Internship at Aspire to Innovate (a2i) Muktopaath, and the Design, Development, and Evaluation of Two Web-Based Software Systems: Chokh-e-Dekha and Pharmazine

by

Mohammad Azmain Hossain Alfi
21201489

A report submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
June 2026.

Declaration

It is hereby declared that

1. The report submitted is my own original work while completing my degree at Brac University.
2. The report does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The report does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. I have acknowledged all main sources of help.
5. The two software systems described in this report, *Chokh-e-Dekha* and *Pharmazine*, were designed and developed by me individually as post-internship projects.

Student's Full Name & Signature:

Mohammad Azmain Hossain Alfi

21201489

Ethics Statement

This work was carried out in accordance with the academic integrity and professional conduct expectations of Brac University. No experiments involving human participants or animals were conducted, and no personal or confidential data belonging to the Aspire to Innovate (a2i) Programme was reproduced in this report. All information relating to the internship is presented at a level appropriate for an academic deliverable and respects the confidentiality of the host organisation.

The two software systems documented here were built using open-source frameworks and libraries, each used in compliance with its respective licence and duly acknowledged. Where the systems handle sensitive information, such as citizen complaint data in *Chokh-e-Dekha* and patient and pharmacy records in *Pharmazine*, the report discloses the relevant privacy and security considerations honestly, including limitations that are not yet fully resolved. No claim is made in this report for any feature that cannot be demonstrated in the corresponding running system.

Abstract

My final-year work has two halves, and this report is where they meet. The first half was a three-month internship at the Aspire to Innovate (a2i) Programme, on the Muktopaath national e-learning platform. The second was two web-based software systems that I built afterwards, by taking what that internship taught me and putting it to work. This is an internship-based report, written under the Internship Track of the Department of Computer Science and Engineering at Brac University, and I have tried to write it as exactly that, not as a research thesis dressed up in internship language.

The first system, **Chokh-e-Dekha**, is a civic-issue reporting and accountability platform for urban Bangladesh. Citizens file geo-tagged, photo-evidenced reports of infrastructure problems; administrators get a command centre with Service Level Agreement (SLA) tracking; and a Right to Information (RTI) wizard helps a citizen escalate when nothing happens. It is built on Laravel and MySQL. The second system, **Pharmazine**, is a pharmacy management and point-of-sale (POS) system for small and medium pharmacies. It ties batch- and expiry-aware inventory directly to a fast POS workflow and surrounds it with procurement, multi-branch, and pharmacy-care modules. It is a React and TypeScript single-page application over a FastAPI and PostgreSQL backend, and it is deployed publicly so that it can be demonstrated rather than only described.

Instead of an academic research gap, I frame the work around an *industry practice gap*: real inefficiencies in everyday civic and pharmacy workflows that I understood far better after seeing how a government-scale platform is actually built and run. The report sets out the requirements, the societal and environmental impact, the ethics, the project management, and the economic analysis of both systems; their design alternatives and final architectures; and an honest evaluation that does not hide the parts that are not finished. There is a dedicated chapter that goes back to the internship itself, tracing how each thing I learned there became a specific design choice in one of the two systems. I should also say upfront that both systems are working prototypes, not finished products, and I have tried to be honest throughout the report about what still needs to be done.

Keywords: Internship; Civic Accountability Platform; Pharmacy Management; Point-of-Sale Systems; Batch and Expiry Tracking; Laravel; FastAPI; React; PostgreSQL; User-Centred Design; Bangladesh.

Dedication

For my family, who lived through the late nights and the conversations I stopped mid-sentence because something in the code had clicked, and for the public servants and engineers at a2i who made me see that building software can mean something beyond the code itself.

Acknowledgment

I owe a lot to a lot of people, and I want to say so properly.

Three months with the **a2i-Muktopaath team** changed the way I think about building software. Being part of a platform with hundreds of thousands of real users taught me things I could not have picked up in a classroom: what user-centred design actually looks like in practice, what real quality assurance involves, how systems-level thinking shapes every small decision. I learned by watching, by doing, and sometimes by getting things wrong. That experience is woven into every design choice in this report.

Nazmul Islam, my primary supervisor, thank you for the guidance and feedback that pushed me to think more carefully about software engineering and about how to present my work. **Mirza Mohammad Didarul Anam**, my workplace supervisor at a2i, thank you for trusting an intern with real work that genuinely affects people's lives.

The **Department of Computer Science and Engineering at Brac University** built a curriculum that actually prepared me for this; data structures, databases, and software engineering all became real when I had to apply them to Chokh-e-Dekha and Pharmazine. To my **family**, thank you for your patience and belief in me. And to the **open-source community** behind Laravel, FastAPI, React, PostgreSQL, and the many libraries I relied on, thank you for the foundations I built on.

Table of Contents

List of Figures

List of Tables

List of Acronyms

Abbreviation	Meaning
a2i	Aspire to Innovate
API	Application Programming Interface
ARIA	Accessible Rich Internet Applications
BaaS	Backend-as-a-Service
BDT	Bangladeshi Taka
CC	City Corporation
CO	Course Outcome
CRM	Customer Relationship Management
CSRF	Cross-Site Request Forgery
CSV	Comma-Separated Values
DNCC	Dhaka North City Corporation
ER	Entity-Relationship
FEFO	First-Expiry-First-Out
FIFO	First-In-First-Out
GPS	Global Positioning System
HR	Human Resources
ICT	Information and Communication Technology
JWT	JSON Web Token
KPI	Key Performance Indicator
MVC	Model-View-Controller
NFR	Non-Functional Requirement
ORM	Object-Relational Mapping
OWASP	Open Worldwide Application Security Project
PDF	Portable Document Format
POS	Point of Sale
PWA	Progressive Web App
RBAC	Role-Based Access Control
RLS	Row-Level Security
RTI	Right to Information
SLA	Service Level Agreement
SPA	Single-Page Application
SQL	Structured Query Language

Abbreviation	**Meaning**
UI/UX	User Interface / User Experience
VAT	Value Added Tax
WCAG	Web Content Accessibility Guidelines
WHO	World Health Organization
XSS	Cross-Site Scripting

Chapter 1

Introduction

1.1 Background

My earlier reports were about learning. This one is about building with what I learned. This is an internship-based project report, not a conventional research thesis, and I have tried to write it as what it actually is: the account of a software engineering journey that began with a three-month internship and continued into the design and construction of two working software systems. The Internship Track at Brac University asks a student to take what they learned inside a real engineering organisation and turn it into something they build themselves. That is exactly the arc this report follows.

From 26 June 2025 to 25 September 2025 I worked with the Muktopaath team at the **Aspire to Innovate (a2i) Programme**, the flagship digital-transformation initiative of the Government of Bangladesh under the ICT Division [1]. Muktopaath is Bangladesh's national e-learning platform and serves more than six lakh (600,000) registered learners [2, 3]. My work there had nothing to do with whatever framework happened to be trending that month. I automated certificate templates, ran content quality checks, hunted bugs, handled user-support tickets, built out a chatbot FAQ dataset, and attended national-level content-review workshops. What stayed with me afterwards was not any particular tool but a way of approaching the work: start from what the user actually struggles with, test to find what breaks rather than what works, plan for growth from the start, write things down as you go, and learn to talk to people who are not engineers. Chapter 6 returns to the internship in detail.

When the internship ended I did not want the lessons to stay abstract, so I chose to build two systems that addressed problems I had either observed directly or come to understand through the kind of real government and field data that a2i works with every day:

- **Chokh-e-Dekha** (a romanisation of a Bengali phrase meaning "as seen by the eye") is a web-based civic-issue reporting and accountability platform. It lets citizens file geo-tagged, photo-evidenced complaints about civic problems such as broken roads, water-logging, and waste, routes those complaints to

the relevant city-corporation jurisdiction, and gives administrators the tools to track and resolve them transparently.

- **Pharmazine** is a web-based pharmacy management and point-of-sale (POS) system for small and medium independent pharmacies. It ties batch- and expiry-aware medicine inventory directly to a fast POS workflow and adds procurement, supplier and customer management, multi-branch operations, and a set of pharmacy-care modules.

Both systems target the same underlying reality: in Bangladesh, a great deal of important everyday work, whether reporting a civic problem or running a neighbourhood pharmacy, still happens through manual, undocumented, and unaccountable processes, even though the infrastructure to digitise it clearly exists. Muktopaath has already demonstrated that a government-run digital platform can reach millions of Bangladeshis at scale. The gap is not that the technical infrastructure does not exist; it is that nobody has yet built focused, locally adapted, and affordable tools for these two specific workflows.

1.2 Rationale of the Study and Motivation

This is an internship report, not a research thesis, so I have framed the motivation around what I would call an **industry practice gap** rather than an academic research gap. It is a set of real, concrete inefficiencies in workflows that I either saw directly or came to understand far more clearly through the internship. Three observations pushed me toward this work.

First, people in Bangladesh clearly want to report civic problems, but the system gives them almost no accountability once they do. The a2i 333 National Helpline has taken well over 100 million calls since 2018 [4, 5], which is a strong enough signal of demand. But every one of those calls disappears into a phone system: no photograph, no GPS coordinate, no public log, and no way for the caller to know what happened next. The consequences are visible every monsoon season. Dhaka North City Corporation (DNCC) mapped 98 water-logging spots across 40 wards in 2024 and 2025 and manages them with a hotline and a reactive control room, yet nothing in that process creates a visual, trackable record [6, 7, 8]. Research on 333 use among marginalised communities also points to real barriers around trust and accessibility [9, 10]. The problem, then, is not that there is no demand or no channel; the problem is that the existing channel produces no evidence, no location data, and no public accountability trail.

Second, small pharmacies are managing critical stock with almost no digital support. There are 264,908 retail pharmacies in Bangladesh according to WHO's 2025 data [11], serving a market worth roughly US$2.7 billion. Studies of these drug shops consistently show that a large proportion of staff have received no formal training, and very few keep proper stock or sales records [12]. The Model Pharmacy initiative has tried to address the regulatory side of this, but the operational reality remains difficult [13]. When a batch of medicine expires without being flagged, that is both money written off and a potential risk to the patient who might

have received it [14]. Paid pharmacy software exists, but enterprise products cost far more than a neighbourhood pharmacy can justify, and generic cloud POS systems do not understand concepts like batch numbers or expiry dates. What is missing is something *web-based, affordable, and actually built around how a pharmacy works.*

Third, and this one is personal: the internship made both problems feel buildable. Before a2i, I would have looked at either issue and stopped at "the government should deal with this." Watching a government-backed platform actually work, at scale, for real people, shifted that. The two gaps started feeling less like things to complain about and more like things someone could actually sit down and build.

1.3 Problem Statement

Put plainly, this project takes on two connected problems within a single internship-driven body of work.

On the civic accountability side: *Bangladesh does not have a structured, transparent, location-aware digital system where citizens can submit evidence-backed reports of civic problems, follow what happens to them, and hold authorities accountable. Administrators, on their end, have no consolidated picture of where problems cluster or how fast they are actually being resolved.*

For pharmacy operations: *small and medium pharmacies in low-infrastructure settings lack an affordable, web-accessible system that tracks inventory at the batch and expiry level, executes fast and reliable point-of-sale transactions tied to that inventory, and supports the surrounding workflows of procurement, customers, and basic pharmacy care, while remaining usable on commodity hardware and intermittent connectivity.*

The unifying thesis of the report is that both problems are, at their core, the same kind of problem, a well-understood real-world workflow that has not yet been given a focused, affordable, locally adapted digital tool, and that the engineering judgement needed to build such tools is exactly what an industry internship develops.

1.4 Objectives

The objectives of this internship-based project are:

1. To consolidate the practical software engineering knowledge gained during the a2i-Muktopaath internship and demonstrate its direct application to original project development.
2. To design and build **Chokh-e-Dekha**, a civic-issue reporting platform providing geo-tagged, photo-evidenced citizen reporting; an administrative command centre with SLA tracking; public transparency; and an RTI letter-generation wizard.

3. To design and build **Pharmazine**, a pharmacy management and POS system providing batch- and expiry-aware inventory, a fast POS workflow with local payment methods, and the surrounding procurement, customer, multi-branch, and pharmacy-care modules.

4. To apply a disciplined, evidence-driven design methodology, comparing alternative solutions against explicit constraints, for both systems.

5. To evaluate both systems honestly against their requirements, including performance, usability, security, and a candid account of current limitations.

6. To reflect on the professional and technical growth produced by the internship and to articulate how that growth shaped the two systems.

