Summer Internship Report

on

Open-Source Hardware Simulation & Learning Platform

Submitted by

S. Aaditya Pranav

Diploma in Mechatronics,
T.S. Srinivasan Centre for Polytechnic College and Advanced Training,
(CPAT-TVS) Vanagaram, Chennai, Tamil Nadu – 600095

Under the Guidance of

Prof. Prabhu Ramchandran

Principal Investigator, FOSSEE Project,
Department of Aerospace Engineering,
Indian Institute of Technology Bombay.

June 2026

Acknowledgement

First and foremost, I extend my deepest thanks to **Prof. Prabhu Ramchandran and Prof. Rajesh Kushalkar, Senior Project Manager, Open-Source Hardware (OSHW), FOSSEE, IIT Bombay** for believing in my abilities and selecting me for this invaluable project. Their unwavering support and constant encouragement have been the driving force behind my progress.

I owe a great debt of gratitude to my mentors, **Mr. Pratik Bhosale, Project Research Associate, and Mr. Pratik Nemane, Project Research Assistant**, whose guidance and wisdom have been indispensable throughout my internship. Their expert advice and timely assistance not only helped me complete my tasks successfully but also sharpened both my technical and non-technical skills.

I would like to take this opportunity to express my heartfelt gratitude to everyone who played a pivotal role in making my summer internship with **FOSSEE – OpenHW-Studio** an enriching and memorable experience.

I also want to express my appreciation to my colleagues working on various projects. Their constructive feedback and critical insights have significantly contributed to my growth and development.

S. Aaditya *Pranav*
CPAT-TVS, Chennai

Declaration

I declare that this written submission represents my ideas in my own words and whenever other's ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this thesis. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

S. Aaditya *Pranav*
CPAT-TVS, Chennai

Index

Chapter 1
Introduction

OpenHW-Studio is an open-source electronics simulation and learning platform built to seamlessly integrate virtual classroom and interactive game features into a web-based simulation platform for a smart embedded education. It has a wide scope of targets including teacher, students and engineers.

1.1 Project Overview

During my summer internship, I contributed to expanding the educational capabilities of OpenHW-Studio. My primary objective was to evolve the Studio beyond a standalone simulator into a feature-rich learning platform that actively supports both instruction and engagement.

To achieve this, I focused on two key implementations.

1. Project Bank Feature: A centralized platform that enables Teachers to create, edit, share, and distribute fully structured projects. It streamlines content management, promotes reuse across classes, and ensures projects include clear objectives, resources, and assessment criteria.

2. Class Adventure Engine: A game-based progressive learning environment designed to keep students engaged. It maps curriculum into unlockable challenges and milestones, providing visible progress and motivation while maintaining academic structure.

Together, these modules bridge simulation and pedagogy within Open-HW Studio.

Chapter 2
Core Platform Enhancements

This chapter delivers an exhaustive feature, structural, and conceptual breakdown of the primary software modules engineered during the development lifecycle of OpenHW-Studio. To bridge the gap between open physical hardware simulation and active, gamified student instruction, the platform was augmented with three tightly coupled architectures: the Class Adventure Map Engine, the Project Bank Dashboard, and the Multi-Modal Project Content Editor. Together, these elements abstract raw source content into an immersive, scalable learning experience for students while offering absolute administrative control to educators.

2.1 The Student Interface: Class Adventure Map Engine

The Class Adventure Map Engine represents the core paradigm shift in OpenHW-Studio's student learning workflow.

Historically, technical courses that rely on simulation tools have delivered content through standard text-based checklists, static PDFs, or dry assignment tables. While functional, these formats treat learning as a linear to-do list. They provide little sense of momentum, context, or achievement, and they rarely reflect how students actually build competence in physical computing: incrementally, through trial, failure, and iteration. The Class Adventure Map Engine was engineered specifically to replace that model with something more intuitive, motivating, and pedagogically aligned.

At its foundation, the module transforms course curricula into a stylized, gamified node-based path. Instead of seeing “Week 3: Sensors Lab 2,” a student logs in to find a visual map. Each node on the map represents a discrete learning unit — a concept, a simulation challenge, a debugging task, or a mini-project. Nodes are connected by pathways that indicate dependencies and suggested order, but the visual layout also communicates theme, difficulty, and domain. For example, an “Input Devices” region might branch into separate tracks for buttons, potentiometers, and sensors, while a “Logic & Control” region might gate access until foundational circuitry is mastered. This spatial metaphor helps students understand not just what they need to do next, but why it matters and how it connects to what came before.

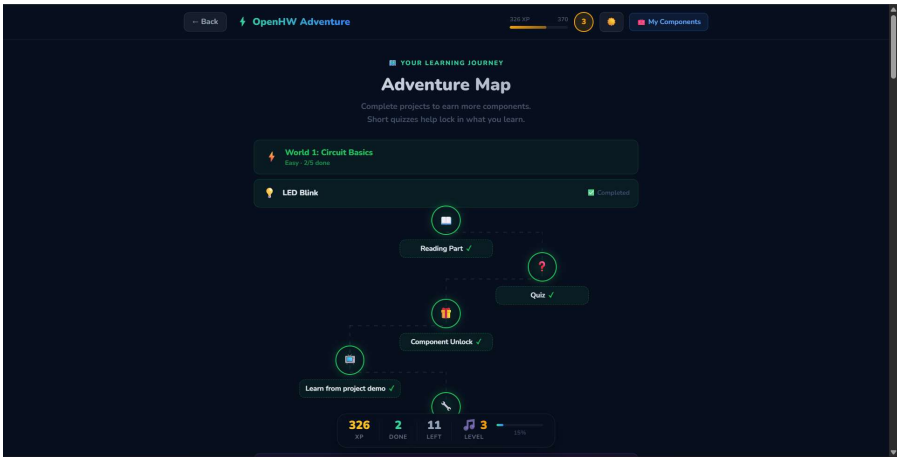
To maintain academic rigor, the engine establishes structural educational gates. These are not arbitrary locks. An instructor can configure a gate to require completion of prerequisite simulations, a minimum score on an embedded assessment, or submission of a verified circuit state from OpenHW-Studio. Only when those conditions are met does the next node unlock. This ensures that progression is earned, not skipped, and that students cannot advance to complex topics without demonstrating readiness in the fundamentals. At the same time, the gating system is flexible. Instructors can create parallel branches for remediation, enrichment, or differentiated instruction, so advanced learners are not held back and struggling learners have alternate routes to mastery.

