A REPORT

ON

DEVELOPMENT OF CUSTOMIZED FEATURES IN A CORE BANKING SYSTEM – PROFINCH SOLUTIONS

Submitted by,

Mr. POKALA SIVA MANI REDDY - 20211CEI0118

Under the guidance of,

Dr. JOE ARUN RAJA

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER ENGINEERING (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

At

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

PRESIDENCY UNIVERSITY

BENGALURU

MAY 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship report **"Development of Customized features in a Core Banking System at PROFINCH SOLUTIONS”** being submitted by “POKALA SIVA MANI REDDY” bearing roll number “20211CEI0118” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Engineering (Artificial Intelligence & Machine Learning) is a bonafide work carried out under my supervision.

Dr. JOE ARUN RAJA
Associate Professor
School of CSE & IS
Bangalore - 560054

Dr. GOPAL KRISHNA SHYAM
HOD & Professor
School of CSE & IS
Bangalore - 560054

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

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

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

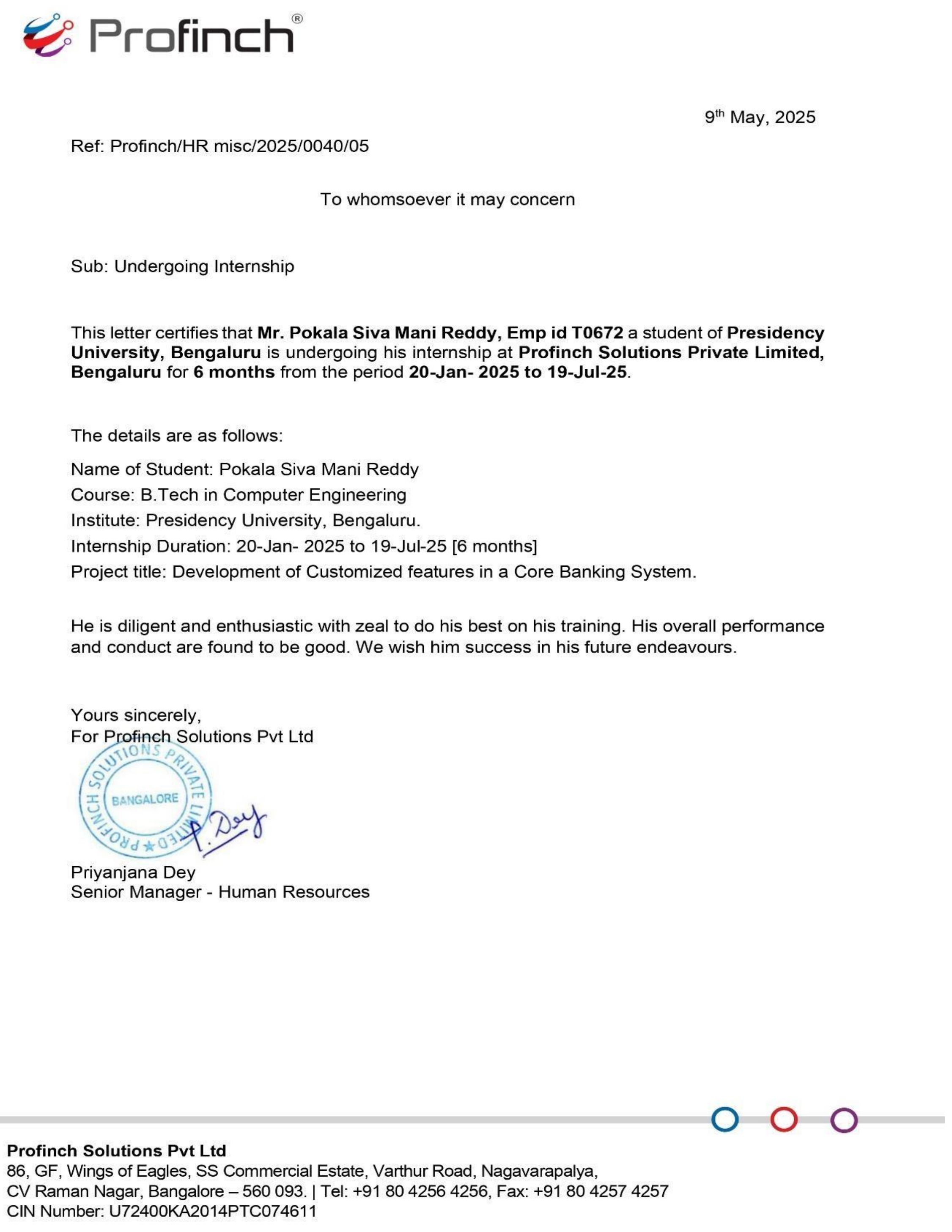
DECLARATION

I hereby declare that the work, which is being presented in the report entitled "**Development of Customized features in a Core Banking System at PROFINCH SOLUTIONS"** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Engineering (Artificial Intelligence & Machine Learning)**, is a record of my own investigations carried under the guidance of **Dr. Joe Arun Raja, Associate Professor, Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

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

POKALA SIVA MANI REDDY
20211CEI0118

INTERNSHIP IN-PROGRESS CERTIFICATE



Profinch®

9th May, 2025

Ref: Profinch/HR misc/2025/0040/05

To whomsoever it may concern

Sub: Undergoing Internship

This letter certifies that **Mr. Pokala Siva Mani Reddy, Emp id T0672** 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: Pokala Siva Mani Reddy

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

P. Dey

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

As part of my final semester internship, I am currently serving as a “**Internship in Development of Customized features in a Core Banking System – PROFINCH SOLUTIONS”**., a fintech firm specializing in delivering digital transformation solutions to banks and financial institutions. This role has allowed me to work closely with CBS (Core Banking System), where I actively engage in backend development and customization of core banking modules using PL/SQL. I am also gaining exposure to payment product processors, contributing to the development of microservices using Java Spring Boot, which supports scalable and modular financial applications. In addition, I have had the opportunity