1.5 Methodology in Brief

Both systems were developed using the same five-stage, evidence-driven methodology that I distilled from the way teams worked at a2i. Each stage produces a concrete output that feeds the next: (1) *problem analysis*, grounding the problem in real government and field data; (2) *design alternatives*, comparing candidate platforms and approaches against explicit criteria; (3) *technology selection*, choosing a stack justified by cost, learning curve, and fitness for local conditions; (4) *architecture and database design*, producing component diagrams, normalised schemas, and an indexing strategy; and (5) *iterative implementation and evaluation*, building features in cycles and testing each against its requirements. This methodology is described in full in Chapter 4 and is applied separately to each system. Chokh-e-Dekha is implemented as a server-rendered Laravel (PHP) application over MySQL; Pharmazine is implemented as a three-tier React/TypeScript single-page application over a FastAPI (Python) service and a PostgreSQL database, deployed on managed cloud platforms.

1.6 Scope and Challenges

Scope. The report covers two web applications (desktop and mobile responsive), not native mobile apps. Chokh-e-Dekha covers civic categories such as roads, water and drainage, waste, electricity, illegal construction, environmental issues, and public safety, routed across the major Bangladeshi city corporations, together with citizen-facing features (submission, tracking, comments, bookmarks, RTI wizard) and administrator-facing features (report management, command centre, analytics, bulk actions, SLA tracking). Pharmazine covers batch/expiry inventory, POS with returns and local payment methods, procurement, supplier and customer management with loyalty, multi-branch operations, pharmacy-care modules, reporting, role-based access control, and audit logging. Neither system claims formal integration with government registries or city-corporation back-office systems, because no such public interfaces exist in standardised form.

Challenges. Building both systems brought up engineering problems that recur throughout the report: browser geolocation that loses accuracy in dense neighbourhoods; photo uploads that stall on 3G connections; query performance once tables grow large; concurrency bugs in simultaneous endorsements and batch stock decrements; type mismatches between the frontend and backend across cloud platforms; and the need to harden the database layer before any real sensitive data goes in. Beyond the technical, the central challenge is institutional rather than software: a civic platform cannot by itself compel a government body to respond, and a pharmacy system is only as good as the small business willing to adopt it. The report is honest about these limits while showing that the technical and design problems are solved to the level of a demonstrable, working prototype.

Chapter 2

Literature Review and Industry Background

This chapter establishes the technical and domain background for both systems and reviews the existing platforms they build on and differ from. Because the report is internship-based, the review pays particular attention to the *practice* context, what comparable systems do in the field, rather than to a purely theoretical research lineage.

2.1 Preliminaries

2.1.1 Civic Technology Concepts

Civic technology refers to digital tools that improve the relationship between citizens and government, increase transparency, or enable civic participation; complaint-management and issue-reporting platforms are one of its most established categories [15]. Three concepts recur in this category and shape Chokh-e-Dekha. **Geo-tagging**, the attachment of latitude/longitude coordinates to a record, allows reports to be mapped, clustered, and routed by jurisdiction; it is made practical by GPS-enabled smartphones. A **Service Level Agreement (SLA)** in public-service delivery specifies the time within which an authority is expected to act on a request; Chokh-e-Dekha adopts a configurable default SLA so that long-unresolved reports become visible. The **Right to Information (RTI) Act, 2009** of Bangladesh gives every citizen the right to request information from a public authority, with a designated officer required to respond within a statutory period [16]; it is a powerful but underused accountability mechanism, partly because drafting a compliant application is difficult for ordinary citizens.

2.1.2 Pharmacy and Inventory Concepts

On the pharmacy side, the relevant concepts are **batch (lot) tracking**, where stock is tracked per delivered batch with its own expiry date and cost; **expiry-aware dis-**

pensing, where stock nearing expiry is prioritised or flagged to avoid both waste and the dispensing of expired medicine; **reorder-level inventory control**, where low-stock thresholds trigger procurement; and the **point-of-sale transaction pattern**, where a sale atomically records line items, applies discounts and tax, accepts one or more payments, and decrements stock. Because medicines carry expiry dates, the safety-correct rule is to consume soonest-expiring stock first; Pharmazine implements expiry-aware batch selection on top of batch tracking so that dispensing follows expiry order rather than purchase order alone.

2.1.3 Technical Foundations

Both systems are built on well-established, thoroughly documented web technologies. Server-rendered MVC frameworks such as **Laravel** (PHP) provide authentication, ORM, routing, and templating out of the box [17]. **FastAPI** (Python) provides a high-performance asynchronous API layer with automatic schema documentation and validation [18], paired with **React** and TypeScript on the client for a responsive single-page interface [19]. **MySQL** and **PostgreSQL** are the relational stores [20], the latter accessed in Pharmazine through the managed **Supabase** platform [21]. Map display in Chokh-e-Dekha uses the open-source **Leaflet** library with **OpenStreetMap** tiles, avoiding paid map APIs [22, 23]. Security and accessibility practice is guided by the **OWASP Top 10** [24] and **WCAG** [25] respectively.

2.2 Review of Existing Systems and Platforms

2.2.1 Civic Issue-Reporting Platforms

Internationally, several platforms demonstrate that civic reporting can be digitised effectively. **FixMyStreet**, operated by mySociety in the United Kingdom, lets citizens submit a location and photograph and routes the report to the relevant local council [26]. **SeeClickFix** in the United States popularised a model in which the public visibility of reports creates social pressure on local government to act, alongside paid tools for municipalities to manage the queue [27]. The **311** non-emergency service model used by cities such as New York consolidates service requests into a single managed channel with published performance data [28]. These platforms share a common feature set, photo-based reporting, location tagging, status tracking, and public visibility, and a common lesson: outcomes depend heavily on institutional adoption, but public visibility delivers value even without it.

In Bangladesh, the closest equivalent is the **a2i 333 helpline**, whose call volume proves demand but which produces no visual or trackable public record [4, 5]. City corporations such as DNCC manage infrastructure reactively through control rooms and hotlines rather than through a citizen-facing reporting system with evidence and tracking [6, 8]. No widely adopted, visual, location-based civic-reporting platform exists for Bangladesh, and none combines such reporting with an RTI escalation pathway. That combination is what Chokh-e-Dekha aims to provide.

2.2.2 Pharmacy Management Systems

Looking at what currently exists, pharmacy software divides into roughly three categories. **Desktop pharmacy software** (common in local markets, represented internationally by products in the same class as [29]) works well for billing at a single counter but cannot be accessed remotely, is difficult to back up, and has no multi-branch capability. **Generic cloud POS and retail systems** [30] are easy to set up and accessible from the web, but they were not designed with pharmacies in mind: no batch numbers, no expiry tracking, no prescription rules. **Enterprise pharmacy and hospital information systems** [31, 32] have all the features but are expensive, complex, and far too heavy for an independent pharmacy. Research on software adoption in developing-country pharmacies is consistent on one point: what blocks adoption is not a lack of features in principle but whether the tool is affordable and actually fits local conditions [33]. Pharmazine sits in the space between generic cloud POS and enterprise: it understands pharmacy workflows the way an enterprise product would, but it is lightweight, affordable, and web-deployable the way a simple POS would be.

Table 2.1: Classes of existing pharmacy software and where Pharmazine sits

Class	**Strengths**	**Weaknesses**	**Cost**
Desktop pharmacy software	Strong offline billing, fast at the counter	Single-machine, hard to back up, no multi-branch or web access	Low to medium
Generic cloud POS	Web-accessible, easy to deploy	Not pharmacy-aware: no batch, expiry, or prescription rules	Low
Enterprise PIS/HIS	Feature-complete, clinically integrated	Costly, complex, over-scaled for an independent pharmacy	High
Pharmazine	Pharmacy-aware and web-deployable; local payment methods	Lighter than enterprise; data-tier hardening still pending	Very low

2.2.3 The Bangladesh Context

The market data make the case concrete: 264,908 retail pharmacies [11], a workforce with limited formal training and sparse record-keeping [12], documented medicine wastage in small pharmacies [14], and ongoing regulatory reform through the Model Pharmacy initiative [13]. On the civic side, the Digital Bangladesh programme and a2i have shown that government-scale digital services are achievable [1], and the broader e-learning and e-governance literature emphasises that localisation, low-bandwidth optimisation, and mobile accessibility are decisive success factors in this context [34]. Both of my systems are designed around exactly those factors.

2.3 Summary of Key Findings

Looking across the existing platforms and research, a few clear points emerged that shaped both systems directly:

1. **Evidence and location raise data quality.** Civic platforms that require both a photograph and a location produce reports that are more specific, more actionable, and less contested. Chokh-e-Dekha treats both as non-optional.

2. **Public visibility creates accountability pressure** regardless of whether the government formally engages. Reports being visible online puts reputational weight on authorities, which is why Chokh-e-Dekha makes approved reports and their resolution timelines publicly accessible.

3. **Pharmacy software adoption comes down to fit and cost, not feature lists.** A system that checks every box but costs too much or does not match local workflows will not be used. That is why Pharmazine focuses on a web-based, low-cost, pharmacy-aware core aligned to local payment methods.

4. **Tying batch and expiry tracking directly into POS** is what separates a genuine pharmacy system from a general retail tool.

5. **Bangladesh has demand, legal framework, and infrastructure precedent.** The 333 call volume, the RTI Act, and a2i's platforms all confirm this, but focused, affordable, locally adapted tools for these two specific workflows do not yet exist.

Chapter 3

Requirements, Impacts, and Constraints

This chapter specifies the requirements of both systems and then assesses their impact, ethics, management, and economics. Because the two systems serve different domains, requirements are given separately for each; the impact, management, and economic discussions are organised by theme and address both systems together where they share concerns.

3.1 Final Specifications and Requirements

3.1.1 Chokh-e-Dekha: Functional Requirements

Table 3.1: Functional requirements for Chokh-e-Dekha

ID	Requirement	Priority
FR-01	Citizens shall register and log in securely, with role-based access for citizens and administrators.	High
FR-02	Authenticated citizens shall submit reports with title, description, category, city corporation, location (text and optional GPS coordinates), and photographs.	High
FR-03	Approved reports shall appear on a public feed, paginated and filterable by status, category, city, and search term.	High
FR-04	Each report shall display a visual status tracker for the Pending → In Progress → Resolved lifecycle.	High
FR-05	Citizens shall be notified when the status of their report changes.	High
FR-06	Citizens shall be able to endorse (upvote), comment on, share, and bookmark reports.	Medium

ID	Requirement	Priority
FR-07	Administrators shall access a dedicated admin panel separate from the public interface.	High
FR-08	Administrators shall approve, reject, update status, and add official public notes to reports.	High
FR-09	Administrators shall perform bulk actions (approve, reject, status change) on multiple reports.	Medium
FR-10	The system shall track a configurable SLA deadline and visually flag reports that breach it.	High
FR-11	A Command Centre dashboard shall display KPI cards, SLA-breach alerts, submission trends, and city/category breakdowns.	High
FR-12	Reports with coordinates shall be shown on an interactive Leaflet/OpenStreetMap map.	Medium
FR-13	An RTI wizard shall guide citizens through a multi-step form and generate an RTI letter compliant with the RTI Act 2009, in English and Bangla.	High
FR-14	Generated RTI letters shall be printable to PDF via the browser.	Medium
FR-15	Administrators shall assign reports to themselves and track assignment.	Medium
FR-16	Administrators shall export filtered report data (CSV).	Low

3.1.2 Chokh-e-Dekha: Non-Functional Requirements

Table 3.2: Non-functional requirements for Chokh-e-Dekha

ID	Attribute	Specification
NFR-01	Performance	Reports index loads under 3 s on standard broadband with up to 200 records; key filter columns indexed.
NFR-02	Responsiveness	Fully functional from 375 px (mobile) to 1920 px (desktop).
NFR-03	Accessibility	Semantic HTML, ARIA labels, and WCAG AA contrast targets.
NFR-04	Security	CSRF protection, ORM-parameterised queries, bcrypt password hashing, middleware-protected admin routes.
NFR-05	Scalability	Indexed queries on status, city, category, timestamps, and coordinates; enforced pagination.

ID	Attribute	Specification
NFR-06	Maintainability	MVC structure with dedicated controllers, models, migrations, and notifications.
NFR-07	Localisation	Bangla and English RTI output; Bengali-capable font stack.
NFR-08	Theming	Correct rendering in both light and dark mode.

