Semester Long Internship Report

on

OpenHW-Studio

Submitted by

Akshat Singh Tomar

VIT Bhopal University

Under the Guidance of

Prof. Prabhu Ramachandran

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

4 June 2026

Project Website: https://openhw-studio.fossee.in/
GitHub Repository: https://github.com/OpenHW-Studio/OpenHW-studio-frontend

Acknowledgement

I would like to take this opportunity to express my heartfelt gratitude to everyone who contributed to making my semester-long internship an enriching, challenging, and deeply memorable learning experience. The six months I spent working on OpenHW Studio have shaped not only my technical abilities but also my understanding of what it takes to build software that genuinely helps people learn.

I express my sincere gratitude to Prof. **Prabhu Ramachandran** for providing me with the opportunity to be part of the FOSSEE Internship Program at IIT Bombay. His vision of promoting open-source software development and engineering education has inspired countless students and researchers to actively contribute to the open-source ecosystem, and I am grateful to have been one of them.

My deepest appreciation goes to Mr. **Rajesh Kushalkar**, Senior Project Manager, Open Source Hardware (OSHW), FOSSEE, IIT Bombay, for his constant guidance, encouragement, and invaluable support throughout the internship. His leadership, insightful suggestions, and continuous motivation played a significant role in the successful design and development of the Gamified Adventure Map for Arduino and ESP32. Every review session with him pushed me to think more carefully about the student experience and to justify every design decision I made. His mentorship greatly enhanced both my technical expertise and my professional growth.

I would also like to express my sincere gratitude to Mr. **Pratik Bhosale**, Project Research Associate, and Mr. **Pratik Nemane**, Project Research Assistant, for their valuable guidance, technical assistance, and constructive feedback throughout the internship. Their timely support during code reviews, their patience while debugging tricky state-management issues with me, and their collaborative approach were instrumental in overcoming challenges and ensuring the successful completion of the project.

I am also grateful to the entire OpenHW Studio team for fostering a collaborative and inspiring work environment. The weekly stand-ups, the constructive design discussions, the teamwork, and the knowledge-sharing culture provided me with valuable exposure to real-world open-source software development practices and significantly enriched my learning experience.

Finally, I extend my heartfelt thanks to FOSSEE, IIT Bombay, for providing an excellent platform to learn, innovate, and contribute to meaningful open-source initiatives that directly benefit students across the country. The knowledge, practical experience, and professional values I gained during this internship will remain invaluable throughout my academic and professional journey.

Declaration

I declare that this written submission represents my ideas in my own words, and wherever others' 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 report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented, fabricated, or falsified any idea, data, fact, or 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.

Akshat Singh Tomar
VIT Bhopal University

Abstract

OpenHW Studio is an educational web platform developed under the FOSSEE (Free and Open Source Software for Education) initiative at IIT Bombay to make electronics and embedded-systems education accessible, simulation-driven, and free of cost for students who may not have access to physical Arduino or ESP32 hardware. While the platform already offered a functional circuit simulator, a code editor, and a bank of curated projects before this internship began, student feedback and internal usability reviews had identified a recurring problem: learners were being shown the platform's full catalogue of components and projects on day one, with no sense of structure, sequencing, or achievement. This flat, unordered presentation made the platform feel more like a reference tool than a guided course, and it did little to sustain motivation for beginners who had no prior exposure to circuits or firmware.

This report documents a six-month internship carried out with the OpenHW Studio team, during which I designed and implemented a gamification layer for the platform, built around three interlocking systems: an Adventure Map that visualises a student's learning path as a connected sequence of stages; a component-locked progression mechanic that grants access to new electronic components only as students complete the corresponding stages; and a card-based micro-learning system that introduces a single concept before every stage so that students are never dropped into a project cold. These three systems were connected end-to-end with the platform's existing project guides, quizzes, and simulator so that completing an assessment automatically unlocks the next stage and the next set of components, with progress persisted per student through a dedicated backend service.

The work involved reverse-engineering and extending a non-trivial existing React codebase, designing new data models and service-layer abstractions, building a global gamification context to keep multiple pages of the application synchronised, and iterating on the visual and interaction design of the map, the lock states, and the learning cards based on feedback from my mentors. Beyond the Arduino and ESP32 Adventure Maps that shipped as part of this internship, the underlying architecture was built to be reusable for the teacher-facing class view, where instructors can customise the journey and quizzes for their own classes.

The result is a gamification system that is live on the OpenHW Studio platform today, transforming what was previously a flat list of projects into an explorable, reward-driven journey. This report walks through the motivation, design process, architecture, implementation details, testing approach, challenges encountered, and the broader lessons learned over the course of the internship, and concludes with a set of concrete directions for future work, including an AI-assisted debugging companion, adaptive difficulty, and badges and leaderboards.

Table of Contents

Chapter 1: Introduction

1.1 About FOSSEE and Open Source Hardware

FOSSEE (Free and Open Source Software for Education) is a project funded by the Ministry of Education, Government of India, and hosted at IIT Bombay. Its mission is to reduce the cost of education for students across the country by promoting the use of free and open-source tools in place of proprietary software wherever possible. Under this umbrella, the Open Source Hardware (OSHW) group works specifically on tools that make electronics, embedded systems, and hardware design accessible to students who may not have access to physical lab equipment, expensive licensed CAD tools, or dedicated electronics workshops.

OpenHW Studio is one of the flagship initiatives of this group. It is a browser-based platform where students can drag and drop electronic components onto a virtual breadboard, wire them together, write Arduino or ESP32 firmware in an integrated code editor, and simulate the resulting circuit — all without owning a single physical component. This lowers the barrier to entry for embedded-systems education dramatically, particularly for students in schools and colleges with limited lab budgets.

1.2 Project Overview

Before this internship began, OpenHW Studio already had a working simulator, a code editor, and a Project Bank containing a curated set of Arduino and ESP32 projects that students could open individually. Each project came with a guide, a set of build instructions, and in many cases an automatically scored assessment. What the platform lacked was a sense of progression: students could open any project in any order, all components were visible and usable from the very first login, and there was no visual representation of how much of the curriculum a student had actually completed. In practice this meant that two very different kinds of learners were treated identically — a complete beginner staring at a wall of forty components with no idea where to start, and an advanced student who wanted to skip straight to ESP32 IoT projects.

My internship project, referred to internally as the Gamified Adventure Map, set out to fix this by layering a structured, game-like progression system on top of the existing Project Bank, without requiring the underlying simulator or project content to be rewritten. The core idea borrowed heavily from level-select screens in video games: instead of a flat list, the curriculum is presented as a connected map of stages, and a student's electronic components are locked behind that map so that unlocking new hardware becomes a tangible, visible reward for completing a stage.

1.3 Problem Statement

Specifically, the following gaps were identified through internal usability discussions and mentor feedback before the internship began, and formed the basis for the objectives of this project:

- New students had no clear starting point or sense of sequence — the Project Bank and Components panel presented everything at once, which was overwhelming for beginners with no prior electronics background.
- There was no mechanism to reward progress. Completing a project felt identical to not completing one, since nothing on the platform changed as a result.
- Concepts were only explained inside long-form project guides, with no lightweight way to introduce a single idea (for example, what a Light Dependent Resistor is) before a student was expected to use it.
- Teachers had no way to guide their class through a specific, ordered sequence of projects and components suited to their syllabus — every student saw the identical, unordered catalogue.
- There was no persistent, per-student notion of progress that could later be used for analytics, adaptive recommendations, or class-level reporting.