to work with an extendibility toolkit, enhancing productivity and accelerating development cycles. This internship is enriching my understanding of enterprise banking systems, reinforcing best practices in software development, and sharpening my analytical and collaboration skills. It offers a practical learning environment where I interact with real-world banking systems and contribute meaningfully to complex, enterprise-level fintech projects.

Throughout the internship, I have been involved in various stages of the software development lifecycle, including requirement analysis, design, coding, deployment, and collaboration with cross-functional teams. I have worked on customizing and enhancing core modules within the CBS to meet specific client requirements, involving extensive use of PL/SQL procedures, triggers, and performance optimization techniques. On the payment product processor side, I have contributed to building RESTful APIs and reusable components using Java Spring Boot, aligning with microservice architecture principles. I have also participated in code reviews, sprint planning, and version control using Git. This hands-on experience has not only strengthened my technical proficiency but also provided insight into compliance standards, security protocols, and integration challenges in large-scale financial systems.

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 - Engineering and Dean, Presiency School of Computer Science and Engineering & Presiency School of Information Science, Presidency University for getting us permission to undergo the project.

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

We are greatly indebted to our guide **Dr. Joe Arun Raja, Associate Professor**, Presidency School of Computer Science and Engineering, Presidency University for his/her inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the CSE7301 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K,** department Project Coordinators 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.

Pokala Siva Mani Reddy

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Profinch Solutions

Profinch Solutions is a fintech company dedicated to providing transformative banking solutions by leveraging advanced technology platforms. Operating under a licensed technology partnership, Profinch focuses on implementing and customizing enterprise-grade financial systems, including core banking systems (CBS) and digital banking components. The company combines domain expertise with innovative tools such as extensibility toolkits, microservices, and modern development frameworks to deliver tailored, scalable, and regulatory-compliant solutions. With a customer-centric approach and strong technical capabilities, Profinch drives digital excellence for banks and financial institutions globally.

