# ECGiga Construction and Deployment Plan

## 1. Project overview

**ECGiga** is an ambitious initiative to build an open‑source electrocardiography (ECG) course that combines a command‑line interface (CLI), an interactive web application, computer‑vision (CV) tools and AI‑assisted features. The repository is organised into modules such as a CLI package (ecgcourse), a Dash‑based web app, a quiz engine with a large bank of multiple‑choice questions (MCQs), scripts for computer‑vision processing, data manifests and prompts for future large language‑model (LLM) integration. The README in the root of the project describes the project’s purpose and architecture, emphasising that the course is aimed at healthcare professionals and will deliver accurate clinical content, heavy interactivity, a quiz engine and AI assistance[[1]](https://www.datacamp.com/tutorial/pydantic#:~:text=Pydantic%20is%20Python%E2%80%99s%20most%20popular,hints%20into%20runtime%20validation%20rules).

The project is being developed in ~30 parts. Parts **0–7** are already delivered and provide the skeleton structure, initial CLI and web app stubs, quiz schemas, and progressively more advanced CV pipelines (deskewing, normalisation, layout detection and R‑peak identification). Future parts will expand the quiz bank, add algorithms to estimate ECG intervals, integrate LLM‑based assistance, and eventually package the application for desktop/mobile use. The roadmap file docs/roadmap.md (not fully included here) details acceptance criteria for each part.

## 2. Architecture and key modules

| Component | Purpose | Key files/notes |
| --- | --- | --- |
| **CLI (cli\_app/ → ecgcourse)** | Provides commands for running quizzes, validating quiz files, analysing structured ECG measurements, ingesting ECG images, and running CV utilities. Built with **Typer** and **Rich** to support commands, arguments and options[[2]](https://realpython.com/python-typer-cli/#:~:text=In%20this%20tutorial%2C%20you%E2%80%99ll%20learn,how%20to). Pydantic models are used internally for data validation; Pydantic turns Python type hints into runtime validation rules and automatically validates and converts input data[[1]](https://www.datacamp.com/tutorial/pydantic#:~:text=Pydantic%20is%20Python%E2%80%99s%20most%20popular,hints%20into%20runtime%20validation%20rules). | cli.py, quiz\_engine.py, CV subcommands, etc. |
| **Web application (web\_app/dash\_app/app.py)** | An interactive dashboard built with Dash. Provides 12‑lead ECG visualisation, calculators (e.g., QTc), image upload and later simulation sliders. When deploying a Dash app, debug mode should be turned off (debug=False) and a server = app.server line should be declared so a WSGI server such as Gunicorn can pick up the Flask server[[3]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5). | app.py, assets folder, future multi‑page and simulation components. |
| **Computer‑vision tools (cv/)** | Scripts to deskew ECG images, normalise scale, detect grid and layout, perform OCR on lead labels, estimate R‑peaks (basic and robust Pan–Tompkins‑style algorithms) and calculate intervals (PR, QRS, QT, QTc). The Pan–Tompkins algorithm works by filtering the ECG signal to enhance the QRS complex, squaring the signal and using adaptive thresholds to detect peaks[[4]](https://en.wikipedia.org/wiki/Pan%E2%80%93Tompkins_algorithm#:~:text=The%20Pan%E2%80%93Tompkins%20algorithm,S%20wave). |  |
| **Quiz engine (quiz/)** | A JSON‑schema–validated quiz bank. MCQ files include the question stem, options, correct answer index and a detailed clinical explanation. The CLI can validate these files and run quizzes. |  |
| **Data assets (assets/)** | Manifests of licensed ECG images and datasets from open sources (Wikimedia, PhysioNet) and scripts to download, verify licences and pre‑process images (convert to WebP/AVIF, generate multiple sizes). |  |
| **Notebooks (notebooks/)** | Educational Jupyter notebooks that demonstrate algorithms (R‑peak detection, QTc formulas, axis calculation), signal processing and simulation of ionic changes. |  |
| **AI prompts (models/prompts/)** | Initial prompts for future GPT‑5–based lauding and tutoring features. These are currently stubs; integration will be implemented in later parts. |  |

## 3. Development environment and tools

### 3.1 Creating the environment

1. **Clone the repository and create a virtual environment:**

* git clone <URL\_DO\_REPO> ecg‑megaprojeto  
  cd ecg‑megaprojeto  
  python3 -m venv .venv  
  source .venv/bin/activate # On Windows: .venv\Scripts\activate

1. **Install dependencies:** The project uses pyproject.toml and requirements.txt. Install the dependencies using pip:

* pip install -r requirements.txt
* Additional development tools (Black, Ruff, pytest) are listed for code formatting and testing.