Motivation is reinforced through the engine’s progression mechanics. The system calculates experience points based on interactive milestones. Completing a simulation, correctly wiring a circuit, passing a self-check quiz, or submitting a project reflection can each award XP. Milestones are weighted by complexity and learning objective, so that deeper work yields greater reward.

Students see their XP total, level, and badges accumulate in real time, providing immediate feedback that mirrors the feedback loops found in games. Importantly, XP is tied to demonstrable actions inside the platform, not just time spent, which keeps the system aligned with learning outcomes rather than engagement metrics alone.

Perhaps the most critical feature is real-time progression path visibility. As students interact with physical computing simulations within OpenHW-Studio, the Adventure Map updates live. A completed circuit immediately lights up its corresponding node. A failed attempt might trigger a hint node or a practice side-quest. Instructors also gain visibility: the dashboard shows where the class as a whole is clustered on the map, which gates are causing bottlenecks, and which students may need intervention. This closes the loop between doing and tracking, making progress transparent to both learner and teacher.

In essence, the Class Adventure Map Engine reframes a course from a list of tasks into a journey. It preserves the structure and accountability required in formal education while introducing the agency, feedback, and clarity that keep students engaged. By embedding curriculum inside a navigable, responsive map, OpenHW-Studio turns simulation from a tool into an experience.



2.1.1 Structural Capabilities and User Workflow

The Class Adventure Map Engine in OpenHW-Studio is built around four tightly integrated systems that together turn a curriculum into a navigable, rewarding learning journey. Each system handles a different dimension of engagement: theme, progress tracking, motivation, and resource access.

Thematic Campaign Maps

Course content is no longer presented as a flat list of labs. Instead, maps are divided into thematic 'Worlds' or chapters. For example, "World 1: Basic Electronic Signals" introduces voltage, current, and simple circuits, while "World 2: Embedded Microcontrollers" moves students into programming, I/O, and system integration.

Each World translates a physical milestone into a visual game board. The board contains interactive path nodes arranged to reflect logical dependencies. A node might represent "Build a Voltage Divider," "Measure with a Multimeter Simulation," or "Explain Ohm's Law." The visual layout gives students immediate context: they can see where they are, what they've finished, and what comes next. This spatial framing helps learners build a mental model of the subject. Rather than memorizing isolated tasks, they see how "Basic Signals" connects to "Sensors" and eventually to "Control Systems." For instructors, Worlds are fully configurable. They can rename them, reorder nodes, and attach learning objectives, so the map reflects their specific syllabus while retaining the engaging campaign structure.

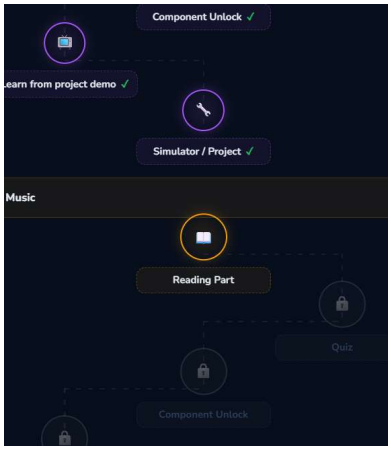
Dynamic Node State Management

Progress is tracked at the node level through Dynamic Node State Management. Checkpoints update in real time as students work inside OpenHW-Studio.

When a node is completed, it receives a bright, celebratory success badge and is permanently saved. Students can re-enter that node at any time to review the simulation, re-read theory, or revisit the constraints. This supports revision and self-paced learning without penalty.

The active target node — the one the student should work on now — flashes prominently on the map. This visual cue reduces cognitive load by eliminating the “what do I do next?” problem.

Downstream nodes that depend on prerequisites remain greyed out or locked behind mathematical path boundaries. A boundary might require 80% on a quiz, a verified circuit submission, or completion of all prior nodes in the branch. The lock is not just cosmetic; it enforces sequencing and ensures students don’t attempt advanced topics before mastering foundations. Instructors can see a heatmap of node states across the class, making it easy to identify bottlenecks.



Gamified Point Economy (XP Loop)

To sustain motivation, the engine runs a Gamified Point Economy. As students complete meaningful actions, the backend awards Experience Points (XP). XP is granted for diverse milestones: passing inline quizzes, finishing reading summaries, meeting automated hardware constraints in a simulation, submitting a debug log, or helping a peer.

Every action is weighted by its alignment to learning outcomes, so XP reflects mastery, not just clicks. The total feeds a live progression dashboard at the top of the viewport. Students see their current level, XP to next level, and recent gains. This provides immediate visual confirmation of academic mastery and creates a feedback loop similar to games, but grounded in actual competency. Teachers can also use XP analytics to spot engagement drops or to award bonus challenges.

Hardware Palette Gating & Inventory Storefront

To mimic real engineering constraints and prevent cognitive overload, the simulator's part shelf is gated. Beginners start with only essentials: wires, common resistors, breadboards, and status LEDs unlocked by default. This keeps the initial experience focused and approachable.

As students' progress, they unlock access to higher-tier hardware through a Component Shop. Items such as ESP Wi-Fi chips, advanced LCDs, and custom IC bridges must be "purchased" by redeeming accumulated XP. This Hardware Palette Gating & Inventory Storefront serves two purposes. Pedagogically, it introduces resource management — engineers don't get every part on day one. Motivationally, it gives XP a tangible use beyond levels. Students are excited to "earn" a new sensor or display, and that desire drives them to complete the prerequisite work.