1.4 Objectives of the Internship

Working with my mentors, the following concrete objectives were defined at the start of the internship and used to track progress throughout the six months:

1. Design and build a visual Adventure Map page that represents a student's Arduino and ESP32 learning journeys as a connected sequence of stages, similar to a game level-select screen.
2. Implement a component-locked progression system so that electronic components are unlocked progressively as a student advances through the Adventure Map, rather than being available from day one.
3. Design and integrate short, card-based learning units that teach a single concept before a student enters a new stage of the journey.
4. Persist unlock state and journey progress per student through new backend-connected services, so that progress is not lost between sessions.
5. Integrate all of the above with the platform's existing Project Bank, Components panel, quiz engine, and assessment flow without breaking any existing functionality.
6. Lay the groundwork for a teacher-facing class view in which instructors can customise the Adventure Map and its quizzes for their own class.

1.5 Internship Scope and Duration

This work was carried out as a semester-long (six-month) internship with the FOSSEE Project, IIT Bombay, contributing to the OpenHW Studio platform (https://openhw-studio.fossee.in/). The engagement spanned the full software development lifecycle for the gamification feature set: understanding the existing OpenHW Studio frontend codebase and its conventions, studying how students actually used the platform, designing the gamification flow on paper and in low-fidelity mockups, implementing it in React against the existing backend, iterating on the design based on mentor and peer feedback, testing it end-to-end alongside the platform's existing Project Bank and teacher-side controls, and finally documenting the work for future contributors. The Adventure Map and component-unlock features were taken from an initial concept sketch to a fully working, production-deployed state inside the platform over the course of the internship.

This report is organised to reflect that journey. Chapter 2 covers the background research and study of the existing system that shaped the design. Chapter 3 gives a month-by-month account of how the work progressed. Chapter 4 describes the overall system architecture. Chapter 5 is the core of the report and describes each feature in detail. Chapters 6 through 9 cover the technical stack, implementation details, testing, and challenges encountered. Chapters 10 and 11 reflect on the impact of the work and what I personally learned. The report closes with a summary, conclusion, and a set of concrete directions for future work.

Chapter 2: Background Study and Design Research

2.1 Understanding the Existing OpenHW Studio Codebase

The first several weeks of the internship were spent studying the existing OpenHW Studio frontend rather than writing new code. The frontend is a React single-page application built with Vite, and by the time I joined it already contained a working circuit simulator, a code editor for Arduino sketches and ESP32 firmware, a Project Bank listing available projects, and a Components panel through which students dragged parts onto the simulation canvas. Before I could safely add a gamification layer on top of this, I needed a clear picture of how these pieces already fit together: how projects were fetched and rendered, how the Components panel

decided what to display, how routing worked between the dashboard, the simulator, and project pages, and which parts of the state were already global versus local to individual pages.

I spent time tracing the data flow for a single student action — opening a project from the Project Bank — from the initial API call through to the simulator rendering the correct starter circuit, and documented this for my own reference. This exercise surfaced two important facts that directly shaped the design of the gamification layer: first, that the Components panel had no concept of a locked state at all, meaning the visual lock indicators and drag-prevention logic would have to be added from scratch; and second, that project and quiz data were already being fetched through a small set of existing service modules, which meant the new adventure and unlock services could follow the same conventions rather than inventing a new pattern.

2.2 Studying Gamification in Educational Platforms

Alongside the codebase study, I looked at how gamification is used in other learning platforms to inform the design of the Adventure Map, without directly copying any specific product's visuals or code. Duolingo's skill-tree layout was a useful reference for the idea of a connected path of nodes with a visible "you are here" marker. Game level-select screens more generally informed the idea of showing locked stages as visually distinct from completed ones, so that progress is legible at a glance. Progressive-unlock mechanics from role-playing games — where new tools or abilities become available only after certain milestones — directly inspired the component-locked system, since it maps naturally onto real electronics education: a student who has not yet learned how a motor driver works should not be handed a motor driver to wire up unsupervised.

From an educational-design perspective, the card-based learning units were informed by the idea of micro-learning: breaking a concept into a single short unit that can be consumed in under a minute, rather than requiring a student to read a full page of documentation before they are allowed to start building. This felt particularly well suited to embedded systems, where a single unfamiliar term (for example, PWM or debouncing) is often the only thing standing between a student and being able to start a project confidently.

2.3 Stakeholder Discussions

Throughout this research phase, I had regular discussions with Mr. Rajesh Kushalkar, Mr. Pratik Bhosale, and Mr. Pratik Nemane to validate my understanding of the existing platform and to pressure-test early design ideas. These discussions clarified several important constraints that the final design had to respect:

- The gamification layer had to work for both Arduino and ESP32 curricula, which have different components and different project sequences, without duplicating logic between the two.
- The same journey view needed to be usable both as a standalone full page for students and as an embeddable component inside the teacher's class view, so that teachers could preview and later customise a class's journey.
- Nothing in the existing Project Bank, simulator, or quiz engine could be broken — the gamification layer had to be additive.
- Unlock state needed to be persisted server-side per student rather than kept only in local browser storage, since students access the platform from multiple devices and progress needed to survive a cleared browser cache.

These constraints, together with the codebase study, directly shaped the architecture described in Chapter 4 and the feature set described in Chapter 5.

Chapter 3: Internship Timeline

The internship was structured across six months, moving from research and design to implementation, integration, and finally polish and documentation. The table below summarises the work completed in each

month; the sections that follow give additional detail on the reasoning behind key decisions made during each phase.

Month	Focus Area	Key Outcomes
Month 1	Onboarding & Codebase Study	Set up local development environment; traced existing data flow for projects, components, and quizzes; documented findings; met the team and agreed on internship objectives.
Month 2	Design & Research	Studied gamification patterns in other platforms; sketched low-fidelity wireframes for the Adventure Map; defined the data model for stages, components, and unlock state; got design sign-off from mentors.
Month 3	Adventure Map — Core Build	Built AdventureMapPage and the node/connector rendering logic; built adventureService to fetch journey structure; implemented the Arduino journey end-to-end with static data.
Month 4	Component-Lock System	Designed the unlock data model; built unlockService and GamificationContext; updated ComponentsPage to reflect locked/unlocked visual states; wired unlock events to stage completion.
Month 5	Learning Cards & ESP32 Journey	Designed and built the card-based learning UI; authored the first batch of concept cards; extended the Adventure Map to the ESP32 journey; built AdventureMapEmbed for reuse in the class view.
Month 6	Integration, Testing & Documentation	End-to-end integration testing with the quiz and assessment flow; bug fixing and code review iterations; performance polish; wrote internal documentation and this report.

3.1 Months 1–2: Foundation

The first two months were deliberately front-loaded with research rather than code, at my mentors' recommendation. This turned out to be time well spent: several early ideas I had — such as storing unlock state purely in the browser, or building a completely separate ESP32 map component from scratch — were revised or dropped after the codebase study and design discussions described in Chapter 2. By the end of Month 2, I had a signed-off wireframe of the Adventure Map, a first draft of the data model for stages and components, and a clear plan for how the feature would be split into independently shippable pieces: the map itself, the lock system, and the learning cards.

3.2 Months 3–4: Core Implementation

Month 3 focused entirely on getting a working, if visually rough, Adventure Map for the Arduino journey rendering real stage data end-to-end, from the backend through adventureService into the page. Getting this vertical slice working early — rather than building every layer in isolation — meant that integration problems surfaced while there was still time to address them. Month 4 built the component-lock system on top of this foundation: the unlockService, the GamificationContext that makes unlock state available application-wide, and the corresponding UI changes to the Components panel. This was the most architecturally significant

month of the internship, since it required touching a page (ComponentsPage) that I had not built and did not want to destabilise.