1. **Run the CLI:** Typer makes it easy to add commands, arguments and options to a CLI[[2]](https://realpython.com/python-typer-cli/#:~:text=In%20this%20tutorial%2C%20you%E2%80%99ll%20learn,how%20to). After installation, run:

* python -m ecgcourse --help
* to see available commands. Example: ecgcourse quiz validate quiz/bank/exemplo\_arrtimias.json to validate an MCQ file.

1. **Run the Dash app locally:**

* python web\_app/dash\_app/app.py
* This should start a local server at http://127.0.0.1:8050 for the interactive dashboard.

### 3.2 Data validation

Pydantic is the core data validation library used in this project. It converts Python type hints into runtime validation rules and automatically validates incoming data, converts types when appropriate and provides clear error messages[[1]](https://www.datacamp.com/tutorial/pydantic#:~:text=Pydantic%20is%20Python%E2%80%99s%20most%20popular,hints%20into%20runtime%20validation%20rules). Using Pydantic models ensures that command inputs (e.g., PR, QRS, QT intervals) and quiz structures conform to expected formats. Pydantic also generates JSON Schema automatically, enabling consistent validation across the CLI and web.

### 3.3 Computer‑vision algorithms

The project’s CV pipeline uses open‑source libraries (NumPy, SciPy, OpenCV, pillow) to process ECG images. The Pan–Tompkins algorithm is used to detect QRS complexes by filtering the ECG signal, squaring it to enhance peaks and applying adaptive thresholds[[4]](https://en.wikipedia.org/wiki/Pan%E2%80%93Tompkins_algorithm#:~:text=The%20Pan%E2%80%93Tompkins%20algorithm,S%20wave). Subsequent parts implement deskewing, normalisation (scale to mm/mV), layout segmentation (e.g., 3×4 vs. 6×2), OCR of lead labels, robust R‑peak detection and interval estimation. These algorithms may require platform‑specific dependencies (e.g., Tesseract for OCR) that are listed in requirements.txt.

## 4. Local development & quick sharing

For rapid collaboration or demonstrations, a Cloudflare Tunnel can expose the local Dash server to the internet without configuring DNS. According to Cloudflare’s **TryCloudflare** documentation, the tool launches a process that generates a random subdomain on trycloudflare.com; requests to that subdomain are proxied through Cloudflare’s network to the local web server[[5]](https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/do-more-with-tunnels/trycloudflare/#:~:text=Quick%20Tunnels). To use it:

# From the project root (where serve\_and\_share\_now.sh is located)  
chmod +x serve\_and\_share\_now.sh  
./serve\_and\_share\_now.sh

This script installs minimal dependencies (if necessary), starts the local server on the specified port and invokes cloudflared tunnel --url http://127.0.0.1:<PORT>. The terminal will display a URL like https://random-subdomain.trycloudflare.com. Note that Quick Tunnels are intended for testing and development; Cloudflare warns that they are subject to limits (e.g., 200 concurrent requests) and do not guarantee uptime[[6]](https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/do-more-with-tunnels/trycloudflare/#:~:text=,Sent%20Events%20%28SSE).

## 5. Permanent deployment strategies

### 5.1 Deploy to GitHub Pages

The repository includes a workflow (.github/workflows/pages\_with\_data.yml) and a helper script deploy\_github\_pages.sh for deploying the static portion of the project (the web\_app/ folder and reports/datasets) to GitHub Pages. The recommended steps are:

1. **Authenticate the GitHub CLI:**

* gh auth login # follow the prompts to authenticate

1. **Create or initialise a Git repository:** If the project is not yet versioned, run git init, add files and commit.
2. **Create a remote repository with the GitHub CLI:** According to the GH CLI manual, gh repo create <name> --public --source=. --remote=origin will create a new repository from the current directory. Using the --push flag pushes existing commits to the remote repository[[7]](https://cli.github.com/manual/gh_repo_create#:~:text=To%20create%20a%20remote%20repository,clone%20the%20new%20repository%20locally).
3. **Run the deployment script:**

* chmod +x deploy\_github\_pages.sh  
  ./deploy\_github\_pages.sh ECGiga # pass your repository name
* The script will create the repository if it does not exist, push the code and rely on the GitHub Action to publish the site. After the workflow finishes, the Dash app will be accessible at https://<username>.github.io/ECGiga/web\_app/.

GitHub Pages is suitable for the static Dash assets (the front‑end files). For dynamic Dash callbacks or server‑side processing, you will still need a Python server; thus, this approach is mainly for static sites or pre‑rendered outputs.

