PHOTOSTUDIO

Photography Booking & Appointment Management System

Designed, developed, integrated and hardened end-to-end

ENGINEERING SNAPSHOT

I built the original booking prototype, then independently reviewed it and turned it into a full-stack system by finding and fixing eight concrete engineering defects.

BACKEND ACCEPTANCE

Success is tied to actual backend acceptance — the UI no longer reports a booking as successful unless the server accepts and persists it.

TRUST BOUNDARY

Pricing is server-authoritative — browser-controlled totals are never trusted as the final business value.

From Prototype to Engineered System

Prototype foundation: I designed the original service-selection, calendar/time-slot and customer-booking experience. **The main engineering work came next:** I audited the system myself, traced the broken request/data paths, corrected the trust boundary, protected database writes, repaired email/dependency configuration and made the booking lifecycle behave as a real system rather than a visual demo.

Eight Defects I Found and Fixed

Defect	Why it mattered	Engineering fix
1. Wrong submit target	The form pointed at an unrelated external sendemail.php path.	Reconnected submission to the application's own PHP booking endpoint.
2. False client success	JavaScript prevented submission and displayed success regardless of persistence.	Success now follows the backend response and actual booking acceptance.
3. Calendar/state mismatch	Custom date state and the appointment-date reference were inconsistent.	Unified selected date/time state across UI, request payload and server processing.
4. Field-contract drift	Frontend names and PHP expectations did not agree.	Established a consistent browser → PHP data contract.
5. Hardcoded price	The early backend placeholder used a fixed 1,800 value instead of evaluating the selected service.	Replaced the placeholder with server-side service/pricing rules; client totals are presentation only.
6. Booking race condition	Two users could see the same slot as free before either write completed.	Added final server-side conflict validation plus transactional slot reservation.
7. Fragile dependency setup	PHPMailer integration had syntax/dependency fragility.	Corrected integration and declared PHPMailer through Composer.
8. Embedded credentials	Database/SMTP credentials were stored in application source.	Moved deployment-sensitive values to environment-driven configuration.

FESTUS AWESOME

What the Hardening Changed

Engineering area	Result
Submission integrity	The browser can request a booking, but only the backend decides whether it succeeded.
Pricing integrity	The server validates the service/options and calculates the authoritative amount; manipulated hidden/browser totals do not control the stored price.
Availability integrity	AJAX availability improves UX, while the final server-side re-check protects the actual reservation decision.
Concurrency	MySQL booking/slot state is protected during the final write path with a transaction and slot reservation logic.
Data contract	HTML/JavaScript/PHP use a consistent request model for customer, service, date/time and booking details.
Persistence	Appointments are stored through mysqli prepared statements rather than treated as a front-end-only interaction.
Email lifecycle	PHPMailer/Gmail SMTP confirmation and scheduled reminder workflows are connected to booking data.
Configuration	Database and SMTP settings are separated from source and supplied through deployment configuration.

Architecture

Browser	Async UX	Authority	State	Messaging
HTML/CSS/JS	Fetch API	PHP	MySQL	PHPMailer/SMTP

Service/options → availability request → server validation/pricing → conflict re-check → transaction/persistence → confirmation

Evidence files: script.js, check_availability.php, submit_booking.php, db.sql / db_migration.sql, config.php, composer.json and Reminder.php.

Why This Project Is Strong Evidence

The most important work was self-directed review after the prototype existed. I did not stop when the interface looked complete: I traced the request path end-to-end and tested the system boundary, found where the client and server disagreed, identified where browser-controlled data was being trusted, recognized a concurrency failure mode, and corrected deployment/security weaknesses.

Summary

"I designed the original PhotoStudio booking prototype, then audited my own implementation and found eight concrete issues across submission, client/server state, field contracts, pricing, concurrency, dependencies and credentials. I fixed them by making backend acceptance authoritative, moving pricing and validation server-side, protecting the final booking write with conflict checks and a transaction, and separating deployment secrets from source. The important lesson was that a polished interface is not the same as a trustworthy system."

Scope & Next Steps

The application does not claim to process raw card data without a real payment provider. The most valuable production next steps are a PCI-compliant payment integration, automated tests, rate limiting, and operational monitoring.

PHP • MySQL • JavaScript • Fetch/AJAX • Prepared Statements • Transactions • Concurrency Control • Server-side Validation • PHPMailer/SMTP • Composer • Security Hardening • Integration Debugging

FELIX'S COPY