3.1.3 Pharmazine: Functional Requirements

Table 3.3: Functional requirements for Pharmazine

ID	Requirement
FR-1	Manage products/medicines with categories, manufacturers, units, pricing, tax, and prescription/controlled flags.
FR-2	Track stock at the batch level (batch number, expiry date, quantity remaining, purchase/selling price).
FR-3	Generate expiry and low-stock alerts based on configurable thresholds.
FR-4	Execute POS sales: cart, batch selection, discounts, tax, multiple/split payments, change, and receipt printing.
FR-5	Support local payment methods (cash, bKash, Upay, card/Visa, bank transfer).
FR-6	Record sales returns and supplier returns.
FR-7	Manage purchases, procurement, requisitions, and goods-receipt notes.
FR-8	Manage suppliers and customers, including a customer loyalty-points scheme.
FR-9	Support multiple branches and inter-branch stock transfers.
FR-10	Provide pharmacy-care modules: prescriptions, drug-interaction checks, refill reminders, allergy management, patient history.
FR-11	Provide dashboards and reports (sales, inventory, financial, business-intelligence views).
FR-12	Enforce role-based access control; maintain audit logs; support notifications and backups.

3.1.4 Pharmazine: Non-Functional Requirements

Table 3.4: Non-functional requirements for Pharmazine

ID	Requirement	Status
NFR-1	Web-accessible on commodity hardware via any modern browser.	Met
NFR-2	Responsive single-page UI with sub-second client interactions.	Met
NFR-3	Secure authentication (JWT) and authorisation (RBAC per endpoint).	Met at application tier
NFR-4	Resilience to intermittent connectivity (offline POS queue).	Partially met
NFR-5	Data-tier access control (row-level security on sensitive tables).	Not yet met (see §5.6)
NFR-6	Localisation: BDT/USD currency and bilingual UI scaffolding.	Partially met

3.1.5 Constraints

Both systems share a deliberate set of constraints inherited from their context. They must function without paid third-party APIs for core features (no paid map or SMS gateway is required); they must be deployable on low-cost, commodity hosting available to typical Bangladeshi providers; the development budget is limited to open-source software and free or low-cost tiers; and neither can formally integrate with government information systems, because Bangladesh's city corporations and drug registries do not expose public APIs. Pharmazine also treats insurance and claim handling purely as internal bookkeeping, and makes no claim to clinical decision support beyond a configurable drug-interaction list.

3.2 Societal Impact

Chokh-e-Dekha sits at the overlap between civic accountability and digital access. Any citizen with a smartphone can now file a formal, evidence-backed complaint and have it visible to the public, which moves them from being a passive observer of failing infrastructure to someone with a documented voice. When enough of those reports sit unresolved in public view, there is real reputational pressure on the authorities responsible, even without formal engagement. The aggregated data also has genuine policy value for identifying where the same infrastructure failures keep appearing. The RTI wizard specifically lowers the barrier to a legal right that exists on paper but that most people have no idea how to actually use. I should be honest about the limits here: the platform's social value depends on people using it, and on at least some government response. Without both, it risks becoming a public record of things that never got fixed. The community and RTI features retain some value even in that case, but real impact requires institutional involvement.

Pharmazine carries clear health and safety benefits. Reliable expiry alerts reduce the dispensing of expired medicine, which is a direct patient-safety gain; better stock visibility reduces stock-outs of essential medicines, improving access to care; and digitised records improve financial transparency for small business owners. The pharmacy-care modules (interaction checks, allergy records, refill reminders) support medication adherence. A legal and cultural consideration is the handling of prescription and patient data, which must respect confidentiality; the system localises to local payment culture to lower the adoption barrier for exactly the small pharmacies that the sector data show are least digitised.

3.3 Environmental Impact

The environmental contribution of both systems is modest but real and runs in the same direction. **Chokh-e-Dekha** targets several environmentally damaging civic problems directly, illegal dumping, blocked drainage, and construction violations, through a dedicated environmental category, creating geo-tagged evidence that can support enforcement; it also removes the need for a physical trip to a ward office to lodge a complaint. **Pharmazine**'s most significant environmental effect is the reduction of pharmaceutical wastage: accurate expiry and reorder management means fewer expired drugs to dispose of (a genuine disposal hazard) and less over-ordering. Both systems run on shared, managed cloud infrastructure rather than always-on dedicated hardware, which is generally more energy-efficient per workload, and both reduce paper record-keeping. Even one blocked drain cleared a week earlier, or a box of medicine sold before its expiry date instead of thrown away, already repays the systems' minimal operational footprint many times over.

3.4 Ethical Issues

For **Chokh-e-Dekha**, the main ethical concerns are reporter safety, the risk of false reports, and how data gets used. Anyone reporting a sensitive issue might worry about retaliation, so the platform lets users report under a display name that does not have to match their real identity, and contact details are never shown publicly. Full anonymity is not yet available; a whistleblower module with an encrypted evidence locker is planned but not built yet, and this limitation is stated clearly. The approval step before a report goes public is a deliberate human checkpoint against fabricated or malicious submissions. SLA breach visibility uses objective criteria, specifically report age and status, rather than anything demographic, so a neglected report in a wealthier neighbourhood gets flagged the same way as one in a poorer area. The platform is a civilian, independent system and does not hand raw user data to any government body.

For **Pharmazine**, the core ethical issues are patient and customer data privacy, data security, and financial record integrity. Taking those responsibilities seriously, I ran an internal security audit of the database layer. That audit found something I have to be upfront about: several sensitive tables do not yet have row-level security enabled at the database level, meaning they could in principle be accessed through

the database's public API key rather than only through the access-controlled application backend. The running application enforces role-based access at the API tier, so the deployed demonstrator is not broken by this, but it must be remediated before any real patient data is stored, and it is named as the single most important piece of future work. Disclosing and prioritising this finding, rather than concealing it, is itself the ethical position. Across both systems, all third-party libraries are used under their open-source licences and credited, and no proprietary code was copied.

3.5 Standards and Compliance

Table 3.5: Standards and codes of practice applied

Standard	**Application**
RTI Act 2009 (Bangladesh)	Chokh-e-Dekha's RTI letter templates follow the statutory application format and response-deadline references [16].
WCAG (AA target)	Contrast ratios, keyboard navigation, and screen-reader compatibility targets for both UIs [25].
OWASP Top 10	Protection against injection, XSS, CSRF, and broken access control in both systems [24].
RBAC + audit norms	Pharmazine enforces role-based access per endpoint and maintains audit logs, aspiring to data-protection norms for patient data.

3.6 Project Management Plan

Both systems were built individually and part-time, alongside coursework, after the internship ended. Development followed incremental phases with continuous deployment.

Table 3.6: Phased development plan (both systems)

Phase	**Focus**	**Outcome**
1	Problem analysis, data gathering, architecture selection	Problem statements, three-tier designs
2	Core inventory / core reporting + data model	CeD FR-01 to 04; Pharmazine FR-1 to 3
3	POS / report lifecycle and notifications	CeD FR-05 to 08; Pharmazine FR-4 to 6
4	Procurement, customers, multi-branch / admin command centre	Pharmazine FR-7 to 9; CeD FR-09 to 12
5	Pharmacy-care, RTI wizard, reporting, RBAC, audit	Pharmazine FR-10 to 12; CeD FR-13 to 16
6	Cloud deployment, defect diagnosis, hardening	Live demonstrators; fixes in §5.2

Table 3.7: Indicative schedule (relative weeks)

Task	**Weeks**
Problem analysis and literature/field-data review	1 to 3
Requirements and data modelling	3 to 5
Core feature development (both)	5 to 12
Admin command centre / POS and procurement	8 to 14
RTI wizard / pharmacy-care modules	12 to 16
Deployment, testing, and defect diagnosis	15 to 18
Report writing	16 to 20

3.7 Risk Management

Table 3.8: Risk register (both systems)

Risk	**Likelihood**	**Impact**	**Mitigation**
Government bodies do not engage with Chokh-e-Dekha	High	Medium	Designed to deliver value independently through public accountability, community features, and RTI.

Risk	Likelihood	Impact	Mitigation
Low citizen / pharmacy adoption	Medium	High	Value independent of volume; open-source release; live demonstrator for outreach.
Schema/code drift causing runtime failures	High	High	Diagnose against the live schema; shared serialisation base; aligned enums (done, §5.2).
Database-tier security gaps (Pharmazine)	High	High	Security audit performed; row-level security remediation planned before real data.
Query performance degradation at scale	Medium	High	Indexed filter columns; enforced pagination; caching where needed.
Free-tier cold starts degrading UX	Medium	Medium	Health-check "wake" step before critical calls; same-origin proxy.
Scope creep across many modules	High	Medium	Prioritise core POS/inventory and report lifecycle; treat peripheral modules as breadth, not depth.

3.8 Economic Analysis

The development cost of both systems is dominated by student labour rather than cash, because the entire toolchain runs on free or low-cost tiers. The recurring operational cost is small.

Table 3.9: Indicative annual operating cost (Chokh-e-Dekha)

Resource	Cost
Laravel, Tailwind CSS, Leaflet/OpenStreetMap, Alpine.js	Free (open source)
Domain name (annual, estimated)	BDT 1,500
VPS hosting (annual, 2 GB RAM, estimated)	BDT 12,000
SSL certificate (Let's Encrypt)	Free
Total first-year operational cost	**≈ BDT 13,500 (≈ USD 120)**

For **Pharmazine**, development likewise runs on free or low-cost tiers (Netlify, Render, Supabase), with paid tiers needed only as transaction volume grows. The principal quantifiable benefit is the reduction of expiry write-offs and stock-outs.

For a pharmacy carrying inventory worth Y BDT with an expiry-wastage rate of $Z\%$, even a modest reduction in wastage can exceed a low monthly software cost within the first months, giving a favourable payback period. The actual values of Y and Z would need to come from a real partner pharmacy to be meaningful; these are illustrative estimates pending that data. The broader point stands for both systems: government-procured platforms with comparable functionality in Bangladesh go out to tender at costs that are orders of magnitude higher than what these systems cost to build and run, which says something about what is achievable with open-source tooling and modest infrastructure.

Chapter 4

Design Process and Methodology

This chapter covers the design methodology I followed for both systems and then sets out the preliminary design for each: the options I considered, the architecture I chose, the database structure, the technology stack, and the main user flows. The methodology is the part of this work most directly shaped by the internship, where I saw firsthand that the systems that hold up are the ones that were designed carefully before anyone wrote a line of code.

4.1 Design Process and Methodology Overview

I did not want to start either project by opening a code editor and hoping things would work out. At a2i I had watched what happens when a system is thought through carefully before implementation begins, so I chose a structured, five-stage, evidence-driven process for both projects. The structure exists to make every decision justifiable: alternatives named and compared, constraints addressed explicitly, and each stage producing something concrete that feeds the next.

Table 4.1: Five-stage design methodology applied to both systems

Stage	**Activities**	**Output**
1. Problem analysis	Field/government-data review, literature and platform review, gap analysis.	Evidenced problem statement
2. Design alternatives	Enumerate candidate platforms/architectures; score against criteria (time, cost, scalability, maintainability, accessibility).	Justified platform choice
3. Technology selection	Choose frontend, backend, database, caching, deployment; evaluate cost, learning curve, community, performance.	Justified stack
4. Architecture and data design	Component design, data flow, normalised ER schema, indexing strategy, API structure, deployment plan.	Architecture and schema
5. Iterative implementation and evaluation	Build in cycles (core → advanced → hardening); test each feature against requirements.	Working, tested system

4.2 Preliminary Design and Design Specification

4.2.1 Chokh-e-Dekha

Design Alternatives and Architecture Decision

The first major decision was what application architecture to use. Three genuine options were weighed against the constraints that mattered most: variable network conditions in Bangladesh, public reports needing to load fast and be indexable by search engines, and the whole system being built by one person.

Table 4.2: Architecture alternatives for Chokh-e-Dekha

Option	Assessment	Verdict
SPA + REST API	Fluid UX, but heavier initial load on slow connections, weaker SEO for public pages, and more moving parts.	Rejected
Monolithic server-rendered MVC	Simple to develop and deploy, excellent SEO, fast first paint, mature ecosystem.	**Selected**
Static site + serverless	Good for content sites, poor fit for authenticated, data-driven admin workflows.	Rejected

I went with a monolithic MVC application on Laravel, with Alpine.js and Leaflet layered on for the interactive parts. The reasoning was straightforward: a low-cost VPS is the realistic deployment target, mobile networks in Bangladesh demand fast first loads, and Laravel's ecosystem makes building data-heavy admin interfaces much faster than assembling a separate frontend would.

