Universidade Federal de Pernambuco :: Centro de Informática
Sistemas de Informação :: Engenharia de Software
Prof. Vinicius Cardoso Garcia

INSTRUÇÕES: Leia as questões com atenção e cuidado, e responda com atenção ao limite de caracteres, quando indicado.

- Esta avaliação é **individual, sem consulta,** e vale **10 pontos** no total.
- A prova é composta por **10 questões objetivas** (8 pontos) e **1 questão discursiva** (2 pontos).
- A duração total é de **até 2 horas**; organize seu tempo.
- Não é permitido **nada em cima da mesa ou no colo**. Guardem os celulares e demais dispositivos inteligentes, digitais, analógicos e mecânicos no bolso ou na mochila (a qual deve permanecer no chão).
- Responda todas as questões no **formulário de respostas, de lápis ou caneta**.
- Dúvidas podem ser expostas, **publicamente**, durante os **primeiros 30 minutos** da realização do Exercício Escolar. **Entender o enunciado** faz parte da avaliação.
- Não é permitido ir ao sanitário durante a realização do exercício, **vá antes**.

INSTRUÇÕES ESPECÍFICAS PARA AS QUESTÕES OBJETIVAS (FORMATO DE SOMATÓRIO BINÁRIO)

As 10 questões objetivas desta prova utilizam um formato especial, com 6 alternativas por questão, cada uma associada a um valor numérico fixo, conforme abaixo:

- (02) Alternativa 1
- (04) Alternativa 2
- (08) Alternativa 3
- (16) Alternativa 4
- (32) Alternativa 5
- (64) Alternativa 6

Cada questão pode ter **uma ou mais alternativas corretas**.

Você **não deve marcar as alternativas individualmente**.

Sua tarefa é informar apenas o *somatório numérico* das alternativas que considera verdadeiras.

Exemplo ilustrativo:

Se você acredita que as alternativas corretas são 1, 3 e 4, então o valor final será: `02 + 08 + 16 = 26`

Esse é o número que deve ser registrado no formulário de respostas.

CRITÉRIOS IMPORTANTES DE CORREÇÃO

1. Se você incluir alguma alternativa falsa no somatório → **a questão será zerada**. Mesmo que todas as demais estejam corretas.
2. Se você deixar de incluir alguma alternativa verdadeira → **receberá pontuação proporcional**. Exemplo:
   - A questão tem 4 alternativas corretas.
   - Você acerta apenas 3 delas → você recebe **3/4 da pontuação** daquela questão.
3. **O valor final de cada questão objetiva deve ser um único número** (ex.: 26, 58, 110...).

Não escreva alternativas, marcações ou justificativas.

Apenas escreva o **valor numérico final** no campo indicado.

INSTRUÇÕES PARA A QUESTÃO DISCURSIVA

A última questão vale **2 pontos** e deve ser respondida de forma textual, no espaço indicado, com clareza, objetividade e articulação técnica.

Leia atentamente o que está sendo solicitado e responda dentro do espaço disponível.

Boa sorte!

Você integra a equipe da **Glink Inc.**, uma startup em rápido crescimento que desenvolve o aplicativo **Glink**, voltado à conexão de pessoas do universo geek. Após o sucesso do MVP, a equipe inicia um ciclo de melhorias e expansão de funcionalidades. O time adota Scrum, utiliza Git Flow como estratégia de versionamento e mantém uma arquitetura de microsserviços para permitir evolução independente dos módulos.

O Product Owner (PO) solicitou que o time priorize, para o próximo sprint, três histórias relacionadas à funcionalidade de "Índice de Compatibilidade Geek", que está gerando alto volume de feedbacks por lentidão na tela de descoberta e inconsistências em alguns cálculos.

