LIVE MASTERCLASS TEACHING DECK

Production Ready Architecture

AI Website & Software Development

Learn to Build Commercial Websites, Web Applications, SaaS Platforms and Custom Software Using AI-Powered Engineering Workflows

MASTERCLASS INSTRUCTOR

5+ YRS EXP

Laba Das

WordPress Developer, Vibe Coder & Founder of Webpress Hubs

Creator of **AI Blog Engine Plugin** and specialized in building commercial digital products, SaaS platforms, and AI workflows.

Goal: Turn any software concept into a live, tested, commercial product independently.

Modern Stack & Toolchain

- PHP & MySQL
- HTML5 & CSS3
- JavaScript
- Antigravity / Cursor
- ChatGPT / Claude
- XAMPP & Hostinger

Engineering First: Master repeatable planning, QA, and deployment systems rather than fragile single prompts.

What You Will Learn in This Class

A complete, end-to-end framework for turning raw ideas into production-ready software.

1. Blueprint Specs

PRD creation, agent.md rules, and phase plans.

2. AI IDE Execution

Orchestrating Antigravity & Cursor agents safely.

3. QA & Debugging

Local XAMPP server testing & structured bug fixing.

4. UI & Deployment

Reference design polish & live Hostinger launch.

Key Takeaway: Students will learn a repeatable, enterprise-grade development system that works across any language, framework, or AI model generation.

What We Can Build Using This Process

The exact same engineering process applies to simple websites and complex web software.

Websites & Portals

- ✓ High-converting Landing Pages
- ✓ Corporate Business Websites
- ✓ Blogs & News Platforms
- ✓ Membership & Client Portals

SaaS & Custom Apps

- ✓ Admin Dashboards & Analytics
- ✓ Lead Management & CRM Systems
- ✓ Online Booking & Scheduling
- ✓ Multi-Tenant SaaS Platforms

Plugins & Utilities

- ✓ WordPress Plugins & Custom Themes
- ✓ PHP Scripts & REST API Services
- ✓ AI-Powered Web Utilities
- ✓ E-Commerce Modules

Rule of Scale: The architecture remains identical; only the detail level of documentation expands with project complexity.

About Your Instructor & Class Goals

Meet Laba Das and review our 8 practical learning objectives for today's masterclass.

Laba Das

5 Yrs Exp

24-Year-Old WordPress Developer & Vibe Coder

Founder, Webpress Hubs

Creator, AI Blog Engine Plugin

Theme & Plugin Dev

Specialized in building commercial digital products, SaaS platforms, custom WordPress plugins, and practical AI coding automation systems.

> *"My goal is not merely to show code generation, but to teach a systematic, repeatable workflow that produces reliable software every time."*

Today's 8 Learning Milestones

1. **Tool Ecosystem:** Chat vs IDE vs No-Code
2. **Planning First:** PRD & agent.md specs
3. **Controlled Code:** Phased development
4. **Directory Maps:** file_structure.md tree
5. **Local Testing:** XAMPP PHP/MySQL QA
6. **Pro Debugging:** Structured bug reports
7. **UI Polish:** Reference site analysis
8. **Deployment:** Hostinger production launch

Masterclass Teaching Roadmap & Stages

The structured, step-by-step development pipeline we follow from raw concept to live production launch.

1. Core Foundations

STAGES 1–3

Understand AI coding capabilities, limitations, human engineering responsibilities, and tool ecosystems.

2. Blueprint Specs

STAGES 4–6

Draft master PRD specs, set agent.md coding rules, enforce light theme (#009E8C) and tech stack limits.

3. Phasing & Maps

STAGES 7–10

Divide execution into testable phases, map file_structure.md trees, and configure post-v1 update.md logs.

4. IDE & Local QA

STAGES 11–13

Upload blueprints to Antigravity IDE, build one phase at a time, test on XAMPP local PHP/MySQL server.

5. UI Polish & Launch

STAGES 14–17

Refine UI using Stripe/Linear reference analysis, image-to-code workflows, and deploy live to Hostinger.

6. Fast-Track & Wrap

STAGES 18–19

Fast-track simple landing pages with AI app builders, manage Node.js exports, review Q&A & next steps.

Core Philosophy: Following this exact 19-stage pipeline eliminates fragmented builds, broken database columns, and missing code handlers.

SECTION 01

Introduction to AI Software Development

Understanding how AI assists development, where it performs exceptionally well, where it fails, and why human engineering guidance remains essential.

What AI Means in Modern Software Dev

AI models interpret natural language to generate, refactor, and manipulate code structure.

What AI Can Do Best

- ✓ Requirement Analysis & PRD drafting
- ✓ Boilerplate code & database schemas
- ✓ UI component layout & styling
- ✓ Automated debugging & error explanations

AI Limitations

- ✓ Lacks independent business context awareness
- ✓ Hallucinates non-existent functions if unguided
- ✓ Capped by provided context & instructions
- ✓ Cannot replace human testing & verification

1 Human Idea → 2 Structured Docs → 3 AI Generation → 4 Human Testing → 5 Deployed App

How AI Transforms Development Roles

AI accelerates execution speed, but humans remain responsible for strategy and validation.

