Relatório de análise Estuda++

18 de setembro de 2025

Chloe Anne Scaramal
Gabriel Augusto Paiva
João Pedro Zanetti
Marcelo Gabriel Milani Santos
Marcos Antônio da Silva Junior
Paulo Daniel Forti da Fonseca
Pedro Henrique Lopes Duarte
Rayane Reis Mote
Vinicius Resende Garcia

SUMÁRIO

1. INTRODUÇÃO.
2. ARQUITETURA E TECNOLOGIAS
2.1. Componentes Principais
2.2. Backend e Gestão de Dados
2.3. Navegação e Serviços
3. ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS
3.1. Autenticação de Utilizadores
3.2. Gestão de Baralhos e Flashcards
3.3. Sessões de Estudo e Estatísticas
3.4. Assistente de IA (MonitorIA)
3.5. Funcionalidades de Geolocalização
4. LLMs
4.1. Uso de LLMs
4.2. Alguns prompts
5. CONCLUSÃO

OBJETIVO

O projeto foi implementado utilizando a linguagem Kotlin e adota uma arquitetura moderna com o padrão MVVM (Model-View-ViewModel). A interface de utilizador foi construída de forma declarativa com o Jetpack Compose, e os serviços de backend, como autenticação e base de dados em tempo real, são geridos pelo Google Firebase. O objetivo desta análise é avaliar como essas tecnologias foram aplicadas para desenvolver as funcionalidades centrais do "Estuda++", que incluem um sistema de flashcards, um assistente de IA e a gestão de locais de estudo por geolocalização. O documento termina com sugestões de melhoria e otimização do código

ARQUITETURA E TECNOLOGIAS

Esta seção descreve a arquitetura e o conjunto de tecnologias que sustentam o Estudapp, detalhando como o cliente Android (Kotlin + Jetpack Compose, em MVVM) se integra de forma segura e escalável a dois pilares de backend: Firebase (Authentication, Realtime Database e Storage) e uma API própria em Ktor dedicada ao processamento com LLMs. Explicamos os componentes principais do app, o desenho de dados e autenticação ponta a ponta via JWT do Firebase, além da navegação e dos serviços que garantem sincronização em tempo real, estudo offline-first e recursos de IA (Gemini e Groq para chat). Por fim, apresentamos o pipeline de CI/CD em Google Cloud, que automatiza build, versionamento e deploy, assegurando observabilidade, estabilidade e evolução contínua do produto.

ARQUITETURA E TECNOLOGIAS

2.1 COMPONENTES PRINCIPAIS

Linguagem e Plataforma: O aplicativo é desenvolvido em Kotlin para Android, garantindo interoperabilidade com o ecossistema Jetpack e performance nativa. Interface do Utilizador (UI): A UI é construída de forma declarativa com Jetpack Compose, permitindo componentes reutilizáveis, estados reativos e suporte a temas (Material 3) sem XML.

Arquitetura do App: Adotamos MVVM com StateFlows/LiveData para gestão de estado e desacoplamento entre UI, camada de domínio (use cases) e camada de dados (repositories). Injeção de dependência é feita com Hilt para modularidade e testabilidade.

Serviços Integrados: O app se conecta a dois serviços principais — Firebase (BaaS para autenticação, dados e mídia) e uma API própria (Ktor) responsável pelo processamento de LLMs e operações avançadas de servidor.

ARQUITETURA E TECNOLOGIAS

2.2 BACKEND E GESTÃO DE DADOS

Firebase como BaaS:

- Authentication: Gerencia cadastro, login e sessões. O token JWT do Firebase é utilizado como credencial de acesso aos endpoints da API.
- Realtime Database: Base NoSQL em tempo real para decks, flashcards, progresso do usuário e conversas. O app utiliza listeners para sincronização instantânea e cache local do SDK (sem uso de Room), mantendo a experiência responsiva mesmo com conectividade variável.
- Storage: Armazenamento de mídia (imagens/áudios) associada aos cards, com regras de segurança vinculadas ao usuário autenticado.