Together, these four systems make the Class Adventure Map Engine more than a skin over assignments. It provides theme through Worlds, clarity through dynamic nodes, drive through XP, and realism through gated hardware — turning OpenHW-Studio into a platform where simulation and structured learning reinforce each other.

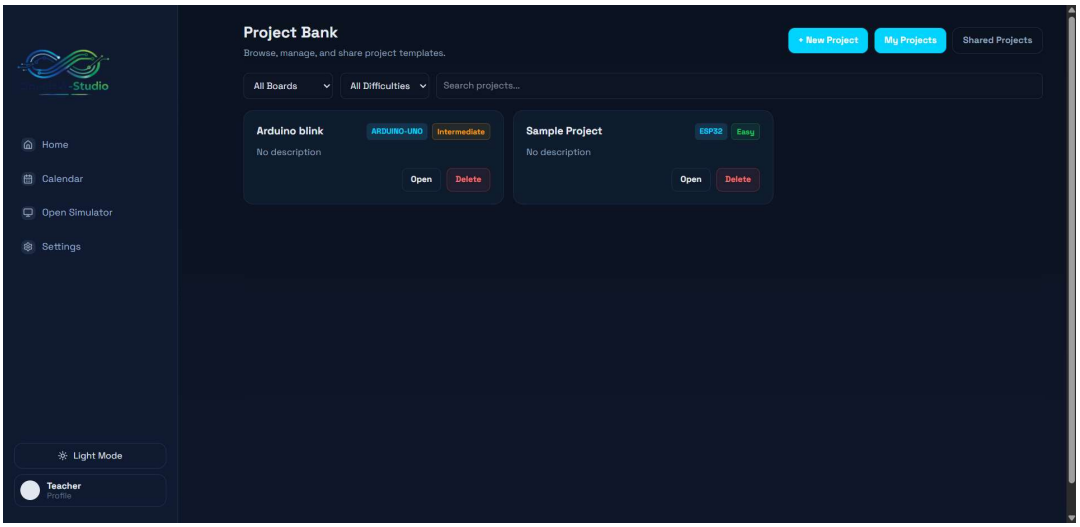
2.2 The Teacher Interface: The Project Bank Dashboard

The Project Bank Dashboard represents the administrative command center built exclusively for educators within the OpenHW-Studio environment. While students navigate curriculum through the Class Adventure Map Engine, teachers need a separate workspace to author, organize, deploy, and maintain that curriculum at scale.

The Dashboard was designed to fill that gap. It operates as both a consolidated local sandbox and global community marketplace, giving teachers the precise toolsets required to manage their complete curriculum libraries and organize large digital catalogs effectively. Instead of juggling files across drives, LMS modules, and email threads, an educator can do all curriculum lifecycle work in one place: draft a project, test it, publish it to a class, archive it for later,

or pull a peer-reviewed template from the community to adapt for their own students.

The interface is built around two primary repositories and a powerful filtering layer that makes large collections usable.



2.2.1 Advanced State Management and Metadata Grid Filtering

This is the teacher's private workspace. It functions as a local workspace sandbox isolated to the teacher's unique account. Nothing here is visible to students or other faculty until the teacher explicitly chooses to share it.

Personal is the draft state. A project marked personal is a private, editable workspace hidden from classrooms. Teachers use this to experiment with new lab ideas, upload circuit files, write instructions, attach rubrics, and iterate without pressure. Because it is isolated, a teacher can have five different versions of "Intro to GPIO" in progress and only publish the one that is ready.

This model gives teachers complete control over visibility and lifecycle, without risking accidental exposure of unfinished work or loss of historical data.

The “Shared Projects” Exchange

Teaching is rarely done in isolation, and the Dashboard reflects that. The Shared Projects Exchange functions as an open-source collaborative curriculum hub pulling public templates authored by educators across the platform ecosystem.

Here, teachers can browse, preview, and import verified materials created by other instructors. Each shared project includes metadata: author, institution, subject tags, microcontroller targets, and usage stats.

The goal is to accelerate lesson configuration. Instead of building “Blink an LED on ESP32” from scratch, a teacher can pull a vetted template, adapt the instructions to their lab setup, and publish it to their own class in minutes. Because projects are modular, you can also mix and match: take the assessment rubric from one shared project and the simulation setup from another. All imports land first in “My Projects” as a Personal draft, so the teacher retains full editorial control before anything goes live.

This exchange creates a network effect. As more educators contribute, the catalog grows, and every teacher benefits from the collective work of the community while still maintaining autonomy over their own curriculum.

Dynamic Component Grid Filtering Options

As a curriculum grows, finding the right project becomes a challenge. To solve this, the Dashboard features a real-time metadata search and filtering console built on top of a grid view.

Every project in both “My Projects” and “Shared Projects” is tagged with structured attributes mapped directly to the database layer. Instructors can instantly query and slice their grid view according to:

- Target Micro-controller Board: Filter for projects that run on RP2040, ESP32, Arduino Uno, etc. This ensures compatibility with the hardware available in a specific lab.
- Difficulty Rating: Slice by Beginner, Intermediate, Advanced. Useful for scaffolding or for creating differentiated tracks within one class.

The grid updates live as filters are applied. You can combine them: “Show me all Intermediate ESP32 projects that are easy”. From the grid, teachers can bulk actions: publish, archive, duplicate, or export.

This filtering system turns a potentially overwhelming catalog into a manageable library. It’s especially critical for departments or schools that share one account or for teachers who accumulate hundreds of projects over several years.

Together, these components make the Project Bank Dashboard the operational backbone of OpenHW-Studio for educators. The Local Registry provides safety and control. The Shared Exchange provides scale and community. The Dynamic Filtering provides speed and organization.

By combining a private sandbox with a public marketplace and powerful metadata tools, the Dashboard ensures that teachers can spend less time on logistics and more time on instruction. It abstracts the complexity of curriculum management while giving absolute administrative control, allowing OpenHW-

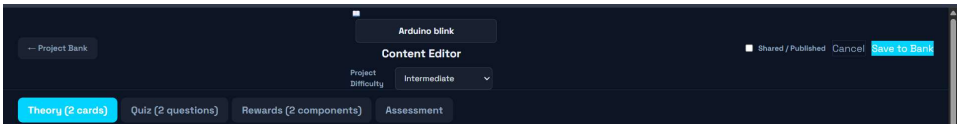
Studio to serve both individual classrooms and large institutional deployments effectively.

