SUMMER INTERNSHIP REPORT

on

Frontend Development of Progress Analytics and Mobile Workspace of the Adaptive Coding Platform

Submitted by

Prem Palhade

B.Tech in Computer Science Engineering,
Vishwakarma Institute of Technology, Pune

Under the Guidance of

Dr. Prabhu Ramachandran

Principal Investigator, FOSSEE Project
Department of Aerospace Engineering,

Indian Institute of Technology Bombay

August 2026

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Prabhu Ramachandran for providing me with the opportunity to be a part of the FOSSEE Internship Program at IIT Bombay and for supporting the development of open-source educational technologies. This opportunity provided me with valuable exposure to real-world software development and the open-source ecosystem.

I would also like to express my heartfelt gratitude to my mentor, Dr. Kushal Shah, for his continuous guidance, valuable feedback, encouragement, and support throughout the internship. His guidance helped me understand project requirements, overcome technical challenges, and improve my contributions to the project.

I am also thankful to my fellow interns and team members for their cooperation, discussions, and support throughout the internship. Working with others in a collaborative environment provided valuable learning experiences and helped me develop a better understanding of professional software development practices. Coordinating with the parallel frontend intern on a shared design system was an important part of delivering a consistent student experience.

Finally, I extend my sincere gratitude to everyone associated with the FOSSEE Project at IIT Bombay for providing an environment that promotes open-source development, innovation, collaboration, and continuous learning. This internship has been a valuable experience that strengthened my technical skills, problem-solving abilities, and professional understanding, while motivating me to continue contributing to the open-source community.

Prem Palhade
Vishwakarma Institute of Technology, Pune

DECLARATION

This internship has been a highly rewarding learning experience that provided me with the opportunity to contribute to the development of the Adaptive Coding Platform under the FOSSEE Project at IIT Bombay. During the internship, I gained practical experience in frontend development, progress analytics design, mobile workspace implementation, React component architecture, unit testing, and collaborative open-source software development.

The experience provided valuable exposure to professional software development practices and real-world challenges involved in building student-facing screens for a web-based educational platform. The technical knowledge, problem-solving skills, and hands-on experience gained throughout the internship will serve as a strong foundation for my future academic projects and career objectives.

I would like to extend my sincere appreciation to the FOSSEE Team for their coordination, guidance, and encouragement throughout the internship. The collaborative environment and support provided during the program contributed significantly to my learning and enabled me to complete my assigned responsibilities effectively.

Prem Palhade
Vishwakarma Institute of Technology, Pune

INDEX

Chapter 1
Introduction

The Adaptive Coding Platform (Yaksh) is an open-source, web-based programming education and assessment platform developed under the FOSSEE (Free/Libre and Open Source Software for Education) project at the Indian Institute of Technology Bombay (IIT Bombay). It enables students, educators, and institutions to conduct programming courses, coding assignments, laboratory exercises, and coding assessments through a unified online environment. By providing an integrated coding workspace, automated evaluation, user management, and course administration features, the platform simplifies the process of teaching, learning, and assessing programming skills without requiring complex local software installations.

The platform provides a modern web interface where instructors can create courses, manage programming assignments, organize assessments, and monitor student progress, while students can attempt coding problems through an interactive programming workspace and submit solutions for automated evaluation. It is particularly suitable for educational institutions, programming laboratories, coding competitions, and remote learning environments where scalable and accessible programming education is essential.

A distinctive direction of the current development effort is adaptivity. Students do not only solve a static list of questions. The system is designed to respond to their performance and error patterns, so that the next question, the visible concept state, and the feedback they receive remain connected to what they have already demonstrated. Progress analytics and a usable mobile workspace are therefore not optional extras. They are the surfaces through which a student can see that path and continue working on it.

1.1 Project Overview

The Free/Libre and Open Source Software for Education (FOSSEE) project, initiated at the Indian Institute of Technology Bombay (IIT Bombay), is a nationally recognized initiative that promotes the adoption, development, and dissemination of Free and Open Source Software (FOSS) across educational institutions in India. The project aims to reduce dependence on proprietary software by providing open-source alternatives for education, engineering, scientific computing, and software development while encouraging collaborative learning and community-driven innovation.

Among the various initiatives under the FOSSEE project, the Adaptive Coding Platform (Yaksh) is an open-source learning and assessment platform developed to simplify programming education and coding-based evaluation. The platform enables instructors to create and manage programming courses, organize coding assessments, evaluate submissions automatically, and monitor learner progress through a web-based interface. Students can access coding assignments, write and test solutions within an integrated coding environment, and submit their programs for evaluation.

By combining course management, programming workspaces, automated assessment, and user administration into a single platform, Yaksh serves as an effective solution for academic institutions, programming laboratories, and self-paced programming education. The internship work described in this report sits on top of that foundation: it gives students a clear view of concept mastery and a complete way to practise on mobile devices.

1.2 Internship Background

During my internship under the FOSSEE project, I contributed to the frontend of the Adaptive Coding Platform, with primary responsibility for Progress Analytics, the mobile student workspace, and the unit-testing layer of the React application. The work began with the student-facing analytics screens and progressively expanded into a full mobile navigation model, a production-ready component handoff for backend integration, and an automated test suite covering core logic, user-interface behaviour, and API contracts.

I designed and implemented multiple screens from scratch, including the analytics dashboard, the concept-tracking system, Question Deep Dive, Error Analysis, and the four-tab mobile workspace. Visual consistency with the rest of the platform was maintained by reading and applying the shared CSS variables and theme tokens used by the parallel frontend intern. Throughout the internship I followed professional software development practices, including Git-based version control, structured commits, component isolation, mocked API testing, and iterative feature development within an open-source environment.

1.3 Problem Statement

The Adaptive Coding Platform already provided a programming workspace and backend services for questions, submissions, and evaluation. What the student still lacked was a coherent way to see learning progress, inspect earlier attempts, and continue solving questions on a phone. Without these surfaces, adaptivity remained largely invisible: the student could keep answering questions, but could not inspect mastery, error patterns, or the path from the current concept to the next.

The primary challenges addressed during the internship included:

- Building a Progress Analytics dashboard that presents attempts, accuracy, concept mastery, and streak in a form a student can actually use.
- Representing six Python concepts as a dependency chain with mastered, active, and locked states, and expanding each concept into a readable summary of what has been learned, what remains, and where errors concentrate.
- Giving students a Question Deep Dive view of every attempt, including submitted code, the error made, and the feedback returned for that attempt.
- Presenting error analysis across five error categories with a per-concept accuracy breakdown.
- Delivering a complete mobile workspace, including bottom navigation, a phone-sized editor experience, hints, and submission, rather than a scaled-down copy of the desktop layout.
- Adapting the existing desktop Workspace to smaller screens by switching to a three-tab Problem / Editor / Output layout.
- Packaging the analytics screen as a production-ready React component tree whose mock data could later be replaced by live API responses without rewriting the UI.
- Establishing a Vitest and React Testing Library suite so core logic, UI behaviour, and API integration could be checked automatically.