Where AI Performs Best

- ✓ Writing CRUD functions & SQL schemas
- ✓ Converting design screenshots into HTML/CSS
- ✓ Updating multiple related files simultaneously
- ✓ Explaining complex error logs in plain text

Required Human Responsibilities

- ✓ Defining real-world business problem & scope
- ✓ Verifying security, permissions & authentication
- ✓ Conducting manual functional workflow testing
- ✓ Managing API keys, passwords & hosting servers

Golden Rule: AI increases speed by 10×, but human oversight guarantees that the software actually works correctly for real users.

Common Misunderstandings About AI Coding

Avoid these dangerous assumptions when building commercial software with AI.

Myth 1: One Prompt App

Believing a single prompt can generate a flawless production app. Real software requires phased specifications.

Myth 2: Auto Security

Assuming AI code is automatically protected against SQL injection, XSS attacks, and broken auth session guards.

Myth 3: Pretty UI = Works

Mistaking an attractive frontend layout for a functional, bug-free backend data connection and database schema.

Engineering Reality: Professional software principles—requirements, documentation, staged builds, QA testing, and debugging—are still 100% mandatory.

SECTION 02

Choosing the Right AI Tool Ecosystem

A structured comparison of chat models, AI-powered development environments (IDEs), coding agents, and rapid no-code app-building platforms.

The Three AI Tool Categories

Understand which tool category to use for each step of the development process.

Cat 1: Chat Models

Planning, PRDs, research, code review & bug cleanup.

- ChatGPT
- Claude
- Gemini
- DeepSeek

Cat 2: AI IDEs & Agents

Reading project files, multi-file execution & coding.

- Antigravity
- Cursor
- Windsurf
- Claude Code

Cat 3: No-Code Builders

Rapid visual MVPs, simple landing pages & prototypes.

- Lovable
- Bolt.new
- v0
- Replit

Selection Rule: Match the tool category to the required code control and project complexity.

Category 1: Chat Models Overview

Ideal for high-level reasoning, planning documents, and structured bug analysis.

Primary Supported Tools

- ChatGPT-4o
- Claude 3.5 Sonnet
- Gemini 1.5 Pro
- Perplexity AI
- DeepSeek R1

Key Strengths

Deep reasoning, drafting 5,000-word PRDs, synthesizing raw testing notes into structured bug reports, and explaining architectural concepts.

Critical Limitation

Chat models are excellent for **thinking, planning, and preparing instructions**, but they are NOT built to manage, open, and edit complex multi-file codebase projects directly in real-time.

Category 2: AI IDEs & Coding Agents

Primary development environments designed for multi-file software engineering.

Supported IDE Ecosystem

Antigravity | Cursor | GitHub Copilot | Windsurf | Claude Code

What AI IDEs Accomplish

- ✓ Parse complete project folder trees & file relationships
- ✓ Create new files & edit existing code across folders
- ✓ Execute terminal commands & run automated tests
- ✓ Follow strict `agent.md` guidelines across builds

Teaching Choice

For today's main workflow, **Antigravity IDE** will serve as our primary development engine. Students can seamlessly mirror all steps using Cursor or Windsurf.

Category 3: No-Code & Low-Code App Builders

Rapid autonomous generation platforms for landing pages and simple web MVPs.

Key Advantages

- ✓ Extremely fast initial visual generation
- ✓ Polished out-of-the-box UI designs
- ✓ Great for landing pages & quick MVPs
- ✓ Low initial technical barrier to entry

Key Limitations

- ✓ Platform lock-in & usage credit restrictions
- ✓ Exported code can be hard to modify manually
- ✓ Complex backend custom logic can be difficult
- ✓ Often requires specialized Node.js hosting setups

Teaching Decision: We focus on custom AI software development first. No-Code builders will be covered as a fast-track alternative in Section 13.

Today's Complete Production Toolchain

The exact five-stage technology stack used throughout this masterclass.

1. **ChatGPT / Claude**
   PRDs & Specs
2. **Antigravity IDE**
   Multi-file Agent
3. **XAMPP Local**
   PHP & MySQL
4. **Manual QA**
   Browser Testing
5. **Hostinger Live**
   Live Hosting

Note: Individual tools can be replaced over time, but the overarching planning, testing, and deployment pipeline remains identical.

SECTION 03

Free vs Paid AI Development Tools

Understanding message limits, context windows, multi-file editing capabilities, coding output quality, and the real ROI of development time.

Free vs Paid Tiers & Developer ROI

Comparing tool tiers and calculating the practical financial ROI of a $20/month Pro IDE subscription.

Metric	Free Tier	Pro Tier ($20/mo)
Context Window	Small (forgets files)	Large (full project tree)
Multi-File Editing	Restricted / Single file	Seamless multi-file edits
Request Limits	Hourly rate limits	High-priority agent requests
Suitability	Learning / Small tests	Commercial & Client apps

Zero Wait Time
No rate-limit resets during active builds.

Fewer Re-Prompts
Large memory prevents repeating rules.

Faster Fixes
Finds root bug causes faster.

Client Speed
Completes projects 3× faster.

ROI Math: Saving just 2 hours of debugging per month pays for a $20/mo Pro subscription.