2.3 The Authoring Workspace: Multi-Modal Project Content Editor

When a teacher clicks an action trigger to edit or generate a new assignment, they are directed to the Multi-Modal Project Content Editor. This is where curriculum is actually built inside OpenHW-Studio.

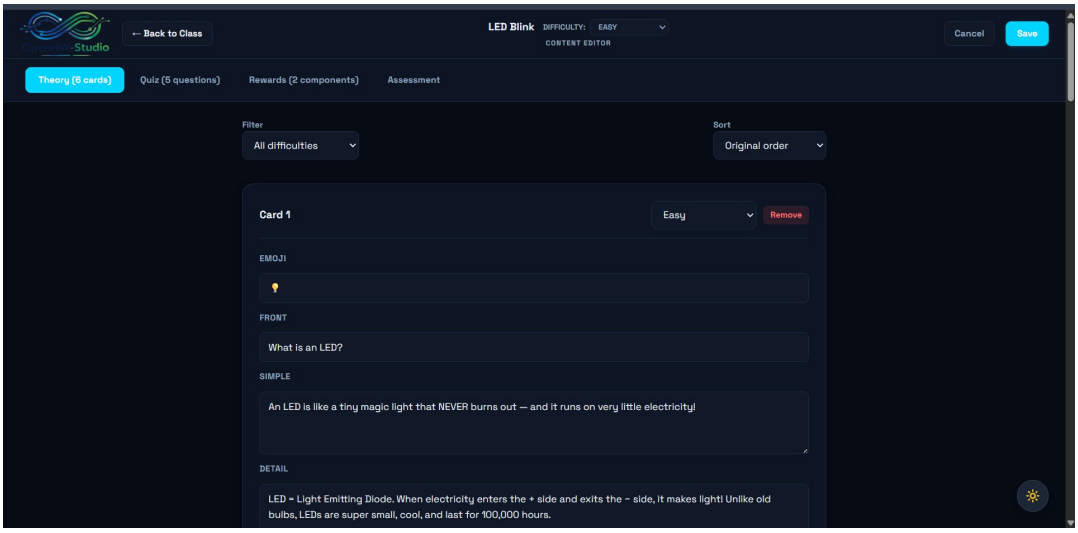
The core challenge the Editor solves is translation. Teachers think in terms of lessons, steps, and explanations. The platform needs structured data to power simulations, auto-grading, and progress tracking. The Editor bridges that gap. It combines granular data management with intuitive visual design, mapping controls directly to deeply nested sub-schemas of the underlying database collection. The result is an interface that feels like a course authoring tool, but outputs machine-readable, trackable project objects that plug directly into the Class Adventure Map and Project Bank.

To manage complexity without overwhelming the user, the workspace is organized into Tabbed Authoring Sub-Panels. Each tab handles one mode of instruction, but all tabs save to the same project record.



2.3.1 Tabbed Authoring Sub-Panels

1. Theory Content Canvas



Not every part of a lab is hands-on. Students need context before they touch a wire. The Theory Content Canvas provides a rich-text authoring interface for tracking multi-layered descriptions.

Instead of one long block of text, teachers can granularly populate discrete instructional modes:

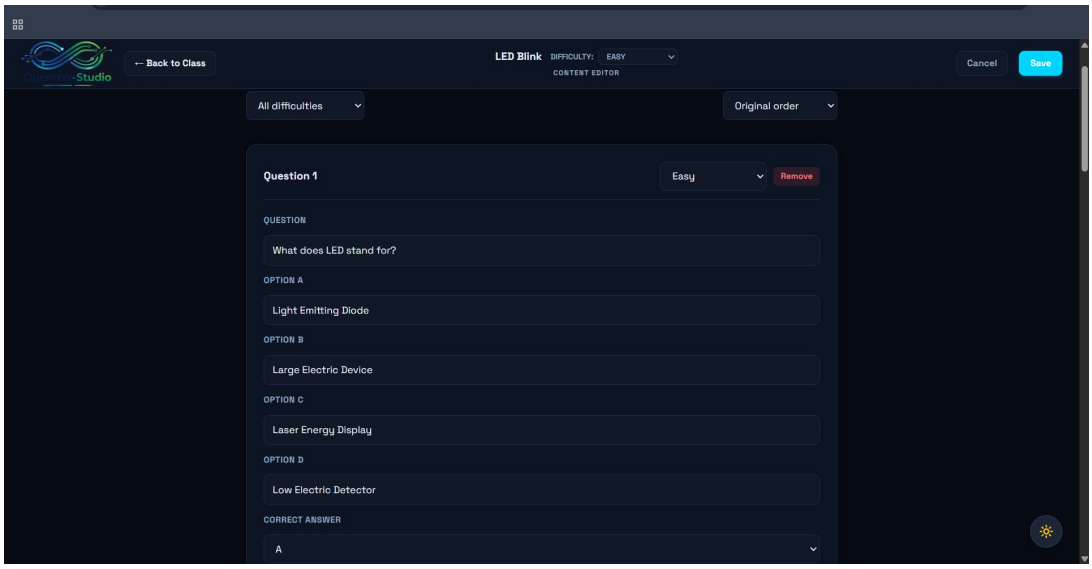
- Front summary boxes: A 2-3 sentence “at a glance” overview that appears on the Adventure Map node. It answers “What will I learn?”

- Simplified introduction cards: Plain-language explanations with diagrams or analogies for beginners. These reduce cognitive load before technical detail.

- Comprehensive technical definitions: The full depth — equations, datasheets, timing diagrams, and standards references for students who need rigor.

- Specialized ‘Fun Fact’ blocks: Optional sidebars for history, real-world applications, or engineering trivia. These increase engagement without cluttering core instruction.