Após o lançamento da primeira versão pública do **Glink**, um pico inesperado de novos usuários ocorreu logo após o app viralizar no TikTok. Em poucas horas:

- O **serviço de cálculo do Índice de Compatibilidade (Compatibility Service)** começou a responder com **alta latência** (até 1,8 segundos).
- O time de mobile reportou que o endpoint `/compatibility/calc` começou a retornar erros **500** intermitentes.
- O PO está preocupado com reclamações e com a nota do app caindo na Play Store.
- A equipe de backend percebe que:
    - Existe **baixo nível de testes automatizados**, especialmente testes de integração.
    - O serviço roda em um microsserviço isolado, mas está fazendo **consultas repetidas** ao banco de dados central de perfis.
    - O pipeline de CI/CD não rodava testes de carga e não bloqueava builds instáveis.
    - A pessoa de DevOps identificou que o autoscaling não subiu novas instâncias porque o script de provisionamento falhou após uma alteração recente no pipeline.
    - A startup precisa corrigir o problema para o próximo release, garantindo **qualidade contínua, segurança, testes automatizados e estabilidade em produção**.

Questão 01– Fundamentos de Teste (Glink/Compatibilidade)

O endpoint `/compatibility/calc` do Glink apresentou erros 500 e alta latência sob pico de uso. Avalie as afirmações abaixo sobre tipos de teste e estratégias de cobertura para explicar por que o problema não foi detectado antes do deploy.

(02) Testes de unidade validam funções/métodos isolados; por isso, não capturam problemas de integração com banco, rede ou outros serviços.

(04) Cobertura de linhas alta (>90%) garante, por si só, detecção de erros de contrato entre microsserviços, tornando desnecessários testes de integração.

(08) Testes de integração exercitam a comunicação entre componentes/serviços (ex.: `/compatibility/calc` ↔ DB), podendo revelar erros 500 causados por dependências.

(16) Testes de sistema (end-to-end) podem evidenciar latência percebida pelo usuário, pois percorrem fluxos completos do app.

(32) Testes de carga antes do deploy ajudariam a expor degradação de performance e gargalos sob pico, reduzindo a chance de incidentes como o observado.

(64) Cobertura por mutação mede a robustez dos testes contra mudanças no código; sozinha é suficiente para garantir que problemas de timeout e autoscaling serão pegos em CI.

Resposta: _____

Questão 02 – Testes Avançados e Qualidade Contínua (Pipeline do Glink)

O pipeline de CI/CD do Glink rodava apenas testes de unidade, não incluía testes de integração nem testes de carga, e tampouco aplicava boas práticas de quality gates. Após o incidente de latência e erros 500, a equipe revisou o pipeline. Analise as afirmações abaixo sobre testes avançados, automação e qualidade contínua em pipelines modernos.

Boa sorte!

(02) Testes de contrato entre microsserviços ajudam a detectar quebras de compatibilidade entre produtor e consumidor, sendo fundamentais em arquiteturas distribuídas como a do Glink.

(04) É uma boa prática que o pipeline execute testes E2E antes dos testes de unidade, garantindo que o fluxo principal do sistema esteja funcionando desde o início.

(08) Testes de carga são essenciais para identificar gargalos de desempenho antes do deploy, especialmente em endpoints críticos como `/compatibility/calc`.

(16) Um quality gate pode impedir o merge se métricas como cobertura mínima, vulnerabilidades ou falhas de linting forem detectadas, aumentando a robustez do CI.

(32) Testes automatizados em pipeline devem ser organizados de modo a fornecer feedback rápido, colocando testes baratos (unidade/lint) primeiro e testes mais pesados (integração/carga) depois.

(64) Um pipeline que falha parcialmente, mas mesmo assim permite deploy, é considerado uma implementação madura de testes contínuos, pois privilegia a entrega rápida em detrimento da estabilidade.

