# Hello Kitty App

## 1. Planejamento e Definição

* - Ideia do projeto
* - Público-alvo
* - Funcionalidades principais
* - Requisitos técnicos e de negócio
* - Análise de concorrentes
* - Roadmap inicial

## 2. Design e UX/UI

* - Wireframes
* - Protótipo navegável
* - Design de interface (UI)
* - Experiência do usuário (UX)

## 3. Arquitetura do Projeto

* - Definição de tecnologias
* - Padrões de arquitetura
* - Regras de segurança

## 4. Desenvolvimento Backend (API)

* - Modelagem de banco de dados
* - Criação de Serializers
* - Desenvolvimento de endpoints REST
* - Configuração de autenticação JWT
* - Logs, testes e tratamento de erros

## 5. Desenvolvimento Frontend Mobile/Web

* - Integração com API
* - Gestão de estado
* - Desenvolvimento de telas
* - Cache local
* - Push notifications

## 6. Segurança

* - Configuração de HTTPS (SSL)
* - Proteções contra SQL Injection, XSS, CSRF
* - Validação forte de dados
* - Armazenamento seguro de arquivos e senhas

## 7. Deploy e Publicação

* - Deploy do backend (Railway, Render, AWS, etc.)
* - Deploy mobile (Play Store, App Store)
* - Configuração de domínio e HTTPS
* - Publicação de landing page (Vercel, Netlify, etc.)

## 8. Testes Finais

* - Testes de usabilidade
* - Testes funcionais
* - Testes de carga
* - Testes de segurança

## 9. Marketing e Lançamento

* - Criação de landing page
* - Estratégia de mídias sociais
* - Parcerias e divulgação
* - Coleta de feedback

## 10. Pós-Lançamento e Escalabilidade

* - Correções e suporte
* - Novas funcionalidades
* - Monitoramento de erros e desempenho
* - Planejamento de escalabilidade

## Documentação

* - Documentação da API
* - Guia de uso para usuários finais
* - Documentação técnica interna

**Introdução**

Este documento tem como objetivo apresentar todas as etapas, processos e decisões envolvidas no desenvolvimento do projeto **“Hello Kitty Tasks”(nome do app sujeito a alterações)** , uma aplicação mobile e web focada no gerenciamento de tarefas de forma leve, divertida e funcional.

O desenvolvimento do projeto foi realizado pela equipe composta por Victor Luis Goedicke, **Eberton Hilário da Rosa Junior**, **Carlos Kenzo**, e cada um atuando em diferentes frentes do projeto, desde design, backend, frontend até segurança,deploy e marketing.

A aplicação foi construída utilizando uma arquitetura baseada em **API RESTful**, com backend desenvolvido em **Django Rest Framework** e frontend em **Flutter**, permitindo uma experiência multiplataforma acessível tanto via mobile quanto web.

Além de oferecer funcionalidades como cadastro, autenticação, criação, edição e visualização de tarefas, o projeto prioriza uma interface intuitiva e acolhedora, com um design que reflete o universo fofo e encantador da Hello Kitty.

Este documento detalha desde o design e a experiência do usuário (UX/UI), passando pela arquitetura do sistema, desenvolvimento backend e frontend, até as implementações de segurança, testes, publicação e estratégias de lançamento.

Nosso objetivo é garantir que o projeto seja bem estruturado, seguro, escalável e proporcione uma experiência excepcional e agradável para todos os usuários.

## 1. Planejamento e Definição

* **- Ideia do projeto:**

-Desenvolver um aplicativo inovador de **gerenciamento de tarefas e lembretes**, gamificado através de um **mini-game 2D ambientado no quarto da personagem principal (Hello Kitty ou outra estética definida)**.  
O diferencial está no sistema de **recompensas virtuais**: a cada tarefa concluída, o usuário ganha moedas que podem ser usadas para **interagir com o cenário, desbloquear itens, personalizar o ambiente e interagir com a IA da personagem**.