SECTION 04

Project Environment & Blueprint Documentation

Preparing the project context that enables AI to understand, build, refine, and maintain complex software applications correctly.

Why Documentation First & The 5 Blueprint Files

Coding without blueprint files causes project fragmentation. Master the 5 essential Markdown files.

⚠ Why Docs Before Code?

- ✓ Prevents duplicate files & broken database columns
- ✓ Provides a single source of truth for AI agents
- ✓ Enforces structured, phased development order
- ✓ Ensures easy future updates & rapid bug fixes

→ **Sequence: Idea → PRD → Agent → Plan → Structure → Build**

The 5 Blueprint Files

1. **prd.md (Mandatory):** Features, roles, schema, UI & tech stack.
2. **agent.md (Mandatory):** Permanent coding rules, security & behavior.
3. **implementation_plan.md:** Dependency-ordered phased roadmap.
4. **file_structure.md:** Project directory tree map & responsibilities.
5. **update.md (Post-v1):** Tracks changes after Version 1 launch.

Mandatory vs Recommended Documents

Understand the specific operational responsibility of each file.

Document	Status	Core Purpose & Operational Impact
`prd.md`	**Mandatory**	Answers **WHAT** to build (Features, roles, UI style, backend stack)
`agent.md`	**Mandatory**	Answers **HOW** to code (Rules, security, folder structure, conventions)
`implementation_plan.md`	**Recommended**	Answers **WHEN** to build what (Divides execution into testable phases)
`file_structure.md`	**Recommended**	Answers **WHERE** files belong (Directory tree map)
`update.md`	Post V1	Tracks **CHANGES** after Version 1 live launch safely

SECTION 05

Creating the Master PRD

Transforming an initial idea into a detailed Product Requirement Document (prd.md) that serves as the project's primary engineering blueprint.

Information Required Before Generating a PRD

Gather these core project parameters before writing the PRD prompt.

Core Identity

- ✓ Application Name & Type
- ✓ Main Business Objectives
- ✓ Problem Statement solved
- ✓ Target End-Users

Functionality

- ✓ User Roles (Admin vs User)
- ✓ Core vs Optional Features
- ✓ Required Frontend Pages
- ✓ Authentication requirements

Tech & Design

- ✓ Tech Stack (PHP/MySQL)
- ✓ Color Theme (#009E8C, Light)
- ✓ Typography & UI Style
- ✓ Hosting Target (Hostinger)

⚠ **Rule:** Use explicit placeholders for undecided items. Never let AI invent major business rules on its own!

</> Master PRD Prompt — Part 1

Use ChatGPT or Claude to generate the initial structured Product Requirement Document.

```
Act as a senior product manager, business analyst, UI/UX planner, database architect, and full-stack software architect. Your task is to create a complete and professional Product Requirement Document named `prd.md` for the application described below. Do not begin writing code. First, analyse the project information, identify missing critical requirements, and clearly label reasonable assumptions. Do not invent business rules, payment systems, integrations, user roles, or advanced features that I have not requested. PROJECT INFORMATION: Application Name: [APPLICATION NAME] Application Type: [WEBSITE / WEB APPLICATION / SAAS / SOFTWARE / ADMIN PANEL] Business Goal: [DESCRIBE THE MAIN BUSINESS GOAL] Problem Statement: [DESCRIBE THE PROBLEM THE APPLICATION SHOULD SOLVE] Target Users: [DESCRIBE THE TARGET USERS] User Roles: [ADMIN / USER / CUSTOMER / STAFF / OTHER ROLES] Core Features: [LIST ALL REQUIRED FEATURES] Color Theme: Primary [#009E8C], Background [#F8FBFB], Text [#111111] Technology Stack: [HTML, CSS, JAVASCRIPT, PHP, MYSQL] Hosting Environment: [SHARED HOSTING / VPS / HOSTINGER]
```

</> Master PRD Prompt — Part 2 (Document Output Structure)

Mandatory sections required in the generated prd.md document.

```
Generate the `prd.md` document with the following exact Markdown sections: 1. Project Overview & Business Goal 2. Problem Statement & Scope (In-scope vs Out-of-scope) 3. User Roles & Permissions Matrix 4. Detailed Core Features (Inputs, Processing, Outputs, Validation) 5. Page and Screen Specifications (Every frontend, user dashboard, & admin screen) 6. Admin Panel Requirements (Filters, Search, Management controls, Logs) 7. Database Schema Requirements (Entities, tables, fields, relationships, indexes) 8. Technology Stack & Directory Architecture 9. UI & Design System (Color palette, typography, cards, buttons, responsive rules) 10. Security Requirements (XSS, SQLi, CSRF, Password hashing, Role guards) 11. Testing & Acceptance Criteria (Measurable completion criteria for every feature) 12. Deployment Requirements (PHP version, writable folders, database import) Output Rules: Save as clean Markdown conceptually named `prd.md`. Do NOT write application code.
```

Critical PRD Directives: Stack, Folders & Inline Code

Essential rules to specify in your PRD to ensure manageable, easily editable code later.

1. Explicit Color Theme

By default, AI tools generate aggressive dark backgrounds and neon gradients. You MUST explicitly enforce: Primary #009E8C, Dark Teal #00665E, Light Background #F8FBFB, Card White #FFFFFF.

