Semester Long Internship Spring 2026

On

Development of eSim IC Management Portal

Submitted by

Devyanshu Negi
VIT Bhopal University

Under the guidance of

Prof. Prabhu Ramachandran
Principal Investigator
Department of Aerospace Engineering
Indian Institute of Technology Bombay

June 5, 2026

Acknowledgment

I express my sincere gratitude to Prof. Prabhu Ramachandran for providing me with the opportunity to be part of the FOSSEE internship program and for his continued efforts in promoting open-source engineering tool development. His leadership and vision have been instrumental in fostering meaningful student participation in the open-source ecosystem.

I also acknowledge Prof. Kannan M. Moudgalya for his foundational role in establishing and strengthening the FOSSEE initiative. His contributions to open-source education and the development of the FOSSEE fellowship framework have been pivotal in creating the academic and organisational platform through which this internship was undertaken.

My sincere appreciation extends to my mentor, Sumanto Kar, for his continual support, technical guidance, and encouragement throughout the duration of this project. His insights and feedback played a key role in refining ideas, overcoming challenges, and ensuring timely completion of the tasks assigned to me.

I would also like to thank my internal mentor, Mr. Varad Patil, for his valuable guidance, coordination, and technical inputs during the internship. His mentorship contributed significantly to the clarity, progress, and successful execution of the work.

This internship has been an enriching learning experience, allowing me to work closely with modern full-stack web development technologies, design and deploy the eSim IC Management Portal, and gain exposure to production-grade application architecture, database design, and authentication workflows. The knowledge acquired during this period will undoubtedly support my future academic and professional pursuits.

I would also like to thank the entire FOSSEE team for their coordination, assistance, and timely interactions at various stages of this work. Their collective efforts ensured smooth workflow, resource accessibility, and effective project execution.

Contents

Chapter 1

Introduction

1.1 Problem Statement

The FOSSEE eSim internship programme onboards multiple batches of interns each semester, with each intern assigned the task of building and verifying simulation files for specific Integrated Circuits. As the programme scaled, administering this process manually became increasingly untenable. Administrators had no centralised mechanism to assign IC tasks, monitor individual intern progress, track submission statuses, or conduct structured reviews. Interns, in turn, lacked a standardised interface to discover available ICs, claim work, or receive actionable feedback on their submissions.

The core problem, therefore, was the absence of a dedicated system capable of managing the full lifecycle of admin–intern coordination — from task assignment and progress tracking through submission review and conflict resolution — across multiple concurrent intern batches. This project was undertaken to address that gap by designing and delivering a purpose-built IC Management Portal for the eSim programme.

1.2 Background of the FOSSEE–eSim Programme

The Free/Libre and Open Source Software for Education (FOSSEE) project is based at the Indian Institute of Technology Bombay (IITB) and operates under the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education, Government of India. The mission of FOSSEE is to promote the adoption of free and open-source software across engineering, scientific, and research domains in India, with the broader goal of reducing dependence on proprietary platforms and building sustainable open-source alternatives in academia.

Among FOSSEE's major outcomes is eSim, an Electronic Design Automation (EDA) platform designed to support circuit design, simulation, and analysis. eSim provides an integrated environment that combines schematic capture with NGSPICE-based simulation and waveform analysis. The platform hosts a large and growing catalog of Integrated Circuit (IC) models, to which each batch of interns contributes by creating simulation files, verifying component behavior, and documenting the ICs.

As the internship program scales and multiple batches of students participate simultaneously, the operational complexity of managing IC assignments, tracking submission progress, conducting reviews, and resolving naming conflicts increases substantially. This need for structured tooling forms the foundation of the present project.

1.3 Rationale for a Dedicated IC Management System

Prior to this project, the coordination of IC simulation tasks across intern batches was managed manually, without a centralized system for task assignment, progress tracking, or review workflows. This created several operational challenges: there was no standardized way for interns to discover available ICs, no mechanism for admins to monitor submission status at scale, and no automated resolution pathway for duplicate IC naming conflicts that arose when interns independently proposed new entries.

The absence of structured tooling increased administrative overhead, reduced visibility into batch-level progress, and made it difficult to maintain data consistency across large datasets. As eSim's IC catalog has grown to over 765 components, the need for a purpose-built portal became evident — one that would streamline both the intern-facing workflow and the admin-facing management capabilities within a single, authenticated, production-grade application.

1.4 Aim and Scope of the Project

The primary aim of this project was to design, develop, and deploy the eSim IC Management Portal — a full-stack web application enabling the eSim team to manage its IC simulation task pipeline efficiently. The portal serves two distinct categories of users: Admins, who manage batches, review submissions, resolve requests, and maintain the IC catalog; and Interns, who browse available ICs, claim simulation tasks, submit work for review, and request new IC additions.

The scope of the project encompasses the full software development lifecycle — from initial database schema design and UI scaffolding through authentication integration and production deployment. Key functional areas include role-based access control, server-side paginated data management, an approval and rejection review workflow with mandatory feedback, conflict resolution for IC naming duplicates, and secure Google OAuth authentication with an email whitelist model.

A key design principle throughout the project was to keep the application lean and maintainable by leveraging native Next.js 15 primitives — React Server Components, Server Actions, and URL-driven state — rather than introducing heavyweight client-side state management libraries.