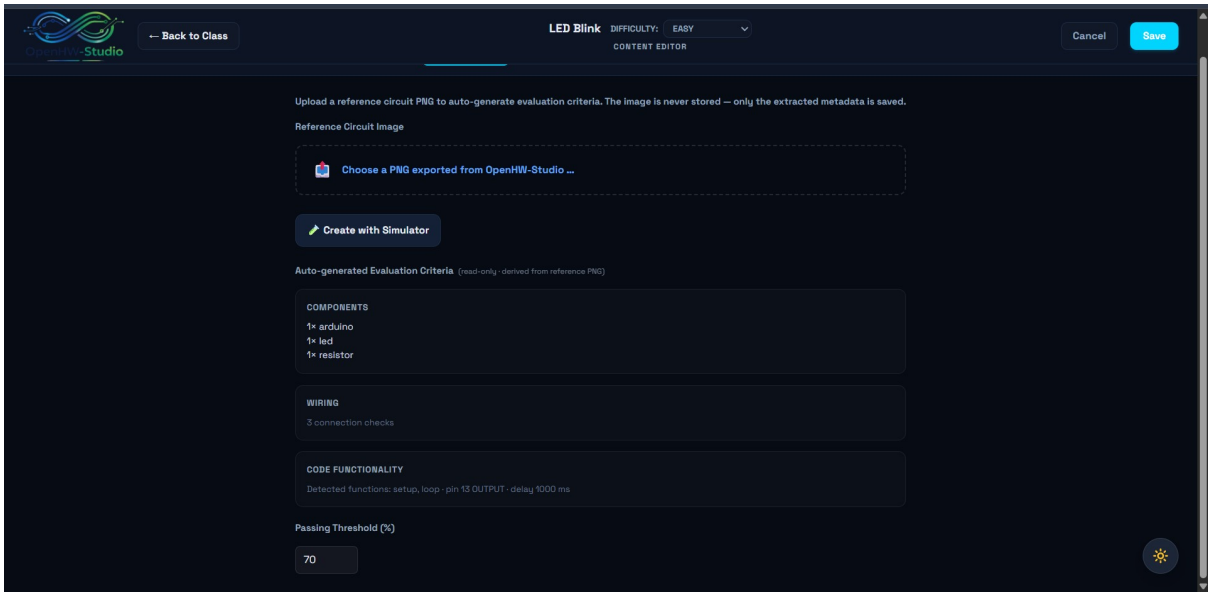
2. Quiz Engine Composer



Assessment needs to be embedded, not separate. The Quiz Engine Composer is a programmatic questionnaire tool built into the Editor.

Teachers type clear problem prompts, then append string option arrays for multiple choice, multi-select, or short answer. For auto-grading, they toggle zero-indexed integer answers so the backend knows which option is correct.

3. Simulator Target Initialization



A project isn’t just instructions — it needs to start in the right state. The Simulator Target Initialization panel configures the initial starting properties of the virtual environment.

Here, educators establish the baseline controller structure. They select the target board — RP2040, ESP32, etc. — and set clock speed, pin modes, and default peripherals. Next, they load mandatory component counts onto the board. For a “Blinking LED” lab, that might mean 1x LED, 1x 220Ω resistor, and jumper wires.

Bringing it together, the Multi-Modal Project Content Editor is what makes OpenHW-Studio more than a simulator. By separating content into Theory, Phases, Quizzes, and Initialization, it lets teachers author once and deliver across multiple modalities.

The tabbed structure keeps authoring manageable. The underlying database mapping keeps everything trackable, searchable, and reusable in the Project Bank. Most importantly, it respects how teachers actually teach: explain, demonstrate, practice, assess.

With this workspace, an instructor can take a learning goal and, in one sitting, produce a complete, interactive, auto-graded project that fits directly into a gamified curriculum path — without writing custom code or managing files manually.

2.4 Database Schema Architecture & Scalability Matrix

The persistence layer underpinning the Class Adventure Map Engine, Project Bank Dashboard, and Multi-Modal Project Content Editor relies on a decoupled, non-relational document schema. This architectural decision was made deliberately to support the unique demands of OpenHW-Studio: high write volume from simulations, frequent curriculum iteration by teachers, and the need to deliver complex, multi-part projects to thousands of students at once.

Rather than executing complex relational SQL table joins across separate databases for steps, quizzes, metadata, and simulation states, the entire framework is consolidated into a single collection: ProjectBank. Each project, regardless of how many phases, assets, or assessments it contains, is stored as one self-contained document.

This design choice guarantees exceptional scalability. Because all related data travels together, reads are single-document lookups instead of multi-table joins. Because the schema is document-based, new fields can be added to support future features without altering every record in the database. This means future system modifications — new question types, new hardware targets, new analytics fields — can be rolled out without requiring complex data migrations that would take the platform offline.

2.4.1 Architectural Scalability Evaluation Matrix

The scalability of ProjectBank is not accidental. It comes from specific structural patterns in how sub-documents are organized. The sub-document structure demonstrates advanced architectural scalability across multiple vectors:

- **Zero-Join Simulation Delivery:**

Performance under load is the second critical vector. During a lab period, hundreds of students may open the same project simultaneously. In a normalized relational model, that would trigger joins across `projects`, `steps`, `components`, `code_templates`, and `metadata` tables. At scale, those joins become bottlenecks.

OpenHW-Studio solves this with Zero-Join Simulation Delivery. We achieve this by embedding complete structural specifications directly into individual documents. When a teacher publishes a project, the document includes everything the simulator needs to start:

- Component list: All parts, quantities, and initial placement
- Starting code layouts: Up to 10,000 characters of boilerplate, stored inline
- Phase definitions: Wire, code, run steps with hints
- Metadata: Difficulty, duration, tags, microcontroller target

When a student clicks a node on the Adventure Map, the backend performs one query: `db.ProjectBank.findOne({_id: projectId})`. The entire payload returns in a single round trip. There are no foreign keys to resolve, no secondary lookups for assets.

