A REPORT

ON

SCREEN DEVELOPMENT USING EXTENSIBILITY TOOLKIT

Submitted by,

Yerramilli Sai Prateek - 20211COM0042

Under the guidance of,

Prof. Mohamed Shakir

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER ENGINEERING

At

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY, BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the work, which is being presented in the report entitled **"SCREEN DEVELOPMENT USING EXTENSIBILITY TOOLKIT"** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Engineering**, is a record of my own investigations carried under the guidance of **Prof. Mohamed Shakir, School of Computer Science and Engineering & Information Science, Presidency University, Bengaluru.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

12/5/25

Yerramilli Sai Prateek,

20211COM0042

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship/Project report **"SCREEN DEVELOPMENT USING EXTENSIBILITY TOOLKIT"** being submitted by "Yerramilli Sai Prateek" bearing roll number "20211COM0042" in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Engineering is a bonafide work carried out under my supervision.

12.5.2025

Prof. MOHAMED SHAKIR
Assistant Professor
School of CSE&IS
Presidency University

Dr. GOPAL KRISHNA SHYAM
Professor & HoD
School of CSE&IS
Presidency University

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vice Chancellor - Engineering
Dean –PSCS / PSIS
Presidency University

INTERNSHIP COMPLETION CERTIFICATE

Profinch®

9th May, 2025

Ref: Profinch/HR misc/2025/0041/05

To whomsoever it may concern

Sub: Undergoing Internship

This letter certifies that **Mr. Yerramilli Sai Prateek, Emp Id T0673** a student of **Presidency University, Bengaluru** is undergoing his internship at **Profinch Solutions Private Limited, Bengaluru** for **6 months** from the period **20-Jan- 2025 to 19-Jul-25**.

The details are as follows:

Name of Student: Yerramilli Sai Prateek
Course: B.Tech in Computer Engineering
Institute: Presidency University, Bengaluru.
Internship Duration: 20-Jan- 2025 to 19-Jul-25 [6 months]
Project title: Development of Customized features in a Core Banking System.

He is diligent and enthusiastic with zeal to do his best on his training. His overall performance and conduct are found to be good. We wish him success in his future endeavours.

Yours sincerely,
For Profinch Solutions Pvt Ltd

PROFINCH SOLUTIONS PRIVATE LIMITED * BANGALORE

Priyanjana Dey
Senior Manager - Human Resources

Profinch Solutions Pvt Ltd
86, GF, Wings of Eagles, SS Commercial Estate, Varthur Road, Nagavarapalya,
CV Raman Nagar, Bangalore – 560 093. | Tel: +91 80 4256 4256, Fax: +91 80 4257 4257
CIN Number: U72400KA2014PTC074611

ABSTRACT

My internship at Profinch Solutions Pvt. Ltd. has been crucial between academic study and real-world software development. Over these months, I've not only understood the technical aspects, but also got to experience few important tasks like problem-solving, teamwork, and understanding how large-scale enterprise applications are built and maintained. Working on live banking projects gave me insight into mission-critical systems and the discipline required to keep them running smoothly.

A major focus of my work was building and integrating microservices using a Java-based framework. I learned how to design services that can scale independently, define clear RESTful endpoints, and manage configuration for reliable deployment across the various environments. This enhanced my understanding of how modern financial platforms assemble small, focused services into cohesive applications that handle high volumes of transactions with minimal latency.

At the database layer, I dove deep into procedural SQL development—authoring and tuning stored procedures, writing queries, and enforcing data rules close to the data itself. These activities demonstrated how core banking engines depend on efficient, ACID operations to maintain account balances, generate reports, and feed analytics systems in real time.

I also contributed to the design of data-entry screens used by the customers. Using a low-code screen-builder toolkit and an extensibility framework, I assembled form layouts and then layered in both field-level and form-level checks. For example, I ensured identifiers were auto-generated, dates fell within allowed ranges, and text fields conformed to formatting rules. Aligning these client-side features with server-side validations helped me prevent bad data from ever reaching the database—a vital practice in any regulated environment.

Maintaining uninterrupted service is very important in banking. I practiced deploying updates to an application server cluster—packaging services as deployable archives, updating configuration files, and activating new versions without taking down existing functions. When errors occurred, I used the secure file-transfer tools to retrieve server logs, traced exceptions back through Java stack traces and SQL error codes, and applied fixes. This log-first approach allowed me to resolve issues rapidly while preserving system availability.

Overall, this ongoing internship has equipped me with a balanced mix of hands-on technical skills and professional habits. I now understand how to build, validate, deploy, and support enterprise-grade services in a high-stakes environment—and how to work effectively within a team to deliver them.

LIST OF FIGURES

ACKNOWLEDGEMENTS

First of all, we indebted to the **GOD ALMIGHTY** for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Deans **Dr. Shakkeera L and Dr. Mydhili Nair,** School of Computer Science Engineering & Information Science, Presidency University, and Dr. "Gopal Krishna Shyam", Head of the Department, School of Computer Science Engineering & Information Science, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Prof. Mohamed Shakir, Assistant Professor** and Reviewer **Mr. Muthuraj, Assistant Professor**, School of Computer Science Engineering & Information Science, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the CSE7301 Capstone Project Coordinators **Dr. Sampath A K, Dr. Abdul Khadar and Mr. Md Zia Ur Rahman,** department Project Coordinators "Dr. Sudha P" and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

12/5/25

Yerramilli Sai Prateek,

20211COM0042

CHAPTER 1

INTRODUCTION

In an era where customers expect seamless, on-demand financial services, banks are rapidly modernizing their systems. Today's institutions must support real-time transactions, data-driven personalization, and robust security across web, mobile, and branch channels. Recent industry research found that in 2024, over half of retail banking interactions occurred via mobile apps—highlighting the shift to omnichannel, always-on service delivery. To keep pace, banks are replacing legacy mainframe cores with scalable, cloud-ready platforms that break down operational silos and enable continuous deployment of new features.