* **- Público-alvo**
* **Faixa etária:** Adolescentes e jovens adultos (14 a 30 anos).
* **Perfis:**
  + Pessoas que gostam de organização, mas se sentem desmotivadas com métodos tradicionais.
  + Fãs de games casuais, estética kawaii, cultura pop, personagens como Hello Kitty e universos similares.
  + Pessoas neurodivergentes (TDAH, ansiedade, etc.), que se beneficiam de métodos lúdicos e visuais para produtividade.
* **- Funcionalidades principais**

**Gerenciamento de Tarefas**

* Cadastro, edição e exclusão de tarefas.
* Organização por **prioridade** (baixa, média, alta).
* Agrupamento por categorias ou áreas da vida.
* Status de conclusão e progresso.

**🔔 Push Notifications**

* Lembretes de tarefas pendentes.
* Notificações de eventos no mini-game.

**🎮 Mini-Game 2D Interativo**

* Ambiente: **quarto da personagem Hello Kitty** (ou outro tema definido).
* O avatar reage ao progresso do usuário.
* Elementos interativos: móveis, itens decorativos, objetos clicáveis.
* Sistema de recompensas:

- Ganho de moedas ou pontos por concluir tarefas.

- Desbloqueio de itens, roupas, decorações ou melhorias no ambiente.

**🧠 IA da Personagem**

* A personagem acompanha, incentiva e conversa com o usuário.
* Feedbacks positivos ao concluir tarefas.
* Dicas, lembretes e frases motivacionais.

**Definição de Prioridades**

* Sistema visual de prioridades.
* Destaque de tarefas mais urgentes ou importantes.

**🌐 Interação com Outros Usuários (Futuro)**

* Rankings de produtividade.
* Perfis de amigos.
* Compartilhamento de progresso ou conquistas no mini-game.

**🔥 Diferenciais do Produto**

* Mistura de **produtividade + entretenimento**.
* Estética kawaii altamente atraente para o público-alvo.
* IA da personagem que gera **laço emocional e engajamento**.
* Gamificação efetiva que estimula a conclusão de tarefas.
* Mini-game que cresce e evolui junto com a disciplina do usuário.
* **- Requisitos técnicos e de negócio**
* *- requisitos Técnicos:*

**Backend (API):**

- Framework:Django+Django Rest Framework

- API RESTiful estruturada

-Endpoints:

- Autenticação (JWT)

- Cadastro de usuarios

- CRUD de tarefas

- Dashboard de desempenho pessoal

- Banco de dados relacional (MySQL):

Modelos:

- User (nickname,id,idade, foto\_perfil, username)

- Task(titulo, descricao, prioridade, status, datas)

- Upload de fotos de perfil

- Deploy em servicos como railway, Render, etc

- Seguranca: validacao de dados, autenticacao forte, HTTPS

* *- requisitos de negócio:*

- Cadastro e login de usuários

- CRUD de tarefas com:(

- titulo

-descricao,

-prioridade,

-status,

datas),

- Dados do perfil:

- nickname,

- username

- idade (privativo),

- id\_user(id unico)

- Relatorios e dashboards futuros

* **- Analise de concorrentes**

**- Com Mini-Game:**

**Habitica**  
🔸 App de tarefas com sistema de RPG.  
🔸 Usuário cria um avatar, ganha recompensas, itens e sobe de nível conforme cumpre tarefas

**Forest**  
🔸 Foca em concentração: planta uma árvore que cresce enquanto o usuário não usa o celular.  
🔸 Embora não tenha tarefas diretas, trabalha com foco e gamificação

