# Plano Robustíssimo de Desenvolvimento do Projeto ECGiga

## Visão Geral e Objetivos do Projeto

O projeto **ECGiga** é um curso interativo e aberto de eletrocardiografia clínica, visando criar a maior e mais abrangente plataforma de aprendizado de ECG para profissionais de saúde. Ele combina múltiplos componentes – uma interface de linha de comando (CLI) didática, um aplicativo web dinâmico, recursos de IA e um extenso banco de questões de quiz – para oferecer uma experiência de aprendizado multifacetada[[1]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L6-L14). Em resumo, o objetivo central é prover uma plataforma robusta onde estudantes e profissionais possam praticar interpretação de ECG por meio de quizzes validados, analisar traçados com ferramentas de visão computacional e receber apoio de modelos de IA no processo de aprendizado.

**Missão:** construir o maior e mais completo curso open-source de ECG, atingindo uma ampla comunidade de estudantes de medicina e profissionais de saúde[[2]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L8-L12). Para isso, o projeto foca em conteúdos de alta qualidade (ex.: explicações clínicas detalhadas em cada questão do quiz) e em ferramentas interativas (CLI, web, notebooks e futuramente aplicativos móveis).

**Escopo Inicial:** abranger os fundamentos de ECG (ondas, intervalos, derivações) até conteúdos avançados, integrando **recursos de IA** gradualmente (como geração assistida de laudos e tutoria guiada por IA) conforme o projeto evolui[[3]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L12-L15). O alcance previsto inclui: - Uma aplicação **CLI** interativa para executar quizzes e análises de ECG via terminal. - Uma **aplicação Web** para visualização de traçados, cálculo de parâmetros e simulações fisiológicas. - Ferramentas de **Visão Computacional (CV)** para processar imagens de ECG (deskew, segmentação, detecção de picos R, etc.). - Um amplo **banco de quizzes** em formato JSON, validado por um esquema, cobrindo diversos tópicos de ECG. - Integração futura de **modelos de Linguagem** para fornecer interpretações automáticas de ECG (laudos preliminares) e feedback inteligente aos usuários.

## Arquitetura do Sistema e Componentes Principais

Para atender aos objetivos, a arquitetura do ECGiga é modular, dividindo responsabilidades em componentes bem definidos:

* **Interface de Linha de Comando (CLI)** – Implementada em Python (utilizando frameworks como Typer e Rich), oferece comandos para interagir com o conteúdo. Por exemplo, é possível rodar quizzes, validar arquivos de perguntas, gerar quizzes a partir de laudos e executar análises numéricas ou via imagem diretamente no terminal[[4]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L23-L31). Esse CLI permite uso rápido por parte do usuário avançado e também serve como backend para certas funções do sistema.
* **Aplicativo Web (Dash)** – Uma aplicação web construída com Plotly Dash que fornece uma interface gráfica interativa. Ela inclui um visualizador de ECG de 12 derivações com zoom e pan, calculadoras interativas (por exemplo, para QTc), upload de imagens de ECG para análise, e sliders de simulação que demonstram efeitos de alterações iônicas nas ondas[[5]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L36-L41). O webapp torna o conteúdo acessível via navegador, atingindo usuários que preferem interface visual.
* **Módulo de Visão Computacional (CV)** – Conjunto de ferramentas Python para processamento de imagens de ECG, encapsulado no diretório cv/. Inclui funções para correção de inclinação de imagem (deskew), normalização de escala, detecção de grade de fundo, segmentação de traçados em derivações individuais, reconhecimento de rótulos de derivações (OCR) e detecção de picos R e intervalos importantes[[6]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L45-L53). Esses utilitários suportam tanto o CLI (ecgcourse ingest image) quanto o Web (upload de imagem) na análise automatizada de ECGs impressos ou em PDF.
* **Motor de Quizzes (Quiz Engine)** – Gerencia o banco de perguntas de múltipla escolha. As questões residem em arquivos JSON (no diretório quiz/bank/), seguindo um esquema rigoroso (quiz/schema/mcq.schema.json) para garantir consistência e qualidade[[7]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L55-L62). Há scripts para gerar novos quizzes a partir de dados de laudos ou outras fontes. Esse motor permite validar arquivos de quiz e executar quizzes tanto via CLI quanto possivelmente via interface web. É fundamental para a componente educacional, cobrindo centenas de questões com explicações detalhadas.
* **Notebooks Educacionais** – Uma coleção de notebooks Jupyter (notebooks/) complementa o aprendizado com exemplos e tópicos específicos (e.g., fórmulas de QTc, eixo cardíaco, filtros de artefatos)[[8]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L63-L67). Esses notebooks servem como material de apoio teórico-prático e também documentam explorações de desenvolvimento (por exemplo, protótipos de análise de sinal).
* **Documentação e Guias** – O diretório docs/ contém guias de estilo, manual de contribuição, e o **Roadmap** do projeto, que delineia as fases de evolução[[9]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L107-L114). Manter a documentação atualizada é crucial para um projeto robusto; portanto, cada componente acima possui README ou seções documentando seu propósito, uso e interfaces.