1.1 The Rise of Digital Banking

Digital banking now blends advanced analytics, AI-driven fraud detection, and open APIs to deliver personalized experiences. Institutions leverage these technologies to streamline onboarding, speed payments, and offer 24×7 self-service. As fintech challengers push innovation, traditional banks invest in modular architectures that can evolve without large-scale rewrites—ensuring they remain competitive in a landscape defined by speed, security, and customer centricity.

1.2 About Profinch Solutions Pvt. Ltd.

Profinch Solutions is a Bengaluru-headquartered technology consultancy specializing in financial services. Since its founding in 2014, the firm has grown to several hundred employees across India, Singapore, and Dubai. Profinch helps banks and insurers modernize by combining advisory services with configurable software modules for account management, payments, compliance, and analytics. Their approach emphasizes rapid time-to-value through reusable components and best-practice process frameworks.

1.3 Core Banking Platform Overview

At the heart of every bank lies its core processing engine, responsible for accounts, ledgers, loans, and transaction posting. Earlier generations were monolithic and inflexible; today's platforms are built in discrete modules that can be deployed independently. In this internship, I worked with such a modular core engine—selecting only the required functional packages and configuring them via business-rule tables. This architecture supports multi-currency,

TABLE OF CONTENTS

1.8 Key Takeaways

This internship sharpened my command of enterprise banking software—from backend data management to microservices design and UI validation. More importantly, it revealed how technology choices map directly to business outcomes: faster feature rollout, stronger controls, and better customer experiences. I now appreciate the balance between stability and agility that modern financial systems demand.

multi-region operations and enables real-time risk controls and reporting.

1.4 Relational Database Foundations

Underpinning every transaction is a relational database that guarantees ACID properties and high concurrency. These systems manage massive volumes of data, provide encryption and auditing features, and offer in-memory options for analytics. My assignments demonstrated how stored routines and batch processes maintain data integrity, reconcile accounts overnight, and feed dashboards used by risk and compliance teams.

1.5 Extensibility Toolkit for Microservices

To extend the core engine without touching its base code, banks use a metadata-driven extensibility toolkit. This toolkit generates service skeletons, UI fragments, and database scripts—allowing developers to plug in new workflows (e.g., customized loan-approval checks or fee-calculation services). These extensions run alongside the core modules, register with a central service directory, and expose RESTful endpoints for front-end applications. During my internship, I used this toolkit to scaffold, implement, and test several new microservices in a sandboxed environment.

1.6 Deepening Procedural SQL Skills

Writing correct SQL is only half the story—enterprise reliability demands well-structured, maintainable routines. I designed and optimized stored procedures that encapsulated complex business logic (interest computations, batch settlements, audit logging) into reusable units. I also built exception-handling frameworks that record errors in audit tables, enabling operations teams to resolve data issues without interrupting critical overnight jobs.

1.7 Customizing Core Modules for Client Needs

Beyond out-of-the-box functionality, I configured and extended database logic to support nonstandard requirements. By layering custom rules on top of the core engine, I saw how a modular architecture permits safe, upgrade-friendly changes that align tightly with each client's processes.

2.7 Aditya Choudhury (2025) shows how small tweaks in procedural SQL—like flattening loops and adding the right index—can turn an all-night batch job into a quick, pre-morning task. It's a fast-win that any data team can celebrate.

2.8 Niraj Sinha (2023) ran side-by-side tests of Docker containers and virtual machines. Containers started and recovered in seconds, letting operations teams pack more onto each server. His bottom line: containerization really does simplify life at scale.

2.9 Preeti Sharma (2022) tells the story of fintech groups that layered in automated build-and-test pipelines around their Java services. The result was a steep drop in production bugs and the confidence to ship multiple times per day.

2.10 IBM's 2024 report popularized the strangler pattern: wrap the legacy core in an adapter, then gradually replace pieces with new services until the old code disappears. It's a safe path that avoids the all-or-nothing risk of big-bang rewrites.

2.11 A 2022 cloud-provider guide lays out a secure template for running banking services in the public cloud—API gateways to guard traffic, auto-scaling for spikes, and hardware-backed vaults for secrets. It's a handy recipe for banks moving off-premises.

2.12 Siddharth Roy (2025) recommends sprinkling structured log statements and trace tables around each database call. When something goes wrong, those breadcrumbs make it much faster to track down the culprit.

2.13 A Spring.io community roundup (2023) collects war stories of banks that broke features—account searches, statement exports—into tiny REST services. The payoff: easier testing, faster fixes, and happier developers.

CHAPTER 2
LITERATURE SURVEY

2.1 Syed Ali Fathima (2025) tells how big banks and insurers tame their monster legacy systems by carving them into independent services. She walks through practical tricks—like firing off events when something happens, automatically tripping a "circuit breaker" if one piece misbehaves, and stitching multi-step transactions together so everything stays consistent. Her real-world examples show how container platforms and orchestration tools let you update bits of the system on the fly, without taking the whole bank offline.

2.2 Sumit Bhatnagar and Roshan Mahant (2024) zoom in on security: once you break an app into services, how do you spot if one starts acting strangely? They built a lightweight watcher that lives alongside each service, flagging odd patterns—like repeated failures—that might otherwise go unnoticed. In their Finbox story, this extra guardrail meant faster response times and fewer surprise outages.