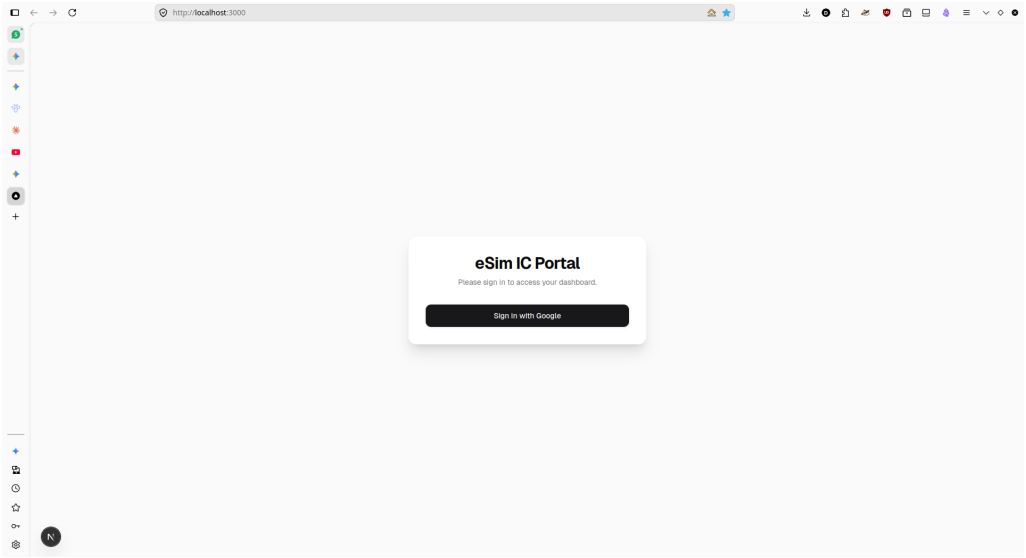

Figure 1.1: eSim IC Management Portal — Application Entry Point

Chapter 2

System Overview

2.1 Project Summary

The eSim IC Management Portal is a full-stack web application built using the Next.js 15 App Router, TypeScript, Prisma ORM, and Neon serverless PostgreSQL. The application is deployed to Vercel and is authenticated using Auth.js v5 with Google OAuth. It operates across two role-based portals — Admin and Intern — each with a distinct set of views and capabilities tailored to the respective user's operational needs.

- **Project Name:** eSim IC Management Portal
- **Programme:** FOSSEE Semester-Long Internship, Spring 2026
- **Hosting Institute:** Indian Institute of Technology Bombay (IIT Bombay)
- **Intern:** Devyanshu Negi, VIT Bhopal University
- **Mentor:** Sumanto Kar
- **Duration:** 19 February 2026 – May 2026 (approximately 3 months)
- **Mode:** Fully Remote
- **Deployment:** Vercel (Production)

2.2 Technology Stack

The technology choices for this project were made to align with modern full-stack web development practices, prioritise server-side rendering performance, and minimise client-side bundle size. Table 2.1 summarises the complete technology stack employed.

Category	Technology	Purpose / Notes
Framework	Next.js 15.5 (App Router)	Full-stack React with RSC, Server Actions, and file-based routing
Language	TypeScript	Strict end-to-end type safety across client and server code
ORM / Database	Prisma + Neon (Postgres)	Type-safe database access over serverless PostgreSQL
Styling	Tailwind CSS v4	Utility-first CSS with zero runtime overhead
UI Components	shadcn/ui	Accessible headless component primitives built on Radix UI
Data Tables	TanStack Table v8	Headless table engine for server-side pagination and sorting
Authentication	Auth.js v5 (NextAuth)	Google OAuth provider with email whitelist and role mapping
Deployment	Vercel	Edge-optimized deployment with preview environments
State Management	URL Query Parameters	Native URL-driven state — no Redux/Zustand required

Table 2.1: Technology Stack of the eSim IC Management Portal

2.3 Development Phases

The project was structured into nine major development phases (Phases 0 through 8), each with clearly defined objectives and deliverables. This phased approach ensured steady, reviewable progress and allowed individual components to be built and validated incrementally before integration.

Phase	Title	Key Deliverables
0	Project Scaffolding	Next.js 15.5 app setup, Tailwind CSS v4, shadcn/ui registry configured.
1	Database Schema & Seed	Prisma schema (IC, User, ICTask, Enrollment, Batch) designed; 765 real ICs seeded into Neon Postgres.
2	Role-Based Layouts	Mock authentication via DevUserSwitcher; server-side cookie parsing for ADMIN/INTERN role enforcement.
3a	Intern Dashboard	Task cards, status badges, and updateTaskStatus Server Action.
3b	Intern IC Browse	TanStack Table with server-side pagination and atomic claimTaskAction transactions.
3c	Request New IC	Modal form with Server Action to prevent duplicate IC requests at the database level.
4a	Admin Dashboard	Statistics cards powered by 9 concurrent Promise.all queries; awaiting-review tables.
4b	Admin Intern Table	Server-side searchable paginated table for intern management.
4c	Admin Intern Detail	Full per-intern task history drill-down view.
4d	Admin IC Catalog	Complete paginated global IC dictionary with search.
4e	Admin IC Detail	Server Actions for inline IC metadata and alias management.
5a	Admin Review Queue	Approve/Reject workflow with mandatory reviewer feedback enforced server-side.
5b	Admin Request Queue	APPROVE_AS_NEW and MERGED resolution logic with naming conflict UI.
6	Pre-Alpha & Deployment	Vercel deployment, mock auth flags, skeleton loaders, task limit refinements.
7	NextAuth & Batch UI	Auth.js v5 Google OAuth with email whitelist; Admin Batch Management with bulk enrollments.
8	Documentation	Architecture summaries, tech stack documentation, and SUMMARY.md generated.

Table 2.2: Development Phase Summary

Chapter 3

Database Design and Schema

3.1 Schema Overview

The data layer of the portal was designed using Prisma ORM connected to a Neon serverless PostgreSQL database. Prisma was selected for its strong TypeScript integration, automatic migration support, and ability to generate a fully typed client from a single schema file — eliminating an entire class of runtime type errors at the database boundary.

The schema was designed around the core domain entities of the eSim IC management workflow. The primary entities are: IC (the catalog of integrated circuit entries), User (both admins and interns), ICTask (the assignment of a specific IC to a specific intern), Batch (a named intern cohort), and Enrollment (the mapping between a User and a Batch). Additional status enumerations capture the lifecycle states of tasks and requests.

3.2 Entity Descriptions

- **IC:** Stores all integrated circuit entries in the catalog, including name, description, category, availability status, and a list of aliases for naming conflict resolution.
- **User:** Represents all system users with a role field (ADMIN or INTERN), email, name, and OAuth-linked identity.
- **ICTask:** Links a User to an IC with a status field (UNCLAIMED, IN_PROGRESS, UNDER_REVIEW, APPROVED, REJECTED) and reviewer feedback text.
- **ICRequest:** Tracks intern requests for new IC additions, with status (PENDING, APPROVED_AS_NEW, MERGED, REJECTED) and admin resolution metadata.
- **Batch:** A named group used to manage intern cohorts and bulk enrollments.
- **Enrollment:** A join table linking users to batches, enabling batch-level access control.

3.3 Database Seeding

A comprehensive seed script was authored to populate the database with all 765 ICs from the existing eSim dataset prior to launch. The seed script parsed the source data, validated entries for completeness, and inserted records in a single transaction to ensure consistency. This allowed the portal to launch with production-level catalog data from day one, enabling realistic testing of all pagination, search, and filtering features.