1.4 Objectives of the Internship

The primary objectives of the internship were:

- Develop the Progress Analytics and mobile layers of the Adaptive Coding Platform using React and modern web development practices.
- Design and implement reusable user-interface components that follow the platform design system.
- Build an interactive concept map covering six Python concepts, with expandable summaries and a path-to-mastery view.
- Implement Question Deep Dive and Error Analysis screens so students can review attempts and error categories in detail.
- Deliver a four-tab mobile application shell with Home, Workspace, Progress, and Profile.
- Make the desktop Workspace usable on phones through screen-size detection and a three-tab layout.
- Separate mock data from presentation so the backend team could connect FastAPI responses without changing component structure.

• Set up Vitest from scratch and write a passing unit-test suite for core logic, UI interactions, and API calls.

• Contribute to a large-scale open-source educational platform while following collaborative software engineering practices.

1.5 Scope of Work

The scope of this internship covered the student-facing analytics and mobile experience of the Adaptive Coding Platform, together with the tests required to keep that experience stable. The work included designing and implementing React screens, matching shared CSS tokens, integrating against documented REST endpoints, isolating mock data from UI logic, detecting viewport size for layout switching, and validating behaviour with Vitest and React Testing Library.

The internship also required a working understanding of the backend surface the frontend consumes. FastAPI endpoints, the PostgreSQL schema that backs progress and profile data, and the Dockerized backend environment were used as the contract against which screens and tests were written. The project required continuous collaboration with mentors and fellow developers while following version control and modern software development workflows to ensure maintainability and code quality.

1.6 Report Organization

This report documents the work carried out during the internship and is organized into the following chapters:

• **Chapter 2** presents the organizational background of the FOSSEE project and provides an overview of the Adaptive Coding Platform.

• **Chapter 3** describes the technologies, frameworks, development tools, and software engineering practices used throughout the internship.

• **Chapter 4** discusses the frontend architecture, Progress Analytics, mobile workspace, and component handoff.

• **Chapter 5** explains REST API integration, dynamic data flow, mock-data separation, and unit testing.

• **Chapter 6** details user-interface refinement, mobile-first responsive design, and validation of the student experience.

• **Chapter 7** presents the technical challenges encountered, solutions implemented, project outcomes, and future scope of development.

• **Chapter 8** summarizes the internship and the professional skills acquired during the work.

• **Chapter 9** concludes the report by presenting the internship learning outcomes and overall experience gained during the programme.

Chapter 2
ORGANIZATION AND PROJECT OVERVIEW

2.1 Introduction to FOSSEE

The Free/Libre and Open Source Software for Education (FOSSEE) project is a flagship initiative funded by the Ministry of Education, Government of India, and coordinated by the Indian Institute of Technology (IIT) Bombay. The primary objective of FOSSEE is to improve the quality of education by promoting the adoption and development of Free and Open Source Software (FOSS) across educational institutions in India. By encouraging the use of open-source technologies, the project aims to reduce dependence on proprietary software while making high-quality educational tools freely accessible to students, educators, and researchers.

FOSSEE contributes to education by developing open-source software, creating technical documentation, preparing learning resources, organizing workshops, and encouraging students to contribute to real-world open-source projects. These initiatives provide practical exposure to collaborative software development while strengthening the open-source ecosystem in engineering, science, and technology education.

The project encompasses multiple domains, including programming education, scientific computing, electronics, mathematics, computational engineering, and software development. Through collaboration between academic institutions, industry professionals, mentors, and student contributors, FOSSEE continues to promote innovation, knowledge sharing, and accessible education throughout the country.

2.2 Adaptive Coding Platform (Yaksh)

The Adaptive Coding Platform (Yaksh) is one of the educational software platforms developed under the FOSSEE project to support programming education and coding-based assessment. The platform provides a web-based environment that enables educators to create programming assignments, conduct coding examinations, evaluate submissions automatically, and monitor student performance through a centralized interface.

Unlike conventional programming laboratories that often require extensive software installation and configuration, Yaksh offers an integrated browser-based

environment where students can access programming assignments, write code, submit solutions, and receive automated evaluation. This simplifies the learning process while reducing administrative overhead for educational institutions.

The current line of work extends that environment with adaptivity. Questions are selected with student performance and error patterns in mind, and the student is expected to see that process rather than experience it as a black box. Progress analytics, concept maps, and a mobile workspace are the means by which that expectation is made concrete.

The primary objectives of the Adaptive Coding Platform include:

- Providing an integrated environment for programming education and assessment.
- Supporting automated evaluation of programming assignments.
- Simplifying course, assignment, and user management for instructors.
- Offering students an intuitive coding workspace for solving programming problems.
- Making learning progress visible through concept tracking, error analysis, and attempt history.
- Promoting open-source software adoption in academic institutions through an accessible web-based platform.

2.3 Introduction to the Adaptive Coding Platform

The Adaptive Coding Platform is a modern web application that combines course management, programming assessment, user administration, and automated code evaluation into a single integrated system. The platform follows a client-server architecture in which the frontend provides an interactive user experience while the backend manages authentication, data storage, assignment management, and code evaluation.

During this internship, the Progress Analytics and mobile layers of the platform were developed using modern web technologies. These interfaces communicate with backend services through REST APIs to retrieve progress overviews, concept state, the next question, submission results, and profile information.

Some of the major features of the platform include:

- **Role-Based User Management:** Separate interfaces and permissions for administrators, instructors, and students.
- **Programming Workspace:** An integrated environment, including a Monaco-based editor, where students attempt programming assignments and submit solutions.

- **Progress Analytics:** A student dashboard for attempts, accuracy, concept mastery, streak, error categories, and question-level review.
- **Adaptive Question Flow:** Selection of subsequent questions informed by performance and error patterns rather than a fixed linear list.
- **Mobile Workspace:** A phone-sized interface with Home, Workspace, Progress, and Profile, so students can continue a session away from a desktop.
- **Automated Evaluation:** Backend integration for evaluating programming submissions and returning feedback to the student interface.

The runtime path of a student session is a single request–response loop. The student opens the React application in a browser. The client routes them to Home, Workspace, Progress, or Profile, then calls FastAPI over HTTPS with JSON. The sequencing service returns the next question. Submission goes back on the same channel and is evaluated in a sandbox. Progress and analytics services then expose the records that the concept map, Question Deep Dive, Error Analysis, and profile screens consume. PostgreSQL holds users, questions, submissions, concept state, and sessions. Nothing in the student interface talks to the database directly; the contract is the REST surface, which is why the frontend could be built and tested against mocks before live payloads were wired.

That split is what made the internship work possible as a frontend assignment. The Progress Analytics tree and the mobile shell only need five calls to cover the student loop: getNextQuestion, submitAnswer, getProgressOverview, getConceptProgress, and getProfile. The rest of the platform — authentication, question sequencing, and evaluation — remains behind those endpoints. The screens described in later chapters are therefore clients of a sealed contract, not a second copy of the backend.

2.4 Chapter Summary

This chapter presented an overview of the FOSSEE (Free/Libre and Open Source Software for Education) project and its contribution to promoting open-source software in education. It also introduced the Adaptive Coding Platform (Yaksh) as a web-based programming education and assessment system developed under the FOSSEE initiative. Finally, the chapter described the platform's objectives, architecture, and major features, providing the foundation for understanding the technical work carried out during the internship.