Essa arquitetura modular facilita a divisão de tarefas de desenvolvimento e permite evolução incremental. Por exemplo, melhorias no módulo de visão computacional podem ser feitas isoladamente e depois integradas ao CLI/Web. Da mesma forma, a expansão do banco de quizzes não afeta diretamente o código do aplicativo, desde que siga o esquema definido.

## Ferramentas, Tecnologias e Abordagem de Desenvolvimento

Para implementar e gerir esse projeto de forma holística, utilizaremos um conjunto robusto de ferramentas e práticas de engenharia de software:

* **Linguagens e Frameworks:** O core do projeto é em **Python** (facilitando a integração entre CLI, web Dash e CV). O CLI usa Typer/Rich para UX no terminal; o web app usa Dash (Plotly) para construir a interface interativa em Python; a visão computacional apoia-se em bibliotecas como OpenCV e PyTorch/TensorFlow (eventualmente, para modelos de detecção de características). Para uma possível aplicação móvel nativa (fase posterior), consideraremos **Swift/SwiftUI** no Xcode (para iOS/macOS) ou **Kotlin** (para Android), ou alternativamente uma abordagem *PWA* (Progressive Web App) para reaproveitar o front-end web.
* **Controle de Versão:** O repositório será gerenciado com **Git/GitHub** para acompanhamento de código, *issues* e *pull requests*. Isso permite colaboração e revisão de código estruturada. Iremos adotar convenções de *branching* (como estratégia Git Flow ou Trunk-Based) e integração contínua para garantir qualidade a cada alteração.
* **Qualidade de Código:** Ferramentas de linting e formatação automáticas, como **Black** e **Ruff**, já são utilizadas no projeto[[10]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L105-L109) e continuarão garantindo um estilo consistente. Adicionaremos testes automatizados abrangentes (unitários e de integração) para o CLI, funções de análise e componentes críticos do web app. Esses testes serão executados localmente e em pipeline CI (por exemplo, via GitHub Actions).
* **CI/CD e Deploy:** Configuraremos *pipelines* de Integração Contínua para executar os testes a cada commit/pull request e *pipelines* de Deploy Contínuo para publicar versões estáveis. Como o app web é em Dash, podemos empacotá-lo em um contêiner **Docker** para implantação em um servidor ou nuvem (AWS, Heroku, etc.). O CLI e pacotes Python podem ser distribuídos via PyPI. Ferramentas de automação permitirão deploy rápido; vale notar que CLIs de IA podem se integrar a essas etapas – por exemplo, os agentes de CLI com IA conseguem interagir com Git e Docker diretamente[[11]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Workflow%20Integration), auxiliando na configuração de contêineres e pipelines.
* **Assistentes de Código com IA:** Aproveitaremos ao máximo ferramentas de IA no terminal, como **OpenAI Codex CLI** e **Google Gemini CLI**, que funcionam como “pares programadores” dentro do fluxo de desenvolvimento. Diferentemente de autocompletes tradicionais, esses CLIs inteligentes entendem o contexto de todo o projeto e podem sugerir modificações multi-arquivo, executar scripts de build/teste e até gerar pull requests automaticamente[[12]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Editor%20tools%20assist%20with%20small,edits%20or%20writing%20functions). Por exemplo, poderemos usar o Codex CLI para gerar scaffolds iniciais de código ou refatorar trechos complexos: ele lerá nosso repositório e proporá mudanças diretamente, que poderemos aprovar ou ajustar no terminal[[13]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Try%20commands%20like%3A). Já o Gemini CLI pode orquestrar tarefas como criar um plano de melhoria de documentação e até propor *diffs* em arquivos para padronizar estilo, conforme praticado nos laboratórios internos. Essas ferramentas trazem ganhos de produtividade significativos, ajudando a gerenciar o projeto inteiro via terminal[[14]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=inside%20editors%20by%20offering%20smart,project%20directly%20from%20the%20terminal)[[15]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=AI%20CLIs%20often%20process%20much,just%20one%20function%20or%20file) – seja renomeando arquivos, rodando suítes de teste ou automatizando tarefas de implantação.
* **Xcode e Desenvolvimento Mobile:** Caso avancemos para aplicativos nativos Apple (macOS/iOS), usaremos o **Xcode** como IDE principal. A estratégia pode ser construir um app iOS que consuma funcionalidades do ECGiga via uma API (provavelmente expondo certas capacidades do backend Python via um serviço web/FastAPI). O Xcode será crucial para projetar uma UI amigável para dispositivos móveis e aproveitar recursos nativos. Planejamos uma arquitetura onde a lógica central (por exemplo, validação de quiz ou análise de ECG) reside no core Python, e o app iOS atua como cliente – isso evita duplicar lógica em duas linguagens. Para integração fluida, poderemos gerar bibliotecas ou usar componentes como *PythonKit* em Swift, embora a opção mais sustentável seja expor serviços web e fazer o app consumir esses serviços.