### 5.2 Deploy to a platform that supports Python (Render/Heroku)

For a fully interactive Dash application with server‑side callbacks, a platform such as **Render**, **Heroku** or **AWS Elastic Beanstalk** can host the Python backend. The Dash tutorial for deployment outlines several important steps:

* **Turn off debug mode:** In the main app file, run the Dash app with debug=False to disable hot reloading[[8]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5).
* **Declare a WSGI server variable:** Add server = app.server after creating the Dash app so that Gunicorn can locate the Flask server[[9]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5).
* **Provide a requirements.txt file:** List all Python dependencies (and gunicorn) in a requirements file. The deployment platform uses this file to install packages[[10]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5).
* **Use Gunicorn as the entry point:** On Render/Heroku, specify a start command like gunicorn app:server (replace app with your filename) so that the platform runs the Flask server[[11]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=Then%2C%20in%20the%20%E2%80%9CStart%20Command%E2%80%9D,Let%E2%80%99s%20break%20down%20this%20command).

Render’s free tier is typically sufficient for small dashboards. For Heroku, you would create a Procfile containing web: gunicorn app:server and enable the Heroku Dyno. Both platforms integrate with GitHub; commit and push to the linked repository triggers redeployment.

### 5.3 Containerised deployment (optional)

For portability and reproducibility, containerise the application using Docker. A Dockerfile can install dependencies, copy project files and run the Dash server. Containerised images can then be deployed to platforms like Google Cloud Run, AWS Fargate or Azure Container Apps.

## 6. Implementation roadmap

### 6.1 Completed phases (p0–p7)

The index summarises features delivered in the first eight parts:

* **p0:** Repository skeleton with CLI/Web stubs, quiz schema and notebooks. Basic CLI runs MCQs; Dash shows a “hello dashboard”.
* **p1:** Full quiz CLI with local performance reports; static 12‑lead display with zoom.
* **p2:** CLI to analyse structured ECG values (PR, QRS, QT, RR, axis); interactive QTc calculator in Dash; additional notebooks and ~150 MCQs.
* **p3:** Ingest ECG images with optional sidecar meta files; generate standardised ECG reports (schema v0.1); upload interface in Dash; sample synthetic images; added notebooks and ~60 new MCQs.
* **p3b/c:** Manifest of open‑source ECG images/datasets; download/verify licences; pre‑process images (WebP/AVIF, multiple sizes); generate derived manifests.
* **p5:** CV tools for deskewing, normalisation and layout segmentation; CLI flags for processing pipeline; Dash controls for deskew/normalise.
* **p6:** Automatic lead‑label detection via template matching/OCR; layout detection (3×4, 6×2, rhythm); initial R‑peak detection from image.
* **p7:** Robust R‑peak detection (Pan–Tompkins‑like) and interval estimation (PR, QRS, QT, QTc); new report schema (v0.4); dynamic quiz generation from reports.

### 6.2 Proposed future phases (p8–p30)

Based on the repository’s roadmap and gaps in current implementation, the following timeline is proposed:

1. **p8:** Expand the quiz bank to >500 items, adding modules on advanced arrhythmias and ECG pitfalls. Include spaced‑repetition features and track user performance over time. Continue to refine the CLI and Dash interfaces to allow category‑based quizzes.
2. **p9:** Integrate robust axis calculation and automatic detection of conduction delays (e.g., bundle branch block). Provide interactive visualisations of axis on a hexaxial reference circle.
3. **p10:** Add support for multi‑page Dash apps (e.g., separate tabs for analysis, quizzes, simulation). Implement user authentication (via GitHub or email) to save progress. Provide an API using FastAPI for external integrations.
4. **p11:** Enhance computer‑vision modules with deep‑learning models for segmentation and R‑peak detection (e.g., U‑Net or wavelet‑based methods), leveraging frameworks like TensorFlow/PyTorch. Provide a training pipeline using open datasets from PhysioNet.
5. **p12:** Integrate LLM‑based assistance for preliminary ECG reports and tutoring. Follow medical AI guidelines: LLMs should not replace clinicians; they sometimes hallucinate and lack uncertainty estimates[[12]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Following%20the%20release%20of%20ChatGPT%2C,22%2C3). Use prompts stored in models/prompts/ and require human verification of AI‑generated reports. Provide transparent explanations and highlight uncertain statements.
6. **p13:** Build physiological simulations: dynamic sliders controlling ionic concentrations (K+, Ca²⁺, Na⁺) and heart rate to show changes in action potentials and ECG morphology.
7. **p14–p20:** Develop offline mode and packaging: a desktop app (using PyInstaller or Electron wrapper) and a Progressive Web App (PWA) for mobile. Offer optional account features (e.g., saving reports) while complying with data‑privacy regulations.
8. **p21–p30:** Conduct extensive user testing with healthcare professionals. Incorporate feedback, refine content for clarity and clinical accuracy, and add advanced modules (e.g., pacemaker interpretation, device troubleshooting, rare channelopathies). Publish final course content under open licences (MIT for code, CC‑BY‑SA for content) and ensure long‑term maintenance.