API de Processamento (Ktor + LLM):

- Função: Centraliza o processamento com LLMs (validação semântica de respostas abertas, geração de flashcards, cálculos de repetição espaçada e chat educativo), além de expor recursos de compartilhamento de decks.
- Provedores de IA: Integração com Gemini (Google AI) e Groq (somente chat). Demais funcionalidades de IA utilizam Gemini.
- Autorização: Todos os endpoints são protegidos com verificação do JWT do Firebase (bearer token), garantindo identidade de usuário ponta a ponta.
- Integração com Firebase: A API consulta o Firebase quando necessário para enriquecer contexto do usuário (perfis, preferências, relacionamentos de decks).

CI/CD e Infra em Nuvem (Google Cloud):

- Pipeline: Um push no ramo master aciona o Cloud Build, que compila, realiza testes e gera a imagem container da API.
- Deploy: O Cloud Build publica e implanta a imagem no Cloud Run, com configuração de variáveis de ambiente/segredos, ajuste de concorrência, autoscaling e políticas de roll-back/traffic-splitting para releases seguras.
- Segurança: Comunicação TLS, verificação de JWT no gateway da API e regras de acesso alinhadas às políticas do Firebase.

ARQUITETURA E TECNOLOGIAS

2.3 NAVEGAÇÃO E SERVIÇOS

Navegação do App:

- Navigation Compose: Rotas tipadas e grafo declarativo para telas de autenticação, listagem/edição de decks, estudo (execução de flashcards), chat com IA, estatísticas e configurações. Suporte a deep links e restauração de estado para uma experiência contínua.

Serviços e Integrações do App:

- Serviço de Sincronização em Tempo Real: Listeners do Realtime Database mantêm decks, cards e progresso sempre atualizados; WorkManager é usado para tarefas resilientes (ex.: upload de mídia ao Storage, reenvio de eventos quando a rede volta).
- Serviço de IA (Cliente): Uma camada de repositório encapsula chamadas à API Ktor (chat, geração/validação, espaçamento), anexando o JWT do Firebase e tratando timeouts/retries.
- Upload/Download de Mídia: Pipeline assíncrono para compressão/otimização de imagens/áudios antes do envio ao Firebase Storage, com obtenção segura de URLs.
- Sessões e Telemetria: Registro de eventos essenciais (estudos, acertos/erros) para métricas individuais no cliente e agregações no backend quando necessário — sempre respeitando autenticação e escopo do usuário.
- Tratamento de Conectividade: Estratégia “offline-first” via cache do SDK do Firebase e filas de operações, garantindo uso estável mesmo com rede instável.

ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS

cada uma das funcionalidades centrais da aplicação Estuda++ é analisada em detalhe, descrevendo o seu propósito

ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS

3.1 AUTENTICAÇÃO DE UTILIZADORES

Descrição Funcional: A aplicação garante que apenas utilizadores registados tenham acesso ao conteúdo. O fluxo inclui ecrãs de boas-vindas, login para utilizadores existentes e um formulário de registo para novos utilizadores, utilizando email e senha.
Implementação Técnica: A lógica de autenticação é centralizada no AuthViewModel.kt, que interage diretamente com o serviço Firebase Authentication. As interfaces de utilizador para este módulo, SignInScreen.kt e SignUpScreen.kt, são compostas por campos de texto e botões que invocam as funções do ViewModel para realizar as operações de login (signIn), registo (signUp) e logout (signOut). O estado da autenticação (se o utilizador está logado ou não) é observado em toda a aplicação para controlar o acesso às rotas protegidas.

ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS

3.2 GESTÃO DE BARALHOS E FLASHCARDS

Descrição Funcional: A aplicação garante que apenas utilizadores registados tenham acesso ao conteúdo. O fluxo inclui ecrãs de boas-vindas, login para utilizadores existentes e um formulário de registo para novos utilizadores, utilizando email e senha.
Implementação Técnica: A lógica de autenticação é centralizada no AuthViewModel.kt, que interage diretamente com o serviço Firebase Authentication. As interfaces de utilizador para este módulo, SignInScreen.kt e SignUpScreen.kt, são compostas por campos de texto e botões que invocam as funções do ViewModel para realizar as operações de login (signIn), registo (signUp) e logout (signOut). O estado da autenticação (se o utilizador está logado ou não) é observado em toda a aplicação para controlar o acesso às rotas protegidas.

ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS

3.3 SESSÕES DE ESTUDO E ESTATÍSTICAS

Descrição Funcional: O utilizador pode iniciar uma "sessão de estudo" a partir de um baralho. A aplicação apresenta os cartões um a um, e o utilizador interage para revelar a resposta e indicar se acertou ou errou. O progresso é registado e apresentado num ecrã de perfil ou estatísticas.
Implementação Técnica: O StudyViewModel.kt é responsável por carregar os flashcards de um baralho, controlar a lógica da sessão (qual cartão mostrar, verificar a resposta) e registar os resultados. A interface da sessão de estudo está em StudyScreen.kt. Os resultados são guardados como objetos DeckPlayStatDTO.kt no Firebase. O StatsViewModel.kt posteriormente lê estes dados para exibi-los no ecrã de perfil (ProfileScreen.kt).

ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS

3.4. ASSISTENTE DE IA

Descrição Funcional: A aplicação disponibiliza um chatbot chamado "MonitorIA", onde o utilizador pode enviar perguntas e receber respostas geradas por uma inteligência artificial, atuando como um tutor virtual.
Implementação Técnica: Esta funcionalidade é encapsulada no ChatScreen.kt (UI) e ChatViewModel.kt (lógica). O ViewModel gere a lista de mensagens e comunica-se com uma API externa de IA (a implementação exata da chamada à API não está visível, mas a estrutura para tal existe). As mensagens trocadas, modeladas pela classe SimpleChatMessageDTO.kt, são guardadas no Firebase Realtime Database para manter o histórico da conversa.

ANÁLISE DAS FUNCIONALIDADES PRINCIPAIS

3.5. FUNCIONALIDADES DE GEOLOCALIZAÇÃO

Descrição Funcional: Os utilizadores podem visualizar um mapa, guardar "locais de estudo" favoritos e receber notificações quando entram ou saem dessas áreas predefinidas, incentivando o início de uma sessão de estudo.
Implementação Técnica: O LocationViewModel.kt gere o estado relacionado com a localização, como a obtenção da posição atual do utilizador e a lista de locais favoritos. O MapScreen.kt integra o componente do Google Maps para exibir os dados. A funcionalidade de notificação por proximidade é implementada através da API de Geofencing, configurada no GeofenceManager.kt e ativada por um GeofenceBroadcastReceiver.kt, que escuta os eventos de transição (entrada/saída) do utilizador nas áreas definidas.

LLMS

Nesta secção, são detalhadas as diretrizes técnicas para a evolução da aplicação Estuda++, a implementação de novas funcionalidades e a resolução de problemas identificados durante o desenvolvimento. A análise a seguir consolida as informações fornecidas pela equipa, definindo os próximos passos para cada área de responsabilidade.

LLMS

4.1. USO DE LLMS

A elaboração deste relatório e do planeamento técnico subsequente foi realizada através de um processo colaborativo e interativo com um Modelo de Linguagem de Grande Escala (LLM). A abordagem transcendeu a simples geração de texto; a IA foi utilizada como uma ferramenta de engenharia de software para acelerar e aprofundar a análise, atuando como um parceiro técnico sob a direção e validação humanas. O método foi fundamentado em um diálogo contínuo, onde o direcionamento humano foi crucial para extrair o máximo potencial da ferramenta. O processo iniciou-se com o fornecimento do contexto completo à LLM, incluindo o acesso integral ao repositório de código e a definição da estrutura da equipa e dos seus objetivos, estabelecendo uma base de conhecimento sólida para as etapas seguintes.