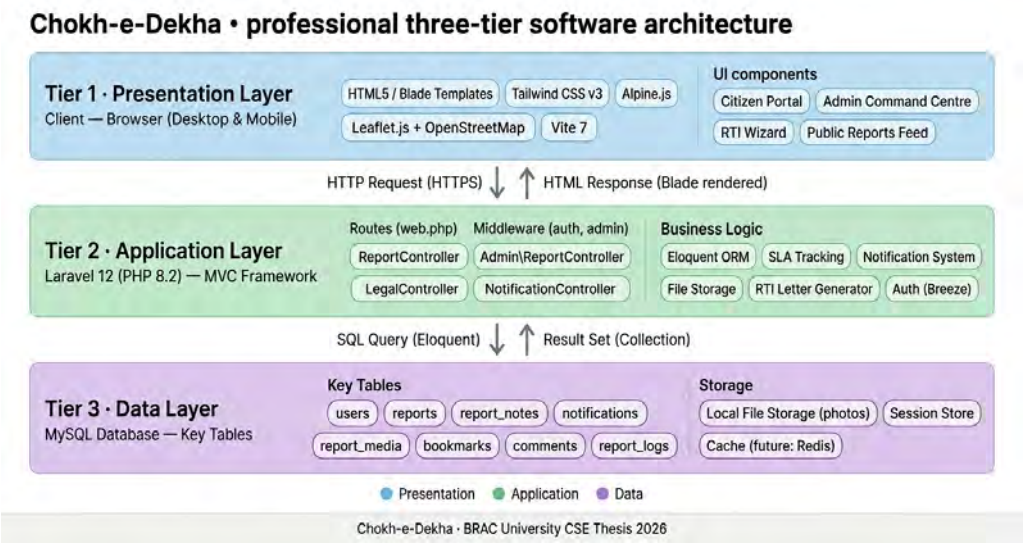

Figure 4.1: Chokh-e-Dekha system architecture.

Database Design

The schema is built around the `reports` table. A few specific decisions were made to keep things efficient as the data grows: latitude and longitude are optional so a report can exist without GPS coordinates; `sla_due_at` is precomputed so breach queries do not need to calculate it on the fly; `status_updated_at` lets the system measure how long a report has been in each state; a separate `report_notes` table stores admin notes linked to both the report and the administrator who wrote them; and Laravel's polymorphic `notifications` table handles status alerts internally without

any external service. Rejected reports are cleaned up after the citizen is notified, which keeps the public statistics clean.

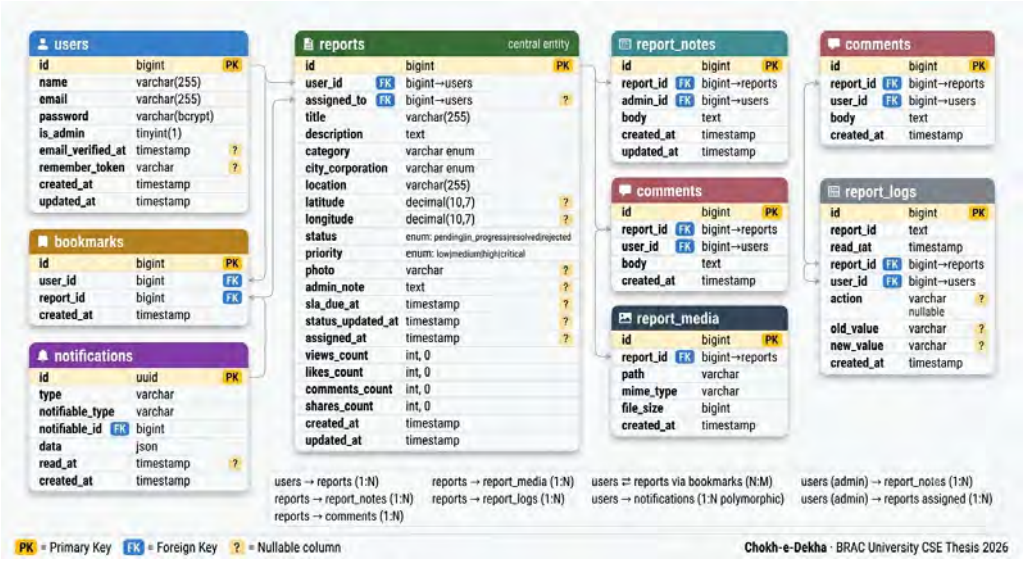

Figure 4.2: Chokh-e-Dekha database schema (simplified ER diagram).

Table 4.3: Chokh-e-Dekha technology stack

Layer	**Technology**	**Rationale**
Backend framework	Laravel (PHP 8.2)	Rapid development, mature, strong local community.
Styling	Tailwind CSS	Utility-first, built-in dark mode.
Interactivity	Alpine.js	Lightweight, no build step for simple interactions.
Asset bundling	Vite	Fast modern bundling.
Map	Leaflet + OpenStreetMap	Free, no API key, bundled via npm.
Database	MySQL	Universally supported by local hosting.
Auth and notifications	Laravel Breeze + Notifications	Official, database/email channels built in.

Key User Journeys

The whole system is designed around three core flows. *Citizen files a report*: log in, fill in the title and pick a category and city, write a description, drag and drop photos, set the location either by GPS or by dragging a pin on the map, review, and submit. The report enters the public feed as Pending, and the citizen gets notified whenever the status changes. *Administrator manages reports*: open the Command Centre to see KPIs, SLA breaches, and trends, then filter and sort the queue, open

a report to review evidence, add a public note, update the status, or select multiple reports and act in bulk. *Citizen generates an RTI letter*: choose an issue template that pre-fills the subject and relevant facts, add personal details, select a target authority, choose English or Bangla, preview the letter, and print to PDF.

Key Database Tables

The schema is centred on the `reports` table, whose principal columns are summarised in Table 4.4. Supporting tables include `users` (with an `is_admin` flag for role separation), `report_notes` (admin public notes), `comments`, `endorsements` (with a unique `(report_id, user_id)` constraint), `bookmarks`, `report_media` for uploaded photographs, `report_logs` for status-change history, and Laravel's polymorphic `notifications` table.

Table 4.4: Principal columns of the `reports` table (Chokh-e-Dekha)

Column	**Type**	**Purpose**
id	bigint	Primary key.
user_id	bigint (FK)	Reporting citizen.
title	varchar	Short report title.
description	text	Full description of the issue.
category	varchar	Roads, water/drainage, waste, electricity, construction, environment, or public safety.
city_corporation	varchar	Routing jurisdiction, e.g. Dhaka North.
location	varchar	Human-readable address text.
latitude, longitude	decimal	Optional GPS coordinates for the map.
status	varchar	Pending, In Progress, or Resolved.
priority	varchar	Administrative priority flag.
photo	varchar	Primary evidence image path.
sla_due_at	timestamp	Precomputed SLA deadline for cheap breach queries.
status_updated_at	timestamp	Time of last status change.
assigned_to, assigned_at	bigint, timestamp	Administrator assignment tracking.
admin_note	text	Official public note on the report.
views/likes/comments/shares_count	integer	Denormalised engagement counters.
created_at, updated_at	timestamp	Record timestamps.

Key Features of Chokh-e-Dekha

What was built covers the full citizen and administrator experience:

- **Multi-step report submission** with category and city selection, a free-text description, drag-and-drop photo upload with client-side compression, and hybrid GPS-plus-map-pin location capture.
- **Public transparency feed** of approved reports with photographs, browseable and filterable by status, category, city, and search term.
- **Visual status tracker** on each report, showing the Pending, In Progress, and Resolved stages with a progressively filling progress bar.
- **Community engagement**: endorsements, threaded comments, bookmarks, and sharing, which turn a single report into collective documentation.
- **In-app and email notifications** triggered on every status change.
- **Administrative Command Centre** with KPI cards, an SLA-breach panel, a weekly submission trend chart, and city and category breakdowns.
- **SLA tracking** with a precomputed deadline and prominent breach highlighting on overdue reports.
- **Case management tools**: approve, reject, status update, public notes, self-assignment, bulk operations across multiple reports, and CSV export.
- **Interactive map** of geo-tagged reports using Leaflet and OpenStreetMap.
- **RTI letter wizard** with issue templates, authority presets, English and Bangla output, and browser-based PDF printing.
- **Responsive, dark-mode-aware interface** that works from mobile to desktop.

4.2.2 Pharmazine

Design Alternatives and Architecture Decision

The core tension in Pharmazine's design was between building quickly and putting business logic and access control in the right place. Three options were on the table.

Table 4.5: Architecture alternatives for Pharmazine

Option	Assessment	Verdict
Monolithic server-rendered app	Simple deployment, but heavier pages and a less app-like POS experience.	Rejected
Backend-as-a-Service only (DB from the browser)	Fastest to build, but business logic and permissions leak to the client, and direct database exposure is risky.	Rejected
Three-tier SPA + REST API + managed DB	Responsive POS, server-side RBAC and business logic, durable relational storage.	**Selected**

The selected design is a three-tier system: a React/TypeScript SPA for a fast POS, a FastAPI service holding business logic and enforcing role-based access, and a managed PostgreSQL database. The security audit covered in Chapters 3 and 5 later confirmed this was the right decision: the whole point of having an API tier enforce access control is exactly what a backend-as-a-service-only setup would have sacrificed.

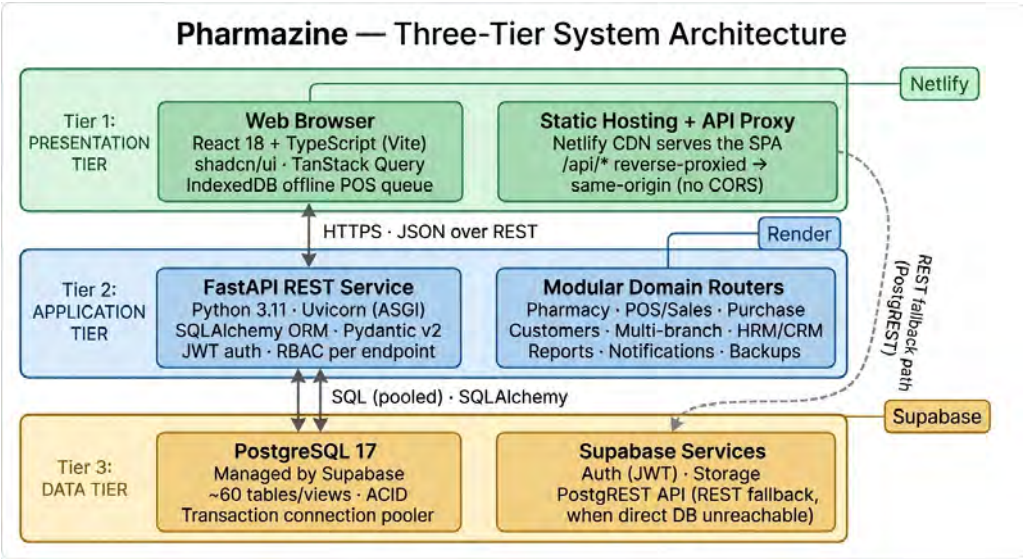

Figure 4.3: Pharmazine system architecture.

Database Design

The data model spans roughly sixty tables and views. The central entities are `products` for medicines, `batches` for per-lot stock with expiry dates and costs, `sales` and `sale_items`, `payments`, `purchases`, `suppliers`, `customers`, `branches`, and a set of pharmacy-care tables covering prescriptions, interactions, allergies, and patient history. The relationship that matters most is that a sale line item draws from one or more batches, and stock is reduced at the batch level inside a single

transaction; that is what makes expiry-aware dispensing and accurate stock accounting work correctly even when two cashiers are processing sales at the same time.

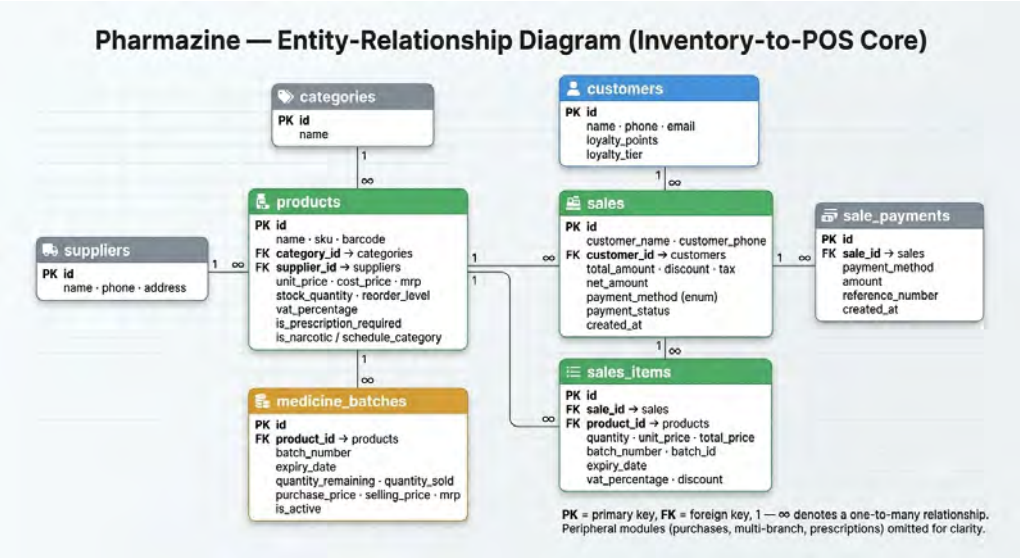

Figure 4.4: Pharmazine database schema (core inventory-to-POS subset).