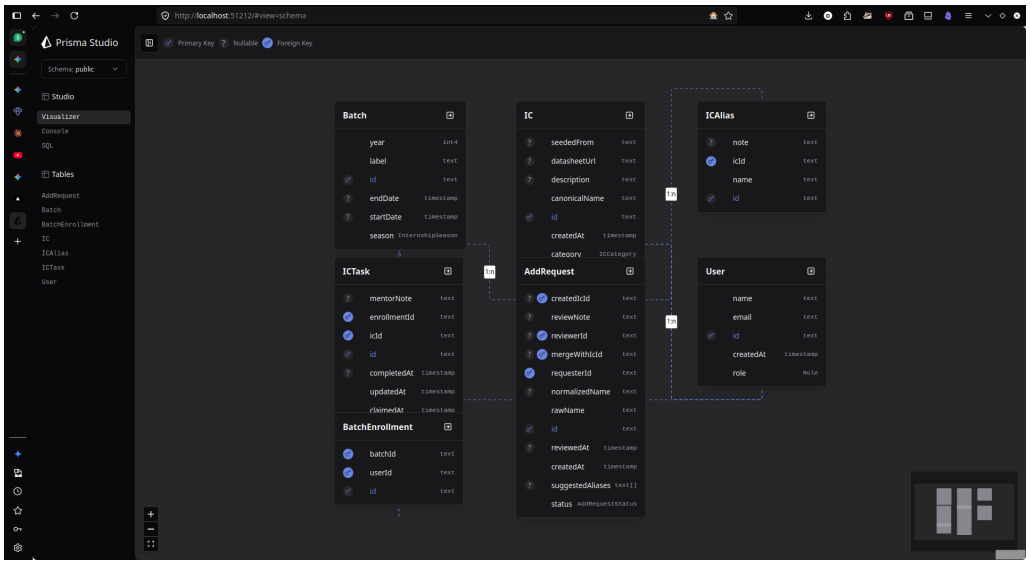

Figure 3.1: Entity Relationship Diagram of the Portal Database Schema

Chapter 4

Intern-Facing Features

4.1 Task Dashboard

The intern dashboard is the primary workspace for each intern, providing a real-time view of all their claimed IC simulation tasks. Tasks are displayed as status-grouped cards with visual status badges corresponding to the current lifecycle state of each assignment. Interns can update task statuses (e.g., move from IN_PROGRESS to UNDER_REVIEW) using Server Actions that validate the transition and persist the change atomically.

The dashboard also surfaces reviewer feedback for rejected tasks prominently, ensuring interns can address the stated issues before resubmitting. An active task counter enforces the per-intern task limit, preventing the claiming of additional ICs once the limit is reached, with tasks that are UNDER_REVIEW excluded from the active count.

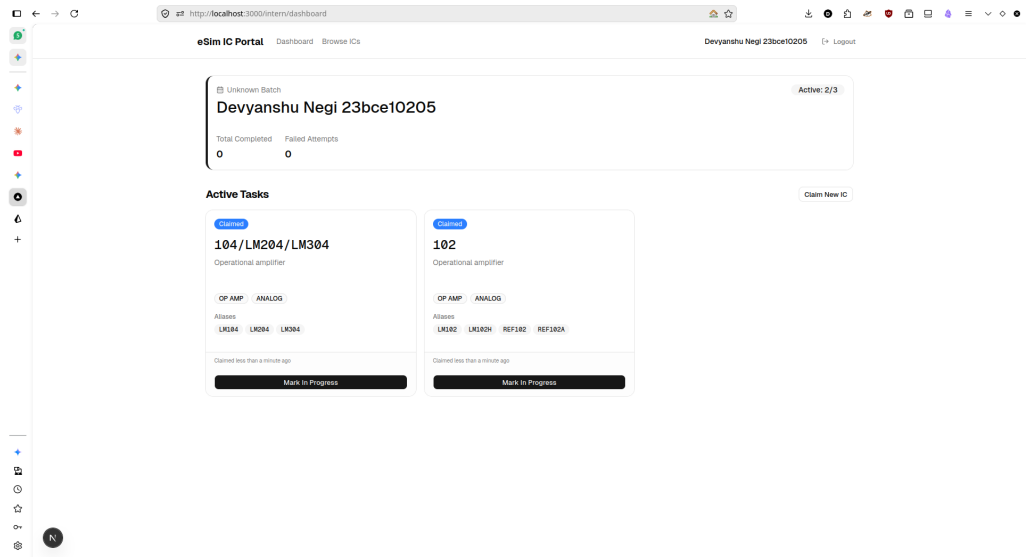

Figure 4.1: Intern Task Dashboard — Status-Grouped IC Task Cards

4.2 IC Browse Catalog

The intern browse view provides a searchable and paginated listing of all ICs in the eSim catalog. It is implemented using TanStack Table v8 in headless mode, with all data operations (pagination, sorting, and search filtering) executed server-side. State is managed through URL query parameters rather than client-side hooks, ensuring that the current view is always shareable and deep-linkable.

Interns can search ICs by both name and description, and filter by availability status. Claiming a task is performed through an atomic Server Action (`claimTaskAction`) that checks availability, enforces the task limit, creates the ICTask record, and updates IC availability within a single Prisma transaction — preventing race conditions under concurrent user access.