Em resumo, a abordagem de desenvolvimento será **iterativa e orientada a melhorias contínuas**. Definiremos metas para cada etapa (veja as fases abaixo) e utilizaremos metodologias ágeis (sprints curtos, acompanhamento de tarefas via issues). As poderosas ferramentas de CLI com IA serão nossas aliadas para acelerar tarefas repetitivas ou complexas, mas sempre com validação humana antes de mesclar mudanças críticas. Cada funcionalidade será integrada gradualmente, garantindo que todo novo componente funcione bem com os existentes.

## Fases e Etapas de Implementação do Projeto

Para tornar o plano mais tangível, dividimos o desenvolvimento em fases incrementais (p0, p1, ..., p5), alinhadas com o **roadmap** definido pelo projeto[[16]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L10). Cada fase introduce um conjunto de funcionalidades construídas sobre as anteriores, permitindo teste e validação progressiva:

1. **Fase p0 – Fundamentos e Estrutura Inicial:** Preparar todo o esqueleto do projeto e ambiente de desenvolvimento. Nesta etapa inicial, cria-se a estrutura de diretórios base (CLI, web\_app, cv, quiz, docs, etc.) e arquivos *stub* principais (ex.: um comando CLI básico, app Dash mínimo rodando, esquema JSON de quiz definido). Também inclui configurar dependências (arquivo requirements.txt) e ambientes virtuais. Será adicionada documentação inicial: guias de contribuição, estilo de código e um README principal explicando o projeto. *Outputs* esperados: projeto compila/roda “vazio” (CLI responde a --help, web app abre página básica), 3 notebooks introdutórios criados com conteúdos seed e um conjunto mínimo de perguntas no banco de quizzes para testar estrutura (por exemplo, 5-10 questões de exemplo validadas pelo schema). Essa fase estabelece a base sólida sobre a qual tudo será construído[[17]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L5).
2. **Fase p1 – MVP do Quiz e Visualização Básica:** Foco em entregar um **Produto Mínimo Viável** com as funcionalidades educacionais centrais. Completar a implementação do CLI de quizzes, permitindo rodar quizzes completos a partir de arquivos JSON e validar esses arquivos contra o esquema. Popular o banco de questões com pelo menos 50 **MCQs (multiple-choice questions)** revisadas, cobrindo tópicos básicos de ECG (ondas P, QRS, T, intervalos PR/QT, etc.). Paralelamente, evoluir o aplicativo Dash para exibir uma ECG de 12 derivações estática com funcionalidade de zoom/pan[[17]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L5) – por exemplo, usar um gráfico estático representando 12 traçados simulados ou carregados de um dataset, dando ao usuário a capacidade de aproximar partes do traçado. Essa prova de conceito visual confirma o pipeline de renderização gráfica no web app. Ao final do p1, o sistema já oferece valor: o usuário pode estudar conceitos básicos respondendo quizzes via CLI e visualizar um ECG no navegador. *Assistência Codex/Gemini:* podemos utilizar o Codex CLI para acelerar a codificação do validador de schema JSON ou da lógica de pontuação do quiz, e usar Gemini CLI para revisar a clareza das explicações das respostas nas 50 questões, garantindo consistência de estilo nas justificativas.
3. **Fase p2 – Expansão do Conteúdo e Análises Numéricas:** Expandir tanto o conteúdo quanto as funcionalidades de análise. Aumentar o banco de quizzes para ~150 questões, abrangendo casos moderadamente avançados e incorporando feedback dos usuários iniciais. Introduzir no CLI (e possivelmente no web) comandos de **análise por valores numéricos**: por exemplo, permitir ao usuário inserir manualmente medidas de intervalos (PR, QRS, QTc) e eixo do QRS, e o sistema fornecer uma interpretação baseada nesses valores (heurísticas do tipo: “PR prolongado sugere bloqueio AV de 1º grau”, etc.). Isso adiciona uma dimensão de