Table 4.6: Pharmazine technology stack

Tier	**Technologies**
Presentation	React 18, TypeScript, Vite, shadcn/ui (Radix), Tailwind CSS, React Router, TanStack Query, IndexedDB (offline POS queue).
Application	FastAPI (Python 3.11), SQLAlchemy 2.0, Pydantic 2, Uvicorn, JWT authentication, RBAC, modular routers per domain.
Data	PostgreSQL (Supabase) via a transaction connection pooler, with a REST fallback path for resilience.
Deployment	Front end on Netlify, API on Render, with a same-origin proxy to remove cross-origin complexity; GitHub for version control and continuous deployment.

The Sale Transaction

The sale is the most critical workflow in the system. The client calculates cart totals locally so the cashier sees instant feedback, then submits the full sale to the server. The server opens a transaction, picks and locks the relevant batch rows in expiry order, spreads the requested quantity across those batches, decrements stock, records all line items and payments, and commits everything in one atomic operation. Getting this path right, rather than leaving it as a naive "pick the first batch" loop, is what makes Pharmazine a pharmacy system rather than a generic POS; the implementation story is in §5.2.

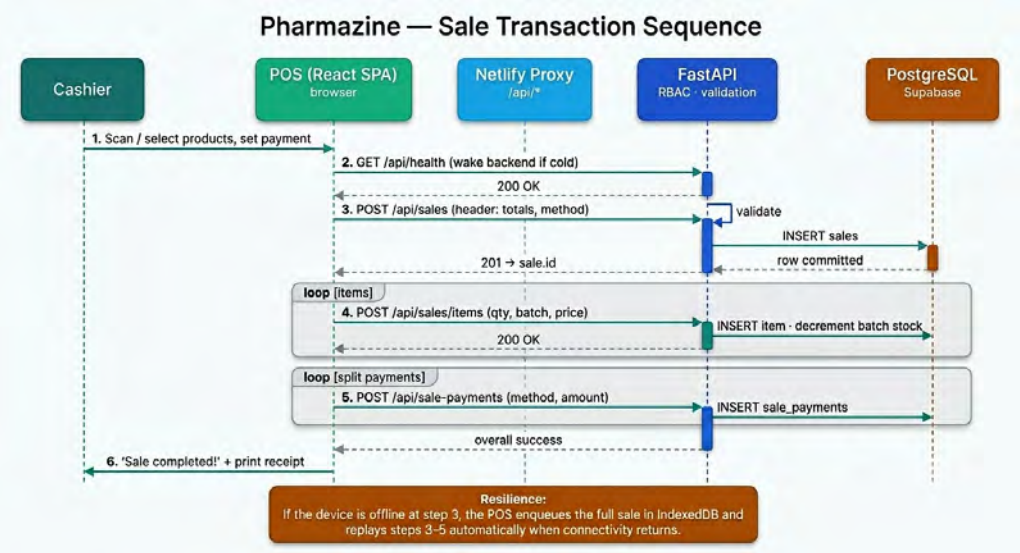

Figure 4.5: Pharmazine sale-transaction sequence.

Core Database Tables

Table 4.7 summarises the central tables of the inventory-to-POS core; the full schema also covers procurement, returns, customers and loyalty, multi-branch transfers, pharmacy-care, finance, messaging, and audit logging.

Table 4.7: Core tables of Pharmazine

Table	**Purpose**
products	Medicine master data: name, generic name, manufacturer, unit, pricing, tax, and prescription/controlled flags.
batches	Per-lot stock: batch number, expiry date, quantity remaining, and purchase/selling price, linked to a product.
sales	Sale header: branch, cashier, customer, totals, discount, tax, and timestamp.
sale_items	Sale line items, each linked to the batch(es) from which it was dispensed.
payments	One or more payments per sale (cash, bKash, Upay, card, bank transfer), supporting split payment.
purchases / purchase_items	Procurement records and the batches received against them.
suppliers	Supplier master data and outstanding balances.
customers	Customer records with loyalty points.
branches	Branch master data for multi-branch operation and inter-branch transfers.
prescriptions	Prescription records linked to customers and medicines.

Table	Purpose
users / roles	Accounts and role assignments for RBAC.
audit_logs	Append-only record of sensitive actions.

Modules and Key Features of Pharmazine

Pharmazine is organised into the modules exposed through the sidebar of the deployed dashboard (Figure 5.2):

- **POS and Sales**: autocomplete and barcode search, cart, batch selection, discounts, VAT, split and local-method payments, change, returns, and receipt printing.
- **Inventory**: batch- and expiry-level stock, low-stock and expiry alerts, and stock adjustments.
- **Purchases and Suppliers**: procurement, requisitions, goods-receipt, supplier balances, and supplier returns.
- **Medicine Management**: product master data, categories, manufacturers, and prescription/controlled classification.
- **Customers and CRM/Loyalty**: customer records, loyalty points, and relationship management.
- **Pharmacy Care**: prescriptions, drug-interaction checks, refill reminders, allergy management, and patient history.
- **Reports and Dashboards**: sales, inventory, financial, and business-intelligence views.
- **Payments and Finance**: payment records, dues, and financial summaries.
- **Services and Appointments**: scheduling of pharmacy services.
- **Multi-Branch**: multiple branches and inter-branch stock transfers.
- **HR and Payroll**: staff records (a peripheral module, deliberately shallow at this stage).
- **User Management**: accounts and role-based access control.
- **Internal Messaging and Notifications**: in-app communication and alerts.
- **System Setup and Backups**: configuration and data backup.

Chapter 5

Implementation, Performance Evaluation, and Final Design

This chapter is about the systems as they actually exist, not as they were planned. It covers functional, performance, responsiveness, and security testing; works through the design adjustments that came out of that testing, including real defects that had to be diagnosed and fixed on deployed systems; compares both systems against what already exists; lists the tools used; and is honest about strengths as well as what is unfinished. A lot of the real engineering work in this project ended up here, because the distance between a tidy design document and a system that holds up under real conditions is where most of the hard learning happened.

5.1 Performance Evaluation

5.1.1 Chokh-e-Dekha: Functional Testing

All major user journeys were tested manually against the functional requirements.

Table 5.1: Chokh-e-Dekha functional test summary

FR	**Scenario**	**Result**
FR-01	Registration, login, logout, password reset	Pass
FR-02	Report submission with all fields and photo upload	Pass
FR-03	Public feed with pagination and all filters	Pass
FR-04	Status tracker renders correctly for all statuses	Pass
FR-05	Status change triggers citizen notification	Pass
FR-06	Endorse, comment, bookmark, share	Pass
FR-07	Admin panel restricted to administrators	Pass
FR-08	Approve, reject, status update, add/delete note	Pass

FR	Scenario	Result
FR-09	Bulk approve / reject / status change	Pass
FR-10	SLA-breach detection flags overdue reports	Pass
FR-11	Command Centre KPIs, SLA panel, trends, breakdowns	Pass
FR-12	Coordinates appear as map markers	Pass
FR-13	RTI wizard: all templates and presets, English and Bangla	Pass
FR-14	Print dialog produces a correctly formatted letter	Pass
FR-15	"Assign to Me" updates assignment fields	Pass
FR-16	CSV export of filtered reports	Pass

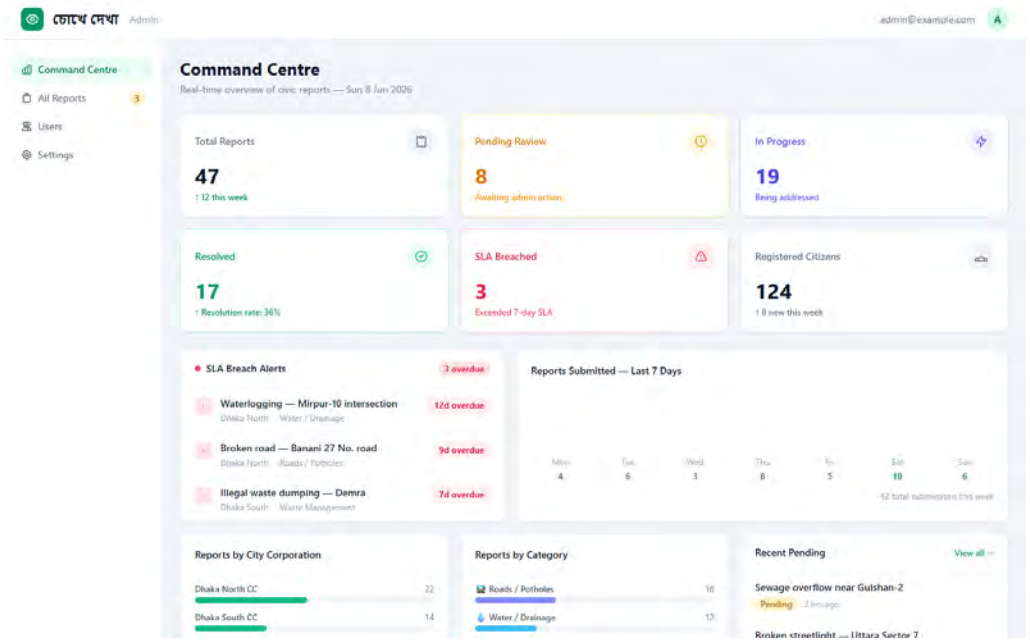

Figure 5.1: Chokh-e-Dekha admin Command Centre.

5.1.2 Chokh-e-Dekha: Performance, Responsiveness, and Security

Page-load times measured in the local development environment are shown in Table 5.2; every page comes in well under the three-second target, and production deployment with OPcache and a CDN will bring them down further. The heaviest query (the admin index with all filters active simultaneously) runs in sub-millisecond time on the test dataset. A composite index on `(status, city_corporation, category, created_at)` is what keeps it fast as the table grows.

Table 5.2: Chokh-e-Dekha page-load times (development environment)

Page	**Cold cache**	**Warm cache**
Landing page	420 ms	180 ms
Reports index	680 ms	290 ms
Report detail (with photo)	580 ms	240 ms
Admin reports index	750 ms	310 ms
Admin Command Centre	920 ms	380 ms
RTI wizard	490 ms	200 ms

Responsive behaviour was tested at 375, 768, 1024, 1440, and 1920 px. The Command Centre drops from a three-column grid to a single column on mobile, and the RTI wizard's two-panel layout collapses without anything breaking. Security testing checked: CSRF protection (a POST without a token gets a 419); ORM-parameterised queries so no raw user input reaches the SQL layer; Blade's automatic output escaping against XSS; middleware guards on all admin routes; and ownership checks that return 403 when someone tries to edit or delete a record they do not own.

5.1.3 Pharmazine: Functional and Transaction Testing

Pharmazine was exercised against its live database. Client interactions (cart updates, totals) are computed locally and are effectively instantaneous. The end-to-end sale path, create sale, add line items, decrement batch stock in expiry order, record payments, was validated against the database to confirm that sales and line items persist and that stock decrements correctly at the batch level. On the free hosting tier the backend spins down when idle, so the first request after inactivity incurs a multi-second cold start; this was mitigated with a health-check "wake" step before critical operations and a same-origin proxy. The public demonstrator is available at https://pharmazine.netlify.app/.