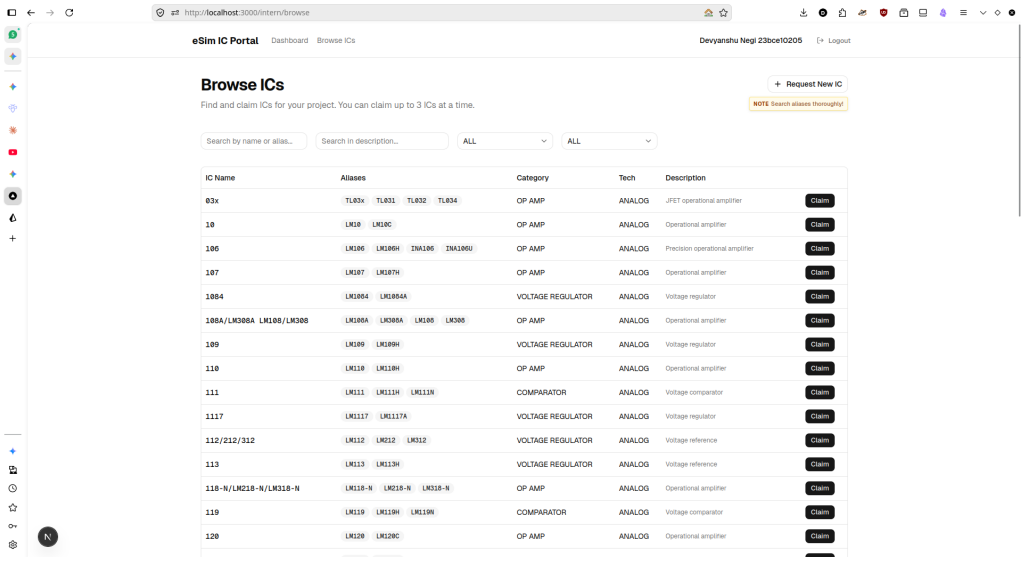

Figure 4.2: Intern IC Browse — Server-Side Paginated and Searchable Table

4.3 Request New IC

When an intern identifies an IC that is not yet present in the catalog, they can submit a new IC request through a modal form. The Server Action behind this form performs a duplicate check against the existing IC catalog before inserting the request record, ensuring that obvious naming duplicates are rejected at submission time. Approved requests are either entered as new ICs (APPROVE_AS_NEW) or merged with an existing entry (MERGED), as determined during admin review.

Chapter 5

Admin-Facing Features

5.1 Admin Dashboard

The admin dashboard provides a centralized management overview of the entire portal. A statistics bar at the top of the page displays key metrics — total ICs in the catalog, pending review count, active intern count, and overall task completion rates — all computed by nine concurrent Prisma queries executed with `Promise.all`, completing in a single server roundtrip.

Below the statistics, the dashboard surfaces the most actionable items for the admin: tasks currently awaiting review and pending IC requests. These tables allow admins to navigate directly to the relevant review interfaces without having to search through full listings.

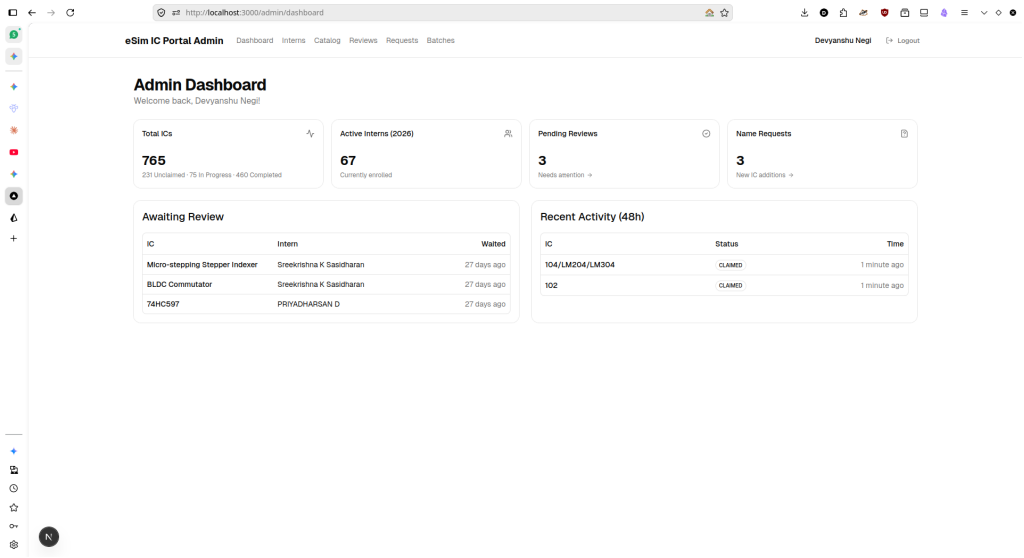

Figure 5.1: Admin Dashboard — Statistics Overview and Pending Action Tables

5.2 Intern Management

The admin intern management interface presents a server-side searchable and paginated table of all registered interns. Admins can search by name or email, and

navigate to a per-intern detail page that shows the complete task history for that intern — including all IC assignments, their current statuses, and any reviewer feedback that has been provided.

5.3 IC Catalog Management

The admin IC catalog view provides a complete paginated listing of all ICs in the system, with search and filtering capabilities. Each IC entry links to a detail page that allows admins to perform inline edits on IC metadata — including name, description, and the list of aliases — through Server Actions backed by Prisma transactions.

Alias management is implemented using a shadcn/ui Command palette component embedded inside a Popover, providing a fast search-and-select interface for associating variant names with canonical IC entries. This capability is essential for the naming conflict resolution workflow.

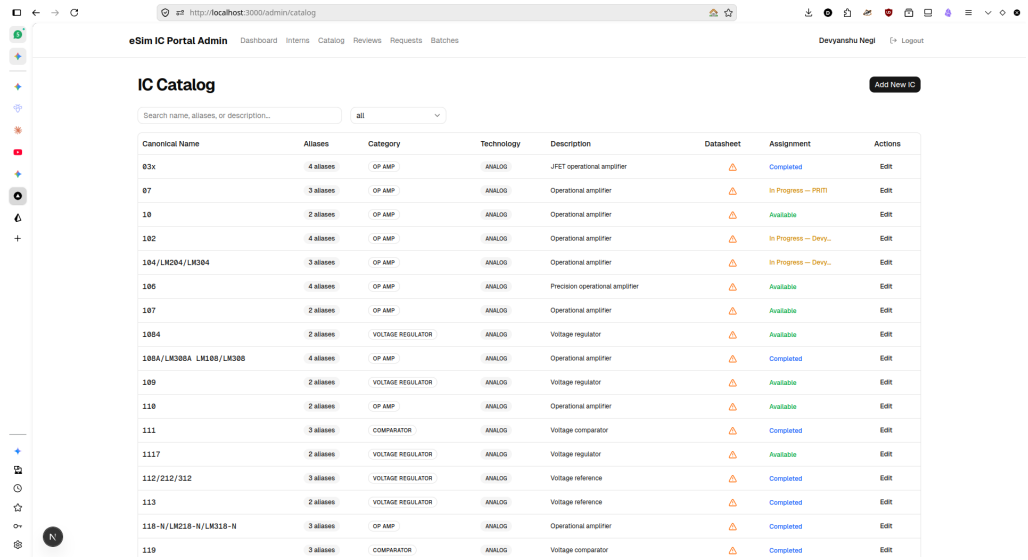

Figure 5.2: Admin IC Detail Page — Inline Metadata and Alias Editing

5.4 Review Queue

The admin review queue (Phase 5a) provides a dedicated interface for processing intern task submissions. For each submission, the admin can view the task details, review the submitted work, and either approve or reject the submission. Rejection requires the admin to provide mandatory written feedback, which is validated server-side before the status transition is committed. This feedback is subsequently surfaced to the intern on their dashboard to guide resubmission.