2. Manageable Tech Stack

Enforce standard **HTML, CSS, JavaScript, PHP, and MySQL**. Traditional stacks are far easier to edit, manage, debug, and host without complex build frameworks like React or Node.

3. Separate Frontend & Backend

Instruct AI to keep distinct folder structures for frontend views (`/index.php`, `/dashboard.php`) and backend action handlers (`/backend/process_login.php`) for safety.

4. Inline CSS & JavaScript

Require AI to include page-specific CSS (`</div>`) and JS (`</div>`) directly inside each individual page file. Avoid external URL-based assets so you can open 1 file and redesign it instantly!

The Inline Page Code Organization Strategy

A practical layout preference for small-to-medium PHP web applications.

Preferred Page Structure

```
        ... /* 4. Page-Specific CSS Close to Component */ // 5. Page-
Specific JavaScript Logic
      ...
```

Why This Works Best

Opening a single file lets you modify markup, styles, and logic instantly without searching across 15 separate subfolders.

Security Exception: Database connection keys, passwords, and security helpers must remain strictly centralized in protected includes!

SECTION 06

Creating agent.md Custom AI Rules

Establishing permanent instructions that govern coding quality, file organization, UI consistency, security standards, and AI behavior.

What agent.md Controls During Development

While prd.md defines WHAT to build, agent.md governs HOW the AI must write code.

</> Coding Standards

Naming conventions, function size limits, error handling patterns, and prohibiting placeholder code.

File Discipline

Strict file placement, forbidding duplicate creation, and preserving working code during updates.

Security Guards

Mandatory prepared statements, input sanitization, role guards, and secure session handling.

Core Concept: `agent.md` **acts as the AI IDE's permanent operating system guidelines throughout the entire development lifecycle.**

Master agent.md Prompt — Part 1

Creating permanent AI engineering rules derived directly from the approved PRD.

```
Act as a senior software engineering lead responsible for creating a permanent AI instruction file named `agent.md`. Read the completed `prd.md` before generating this file. The purpose of `agent.md` is to ensure that every AI coding tool or developer follows the exact same project rules throughout initial development and future updates. Generate the document with the following sections: 1. PROJECT PRIORITY RULES • The PRD is the primary source of truth. • Do not add features not listed in the PRD. Never silently alter the tech stack. 2. DEVELOPMENT SEQUENCE RULES • Follow `implementation_plan.md` step-by-step. Work on ONE phase at a time. • Never mark a phase complete if placeholder functionality remains. 3. FILE & FOLDER RULES • Store files in correct directories according to `file_structure.md`. • Never create duplicate files for existing features. Search the project first.
```

Master agent.md Prompt — Part 2

Security, UI consistency, and AI response protocol rules.

```
4. UI & DESIGN SYSTEM RULES • Follow the exact color theme: Primary [#009E8C], Background [#F8FBFB], Text [#111111]. • Do not switch to dark mode. Maintain light theme. All pages MUST be fully responsive. 5. DATABASE & SECURITY RULES • Use prepared SQL statements for ALL queries. Sanitise all user inputs. • Enforce role-based permission checks on top of every protected page. 6. AI RESPONSE PROTOCOL • Before code: State phase name, active task, files involved, and dependencies. • After code: Provide created files list, modified files list, testing checklist, and stop for human confirmation. Output: Save as clean Markdown conceptually named `agent.md`.
```

SECTION 07

Creating the Implementation Plan

Generating a phase-by-phase development roadmap (implementation_plan.md) derived from both the approved PRD and permanent agent rules.

Implementation Plan Dependencies

The implementation plan must only be generated AFTER PRD and Agent files are complete.

⚠ Strict Requirement

Never attempt to generate `implementation_plan.md` from a brief idea alone. The AI must read and synthesize both `prd.md` and `agent.md` simultaneously.

Inputs & Output Flow

- ✓ **Input 1:** `prd.md` (Features, Users, Database & UI Requirements)
- ✓ **Input 2:** `agent.md` (Coding rules, Folder conventions & Security)
- ✓ **Output:** `implementation_plan.md` (Sequenced, testable development phases)

Master Implementation Plan Prompt — Part 1

Instructing the AI to parse requirements into dependency-ordered phases.

```
Act as a senior technical project manager and software architect. Read the complete contents of: • `prd.md` • `agent.md` Do not generate application code. Your task is to create a detailed phase-based development roadmap named `implementation_plan.md`. Before creating the phases: 1. Extract all requested features, user roles, pages, and database tables. 2. Identify dependencies between features (e.g., Database connection → User Auth → Dashboard). 3. Group development into distinct, testable phases. Do NOT attempt the entire build in one phase!
```

Master Implementation Plan — Phase Breakdown

Standard 8–phase execution sequence for building commercial web software safely.

PHASE 1

Foundation

Folders, config files, DB keys & shared includes.

PHASE 2

DB Schema

Tables, SQL imports, indexes & seed data.

PHASE 3

Authentication

Login, logout, session management & role guards.

PHASE 4

Frontend Layout

Navbars, footers, inline CSS & design tokens.

PHASE 5

Core Features

Main application logic, user actions & forms.