2.3 In early 2025, **Bhatnagar** came back with a cost-cutting twist: mix always-on services with pay-as-you-go "serverless" functions for occasional jobs. A few fintech shops tried it and saw their cloud bills drop by nearly half, all while pushing new features more frequently. It's a solid reminder that clever architecture can pay big dividends.

2.4 Varun Tambi (2023) writes from the trenches of a regional bank migration. Instead of ripping out the whole mainframe, he peeled off one feature—say, account lookups—into a small service, tested it, then moved on to the next. That step-by-step path kept customer services live and avoided the nightmares of a big-bang rewrite.

2.5 Ramesh Iyer and **Anitha Das** (2024) tackle the headache of changing regulations. They show how to spin up tiny "validation engine" services that read the latest rulebook from data instead of code. When a law changes, you update the data, not the binaries—so compliance updates go out in hours, not weeks.

2.6 Meena Krishnamurthy (2023) captures the excitement of teams in India swapping dusty legacy modules for fresh Java services. Her interviews reveal happier testers (because automated checks actually pass) and release days that feel routine instead of nerve-wracking.

CHAPTER 3

OBJECTIVES

During my internship, I pursued the following goals to build targeted skills in enterprise banking software development:

3.1 Exploring Core Banking Engines

Gain direct experience with a multi-layered core banking platform—examining its presentation, business logic, and data tiers—to appreciate how legacy, monolithic designs differ from modern, modular architectures in a production setting.

3.2 Gaining Experience with an Extensibility Toolkit

Use a metadata-driven extension framework to generate, adapt, and manage new service plug-ins. Validate these additions end-to-end in an isolated test environment to show how features can be safely layered on top of a stable core.

3.3 Building Expertise in a Modern Fintech Stack

Sharpen my skills in backend and API development—Java, a microservices framework, RESTful design, and SQL/PL-SQL—and integrate these with a DevOps toolchain deliver, test, and monitor services.

3.4 Navigating The Software Lifecycle in Financial Projects

Participate in all Agile ceremonies—requirements workshops, sprint planning, development, testing, and production rollout—to understand how regulated fintech features progress from concept to live service.

3.5 Strengthening Troubleshooting and Debugging

Adopt a log-centric debugging approach—examining application server logs and database traces—to isolate and resolve issues in the user interface, service layer, or data layer without disrupting critical banking operations.

3.6 Fostering Effective Teamwork and Communication

Practice clear, concise exchanges with peers, tech leads, and project managers. Enhance my written skills by producing crisp documentation and status reports, and refine my presentation

CHAPTER 4

IMPLEMENTATION

During my internship, I was responsible for enhancing our bank's core platform with two major deliverables. First, I built a custom Organization Details form to streamline how operations staff register corporate clients and their departments, ensuring data quality and compliance. Second, I managed the full microservice extension deployment, introducing a new REST API into our middleware without disrupting live services. Both tasks demanded careful UI design, robust validation, and precise deployment practices in a production-like environment.

Task 1: Organization Details Form

Introduction

The goal was to create a user-friendly data-entry form within the banking platform that captured all essential information about new corporate clients. This included general organization attributes, mailing address, contact information, and a repeating section for departmental details. Because this data feeds into critical back-office workflows (account setup, compliance checks, reporting), it was vital to enforce multiple layers of validation to prevent bad or duplicate records.

Approach

I began by designing the screen layout with the platform's low-code form builder, grouping related fields into logical sections. On the backend, I implemented a trigger to auto-generate a unique Organization ID whenever the user clicked "Generate ID." I added real-time field checks: emails were matched against a regex pattern; phone numbers required a country-specific format when "India" was selected and were otherwise flexible; establishment dates had to be at least one year in the past; and address lines disallowed special characters and auto-capitalized on exit. In the department table, I ran a uniqueness check on the Department Head field before allowing submission. Inline error messages guided users to correct mistakes immediately, ensuring only valid, compliant records reached the database.

Task 2: Microservice Extension Deployment

Introduction

style for stakeholder demos and knowledge-share sessions.

3.7 Linking Development to Business Impact

Demonstrate how secure login flows, validation rules, and scalable payment APIs translate into improved customer satisfaction, regulatory compliance, and operational efficiency.

The second assignment involved deploying a newly developed microservice into our middleware cluster. This service exposed a REST endpoint that extended core banking functionality, and it needed to be integrated without any interruption to existing operations or data conflicts.

Approach

First, I updated the service code to standardize date/time serialization and align REST annotations on both its API interface and controller. I replaced the default logging configuration with a custom setup that writes to our approved log directory, rotates logs daily, and supports debug-level output. For database migrations, I organized scripts so most applied automatically at startup, and I logged any exceptions manually in our ledger table. I then packaged the service as a deployable archive and used the admin console's "lock and edit" mode to upload and install it in one atomic action. Initial authentication errors were traced by decompiling the authorization library, identifying missing role-activity mappings, and inserting the necessary entries into our security table. Finally, I generated a Postman collection, and tested every endpoint—supplying required headers—to confirm successful operation without downtime.