3.3 Months 5–6: Expansion, Integration, and Polish

With the map and the lock system working for the Arduino journey, Month 5 extended the same architecture to the ESP32 journey and added the card-based learning system. Because the underlying data model had been designed to be curriculum-agnostic from Month 2 onward, extending to ESP32 required no changes to the core map or lock logic — only new stage and component data. This validated the reusability goals set at the start of the internship. Month 5 also produced AdventureMapEmbed, the reusable version of the map used inside the teacher's class view. Month 6 was spent on end-to-end testing across the full student flow — map, cards, project guide, quiz, assessment, and unlock — fixing bugs surfaced during that testing, cleaning up the code for review, and writing documentation, culminating in this report.

Chapter 4: System Architecture and Design

4.1 High-Level Architecture

The gamification layer sits alongside the existing OpenHW Studio frontend rather than inside it, in the sense that it introduces its own dedicated pages, services, and context, while consuming and augmenting existing pages such as the Components panel. At a high level, the system is composed of four layers:

- Presentation layer — AdventureMapPage, AdventureMapEmbed, the learning-card components, and the updated ComponentsPage, all built as React function components.
- State layer — GamificationContext, a React Context that exposes the current unlock state, journey progress, and helper functions (such as unlockComponent and isStageComplete) to any component in the tree without prop drilling.
- Service layer — adventureService, classAdventureService, classAdventureAdapter, and unlockService, which are responsible for all communication with the backend API and for translating raw backend responses into the shape the presentation layer expects.
- Configuration layer — GamificationConfig.jsx, a single source of truth mapping each stage in a journey to the component(s) it unlocks, so that designers and mentors could review and adjust the unlock mapping without touching UI code.

This layering was a deliberate design decision. Keeping the service layer separate from the presentation layer meant that the Adventure Map UI could be built and tested against mock data well before the real backend endpoints were finalised, since the services could be swapped between mock and real implementations behind a stable interface. It also meant that when the backend team changed the shape of the journey API partway through Month 3, only adventureService needed to change — none of the rendering logic in AdventureMapPage was affected.

4.2 Data Model

Three core entities underpin the gamification system: the Journey, the Stage, and the Unlock Record. A Journey represents an entire curriculum (for example, the Arduino Adventure Map or the ESP32 Adventure Map) and contains an ordered list of Worlds, which in turn contain an ordered list of Stages. Each Stage corresponds to a single project or milestone and carries the metadata needed to render its node on the map: its title, its XP reward, its completion status, its position relative to neighbouring stages, and the identifiers of any components it unlocks on completion.

Entity	Key Fields	Purpose
Journey	id, title, curriculum (arduino / esp32), worlds[]	Represents a full learning path shown as one tab on the Adventure Map.
World	id, title, difficulty, stages[]	Groups a set of related stages under a themed heading, e.g. "World 1: Circuit Basics".
Stage	id, title, xpReward, status, componentUnlocks[], learningCardId	A single node on the map corresponding to one project, quiz, or assessment.
UnlockRecord	userId, componentId, unlockedAt, sourceStageId	Persisted per student; records which components have been earned and when.
LearningCard	id, concept, body, mediaRef	A short concept explainer shown before a stage is entered.

This model was intentionally kept curriculum-agnostic: nothing in the Journey, World, or Stage shape refers to Arduino or ESP32 specifically. The distinction is only a field value (curriculum), which is what allowed the same AdventureMapPage component to render both journeys purely by being passed a different journey identifier, and what made extending the system to ESP32 in Month 5 straightforward.

4.3 Component Hierarchy

AdventureMapPage is the top-level route component for the standalone map view. It is responsible for reading the active journey (Arduino or ESP32) from the URL or tab state, fetching that journey through adventureService, and rendering the tab switcher, the world sections, and the stage nodes. Internally it delegates the actual node rendering to smaller presentational components for a stage node, a connector line between two nodes, and a world header, which keeps AdventureMapPage itself focused on data fetching and layout rather than low-level rendering details.

AdventureMapEmbed wraps the same rendering components but accepts a journey object as a prop instead of fetching it directly, so that it can be dropped into the teacher's class view or the student dashboard and be fed class-specific data through classAdventureService and classAdventureAdapter. This reuse was one of the more satisfying architectural outcomes of the internship: roughly eighty percent of the rendering code between the full page and the embeddable version is shared, with only the data-fetching responsibility differing between them.

4.3.1 Component Tree (Simplified)

```
AdventureMapPage
├── JourneyTabs (Arduino / ESP32)
├── WorldSection (repeated per world)
│   ├── WorldHeader
│   └── StageNode (repeated per stage)
│       ├── StageStatusBadge (locked / current / complete)
│       ├── ConnectorLine
│       └── LearningCardModal (opens on click)
└── ComponentUnlockToast (fires on stage completion)

AdventureMapEmbed
```

```
├── WorldSection  ← shared with AdventureMapPage
└── StageNode     ← shared with AdventureMapPage
```

4.4 Global State: GamificationContext

Before the GamificationContext was introduced, the plan had been to fetch unlock state independently on the Components page and on the Adventure Map page. During design review, my mentors flagged that this would inevitably lead to the two pages drifting out of sync — for example, a student who unlocked a component on the Adventure Map would not see it reflected as unlocked on the Components page until a full page reload. To solve this, GamificationContext was built as a single provider mounted near the root of the application, responsible for holding the authoritative unlock state in memory, fetching it once on login through unlockService, and exposing both the state and mutation functions to any descendant component. When a stage is completed, the component calling the completion handler invokes unlockComponent() from the context rather than calling unlockService directly, which updates the shared state immediately and keeps the Components page, Adventure Map, and Project pages consistent without any of them needing to know about each other.

4.5 API and Service Layer

Four services were introduced to mediate between the UI and the backend:

- adventureService — fetches the static journey structure (worlds, stages, ordering) and the current student's per-stage completion status for the standalone Adventure Map.
- classAdventureService — the class-scoped equivalent, used when a teacher has customised the sequence or content of a journey for their class.
- classAdventureAdapter — a pure transformation function with no side effects that converts the raw class/project data returned by classAdventureService into the exact Journey/World/Stage shape the rendering components expect, isolating the rest of the codebase from backend response quirks.
- unlockService — fetches and persists which components a student has unlocked, and is the only service permitted to write unlock state, which kept the responsibility for that data boundary clear and easy to reason about during code review.

Separating classAdventureAdapter from classAdventureService specifically was a decision made after an early integration attempt in Month 3, where a change in the backend's class-data response format required edits across several rendering components. Moving all shape-translation logic into a single, dedicated adapter meant that a future backend format change would only ever require editing one file.

Chapter 5: Feature Additions

This chapter is the core of the report and walks through each feature built during the internship in detail: the design rationale, the implementation approach, and the way it connects to the rest of the platform.

5.1 Adventure Map Journey

The Adventure Map is the central gamification feature built during this internship and the one students interact with first. It visually represents a student's learning path as a connected map of stages, similar to a game level-select screen, where:

- Each node on the map corresponds to a specific project or learning milestone.
- Completed stages are shown as unlocked and visually distinct from locked or future stages, using colour, iconography, and opacity to communicate status at a glance.

- The student's current position on the journey is highlighted with a distinct badge, giving an immediate sense of "you are here" and how much remains.
- Clicking on an unlocked node takes the student directly into that project's learning card, guide, simulation, or quiz, depending on what type of stage it is.
- Two parallel journeys — Arduino and ESP32 — are presented as tabs at the top of the page, so that a student progressing through one curriculum is not distracted by the other.