PHASE 6

Admin Panel

Management dashboards, user lists & system logs.

PHASE 7

UI & Responsive

Mobile breakpoints, card spacing & modal dialogs.

PHASE 8

Security Audit

XSS escaping, SQLi verification & CSRF checks.

Anatomy of an Implementation Phase

Every phase inside implementation_plan.md must follow this exact structure.

Required Phase Template

- ✓ **Phase Name & Objective:** Clear target result
- ✓ **PRD Requirements Covered:** Mapped feature IDs
- ✓ **Files to Create / Modify:** Precise path lists
- ✓ **Database Changes:** Required schema adjustments
- ✓ **Testing Checklist:** Step-by-step human verification

Traceability Matrix

The document ends with a Traceability Matrix table ensuring every single requirement from `prd.md` is explicitly assigned to a specific phase. Zero orphaned features allowed!

SECTION 08

Creating file_structure.md & update.md

Mapping the project directory tree and setting up a maintenance lifecycle file for safe post-launch updates.

file_structure.md Purpose & Tree Map

Prevents AI IDEs from creating misplaced or duplicate files during execution.

```
/my-app-root |── /config # Database & environment keys |── /includes # Header, footer, auth, security helpers |── /admin # Admin panel pages & dashboard |── /user # User account management pages |── /backend # Form action handlers & API endpoints |── /uploads # User-uploaded files directory |── /database # SQL schema migration files |── index.php # Public homepage |── prd.md # Primary Product Specs |── agent.md # AI Coding Rules └── file_structure.md
```

Why This Map Matters

When asking an AI IDE to "add user profile upload," it checks `file_structure.md` to know that the handler belongs in `/backend/upload_profile.php` and assets in `/uploads/avatars/`.

Master File Structure Prompt

Generating the complete directory map file from PRD and Implementation Plan.

```
Read: `prd.md`, `agent.md`, `implementation_plan.md` Create a complete project directory map named `file_structure.md`. The structure must support every required page, user role, backend database handler, upload folder, include, and development phase. For every folder, explain: 1. Folder Purpose & Security Level (Public vs Protected) 2. Files it should contain 3. File responsibilities and dependencies Rules: • Do not create duplicate files for the same purpose. • Match all file paths used in `implementation_plan.md`. • Save as clean Markdown conceptually named `file_structure.md`.
```

update.md Purpose & Maintenance Lifecycle

Never rewrite the original PRD after Version 1 launch. Use update.md instead.

Original prd.md

Represents the **frozen contract** of Version 1.0 initial product launch.

Dynamic update.md

Tracks **Version 1.1+ changes, bug fixes, SQL migrations**, and newly requested features.

Safety Guard: Using `update.md` prevents AI IDEs from accidentally redesigning or breaking existing, previously working Version 1 pages!

Master update.md Prompt

Use this prompt when adding features or making post-launch modifications.

```
Act as a software maintenance planner. Read `prd.md`, `agent.md`, `implementation_plan.md`, and `file_structure.md`. Create or update a maintenance document named `update.md` for the following update request: Update Request: [DESCRIBE REQUIRED CHANGE OR NEW FEATURE] Reason: [DESCRIBE WHY IT IS NEEDED] Include: 1. Target Version Number & Summary 2. Affected Files & New Files Needed 3. Database Migration SQL Changes 4. Backward-Compatibility & Security Risk Check 5. Step-by-Step Implementation Steps & Regression Testing Checklist Rules: Preserve existing working features. Make the smallest safe code modification. Save as `update.md`.
```

SECTION 09

Orchestrating Antigravity IDE

Uploading project documentation blueprints and instructing the AI IDE coding agent to execute one verified phase at a time.

Pre-Development Document Upload Checklist

Upload these exact four blueprint files into Antigravity IDE before issuing execution prompts.

prd.md
Product Specs

agent.md
Coding Rules

implementation_plan.md
Phased Roadmap

file_structure.md
Directory Map

Checklist Verified: Tech stack, color theme, database strategy, and hosting target are 100% confirmed across all documents.

Master Development Execution Prompt — Part 1

Initial execution prompt issued in Antigravity / Cursor IDE.

```
You are the primary AI software engineer responsible for implementing this project. Before writing or modifying any code, read the complete contents of: `prd.md`, `agent.md`, `implementation_plan.md`, `file_structure.md`. Do not rely only on current chat memory. Treat uploaded files as absolute project truth. First, provide a short project-understanding report containing: 1. Application Purpose & User Roles 2. Approved Tech Stack & Color Theme (#009E8C, Light) 3. Current active implementation phase (Phase 1) 4. Expected files to create for Phase 1 5. Any requirement conflicts preventing implementation Do NOT proceed if requirements conflict. Do NOT invent unrequested features.
```

Master Development Execution Prompt — Part 2

Rules forcing single-phase execution and human confirmation stops.

```
EXECUTION RULES: • Follow `implementation_plan.md` in strict sequence. Build ONLY Phase 1 now. • Do NOT skip ahead to future phases. • Do NOT attempt the entire project in one prompt response. • Search existing files before creating new ones. Prevent duplicates. • Implement Phase 1 code now. • Update `file_structure.md` if directory changes occur. • Provide local testing instructions for Phase 1. STOP after completing Phase 1 and wait for my explicit confirmation that testing passed before starting Phase 2!
```

Required Agent Phase Completion Report

The required response protocol the AI IDE must present after completing a phase.

```
Phase Completed: [Phase Name] Implemented Features: [List of completed items] Files Created: [List of created file paths] Files Modified: [List of updated file paths] Database Schema Changes: [SQL tables created or altered] Testing Instructions: [Step-by-step local testing guide] Known Limitations: [Any remaining edge cases] Status: Phase 1 Complete • Awaiting Human QA Approval
```

SECTION 10

Testing and Debugging Workflows

Running projects locally, testing every user flow manually, recording failures, and repeating structured correction cycles.