Chapter 3
TECHNOLOGIES AND FRAMEWORKS USED

The development of the Progress Analytics and mobile layers of the Adaptive Coding Platform required a modern web technology stack capable of building responsive user interfaces, hosting an in-browser code editor, integrating with backend services, and supporting automated tests. During the internship, the frontend work was carried out using contemporary web technologies and open-source development tools that supported modular development, API integration, and collaborative software engineering.

The selected technology stack enabled reusable component architecture, responsive layouts, a production editor experience on both desktop and mobile, and a testable boundary between the user interface and FastAPI services. Version control and collaborative workflows further ensured that development could be carried out efficiently within the FOSSEE open-source environment.

3.1 Frontend Technology Stack

• HTML5
HTML5 was used to provide the structural foundation of the application. Semantic HTML elements were utilized to improve readability, maintainability, and accessibility while creating responsive web pages for analytics, workspace, and profile modules.

• CSS3
CSS3 was used for the appearance of the application, including layouts, typography, spacing, and responsive behaviour. Custom properties and theme tokens from the shared design system were applied so that Progress Analytics and the mobile shell matched the visual language already established on the rest of the platform. Flexbox and related layout techniques were used for the concept map, stat cards, bottom navigation, and expandable panels.

• JavaScript (ES6+)
JavaScript served as the primary programming language for client-side functionality. Modern ES6 features including modules, destructuring, and asynchronous programming using Promises and async/await were used throughout the analytics screens, mobile workspace, and test helpers.

• React.js with Vite
React.js was used as the primary frontend library for building the Progress

Analytics and mobile interfaces. The application was developed using reusable components, allowing the analytics dashboard, concept map, deep-dive views, and mobile tabs to share structure while remaining independently testable. React Hooks such as useState and useEffect were used to manage component state, viewport detection, and data loading. Vite provided the development server and production build toolchain for the project.

• React Router
React Router was used to implement client-side routing between student screens, including the analytics views and the mobile tab destinations, without requiring full page reloads.

• Monaco Editor
Monaco Editor, the editor that powers Visual Studio Code, was integrated inside the platform so that students write Python in a familiar editing environment. On mobile, the same editor is presented inside the Workspace tab, with surrounding chrome reduced so that the phone remains usable.

• REST API Integration
The frontend communicates with a FastAPI backend through REST endpoints. The screens and tests were written against the contracts for retrieving the next question, submitting an answer, loading a progress overview, loading per-concept progress, and loading the student profile. PostgreSQL stores the underlying records; the frontend consumes the JSON shaped by those tables rather than querying the database directly.

Taken together, the frontend stack used during the internship was React with Vite for the application shell, React Router for screen changes, Monaco Editor for the in-browser Python workspace, CSS custom properties for the shared theme, and the Fetch/REST layer against FastAPI. Vitest and React Testing Library sat beside that stack rather than after it: the same modules that render Home, Workspace, Progress, and Profile are the modules the 81 tests import. Docker provided a reproducible backend for understanding the PostgreSQL-backed payloads. Vercel was used to share running builds of the student interface. This combination kept the work modular, testable, and aligned with the rest of the FOSSEE frontend.

3.2 Development Tools and Collaboration

Working within a large open-source project required effective collaboration, version control, and organized development practices.

• Git
Git was used as the distributed version control system for managing source code throughout the internship. Changes were recorded in structured commits with conventional commit messages so that the history of analytics, mobile, and test

work remained readable to the rest of the team.

• GitHub
GitHub served as the central repository for project collaboration. I worked as a collaborator on the team repository, pushing frontend and test work for review and integration with the parallel frontend and backend streams.

• Visual Studio Code
Visual Studio Code was used as the primary development environment for writing, debugging, and maintaining the frontend source code. Its Git integration and editor tooling improved day-to-day development efficiency.

• Docker and Vercel
The backend was used in a containerized Docker environment so that API behaviour could be understood against a reproducible stack. Frontend deployment was carried out with Vercel for sharing running builds of the student interface.

3.3 Testing Frameworks and Quality Assurance

An important part of the internship was making the new student surfaces testable rather than relying only on manual clicking through the interface.

Vitest was introduced from scratch in the React Vite project as the unit-testing runner. React Testing Library was used for component and interaction tests, so that assertions described what the student sees and does rather than internal implementation details. API calls were mocked at the boundary, which allowed the suite to validate request wiring without depending on a live server during every run.

The resulting suite contains 81 tests in three groups: core logic, user-interface behaviour, and API integration. All 81 tests passed. Coverage included starter-code generation, mobile detection, answer-correctness heuristics, error-label validation from model output, mobile tab-bar rendering, tab switching, submit-button states, back-button navigation, and the calls for getNextQuestion, submitAnswer, getProgressOverview, getConceptProgress, and getProfile.

3.4 Chapter Summary

This chapter presented the technologies, frameworks, and development tools used during the internship. It described the use of React.js, Vite, HTML5, CSS3, and JavaScript for frontend development, Monaco Editor for the in-browser workspace, and FastAPI with PostgreSQL as the backend contract. The chapter also highlighted the role of Git, GitHub, Docker, and Vercel in managing the project, as well as the Vitest and React Testing Library suite used to keep the new screens verifiable.

Chapter 4
FRONTEND DEVELOPMENT AND USER INTERFACE IMPLEMENTATION

The primary responsibility assigned during the internship was the development of the Progress Analytics and mobile student layers of the Adaptive Coding Platform. While the backend services provided questions, evaluation, and progress records, and while a parallel frontend intern owned other desktop surfaces, students still needed a way to see their learning path and to keep working on a phone.

These screens were developed using React.js and modern web development practices. The implementation used a modular component-based architecture to improve maintainability, scalability, and code reusability. Throughout development, emphasis was placed on matching the shared design system, keeping navigation obvious, and leaving a clean boundary for later API substitution.

4.1 Frontend Architecture

The frontend follows a component-based architecture in which the application is divided into reusable modules responsible for different functionalities. This architecture improves maintainability by allowing independent development and testing of individual components while reducing code duplication.

React Router was used to manage client-side navigation. Viewport width is detected in the Workspace so that desktop and mobile presentations can share one route while switching layout. Each module is written to consume a narrow data contract, which is filled by mock objects during development and by REST responses after handoff.

The overall frontend architecture of the work consists of:

- Progress Analytics dashboard and stat cards
- Concept dependency map and expandable concept summaries
- Question Deep Dive
- Error Analysis
- Mobile application shell with four-tab bottom navigation
- Mobile Workspace with editor, hints, and submit
- Desktop Workspace adaptation for small screens

- Profile and session actions on mobile
- Shared design-system tokens and reusable UI pieces
- API communication layer with a replaceable mock

This modular design allows future enhancements, including live backend responses, to be implemented with minimal impact on existing presentation code.

4.2 Progress Analytics Interface

One of the major contributions during the internship was the Progress Analytics screen. The interface was built so that a student can see the entire learning journey in one place rather than inferring progress from the last question alone.

The dashboard opens with always-visible stat cards for total attempts, overall accuracy, concepts mastered, and current streak. These figures remain on screen so that deeper views never replace the student's basic standing in the course.