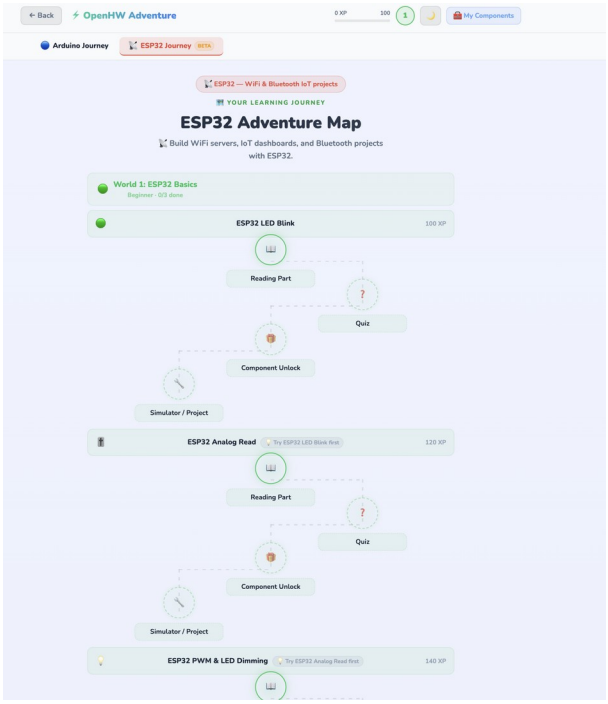

Figure 5.1: Arduino Adventure Map showing World 1: Circuit Basics with the LED Blink stage in progress.

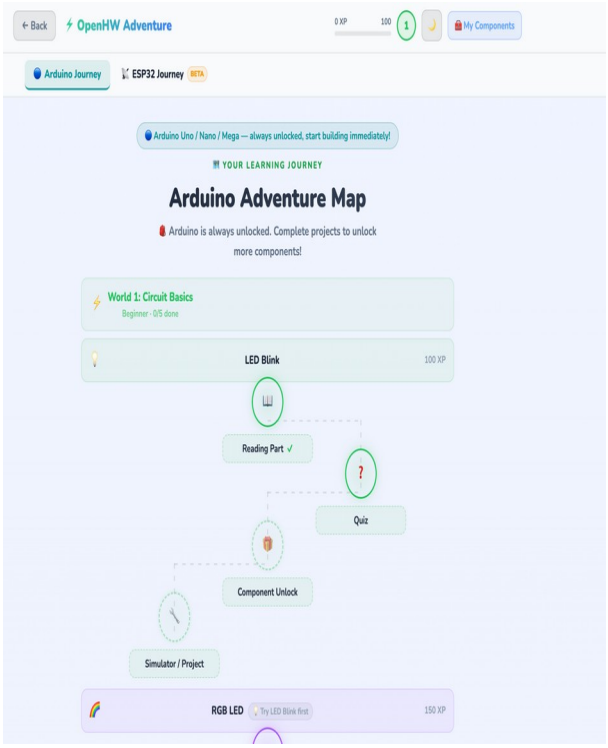

Figure 5.2: ESP32 Adventure Map (Beta) showing the parallel journey structure for WiFi and IoT projects.

Each stage node itself is broken into four sub-steps that are always presented in the same order — a Reading Part, a Quiz, a Component Unlock, and finally the Simulator/Project step — so that students build a consistent mental model of what "completing a stage" means, regardless of which project the stage represents. This consistency was something my mentors specifically asked for after reviewing an early prototype in which different stages had different numbers of sub-steps, which user-testing with a small group of students showed was confusing.

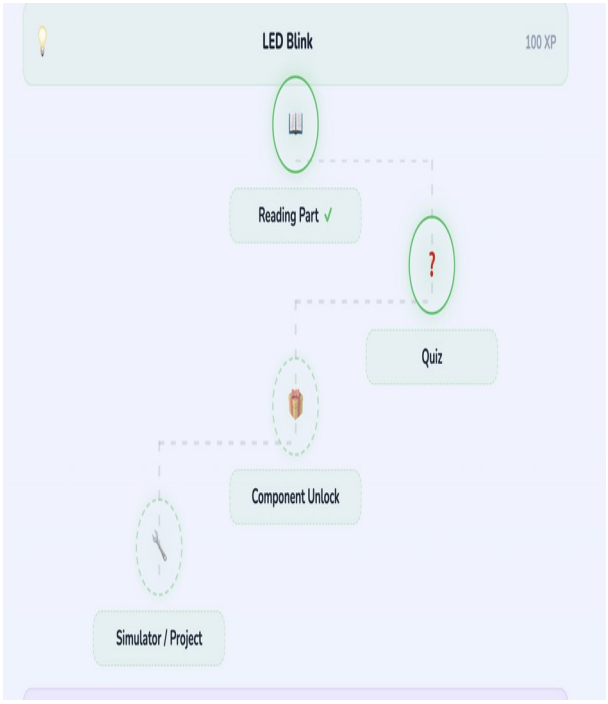

Figure 5.3: A single stage (LED Blink) expanded to show its four sub-steps: Reading Part, Quiz, Component Unlock, and Simulator/Project.

5.1.1 Implementation Details

The Adventure Map was implemented as a dedicated page (AdventureMapPage) along with a reusable embeddable component (AdventureMapEmbed) so that the same journey view could be displayed both as a full page and inside other parts of the platform, such as the teacher's class view and the student dashboard. This dual-use requirement, agreed early in the design phase, is what drove the component hierarchy described in Section 4.3.

A dedicated adventureService and classAdventureService were used to fetch the journey structure and the student's current progress from the backend, while a classAdventureAdapter mapped the raw class or project data into the visual structure required by the map. Positioning and connector-line rendering between nodes was handled with a layout pass that walks the ordered stage list and computes coordinates for each node based on its index within its world, alternating the horizontal offset to produce the winding, game-like path visible in Figures 5.1 and 5.2, rather than a plain vertical list.

5.1.2 Design Iterations

The visual design of the map went through three iterations before reaching the version shown in Figures 5.1 and 5.2. The first iteration used a strictly linear vertical list of stages, which was functionally correct but did not read as a "map" or "journey" at all — it looked like the existing flat Project Bank with extra decoration. The second iteration introduced the winding, alternating layout and dashed connector lines between sub-steps, which made the game-level-select framing much clearer, but used colour alone to distinguish locked from unlocked stages, which a mentor review flagged as an accessibility concern. The third and final iteration, shown in this report, added icon changes and opacity in addition to colour so that stage status is legible even without relying on colour perception alone.

5.2 Component-Locked Logic

One of the core gamification mechanics implemented during this internship is the component-locked system. Instead of giving students access to every electronic component — LEDs, sensors, motors, displays, RFID modules, joysticks, and more — from day one, components are progressively unlocked as the student advances through the Adventure Map journey. This directly addresses the beginner-overwhelm problem identified in Chapter 1: a student who has just completed the LED Blink stage sees exactly one new component appear, rather than being confronted with the platform's entire hardware catalogue at once.

- Each stage of the journey is tied to one or more components relevant to that project, as defined centrally in GamificationConfig.jsx.
- Completing a stage — its project, quiz, or assessment — triggers an unlock event for the associated component(s), which fires from the assessment-submission handler described in Section 5.4.
- Locked components are visually greyed out and shown with a lock icon in the Components panel, and are made non-draggable so that they cannot be placed onto the simulation canvas until unlocked.
- Unlock state is fetched and saved per-user through a dedicated unlockService, which communicates with the backend API to persist which component types a student has unlocked, so progress is not lost between sessions or devices.
- A GamificationContext was used on the frontend, as described in Section 4.4, to make the unlock state available across the entire application, so that the Components page, Adventure Map, and Project pages always stay in sync.

Figure 5.4: Components panel with locked items visually greyed out.