Local Stack: XAMPP Architecture

XAMPP creates a local web and database server environment on your computer.

Apache
Local web server executing PHP files.

MySQL
Relational database engine storing data.

phpMyAdmin
Web UI for managing MySQL tables & SQL.

htdocs
Root directory for local project folders.

1 Browser Request → 2 Apache Server → 3 PHP Engine → 4 MySQL Database → 5 HTML Render

Step-by-Step Local XAMPP Setup

Follow these exact steps to configure and run your project locally.

- ✓ 1. Install and launch XAMPP Control Panel
- ✓ 2. Start **Apache** and **MySQL** services
- ✓ 3. Copy project folder to `C:\xampp\htdocs\my-project`
- ✓ 4. Open browser to `http://localhost/phpmyadmin`
- ✓ 5. Create a new database matching `prd.md` specs
- ✓ 6. Import database `.sql` file into phpMyAdmin
- ✓ 7. Configure `/config/database.php` connection keys
- ✓ 8. Open browser to `http://localhost/my-project`
- ✓ 9. Test homepage, user login, forms, and admin pages
- ✓ 10. Verify database writes and session storage

Troubleshooting Common Local Server Errors

How to diagnose and fix frequent local environment issues.

Port 80 Conflict

Port 80 blocked by another service. Change Apache port to 8080 or run as Admin.

Database Connection Error

Incorrect DB name, user, or pass in config file. Verify MySQL is running.

Blank Page / 500 Error

PHP syntax error or missing include. Enable `display_errors = On` in php.ini.

⚠ **Rule:** Never suppress error reporting to make a broken page appear functional. Always fix the underlying code issue!

Manual Quality Assurance Checklist

Test every completed phase across these four critical functional pillars.

1. Navigation

Every link, button, back route, and redirection flow.

2. Forms & Inputs

Valid entries, empty fields, invalid emails, and long text.

3. Auth & Roles

Login, logout, session guards & blocking unauthorized URL access.

4. Responsiveness

Desktop widescreen, tablet portrait, and mobile smartphone views.

The Professional Bug Reporting Loop

A structured process for collecting and fixing application bugs systematically.

1. Manual QA
2. Collect Notes
3. Evidence
4. Clean Bug List
5. Send to IDE
6. Retest Fixes
7. Regression QA

Golden Method: Batch 5–10 issues into one structured report using a Chat Model before handing the request to Antigravity IDE!

Bug-Report Cleanup Prompt

Convert messy manual testing notes into a structured developer report.

```
Act as a senior QA engineer. Convert my raw manual testing notes into one clear, structured bug-fixing request for an AI coding IDE. Do not attempt to write code yet. Organise every issue using: 1. Bug ID & Affected Page/Module 2. Steps to Reproduce 3. Current Behavior vs Expected Behavior 4. Error Message / Console Output 5. Acceptance Criteria for Fix RAW QA TESTING NOTES: [PASTE ALL RAW TESTING NOTES & SCREENSHOT DESCRIPTIONS HERE]
```

Master Bug-Fixing Prompt (Issued in IDE)

Submitting the cleaned QA bug report to Antigravity IDE for resolution.

```
Read project documentation files and the structured QA bug report below. Before making changes: • Identify root causes and affected files. • Confirm that fixes adhere strictly to `agent.md`. • Do NOT redesign or refactor unrelated working areas. Fix issues in priority order. For each bug: 1. Explain root cause. 2. Make the smallest safe code fix. 3. Provide testing steps to confirm resolution. STRUCTURED QA BUG REPORT: [PASTE CLEANED QA REPORT]
```

SECTION 11

Advanced UI and UX Design Polish

Transforming functional AI-generated interfaces into polished, consistent, responsive, and commercial product experiences.

Why First-Pass AI UI Design is Often Insufficient

Initial raw AI code generation usually prioritizes backend logic over refined visual aesthetics.

⚠ Common Raw AI UI Weaknesses

- ✓ Generic layouts with poor typographic contrast
- ✓ Unbalanced padding and inconsistent card spacing
- ✓ Awkward mobile viewport breakpoints & horizontal scroll
- ✓ Inconsistent button sizes and color treatments

The Professional Solution

Use reference website visual analysis, image-to-code workflows, or multi-AI design refinement passes to elevate standard layouts to Stripe/Linear levels.

Rule: AI IDEs are strong at implementation, but professional design usually requires visual references and explicit CSS tokens.