Resposta: _____

Questão 03 – DevOps e CI/CD (Orquestração do Deploy no Glink)

Após o incidente no serviço de compatibilidade, descobriu-se que o script de provisionamento falhou devido a uma alteração em um stage do pipeline, impedindo o autoscaling. A equipe decidiu revisar o fluxo de build → test → release → deploy e aplicar boas práticas de DevOps. Analise as afirmações sobre CI/CD, orquestração de deploy e práticas modernas de entrega contínua.

(02) Em pipelines modernos, é boa prática armazenar artefatos de build em um repositório de artefatos imutáveis, garantindo que o mesmo pacote validado nos testes seja o que será de fato implantado em produção.

(04) Estratégias como blue-green e canary releases reduzem o risco de deploy, permitindo troca gradual de tráfego e detecção prévia de erros críticos antes de impactar todos os usuários.

(08) O uso de feature toggles contribui para entregas contínuas, pois permite ativar funcionalidades no ambiente de produção sem necessidade de criar novas versões do backend.

(16) O uso de Git Flow para todas as equipes (backend, mobile, DevOps) sempre acelera ciclos de entrega em startups como o Glink, pois cria previsibilidade e reduz quantidade de merges.

(32) Em um pipeline bem projetado, o deploy deve ocorrer somente se todos os stages anteriores (build, testes, segurança, qualidade) forem concluídos com sucesso.

(64) Uma vez que o deploy foi realizado, se surgir um problema em produção, a única opção segura é reverter manualmente os commits problemáticos, pois rollback automático não é considerado boa prática em DevOps.

Resposta: _____

Questão 04 – DevSecOps (Segurança no Ciclo de Desenvolvimento do Glink)

A equipe do Glink identificou que, apesar das melhorias no CI/CD, ainda havia riscos de segurança: bibliotecas desatualizadas, dependências vulneráveis e possibilidade de vazamento de segredos no repositório. Para mitigar riscos, decidiu adotar práticas de DevSecOps em todo o pipeline. Analise as afirmações abaixo, todas relacionadas à segurança aplicada ao ciclo de desenvolvimento.

(02) SAST (Static Application Security Testing) analisa o código-fonte em busca de vulnerabilidades estruturais, como SQL injection, XSS, falhas de validação e uso inseguro de APIs.

(04) DAST (Dynamic Application Security Testing) depende de acesso ao código-fonte e verifica se os testes de unidade cobrem todos os caminhos vulneráveis do sistema.

(08) Ferramentas de scanning de dependências (ex.: Dependabot, Snyk) ajudam a identificar bibliotecas vulneráveis, algo muito relevante no backend do Glink.

(16) Secrets scanning detecta chaves expostas em commits, evitando que credenciais de produção vazem — um risco comum em equipes pequenas como a do Glink.

Boa sorte!

(32) A criação de um SBOM (Software Bill of Materials) ajuda a rastrear cada componente e dependência usada no sistema, facilitando auditorias e respostas rápidas a incidentes de segurança.

(64) Uma boa prática de DevSecOps é permitir que a aplicação acesse diretamente o banco de dados de produção com permissões totais durante testes automáticos, garantindo completude nas validações.

Resposta: _____

Questão 05 – Arquitetura de Software e C4 Model (Aplicação no Glink)

A equipe do Glink percebeu que a ausência de diagramas atualizados dificultou tanto a identificação dos gargalos quanto a comunicação entre backend, mobile e DevOps. Para melhorar a clareza arquitetural, decidiu adotar o C4 Model para representar sua arquitetura de microsserviços e integrações. Analise as afirmações sobre modelagem arquitetural e uso correto dos níveis do C4 Model.

(02) O Diagrama de Contexto (C4 – Nível 1) mostra o sistema do Glink como uma "caixa preta", destacando atores externos e sistemas com os quais o app interage.

(04) O Nível 2 (Container) descreve blocos executáveis independentes, como o serviço de perfis, o serviço de compatibilidade e o serviço de chat, incluindo tecnologias e responsabilidades.

(08) O Nível 3 (Component) representa como cada container é internamente estruturado, exibindo módulos, controladores, repositórios e fluxos internos.