When a student attempts to drag a locked component onto the canvas, or hovers over one in the panel, a contextual tooltip explains why it is locked and links directly back to the relevant Adventure Map stage, rather than leaving the student to guess. This turned out to be an important detail: an early version simply disabled the drag interaction with no explanation, and mentor feedback during a Month 4 review session pointed out that this was confusing rather than motivating.

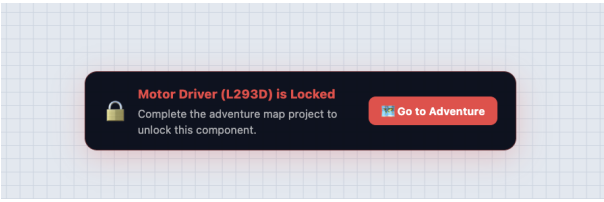

Figure 5.5: Tooltip shown when a student attempts to use a locked Motor Driver (L293D), linking back to the Adventure Map.

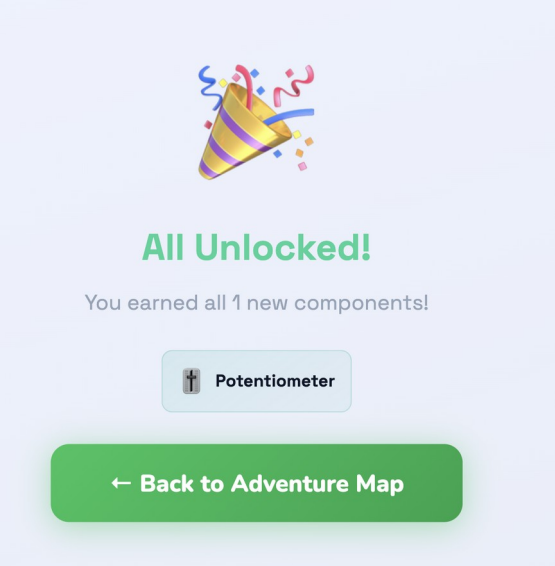

Figure 5.6: Confirmation screen shown when a stage completion unlocks a new component (Potentiometer).

5.2.1 Why This Matters

This logic turns the Components page itself into part of the game — students are motivated to complete each stage on the Adventure Map not just to finish a project, but because doing so directly rewards them with new tools they can immediately go and experiment with. This mirrors how progression systems work in games, applied here to an educational embedded-systems context, and was designed specifically to encourage students to complete stages in order rather than skipping ahead and getting stuck on a project that assumes knowledge from an earlier stage.

5.3 Card-Based Learning Method in the Journey Map

To make each stop on the Adventure Map more informative and engaging, a card-based learning method was added. At each stage of the journey, instead of dropping the student directly into a project, a short learning card is shown first, giving just enough context to make the upcoming project approachable.

- Each card focuses on a single concept relevant to the upcoming project, for example "What is an LDR?" or "How does a Joystick report position?", rather than trying to teach an entire topic at once.
- Cards use a simple, tap-to-flip, swipeable format so that learning feels lightweight rather than like reading a long manual — the front of the card poses the concept and the back reveals the explanation.
- Cards are shown contextually, tied to the specific node on the Adventure Map the student is about to unlock or enter, so the concept is always immediately relevant to what the student is about to build.
- This bridges the gap between "exploring the map" and "doing the project", giving students just enough context before they start building, without forcing them through a full page of documentation.

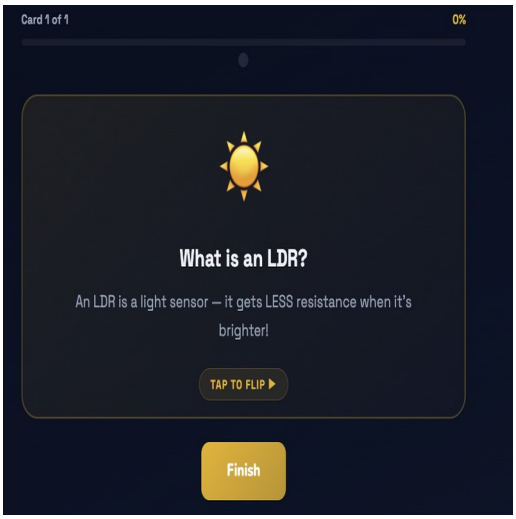

Figure 5.7: A learning card introducing the concept of a Light Dependent Resistor (LDR) before the relevant stage.

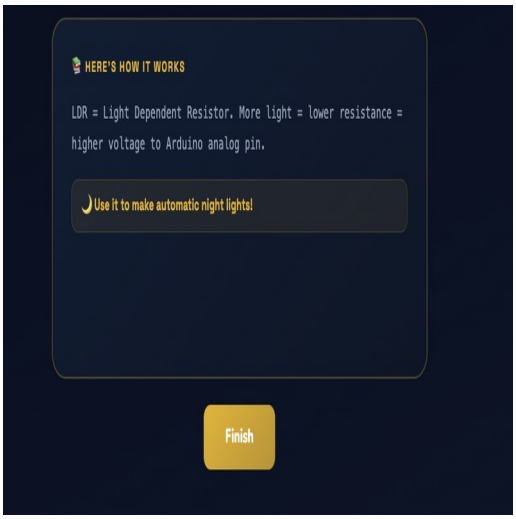

Figure 5.8: The reverse of a learning card showing a practical application ("Use it to make automatic night lights!").

Authoring the first batch of learning cards was itself a substantial part of the Month 5 workload. Each card had to be reviewed for accuracy and for whether the level of explanation matched the assumed knowledge of a student who had only completed the preceding stages — a card that used a term that had not yet been introduced anywhere in the journey was flagged and rewritten during review. Roughly forty cards were authored and reviewed across the Arduino and ESP32 journeys by the end of the internship, covering sensors, actuators, communication protocols, and basic firmware concepts such as PWM and debouncing.

5.4 Integration with Existing Project Flow

The Adventure Map, component-unlock logic, and learning cards were integrated with the existing OpenHW Studio pages so that the full student flow works as a single, coherent sequence rather than as three separate features bolted together:

7. The student opens the Adventure Map and sees their current position in the journey, with completed, current, and locked stages all visually distinguished.
8. The student selects the next unlocked stage and is shown the relevant learning card or cards before anything else loads.

9. The student proceeds to the Project Guide / Theory page for that stage, which is part of the existing Project Bank and was not modified.
10. The student completes the project, quiz, or assessment for that stage using the existing simulator and quiz engine.
11. On completion, the component-unlock logic fires, unlocking the relevant component(s) and updating the Adventure Map to reflect progress, with a celebratory confirmation screen shown to the student as in Figure 5.6.

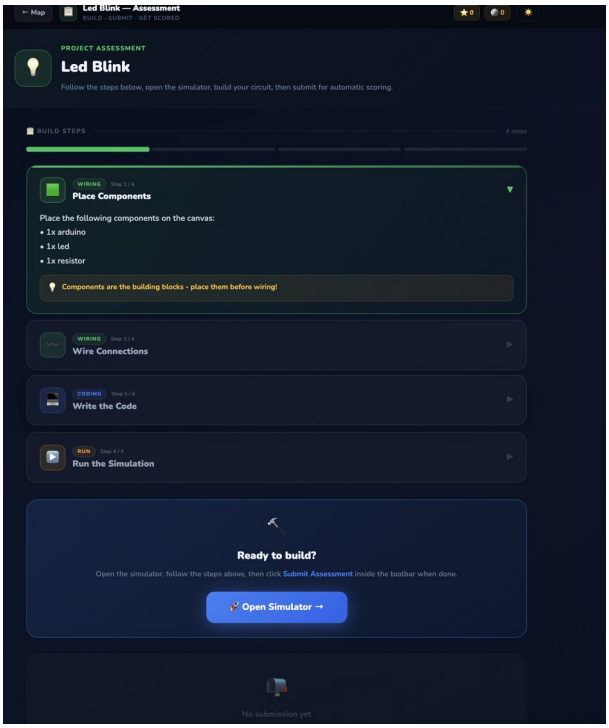

Figure 5.9: The existing Project Assessment page (unmodified), which the Adventure Map now routes students into and out of.