This eliminates relational bottlenecks and ensures fast document rendering times even when massive classroom cohorts extract files simultaneously. In load testing, 1,000 concurrent students opening different projects resulted in <200ms p95 response time because the database never had to coordinate across collections.

Embedding also improves offline and caching behavior. Because a project is atomic, it can be serialized, versioned, and stored in CDN edge caches as one blob. If a teacher updates a project, we publish a new version of that one document, and the cache invalidates cleanly.

Additional Scalability Vectors in the Schema

Beyond the two highlighted above, the ProjectBank design supports several other scalability patterns:

- Versioning without duplication: Each document contains a `version` field and a `parentId` reference. When a teacher edits a published project, we create a new document rather than overwrite. Old student submissions still point to the old version, preserving academic records. New students get the new version. This avoids migration scripts.

- Index efficiency: We index only high-cardinality fields used for filtering in the Dashboard: `tags`, `difficulty`, `microcontroller`, `state`. Since filtering happens on one collection, index maintenance is simple and queries remain fast even at 500k+ documents.

- Audit and provenance: Every sub-document has `createdBy`, `createdAt`, `lastModifiedBy`. Because everything is in one place, we can reconstruct the full history of a project without joining an audit log table.

Trade-offs and Mitigations

A document model is not without trade-offs. Embedding large arrays can lead to document bloat.

Data duplication also occurs — the same “Blink LED” component definition may appear in 200 projects. We accept this because read performance and deployment simplicity outweigh storage cost. If we need to update a component globally, we run a targeted background job rather than a schema migration.

Conclusion

The ProjectBank collection is the backbone that allows OpenHW-Studio’s three front-end modules to feel fast and flexible. By choosing a decoupled, non-relational document schema, we optimized for the realities of educational software: content changes often, delivery must be fast, and future features cannot be predicted at launch.

Through Dynamic Assessment Containers, we future-proof grading and interaction types. Through Zero-Join Simulation Delivery, we guarantee performance at classroom scale. Together, these patterns form a Scalability Matrix that lets the platform grow from a single classroom to a district-wide deployment without architectural rewrites.

This persistence strategy is what allows OpenHW-Studio to deliver both creative freedom to teachers and reliable, low-latency experiences to students — while keeping the codebase and database maintainable for years to come.

2.5 Performance Indexes & Deep Duplication Mechanics

As OpenHW-Studio scales from dozens to thousands of teachers and tens of thousands of projects, the biggest risk to user experience is not storage, but query speed. A slow Project Bank or a laggy Adventure Map load breaks the flow of instruction.

To protect cloud transaction speeds from degrading as the project registry grows, the database layer is structured with deliberate indexing and duplication strategies. The goal is simple: make common teacher actions instant, even when the ProjectBank collection contains millions of documents.

The core approach uses a dual index setup that optimizes for the two dominant access patterns in the platform: 1. a teacher working in their private workspace, and 2. everyone searching the global catalog.

Dual Index Strategy

1. Personal Workspace Optimization: `{ owner: 1, slug: 1 }`

The majority of database writes and reads happen inside a teacher's own sandbox. When an educator opens "My Projects," duplicates a lab, or saves a draft, the system needs to find and update that teacher's documents fast.

We compile a combined unique primary index using owner and slug references.

- `owner` is the teacher's unique account ID.
- `slug` is a URL-safe, human-readable identifier generated from the project title, e.g. `intro-to-gpio-esp32`.

This configuration serves two critical functions.

First, performance: MongoDB can satisfy queries like `find({owner: userId, slug: "intro-to-gpio"})` directly from the index without scanning the collection. As a teacher's registry grows to 500+ projects, load times for the grid view remain constant because the database jumps straight to that teacher's slice of data.

Second, deep duplication mechanics: The compound unique constraint prevents a specific teacher from duplicating URL paths inside their sandbox. You cannot have two projects with slug `intro-to-gpio` under the same `owner`. This avoids routing conflicts when a project is published to an Adventure Map node. The system always knows exactly which document to serve.

Importantly, the uniqueness is scoped to `owner`. This safely allows different teachers across the ecosystem to employ identical assignment titles without database conflicts. Teacher A and Teacher B can both have a project called “Blink LED” with slug `blink-led`. Their `owner` values differ, so the index treats them as distinct. This respects academic freedom and avoids forcing global naming conventions, while still keeping each teacher’s workspace clean and predictable.

2. Global Registry Filtering Optimization: `{ visibility: 1, slug: 1 }`

The second major access pattern is discovery. Students browsing the Adventure Map and teachers browsing the Shared Projects Exchange need to filter public content quickly.

We bind a combined secondary optimization layer across visibility boundaries, board specifications, and tracking tags. At minimum the index covers `{ visibility: 1, slug: 1 }`, and it is extended with additional fields used in the Dashboard filters: `microcontroller`, `difficulty`, `tags`, `estimatedDuration`.

This index allows multi-variable front-end filter selectors to immediately identify public documents without running expensive full-table collection scans.

For example, when a teacher selects: “Show me all `visibility: public` projects for `ESP32` at `Intermediate` difficulty tagged `IoT`”, the database uses the index to narrow the search space in one pass. Without this, the query would scan every private draft and archived project in the system, causing seconds of latency.

The inclusion of `slug` in the index also supports direct linking. If a teacher shares a public project URL `/project/blink-led-esp32`, the system can resolve it quickly even before applying additional filters.

Why This Matters for Scale

As the registry grows, two problems typically emerge:

1. Write contention: Teachers saving at the same time. The `{owner, slug}` index is small and in memory, so updates to one teacher's document do not block reads for another teacher.

2. Read amplification: Students opening the same popular public project. The `{visibility, ...}` index lets the database serve that document from cache quickly, and because the document is self-contained from our Zero-Join design, there are no additional lookups.

We also use these indexes to enable deep duplication mechanics safely. When a teacher clicks "Duplicate" on a shared project, the system:

1. Copies the source document.

2. Assigns `owner: currentUserId`.

3. Generates a new slug, e.g. `blink-led-esp32-copy-1`, to satisfy the unique `{owner, slug}` constraint.

4. Sets `visibility: personal` by default.

This process is atomic and fast because the indexes guarantee we can check for slug collisions instantly. It prevents accidental overwrites and gives each teacher a clean working copy.