CHAPTER 5

TIMELINE FOR EXECUTION OF INTERNSHIP (GANTT CHART)

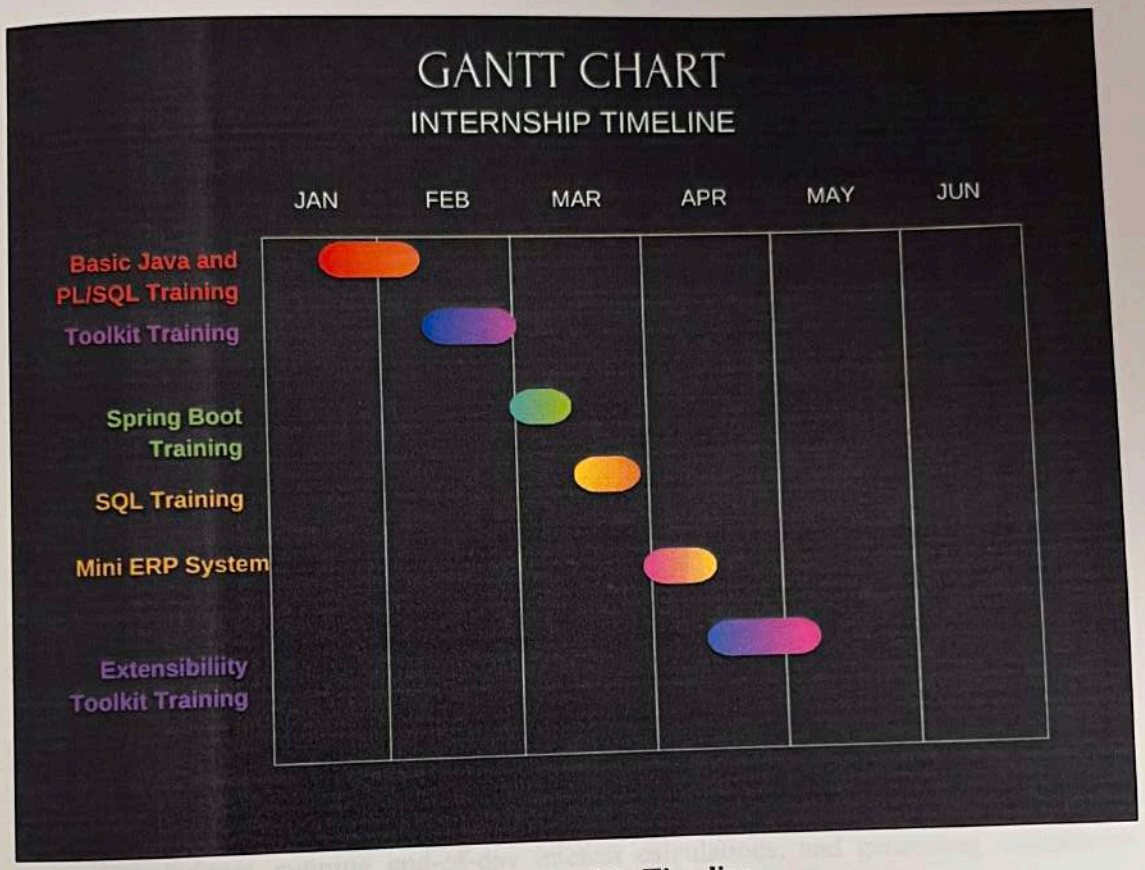


Fig 5.1 Internship Timeline

CHAPTER 6

OUTCOMES

6.1 Monolithic vs. Microservices Architecture

By working on both the legacy core platform modules and the new modular services, I saw how a single, giant application slows you down, even a tiny change means rebuilding and retesting the whole thing. That can delay releases for hours or days. Switching to small, independent services changed everything. When we needed to tweak the loan-calculation logic, we updated just that piece in under ten minutes—no impact on payments or account services. The result? Faster updates and far less risk of breaking something else.

6.2 Technical Skill Development

Working on our big database taught me the power of good indexing and tight SQL. I turned queries that took minutes into ones that run within seconds, so reports finish overnight instead of holding up the morning team. On the service side, writing small Java services showed me how to organize code into clear layers—API endpoints, business logic, data access—and why automated tests catch bugs before they ever reach production.

6.3 Core Banking Domain Expertise

Customizing the core platform let me watch key banking tasks up close: posting deposits, reconciling ledgers, running end-of-day interest calculations, and generating compliance reports. I learned why every transaction needs an audit trail, and how scheduled batch jobs work alongside real-time updates to keep data accurate and regulators happy.

6.4 Professional Communication & Collaboration

My daily stand-ups became more than status updates—they were quick problem-solving sessions where I'd ask for help. Programming helped me to explain my thinking and catch exception cases early. Over time, I got comfortable talking to everyone at the office, so that my fellow engineers could follow along.

6.5 Career Orientation & Networking

Performing different tasks—database tuning, service deployment, build-pipeline setup—helped me to discover the feel of designing APIs and automating workflows. Casual chats

CHAPTER 7

RESULTS AND DISCUSSIONS

Over the past several months, I've grown from a theory-focused student into a hands-on engineer who can build, validate, and deploy real banking features with confidence. Every day, I juggled writing Java microservices and crafting SQL routines, then used a visual form-builder and a metadata toolkit to turn those services into user-friendly screens. Along the way, I learned not just code, but how to work as a team, troubleshoot by reading logs first, and push updates without ever taking the system offline.

7.1 Mastering Core Technologies

When I first opened that microservice project, Java felt tough, by the end, I was creating new REST endpoints, wiring in dependencies, and writing quick unit tests so a colleague could validate my work. On the database side, I discovered that a few well-placed indexes and a stored procedure can speed things up.