5.5 IC Request Resolution

The IC request resolution interface (Phase 5b) allows admins to process new IC requests submitted by interns. For each pending request, the admin selects one of

two resolution paths: APPROVE_AS_NEW inserts the proposed IC as a new catalog entry, while MERGED links the request to an existing IC that is deemed a sufficient match. The naming conflict resolution UI uses a shadcn/ui Command palette inside a Popover to provide fast search-as-you-type selection of existing catalog entries.

All resolution actions are executed as atomic Prisma transactions, ensuring that the IC catalog, request record, and related task statuses are updated consistently in a single operation.

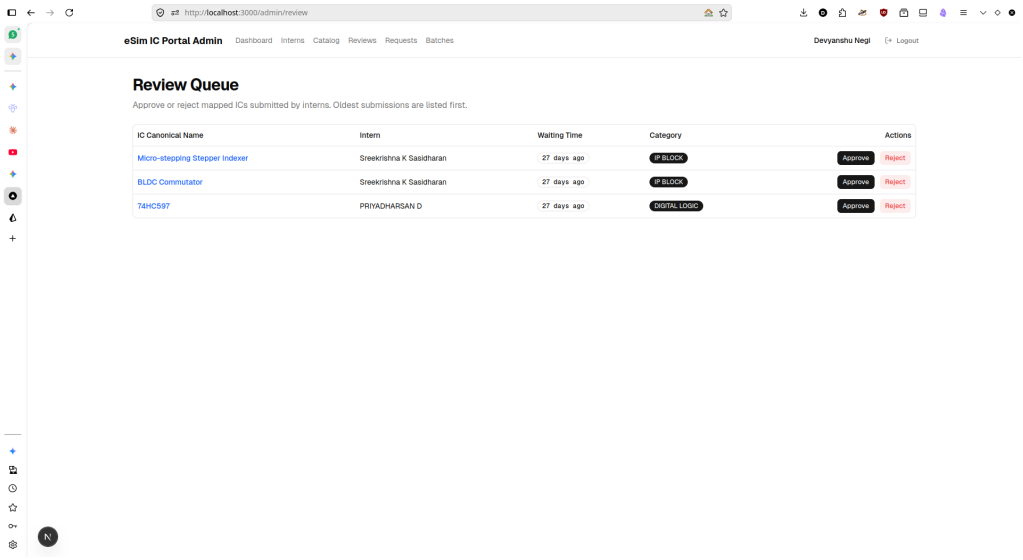

Figure 5.3: Admin Review Queue — Approve/Reject Workflow with Mandatory Feedback

5.6 Batch Management

The batch management interface (Phase 7) allows admins to create named intern batches and perform bulk enrollment operations. Admins can add multiple interns to a batch simultaneously via a multi-select UI, simplifying onboarding for new cohorts. Batch membership determines which interns have access to the portal and informs filtering and reporting at the admin level.

Chapter 6

Architecture and Design Decisions

6.1 Data Fetching and State Management

All data fetching in the portal is performed exclusively at the React Server Component (RSC) level. When users interact with pagination controls, search fields, or sorting toggles, the state change is reflected as a URL query parameter update. The server reads these parameters on each request and returns a fresh, fully rendered page. This approach eliminates the need for client-side state management libraries entirely, resulting in lean browser bundles and inherently shareable, bookmarkable URLs.

The primary tradeoff of this approach is a slight latency on each state change compared to optimistic client-side updates, since each change triggers a full server round-trip. This is most noticeable on slower connections when querying large datasets; however, skeleton loading states were added to all major data views to maintain a smooth perceived performance under these conditions.

6.2 Form Mutation and Atomic Validation

All form mutations — task status updates, IC claims, approvals, rejections, and request resolutions — are implemented as Next.js Server Actions declared with the `use server` directive. Each action performs its database writes within a Prisma transaction (`prisma.$transaction()`) to guarantee atomicity, ensuring that no partial state is ever committed in the event of a mid-operation failure. The Next.js `revalidatePath` function is invoked after each successful mutation to invalidate the relevant cached page segments and ensure the UI reflects the latest state immediately.

The principal architectural tradeoff of this approach is tight coupling between the business logic validation layer and the Next.js server boundary. Should the eSim team wish to decouple the portal's backend into a standalone REST API (e.g., a Go or Python microservice), the validation logic embedded in Server Action files would need to be re-implemented as independent API endpoint guards.

6.3 Authentication and Access Control

Authentication was originally implemented via a mocked DevUserSwitcher component for rapid UI development without requiring an active OAuth provider. In Phase 7, this was replaced with Auth.js v5 (NextAuth) integrating Google OAuth as the identity provider. Access control is enforced by matching the authenticated user's email address against a whitelist in the database, where each whitelisted entry is associated with either an ADMIN or INTERN role. Unauthenticated users are redirected to the sign-in page by Next.js middleware.

The development mock switcher was preserved behind a `NEXT_PUBLIC_ENABLE_MOCK_AUTH` environment flag to allow isolated UI testing in Vercel preview deployments without requiring live Google credentials. The tradeoff of real OAuth integration is that local development now requires an active internet connection to Google's authentication servers, and strict environment variable configuration is mandatory for all deployment environments.