Below the cards is an interactive concept dependency map of six Python concepts arranged as a chain. Each concept is shown in one of three states: mastered, active, or locked. Clicking a concept circle expands a full summary beneath the map. The summary states what the student has learned, what remains, accuracy by error type, and a path-to-mastery section with progress bars. Locked concepts stay visible but are not presented as if they were already available, which keeps the dependency order honest.

Two further screens sit behind this overview. Question Deep Dive lists every question the student has attempted, together with the exact code submitted, the error made, and the feedback returned for that attempt. Error Analysis presents five error categories and breaks accuracy down per concept, so that a repeated mistake is visible as a pattern rather than as isolated failures. When a concept is completed, a mastery screen records attempts, accuracy, and streak, then unlocks the next concept in the chain.

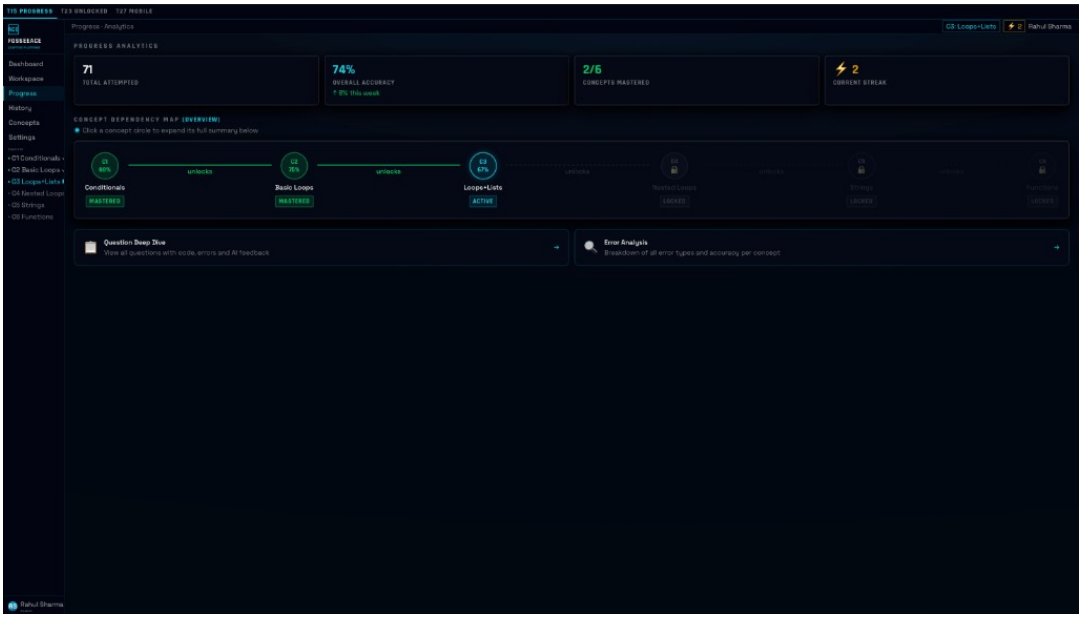


Figure 4.1 – Progress Analytics screen of the Adaptive Coding Platform

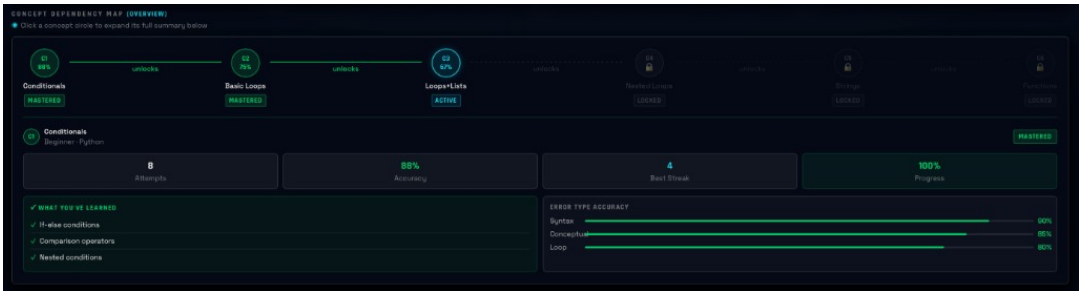


Figure 4.2 – Concept Dependency Map with expanded concept summary

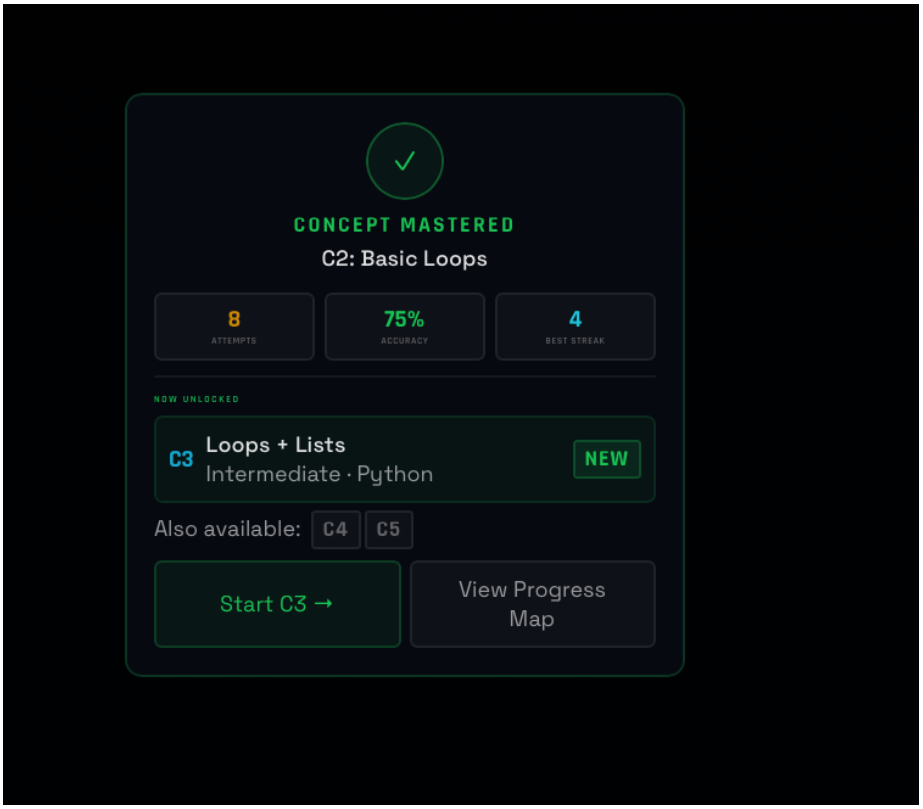


Figure 4.3 – Concept mastered and next-concept unlock screen

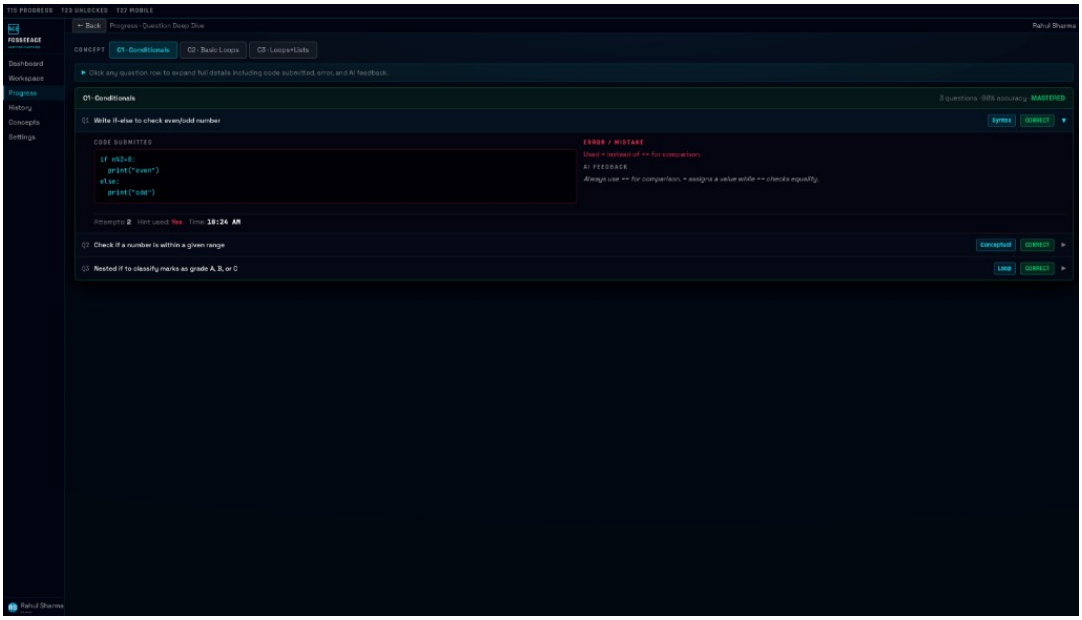


Figure 4.4 – Question Deep Dive showing submitted code, error and AI feedback

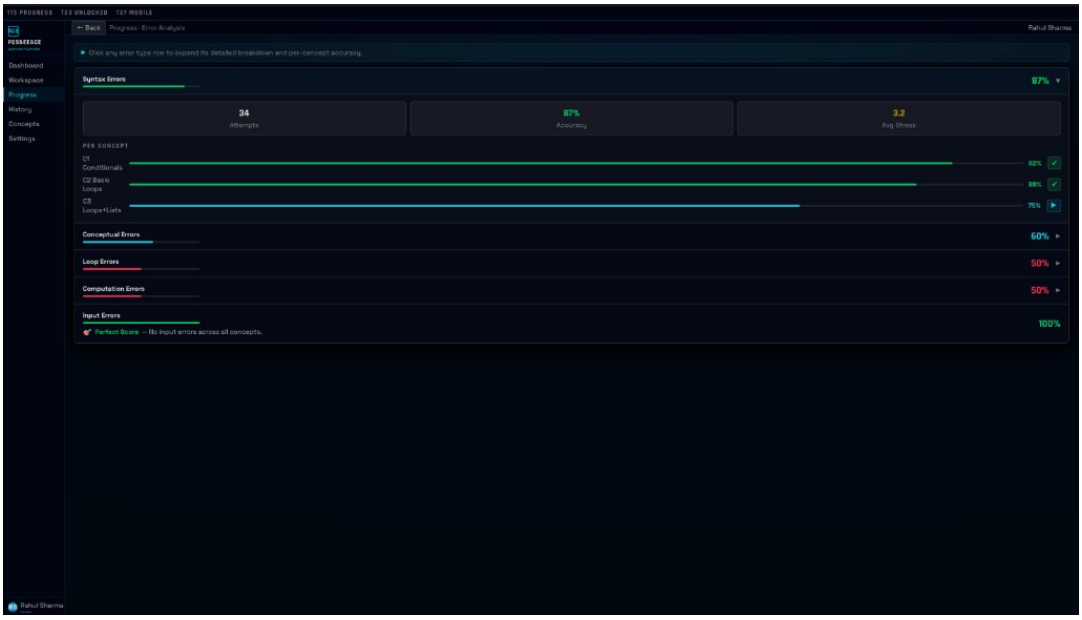
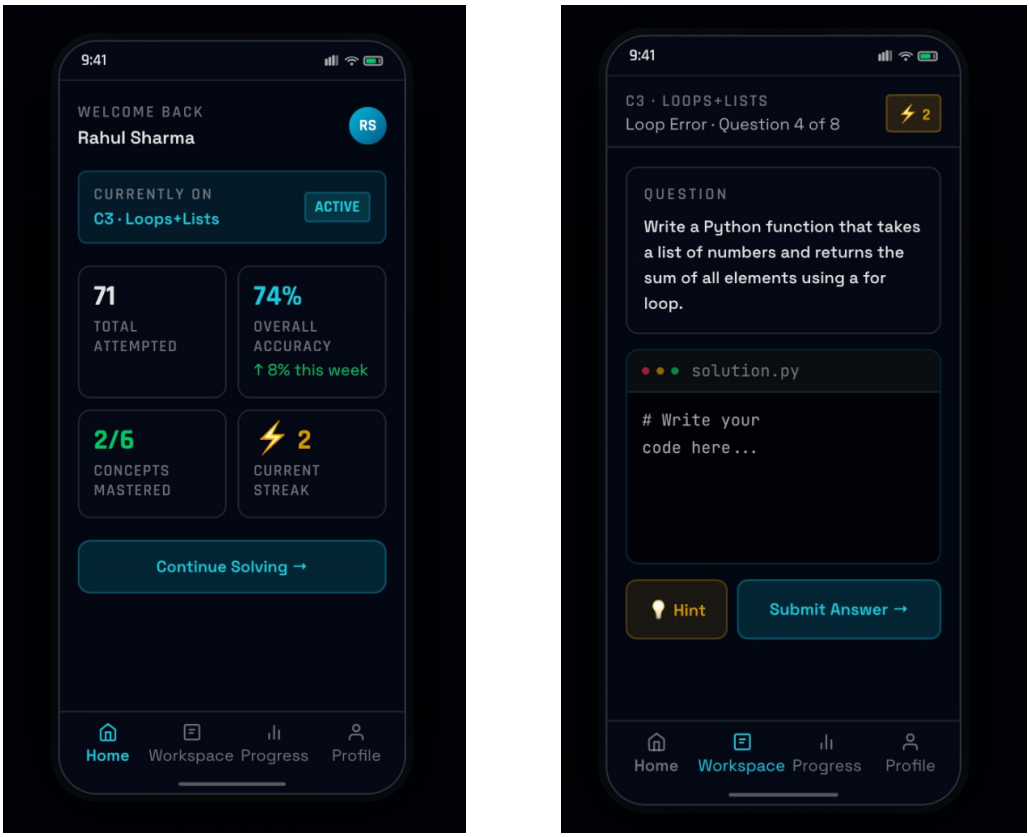