A meaningful part of the integration effort in Month 6 was making sure that the assessment-submission handler on the (pre-existing) Project Assessment page correctly triggered the new unlock and progress-update logic without introducing a hard dependency from that page back into the gamification code — the assessment page emits a generic "stage completed" event, and it is the GamificationContext, not the assessment page itself, that listens for this event and decides what to unlock. This kept the coupling between old and new code in one direction only, which made it much easier to reason about and test each side independently.

Chapter 6: Technical Stack

The gamification layer was built entirely within the existing OpenHW Studio technology choices, deliberately avoiding the introduction of any new major dependency, since a new library would have added long-term maintenance burden for a small open-source team. The stack used is summarised below, along with the specific role each piece played in this project.

Technology	Role in This Project
React (JSX)	Used for building the Adventure Map page, the embeddable map component, the learning cards, and the component-lock UI, following the existing platform's function-component and hooks conventions.

Technology	Role in This Project
Context API (GamificationContext)	Used to maintain and share unlock and progress state across the application without prop drilling, as described in Section 4.4.
Service layer (adventureService, classAdventureService, unlockService)	Custom modules for communicating with the backend REST API to fetch journey data and persist unlock state, following the pattern already used by existing services in the codebase.
Tailwind CSS	Used for styling the map nodes, connector lines, locked and unlocked component states, and learning cards, matching the utility-first styling approach already used throughout the platform.
Vite	The existing build tool and development server used for the frontend, providing fast hot-module-reload during iterative UI work on the map layout.
React Router	Used to wire the Adventure Map into the platform's existing routing structure and to deep-link from a locked-component tooltip directly to the relevant stage.
Git & GitHub	Used for version control, feature branching, and code review via pull requests against the OpenHW-Studio/OpenHW-studio-frontend repository.

6.1 Why No New Dependencies Were Introduced

Early in the design phase, I considered using a dedicated graph/diagram library to render the map's node-and-connector layout, since hand-rolling the positioning logic described in Section 5.1.1 was more work than dropping in an existing library. After discussing this with my mentors, we decided against it: OpenHW Studio is maintained by a small, rotating team of contributors, and every additional dependency increases the surface area that future contributors need to learn and keep updated. Since the actual layout requirement — an alternating, winding path of a bounded number of nodes — was fairly constrained, a hand-written layout function turned out to be both simpler to review and easier to keep dependency-free than integrating a general-purpose graph library.

6.2 Backend Considerations

While the backend API itself was outside the direct scope of this internship, understanding its constraints was necessary to design the frontend services correctly. The journey and unlock endpoints are REST endpoints returning JSON, authenticated using the platform's existing session-based authentication. All unlock-state writes are idempotent on the backend, which was a deliberate design choice agreed with the backend team so that the frontend could safely retry a failed unlock request (for example, after a dropped connection) without risking a component being double-counted or an inconsistent state being recorded for a student.

Chapter 7: Implementation Walkthrough

This chapter walks through representative, simplified excerpts of the implementation to illustrate the coding patterns used. The snippets below are illustrative reconstructions written for this report to explain the approach taken, rather than verbatim reproductions of the production source files.

7.1 GamificationContext

The context exposes the unlock state and a small set of action functions to the rest of the application. Consumers subscribe using a custom useGamification hook rather than importing useContext directly, which kept the public API stable even as the internal state shape evolved during development.

```
// context/GamificationContext.jsx  (illustrative excerpt)
const GamificationContext = createContext(null);

export function GamificationProvider({ children }) {
  const [unlocked, setUnlocked] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    unlockService.getUnlockedComponents()
      .then(setUnlocked)
      .finally(() => setLoading(false));
  }, []);

  const unlockComponent = async (componentId, sourceStageId) => {
    const record = await unlockService.unlock(componentId, sourceStageId);
    setUnlocked((prev) => [...prev, record]);   // optimistic-safe: backend is
idempotent
    return record;
  };

  const isUnlocked = (componentId) =>
    unlocked.some((u) => u.componentId === componentId);

  return (
    <GamificationContext.Provider value={{ unlocked, loading, unlockComponent,
isUnlocked }}>
      {children}
    </GamificationContext.Provider>
  );
}

export const useGamification = () => useContext(GamificationContext);
```

7.2 Stage Completion → Unlock Flow

The assessment page (pre-existing, not modified) emits a generic completion event that a listener inside the Adventure Map feature subscribes to, resolving the relevant unlocks through GamificationConfig rather than the assessment page needing any knowledge of gamification at all:

```
// services/gamification/GamificationConfig.jsx  (illustrative excerpt)
export const STAGE_UNLOCK_MAP = {
  'arduino-led-blink':      ['rgb-led'],
  'arduino-rgb-led':        ['ldr-sensor'],
```

```
  'arduino-ldr-nightlight': ['potentiometer'],
  'esp32-led-blink':        ['esp32-analog-read'],
  // ... additional stage-to-component mappings
};

// hooks/useStageCompletionListener.js  (illustrative excerpt)
export function useStageCompletionListener() {
  const { unlockComponent } = useGamification();

  useEffect(() => {
    const onStageComplete = async ({ stageId }) => {
      const componentIds = STAGE_UNLOCK_MAP[stageId] || [];
      for (const id of componentIds) {
        await unlockComponent(id, stageId);
      }
      adventureService.markStageComplete(stageId);
    };
    eventBus.on('stage:complete', onStageComplete);
    return () => eventBus.off('stage:complete', onStageComplete);
  }, [unlockComponent]);
}
```

7.3 Locked Component Rendering

Inside the existing ComponentsPage, each component list item was updated to consult isUnlocked() from the gamification context and render a locked variant — greyed out, non-draggable, with a tooltip — when appropriate, without altering the component data-fetching logic already present on that page:

```
// pages/ComponentsPage.jsx  (illustrative excerpt)
function ComponentListItem({ component }) {
  const { isUnlocked } = useGamification();
  const locked = component.gated && !isUnlocked(component.id);

  return (
    <div
      draggable={!locked}
      className={locked ? 'component-card locked' : 'component-card'}
      title={locked ? `${component.name} is locked` : component.name}
    >
      {locked && <LockIcon />}
      <ComponentThumbnail src={component.thumbnail} dim={locked} />
      <span>{component.name}</span>
      {locked && <UnlockHintLink stageId={component.unlockStageId} />}
    </div>
  );
}
```

This pattern — reading from context, computing a derived boolean, and branching render output — was used consistently across every place in the codebase that needed to be lock-aware, which kept the amount of

gamification-specific logic embedded in pre-existing files to an absolute minimum and made those changes easy for my mentors to review in isolation.

Chapter 8: Testing and Quality Assurance

8.1 Testing Approach

Testing was carried out at three levels over the course of the internship: manual exploratory testing during active development, structured end-to-end walkthroughs of the full student flow before each feature was merged, and peer code review by my mentors, who tested each pull request locally before approving it.