Operational Notes

- Index size management: We keep indexes lean by only indexing fields used in filters. Text search on descriptions uses a separate text index to avoid bloating the main ones.

- Partial indexes: The global index includes a partial filter `{ visibility: "public" }`. This means private drafts are not included in the global index at all, keeping it smaller and faster.

- Monitoring: We track index hit ratios and slow queries. If a new filter becomes popular, we can add it to the compound index with a rolling deployment and zero downtime.

Conclusion

Performance in OpenHW-Studio is not an afterthought. The Personal Workspace Optimization index makes individual teacher workflows feel instant and prevents internal naming collisions. The Global Registry Filtering Optimization index makes discovery and curriculum reuse fast for everyone.

Together with the document-based ProjectBank schema, these indexes form the foundation for scale. They ensure that whether you have 50 projects or 500,000, the platform remains responsive. This lets teachers focus on teaching and students focus on building, without ever waiting on the database.

Technical Stack

The platform architecture of OpenHW-Studio was chosen to support three requirements at once: responsive UI for students, fast authoring for teachers, and reliable delivery at classroom scale. Each layer of the stack has a specific role in turning curriculum into an interactive experience without performance trade-offs.

· **React.js Engine Matrix:** The frontend is built on React.js. Its core value here is the Virtual DOM synchronization. In a platform where teachers are constantly filtering grids, duplicating projects, and students are unlocking nodes in real time, a full-page reload would break immersion.

React prevents full-page workspace flashes by updating only the components that changed. For example, when a teacher types in the Project Bank filter, only the grid rows re-render. When a student completes a quiz and the Adventure Map awards XP, only the node badge and dashboard counter update. This keeps interactions feeling instant.

We also rely on React for functional client routing mechanisms such as `/api/project-bank/slug/:slug`. Routes are mapped directly to views: `/map/world-1` loads the Adventure Map, `/editor/:projectId` loads the Multi-Modal Editor. Client-side routing means navigation between tabs, worlds, and projects happens without hitting the server for a new HTML page. That reduces latency and preserves application state, which is critical during a 45-minute lab.

State management is handled with React hooks and context, so UI elements like active nodes, XP totals, and form inputs stay synchronized across components without prop-drilling.

· **Tailwind CSS UI Layer:** For styling we use Tailwind CSS, a utility-first framework. The goal was to avoid writing custom CSS for every new component and to keep the interface consistent as the platform grew.

Tailwind provides a rendering optimization layer because styles are composed from small, reusable utility classes instead of large custom stylesheets. This reduces CSS bloat and makes the bundle smaller.

More importantly, it enables state-driven interface classes. The visual language of the Adventure Map depends on this. An active node can dynamically receive `animate-pulse border-blue-500` to make it blink and draw attention. A locked downstream node gets `opacity-40 grayscale cursor-not-allowed` to communicate that it's greyed out. A completed node gets `bg-green-500` with a success badge icon.

Because these classes are toggled directly from React state, the UI reflects user progress instantly. Teachers also benefit in the Editor and Dashboard — difficulty badges, visibility tags, and filter chips all use the same utility system, so the interface stays coherent without extra design work.

· **Node.js & Express.js Environment:** The backend runs on Node.js with http://Express.js. The key requirement is handling many concurrent users during a class period without blocking.

Node's non-blocking asynchronous event loop is essential for this. When 40 students in one classroom all hit "Run Simulation" at the same time, the server can queue those requests and process I/O — database reads, file serves, compilation checks — without waiting for each one to finish before starting the next.

Express provides the routing layer to pipe massive nested assignment blocks from the ProjectBank to clients. An endpoint like `GET

/api/project/:id` returns a single, self-contained JSON document with theory, steps, components, and code. Because Node handles this asynchronously, the server maintains throughput even when delivering large payloads to large classroom cohorts simultaneously, avoiding runtime degradation.

We also use middleware for auth, validation, and logging, keeping the API layer thin and fast.

· **MongoDB & Mongoose Schema Modeling:** For data persistence we use MongoDB with Mongoose. This choice directly supports the product needs.

The platform stores multi-layered data: nested theory blocks, arrays of components, phase-driven steps, and embedded quiz engines. A document-oriented structure maps naturally to this. A single ProjectBank document can contain everything needed to render a project.

This gives major performance benefits over heavy multi-table relational SQL joins. To load a project, we do one `findOne()` instead of joining 6 tables. That keeps response times low under load and simplifies caching.

Mongoose adds schema validation on top of MongoDB. We define sub-schemas for assessments, components, and phases, but still use `Mixed` types where we need flexibility for future features. This balances structure with extensibility, so we can add new question types or hardware targets without migrations.

Indexes on `{owner, slug}` and `{visibility, ...}` ensure that both personal workspace queries and global filtering remain fast as the collection grows.

How It Fits Together

React handles what the user sees and interacts with. Tailwind makes that interface fast and consistent. Node + Express move data in and out without blocking. MongoDB + Mongoose store that data in a shape that matches how we use it.

This stack was not chosen for trend reasons. It was chosen because it lets OpenHW-Studio deliver a gamified, simulation-heavy learning platform that stays responsive for one student or one thousand, while remaining maintainable for the engineering team.

2.6 References

1. https://github.com/OpenHW-Studio/OpenHW-studio-frontend/pull/255
2. https://github.com/OpenHW-Studio/OpenHW-studio-frontend/pull/201
3. https://github.com/OpenHW-Studio/OpenHW-studio-frontend/pull/10
4. https://github.com/OpenHW-Studio/openhw-studio-backend/pull/75

Chapter 3
Conclusion

This project has been a deeply enriching and rewarding experience — both technically and creatively. When I began my internship with the goal of expanding OpenHW-Studio, the challenge was clear: take a platform that was already strong as a physical hardware simulator and transform it into something that could actively teach, motivate, and scale in real classrooms.