treinamento focada em parâmetros quantitativos. Incluir pelo menos 5 notebooks úteis cobrindo tópicos como cálculo do eixo elétrico e exemplos de identificação de arritmias simples – esses notebooks servem tanto para aprendizado do usuário quanto para validar algoritmos que poderão ser integrados no código. Do lado do web app, podemos implementar um painel simples onde o usuário inputa valores e vê dicas/resultados (ou integrar isso na CLI/Web existente). Essa fase consolida o conteúdo fundamental e começa a trazer inteligência ao sistema por meio de análises programáticas de valores típicos de ECG[[18]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L4-L6).
4. **Fase p3 – Integração de Imagem (Visão Computacional) e IA Inicial:** Marco importante onde unimos visão computacional e inteligência artificial ao projeto. Implementar o recurso de **upload de imagem de ECG** no web app e o comando correspondente no CLI (ecgcourse ingest image <arquivo> já previsto) para processar um arquivo de ECG escaneado. Integrar as ferramentas do módulo CV: ao receber uma imagem, aplicar deskew, detectar grade, segmentar os traçados, extrair picos R e intervalos-chave. Com isso, gerar um conjunto de dados estruturados (por exemplo, frequência cardíaca estimada, durações de intervalos, etc.). Nesta fase, introduzimos uma **IA assistiva** simples: usar um modelo de linguagem (via API de um LLM como GPT-4 ou Gemini 2.5) para produzir um **laudo preliminar** baseado nos dados extraídos – por exemplo, “ritmo sinusal, FC ~75 bpm, PR e QRS normais, QTc levemente prolongado...”[[3]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L12-L15). Esse laudo automático será exibido ao usuário como um ponto de partida, **com disclaimers** de que não é diagnóstico médico oficial. Além disso, desenvolver **casos interativos**: cenários clínicos montados onde o usuário recebe uma história + ECG (imagem ou valores) e deve responder perguntas ou identificar achados, combinando assim quizzes com análise de imagem. Aumentar o banco de questões para ~300, incluindo itens referentes aos novos tópicos (interpretação de achados de imagem, casos clínicos). Permitir que o usuário exporte um relatório/laudo dos resultados de uma análise de ECG em formato texto ou PDF. *Detalhe técnico:* talvez seja necessário um backend mais robusto aqui – considerar introduzir um serviço **FastAPI** (ou expandir Dash callbacks) para lidar com processamento pesado de imagens e requisições ao modelo de IA, evitando travar a interface. Ao final do p3, o sistema será capaz de receber um ECG real e produzir uma análise inicial automatizada, representando um grande avanço em funcionalidade[[19]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L6-L8).
5. **Fase p4 – Aprimoramento da Plataforma de Aprendizado:** Aqui refinamos a experiência do usuário e adicionamos funcionalidades avançadas para tornar o aprendizado mais profundo. No aplicativo web, implementar ferramentas de **filtro e simulação**: por exemplo, permitir aplicar filtros de frequência no traçado (para ensinar sobre ruídos e artefatos) e incluir a **simulação de efeitos iônicos** nas ondas (um componente onde o usuário ajusta sliders de níveis de potássio, cálcio, etc., e o traçado se modifica em tempo real, ilustrando princípios eletrofisiológicos)[[19]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L6-L8). Tornar a aplicação mais **adaptativa**, registrando o desempenho do usuário nos quizzes/casos e fornecendo relatórios de progresso: ao final de cada quiz ou caso, mostrar áreas de acerto e pontos a melhorar, possivelmente recomendando leituras nos notebooks correspondentes. Essa fase pode envolver o desenvolvimento de um sistema simples de perfil do usuário (mesmo que local, armazenando histórico em JSON ou base de dados leve) para personalizar a experiência. Em termos de documentação, consolidar um manual do usuário tanto para o CLI quanto para o web, e um guia de referência de ECG cobrindo todos os tópicos do conteúdo – esses materiais podem ser gerados parcialmente a partir dos notebooks ou usando a IA para compilar textos, depois revisados por humanos. Após p4, o ECGiga deve apresentar uma experiência rica: o usuário tem conteúdo teórico, prática por quiz, análise automática e feedback adaptativo, cobrindo um espectro amplo do aprendizado de ECG.