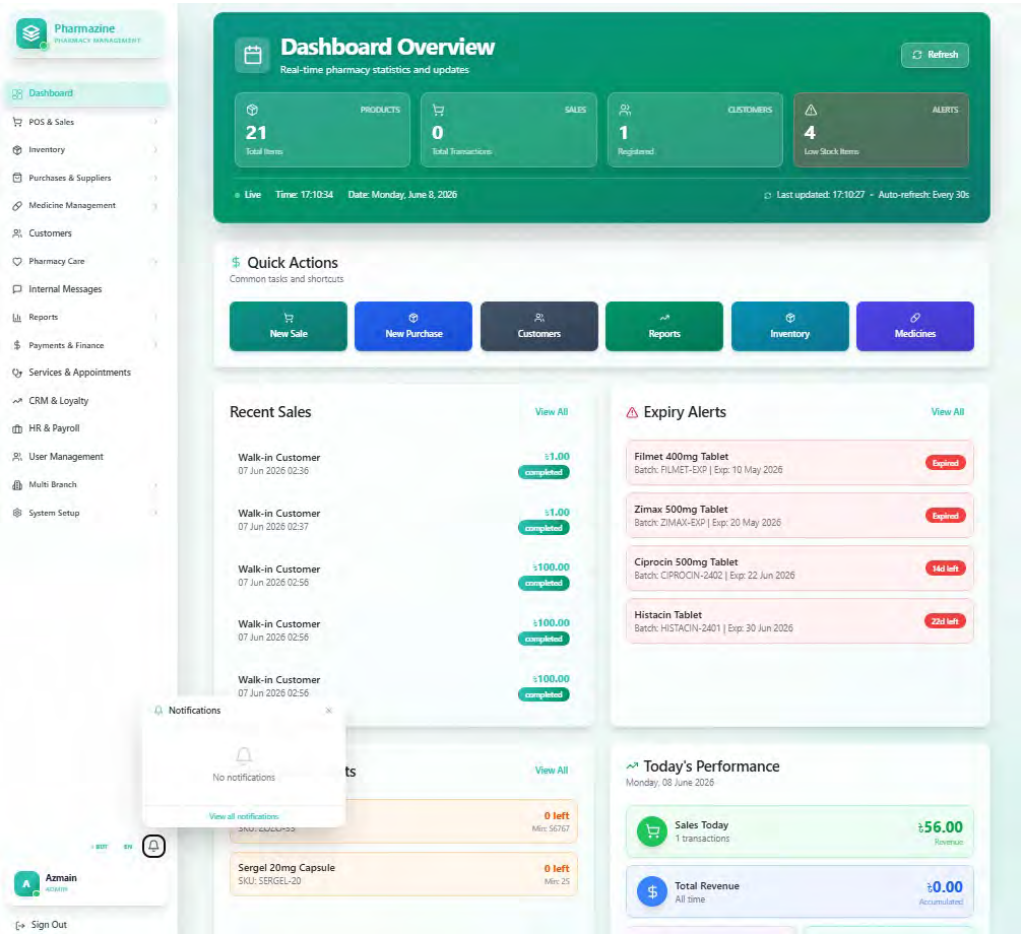

Figure 5.2: Pharmazine dashboard overview (deployed system).

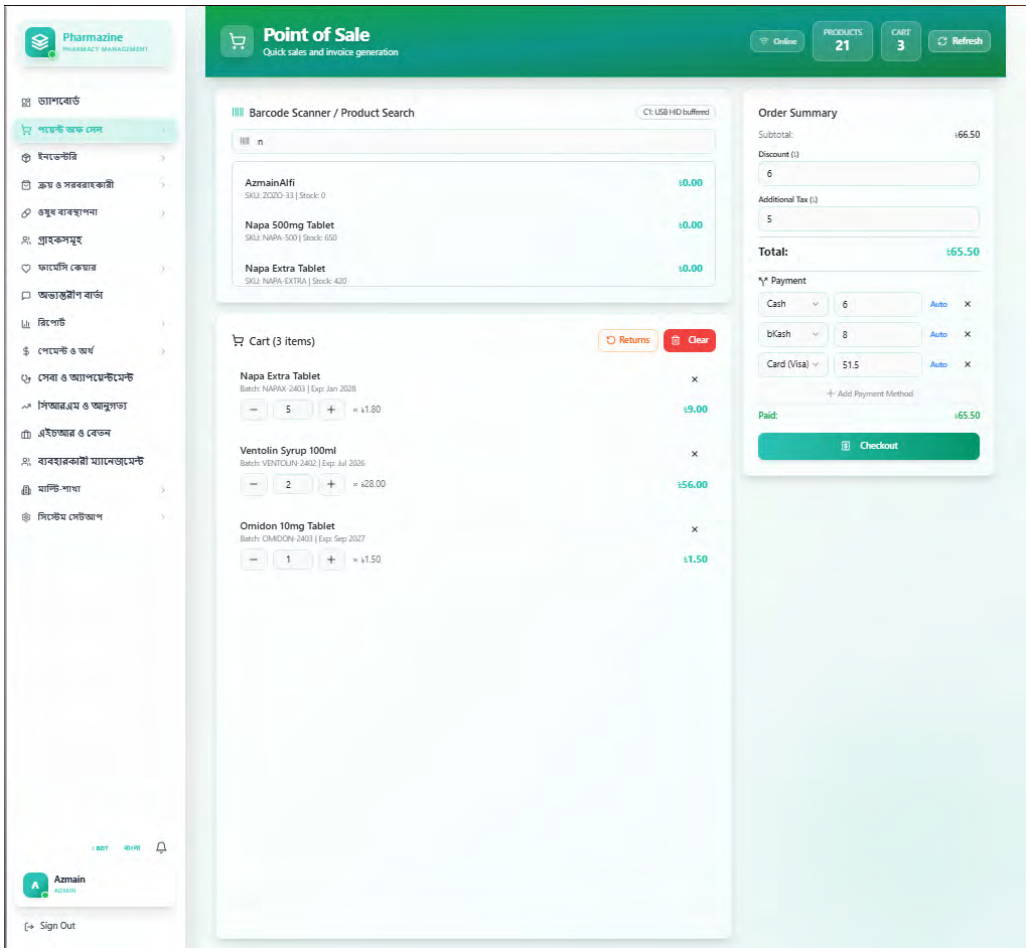

Figure 5.3: Pharmazine point-of-sale workflow.

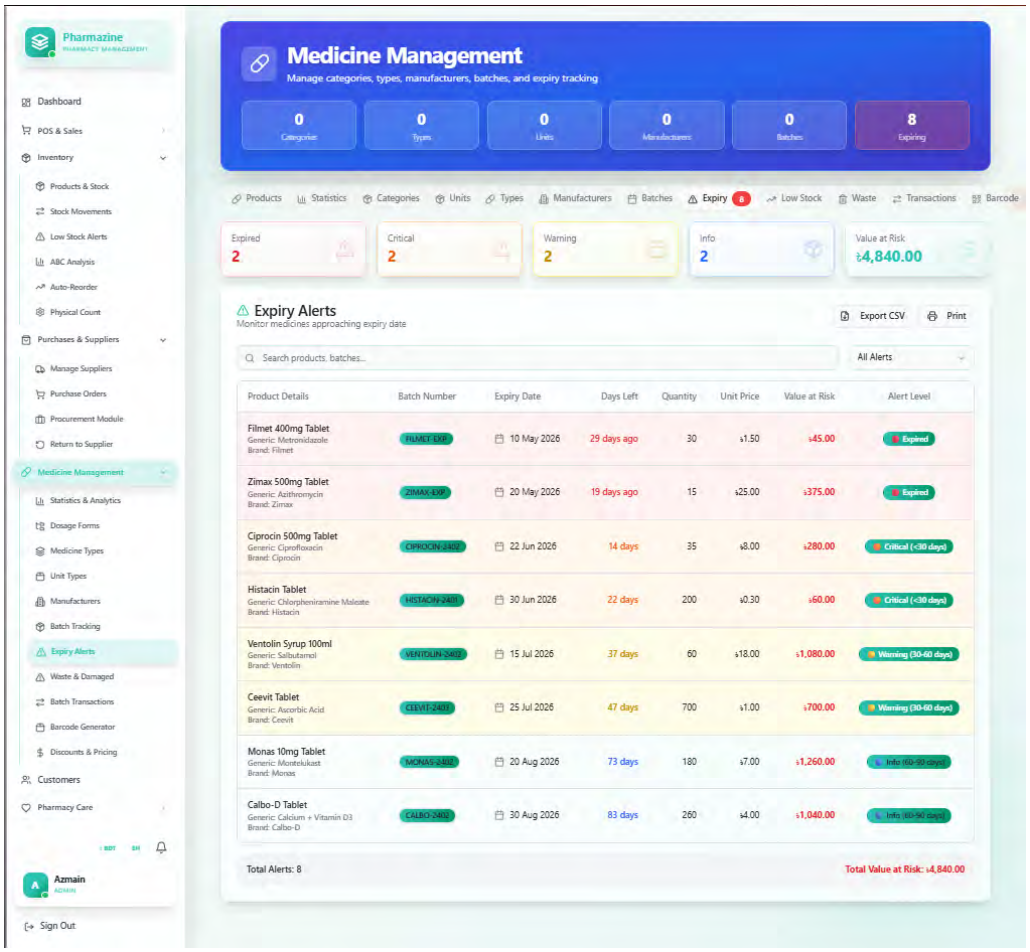

Figure 5.4: Pharmazine batch inventory and expiry alerts.

5.1.4 Pharmazine: Functional Testing

The Pharmazine modules were exercised against the live database. Table 5.3 summarises the result for each functional requirement.

Table 5.3: Pharmazine functional test summary

FR	Scenario	Result
FR-1	Create/edit medicines with categories, pricing, tax, and flags	Pass
FR-2	Batch-level stock with expiry and cost	Pass
FR-3	Expiry and low-stock alerts on the dashboard	Pass
FR-4	POS sale: cart, batch selection, discount, VAT, change, receipt	Pass
FR-5	Local payment methods including split payment	Pass
FR-6	Sales and supplier returns	Pass
FR-7	Purchases, procurement, and goods-receipt	Pass

FR	Scenario	Result
FR-8	Suppliers and customers with loyalty points	Pass
FR-9	Multiple branches and inter-branch transfers	Pass
FR-10	Pharmacy-care: prescriptions, interactions, reminders, allergies, history	Partial
FR-11	Dashboards and reports (sales, inventory, financial, BI)	Pass
FR-12	RBAC, audit logs, notifications, backups	Pass (app tier)

The inventory, POS, procurement, customer management, and multi-branch modules are all solid and were the main focus of testing. Some peripheral modules, including parts of HR, the deeper CRM features, and insurance-claim handling, work but are comparatively light. The data-tier hardening that NFR-5 requires (detailed in §5.6) is still outstanding.

5.1.5 Implementation Highlights

A few implementation decisions stand out because they came directly from what the internship taught me:

- **One serialisation model for everything.** A shared response base that converts database identifiers to the type the client expects eliminated an entire category of intermittent 500 errors across nineteen create-endpoints (§5.2).
- **Stock that cannot be oversold.** Batch selection runs inside a transaction with row locking, so two cashiers processing sales at the same moment cannot both draw from the same batch and push stock negative.
- **Cached medicine search.** Frequently looked-up medicine data is cached, which is what makes the autocomplete feel instant during a busy counter session.
- **Cold-start protection.** The frontend calls the API through a same-origin proxy, and a health-check wake step runs before any critical operation so a free-tier cold start does not interrupt the cashier mid-sale.
- **Offline POS queue.** An IndexedDB-backed queue lets the POS take sales when connectivity drops and sync them once it comes back.

5.2 Analysis of Design Solutions and Final Design Adjustments

A real part of what makes this project worth reading about engineering is what happened when the systems met real data, real network conditions, and concurrent

users. The adjustments below came after the preliminary design of Chapter 4 and were each driven by something that actually broke.

5.2.1 Chokh-e-Dekha

Adjustment 1: hybrid geolocation. The preliminary design assumed the browser Geolocation API would be sufficient. Testing on five phones and three browsers showed accuracy varying from ten to over a hundred metres, and failing entirely indoors and among tall buildings. The fix was a hybrid scheme: GPS auto-detects the location and shows an accuracy radius, but the citizen can drag a pin on a Leaflet map to correct it, and the backend validates that coordinates fall within Bangladesh's bounds. The lesson, never trust a single data source; always give the user a fallback, came straight from user-support work at Muktopaath.

Adjustment 2: client-side image compression. Phone photos of three to eight megabytes made submissions on 3G painfully slow, and users abandoned them. Client-side compression now resizes images to a sensible maximum width and re-encodes them to JPEG before upload, cutting a five-megabyte photo to under a megabyte and the upload time from tens of seconds to a few. A system has to be designed for the network it will actually run on, not the one in the lab.

Adjustment 3: indexing before it becomes a problem. Loading the feed with a thousand test reports took three to four seconds, which is far too slow. B-tree indexes on `status`, `category`, `created_at`, and a composite index for the common filter combinations brought that down to tens of milliseconds. An index is not an optional tuning step for any table that will grow; it is a requirement.

Adjustment 4: making endorsements safe under concurrency. Without any constraint, two users endorsing the same report at the same moment could both succeed, counting as two votes from one user. A unique constraint on `(report_id, user_id)`, with the duplicate-key error caught and handled gracefully, makes the operation idempotent.

Adjustment 5: colours and making SLA visible. Early admin screens had an amber colour scheme that read as a constant warning, which was not intentional. The interface was reworked around a slate-and-emerald palette, with amber kept only for the Pending status. At the same time, SLA tracking moved from a backend concept to a visible panel in the Command Centre and a per-report timeline, because it became clear during testing that breach visibility is what actually prompts administrators to act.