(16) O C4 Model substitui completamente a UML, pois fornece diagramas mais formais, detalhados e com semântica rigorosa padronizada.

(32) Em arquiteturas de microsserviços, o C4 ajuda a identificar limites de contexto, fluxos de dados e dependências entre serviços — inclusive gargalos como chamadas repetidas ao banco.

(64) O Nível 4 (Code) é usado para representar detalhes de classes e código-fonte, mas sua utilização é opcional e raramente recomendada em grandes sistemas por causa da manutenção complexa.

Resposta: _____

Questão 06 – Arquitetura na Prática, Padrões e Evolução do Sistema (Glink)

À medida que o Glink cresce, novos requisitos e funcionalidades precisam ser adicionados rapidamente, como o Party Finder e a integração com plataformas externas. A equipe precisa manter o sistema flexível, reduzindo acoplamento, prevenindo dívida técnica e facilitando a evolução de cada microsserviço. Analise as afirmações abaixo sobre padrões de design, boas práticas arquiteturais e gestão da evolução de sistemas.

(02) O uso adequado de princípios como SRP (Responsabilidade Única) e baixo acoplamento facilita a evolução dos microsserviços, pois reduz impactos colaterais em funcionalidades não relacionadas.

(04) O padrão Singleton costuma ser recomendado para gerenciar estados de sessão e dados de usuários em aplicações distribuídas, pois simplifica a sincronização entre instâncias.

(08) O padrão Factory Method pode ajudar na criação de objetos relacionados à recomendação do Glink (ex.: diferentes calculadoras de compatibilidade), encapsulando lógica de instanciação.

(16) Refatorações frequentes que reduzem complexidade ciclomática e melhoram legibilidade fazem parte das boas práticas de manutenção evolutiva.

(32) A ausência de versionamento de APIs entre microsserviços é uma fonte comum de dívida técnica e pode causar quebras inesperadas em deploys independentes.

(64) Uma arquitetura com alto acoplamento entre serviços é benéfica para startups no início, pois acelera o desenvolvimento e reduz problemas de manutenção no longo prazo.

Resposta: _____

Questão 07 – Microsserviços, SOA e Resiliência (Glink)

O crescimento do Glink trouxe desafios típicos de arquiteturas distribuídas: aumento de chamadas entre serviços, riscos de timeouts, necessidade de padrões de resiliência e adoção de boas práticas de comunicação entre microsserviços. A

equipe quer evitar falhas em cascata e comportamentos imprevisíveis no fluxo de matching e chat. Analise as afirmações abaixo sobre integração, resiliência e boas práticas em microsserviços.

(02) Em um sistema de microsserviços, chamadas síncronas em cascata podem aumentar a latência total e gerar falhas em cadeia; por isso, padrões como circuit breaker ajudam a isolar problemas.

(04) A abordagem API First reforça que contratos de API sejam definidos, documentados e acordados antes da implementação, reduzindo retrabalho entre backend e mobile.

(08) Em microsserviços, idempotência em operações sensíveis (ex.: criação de match ou envio de mensagens) evita efeitos colaterais quando há retries automáticos.

(16) Em arquiteturas distribuídas, o uso de fila de mensagens (ex.: Kafka, RabbitMQ) pode reduzir acoplamento e ajudar a lidar com picos de carga, mas introduz maior dependência temporal entre serviços.

(32) A estratégia de consistência eventual é comum em sistemas distribuídos, aceitando que atualizações possam levar algum tempo para se propagar, o que é adequado para funcionalidades como "última vez online" ou atualização de interesses no Glink.

(64) A orquestração de microsserviços é sempre superior à coreografia, pois centraliza o controle do fluxo e simplifica o entendimento do sistema em qualquer cenário.