Figure 4.5 – Error Analysis with five error categories and per-concept accuracy

4.3 Mobile Workspace Implementation

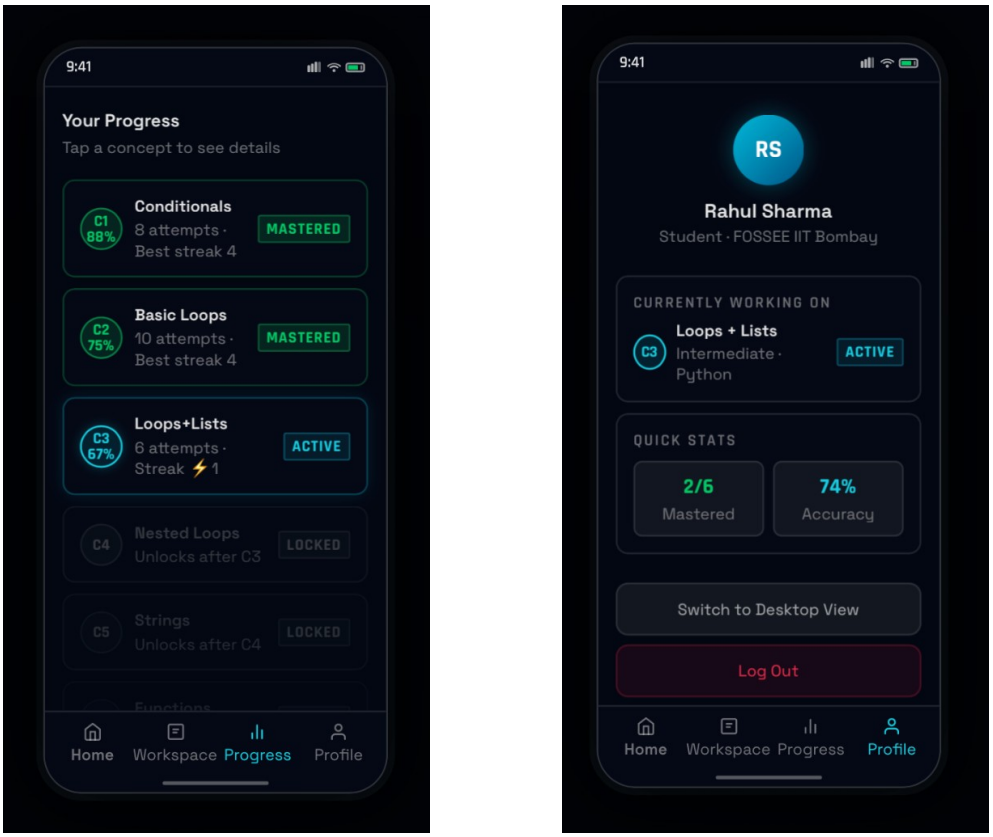
A second major contribution was a complete mobile-responsive interface. The desktop layout does not collapse cleanly onto a phone if the editor, problem text, and output are all asked to share one viewport. The mobile shell was therefore designed as its own information architecture rather than as a compressed copy of the desktop.

A four-tab bottom navigation bar provides Home, Workspace, Progress, and Profile. Home shows student stats and the current concept. Workspace is the solving surface: the problem, a Monaco editor, a hint system presented as a bottom sheet, and submit. Progress lists all six concepts with tap-to-expand statistics. Profile shows student information and logout.



(a) Home (b) Workspace

Figure 4.6 – Mobile Home and Workspace tabs



(a) Progress (b) Profile

Figure 4.7 – Mobile Progress and Profile tabs

The existing desktop Workspace was also made usable on phones. Screen size is detected at runtime. On a small viewport the same screen switches to a three-tab layout that shows Problem, Editor, and Output one at a time. This preserves the desktop information model without forcing three columns onto a narrow display.

4.4 React Component Handoff

The T15 Progress Analytics screen was converted into a production-ready React component tree with more than ten sub-components so that a teammate could take the work into the main application without reconstructing the layout. Presentation was kept separate from data. Mock objects supplied the dashboard during development; the same props are intended to be filled by FastAPI responses later.

This separation was deliberate. A screen that mixes fetch logic with markup is difficult to test and difficult to hand over. By keeping the component structure stable and isolating the mock, the backend team can plug in live payloads for progress overview, concept progress, and profile without rewriting the analytics UI.

4.5 User Interface Consistency and Design System

Visual consistency with the rest of the Adaptive Coding Platform was treated as a requirement, not as polish applied at the end. The parallel frontend intern had already established CSS variables and theme tokens. Those tokens were read and applied across analytics cards, the concept map, mobile chrome, and expanded summaries so that a student moving from Workspace to Progress does not feel as if they have entered a different product.

The same discipline applied to spacing, typography, and interactive states. Click targets on the concept map, expand and collapse behaviour on mobile Progress, and the hint bottom sheet were implemented so that they remain usable with a finger as well as with a pointer.

4.6 Chapter Summary

This chapter described the frontend work carried out during the internship. It discussed the component architecture, the Progress Analytics dashboard and its supporting screens, the mobile workspace and bottom navigation, the production component handoff, and the use of shared design tokens for visual consistency. Together these pieces give students a way to see mastery and a way to keep practising on a phone, without breaking the look of the wider platform.

Chapter 5
BACKEND INTEGRATION AND SYSTEM FUNCTIONALITY

After the analytics and mobile interfaces were in place, the next major phase of the internship was to treat those screens as clients of the backend rather than as static mockups. The FastAPI services already held the business logic for questions, submissions, progress, and profile. Successful operation of the student experience depended on consuming those services through a stable REST contract and on proving that contract with tests.

The integration work therefore had two sides. One side was understanding the endpoints and the PostgreSQL-backed records they expose. The other was keeping the React tree independent of any one payload source, so that mocks, tests, and live responses could occupy the same slots.

5.1 REST API Integration

The Adaptive Coding Platform follows a client-server architecture in which the React frontend communicates with FastAPI services through RESTful APIs. Every student action that requires server-side processing is performed by sending an HTTP request and rendering the corresponding response.

The frontend was integrated against backend operations including:

- Retrieving the next question for the current student and concept (getNextQuestion).
- Submitting an answer and receiving evaluation feedback (submitAnswer).
- Loading the progress overview used by the analytics dashboard (getProgressOverview).
- Loading per-concept progress for the dependency map and expanded summaries (getConceptProgress).
- Loading student profile information for the Profile tab (getProfile).

These five calls define the student loop: see standing, open a concept, receive a question, submit a solution, and inspect what changed. The analytics and mobile screens are arranged around that loop rather than around a separate, decorative data model.

5.2 Dynamic Data Communication

The frontend was designed to display information retrieved from backend services instead of relying on hard-coded copy inside each component. When a student opens Progress, the overview and concept records are requested. When a concept is expanded, the detailed progress for that concept is shown. When the Workspace asks for the next problem, the editor is filled from the response rather than from a fixture baked into the view.

This dynamic communication architecture improves scalability by separating presentation from backend business logic. It also allows the same component to be driven by a mock during development and by a live FastAPI process once the payload shape is stable.

5.3 Mock Data Separation and Backend Readiness

A recurring risk in frontend internships is that screens ship with sample numbers woven through the markup. Replacing those numbers later requires touching every component. The Progress Analytics tree was therefore written so that mock data lives outside the visual components. Cards, the concept map, deep dive, and error analysis all render from props. Substituting a JSON response from FastAPI does not require a redesign of the tree.