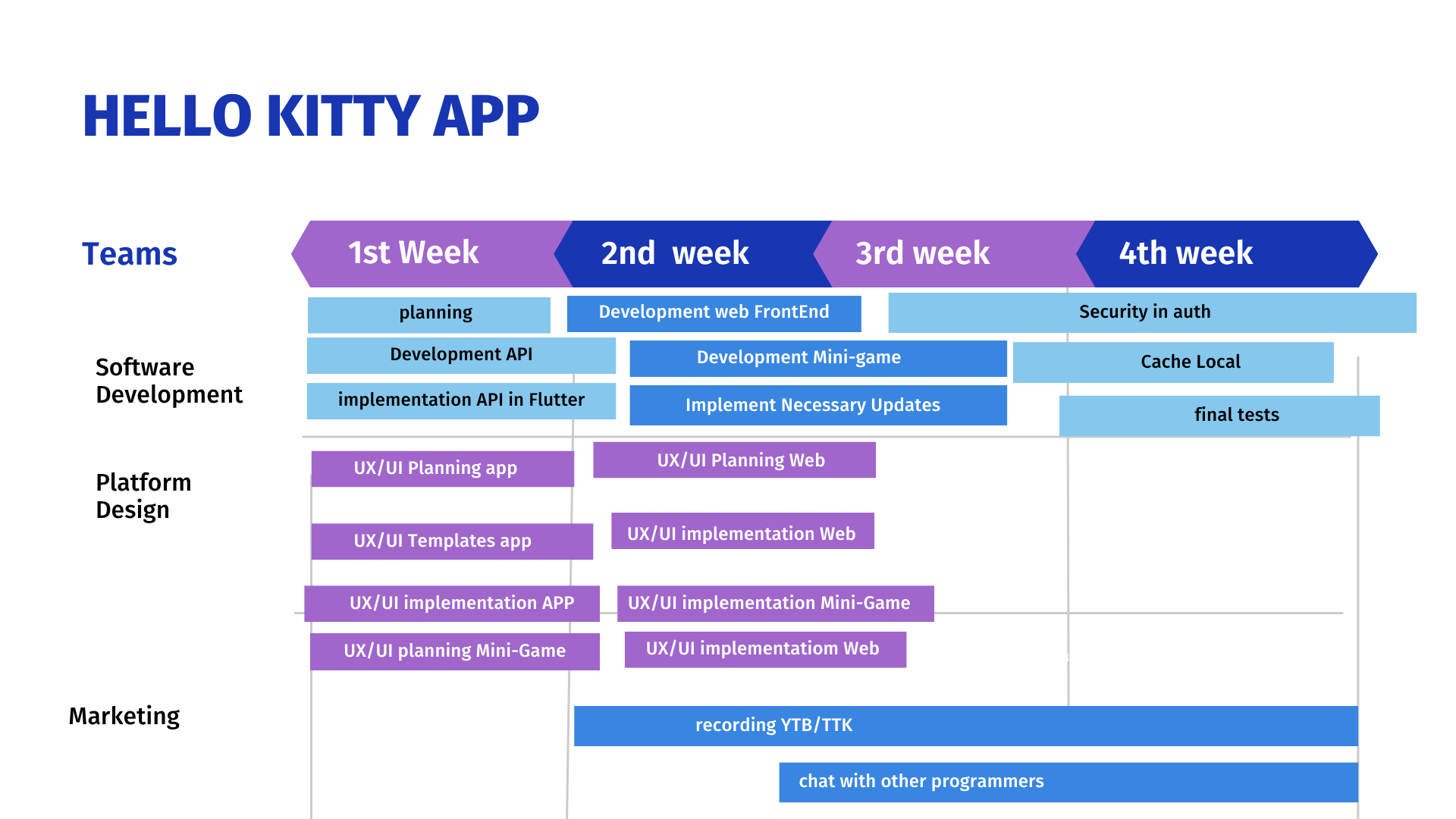
**LifeRPG**  
🔸 Organizador de tarefas em formato de missões.  
🔸 Usuário ganha XP e sobe de nível ao concluir atividades.

**Do It Now**  
🔸 Tarefas com gamificação estilo RPG.  
🔸 Oferece controle de vida, habilidades e atributos do usuário.

**- Sem Mini-Game:**

1. **Todoist**  
   🔸 Um dos apps mais populares de organização de tarefas.  
   🔸 Sem gamificação, mas muito robusto.
2. **Microsoft To Do**  
   🔸 Organizador de tarefas simples e eficiente.  
   🔸 Integração com o ecossistema Microsoft.
3. **Google Tasks**  
   🔸 App básico da Google para lembretes e tarefas.  
   🔸 Totalmente integrado ao Gmail e Google Calendar.