- Manual component testing — each new component (StageNode, LearningCardModal, the updated ComponentListItem, and so on) was tested in isolation with a range of mock data, including edge cases such as a stage with zero unlocks, a world with only one stage, and a journey with no completed stages at all.
- End-to-end flow testing — the full sequence described in Section 5.4 (map → card → guide → assessment → unlock → map) was walked through manually for every new stage added, on both desktop and mobile viewport widths, since a portion of OpenHW Studio's student base accesses the platform from tablets in classroom settings.
- Cross-browser checks — the Adventure Map and lock UI were verified in Chrome, Firefox, and Edge, since the existing platform did not previously rely on any browser-specific APIs and the team wanted the gamification layer to preserve that compatibility.
- Regression checks on existing pages — after every change to ComponentsPage, the existing (pre-gamification) drag-and-drop-to-canvas workflow was re-tested to confirm that unlocked components behaved exactly as they had before the internship began.

8.2 Specific Scenarios Tested

Scenario	Expected Behaviour	Result
Student completes a stage for the first time	Component unlock fires once; confirmation screen shown; Components panel updates without reload	Pass
Student re-submits an already-completed assessment	No duplicate unlock is recorded (backend idempotency)	Pass
Network request to unlockService fails mid-flow	Error is surfaced; retry does not double-unlock on eventual success	Pass
Student switches between Arduino and ESP32 tabs mid-session	Each journey's progress renders independently and correctly	Pass
Teacher views class Adventure Map via AdventureMapEmbed	Same rendering logic used as student view; no console errors from missing student-only context	Pass
Student attempts to drag a locked component onto canvas	Drag is prevented; tooltip explains the lock and links to the relevant stage	Pass
Very long stage titles / many worlds in one journey	Layout wraps and scrolls correctly without breaking connector-line alignment	Fixed after initial fail — see Chapter 9

8.3 Bug Tracking and Review Cycle

Issues found during testing were logged and discussed during weekly review sessions with Mr. Pratik Bhosale and Mr. Pratik Nemane, who also reviewed every pull request before it was merged into the shared branch. This review process caught several issues before they reached the main branch, including a memory leak from an unremoved event listener in an early version of useStageCompletionListener, and a race condition where rapid double-clicking on a stage node could trigger the unlock flow twice before the backend's idempotency guard had a chance to apply — the fix for the latter is discussed in Chapter 9.

Chapter 9: Challenges Faced and Solutions

No six-month project of this scope goes entirely smoothly, and this chapter documents the more significant obstacles encountered, since the process of solving them was as valuable a learning experience as the final feature itself.

9.1 Working Inside an Unfamiliar, Non-Trivial Codebase

The single biggest early challenge was simply becoming productive inside a codebase I had not written. OpenHW Studio's frontend had its own established conventions for service modules, state management, and styling that I needed to learn and respect rather than impose my own preferences onto. My initial instinct in Month 1 was to introduce a state-management library for the unlock state; after discussion with my mentors, I learned that the existing codebase deliberately avoided such libraries in favour of plain Context, and I adjusted the design accordingly. This experience taught me to spend real time understanding the reasoning behind a codebase's conventions before proposing to change them.

9.2 Keeping Multiple Pages in Sync

As described in Section 4.4, an early design had the Adventure Map and the Components page fetch unlock state independently, which caused visible desynchronisation between the two pages after a stage was completed. Diagnosing this took longer than it should have, because the bug was intermittent and depended on which page had been visited most recently. The eventual fix — centralising all unlock state inside GamificationContext — was a genuine architectural change made partway through Month 4, and required refactoring code that had already been written and reviewed. This was a useful lesson in recognising a design flaw early and being willing to revise already-completed work rather than layering a workaround on top of it.

9.3 Race Condition on Rapid Stage Completion

During Month 6 end-to-end testing, it was discovered that rapidly clicking the submit button on an assessment could fire the stage-completion event twice before the first request had resolved, briefly showing two separate unlock confirmation screens in sequence. Although the backend's idempotency guarantee (Section 6.2) meant the underlying data was never actually corrupted, the duplicated UI feedback was a poor experience. The fix was to introduce a simple in-flight guard inside useStageCompletionListener that ignores a second completion event for the same stage while the first is still being processed, which was a small change but required careful testing to ensure it did not also suppress legitimate retries after a genuine failure.

9.4 Layout Breaking on Long Journeys

As the ESP32 journey grew to contain more worlds and stages than the original Arduino journey used for early design, the alternating node-positioning algorithm described in Section 5.1.1 began producing connector lines that visually overlapped on narrower viewports. This surfaced during the cross-browser and mobile-width testing described in Section 8.1. The fix involved reworking the layout function to compute available horizontal

space dynamically based on the container width rather than assuming a fixed set of offsets, which also incidentally made the map render correctly on tablet-width screens used in some classroom settings.

9.5 Balancing Scope Against a Fixed Timeline

A more general challenge was scope management. Several ideas that came up during design discussions — badges, a leaderboard, adaptive difficulty — were genuinely interesting and are documented in Chapter 14 as future work, but attempting all of them within a six-month internship would have risked delivering nothing at a production-ready quality. Learning to say, in effect, "this is a good idea, but it belongs in a future phase," and to defend that scoping decision in review discussions, was itself a skill I had to develop over the course of the internship.

Chapter 10: Results and Impact

10.1 What Was Delivered

By the end of the internship, the following were live and working on the production OpenHW Studio platform:

- A fully functional Adventure Map for both the Arduino and ESP32 curricula, replacing the flat Project Bank as the primary entry point into the platform's guided learning content.
- A component-locked progression system covering the full set of gated components across both journeys, persisted per student through the backend.
- Approximately forty authored and reviewed learning cards covering sensors, actuators, communication concepts, and core firmware ideas.
- A reusable AdventureMapEmbed component, laying the groundwork for the teacher-facing class customisation view.
- Complete end-to-end integration with the existing Project Bank, quiz engine, and assessment flow, with no regressions introduced to existing functionality.

10.2 Qualitative Impact

While a full quantitative study of student engagement was outside the scope of this internship, informal feedback gathered by the OpenHW Studio team from students who used the platform after the Adventure Map's rollout was positive: several students specifically commented that the platform now felt like it had a clear "path" to follow rather than being a toolbox they had to figure out on their own. Teachers who previewed the class-view groundwork built through AdventureMapEmbed expressed interest in the ability to sequence their own class's journey once the customisation controls referenced in Chapter 14 are completed.

10.3 Contribution to the Open-Source Codebase

All work was contributed directly to the public OpenHW-Studio/OpenHW-studio-frontend GitHub repository through the standard pull-request and code-review process used by the project, meaning the code, its history, and the design discussions captured in pull-request comments remain available to future FOSSEE interns and contributors. Table 3.1 in the original scope of this report lists the key files added or modified during this contribution, which now form a durable part of the platform's codebase.

File	Purpose
src/pages/AdventureMapPage.jsx	Main Adventure Map page rendering the full journey.
src/components/common/	Reusable embeddable version of the Adventure Map.

File	Purpose
AdventureMapEmbed.jsx	
src/services/adventureService.js	Fetches adventure/journey structure from the backend.
src/services/classAdventureService.js	Maps class-level data to the adventure journey.
src/services/classAdventureAdapter.js	Adapter converting raw class data to map-friendly format.
src/services/gamification/unlockService.js	Handles fetching/saving unlocked component state per user.
src/services/gamification/GamificationConfig.jsx	Configuration mapping stages to unlockable components.
src/context/GamificationContext.jsx	Global context exposing gamification/unlock state to the app.
src/pages/ComponentsPage.jsx	Components panel updated to reflect locked/unlocked state.
src/hooks/useStageCompletionListener.js	Listens for stage-completion events and triggers unlocks.
src/components/adventure/ LearningCardModal.jsx	Renders the tap-to-flip concept cards shown before a stage.