6. **Fase p5 – Distribuição e Acesso Ampliado:** Com as funcionalidades principais implementadas e validadas, focamos em **distribuir** e facilitar o acesso ao ECGiga. Uma iniciativa importante é o **empacotamento da aplicação**: transformar o web app em uma aplicação instalável (considerar um *Progressive Web App* para uso offline no navegador e/ou empacotar via Electron ou similar para desktop). Para o público de macOS/iOS, podemos criar um app nativo (utilizando Swift/Xcode) que consuma internamente os componentes do projeto – por exemplo, rodando um pequeno servidor local com as funcionalidades ou acessando uma API na nuvem. Alternativamente, um **app macOS** poderia ser feito via PyInstaller/PyOxidizer para distribuir o CLI + Dash como um aplicativo desktop unificado. Também avaliar lançar uma versão Android (Kotlin) ou ao menos garantir que o PWA funciona bem em dispositivos móveis Android. Durante essa fase, podemos introduzir um backend FastAPI completo (se não feito no p3) para oferecer uma **API REST** do ECGiga – isso permitiria a outros clientes (móvel, por exemplo) utilizarem as funcionalidades (quiz, análise) via internet. Além disso, configurar a publicação oficial: docker images para o servidor, pacote no PyPI, talvez um executável standalone. **Segurança e validação** também são pontos de atenção aqui: antes de liberar amplamente, realizar testes extensivos de usuário (incluindo profissionais médicos se possível) para validar a acurácia do conteúdo e das análises automáticas. Qualquer ajuste de última hora no algoritmo de análise ou correções de conteúdo deverão ser feitos neste estágio. Em suma, p5 é sobre tornar o projeto robusto e fácil de usar “fora do ambiente de desenvolvimento”, alcançando usuários finais nas plataformas desejadas[[20]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L7-L9).
7. **Fase p6+ – Expansão Contínua e Manutenção:** Após a versão inicial completa, o projeto entrará em um ciclo contínuo de melhorias **(p6–p30)**. As diretrizes futuras incluem: expansão do conteúdo avançado (arritmias complexas, marca-passo, síndromes raras), incorporação de novos datasets de ECG para treino e validação (e.g. bases do PhysioNet), e validação clínica ampla dos algoritmos com feedback de especialistas[[21]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L7-L10). Do ponto de vista de IA, à medida que modelos mais poderosos surgirem (por exemplo, versões futuras do Gemini ou modelos especializados em ECG), poderemos integrá-los para melhorar as interpretações automáticas e fornecer tutoria adaptativa mais sofisticada. Na parte de infraestrutura, planejamos manter o projeto atualizado com dependências seguras e performáticas, melhorar a eficiência do código (por ex., otimizar o processamento de imagem em C++/Rust se necessário) e escalar a arquitetura (se o uso crescer, talvez migrar a nuvem com auto-scaling para a interface web/API). **Manutenção**: estabelecer um processo de revisão contínua de conteúdo – medicina é uma área em evolução, então revisar periodicamente as diretrizes de interpretação de ECG e atualizar o material de acordo. Continuar utilizando as ferramentas de automação (Codex CLI, Gemini CLI) nas manutenções, por exemplo, para refatorar código legadamente quando ampliarmos funcionalidades, ou gerar documentação das novas APIs automaticamente.

