Project Planning Document

Project: MCP Agents Laravel UI Generator
Sponsor: Internal Development Team
Project Manager: Fikri Armia Fahmi (2023071018)
Frontend Developer: **Nadia (2024071004)**
Tanggal: 18 September 2025

---

1. Project Initiation

1.1 Latar Belakang

Bagi mahasiswa maupun developer pemula, pengembangan komponen UI di Laravel seringkali memakan waktu karena harus membuat **Blade components**, **routes**, dan melakukan **validasi manual** satu per satu. Proses ini tidak hanya teknis, tapi juga menyita banyak waktu belajar. Dengan hadirnya **AI multi-agent system**, proses tersebut dapat dipercepat melalui otomatisasi sehingga mahasiswa dapat lebih fokus pada **logika aplikasi, inovasi, dan penelitian**, bukan hanya pada pekerjaan repetitif.

1.2 Tujuan

- Mempercepat pembuatan Laravel UI dari natural language prompts.
- Menyediakan preview interaktif sebelum integrasi.
- Meningkatkan kualitas kode dengan sistem validasi otomatis.

1.3 Stakeholder

- **Sponsor:** Internal Research & Development Team
- **Developer:** AI/ML Engineers, Laravel Developers, Frontend Developer
- **User:** Laravel developers & software houses

---

2. Feasibility Study

Aspek	Evaluasi	Detail
Teknis	✅ Layak	Python 3.7+, Laravel 8+, Cerebras SDK, Mistral SDK
Ekonomi	✅ Layak	Open-source MIT License, biaya server minimal
Operasional	✅ Layak	Developer dapat menjalankan script Python, integrasi Laravel
Hukum	✅ Layak	MIT License, sesuai standar open-source
Waktu	✅ Layak	Estimasi 12–16 minggu development agile

3. Project Planning (updated)

3.1 Information (basic)

- **Project Name:** MCP Agents Laravel UI Generator
- **Primary Goal:** Automate Laravel UI creation from natural-language prompts using an AI multi-agent system.
- **Main Deliverables:**
    - AI agent modules (10 agents)
    - Laravel UI generator (Blade files, layouts, components)
    - Validation & integration system
    - Front-end preview UI (HTML/CSS/JS)
    - Agent backend services & API
    - Documentation & user guide

3.2 Scope & Deliverables (expanded)

1. **Prompt Processing Agent** — expand & plan user prompts
2. **Draft Generator** — generate HTML preview (front-end + backend integration)
3. **Layout & Component Generator** — Blade files, layouts, components
4. **Route Generator** — update routes/web.php and scaffolding
5. **Validator** — syntax, structure, linting, acceptance criteria checks
6. **Project Integrator** — move validated files into Laravel project
7. **Documentation** — README, Wiki, tutorials
8. **Front-end Preview UI** — static + interactive preview built with HTML / CSS / JS
9. **Agent Backend Services** — API endpoints, orchestration, queueing, logs

3.3 Development Approach & Lifecycle

- **Method:** Agile — 1 week / sprint, iterative increments, weekly demo
- **Tools / Tech Stack:**
    - Front-end: HTML5, CSS3 (Tailwind or plain), Vanilla JS or small framework (Alpine.js / HTMX) for interactivity
    - Backend agents/orchestrator: Python (existing agents), FastAPI or Flask for HTTP endpoints, Redis/RQ or Celery for queues
    - Integration: Laravel (target project), filesystem staging area, GitHub Actions for CI

    - AI: Mistral API (agent prompts & inference)
- **Testing:** Unit tests for agents, integration tests for file move & route updates, E2E for preview pipeline
- **CI/CD:** GitHub Actions for linting, tests, and optional deployment of preview server

3.4 Timeline (sprint mapping) — total ≈ 14 weeks

- **Sprint 1–2:** Project setup, infra, repo templates, env, CI pipelines
- **Sprint 3–4:** Prompt Expander & Planner agent (backend)
- **Sprint 5–6:** Draft Generator core + simple HTML preview (front-end scaffold)
- **Sprint 7–8:** Layout & Component Generator (agent backend) + preview UI enhancements (HTML/CSS/JS)
- **Sprint 9–10:** Route Agent & Project Integrator (integration with Laravel)
- **Sprint 11:** Validator Agent (syntax, structure, acceptance criteria)
- **Sprint 12:** Agent backend polishing — queueing, retries, logging, security
- **Sprint 13:** Documentation, examples, repo templates, demo app
- **Sprint 14:** Final testing, bugfix, release / handover

Note: Sprints above are 1-week units; you can split or merge according to team velocity.

