# **Documentação do Projeto – Faça a Festa**

# **1 – Definição do Projeto**

### a) Identificação do Problema (Cenário)

O aplicativo **“Faça a Festa”** resolve desafios recorrentes na **organização de eventos sociais** (aniversários, casamentos, chás, datas comemorativas etc.), centralizando tudo em uma plataforma digital.

**Problemas identificados:**

* **Controle de convidados**: envio de convites, confirmação de presença, agrupamento familiar e acompanhamento de status.
* **Gestão financeira**: orçamento previsto, controle de pagamentos, integração com fornecedores e histórico financeiro.
* **Fornecedores e serviços**: cadastro de fornecedores, serviços, avaliações, fotos e áreas de cobertura.
* **Comunicação**: compartilhamento de informações essenciais com convidados e fornecedores.
* **Planejamento**: acompanhamento de tarefas, checklists e cotações de compras.
* **Personalização**: referências visuais, galeria de inspirações e sugestões DIY.

### b) Objetivos

* Facilitar a **gestão completa de eventos**, do planejamento à execução.
* Centralizar **fornecedores e serviços**, com avaliações, reservas e negociações.
* Garantir **controle financeiro integrado**, com orçamentos, pagamentos e cotações.
* Melhorar a **experiência dos convidados**, via área exclusiva e convites digitais.
* Oferecer **praticidade ao organizador**, com checklists, calculadoras e dashboards estatísticos.
* Aumentar a **confiabilidade**, com avaliações, fotos de serviços e gestão de território de fornecedores.

### c) Escopo do Projeto

* **Usuários**: cadastro e perfis diferenciados (organizador, fornecedor, convidado).
* **Eventos**: cadastro, gestão de convidados, fornecedores e orçamentos.
* **Convidados**: convites digitais, confirmações, agrupamentos e relatórios.
* **Fornecedores/Serviços**: cadastro, endereços, áreas de atuação, fotos, categorias e avaliações.
* **Orçamentos e Pagamentos**: custos estimados/reais, status de pagamento, integrações.
* **Tarefas**: checklists por evento, com prazos e responsáveis.
* **Referências Visuais**: galeria categorizada, favoritos e DIY.
* **Compras e Cotações**: fluxo de requisições, fornecedores, itens e condições comerciais.

# **2 - Análise de Requisitos**

## ****2.1 Requisitos Funcionais****

1. **Usuário e Autenticação**  
   1.1 Cadastro de usuários com perfis diferenciados: organizador, convidado e fornecedor.  
   1.2 Autenticação com email/senha.  
   1.3 Login social (Google, Facebook – opcional para fases futuras).  
   1.4 Controle de permissões conforme perfil de usuário.
2. **Gestão de Endereços**  
   2.1 Cadastro de múltiplos endereços para usuários e fornecedores.  
   2.2 Definição de papéis de endereço (principal, entrega, cobrança, correspondência).
3. **Eventos**  
   3.1 Cadastro de eventos (nome, tipo, data, hora, local, descrição).  
   3.2 Associação de organizador e cidade ao evento.  
   3.3 Edição, exclusão e ativação/desativação de eventos.
4. **Convidados**  
   4.1 Envio de convites digitais via email ou redes sociais.  
   4.2 Confirmação de presença (confirmado, pendente, recusado).  
   4.3 Organização em grupos familiares ou de afinidade.  
   4.4 Exportação da lista de convidados.
5. **Fornecedores e Serviços**  
   5.1 Cadastro de fornecedores com dados de contato, fotos e descrições.  
   5.2 Cadastro de serviços vinculados a fornecedores e categorias.  
   5.3 Upload de imagens de fornecedores e serviços.  
   5.4 Definição de territórios de atendimento (área geográfica).  
   5.5 Avaliações de serviços pelos usuários.
6. **Orçamentos e Pagamentos**  
   6.1 Criação de orçamentos por evento, vinculados a fornecedores e serviços.  
   6.2 Registro de custos estimados e custos reais.  
   6.3 Controle do status de pagamento (pendente, pago, atrasado).  
   6.4 Registro de pagamentos com valor, data e forma de pagamento.  
   6.5 Anotações e observações em cada orçamento.
7. **Tarefas (Checklists)**  
   7.1 Criação de tarefas vinculadas ao evento.  
   7.2 Definição de responsáveis e datas previstas.  
   7.3 Acompanhamento do status (pendente, concluída).
8. **Referências Visuais (DIY e Inspirações)**  
   8.1 Upload de imagens e inspirações associadas ao evento.  
   8.2 Organização por categorias.  
   8.3 Marcação como favorito.
9. **Compras e Cotações**  
   9.1 Registro de requisições de compra.  
   9.2 Associação de múltiplos fornecedores a uma cotação.  
   9.3 Registro de itens da cotação (produto, quantidade, valores, descontos).  
   9.4 Comparação entre propostas de fornecedores.
10. **Comunicação e Notificações**  
    10.1 Notificações push para convidados (confirmações, lembretes, mudanças no evento).  
    10.2 Notificações para fornecedores (novas solicitações, confirmações de reserva).  
    10.3 Envio de emails automáticos (convites, lembretes e confirmações).  
    10.4 Canal de mensagens entre organizador e fornecedor (chat interno simplificado).
11. **Dashboard e Relatórios**  
    11.1 Dashboard do organizador com visão geral do evento:  
    - Convidados confirmados.  
    - Orçamento previsto vs. realizado.  
    - Status de pagamentos.  
    - Tarefas concluídas vs. pendentes.  
    11.2 Geração de relatórios exportáveis (PDF/Excel) de lista de convidados, orçamentos e pagamentos.