Resposta: _____

Questão 08 – RESTful APIs & Plataformização (Glink)

Para acelerar integrações (app mobile, parceiros e futuro Party Finder), o Glink decidiu fortalecer o desenho das APIs, padronizar documentação com OpenAPI e posicionar um API Gateway à frente dos microsserviços. Analise as afirmações abaixo sobre desenho REST, versionamento, documentação e gateway.

(02) Um API Gateway pode centralizar preocupações transversais como autenticação/autorização, rate limiting, roteamento, observabilidade e caching, reduzindo acoplamento entre clientes e serviços.

(04) O versionamento de API só é correto quando feito por headers (ex.: `Accept: application/vnd.glink.v2+json`); usar URI (ex.: `/v2/`) é anti-padrão e deve ser evitado.

(08) OpenAPI (Swagger) viabiliza API-First: contratos claros, geração de stubs de cliente/servidor, documentação viva e base para testes de contrato.

(16) Em criação de recursos, o servidor deve preferir `201 Created` e enviar o header `Location` com a URL do recurso criado; usar `200 OK` genérico pode empobrecer a semântica.

(32) Paginação deve ser consistente (ex.: limit/offset ou cursor-based), expor metadados (próxima/anterior) e manter GET idempotente; para coleções grandes, cursor tende a reduzir drift.

(64) Em REST, `POST` é idempotente e seguro, enquanto `PUT` não é idempotente, por isso POST deve ser usado para atualizações repetidas sem efeitos colaterais.

Resposta: _____

Questão 09 – DDSD (Desenvolvimento Dirigido a Dados no Glink)

Após o incidente de performance no cálculo de compatibilidade, a equipe do Glink decidiu adotar práticas de Desenvolvimento Dirigido a Dados (DDSD) para orientar decisões técnicas com base em métricas reais, reduzir riscos e melhorar a evolução contínua do produto. Analise as afirmações abaixo sobre os princípios, práticas e benefícios do DDSD.

(02) O DDSD valoriza a observabilidade, incluindo métricas, logs estruturados e distributed tracing, permitindo identificar gargalos como os que ocorreram no endpoint `/compatibility/calc`.

(04) O DDSD incentiva experimentação controlada, como testes A/B e feature flags, para validar hipóteses com usuários reais antes de expandir uma funcionalidade para todos.

(08) No DDSD, é importante captar telemetria de produto, como latência média, taxa de conversão e engajamento, para guiar decisões de priorização e design.

(16) Um princípio do DDSD é que decisões arquiteturais devem ser tomadas exclusivamente com base em métricas de performance, sem considerar feedback do usuário ou fatores de negócio.

Boa sorte!

(32) DDSD favorece pipelines que capturam dados de uso automaticamente, gerando ciclos curtos de feedback para ajustar algoritmos, como o índice de compatibilidade do Glink.

(64) O DDSD desencoraja automação de coleta de dados, pois métricas manuais tendem a ser mais confiáveis em ambientes de produção distribuídos.

Resposta: _____

Questão 10 – Confiabilidade, Escalabilidade e Performance (Glink)

Após o aumento explosivo de usuários, o Glink enfrentou problemas de latência, saturação de banco de dados, falhas no autoscaling e indisponibilidade parcial. A equipe revisou práticas de performance, caching, filas e observabilidade para melhorar a confiabilidade do sistema. Analise as afirmações abaixo sobre confiabilidade, escalabilidade e performance em ambientes distribuídos.

(02) Caching distribuído (ex.: Redis) pode reduzir drasticamente a carga sobre o banco ao armazenar resultados frequentes, como cálculos de compatibilidade, diminuindo latência e número de consultas.

(04) Adoção de fila de mensagens permite aliviar picos de carga ao desacoplar produtores e consumidores, reduzindo a chance de falhas em cascata, útil para eventos como criação de matches e notificações.

(08) O uso de autoscaling baseado apenas no uso de CPU é suficiente para manter a estabilidade do sistema em ambientes de microsserviços e cargas variáveis.