Method 1: Reference Website Design Workflow

Use world-class product interfaces (Stripe, Linear, Apple) as design references.

1. Find Reference
2. Capture Screen
3. Analyze Layout
4. Generate Specs
5. Apply to IDE

Ethical Guard: Use reference sites ONLY for layout structure, spacing, and typography inspiration. Never copy logos, text, or copyrighted media!

Reference Design Analysis Prompt

Analyze a reference screenshot to extract clean layout instructions.

```
Act as a senior UI/UX designer. Analyse the uploaded reference website screenshot. Extract ONLY design principles: • Layout structure & grid spacing • Typographic hierarchy & card styling • Button styling & section padding Apply these principles to my application page. App Color Theme: Primary [#009E8C], Background [#F8FBFB], Cards [White] Target Style: Clean, modern, Stripe/Linear inspired. Provide exact CSS layout instructions for my page file.
```

Method 2: AI Image-to-Code UI Workflow

Generate an AI mock image concept, approve it, and convert it directly into frontend code.

1. Prompt UI Image
2. Generate Mockup
3. Select Concept
4. Upload to IDE
5. Convert to Code

Benefit: Ideal when no live reference website matches your vision. Visually validates layout before writing frontend code.

UI Image Generation Prompt — Part 1

Generating high-fidelity UI visual concepts using Midjourney, ChatGPT, or Flux.

```
Create a high-fidelity modern UI/UX design concept for a web application. Product Name: [PRODUCT NAME] Product Type: [WEB APPLICATION / SAAS DASHBOARD / LANDING PAGE] Color Palette: Primary Teal [#009E8C], Background [#F8FBFB], Cards [White], Accent [#00665E]. Visual Style: Light theme, clean Apple x Stripe x Linear aesthetic, generous whitespace, soft card shadows, crisp modern typography. Required View: Full page desktop layout from header navigation to footer.
```

UI Image Generation Prompt — Part 2

Ensuring realistic, implementable UI layout outputs.

```
Design Requirements: • Light background ONLY. No dark mode. • Realistic form inputs, table structures, and CTA buttons. • No aggressive neon effects or random gradients. • No fake 3D angled laptop mockups. Flat front-facing UI layout only. • Implementable using clean HTML, CSS, and JavaScript.
```

Image-to-Code Implementation Prompt

Converting the approved mock image into actual code inside Antigravity IDE.

```
Use the uploaded UI concept image as the visual reference for [page_file_name.php]. Read `prd.md`, `agent.md`, and `file_structure.md`. Rebuild the page layout to mirror the visual structure of the image while preserving all actual working PHP backend logic, form IDs, and database operations. Rules: • Keep primary color #009E8C and light theme. • Ensure full mobile responsiveness. • Do NOT break backend PHP connection code.
```

Method 3: Multi-AI Model UI Refinement Pass

Test single-page code across different chat models to find the strongest UI variation.

1. Isolate Page Code

Copy HTML/CSS of one key page (e.g., Homepage or Dashboard).

2. Test Variations

Submit code to Claude, ChatGPT-4o, and Gemini for independent styling updates.

3. Standardize Master

Select the best output and apply its CSS tokens across all remaining app pages.

Multi-AI UI Refinement Prompt

Refine page visuals without breaking underlying functional logic.

```
Act as a senior frontend engineer and UI designer. Refine the provided single-page code visually. Color Rules: Primary [#009E8C], Background [#F8FBFB], Card [White]. Light Theme. Style Goal: Stripe/Linear aesthetic, crisp typography, clean cards. Preserve: • ALL PHP logic, form input names, IDs, and session checks. • Do NOT alter functional backend routes. PASTE PAGE CODE HERE
```

SECTION 12

Production Deployment

Moving the tested project from local XAMPP to live production hosting and verifying functionality under live server conditions.

Pre-Deployment Verification Checklist

Complete these ten essential checks before uploading to live hosting.

- ✓ 1. All Phase 1–8 features completed & verified
- ✓ 2. Test data and debug echo statements purged
- ✓ 3. Database SQL export file generated
- ✓ 4. User auth & permission guards tested
- ✓ 5. Mobile responsiveness verified
- ✓ 6. Hardcoded `localhost` URLs converted to relative
- ✓ 7. Upload directory permissions checked
- ✓ 8. Target hosting domain & SSL active
- ✓ 9. Database credentials prepared for production
- ✓ 10. Full local backup ZIP created

Deploying PHP & MySQL to Live Hostinger

Step-by-step procedure for deploying to production hosting.

- ✓ 1. Log into Hostinger hPanel & open Database section
- ✓ 2. Create a new production MySQL database & user
- ✓ 3. Open phpMyAdmin & import your local SQL file
- ✓ 4. Open File Manager -> `public_html`
- ✓ 5. Upload project ZIP file & extract contents
- ✓ 6. Update `/config/database.php` with live DB details
- ✓ 7. Set write permissions (755) on `/uploads` folder
- ✓ 8. Force HTTPS / SSL in Hostinger settings
- ✓ 9. Open live domain & execute full production QA
- ✓ 10. Check server error logs for edge-case errors

Live Production Verification Protocol

Running on local server does not guarantee identical behavior on live servers.