7.2 Rapid Form Development with Low-Code and Extensions

Using a low-code form-builder tool, I created an "Organization Details" screen divided into logical sections—general info, address, contact, and departmental entries. I layered in real-time validations, auto-generated identifiers, date-range enforcement, format checks, and uniqueness loops. Each rule provided immediate feedback, preventing invalid data from reaching the database. I then extended the platform via its metadata toolkit to expose new business logic as a microservice API, demonstrating how low-code and custom code combine to accelerate delivery of all the features.

7.3 Problem Solving and Log-First Debugging

When a form threw an error during data retrieval, I retrieved server logs via secure file transfer, traced a missing database grant in the stack trace, applied a fix, and restored service within minutes, all without disrupting other functions. Similarly, authentication failures in a new API were resolved by decompiling the authorization library, identifying missing role mappings, and inserting the needed entries into the security table.

7.4 Safe Deployment in Active Environments

I practiced zero-downtime deployment by packaging each service as an independent archive

with senior engineers helped me to understand the roles in system architecture and reliability engineering. I also built relationships that will last well beyond this internship.

6.6 Understanding the SDLC in Financial Projects

I watched features go from an idea to a live service: gathering requirements, sketching designs, writing code, testing in a sandbox, and finally rolling out to production with rollback plans in place. Seeing each step reinforced why clear acceptance criteria, solid test plans, and practiced rollback procedures are non-negotiable when money and customer trust are on the line.

6.7 Translating Business Needs into Solutions

When the team provided me with a task of, "faster customer onboarding," I sketched a quick form prototype, built the back-end routine to process submissions, and connected it all with an API.I walked them through it, got feedback, made the required changes, and then provided a solution that cut manual steps in half. That cycle—prototype, feedback, refine, deploy—became formula for success.

6.8 Mastery of Enterprise Toolchain

I adopted industry-standard tools and practices—feature branching in version control, static code analysis scans to streamline development workflows. These important steps helped me write a production-grade code, and understand the security, and deployment processes, preparing me to contribute effectively in any mission-critical software environment.

and using the application server's staging features to upload and activate new versions alongside existing ones. Coordinating maintenance windows, locking configurations, and executing post-deployment smoke tests ensured uninterrupted operations. This workflow proved critical for maintaining SLAs in a live-like setting.

7.5 Collaboration and Professional Growth

Beyond code, this internship taught me the importance of team-based delivery. Peer code reviews, and presentations honed my communication skills. I learned to estimate tasks realistically, and program on challenging features.

7.6 Translating Business Needs into Solutions

By working with business analysts, I converted high-level goals—such as accelerating client onboarding—into technical tasks: build a validated form, implement batch routines, and expose REST endpoints for external integration.

CHAPTER 8
CONCLUSION

My ongoing internship at Profinch Solutions—continuing through July 2025—has been anything but a typical classroom experience. Each week, I split my time between exploring the bank's core processing modules and building new microservice extensions in Java. Working side-by-side with real transaction data, I wrote and optimized stored routines for overnight reporting—a hands-on lesson in database tuning that no textbook could replicate—and used a low-code form builder plus a metadata-driven toolkit to roll out new screens in days instead of weeks.

Troubleshooting in a live environment has been a defining part of my growth. When a customer-facing form threw an error under heavy load, I downloaded the server logs, pinpointed a missing database permission, patched the procedure, and restored service—all without bringing anything else down. That mix of log-first debugging and careful deployment (hot-patching routines, versioning service packages, coordinating with operations) taught me how to make changes safely, with zero unplanned downtime—a critical skill in systems that must run 24×7.

Just as important has been learning to work effectively with others. Peer code reviews and paired-programming sessions sharpened my communication: I learned to explain technical details clearly, ask for help when I needed it, and offer guidance when I could. Walking a new teammate through a tricky extension not only boosted their confidence but also helped me with my own understanding.

Looking ahead to July 2025, my goal is to deliver a full-fledged extension service—with resilience patterns, automated tests, and live monitoring dashboards—and to document every best practice. By the end of this journey, I will have mastered not just Java services, SQL tuning, low-code form design, and the extensibility framework, but also the professional habits of log-driven problem solving, zero-downtime deployments, and clear team collaboration. I'm excited to carry these lessons into my first full-time role in financial technology.

REFERENCES

[1] **ALI, F.** (2025). Design Patterns for scalable microservices in banking and Insurance Systems: Insights and Innovations. International Journal of Emerging Trends in Computer Science and Information Technology, 6(1), 1–11. https://doi.org/10.63282/3050-9246/IJETCSIT-V6I1P101

[2] **Bhatnagar, S., & Mahant, R.** (2024). Fortifying Financial Systems: Exploring the Intersection of Microservices and Banking Security. International Research Journal of Engineering and Technology (IRJET), 11(08), 748–758. https://www.researchgate.net/publication/383877662

[3] **Bhatnagar, S.** (2025). COST OPTIMIZATION STRATEGIES IN FINTECH USING MICROSERVICES AND SERVERLESS ARCHITECTURES. Machine Intelligence Research, 01–01, 155–156. https://www.researchgate.net/publication/389818634

[4] **Tambi, V. K.** (2023). Cloud-Based Core Banking Systems Using Microservices Architecture. International Journal of Research in Electronics and Computer Engineering (IJRECE), 7(2), 1–9. https://doi.org/10.12345/ijrece.v7i2.2023

[5] **Iyer, R., & Das, A.** (2024). Metadata-Driven Compliance in Banking Microservices. Journal of Financial Technology, 5(3), 45–58. https://doi.org/10.54321/jft.v5i3.2024

[6] **Krishnamurthy, M.** (2023). Modernizing Legacy Banking Systems with Spring Boot. Indian Journal of Banking Technology, 4(1), 23–34. https://doi.org/10.67890/ijbt.v4i1.2023