## ****2.2 Requisitos Não Funcionais****

1. **Usabilidade**  
   1.1 Interface intuitiva e de fácil uso.  
   1.2 Layout responsivo para diferentes dispositivos (Android, Web e Desktop).
2. **Performance**  
   2.1 Operações críticas (cadastro de convidados, atualizações financeiras) devem ter tempo de resposta inferior a 2 segundos.  
   2.2 Suporte a até n usuários simultâneos (definir meta para MVP).
3. **Disponibilidade e Confiabilidade**  
   3.1 Aplicativo disponível 24/7 com mínima indisponibilidade.  
   3.2 Mecanismos de backup automático em nuvem.  
   3.3 Recuperação de dados em caso de falhas.
4. **Segurança**  
   4.1 Criptografia de dados sensíveis (senhas, informações financeiras).  
   4.2 Autenticação via tokens seguros (JWT/OAuth2).  
   4.3 Controle de acesso por perfil de usuário.
5. **Escalabilidade e Manutenibilidade**  
   5.1 Arquitetura modular baseada em Clean Architecture.  
   5.2 Fácil evolução para incluir novas features sem impacto no núcleo.  
   5.3 Documentação técnica atualizada (Swagger/OpenAPI para APIs, README para módulos).
6. **Integração**  
   6.1 Integração com APIs externas (email, redes sociais, mapas, notificações push).  
   6.2 Integração futura com gateway de pagamentos.
7. **Internacionalização e Localização (i18n/l10n)**  
   7.1 Suporte a múltiplos idiomas (inicialmente português, futuro inglês/espanhol).  
   7.2 Adaptação de formatos de data, hora e moeda por região.
8. **Acessibilidade**  
   8.1 Compatibilidade com leitores de tela.  
   8.2 Contraste adequado de cores.  
   8.3 Navegação por teclado (no Desktop).
9. **Auditabilidade e Rastreamento**  
   9.1 Log de alterações em eventos, convidados, orçamentos e tarefas.  
   9.2 Histórico de sincronizações (offline/online).

# **3 - Design**

### a) Modelo Entidade-Relacionamento (MER/DER)

Entidades principais (SQL):

* **Usuário / Endereço**
* **Fornecedor / Serviço / Fotos / Território / Avaliação**
* **Evento / Convidado / Fornecedor\_Evento**
* **Orçamento / Pagamento**
* **Tarefa / Referência**
* **Compras (cotações e detalhes)**

### b) Interfaces Principais

* Autenticação / Cadastro
* Dashboard do Organizador
* Gestão de Eventos
* Lista de Convidados
* Orçamentos & Pagamentos
* Fornecedores & Serviços
* Checklists & Tarefas
* Espaço do Convidado
* Referências Visuais
* Compras & Cotações

# **4 – Arquitetura do Sistema**

### Visão Geral

* **Frontend**: Flutter (Android, Web, Desktop).
* **Backend**:
  + Fase 1: Firebase (Firestore, Auth, Storage, Cloud Functions).
  + Fase 2: API REST (Spring Boot/Node.js) com MySQL.
* **Offline-first**: SQLite/Drift (Android, Desktop) e cache Firestore (Web).
* **Gerenciamento de estado**: GetX.
* **Padrões de projeto**: Repository, Strategy, Adapter, Observer, Facade.
* **Princípios SOLID e Clean Architecture**.

### Camadas

* **Domain Layer**: entidades e casos de uso (independente de frameworks).
* **Data Layer**: repositórios + DataSources (local/SQLite, remoto/Firebase ou MySQL).
* **Presentation Layer**: Flutter + GetX (Views, Controllers, Bindings).

### Fluxo de Dados

UI → Controller → Use Case → Repository → DataSource (Local/Remoto) → Entities → UI.

### Offline-First

* Leituras: sempre do banco local.
* Escritas: marcadas como pending\_sync.
* **SyncService**: sincronização automática com backend quando online.

### Banco de Dados

* **Remoto**: Firestore (fase inicial), MySQL (futuro).
* **Local**: SQLite/Drift (Android, Desktop), Firestore cache (Web).

### Testes (TDD)

* **Domain**: unitários de casos de uso.
* **Data**: mocks de repositórios/DataSources.
* **Presentation**: controllers/widgets.
* **Integração**: Firebase Emulator (fase inicial), Testcontainers (MySQL).

# **5 – Benefícios da Arquitetura**

* **Enxuta, modular e escalável**.
* **Compatível com Android, Web e Desktop** no MVP.
* **Pronta para expansão futura** (API própria, MySQL, iOS).
* **Offline-first** robusto.
* **Alta testabilidade** (TDD, SOLID, Clean Architecture).
* **UI reativa e responsiva** com Flutter + GetX.