e formatação definido pela equipe.
- **Pergunta 2.2 (Q):** O protótipo funcional atende aos requisitos não funcionais essenciais?
    - **Métrica 2.2.1 (M):** Garantir que o tempo de resposta para **90% das interações críticas** do usuário (login, carregar lista de serviços) seja **inferior a 2 segundos** durante a demonstração final.

Meta 3 (G): Concluir o Projeto e Todos os Seus Entregáveis Dentro do Prazo.

- **Pergunta 3.1 (Q):** O escopo do MVP está sendo gerenciado de forma eficaz para caber no cronograma?

        - **Métrica 3.1.1 (M):** Concluir **100% das funcionalidades priorizadas** para o escopo do MVP até a data de entrega final do projeto.
- **Pergunta 3.2 (Q):** Todos os artefatos de documentação exigidos pelo trabalho acadêmico serão entregues?
    - **Métrica 3.2.1 (M):** Entregar **100% dos documentos obrigatórios** (Documento de Arquitetura, Plano de Testes, Relatórios de Execução, etc.) com a qualidade definida pela equipe e pelo professor até a data final.

b. Levantamento Detalhado de Requisitos (Incluindo NFRs)

Requisitos Funcionais (O que o sistema faz):

1. **RF01:** O sistema deve permitir que usuários se cadastrem como "Cliente" ou "Prestador".
2. **RF02:** O Prestador deve poder criar e editar seu perfil, incluindo descrição e fotos.
3. **RF03:** O Cliente deve poder postar uma nova tarefa, com descrição, categoria e fotos.
4. **RF04:** O Prestador deve poder visualizar uma lista de tarefas disponíveis.
5. **RF05:** O Prestador deve poder enviar uma proposta para uma tarefa.
6. **RF06:** O Cliente deve poder aceitar ou recusar uma proposta.
7. **RF07:** Um chat deve ser disponibilizado entre Cliente e Prestador após o aceite da proposta.
8. **RF08:** O Cliente deve poder marcar um serviço como "Concluído".
9. **RF09:** Após a conclusão, o Cliente deve poder avaliar o Prestador com uma nota (1-5) e um comentário.

Requisitos Não Funcionais (NFRs - Como o sistema se comporta):

- **Desempenho:**
    - **NFR01:** O tempo de carregamento de qualquer tela principal do aplicativo não deve exceder 2 segundos.
    - **NFR02:** A API do backend deve responder a 99% das requisições em menos de 500ms.
- **Usabilidade:**
    - **NFR03:** Um novo usuário deve ser capaz de postar seu primeiro serviço em menos de 3 minutos, sem necessidade de um tutorial.
    - **NFR04:** A interface deve ser intuitiva e seguir as diretrizes de design pré-estabelecidos
- **Segurança:**
    - **NFR05:** Todas as senhas de usuários devem ser armazenadas de forma criptografada (hashed com salt).
    - **NFR06:** Toda a comunicação entre o aplicativo e o servidor deve ser feita via HTTPS.
- **Manutenibilidade:**

- **NFR07:** O código deve ser modular, com clara separação de responsabilidades (ex: UI, lógica de negócio, acesso a dados).
- **NFR08:** O projeto deve conter documentação técnica que explique a arquitetura e as decisões importantes.

c. Processo de Planejamento TSP

Tarefa Principal (Épico)	Estimativa em Story Points	Detalhes / Justificativa
1. Configuração do Ambiente e Arquitetura	8	Tarefa complexa e fundamental, envolve decisões técnicas importantes, mas é um trabalho concentrado.
2. Módulo de Usuários (Cadastro, Login, Perfil)	13	Envolve várias telas, validações de formulário e lógica de backend. Pode ser quebrado em histórias menores (ex: Cadastro - 5 pts, Login - 3 pts, Perfil - 5 pts).
3. Módulo de Serviços (Postar, Listar, Buscar)	21	Este é o coração do app. Envolve upload de imagens, filtros, lógica de negócio complexa. Definitivamente um épico a ser quebrado.
4. Módulo de Propostas e Chat	13	O chat em tempo real tem sua complexidade (WebSockets), e a lógica de aceitar/recusar propostas também.
5. Sistema de Avaliação	8	Envolve a lógica de permitir avaliação apenas após a conclusão, salvar as notas e calcular a média do prestador.
6. Criação e Execução dos Planos de Teste	(Contínuo)	No Scrum, o teste não é uma fase, mas uma atividade contínua. O esforço de teste já está "embutido" na estimativa de cada tarefa.
Total Estimado	~63 Pontos	

3. Processo e Padrões da Equipe

a. Ciclo de Vida Adaptado: Scrum

Adotaremos o Scrum como nosso ciclo de vida, pois ele é ideal para projetos com requisitos que podem evoluir e foca em entregas de valor rápidas.

- **Sprints:** Ciclos de trabalho de **1 semana**. Ao final de cada sprint, a equipe deve entregar uma parte funcional do software.
- **Cerimônias do Scrum:**
    - **Sprint Planning (Planejamento):** No início de cada sprint, a equipe se reúne para definir o que será construído na próxima 1 semana.
    - **Daily Scrum (Reunião Diária):** Uma reunião rápida de 15 minutos todo dia para sincronizar o time (o que fiz ontem, o que farei hoje, há impedimentos?).
    - **Sprint Review (Revisão):** Ao final do sprint, a equipe apresenta o que foi construído para as partes interessadas (neste caso, para o próprio time, agindo como cliente).
    - **Sprint Retrospective (Retrospectiva):** Após a Review, a equipe se reúne para discutir o que deu certo, o que deu errado e como melhorar no próximo sprint.

b. Team Process (Padrões da Equipe)

- **Padrões de Codificação:**
    - **Linguagem e Nomenclatura:** Todo o código será escrito em inglês. Nomes de variáveis devem ser claros e descritivos (ex: userProfile em vez de uP). Nomes de funções devem ser verbos (ex: fetchUserData()).
    - **Formatação:** Será utilizado um formatador de código automático (como o formatador da IDE) para garantir um estilo consistente em todo o projeto.
    - **Comentários:** Comentar apenas o "porquê" de um código complexo, não o "o quê". O código deve ser legível o suficiente para explicar a si mesmo.
- **Revisões de Código (Code Reviews):**
    - **Processo:** Nenhuma funcionalidade nova (Pull Request) pode ser integrada à base de código principal (branch main) sem a aprovação de **pelo menos 1 outro membro da equipe** (além do autor).
    - **Checklist de Revisão:** O revisor deve verificar:
        1. O código funciona e atende aos requisitos?
        2. Existem testes para a nova funcionalidade?
        3. O código segue os padrões de codificação?
        4. O código é legível e fácil de manter?
        5. A solução é segura e performática?
- **Documentação dos Testes:**

- **Plano de Teste:** Para cada funcionalidade principal (ex: "Publicação de Serviço"), será criado um documento simples (no Google Docs ou Confluence) descrevendo os casos de teste.
- **Formato do Caso de Teste:** Cada caso de teste deve ter:
    - **ID:** Um identificador único (ex: TC-01).
    - **Descrição:** O que está sendo testado (ex: "Cliente posta um serviço sem anexar foto").
    - **Passos para Reprodução:** A sequência exata de ações.
    - **Resultado Esperado:** O que deveria acontecer.
    - **Resultado Obtido:** (Preenchido durante a execução).
    - **Status:** (Passou / Falhou).