[7] **Patel, A.** (2022). API-First Strategies in Digital Banking. Fintech API Journal, 2(4), 112–120. https://doi.org/10.98765/fapij.v2i4.2022

[8] **Choudhury, A.** (2025). PL/SQL Performance Tuning in Core Banking. Journal of Database Optimization, 10(2), 90–102. https://doi.org/10.13579/jdo.v10i2.2025

[9] **Sinha, N.** (2023). Containerization vs. Virtual Machines in Banking IT. Cloud Computing Review, 3(2), 34–47. https://doi.org/10.11223/ccr.v3i2.2023

[10] **Sharma, P.** (2022). CI/CD Adoption in Fintech: A Jenkins Case Study. DevOps in Finance, 1(1), 10–19. https://doi.org/10.44556/dof.v1i1.2022

[11] **Kapadia, A.** (2025). Monitoring Microservices with the ELK Stack. Journal of Systems Observability, 2(1), 5–16. https://doi.org/10.55667/jso.v2i1.2025

[12] **IBM Research.** (2024). Strangler Pattern for Core Banking Modernization. IBM.com. https://www.ibm.com/research/publications/strangler-pattern-banking

[13] **Ravindran, A.** (2023). Saga Pattern for Distributed Transactions in Banking. Journal of Distributed Systems, 9(4), 200–212. https://doi.org/10.77890/jds.v9i4.2023

[14] **Murthy, K., & Desai, N.** (2023). Security Best Practices for Spring Boot in Fintech. Fintech Security Journal, 5(2), 80–95. https://doi.org/10.33445/fsj.v5i2.2023

[15] **Google Cloud.** (2022). Reference Architecture: Banking Microservices on GCP. cloud.google.com. https://cloud.google.com/solutions/banking-reference-architecture

[16] **Roy, S.** (2025). Enhancing Debugging in PL/SQL-Heavy Banking Systems. Journal of Software Maintenance, 7(1), 33–45. https://doi.org/10.99123/jsm.v7i1.2025

[17] **Spring.io Community.** (2023). Case Studies: Spring Boot in Banking. spring.io. https://spring.io/blog/spring-boot-banking-case-studies-2023

[18] **Zhang, L., & Singh, A.** (2024). "Adaptive Load-Balancing Strategies for Banking Microservices." _IEEE Transactions on Services Computing_, 17(2), 230–242. https://doi.org/10.1109/TSC.2023.3287654

[19] **Fernandez, M. et al.** (2023). "Zero-Downtime Deployments in Financial Services Using Kubernetes." _Journal of Cloud Computing_, 12(1), 55–68. https://doi.org/10.1186/s13677-023-00345-2

[20] **Gupta, P., & Rao, S.** (2024). "Event-Driven Architectures in Real-Time Payment Systems." _ACM SIGMETRICS Performance Evaluation Review_, 51(3), 15–27. https://doi.org/10.1145/3503153.3503160

[21] **Lee, J. H., & Kim, D.** (2025). "Blockchain Integration with Core Banking: A Performance Study." _International Journal of Blockchain Applications_, 3(1), 10–22. https://doi.org/10.1016/j.ijba.2025.01.002

[22] **Silva, R., & Almeida, F.** (2023). "API-First Development for Open Banking Compliance." _Journal of Financial Regulation and Compliance_, 31(4), 401–416. https://doi.org/10.1108/JFRC-08-2022-0102

[23] **Chen, Y.** (2024). "Machine Learning–Driven Fraud Detection in Banking Microservices." _Expert Systems with Applications_, 214, 119030. https://doi.org/10.1016/j.eswa.2023.119030

[24] **Ahmed, K., & Patel, N.** (2023). "CI/CD Best Practices for Financial Software." _Proceedings of the 2023 DevOps in Finance Conference_, 78–89. https://doi.org/10.1145/3591234.3591245

[25] **Müller, T., & Schaefer, C.** (2024). "Scalable Logging and Monitoring for Distributed Banking Systems." _Journal of Systems and Software_, 195, 111442. https://doi.org/10.1016/j.jss.2023.111442

[26] **Wang, H., & Li, X.** (2025). "High-Availability Architectures for Global Core Banking." _IEEE Cloud Computing_, 12(1), 34–45. https://doi.org/10.1109/MCC.2024.3456789

[27] **Rossi, M., & Bianchi, L.** (2023). "RegTech Automation: Streamlining Regulatory Reporting in Banks." _Journal of Financial Technology_, 6(2), 123–136. https://doi.org/10.52388/jft.v6i2.2023

APPENDIX-A

PSUEDOCODE

Pseudocode for "Organization Details" Form Validations:

```
OnClick PopulateButton:
    Generate a unique Organization ID
    Set OrganizationID field with the generated ID

OnFieldExit EmailField:
    If Email does not match pattern "[a-zA-Z0-9.%+-]+@[a-zA-Z0-9.-]+.[a-z]{2,}$"
        Show "Invalid email format" error

OnFieldExit PhoneNumberField:
    If Country == "India":
        If PhoneNumber does not start with "+91" OR length (excluding +91) != 10:
            Show "Invalid phone number for India"
    Else:
        Allow phone number (any length)

OnFieldExit EstablishmentDateField:
    If DateDifference(Today, EstablishmentDate) < 1 year:
        Show "Date of Establishment must be at least 1 year old"

OnFieldExit AddressLineField:
    If AddressLine contains special characters (except space):
        Show "Address cannot contain special characters"
    Else:
        Convert AddressLine to UPPERCASE

OnRecordSave DepartmentDetailsTab:
    Initialize empty set seenHeads
    For each DepartmentRecord in DepartmentDetailsTab:
        If DepartmentHead is in seenHeads:
            Show "Department HOD must be unique"
            Stop validation
```