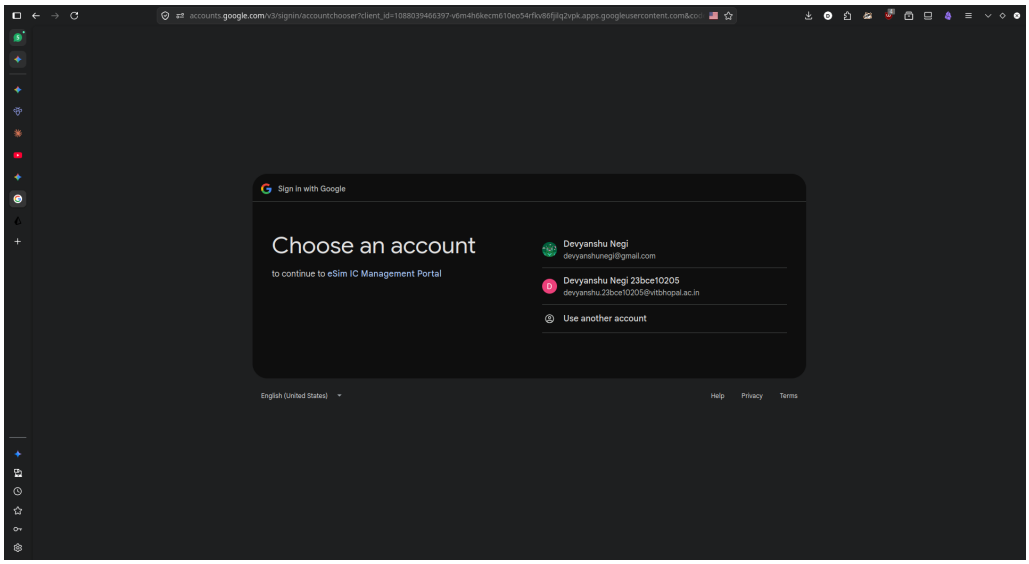

Figure 6.1: Production Authentication — Google OAuth Sign-In Interface

6.4 Code Structure and Library Selection

UI components were built using shadcn/ui primitives copied directly into the project's component directory, rather than imported as a black-box dependency. This approach gives full ownership over component markup while leveraging the accessibility foundations of Radix UI. Debouncing of search input fields was implemented using native React hooks (`useEffect` and `setTimeout`) to avoid introducing an additional utility library for a single use case.

Strict TypeScript mode, combined with Prisma-generated ENUM types and React 19's generic attribute system, produced type conflicts at several integration points. These were resolved with targeted inline type assertions, with all changes validated by running `npx tsc --noEmit` to ensure no build-time type errors remained. Purposely avoiding libraries such as Zod for runtime validation kept the

dependency footprint minimal without sacrificing type safety at build time.

Chapter 7

Key Technical Challenges and Resolutions

7.1 Server Action Nested Relational Revalidation

Server Actions that mutated nested relational data — such as approving a task linked through enrollment, user, and IC relations — initially failed to refresh the correct UI paths after mutation. The issue was traced to incorrect `revalidatePath` invocations that targeted parent routes rather than the specific route segments displaying the affected data. This was resolved by explicitly mapping each mutation's affected relations to their corresponding route segments and calling `revalidatePath` with precision after each commit.

7.2 Naming Conflict UI — Popover and Command Palette

The shadcn/ui Dialog component's trigger wrapper conflicted with the Command palette component used for the IC naming conflict search interface. The Dialog's direct children wrapper was intercepting focus and event propagation in a way that prevented the Command's internal search state from updating correctly. The fix involved decoupling the Command component from the Dialog trigger entirely by encapsulating it inside a standalone Popover layout, which resolved the interaction without altering either component's internal behaviour.

7.3 Vercel Build Failures — Prisma Client Generation

Initial Vercel deployments failed at runtime with the error "PrismaClient is unable to run in this environment," because Vercel's serverless build environment does not automatically execute `prisma generate`. This was resolved by adding a `postinstall` script to the project's `package.json` that invokes `prisma generate` as part of the

npm install lifecycle, ensuring the Prisma client is always regenerated during both local setup and Vercel build processes.

7.4 Production Mock Authentication

Enabling the DevUserSwitcher in Vercel preview environments for team testing required careful conditional logic to ensure it would never appear in the production deployment. This was implemented by reading the `NEXT_PUBLIC_ENABLE_MOCK_AUTH` environment variable inside a NextRequest middleware check, rendering the switcher only when the flag is explicitly set and the deployment context is not the production domain.

7.5 TypeScript Strict Mode and Prisma ENUM Bindings

Next.js 15, React 19, Prisma, and shadcn/ui collectively produced TypeScript type conflicts at several integration points — particularly where Prisma-generated ENUM types were used as values for HTML select attributes or passed into generic component props from headless UI libraries. Each conflict was analysed individually to confirm the runtime behaviour was correct, and resolved with a targeted inline type assertion. The full build was continuously verified with `npx tsc --noEmit` to prevent regressions.

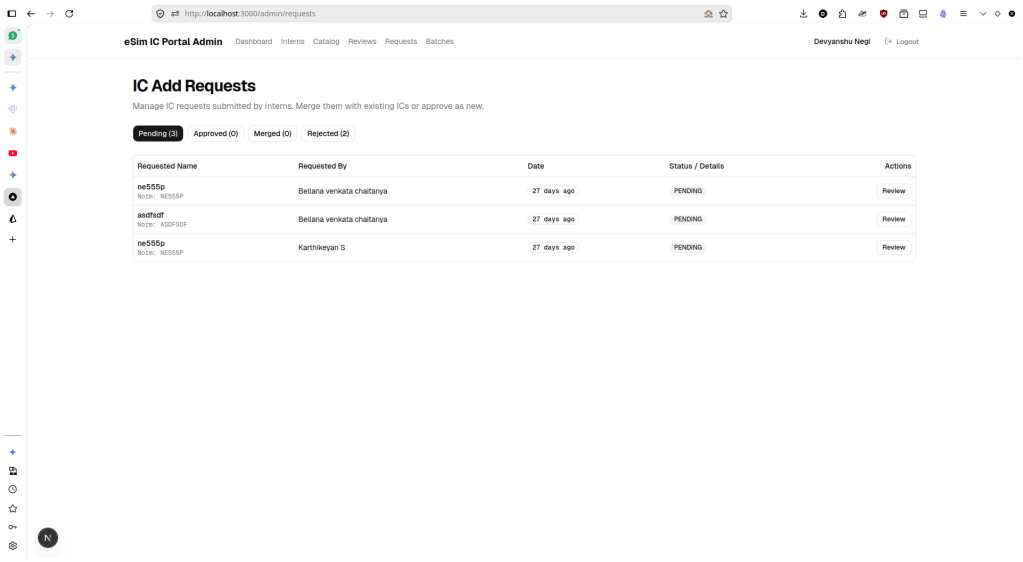

Figure 7.1: Representative Technical Resolution — Before and After Comparison

Chapter 8

Challenges Faced and Key Learnings

8.1 Challenges Encountered

8.1.1 Next.js 15 App Router Learning Curve

The Next.js 15 App Router introduces a strict mental model for the boundary between Server Components, Client Components, and Server Actions. Early in the project, data fetching was occasionally attempted within Client Components, or revalidation was called from an incorrect context, requiring non-trivial refactoring of page structures. Developing fluency in this model took several iterations but ultimately led to a cleaner, more performant architecture.