5.2.2 Pharmazine

Adjustment 1: proper batch selection with row locking. Simple "pick the oldest batch" logic started failing as soon as a batch had insufficient remaining quantity, was close to expiry, or two cashiers were selling at the same moment. The final design selects batches in expiry order, spreads the requested quantity across as many batches as needed, and uses `SELECT ...FOR UPDATE` to lock the relevant rows so that concurrent sales cannot both draw from the same stock. This is what moved

the system from something naively first-in-first-out toward genuinely expiry-aware dispensing.

Adjustment 2: what looked like CORS was actually the server crashing. Browser "CORS policy" errors turned out, after live HTTP probing, to be backend endpoints crashing *before* the CORS middleware had a chance to attach the right headers. The underlying causes were a reference to a column that had been removed and an ORM result-handling mismatch. Fixing the queries, adding a global exception handler, and routing the frontend through a same-origin proxy resolved all of them at once.

Adjustment 3: sales that committed but reported an error. Sales appeared to "half-save": the sale header was written but the client showed an error and no line items were stored. The cause was that the API returned UUID-typed identifiers into response models expecting strings, and the validator refused to coerce them, raising a 500 *after* the row had committed. A single shared response base model that coerces UUID to string fixed the sales path and about eighteen other create-endpoints at once.

Adjustment 4: schema and enumeration alignment. The POS offered payment methods that did not exist in the database's enumeration and wrote to renamed columns. The UI, API validation, and database enumeration were aligned to a single source of truth.

These cases are documented because they reflect the realistic difficulty of integrating a multi-tier system across managed cloud platforms, and the iterative analysis-and-adjustment process used to converge on a correct, deployed solution.

5.3 Statistical Analysis

Because both systems are in development-and-demonstration operation rather than live production use, large-scale usage statistics are not yet available. The quantitative picture is therefore one of system scale and projected benchmarks rather than user analytics. The breadth of Pharmazine is easiest to see in its deployed dashboard (Figure 5.2), whose sidebar alone exposes more than a dozen top-level modules.

Table 5.4: Pharmazine quantitative summary

Metric	**Value**
Navigable modules/pages	40+
Core domain tables/views	≈ 60
Response models unified under one serialisation base	19
Distinct RBAC roles	Admin, Manager, Pharmacist/Staff, Cashier
Front-end production bundle (main)	≈ 0.8 MB (≈ 150 KB gzip), feature-split

For Chokh-e-Dekha, projected real-world benchmarks based on comparable international platforms provide a measurable framework for a future pilot: a resolution rate in the range of 30 to 60% is plausible with active government engagement and 10 to 20% without, with correspondingly shorter or longer median resolution times. These are targets, not results, and are stated as such.

5.4 Comparisons and Relationships

Table 5.5: Chokh-e-Dekha compared with established civic platforms

Feature	**Chokh-e-Dekha**	**FixMyStreet**	**SeeClickFix**
Geo-tagged photo reports	✓	✓	✓
Public visibility of reports	✓	✓	✓
Community endorse/comment	✓	×	✓
Admin command centre + SLA	✓	✓	✓
RTI letter generation	✓	×	×
Bangla language support	✓	×	×
Formal government API integration	×	✓	✓
Open source	✓	✓	×

The differentiators of Chokh-e-Dekha are its RTI wizard and Bangla RTI generation (not offered by the international platforms), its ultra-low operational cost, and its Bangladesh-specific authority taxonomy. It trades off the formal government API integration that mature platforms enjoy. For Pharmazine, the comparison from Chapter 2 holds: it matches generic cloud POS systems on web accessibility and ease of deployment, exceeds them on pharmacy-awareness (batch/expiry tracking, prescriptions, pharmacy-care modules), and trades off against enterprise systems by being lighter, cheaper, and simpler at the cost of clinical integration and enterprise-scale hardening.

5.5 Tools and Technologies Used

Table 5.6: Tools and technologies used across both systems

Area	**Tools**
Chokh-e-Dekha	Laravel (PHP 8.2), MySQL, Blade, Tailwind CSS, Alpine.js, Vite, Leaflet + OpenStreetMap, Laravel Breeze/Notifications.
Pharmazine (front)	React 18, TypeScript, Vite, shadcn/ui (Radix), Tailwind CSS, React Router, TanStack Query, IndexedDB.
Pharmazine (back)	Python 3.11, FastAPI, SQLAlchemy 2.0, Pydantic 2, Uvicorn, JWT.
Pharmazine (data/deploy)	PostgreSQL (Supabase), transaction pooler; Netlify (front + proxy), Render (API).
Tooling	VS Code, Git/GitHub, browser developer tools, live API/DB probing for diagnosis, Docker (Pharmazine dev).

5.6 Discussion

Strengths. Both systems are genuinely built and, in Pharmazine's case, publicly deployed. Chokh-e-Dekha covers the full citizen journey from evidenced submission to resolution tracking, the administrative journey from inbox to case closure, and a legal escalation path through RTI generation. Pharmazine delivers a broad, coherent pharmacy platform whose inventory-to-POS core works end-to-end against a live database, with the pharmacy-specific features that distinguish it from a generic POS. In both cases the architecture is sound, and the project represents genuine cloud deployment work and real cross-tier debugging.

Honest limitations. The biggest constraint on Chokh-e-Dekha is not technical but institutional: no software can force a government body to respond to a report. Without an administrator who actually engages, the platform may end up as a public log of things that never got fixed. It also lacks full reporter anonymity, a native mobile app, and automated routing to specific authorities, and the citizen-facing interface is mostly in English. For Pharmazine, the most pressing limitation is that the database layer is not yet hardened: the security audit found several sensitive tables that still lack row-level security, which has to be remediated before real patient records go anywhere near the system. On top of that, some peripheral modules are broader than they are deep, there is no automated test suite or CI pipeline, free-tier hosting causes cold starts, and offline support covers only the POS queue. I think naming these things precisely matters; an examiner or future maintainer is better served by knowing exactly what is unfinished than by a report that papers over the gaps.

Chapter 6

Internship Reflection and Professional Growth

This is the chapter where the internship and the two projects connect directly. It covers what I actually did at a2i-Muktopaath, what I came away having learned, and most importantly how each of those lessons became a real, specific decision in Chokh-e-Dekha or Pharmazine. For an internship-based report, this is not a sidebar from the engineering; it is the explanation for why the engineering ended up looking the way it does.

6.1 The Internship: a2i and Muktopaath

From 26 June to 25 September 2025 I worked with the Muktopaath team at the Aspire to Innovate (a2i) Programme, based at the ICT Tower in Agargaon, Dhaka. a2i is the Government of Bangladesh's flagship digital-transformation initiative under the ICT Division; it has digitised over two thousand government services and operates platforms that, collectively, have served citizens at a scale measured in hundreds of millions of transactions [1]. Muktopaath is its national e-learning platform, built on a microservices architecture with single sign-on, serving more than six lakh learners and governed against a formal Software Requirements Specification [2, 3]. Working inside an organisation operating at that scale, even as an intern, was the most formative professional experience I have had.

6.2 Company Profile: a2i and the Digital Bangladesh Ecosystem

Aspire to Innovate (a2i) was launched in 2007 with support from the UNDP and international partners and operates under the ICT Division and the Cabinet Division. It has digitised over two thousand government services, established thousands of Digital Centres that have collectively delivered hundreds of millions of services, and connected tens of thousands of government offices through the National Portal

Framework [1]. Beyond Muktopaath, a2i runs initiatives including the Digital Centres, the National Portal Framework, e-Nothi (digital file management), the National Helpline 333, ekPay, myGov, the Teachers Portal, NISE (skills-to-jobs matching), and the Innovation Lab. Being an intern there meant I was contributing to an ecosystem that reaches ordinary citizens across the country, not just a single product in isolation. That is what made me want to build for users who are not well served by existing tools.

6.2.1 How Muktopaath is Built

Muktopaath is a Learning Management System on a microservices architecture. Services for courses, assessments, webinars, training management, and reporting each run independently but tie together under a single sign-on. The platform is designed mobile-first, supports both Bangla and English, uses gamification, and provides customised dashboards for learners, partners, and administrators. The development process is structured and explicit: requirements go into a formal SRS document [3], every service has its context and flow documented, prototypes are validated before they become production code, and releases are versioned. Getting to observe that kind of discipline firsthand (requirements written before code, services designed to scale, quality assurance as a built-in part of the release cycle) gave me a concrete model to work from in my own projects.

6.2.2 Problems I Observed at Muktopaath

A good portion of the QA and support work involved diagnosing real, live problems, which trained me to recognise the kinds of failures that users actually care about: OTP and password-reset emails that did not arrive reliably; video players that forgot where you left off; certificate generation problems caused by name-spacing issues and the lack of a preview step; mixed Bangla-English search that gave poor results; and language switching that behaved inconsistently from page to page. None of these required clever algorithms to fix. They are the unglamorous, often tedious reliability and usability problems that determine whether users trust a platform or abandon it. That realisation, that trust is built or lost on the dull details, is the lesson I took most directly into both Chokh-e-Dekha and Pharmazine.

6.3 My Responsibilities and Contributions

Looking back, the range of my work (design, QA, user support, and content) turned out to be exactly the cross-section that teaches you how a platform actually holds together.

- **Certificate template design and automation.** I redesigned certificate templates and coordinated their integration with Muktopaath's certificate microservice, mapping database fields to template placeholders and handling

edge cases such as very long names. I proposed a certificate *preview* step before issuance to reduce repeated generation and support tickets, my first taste of fixing a problem by changing the workflow rather than the code.

- **Content and reading-material updates.** I reviewed course materials, replaced broken links, and standardised file naming, learning how small content inconsistencies multiply into support load at scale.
- **Bug detection and quality assurance.** I worked as a QA tester: choosing a feature, brainstorming normal and irregular paths, reproducing and documenting bugs with screenshots, reporting them through the internal ticketing system, and verifying fixes. I found and reported issues including video freezing on certain Android devices, unexpected session loss, mixed Bangla-English search failures, and certificate name-spacing errors.
- **User support and certificate issue handling.** Using internal admin tools I resolved real tickets, mostly certificate failures and access issues, which meant reading logs, retrying generation, and occasionally writing small SQL queries to verify data integrity. This is where I learned, viscerally, the value of clean data entry and clear error messages.
- **Video strategy and chatbot dataset.** I drafted a multi-part video storyboard to guide learners from registration to certification, and I collected, cleaned, and structured a chatbot FAQ dataset from WhatsApp and email support threads, categorised by topic and tagged by language and user type.
- **Social media and content management; national workshops.** I helped manage Muktopaath's presence across YouTube and LinkedIn and reported impersonating handles. I also assisted at a high-level content-review workshop at the Bangladesh Secretariat for an e-learning module on the Government Servants (Discipline & Appeal) Rules, and sat in on strategic meetings about national education platforms.

6.4 Lessons Learned and How They Shaped the Projects

None of the lessons from the internship came packaged as lessons; I picked them up while building, when a problem I remembered from Muktopaath showed up again in my own code. Five of them shaped the projects in ways I can point to directly.

Lesson 1: start with the user, not the technology. At Muktopaath, every feature decision started from what the user was struggling with, not from whatever tool was interesting that week. That is why Chokh-e-Dekha has a hybrid GPS-plus-map-pin location option rather than GPS alone, and why Pharmazine uses autocomplete text search rather than a barcode scanner that pharmacy staff with no formal training would find difficult. Both choices are technically less impressive. Both are better for the person actually using the system.

Lesson 2: testing is not optional. Working as a QA tester taught me that users do not remember features; they remember when things broke. My habit now is to build a feature and then deliberately try to make it fail. That habit is directly responsible for the lat/long bounds validation in Chokh-e-Dekha, the unique constraint on endorsements, and the concurrency testing that eventually surfaced the batch race condition in Pharmazine.

Lesson 3: design for the whole system, not the next feature. Muktopaath was architected to scale from the beginning: indexing and caching were treated as core design requirements, not things to add later when things got slow. I followed the same principle in both systems, with indexes and normalised schemas from day one, and in Pharmazine, business logic and access control kept firmly in the server tier rather than drifting to the client.

Lesson 4: a good document scales your knowledge. Putting together FAQ datasets and video scripts for Muktopaath made something obvious: one clear document can replace thousands of individual answers. Both systems ship with READMEs covering setup, architecture, schema, and common troubleshooting, partly for anyone else who might read the code, but mainly because the person most likely to need them is me, six months from now with no memory of why a particular decision was made.