3.5 Resource Planning (updated)

- **Team**
    - 1 Project Manager + Backend Engineer (agent API, orchestration)
    - 1 Frontend Engineer (HTML/CSS/JS)
- **Tools:** Python, FastAPI/Flask, Redis/Celery, Laravel, GitHub, Mistral API, Docker (optional)

3.6 Front-end Plan (HTML / CSS / JS) — deliverables & responsibilities

- **Goal:** provide interactive preview where user can see generated HTML drafts and trigger generation into Laravel.
- **Deliverables**
    - Lightweight preview page (served from output/ or small preview server)
    - Components: preview pane, prompt input, agent status panel, accept/modify controls, download / integrate buttons
    - Accessibility basics and responsive layout
- **Tech details**

    - Static markup generated by Draft Generator (server) and rendered in preview pane
    - Interactions via vanilla JS or Alpine.js: accept changes, open code editor modal, copy to clipboard, send approve-to-integrate request
    - CSS: Tailwind (if allowed) or scoped CSS; keep styling minimal, focus on clarity
- **Testing**
    - Cross-browser basic tests (Chrome, Firefox)
    - Visual inspection acceptance per sprint
- **Security**
    - Sanitize previewed HTML (avoid executing untrusted scripts) — render in sanitized iframe or text renderer

3.7 Agent Backend Plan — architecture & responsibilities

- **Responsibilities**
    - Orchestrate agents in sequence: prompt expander → draft → planner → generator → validator → integrator
    - Expose HTTP endpoints for front-end to request draft, check status, and trigger integration
    - Manage asynchronous jobs (queue), retries, and error handling
- **Architecture**
    - **API Layer:** FastAPI/Flask — endpoints: /generate-draft, /status/{job}, /approve/{job}, /integrate/{job}
    - **Worker Layer:** Python workers (Celery/Redis or RQ) to run agents (calls to Mistral + codegen)
    - **Storage:** staging filesystem (output/), DB-lite (sqlite/postgres) for job metadata and logs
    - **Logging & Monitoring:** structured logs, simple dashboard for job queue / failures
    - **Security:** API keys for access, rate-limits, input validation, secrets management via .env
- **Integration points**
    - Mistral AI API (agent inference)
    - Laravel project (move files via staging or PR creation)
    - GitHub (optional): create branch / PR when integrating into my-laravel/

3.8 Risk & Mitigation (extended)

Risk	Impact	Mitigation
Untrusted HTML executes in preview	High	Render in sanitized iframe/escape scripts; do not execute user-generated JS
Agent output incorrect (invalid Blade)	High	Validator agent, linting, unit tests, sample acceptance criteria
Queue/backlog overload	Medium	Rate-limit requests, scale workers, backpressure to UI
Mistral downtime	Medium	Exponential retry, fallback cached templates, graceful degrade message
Integrating to user Laravel breaks project	High	Create branch or staging folder instead of directly overwriting; provide rollback

3.9 Stakeholder Engagement & Communication

- Weekly sprint review + demo of preview UI and a sample generated Blade file
- Issues & backlog management on GitHub (issues per sprint)
- Progress via GitHub Project Board; monthly stakeholder report

3.10 Success Criteria (updated)

- ≥ 90% successful generation of syntactically valid Blade components (per validator)
- Preview UI allows users to inspect and approve generated templates reliably
- Integration process updates Laravel project without breaking (use PR/staging)
- Internal developer team adopts tool for at least 3 example pages within first month
- Complete documentation & example repository available