8.1.2 Atomic State Across Complex Relations

Ensuring complete atomicity for multi-step database mutations — such as claiming a task, decrementing IC availability, and logging an enrollment event simultaneously — required careful use of Prisma transactions with explicit rollback handling. Designing these transactions to remain safe under concurrent access by multiple interns was an iterative process that required thinking carefully about optimistic locking and failure modes.

8.1.3 Auth.js v5 Integration

Auth.js v5 is a significant rewrite from its predecessor with substantially restructured APIs. Documentation for edge cases such as session callback signatures, database adapter integration, and middleware configuration within the App Router was incomplete at the time of integration. Resolving these required reading the library source code and engaging with community issue threads, which was time-consuming but ultimately deepened the understanding of the authentication layer.

8.1.4 TypeScript in a Rapidly Evolving Stack

Working with Next.js 15, React 19, and Prisma simultaneously — all at versions that were relatively new at the time — meant that some TypeScript generic constraints were not yet fully stabilised. Resolving type conflicts without introducing runtime errors required careful analysis of each library's type definitions, which was a non-trivial exercise in TypeScript proficiency.

8.2 Key Learnings

- Acquired deep practical experience with the Next.js 15 App Router paradigm — React Server Components, data streaming, Server Actions, and fine-grained cache invalidation with `revalidatePath`.
- Developed confidence in designing production-grade database schemas with Prisma for multi-role, relational applications, including the use of transactions, ENUMs, and relational seeding.
- Learned the value of URL-driven state management for server-rendered data-heavy admin interfaces — a pattern that provides shareability, deep-linkability, and SSR compatibility without client-side overhead.
- Gained hands-on experience deploying a full-stack application to Vercel with serverless constraints, including resolving build pipeline issues and managing environment variables across preview and production environments.
- Appreciated the importance of iterative, phase-based project management — structuring the work into well-scoped phases maintained momentum, enabled regular review checkpoints, and allowed problems to be isolated early.
- Understood the practical realities of integrating real OAuth (Google) into a production application, including the environment dependency implications for local development and preview deployments.

Chapter 9

Results and Deliverables

9.1 Delivered Artefacts

The following artefacts were produced and handed over at the conclusion of the internship:

- A fully functional, production-deployed web application accessible at the Vercel Deployment URL.
- Role-based portals for Admin and Intern users, authenticated via Google OAuth with an email-whitelist access control model.
- A complete IC management system covering all 765 catalog entries, with task assignment, review workflows, naming conflict resolution, and batch management.
- A production Vercel deployment with properly configured environment variables, Prisma client generation, and preview environment support.
- Architecture documentation in SUMMARY.md, describing system design, component responsibilities, data flows, and deployment configuration.
- All source code committed to the project repository with a structured commit history across all phases.