1.2 Core Banking System (CBS)

A Core Banking System (CBS) is a comprehensive and integrated solution designed to streamline and modernize banking operations. It provides real-time processing capabilities across deposits, loans, payments, and other financial services. By leveraging CBS, financial institutions can enhance customer experience, reduce operational costs, and accelerate digital transformation. CBS solutions are built to support scalability, regulatory compliance, and seamless integration with external banking channels, making them a preferred choice for modern banks seeking efficient and future-ready core platforms.

1.3 Enterprise Database Systems

Enterprise-grade relational database management systems (RDBMS) are central to the infrastructure of financial technology solutions. Known for high performance, robust security features, and strong scalability, these systems support mission-critical applications. Their capabilities in handling complex transactions, backup and recovery processes, and optimized data queries make them ideal for financial environments. Innovations such as self-managing databases and support for multi-tenant deployments further enhance their ability to meet evolving business needs.

1.4 Integration and Middleware in Financial Systems

In a modern financial technology environment, seamless integration between various banking systems is critical. Middleware platforms play a key role in enabling communication across different services such as core banking systems, payment processors, and digital banking channels. These platforms handle data transformation, message routing, and protocol mediation to ensure consistent and real-time operations. Middleware also supports scalability by decoupling system components and enabling modular development through service-oriented or microservices-based architectures. Reliable middleware ensures secure data exchange, performance optimization, and flexibility in deploying new features across the banking ecosystem.

1.5 Financial Technology Components: CBS, Payment Processors, and Extensibility Tools

Modern financial service applications comprise a range of components, including core banking systems (CBS), payment product processors, and digital banking interfaces. These platforms enable omnichannel banking experiences and operational agility. Fintech solutions often extend their capabilities using microservices architectures, extensibility toolkits, and modern development frameworks to ensure modularity and rapid deployment. Development and customization activities rely on structured programming, middleware platforms, and secure file transfer and configuration tools to maintain performance and security standards.

1.6 Working in a Licensed Financial Technology Environment

As a licensed provider within a regulated technology ecosystem, we develop financial solutions that align with industry compliance and performance standards. This relationship ensures security, architectural consistency, and access to continuous improvements and expert support. It enables us to deliver robust, high-quality banking technology that adapts to both regulatory changes and market needs.

1.7 Leveraging Core Banking Systems in Fintech Environments

The Core Banking System (CBS) forms the backbone of our banking platforms, enabling secure, scalable, and compliant services. Through integration with payment processors, extensibility toolkits, and modern frameworks like microservices, we deliver flexible and innovative solutions. These platforms support rapid adaptation to changing customer needs and market conditions while maintaining stability and performance.

CHAPTER 2

LITERATURE SURVEY

2.1 Anjali R. Nair (2024) investigates the integration of core banking systems (CBS) with modern digital banking layers to enable seamless omnichannel services. The study outlines how digital banking interfaces communicate with CBS platforms using REST APIs and middleware services to deliver consistent customer experiences across mobile, web, and branch channels. The research highlights improvements in transactional throughput, reduced system downtime, and enhanced user satisfaction. The author also examines security models, role-based access, and session management frameworks to ensure compliance with industry standards such as PCI-DSS and ISO 27001. The paper concludes that layered banking architectures are effective for scalable and secure digital transformation.

2.2 Rajeev Kumar and Meenal Deshpande (2023) provide insights into the role of middleware platforms in enhancing the performance and fault tolerance of enterprise financial applications. The authors assess the deployment of clustered environments and how they facilitate load balancing, failover, and session persistence in core banking services. The paper further analyzes the use of enterprise components such as messaging queues and business logic containers to support high-throughput banking operations. Performance benchmarks reveal significant gains in latency reduction and transaction integrity under stress conditions. The study emphasizes that effective middleware design is key to achieving high availability in mission-critical financial systems.