Cada fase acima termina com uma versão utilizável do sistema, permitindo coletar feedback antes de prosseguir. As etapas foram definidas com base no roadmap original do projeto para garantir alinhamento com os objetivos traçados[[16]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L10). Esse desenvolvimento incremental minimiza riscos, pois funcionalidades centrais são validadas primeiro (quiz, visualização), antes de investir em recursos mais complexos (visão computacional, IA). Além disso, facilita contribuições open-source: outros desenvolvedores podem se engajar focando em módulos específicos conforme vão ficando maduros (por exemplo, alguém pode contribuir com novos quizzes enquanto o time principal trabalha na análise de imagens).

## Organização do Repositório e Documentação

Para manter tudo **organizado e explicado**, a estrutura do repositório seguirá padrões claros, e cada parte terá documentação associada:

* cli\_app/ – Código fonte do CLI **ecgcourse**. Contém submódulos para comandos específicos (quiz, analyze, ingest, etc.). Inclui um README.md descrevendo o uso do CLI e exemplos de comandos.
* web\_app/ – Código do aplicativo web (Dash). Inclui o módulo principal dash\_app/app.py[[22]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L34-L41) e possíveis submódulos para layouts, callbacks e assets estáticos. Terá documentação de como rodar o servidor web e estrutura das páginas.
* cv/ – Módulo de visão computacional com scripts como deskew.py, normalize.py, segmentation.py etc., cada um responsável por uma etapa de processamento de imagem de ECG[[6]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L45-L53). Pode incluir também modelos de ML treinados ou utilidades para carregá-los. Incluiremos docstrings detalhadas e possivelmente um guia em docs/CV\_TOOLKIT.md explicando o pipeline de processamento de imagens.
* quiz/ – Diretório do motor de quizzes: contém bank/ (arquivos JSON das perguntas) e schema/ (esquema JSON para validá-las)[[7]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L55-L62), além de scripts utilitários (generate\_quiz.py, etc.). A documentação (ex: docs/QUIZ.md) detalhará o formato das questões, como adicionar novas e o processo de validação.
* notebooks/ – Coleção de Jupyter notebooks demonstrativos. Esses notebooks estarão organizados por tema (numerados ou nomeados conforme conteúdo, e.g., notebooks/01\_qtc\_calculation.ipynb). Manteremos um índice em docs/NOTEBOKS\_INDEX.md descrevendo cada notebook e sua finalidade educacional.
* docs/ – Documentação geral do projeto. Aqui residem arquivos como:
* ROADMAP.md – o planejamento de alto nível (fases p0–p30) com revisões conforme o projeto avança[[23]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L113-L120).
* GUIDE.md ou MANUAL.md – guia do usuário final (como usar o CLI e o web app, explicado passo a passo).
* DEV\_SETUP.md – instruções de configuração do ambiente de dev (virtual env, dependências) e dicas para novos contribuidores.
* STYLE\_GUIDE.md – guia de estilo de código e convenções do projeto.
* Qualquer documentação de arquitetura adicional (diagramas de módulos, por exemplo, em formato Markdown ou imagens exportadas de ferramentas de modelagem – se usarmos diagramas UML ou de fluxo, podemos incluí-los aqui).
* tests/ – Suíte de testes automatizados (por convenção, separados por subdiretórios espelhando a estrutura do código: testes para CLI, testes para CV, etc.). Incluir um README breve sobre como rodar os testes e interpretar resultados.
* Arquivos de configuração no root – por exemplo, requirements.txt (lista de dependências), pyproject.toml ou setup files se empacotarmos como biblioteca, .pre-commit-config.yaml (para hooks de lint/format), e possivelmente docker-compose.yml se for relevante para orquestrar serviços (ex.: rodar app + um contêiner de banco de dados se necessário no futuro).
* **Sub-repositório de Planejamento (opcional):** Caso o projeto cresça muito, poderemos adotar um submódulo ou pasta separada para planos e design (por exemplo, um diretório design/ contendo decisões de arquitetura registradas, histórico de propostas de mudança, etc.). Porém, inicialmente, centralizaremos tudo em docs/ para facilitar o acesso.