9.2 Application Screenshots

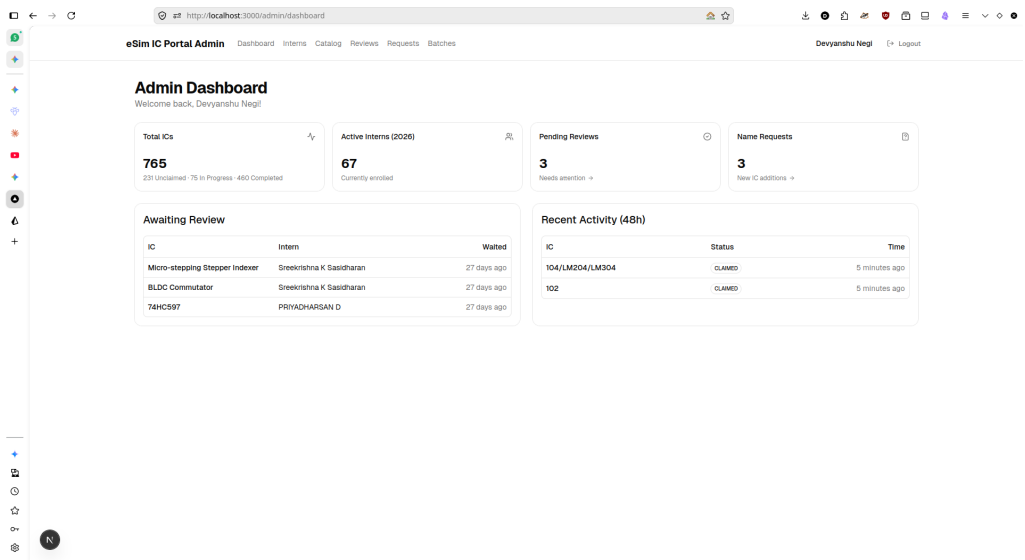

Figure 9.1: Final Application — Admin Dashboard

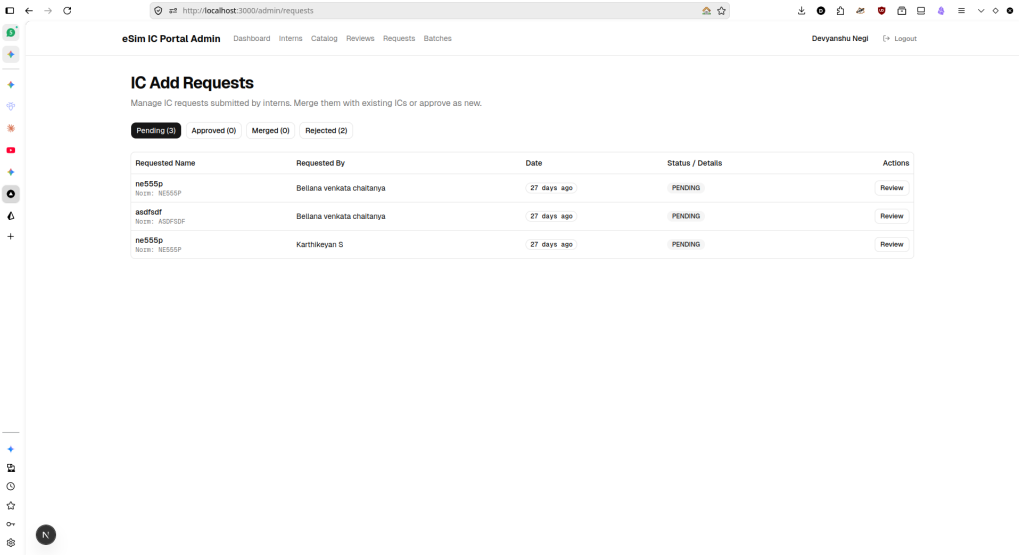

Figure 9.2: Final Application — Intern Task Dashboard

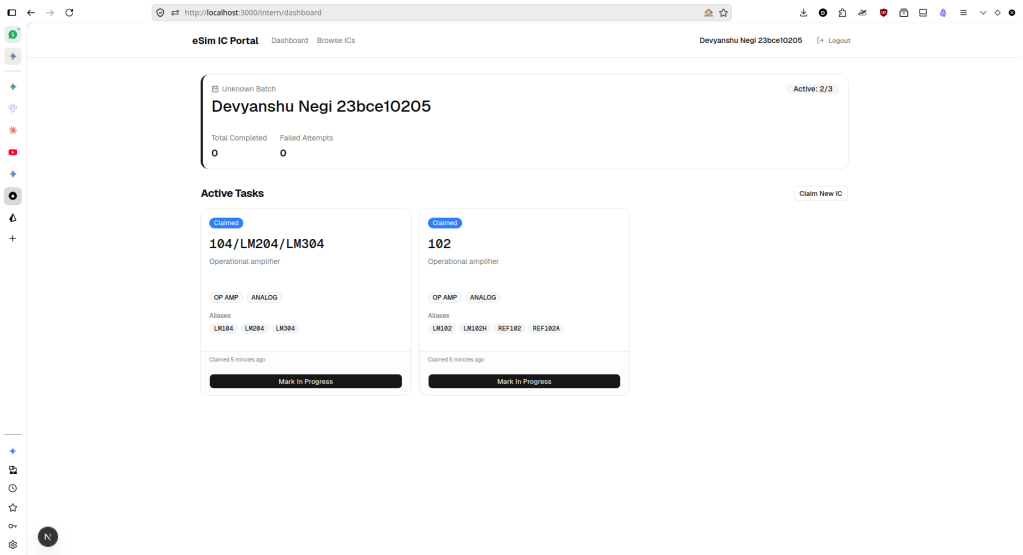

Figure 9.3: Final Application — Admin Review Queue

Chapter 10

Conclusion

10.1 Summary of Contributions

This project presents a production-ready IC management portal developed from the ground up during a three-month semester-long internship at FOSSEE, IIT Bombay. The system addresses a genuine operational need within the eSim team — the structured, scalable management of IC simulation tasks across multiple intern batches — and delivers a fully authenticated, role-segregated web application to meet that need.

By progressing through structured development phases, the portal evolved from a basic scaffolded project to a deployed system with Google OAuth authentication, TanStack-powered data tables, an atomic review workflow, naming conflict resolution, and batch management. The system demonstrates how modern Next.js 15 patterns — React Server Components, Server Actions, and URL-driven state — can be used to build lean, performant full-stack applications without the overhead of traditional client-side data management frameworks.

10.2 Observed Impact on eSim Operations

The IC Management Portal directly replaces a manually coordinated task assignment process with a structured, automated workflow. Interns can now independently discover and claim IC tasks, track their submission status, and receive actionable feedback on rejected work — all without requiring direct communication overhead with the admin team for routine operations. Admins gain a consolidated dashboard for monitoring batch progress, resolving requests, and maintaining catalog integrity, improving operational visibility substantially compared to prior manual approaches.

10.3 Closing Remarks

This internship was an invaluable opportunity to contribute a real, production-deployed tool to one of India's most prominent open-source educational initiatives. The experience of building software under the practical constraints of a real user base, a genuine dataset, and a production deployment pipeline has significantly

strengthened my understanding of modern full-stack development and professional software engineering practices.

I am deeply grateful to the FOSSEE eSim team and my mentor for the opportunity, structured guidance, and thoughtful feedback throughout this engagement.

Bibliography

1. FOSSEE Project, IIT Bombay. Available: https://fossee.in
2. eSim – Open Source EDA Tool. Available: https://esim.fossee.in
3. Next.js 15 Documentation. Available: https://nextjs.org/docs
4. Prisma ORM Documentation. Available: https://www.prisma.io/docs
5. Auth.js v5 (NextAuth) Documentation. Available: https://authjs.dev
6. shadcn/ui Component Library. Available: https://ui.shadcn.com
7. TanStack Table v8. Available: https://tanstack.com/table/v8
8. Neon Serverless Postgres. Available: https://neon.tech
9. Vercel Deployment Platform. Available: https://vercel.com/docs
10. Tailwind CSS v4 Documentation. Available: https://tailwindcss.com

Appendix

A Glossary of Terms

- **RSC — React Server Component:** A React component that renders on the server with no client-side JavaScript bundle.
- **Server Action:** A Next.js feature that allows server-side functions to be called directly from React components.
- **ORM — Object-Relational Mapper:** An abstraction layer over SQL queries; Prisma is the ORM used in this project.
- **OAuth — Open Authorization:** An authentication protocol; here implemented via Google's OAuth 2.0.
- **IC — Integrated Circuit:** The primary unit of work in eSim, representing a component whose simulation file interns create.
- **Neon:** A serverless PostgreSQL provider used as the cloud database for this project.
- **shadcn/ui:** Accessible, headless UI component primitives built on Radix UI.
- **TanStack Table:** A headless, framework-agnostic table and data grid library.
- **Prisma:** A type-safe ORM for TypeScript/Node.js with automatic migration support.

B Figure Index

The following screenshot placeholders appear throughout this report and should be replaced with actual application screenshots before submission:

- **Figure 1.1** — Application Entry Point / Sign-in Page
- **Figure 3.1** — Database Entity Relationship Diagram
- **Figure 4.1** — Intern Task Dashboard
- **Figure 4.2** — Intern IC Browse / Catalog Page
- **Figure 5.1** — Admin Dashboard Statistics

- **Figure 5.2** — Admin IC Detail / Alias Editing
- **Figure 5.3** — Admin Review Queue
- **Figure 6.1** — Google OAuth Sign-in Interface
- **Figure 7.1** — Technical Resolution Before/After
- **Figure 9.1** — Final Admin Dashboard
- **Figure 9.2** — Final Intern Dashboard
- **Figure 9.3** — Admin Review Queue (Final)