Understanding the PostgreSQL schema was part of making that substitution realistic. The mock objects were shaped against the fields the backend actually stores for attempts, accuracy, concept state, and profile, rather than against an invented dashboard model that the API would later be forced to match.

5.4 Unit Testing and Validation

Vitest was set up from scratch in the React Vite project. The suite is organized in three groups and contains 81 tests, all of which passed.

The first group covers core logic: starter-code generation, mobile detection, the heuristic used to judge whether an answer is correct, and validation of error labels returned from the model. These tests pin down behaviour that would otherwise drift when UI code is refactored.

The second group covers UI components and interactions using React Testing Library. It includes rendering of the mobile tab bar, tab switching, submit-button states, and back-button navigation. The tests describe the student-visible contract of the mobile shell.

The third group covers API integration. The calls for getNextQuestion, submitAnswer, getProgressOverview, getConceptProgress, and getProfile are asserted with mocks, so that a broken path or a renamed endpoint fails in CI instead of in a live session.

5.5 Testing Activities

Testing activities during this phase included:

- Verifying that analytics cards render the values supplied by the progress overview.
- Validating concept states on the dependency map against per-concept progress.
- Confirming that Question Deep Dive and Error Analysis consume attempt history without mixing concepts.
- Checking mobile tab rendering, switching, submit states, and back navigation.
- Asserting the five primary API calls used by the student loop.
- Running the full suite to completion with zero failures before the work was pushed for review.

5.6 Chapter Summary

This chapter presented the backend integration process carried out during the internship. It discussed REST communication with FastAPI, the student API surface, dynamic data handling, and the separation of mock data from presentation. It then described the Vitest suite of 81 passing tests across core logic, UI behaviour, and API integration, which made the new screens checkable rather than only demonstrable.

Chapter 6
USER INTERFACE ENHANCEMENT, ACCESSIBILITY AND APPLICATION OPTIMIZATION

Developing a functional screen is only one aspect of modern software engineering. Equally important is ensuring that the application remains readable, responsive, and consistent across devices. After the analytics dashboard and mobile shell were in place, the next phase of the internship focused on refining the interface, tightening the match with the shared design system, and confirming that both phone and desktop layouts remained usable.

6.1 User Interface Refinement

The analytics and mobile interfaces were refined to improve visual consistency and simplify interaction. Stat cards, the concept chain, expandable summaries, and the mobile tab bar were adjusted so that spacing, type, and alignment followed the same rules as the rest of the platform.

The major interface improvements included:

- Applying shared CSS variables and theme tokens across analytics and mobile screens.
- Standardizing card, map, and list layouts so Progress feels like part of the same product as Workspace.
- Keeping attempts, accuracy, mastery, and streak visible while deeper views are open.
- Making concept circles clickable with a clear expanded summary rather than a hidden tooltip.
- Presenting hints on mobile as a bottom sheet so the editor is not permanently covered.
- Refining submit and back controls so their enabled and disabled states are obvious.

6.2 Mobile-First Responsive Design

Students cannot be assumed to work only at a desktop. Responsive design was therefore treated as part of the product, not as a later adaptation. The mobile shell

uses a four-tab bottom navigation model that is familiar on phones and keeps Home, Workspace, Progress, and Profile within one thumb reach.

On Progress, the six concepts remain available as a tap-to-expand list rather than as a wide map that would overflow a narrow viewport. On Workspace, the editor, hint sheet, and submit control are stacked so that a student can write and send a solution without horizontal panning.

6.3 Desktop Workspace Adaptation for Mobile

The existing desktop Workspace still needed to work when a student rotated a laptop or opened the same route on a phone. Rather than maintaining two unrelated implementations, screen size is detected and the desktop Workspace switches to a three-tab layout. Problem, Editor, and Output occupy the screen one at a time.

This approach preserves a single source of workspace behaviour while changing only the chrome that the viewport can support. The student does not lose output; they choose when to look at it.

The three-tab layout was chosen after trying to keep problem text, editor, and output on one phone screen. That arrangement left too little vertical space for Monaco and forced horizontal panning. Splitting the same route into Problem, Editor, and Output tabs keeps one source of question data, one submit handler, and one output buffer. Viewport detection is a width check rather than a separate mobile application. Below the breakpoint the tabs appear; above it the desktop three-pane workspace remains. Hints continue to open as a bottom sheet so they do not cover the editor permanently. The result is that a student who starts a question on a laptop can finish it on a phone without a second product to learn.

6.4 Application Optimization

Optimization work focused on keeping the new screens reliable rather than on adding further features. Components were split so that the concept map, deep dive, and error analysis could be tested and reused independently. Mock data was kept out of the visual tree. Mobile detection was isolated so that layout switching could be unit-tested instead of being buried in a large page component.

These choices reduced the chance that a later API substitution or a small visual change would require rewriting the analytics or mobile shell.

6.5 Manual Testing and Validation

Automated tests were complemented by walking through the student flows by hand, particularly for interactions that are awkward to fully encode in a unit suite, such as the feel of the hint bottom sheet and the concept-map expansion.

Manual testing activities included:

- Walking the analytics dashboard from stat cards through concept expansion.
- Opening Question Deep Dive and Error Analysis with representative attempt data.
- Using the four-tab mobile shell as a student would: Home, solve, inspect Progress, open Profile.
- Checking the three-tab desktop-on-mobile Workspace layout.
- Confirming that design-system colours and spacing matched neighbouring screens.
- Re-running the 81-test suite after interface changes.

6.6 Chapter Summary

This chapter discussed the improvements made to the student interface after the first implementation of analytics and mobile. It highlighted design-system alignment, mobile-first navigation, the three-tab adaptation of the desktop Workspace, component isolation, and the combination of automated and manual checks used to keep the experience stable.

Chapter 7
RESULTS, CHALLENGES AND FUTURE SCOPE

Throughout the internship, the work moved from individual analytics screens to a mobile student shell, a handoff-ready component tree, and an automated test suite. The challenges were less about missing backend services and more about making adaptivity visible, making a phone a serious workspace, and keeping that work integrable by the rest of the team.

7.1 Technical Challenges and Solutions

The major challenges encountered during the internship included:

Making Progress Visible

Adaptive selection is useless to a student who cannot see mastery, errors, or the next concept. The analytics dashboard, six-concept map, Question Deep Dive, and Error Analysis were built so that standing, history, and patterns occupy the same product surface.

Concept Map Interaction

A static diagram of six concepts would have been easy and largely unread. The map was implemented as a chain of mastered, active, and locked states. Each circle opens a full summary with remaining work, error-type accuracy, and a path to mastery. The difficulty was packing that information without turning the screen into a wall of text.

Mobile Editor Experience

Hosting Monaco on a phone, together with problem text, hints, and submit, required a different chrome from the desktop. Bottom navigation, a hint bottom sheet, and a single-column Workspace were used so that the editor remains the focus.

One Workspace, Two Layouts

Duplicating the desktop Workspace for mobile would have split behaviour. Screen-size detection and a three-tab Problem / Editor / Output layout allowed one workspace implementation to serve both classes of device.

Design-System Alignment

Analytics and mobile work landed in parallel with other frontend development. Exact CSS variables and theme tokens were applied so that the new screens did not drift from the platform look.