A partir dessa base de conhecimento, a LLM foi instruída a atuar em diferentes papéis especializados, demonstrando sua capacidade de contextualização. Através do prompt "Atue como um engenheiro de software que precisa implantar uma API...", a IA gerou um plano de ação detalhado e prático para a equipa de backend, incluindo a configuração de um pipeline de CI/CD no Google Cloud. De forma análoga, para a equipa de Android, ao ser instruída a atuar "como especialista em desenvolvimento android kotlin/compose", a ferramenta não apenas gerou o código para o novo componente de estatísticas, mas também diagnosticou um erro crítico (NullPointerException) a partir de um stack trace. A solução proposta foi a implementação de uma prática recomendada de arquitetura — o uso de StateFlow com collectAsStateWithLifecycle — garantindo a reatividade e a segurança da interface, o que efetivamente reforçou os padrões de qualidade do projeto. Em suma, a LLM foi empregue como um catalisador de produtividade, transformando requisitos de alto nível em soluções técnicas detalhadas e estruturando o conhecimento do projeto neste documento, sempre em sinergia com a validação e o refinamento da equipe.

Lmm usadas foram a Gemini, chat gpt e groq.

LLMS

4.1. USO DE LLMS

Durante a execução do projeto Estuda++, a equipe enfrentou desafios técnicos e de gestão inerentes ao desenvolvimento de um sistema complexo com múltiplos integrantes. Estes obstáculos, embora exigentes, foram cruciais para o amadurecimento técnico e a otimização dos nossos processos de colaboração.

Um dos principais desafios surgiu na estruturação e integração com o Firebase. A definição de um modelo de dados coeso e escalável, que atendesse às necessidades de diferentes funcionalidades, exigiu um esforço de comunicação significativo. A funcionalidade de chat, por exemplo, necessitava de uma estrutura específica para as mensagens, que precisava ser alinhada com a equipe responsável pela integração geral do Firebase, para evitar inconsistências e retrabalho. Esta situação evidenciou a necessidade de definir contratos de dados claros e de realizar sessões de alinhamento frequentes entre os programadores.

A integração geral entre as diferentes frentes de trabalho também se revelou um ponto de atenção. As dependências entre tarefas — por exemplo, a interface de utilizador (UI) que dependia da lógica de acesso aos dados já estar funcional — criaram alguns blocos temporários. Superar isto exigiu uma gestão de tarefas mais dinâmica, onde a comunicação sobre o progresso e os impedimentos de cada um se tornou fundamental para o avanço do projeto como um todo.

Tecnicamente, a configuração dos serviços de geolocalização, como a API do Google Maps e o Geofencing, apresentou uma curva de aprendizagem acentuada. A correta configuração das chaves de API, a gestão das permissões no Android e a implementação de um sistema de notificações que não consumisse excessivamente a bateria do dispositivo foram desafios que exigiram pesquisa dedicada e várias iterações de testes para garantir o funcionamento esperado.

Por fim, estes pontos convergiram para um desafio maior de gestão e sincronização da equipe. Manter todos os nove membros alinhados, cientes das dependências e a trabalhar de forma coesa, foi uma lição contínua em comunicação e gestão de projetos. A experiência reforçou a importância de ter papéis bem definidos, como o do arquiteto cross, que atuou como um facilitador para resolver impasses e garantir que a visão técnica do produto se mantivesse consistente entre o frontend, o backend e os serviços integrados. Cada um destes desafios, ao ser superado, não só melhorou a qualidade do código, como também fortaleceu a nossa capacidade de operar como uma equipe de desenvolvimento unificada e eficiente.