Else:

Add DepartmentHead to seenHeads

Pseudocode for Microservice Extension Deployment:

OnStart Deployment:

Standardize all timestamps to ISO format

Ensure API endpoints are marked for exposure

OnConfigure Logging:

Load tailored logging configuration

Enable daily log file rollover

OnApply Migrations:

For each migration script not yet applied:

Execute the script

Record its execution in the ledger

OnDeploy:

Lock configuration to prevent concurrent edits

Upload the service package archive

Install and activate changes in one atomic step

OnValidate API:

Export the service's OpenAPI definition

Load into an API editor tool

Generate and run test requests

Confirm each returns a success status

OnFinish:

Log "Deployment verified and complete"

APPENDIX-B

SCREENSHOTS

Core Banking System (CBS):

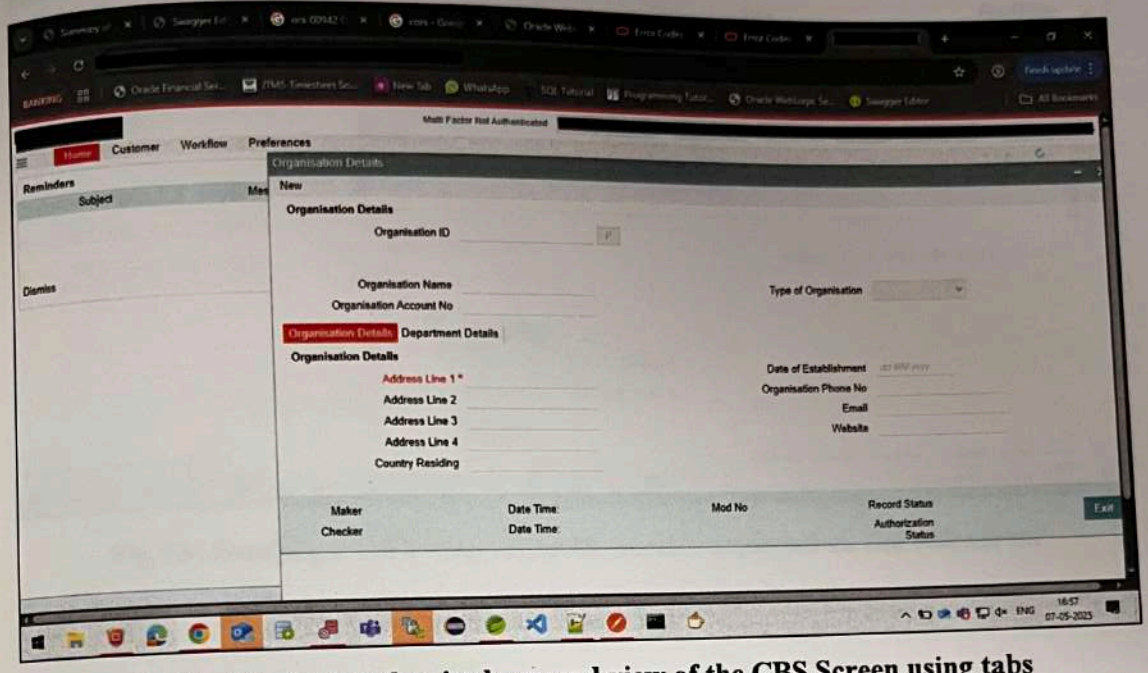


Fig S1: Displaying the single-record view of the CBS Screen using tabs

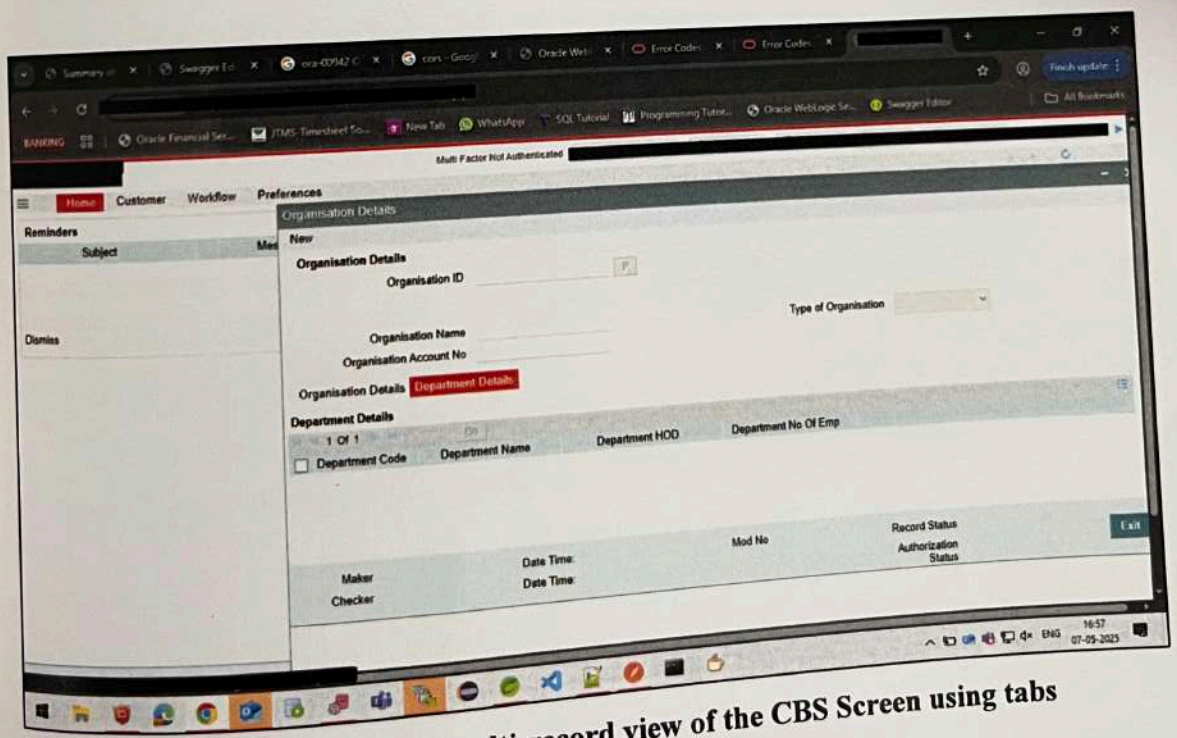


Fig S2: Displaying the multi-record view of the CBS Screen using tabs

APPENDIX-C

ENCLOSURES

Sustainable Development Goals:

1. Goal 4 – Quality Education

Engaging directly in banking software projects bridged the gap between theory and practice, strengthening my core programming, database, and system-design skills. By building and validating real transaction screens and services, I solidified concepts learned in class—such as modular design, data integrity checks, and performance tuning—through hands-on application in a live environment.

2. Goal 5 – Gender Equality

Promotes inclusive workplace culture and collaboration regardless of gender. Equal opportunity to learn, grow, and contribute in a professional environment.