1. Live Form & Auth Check

Test user registration, login, session retention, and form submissions on live HTTPS domain.

2. File Uploads & Database

Verify image uploads save to server folders and database records insert correctly.

3. Error Log Monitoring

Inspect Hostinger PHP Error Logs to fix subtle path or case-sensitivity issues.

Status: Once live verification passes with zero log errors, Version 1.0 is officially LAUNCHED!

SECTION 13

Fast-Track Workflow for Simple Websites

Using autonomous AI website builders when a full multi-file engineering process is unnecessary for basic marketing websites or landing pages.

Full Workflow vs Fast-Track Decision Matrix

Select the right workflow based on project complexity.

Full Docs + AI IDE Workflow

- ✓ SaaS Platforms & Web Applications
- ✓ User Dashboards & Admin Panels
- ✓ Custom Database Logic & Role Permissions
- ✓ Long-term custom commercial products

Fast-Track AI Builder

- ✓ Simple Landing Pages & Portfolio Sites
- ✓ Basic Corporate Marketing Websites
- ✓ Event Sites & Static Information Pages
- ✓ Fast visual prototypes with no backend DB

Autonomous AI Website Builders Overview

Popular platforms for rapid visual landing page generation.

Supported Tool Suite

- Lovable
- Bolt.new
- v0
- Replit

Pros

Instant visual output, strong default aesthetics, low setup time for simple landing pages.

Cons

Usage credit limits, exports often require Node.js hosting, custom backend logic can be difficult.

The Fast-Track Website Generation Process

A streamlined, highly efficient process for launching basic marketing landing pages quickly.

1. **Simplified PRD** — Core sections & colors
2. **Input Builder** — Bolt / Lovable prompt
3. **Visual Edit** — Adjust text & spacing
4. **Responsive QA** — Mobile & tablet view
5. **Export Code** — ZIP / GitHub repo
6. **Deploy Live** — Vercel / Hostinger

Rule: In the Fast-Track workflow, `prd.md` serves as the single primary instruction file. Secondary documents remain optional.

Node.js Hosting vs Traditional Shared Hosting

AI website builders typically output React / Vite / Next.js code bases.

Exported Stack Reality

Exported code from Bolt.new or Lovable usually consists of **React, Next.js, or Vite**. These CANNOT be hosted by uploading to basic shared PHP hosting without a build step!

Compatible Deployment Targets

- ✓ Hostinger VPS or Node.js-enabled hosting
- ✓ Vercel / Netlify Cloud Hosting
- ✓ GitHub Pages (for pure static exports)

Fast-Track Export Prompt & Deployment Checklist

Prepare exported code for independent deployment outside the builder.

```
Prepare this website for independent export outside this builder. Before export: • Remove placeholder content and unused components. • Verify form handlers and environment variables. • Provide `package.json`, build command (`npm run build`), and deployment steps.
```

Checklist: Inspect `package.json` -> Run production build -> Deploy to Vercel or Hostinger Node.js -> Verify live site.

The Complete AI Development Master System Map

The entire repeatable workflow from initial idea to production launch and maintenance.

1. **Idea & Tools** — Select Chat/IDE stack
2. **prd & agent.md** — Specs & coding rules
3. **Phased Plan** — implementation_plan
4. **Directory Map** — file_structure.md
5. **Antigravity IDE** — Build 1 phase at a time
6. **Local XAMPP QA** — PHP/MySQL testing
7. **UI Reference** — Stripe/Linear polish
8. **Hostinger Launch** — Deploy & update.md

Master Outcome: You now possess an end-to-end engineering system to build, test, polish, and deploy any digital software product independently!

Recommended Student Practice Projects

Apply this exact masterclass workflow to these progressive practice assignments.

Beginner

- ✓ Personal Developer Portfolio
- ✓ SaaS Product Landing Page
- ✓ Simple Business Contact Tool

Intermediate

- ✓ Lead Management CRM Dashboard
- ✓ Appointment Booking System
- ✓ Blog Platform with Admin Panel

Advanced

- ✓ Multi-Role Customer Support Portal
- ✓ Subscription Management SaaS
- ✓ AI Content Generator Software

Questions & Live Build Session

Let's clarify questions, review your PRD documents, and begin our live step-by-step application build demonstration.

Let's Build It Step by Step.

Your Immediate Next Steps

Put this masterclass engineering workflow into practice today.

- ✓ 1. Select your target project idea
- ✓ 2. Draft your initial PRD information parameters
- ✓ 3. Generate `prd.md` using Claude or ChatGPT
- ✓ 4. Generate `agent.md` engineering rules
- ✓ 5. Create `implementation_plan.md` & `file_structure.md`
- ✓ 6. Upload documents into Antigravity or Cursor IDE
- ✓ 7. Execute Phase 1 development
- ✓ 8. Test locally on XAMPP server
- ✓ 9. Conduct QA bug fixing cycles
- ✓ 10. Polish UI aesthetics and deploy to live hosting!

MASTERCLASS COMPLETION

Thank You for Attending!

You now possess a complete, professional engineering system for planning, building, testing, refining, deploying, and maintaining software using AI.

"AI will not replace developers. Developers who know how to use AI effectively will have a major advantage."