* **- Roadmap inicial**



## 2. Design e UX/UI

## 3. Arquitetura do Projeto

* - Definição de tecnologias:

***- Backend(API):***

- **Django**: Framework backend em python para web e APIs

- **Django** REST Framework Extensão do Django para construir APIs REST

- **SQLite** : banco de dados relacional , rodando localmente

-**Autenticação JWT** : Login e proteção de endpoints

***- Frontend:***

- **Flutter**: Framework da Google para desenvolvimento multiplataforma(Android, IOS, Web e Windows)

-**Dart** : Linguagem utilizada no flutter  
- **HTTP Package**: Para requisições da API

- **Gerenciamento de estado**: Atualmente via StatefulWidget simples(local,não global)

- **Windows Desktop**: exportando o projeto para desktop Windows

* - Padrões de arquitetura:

***- Backend(API):***

*Padrão REST + MVC(Model-View-Controller) simplificado:*

***- Frontend******:***

Padrão Clean Architecture simplificado + MVC

***- Regras de segurança:***

**- Backend(API):**

1. Autenticação JWT (✔️ Implementada)

2. Validação de dados no backend(✔️ Implementada)

3. Proteção contra SQL injection (✔️ Implementada)

4. Proteção contra XSS(Cross-Site-Scripting) ( Parcial)

5. Proteção contra CSRF ( verificar)

6. CORS Headers (✔️ Implementada)

7. Hash de senhas (✔️ Implementada)

8. Permissões por endPoint ( verificar/Parcial)

9. Rate Limiting/ Throttling ( não implementado)

10 . Protecao de dados Sensiveis (✔️ Implementada)

***-Frontend :***

*- Armazenamento seguro de tokens JWT (não implementado)*

*- Validação de inputs no app ( parcial)*

*- Tratamento de logs e erros ( parcial)*

*- Protocolo HTTPS (parcial)*

## 4. Desenvolvimento Backend (API)

* ***- Modelagem de banco de dados***

-modelos de usuario e tarefas ja implementados no django

* ***- Criação de Serializers:***

- Serializers funcionando para autenticação e criação de tarefas

* ***- Desenvolvimento de endpoints REST***

-Endpoints ja definidos:

**- /api/register**  - Registro de usuarios

**- /api/login** - Login e autenticacao de usuarios

**- /api/tasks** - metodos GET(read) e POST(create) das tasks

**- /api/tasks/<int:pk>/** - métodos PUT, DELETE, para tasks especificas

* **- Configuração de autenticação JWT**

- login com geração de token JWT funcionando ( no momento a funcao esta desativada, para fins de melhorias internas do codigo e velocidade em testes)

* **- Logs, testes e tratamento de erros**

- logs basicos feitos. Implementando tratamento de eros mais robusto

## **5**. Desenvolvimento Frontend Mobile/Web

* ***- Integração com API******: ✅ Concluído***

- integração ja concluida para as seguintes funções:

**tasks:**

-Create,

-Read,

-Delete,

- Update

**usuários:**

- login

- logout

- cadastro

* ***- Gestão de estado******: 🔧 Parcial***

- gestão simples com setState no flutter(local e funcional)

- futura atualização para gerenciadores mais robustos

* **- Desenvolvimento de telas:****✅ *Em andamento***

***-Telas ja desenvolvidas:***

***- home\_screen -*** tela inicial onde lista todas as tasks do usuario logado e permite a navegacao entre as tasks.

***- add\_task\_screen*** *- tela de criação de novas tasks .*

***- register\_screen -***tela de cadastramento de usuarios.

***-login\_screen* -** função implementada no codigo 'home\_screen' (sujeito a mudancas futuras).

* **- Cache local:** **Pendente**

- não implementado ainda

- ferramentas como Hive,SharedPreferences em analise para implementação

* **- Push notifications:** **Pendente**

- não implementado ainda

- será utilizado Firebase Cloud Messaging (FCM) ou serviços como OneSignal