LLMS

4.2. ALGUNS PROMPTS

De acordo com o arquivo anexado, estamos desenvolvendo um projeto que visa possibilitar o estudo por meio de flashcards. Somos uma equipe de 9 integrantes, e as responsabilidades foram divididas da seguinte forma: Marcos: Desenvolvimento da API em Ktor João: Integração com ferramentas de AI Marcelo: Geolocalização Paulo e Pedro: Integração do App com Firebase e API Chloe e Rayane: Desenvolvimento das interfaces Gabriel: Refinamento de produto, arquitetural e atuação cross Vinícius: Documentação do projeto Faça o entendimento do projeto e posteriormente discutiremos detalhes mais específicos sobre a aplicação

Atue como um engenheiro de software que precisa implantar uma API em Kotlin (utilizando o Ktor). Inicialmente deve ser desenvolvido uma PoC com apenas um hello world e submetida utilizando o Jib para "containalizar" o app e implata-lo no Cloud Run por meio de uma sincronização direta do repositório no github. Utilize o Cloud Build, com o trigger apropriado

como especialista em desenvolvimento android kotlin/compose, desenvolva o componente responsável por apresentar as estatísticas de respostas nos Decks de um determinado usuário, seguindo o padrão já implementado no repositório anexado o componente deve ser implementado na tela de perfil, de forma que possibilite dois tipos de visão: Por Localização e por Deck, que podem ser alterados por meio de um toggle panel

a interface deve seguir o padrão da imagem anexada, com o seguinte padrão para definição das cores: 0 - 49,999%: Vermelho 50% - 79,9999% - Amarelo 80% - 100% Verde

deve-se seguir o padrão utilizado no repositório, alterando minimamente a estrutura do código essas informações devem ser extraídas do firebase realtime database, verifique se já não há uma estratégia extração das informações no repositório. Além disso, deve ser realizado o cálculo de acordo com a localização informada na resposta cruzando com as localizações favoritadas. O mesmo para o a definição dos decks, consultando seu nome com base no id

CONCLUSÃO

---

5. FUNÇÕES DOS PARTICIPIANTES

Chloe – Implementou as interfaces em Compose conforme o Figma, cobrindo navegação, estados (carregando/erro/offline) e componentes reutilizáveis. Realizou QA visual, ajustes de acessibilidade e testes instrumentados, integrando UI aos ViewModels e à resposta da IA.

Gabriel – Liderou produto, arquitetura e gestão do time, priorizando o backlog e definindo critérios de aceite. Orquestrou handoffs entre API, IA, UI e dados, removeu bloqueios e estruturou o roteiro/ensaio da demo para garantir a entrega no prazo.

João – Orquestrou os provedores de LLM para geração e validação de flashcards (incluindo Cloze e dicas). Definiu prompts e contratos JSON, configurou fallback e telemetria, e estabilizou latência/erros para respostas úteis e controladas.

Marcelo – Entregou geolocalização com permissões, locais favoritos e geofences (enter/exit) alimentando analytics por local.

Marcos – Desenvolveu a API em Ktor com rotas de flashcards, serviços de IA, estatísticas e compartilhamento, documentadas em OpenAPI/Swagger. Implementou logging, segurança mínima (JWT/sessions), rate limiting e contratos estáveis.

Paulo – Modelou dados (decks, cards, sessões) e implementou Firebase Auth, Firestore e regras de segurança. Integrrou eventos do RTDB ao app e produziu guias de execução, garantindo coerência e confiabilidade do armazenamento.

Pedro – Integrau o app à API via Retrofit/Interceptors e implementou modo offline (queues/retry). Entregou o fluxo de share-link (follow/unfollow) e a sincronização híbrida, monitorando erros e tempos de carregamento.

Rayane – Prototipou no Figma a experiência ponta a ponta (Home, Deck, Estudo, Estatísticas, Perfil) e definiu o Design System. Entregou assets/specs, validou usabilidade/acessibilidade e apoiou o desenvolvimento com ajustes finos.

Vinícius – Documentou o projeto (relatório) e elaborou os slides da apresentação.