Toda mudança significativa no código deverá vir acompanhada de atualização na documentação pertinente. Para garantir isso, no fluxo de CI poderemos adicionar verificações que alertem se, por exemplo, o schema JSON do quiz mudar mas o guia de quiz não for atualizado. Além disso, usaremos nosso **agente CLI de IA** para ajudar a manter a documentação sincronizada – por exemplo, pedir ao Gemini CLI para gerar um resumo das mudanças de um PR e sugerir trechos a acrescentar no CHANGELOG ou nos docs, que depois revisamos e aprovamos[[24]](https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md#L6-L14)[[25]](https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md#L24-L32). A ideia é que o repositório em si seja um **retrato vivo e auto-explicativo** do projeto: um novo colaborador deve conseguir entender a estrutura e configurar o ambiente seguindo os arquivos de orientação, e um usuário final deve achar instruções claras de uso.

## Considerações Finais

Com este plano robusto, cobrimos do planejamento à implantação do ECGiga de forma holística. Usando plenamente as ferramentas modernas (integração de AI no desenvolvimento, pipelines de CI/CD, boas práticas de modularização), garantimos que cada etapa – arquitetura, engenharia, codificação, criação de conteúdo e deploy – seja executada com qualidade e eficiência. Importante reforçar que embora disponhamos de CLIs inteligentes para ajudar a **"gerenciar o projeto diretamente do terminal"**[[14]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=inside%20editors%20by%20offering%20smart,project%20directly%20from%20the%20terminal), a supervisão humana e validação contínua são fundamentais, principalmente em um domínio sensível como saúde.

Ao seguir as fases delineadas alinhadas ao roadmap original[[16]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L10), vamos gradativamente construindo uma plataforma educacional de ECG de alto nível. A flexibilidade do plano permite ajustes conforme feedback dos usuários e avanços tecnológicos (por exemplo, se o modelo *Gemini* evoluir, poderemos incorporá-lo para aprimorar os laudos automáticos). Em suma, o projeto será conduzido de forma iterativa, documentada e aberta a contribuições, resultando em um repositório exemplar – bem organizado, explicado e capaz de entregar um produto final confiável. Com arquitetura sólida e um ciclo de desenvolvimento alimentado tanto pela expertise da equipe quanto pela colaboração com ferramentas de IA, o ECGiga tem todos os ingredientes para se tornar uma referência em aprendizado de ECG **open-source** e de alta qualidade.

**Fontes Utilizadas:**

* Documentação do projeto ECGiga (contexto e roadmap)[[1]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L6-L14)[[16]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L10)
* Artigo comparativo de ferramentas CLI de IA (Codex CLI vs Gemini CLI)[[12]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Editor%20tools%20assist%20with%20small,edits%20or%20writing%20functions)[[11]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Workflow%20Integration)
* Instruções e objetivos de laboratórios práticos relacionados (Gemini CLI, Codex CLI)[[24]](https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md#L6-L14)[[25]](https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md#L24-L32) (forneceram insights sobre geração de planos incrementais e uso de IA para melhorias de código/documentação).

[[1]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L6-L14) [[2]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L8-L12) [[3]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L12-L15) [[4]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L23-L31) [[5]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L36-L41) [[6]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L45-L53) [[7]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L55-L62) [[8]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L63-L67) [[9]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L107-L114) [[10]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L105-L109) [[22]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L34-L41) [[23]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md#L113-L120) gemini.md

<https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/gemini.md>

[[11]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Workflow%20Integration) [[12]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Editor%20tools%20assist%20with%20small,edits%20or%20writing%20functions) [[13]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=Try%20commands%20like%3A) [[14]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=inside%20editors%20by%20offering%20smart,project%20directly%20from%20the%20terminal) [[15]](https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025#:~:text=AI%20CLIs%20often%20process%20much,just%20one%20function%20or%20file) Claude Code CLI vs Codex CLI vs Gemini CLI: Best AI CLI Tool for Developers in 2025?

<https://www.codeant.ai/blogs/claude-code-cli-vs-codex-cli-vs-gemini-cli-best-ai-cli-tool-for-developers-in-2025>

[[16]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L10) [[17]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L3-L5) [[18]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L4-L6) [[19]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L6-L8) [[20]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L7-L9) [[21]](https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md#L7-L10) roadmap.md

<https://github.com/Drmcoelho/ECGiga/blob/f087a2cd5d2cd1d31eff6b498146771c0dd071cf/src_p11/docs/roadmap.md>

[[24]](https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md#L6-L14) [[25]](https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md#L24-L32) README.md

<https://github.com/Drmcoelho/Codespace/blob/f2e173ad0479fe9f3e64a4ad82ab4a4bb5e2e726/labs/lab37_gemini_cli_agent/README.md>