Lesson 5: how you communicate matters as much as what you build. I had to explain technical ideas to BCS cadre officials and senior policymakers who had no software background, and occasionally something landed: once a senior official told me "that is a good idea, let us explore that." What those moments taught me was to explain the *why* before the *how*. This report is an application of that lesson: I explain why expiry-aware dispensing matters for a patient before I explain the row-locking implementation that makes it work correctly.

6.5 Technical Growth

The internship gave me skills; the projects turned those skills into judgement. Before a2i I could have said "I know how to build with a framework"; what I can say now is different: "I know when a database index is not optional, when caching is worth the complexity it adds, and why one shared serialisation base saves you from patching nineteen endpoints one at a time." Concurrency was something I understood in theory; it became real when a batch race condition only showed up under simultaneous sales in a deployed system. I also learned to diagnose a live system by probing it directly rather than staring at error messages and guessing. None of this is the kind of knowledge that sticks from a textbook; it stuck because the internship had already shown me what good practice looked like, so when the same problem appeared in my own code, I knew what to do with it.

6.6 Professional Development and Individual Contribution

Beyond code, the internship developed the professional habits this degree is meant to produce: working with designers, developers, and policy analysts; handling ambiguous, under-specified tasks by asking questions and iterating; and presenting technical constraints to non-technical stakeholders in terms they care about. It also gave me a clearer sense of why citizen-centric work matters, which is ultimately why both of my projects target underserved users rather than the easiest market.

In keeping with the individual nature of this submission, I confirm that both Chokhe-Dekha and Pharmazine were designed, implemented, deployed, tested, and documented by me alone. The internship work described above was carried out as a member of the a2i-Muktopaath team under the supervision of my workplace supervisor, and I have described only my own contributions to that team's work.

Chapter 7

Conclusion and Future Work

7.1 Summary

This report has documented an internship-based undergraduate project: a three-month internship at the a2i-Muktopaath platform, followed by the design, construction, and evaluation of two web-based software systems that apply what the internship taught me. **Chokh-e-Dekha** gives citizens a visual, location-based, trackable channel for reporting civic problems, gives administrators a command centre with SLA tracking, and lowers the barrier to the Right to Information Act through a guided letter wizard. **Pharmazine** gives small and medium pharmacies an affordable, web-based, pharmacy-aware management and point-of-sale system whose inventory-to-POS core, with batch- and expiry-aware dispensing, works end-to-end against a live database and is publicly deployed for demonstration. Throughout, the work was framed around a real industry practice gap rather than an academic research gap, and was driven by a single, evidence-based design methodology carried over from the internship.

7.2 Reflections

The most honest thing I can say is that this project showed me exactly how wide the gap is between a clean design and something that actually works. Chapter 4 has tidy diagrams and well-justified architectural choices. Chapter 5 has the geolocation that was consistently wrong until I gave users a manual pin as a fallback, the photo uploads that made 3G connections unusable until I added client-side compression, the queries that took seconds until I added the right indexes, the batch selection logic that needed several attempts and a row lock before it handled concurrent sales correctly, and the "CORS errors" that turned out to be the backend crashing before it could set any headers. You only discover that gap by actually building something, and you only know what direction to move in if you have already watched someone do it well. For me, that was a2i. Those three months were not just tasks to complete; they were where I learned to think like someone who actually ships software, and these two systems are the evidence that the thinking stuck.

7.3 Future Work

Pharmazine, in priority order: (1) database-tier hardening with row-level security and least-privilege access (this must happen before any real patient data is stored, and it is the single most critical outstanding item); (2) an automated test suite and CI/CD pipeline; (3) deepening the peripheral modules that are currently broad but shallow; (4) moving to always-on hosting with monitoring and logging in place; (5) extending offline support beyond the POS queue; and (6) running a pilot with a real partner pharmacy to collect genuine usage data and put concrete numbers behind the economic benefit.

Chokh-e-Dekha, in priority order: (1) run a formal pilot with a single ward office or city-corporation department, since even one engaged administrator would demonstrate the accountability mechanism; (2) build the whistleblower module with pseudonymous reporting and an encrypted evidence locker; (3) fully localise the citizen-facing UI into Bangla; (4) add Progressive Web App capabilities for an offline-tolerant mobile experience without an app store; and (5) publish a public open- data API so that researchers, NGOs, and journalists can use aggregate report data.

For both systems, the highest-value next step is the same: move from a demonstrable prototype to a real pilot with a real user, because the remaining challenges are now as much social and institutional as they are technical.

7.4 Concluding Remarks

Bangladesh does not lack demand for better civic and healthcare tooling, nor does it lack the infrastructure precedent; a2i has already proved that government-scale digital services are achievable. What is missing, in the two workflows this report addresses, is focused, affordable, locally adapted software. Chokh-e-Dekha and Pharmazine are my attempt to show that such software can be built with open-source tools, at negligible monetary cost, by a single graduating student, provided that student has had the chance to learn how real systems are made. That chance was my internship, and this report is both the product of it and my account of what it meant.

Bibliography

[1] M. M. Rahman, S. Ahmed, and K. Islam, "Digital Bangladesh: A study of the national ICT policy and its implementation," *Journal of Information Technology & Politics*, vol. 17, no. 3, pp. 272-288, 2020.

[2] Muktopaath, "Muktopaath: National E-Learning Platform of Bangladesh." [Online]. Available: https://www.muktopaath.gov.bd. (Accessed: 2026).

[3] Aspire to Innovate (a2i) Programme, "Software Requirements Specification for the AI-Driven Interactive E-Learning Platform (Muktopaath)," ICT Division, Government of Bangladesh, Tech. Rep., 2021.

[4] Aspire to Innovate (a2i) Programme, "333 Helpline: Official Statistics," 2024. [Online]. Available: https://a2i.gov.bd. (Accessed: 2026).

[5] Aspire to Innovate (a2i) Programme, "Impact Study on 333 National Helpline Service," ICT Division, Government of Bangladesh, Impact Study, 2024.

[6] The Daily Star, "DNCC flags 98 spots prone to waterlogging," *The Daily Star*, Dec. 2024. [Online]. Available: https://www.thedailystar.net.

[7] Business Post Bangladesh, "DNCC mobilises sanitation workers to combat waterlogging," *Business Post Bangladesh*, 2024.

[8] The Business Standard, "Dhaka North sets up control room to tackle waterlogging," *The Business Standard*, 2024.

[9] M. Rahman *et al.*, "The adoption of 333 National Helpline Services among the marginalized people in Bangladesh," 2024.

[10] World Bank, "Bangladesh governance and citizen engagement," World Bank Group, 2024. [Online]. Available: https://www.worldbank.org/en/country/bangladesh.

[11] World Health Organization, Regional Office for South-East Asia, "Bangladesh Medical Products Profile 2025," WHO, Medical Products Profile, 2025.

[12] S. M. Ahmed, M. A. Hossain, A. M. R. Chowdhury, A. U. Bhuiya, *et al.*, "Exploring the status of retail private drug shops in Bangladesh and action points for developing an accredited drug shop model: a facility based cross-sectional study," *Journal of Pharmaceutical Policy and Practice*, vol. 10, no. 21, pp. 1-13, 2017.

[13] M. A. Habib, M. M. Rahman, M. S. Islam, and F. Akter, "Present scenario and prospect of Model Pharmacy in Dhaka City," *Bangladesh Pharmaceutical Journal*, vol. 23, no. 2, pp. 112-120, 2020.

[14] M. Hasan and F. Akter, "Medicine wastage in small pharmacies: A study of Bangladesh," *Bangladesh Pharmaceutical Journal*, vol. 24, no. 2, pp. 112-120, 2021.

[15] United Nations Development Programme, "Civic tech initiatives," 2024. [Online]. Available: https://www.undp.org.

[16] Government of Bangladesh, "Right to Information Act, 2009," Act No. 20 of 2009, 2009.

[17] Laravel, "Laravel documentation." [Online]. Available: https://laravel.com/docs. (Accessed: 2026).

[18] S. Ramírez, "FastAPI documentation." [Online]. Available: https://fastapi.tiangolo.com. (Accessed: 2026).

[19] Meta Open Source, "React documentation." [Online]. Available: https://react.dev. (Accessed: 2026).

[20] The PostgreSQL Global Development Group, "PostgreSQL documentation." [Online]. Available: https://www.postgresql.org/docs/. (Accessed: 2026).

[21] Supabase, "Supabase documentation." [Online]. Available: https://supabase.com/docs. (Accessed: 2026).

[22] V. Agafonkin, "Leaflet: an open-source JavaScript library for interactive maps." [Online]. Available: https://leafletjs.com. (Accessed: 2026).

[23] OpenStreetMap contributors, "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org. (Accessed: 2026).

[24] OWASP Foundation, "OWASP Top 10," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/.

[25] World Wide Web Consortium (W3C), "Web Content Accessibility Guidelines (WCAG) 2.1," 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/.

[26] mySociety, "FixMyStreet." [Online]. Available: https://www.fixmystreet.com. (Accessed: 2026).

[27] CivicPlus, "SeeClickFix: citizen request and 311 management." [Online]. Available: https://seeclickfix.com. (Accessed: 2026).

[28] City of New York, "NYC311." [Online]. Available: https://portal.311.nyc.gov. (Accessed: 2026).

[29] PharmaSoft BD, "PharmaSoft Bangladesh: pharmacy management solution," 2022. [Online]. Available: https://www.pharmasoft.com.bd.

[30] Intuit QuickBooks, "QuickBooks for pharmacies," 2023. [Online]. Available: https://quickbooks.intuit.com.

[31] MedPlus, "MedPlus pharmacy management," 2023. [Online]. Available: https://www.medplusmart.com.

[32] RxNT, "RxNT pharmacy management software," 2023. [Online]. Available: https://www.rxnt.com.

[33] S. Ahmed, R. Khan, and M. Ali, "Impact of digitalization on pharmacy efficiency in developing countries," *Journal of Healthcare Management*, vol. 18, no. 3, pp. 234-248, 2020.

[34] N. Islam and Å. Grönlund, "E-learning adoption in developing countries: A systematic review," in *Proc. Int. Conf. e-Learning*, 2016, pp. 156-165.

Appendix A: Course Outcome (CO) Mapping

The table below maps each course outcome to the section(s) of this report where it is addressed, following the section numbering expected by the marking rubric.

CO	**Outcome**	**Where addressed**
CO1	Problem formulation	Ch. 1 (1.1 to 1.6)
CO2	Requirements and specifications	Ch. 3, §3.1
CO3	Societal impact	Ch. 3, §3.2
CO4	Environmental impact	Ch. 3, §3.3
CO5	Design (multiple solutions)	Ch. 4, §4.1 to 4.2
CO6	Analysis of design solutions	Ch. 5, §5.1, 5.2, 5.4, 5.6
CO7	Final design and development	Ch. 5, §5.2
CO8	Tool usage	Ch. 5, §5.5; defense demo
CO9	Research and literature survey	Ch. 2
CO10	Resource and risk management	Ch. 3, §3.6 to 3.7
CO11	Economic analysis	Ch. 3, §3.8
CO12	Ethics and academic responsibility	Ch. 3, §3.4; Ethics Statement
CO13	Functioning as an individual	Ch. 6, §6.5 (individual contribution)
CO14	Communication	Whole report; defense presentation; IEEE journal version

Appendix B: Internship Certificate

The internship at the Aspire to Innovate (a2i) Programme was completed under the Information and Communication Technology Division, Government of Bangladesh. The text of the issued internship certificate is reproduced below; a scanned copy can be inserted as an image in place of the placeholder.

05 October 2025

TO WHOM IT MAY CONCERN

We hereby certify that **Mohammad Azmain Hossain Alfi** has worked with the Aspire to Innovate (a2i) Programme under the Information and Communication Technology Division since **2025**. His tenure with a2i includes:

- Serving as an **Intern** from **26th June 2025 to 25th September 2025**.

During his tenure with a2i, **Mohammad Azmain Hossain Alfi** consistently demonstrated diligence, integrity, and a strong work ethic. His contributions to the team have been highly valued, particularly in prioritizing and coordinating efforts for the **Future of Education** Team of a2i.

He has completed many tasks beyond his responsibilities, proving himself to be a result-oriented, hardworking, and motivated individual with excellent communication skills. His dedication and sincerity in performing his tasks have been commendable.

We extend our best wishes to him as he continues his career journey.

Best wishes

5·10·25

Mohammad Arifur Rahman
Head of HR
Aspire to Innovate (a2i) Programme
Information & Communication Technology Division
E-14/x, ICT Tower, Agargaon, Dhaka-1207
E-mail: arifur.rahman@a2i.gov.bd

Aspire to Innovate-a2i

Cabinet Division

ICT DIVISION

INNOVATE FOR ALL

Figure 7.1: a2i internship certificate (scanned copy).