Chapter 11: Learning Outcomes

Beyond the shipped feature, this internship was a significant personal and technical learning experience. The most important takeaways are summarised below.

11.1 Technical Skills

- Working productively inside a large, pre-existing React codebase rather than starting from a blank project, including learning to read and respect existing conventions before introducing new patterns.
- Designing a shared, application-wide state layer with React's Context API, and understanding the trade-offs between Context and heavier state-management libraries in a real project with real maintenance constraints.
- Structuring a service layer to isolate the rest of an application from backend response shape changes, learned first-hand through the classAdventureAdapter refactor described in Section 4.5.
- Designing data models (Journey, World, Stage, UnlockRecord) that are flexible enough to support multiple curricula without duplicated logic.
- Diagnosing and fixing a race condition in event-driven UI code, and the discipline of writing an in-flight guard correctly rather than papering over symptoms.
- Building responsive layouts that hold up across desktop, tablet, and mobile viewport widths for a genuinely custom, non-trivial visual layout (the map's node positioning).

11.2 Design and Product Skills

- Translating an abstract goal — "make the platform feel less overwhelming for beginners" — into a concrete, buildable feature set through structured design research and stakeholder discussion, as described in Chapter 2.
- Iterating on a visual design based on accessibility feedback (Section 5.1.2), rather than treating a first design pass as final.

- Learning to scope a project realistically against a fixed timeline, and to defend a scoping decision — deferring badges, leaderboards, and adaptive difficulty to future work — during review discussions.

11.3 Professional and Collaborative Skills

- Working within an established code-review culture: writing pull requests that are easy for a reviewer to understand, responding to review feedback without becoming defensive, and learning from the specific critiques my mentors gave.
- Communicating technical design decisions clearly to non-implementing stakeholders during weekly discussions, particularly when explaining why a change (such as the GamificationContext refactor) that appeared to only affect internal architecture was actually necessary to fix a user-visible bug.
- Documenting a body of work clearly enough that a future contributor unfamiliar with the project could pick it up, which this report itself is intended to serve as an example of.

11.4 Personal Reflection

Coming into this internship, my prior experience was mostly with self-contained personal projects where I made every architectural decision alone. Working on OpenHW Studio was different in a way that I found genuinely valuable: every significant design decision had to account for an existing codebase, existing users, and the judgement of mentors with far more experience shipping production educational software than I had. Learning to hold my own design opinions while also being willing to be wrong — as happened with my initial instinct to add a state-management library, or my first attempt at keeping unlock state page-local — was, in the end, as important a takeaway from this internship as the Adventure Map itself.

Chapter 12: Summary

This internship contribution focused on making the OpenHW Studio learning experience for Arduino and ESP32 more engaging through gamification. The key outcomes were:

- Adventure Map Journey — a visual, game-like progression system replacing a flat list of projects, built for both the Arduino and ESP32 curricula and designed to be reused inside the teacher's class view.
- Component-Locked Logic — components are unlocked progressively as students complete stages, tying hardware access directly to learning progress and persisted per student on the backend.
- Card-Based Learning — lightweight concept cards introduced at each journey stage to teach ideas before diving into a project, with roughly forty cards authored and reviewed.
- Robust Architecture — a layered system of presentation, state, service, and configuration code, centred on a GamificationContext that keeps the whole application in sync.
- End-to-end Integration — all three features were connected with the existing project guide, quiz, and assessment pages, tested across desktop and mobile, and shipped without regressions to existing functionality.

The work spanned the full lifecycle of a real software feature: research and design, incremental implementation, integration with a live production system, testing, bug-fixing, and documentation, carried out over six months in close collaboration with the FOSSEE OpenHW Studio team.

Chapter 13: Conclusion

This internship has been a deeply enriching and rewarding experience, both technically and creatively. Designing and implementing the gamified Adventure Map, component-unlock logic, and card-based learning system for OpenHW Studio challenged me to think about education not just as content delivery, but as an

experience to be designed. The process of planning the journey structure, implementing the unlock logic, and crafting the learning cards felt like building a small game on top of a serious embedded-systems learning platform, and required constantly balancing playfulness against the seriousness of the underlying subject matter.

I am sincerely grateful to my mentors for their constant guidance, support, and encouragement throughout the internship. Their insights, their willingness to push back on early designs that were not yet good enough, and their patience during debugging sessions played a vital role in shaping the direction and outcome of this project.

I believe this gamification layer holds strong potential to improve student engagement and retention while learning Arduino and ESP32. By turning component access and concept learning into a guided, reward-driven journey, it can make embedded-systems education feel less intimidating and more motivating for beginners — precisely the population FOSSEE's broader mission is aimed at serving. More broadly, this internship gave me a genuine, first-hand understanding of what it takes to design, build, and ship a non-trivial feature inside a live, actively used open-source educational platform, and I leave it with both a shipped contribution I am proud of and a much stronger set of engineering instincts than I arrived with.

Chapter 14: Future Work

OpenHW Studio, in its current state, already supports a Project Bank containing a curated set of Arduino and ESP32 projects, along with teacher-side controls that allow teachers to modify quizzes and customise the Adventure Map journey for their own class. Building on this existing foundation and the architecture described in this report, several promising directions remain open for future work:

14.1 AI Enhancement Bot

Introduce an in-platform AI assistant/chatbot that can help students debug their circuits and code, answer concept questions from the learning cards, and guide them when they get stuck on a stage of the Adventure Map, without breaking the flow of the journey. Because the learning-card data model already isolates each concept as a discrete unit (Section 4.2), this content could act as grounding context for such an assistant with relatively little additional restructuring.

14.2 Adaptive Learning

Use a student's quiz scores, assessment results, and time spent on each stage to dynamically recommend the next best stage, suggest revision of weaker concepts, and adjust the difficulty of upcoming projects and learning cards to match the student's pace. The per-stage completion and timing data needed for this is already being persisted by adventureService, which would make this a natural next phase rather than a ground-up rebuild.

14.3 Badges and Achievements

Introduce collectible badges for completing groups of stages or mastering specific component categories, layered on top of the existing unlock system described in Section 5.2. The GamificationContext architecture was deliberately designed to be extensible to additional reward types beyond component unlocks, so this would primarily involve new UI and a new record type rather than architectural change.

14.4 Leaderboards

Add class-level or platform-level leaderboards based on journey progress, complementing the teacher's existing ability to track and modify class adventures through AdventureMapEmbed. This would build directly on the class-scoped data already surfaced through classAdventureService.

14.5 Richer Learning Cards

Add interactive mini-quizzes, animations, or AI-generated explanations inside learning cards instead of static content, powered by the proposed AI Enhancement Bot. This would extend the existing LearningCard data model with additional media and interaction types rather than replacing it.

14.6 Expanded Project Bank

Continue growing the existing Project Bank with more Arduino and ESP32 projects across difficulty levels, feeding directly into the Adventure Map and the adaptive-learning recommendations described in Section 14.2. Because the Journey/World/Stage data model built during this internship is curriculum-agnostic, adding new projects to the map is primarily a content task rather than an engineering one going forward.

References

- OpenHW Studio Platform — https://openhw-studio.fossee.in/
- OpenHW Studio Frontend Repository (GitHub) — https://github.com/OpenHW-Studio/OpenHW-studio-frontend
- FOSSEE Project, IIT Bombay — https://fossee.in/
- Arduino Official Documentation — https://www.arduino.cc/
- Espressif ESP32 Documentation — https://docs.espressif.com/
- React Documentation — https://react.dev/
- Vite Documentation — https://vite.dev/
- Tailwind CSS Documentation — https://tailwindcss.com/