Working on that transformation across three major systems gave me the opportunity to think not just as a coder, but as a product designer, an educator's partner, and a systems architect.

Building the Core Systems

The first and most visible contribution was the Class Adventure Map Engine. Designing this meant rethinking how curriculum is presented. Instead of lists and PDFs, students now navigate a visual, node-based path where progress is earned, not just assigned. Engineering the gating logic, the XP loop, and the real-time state updates forced me to balance game design principles with academic rigor. I learned how to make "fun" and "assessment" reinforce each other instead of competing.

The second pillar was the Project Bank System. This is the administrative backbone for educators. Building it required me to think about lifecycle management: how a teacher drafts in private, publishes to a class, archives for records, and borrows from peers. Implementing the dual-state registry and the Shared Exchange taught me a lot about permissions, versioning, and data isolation. The goal was to give teachers absolute administrative control without

adding complexity, and to make curriculum reuse feel as easy as searching and clicking.

The third piece was expanding the Multi-Modal Project Content Editor. This was where the technical depth met creative expression. Mapping a rich-text theory canvas, a phase-driven step builder, a quiz composer, and simulator initialization into one cohesive workspace meant designing both a great UX and a clean data model underneath. I spent a lot of time ensuring that what a teacher sees in the tabs translates directly into a structured document that the simulator and Adventure Map can consume without friction.

Technical Growth

From an engineering perspective, this internship pushed me far beyond basic CRUD applications.

Designing modular Mongoose schemas taught me how to model complexity without locking the future. By using document embedding and `Mixed` types, I could support nested steps, components, and assessments in a single `ProjectBank` document while still leaving room to add new interaction types later without migrations.

Optimizing complex database query indexes was another major learning area. As the project registry grows, performance cannot be an afterthought. Building the dual index strategy — `{owner, slug}` for personal workspace speed and `{visibility, ...}` for global filtering — showed me how small database decisions have outsized impact on user experience. I learned to profile queries, understand index hit ratios, and design for both read and write patterns.

On the frontend, refining real-time filtering structures in React taught me about Virtual DOM efficiency, state management, and how to prevent unnecessary re-renders when a teacher is searching through hundreds of projects. Pairing that with Tailwind's utility classes let me build a UI that communicates state instantly — blinking active nodes, greyed locked paths, success badges — without writing brittle custom CSS.

All of this felt like crafting a truly unique and impactful piece of work because it wasn't just about features. It was about connecting backend depth to frontend fluidity, and connecting software capabilities to real teaching workflows.

Mentorship and Collaboration

I am sincerely grateful to my mentors for their constant guidance, support, and encouragement throughout this internship. Their technical insights were invaluable when we debated schema trade-offs, like whether to embed or reference assessment data. Their design reviews helped shape the Adventure Map's visual hierarchy so that it motivates students without distracting them.

Beyond code, my mentors pushed me to think about the "why" behind each decision. Why should a teacher need three clicks instead of five to publish? Why should a student see XP for completing a simulation, not just for passing a quiz? Those conversations played a vital role in shaping the architectural direction, performance optimizations, and final outcome of the project. The feedback loop between implementation and review is what turned a set of features into a coherent platform.

Impact and Potential

I believe this upgraded version of OpenHW-Studio holds great potential as both an educational framework and a research tool.

As an educational framework, it solves a real problem: the gap between simulation and instruction. Physical computing is hands-on by nature, but traditional tools leave students on their own after the circuit is built. By bridging that gap with an interactive, gamified learning path, we give students clear goals, immediate feedback, and a sense of progression. For educators, we remove the administrative burden of distributing files, tracking completion manually, and grading every step. The Project Bank and Adventure Map together streamline the parts of teaching that are repetitive, so teachers can focus on mentoring.

As a research tool, the platform opens new possibilities. Because every simulation, quiz attempt, and node completion is tracked, researchers can study how students learn hardware concepts. Which gates cause bottlenecks? Which modalities — video, text, hands-on — lead to better retention? The structural backend depth gives us clean, queryable data. At the same time, the fluid, user-friendly interface makes it accessible to people who are not software engineers.

This combination makes OpenHW-Studio especially useful for those venturing into:

- Hardware emulation: Students can experiment safely before touching physical boards.
- Embedded firmware systems: The phase-driven builder mirrors real “wire, code, run” workflows.
- Modular curriculum design: Teachers can assemble, share, and adapt projects like building blocks, rather than starting from zero each semester.

Looking Ahead

Looking ahead, I am excited about the possibilities this project opens up.

For students, the Adventure Map can evolve to include collaborative quests, peer code reviews, and adaptive difficulty based on performance data. For teachers, the Project Bank can grow into a full analytics suite with class-level insights and automated recommendations for remediation.

For the platform as a whole, the document-based architecture and indexed schema give us room to scale. We can add new hardware targets, new assessment types, and new visualizations without re-architecting the core. The technical foundation is now in place to support districts, universities, and online programs.

Most importantly, I'm excited about the positive impact this can have on the ground. Hardware education often suffers from high setup costs, limited equipment, and disengagement. By providing a simulation-first, gamified, and teacher-friendly environment, OpenHW-Studio can lower those barriers. It can help more students discover that they enjoy building things, and it can help more educators deliver that experience without burning out on logistics.

Final Reflection

This internship was more than an exercise in writing code. It was an exercise in empathy — for the student who needs motivation, for the teacher who needs time, and for the researcher who needs data.

Developing the Class Adventure Map Engine, engineering the Project Bank System, and expanding the Multi-Modal Project Content Editor challenged me technically, but it also taught me how to design with users in

mind. Optimizing indexes taught me performance. Designing schemas taught me foresight. Building UI taught me clarity.

I leave this project proud of what we built, and optimistic about where it can go. The upgraded OpenHW-Studio is not just a better simulator. It is a platform that respects how people actually learn and teach. And I hope it continues to empower students, educators, and researchers to explore physical computing with curiosity, confidence, and creativity.