## 7. Considerations for AI/LLM integration

Large language models such as GPT‑4 and GPT‑5 offer promising capabilities for assisting with ECG interpretation and tutoring. However, researchers note that current models frequently hallucinate and provide false or unverified information; they lack uncertainty indicators and are not ready for autonomous deployment in clinical settings[[13]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Following%20the%20release%20of%20ChatGPT%2C,22%2C3). Medical AI tools are most appropriate for administrative tasks, rephrasing content or providing preliminary guidance; they **cannot replace clinicians**[[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Despite%20these%20fears%20and%20hype%2C,practical%20tasks%20undertaken%20in%20healthcare). When integrating LLMs into ECGiga:

* Label the AI outputs as *draft* or *preliminary* and require human verification before acting on any recommendation.
* Use structured prompts that explicitly request the model to cite guidelines or known sources, and cross‑check the results.
* Store prompts and model responses in version‑controlled files for auditability.
* Provide users with the ability to flag incorrect AI outputs for correction in future iterations.

## 8. Conclusion & next steps

The ECGiga project lays a solid foundation for an innovative, open‑source ECG course that combines command‑line tools, interactive web dashboards, computer‑vision algorithms and, eventually, AI assistance. To move from prototype to a fully deployed educational platform, the team should:

1. Finalise the outstanding features in p0–p7 (polishing the CLI, expanding the quiz bank, improving CV accuracy).
2. Implement the future parts sequentially, prioritising features that add the most educational value (e.g., robust axis calculation, advanced quizzes, and dynamic simulations).
3. Set up permanent deployment pipelines using GitHub Pages for static assets and a PaaS (Render/Heroku) for the interactive Dash app; include appropriate deployment settings such as turning off debug mode and declaring the server[[3]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5).
4. Begin integration of LLMs cautiously, always following ethical guidelines that emphasise human oversight and make clear that AI outputs do not substitute for professional clinical judgment[[12]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Following%20the%20release%20of%20ChatGPT%2C,22%2C3)[[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Despite%20these%20fears%20and%20hype%2C,practical%20tasks%20undertaken%20in%20healthcare).

By following this roadmap and adhering to best practices in software development and medical AI ethics, ECGiga can become a comprehensive and trusted resource for clinicians and trainees around the world.

[[1]](https://www.datacamp.com/tutorial/pydantic" \l ":~:text=Pydantic%20is%20Python%E2%80%99s%20most%20popular,hints%20into%20runtime%20validation%20rules) Pydantic: A Guide With Practical Examples | DataCamp

<https://www.datacamp.com/tutorial/pydantic>

[[2]](https://realpython.com/python-typer-cli/#:~:text=In%20this%20tutorial%2C%20you%E2%80%99ll%20learn,how%20to) Build a Command-Line To-Do App With Python and Typer – Real Python

<https://realpython.com/python-typer-cli/>

[[3]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5) [[8]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5) [[9]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5) [[10]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=5) [[11]](https://open-resources.github.io/dash_curriculum/part1/chapter5.html#:~:text=Then%2C%20in%20the%20%E2%80%9CStart%20Command%E2%80%9D,Let%E2%80%99s%20break%20down%20this%20command) Chapter 5 - App Deployment — Dash Tutorial

<https://open-resources.github.io/dash_curriculum/part1/chapter5.html>

[[4]](https://en.wikipedia.org/wiki/Pan%E2%80%93Tompkins_algorithm#:~:text=The%20Pan%E2%80%93Tompkins%20algorithm,S%20wave) Pan–Tompkins algorithm - Wikipedia

<https://en.wikipedia.org/wiki/Pan%E2%80%93Tompkins_algorithm>

[[5]](https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/do-more-with-tunnels/trycloudflare/#:~:text=Quick%20Tunnels) [[6]](https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/do-more-with-tunnels/trycloudflare/#:~:text=,Sent%20Events%20%28SSE) Quick Tunnels · Cloudflare Zero Trust docs

<https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/do-more-with-tunnels/trycloudflare/>

[[7]](https://cli.github.com/manual/gh_repo_create#:~:text=To%20create%20a%20remote%20repository,clone%20the%20new%20repository%20locally) GitHub CLI | Take GitHub to the command line

<https://cli.github.com/manual/gh_repo_create>

[[12]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Following%20the%20release%20of%20ChatGPT%2C,22%2C3) [[13]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Following%20the%20release%20of%20ChatGPT%2C,22%2C3) [[14]](https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/#:~:text=Despite%20these%20fears%20and%20hype%2C,practical%20tasks%20undertaken%20in%20healthcare) Large language models will not replace healthcare professionals: curbing popular fears and hype - PMC

<https://pmc.ncbi.nlm.nih.gov/articles/PMC10331084/>