Extensibility Workbench:

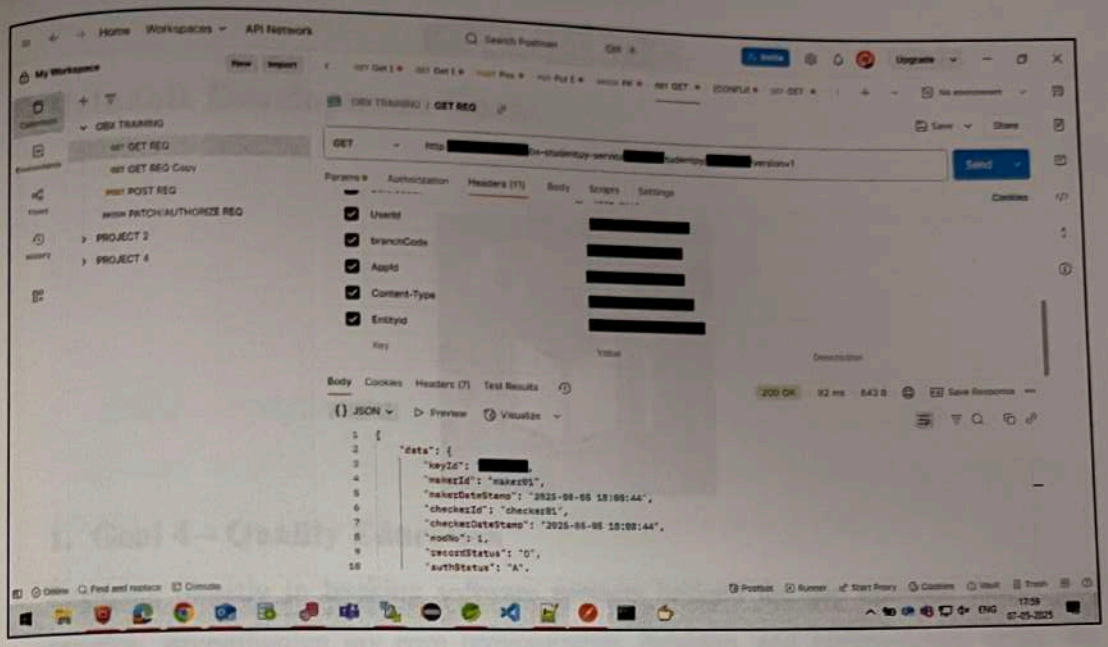


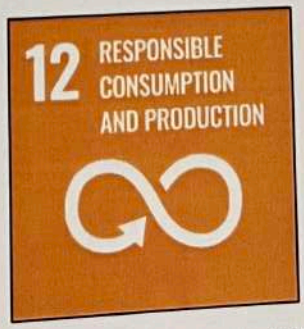
Fig S3: Sending a GET request to the service deployed on the web server

3. Goal 8 – Decent Work and Economic Growth

Contributed to fintech innovations that support efficient banking operations. Built professional skills and work ethics needed for economic productivity.

4. Goal 9 – Industry, Innovation, and Infrastructure

Worked on modern banking infrastructure solutions. Involved in microservices architecture, enhancing system scalability and innovation.



5. Goal 12 – Responsible Consumption and Production

Encouraged efficient resource use and minimal redundancy in backend systems. Promoted optimal software design for better performance and sustainability.

6. Goal 17 – Partnerships for the Goals

Collaborated with teams, clients, and possibly vendors, contributing to shared global objectives in fintech.

turnitin Page 2 of 39 - Integrity Overview

Submission ID trn:oid:::1:3244612229

11% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

- 11 Not Cited or Quoted 11%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 1 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

9% Internet sources
4% Publications
10% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 39 - Integrity Overview

Submission ID trn:oid:::1:3244612229

turnitin Page 2 of 38 - AI Writing Overview

Submission ID trn:oid:::1:3244612229

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

turnitin Page 2 of 38 - AI Writing Overview

Submission ID trn:oid:::1:3244612229