(16) Métricas como P95 e P99 de latência são mais relevantes que a média para identificar problemas reais de performance percebidos pelos usuários[1].

(32) Circuit breaker + timeout + retry formam um padrão de resiliência importante para evitar chamadas travadas entre serviços e reduzir amplificação de latência.

(64) Testes de performance devem ser executados apenas manualmente antes de grandes releases, pois automatizá-los em pipelines tende a gerar instabilidades e atrasos desnecessários.

Resposta: _____

Questão Discursiva (2,0 pontos) - Tema Integrador: Qualidade Contínua, Arquitetura e Resiliência no Glink

Durante o pico de usuários após viralizar no TikTok, o Glink sofreu com alta latência, falhas em cascata entre microsserviços, indisponibilidade parcial e um autoscaling ineficaz. Após os incidentes, a equipe decidiu revisar três áreas fundamentais:

1. pipeline de CI/CD e testes contínuos,
2. arquitetura e padrões de resiliência,
3. observabilidade e práticas de DDSD.

Pergunta: Explique como essas três áreas (testes contínuos no CI/CD, decisões arquiteturais e observabilidade orientada a dados) se complementam para prevenir incidentes como os que ocorreram no Glink. Em sua resposta, apresente um raciocínio integrado que mostre como práticas de qualidade, arquitetura bem definida e telemetria contínua formam um ciclo de melhoria constante para sistemas distribuídos.

(Resposta esperada: ~10 a 15 linhas.)

[1] P95 e P99 são métricas de latência baseadas em percentis e indicam, respectivamente, o tempo abaixo do qual 95% e 99% das requisições são atendidas. Enquanto a média pode mascarar lentidão em casos extremos, o P95 revela comportamentos mais lentos vividos por uma parcela relevante dos usuários, e o P99 mostra o desempenho do 1% mais lento, normalmente onde aparecem gargalos severos, saturação e problemas reais de experiência. Essas métricas são essenciais para compreender a performance percebida no mundo real.

Boa sorte!

Boa sorte!

Cenário de Avaliação: Estudo de Caso da Startup "Glink"

Visão Geral do Projeto

Em um mercado digital saturado por aplicativos de relacionamento genéricos, um grupo de empreendedores e entusiastas da cultura pop identificou uma lacuna significativa: a dificuldade de criar conexões verdadeiramente autênticas baseadas em paixões específicas. Para muitos, hobbies como games, animes, RPG de mesa e universos de ficção científica não são apenas um passatempo, mas uma parte central de sua identidade.

Assim nasce o **Glink**, um aplicativo de relacionamento e comunidade projetado desde o início para o público nerd/geek. A plataforma busca ser um ecossistema inovador que conecta pessoas com base em seus interesses, facilitando a descoberta de novos amigos, parceiros para jogar online ou um relacionamento amoroso com alguém que entenda a diferença entre um "Mecha" e um "Kaiju".

O aplicativo visa atender a uma crescente demanda por espaços digitais seguros e segmentados, onde os usuários podem expressar suas paixões sem receio, garantindo uma experiência de descoberta divertida, relevante e focada na compatibilidade de interesses.

Proposta de Valor e Diferenciais Competitivos (Funcionalidades do MVP)

Para se destacar, o **Glink** focará em quatro pilares centrais:

1. **Perfil Profundo e Personalizado**: Diferente dos perfis superficiais, o **Glink** permite que os usuários construam uma identidade digital rica.
    a. **Requisito Funcional Chave**: Implementação de um sistema de tags de interesse com múltiplas categorias (Games, Animes, Filmes, HQs, RPG, etc.), permitindo ao usuário detalhar seus gostos específicos (ex: "JRPG", "FromSoftware", "Studio Ghibli", "Sci-Fi dos anos 80").
    b. **User Story Exemplo**: “Como um fã de RPG de mesa, eu quero adicionar a tag ‘Dungeons & Dragons 5e’ ao meu perfil e destacar que sou ‘Mestre de Jogo’ para que outros jogadores possam me encontrar facilmente.”
2. **Índice de Compatibilidade Geek**: O coração do sistema. Em vez de apenas geolocalização, o algoritmo prioriza a afinidade.
    a. **Requisito Funcional Chave**: O sistema deve calcular e exibir um percentual de compatibilidade entre dois perfis, com base na sobreposição e relevância das tags de interesse.
    b. **Requisito Não-Funcional Associado**: O cálculo e a exibição do Índice de Compatibilidade na tela de descoberta não devem adicionar mais de 300ms de latência ao carregamento do perfil.
3. **Quebra-Gelos Contextuais**: O aplicativo ajuda a iniciar a conversa, diminuindo a ansiedade do "primeiro oi".
    a. **Requisito Funcional Chave**: Após um match, a tela de chat deve sugerir automaticamente perguntas ou tópicos de conversa baseados nos interesses em comum de maior destaque.
4. **Comunidade Segura e Moderada**: A criação de um ambiente acolhedor é uma prioridade máxima.
    a. **Requisito Não-Funcional Chave**: O sistema deve incluir ferramentas de denúncia de perfis e um processo de moderação claro para garantir a segurança e o bem-estar dos usuários, alinhando-se às melhores práticas de qualidade de software no quesito segurança.

Cenário da Equipe, Processo e Tecnologia

A startup "**Glink Inc.**" opera em um ambiente de alta incerteza e rápida evolução. A equipe inicial é pequena e multidisciplinar, composta por 8 pessoas: 1 Product Owner (PO), 1 Scrum Master, 3 Desenvolvedores Backend, 2 Desenvolvedores Mobile (React Native) e 1 profissional de DevOps/Infraestrutura.

- **Modelo de Processo**: A equipe adotou o framework Scrum com sprints de duas semanas para garantir entregas de valor contínuas e capacidade de adaptação rápida ao feedback dos primeiros usuários.
- **Gestão de Configuração**: Todo o código-fonte (backend, frontend e scripts de infraestrutura) é versionado em um repositório Git (no GitHub), utilizando a estratégia de Git Flow para gerenciar o desenvolvimento de novas features, correções e releases.
- **Arquitetura**: Aprendendo com os desafios históricos de sistemas monolíticos que se tornaram difíceis de manter, a equipe optou por uma arquitetura baseada em microsserviços desde o início, para permitir a evolução independente dos componentes (ex: serviço de perfis, serviço de match, serviço de chat).

Boa sorte!

Visão de Futuro e Evolução do Produto (Pós-MVP)

Para manter o engajamento e expandir o modelo de negócio, o roadmap do Glink inclui as seguintes evoluções:

- **Modo "Party Finder"**: Uma funcionalidade dedicada para usuários que não buscam um relacionamento, mas sim encontrar grupos para jogar online (LoL, Valorant, etc.) ou para formar uma mesa de RPG.
- **Eventos da Comunidade**: Integrar o app com eventos do mundo real (como a CCXP) ou eventos online (como campeonatos de e-sports), permitindo que os usuários combinem de se encontrar ou participem de atividades juntos. Isso exigiria a coordenação de múltiplas equipes, um cenário ideal para a aplicação de frameworks de escala como o SAFe.
- **Gamificação Avançada**: Implementar um sistema de conquistas e badges (ex: "Conversador Nível 5", "Mestre em Sci-Fi") para incentivar a interação e o preenchimento completo do perfil.
- **Integração com Plataformas de Conteúdo**: Permitir que os usuários conectem suas contas da Steam, PSN, ou MyAnimeList para importar e exibir seus jogos e animes favoritos automaticamente.

Este aplicativo não apenas resolverá a necessidade de conexão dentro de um nicho, mas também tem o potencial de criar uma comunidade vibrante e engajada, tornando-se o ponto de encontro digital definitivo para a cultura geek.

Boa sorte!