Handoff Without Rework

A large analytics screen is difficult to integrate if mock numbers live in the markup. The T15 tree was split into more than ten sub-components and the mock was kept outside the UI so FastAPI responses can be connected without restructuring the view.

Automated Confidence

Manual demonstration does not protect refactors. Vitest was introduced from scratch and 81 tests were written across logic, UI, and API mocks so that regressions fail the suite instead of a live student session.

7.2 Results and Project Impact

The work completed during the internship gave students a way to see their path through the Adaptive Coding Platform and a way to continue that path on a phone.

Progress Analytics presents attempts, accuracy, mastery, and streak, then lets the student walk a six-concept chain, inspect every attempt in Question Deep Dive, and read five error categories with a per-concept breakdown. The mobile shell provides Home, Workspace, Progress, and Profile. The desktop Workspace remains usable on small screens. The analytics tree is packaged for backend integration, and the test suite reports 81 passing tests.

The overall outcome of the internship included:

- A complete Progress Analytics dashboard with always-visible standing and a six-concept dependency map.
- Question Deep Dive and Error Analysis covering attempt history and five error categories.
- A four-tab mobile student shell with a usable Workspace, including editor, hints, and submit.
- A three-tab adaptation of the desktop Workspace for small screens.
- A production-ready React component tree for T15 Progress Analytics, with mock data isolated from presentation.
- A Vitest suite of 81 tests covering core logic, UI interactions, and API integration, all passing.
- Visual alignment with the shared platform design system.

7.3 Future Scope

Although the Adaptive Coding Platform now has student analytics and a mobile workspace, several enhancements can further improve its capabilities.

• Connecting the analytics component tree to live FastAPI responses in production and retiring the development mock.

• Extending the concept map beyond the present six Python concepts as the curriculum grows.

• Adding instructor-facing analytics that reuse the same progress and error records.

• Deepening mobile editor ergonomics, including better keyboard handling around the hint sheet.

• Expanding automated coverage from unit tests into broader integration tests against a running backend.

• Strengthening accessibility of the concept map and mobile tabs against WCAG guidance.

• Using the error-category breakdown to drive the next-question policy more explicitly in the interface.

7.4 Chapter Summary

This chapter discussed the major technical challenges encountered during the internship, the solutions implemented, and their impact on the Adaptive Coding Platform. It highlighted Progress Analytics, the mobile workspace, component handoff, design-system alignment, and a passing unit-test suite. Finally, the chapter outlined future enhancements that can further improve the platform's usefulness to students and instructors.

Chapter 8
CONCLUSION

The internship under the FOSSEE (Free/Libre and Open Source Software for Education) project at the Indian Institute of Technology Bombay (IIT Bombay) provided an excellent opportunity to apply theoretical knowledge to the development of a real-world open-source software project. Working on the Adaptive Coding Platform (Yaksh) enabled me to gain practical experience in modern frontend development, mobile interface design, REST integration, unit testing, and collaborative software engineering.

A major achievement during the internship was the Progress Analytics layer. Students can now see attempts, accuracy, mastery, and streak, walk a six-concept dependency map, review every submitted attempt, and inspect five error categories. That work turns adaptivity into something a learner can inspect rather than something that only happens behind the next-question call.

A second achievement was the mobile student experience: four-tab navigation, a phone-sized Workspace with Monaco, hints, and submit, tap-to-expand Progress, and a Profile tab, together with a three-tab layout of the desktop Workspace on small screens. A third was engineering hygiene: a handoff-ready component tree with mock data held apart from the UI, and 81 passing unit tests across logic, interface, and API contracts.

Throughout the internship I also strengthened my understanding of professional software development practices, including component-based application development, design-system coordination, REST API integration, version control using Git and GitHub, automated testing, and open-source contribution workflows. Working under experienced mentors and collaborating with fellow developers provided valuable insight into large-scale software engineering and team-based project development.

Overall, this internship significantly enhanced both my technical knowledge and professional skills. It provided practical exposure to building student-facing product surfaces on a production-level educational platform while reinforcing the importance of clean component boundaries, collaboration, and systematic testing. The experience has prepared me for future opportunities in software development and has strengthened my confidence in contributing to complex open-source projects and modern web application development.

Chapter 9
CONCLUSION AND LEARNING OUTCOMES

The internship under the FOSSEE (Free/Libre and Open Source Software for Education) project at the Indian Institute of Technology Bombay (IIT Bombay) provided invaluable practical experience in modern web application development and open-source software engineering. Working on the Adaptive Coding Platform (Yaksh) enabled me to apply theoretical concepts acquired during my academic studies to the development of a real-world software system used for programming education and assessment.

One of the most significant achievements of the internship was the Progress Analytics and mobile workspace work. Starting from the need to make learning progress visible, the project grew into a dashboard, a concept map, deep-dive and error views, a four-tab mobile shell, and a tested component handoff. The experience strengthened my understanding of component-based architecture, mobile-first layout, REST API contracts, and frontend testing with Vitest and React Testing Library.

The internship also involved coordinating with a parallel frontend intern on a shared design system and with the backend team on the shape of progress and profile payloads. Separating mock data from presentation, and pinning the student API loop with mocked tests, improved my sense of how a frontend is actually integrated rather than only drawn.

Throughout the internship, I gained practical experience in collaborative software development by working with mentors and fellow developers in an open-source environment. Regular integration, version control using Git and GitHub, structured commit messages, automated testing, and iterative development provided valuable exposure to professional software engineering workflows followed in real-world projects.

The internship significantly strengthened both my technical and professional competencies. In addition to improving my programming and development skills, it enhanced my ability to break a student product problem into screens, data contracts, and tests, to communicate those pieces to teammates, and to keep visual consistency across a shared codebase. The experience also increased my confidence in contributing to complex open-source projects and understanding software development from interface design through to verification.

Learning Outcomes

The internship helped me achieve the following learning outcomes:

- Developed a strong understanding of React.js and component-based frontend development, including a multi-screen analytics tree and a mobile application shell.
- Learned to design student-facing analytics: stat cards, concept dependency maps, attempt history, and error-category breakdowns.
- Gained practical experience in mobile-first layout, bottom navigation, and adapting a desktop workspace to a three-tab phone layout.
- Learned to integrate frontend screens with backend REST APIs, including next-question, submit, progress, concept, and profile calls.
- Understood how to isolate mock data from UI components so that FastAPI responses can replace fixtures without a redesign.
- Gained practical experience in Git, GitHub, and collaborative software development workflows, including conventional commit messages.
- Set up Vitest and React Testing Library from scratch and wrote a passing suite covering logic, UI, and API integration.
- Learned to match a shared CSS design system so that new screens remain visually consistent with work owned by other interns.
- Acquired experience in contributing to a large-scale open-source educational software project.
- Strengthened communication, teamwork, and professional development skills.

In conclusion, the internship served as an excellent bridge between academic learning and professional software development. It provided practical exposure to building student-facing web applications, solving real product constraints on mobile and desktop, and contributing to an impactful open-source project. The knowledge, experience, and skills gained during this internship will serve as a strong foundation for my future career in software engineering and frontend web development.