2.3 Sneha Murthy (2025) explores the use of extensibility toolkits and procedural programming for rapid customization of banking workflows in licensed fintech environments. The research presents case studies where business logic for processes such as loan origination, customer onboarding, and compliance checks was developed using stored procedures and toolkit-generated components. Findings show that these tools significantly reduce development timelines while maintaining accuracy and auditability. The paper also discusses integration with version control, deployment automation, and configuration management using secure file transfer and build tools. The author concludes that toolkit-based development ensures fast and secure rollout of financial products when aligned with regulatory and quality standards.

CHAPTER 3
RESEARCH GAPS OF EXISTING METHODS

Despite advancements and successful deployments of core banking systems (CBS) and digital banking interfaces, several research gaps remain unaddressed in current implementations and academic studies. Firstly, while integration between digital banking platforms and CBS through middleware and APIs is widely practiced, there is limited research on real-time performance benchmarking and latency optimization in high-volume transaction environments. Most existing studies emphasize architectural design and functional capabilities but often lack quantitative data on user response times and throughput under stress conditions.

Secondly, although extensibility toolkits and procedural programming techniques have proven effective for rapid customization, there is a noticeable gap in the availability of automated testing frameworks and CI/CD compatibility for toolkit-generated components. Current practices still rely heavily on manual testing and deployment, which hinders agility in fintech environments where frequent updates and iterative releases are essential.

Lastly, while microservices and container-based architectures have gained traction for scalability, integration patterns between legacy CBS modules and containerized services remain underdeveloped. There is minimal literature addressing best practices for hybrid deployments—where monolithic core systems must coexist and communicate seamlessly with modern microservices-based modules. Bridging these integration and modernization challenges is vital to achieving comprehensive digital transformation across today’s fintech ecosystems.

CHAPTER 4
PROPOSED MOTHODOLOGY

4.1 Requirement Analysis

In this stage, a comprehensive analysis of the client’s current banking operations is performed. This includes gathering requirements from stakeholders to understand their business processes, systems, and challenges. The main objective is to identify the key functionalities needed for the digital transformation, such as customer account management, transaction processing, loan management, and ensuring regulatory compliance. The analysis also includes understanding the integration needs with existing systems to ensure smooth data exchange and improve the overall customer experience.

Table 4.1: Key Requirement Areas

Requirement Area	**Description**
Account Management	Managing customer accounts, balances, transactions, etc.
Payment Processing	Integration of various payment gateways for smooth operations.
Loan Management	Automating loan approvals, disbursements, and tracking.
Regulatory Compliance	Ensuring the system adheres to the latest regulations.
Digital Transformation	Supporting omnichannel banking services (web, mobile, etc.).

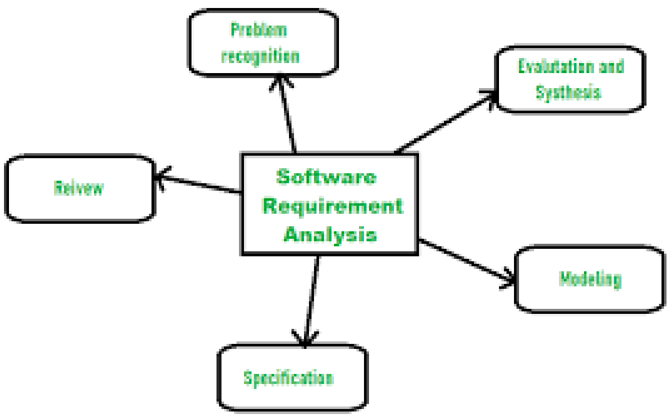


Figure 4.1: Requirement Gathering Process

4.2 System Design and Architecture

The system design is based on the requirements gathered in the previous step, aiming to build an architecture that integrates a core banking system (CBS) for core banking operations, a comprehensive banking service platform for full-spectrum financial functionality, and a digital banking interface for omnichannel service delivery. The system will be developed using a microservices architecture to ensure modularity, flexibility, and scalability. This architecture will be designed to handle various banking services while ensuring seamless communication between the system components, with REST APIs facilitating integration.

Table 4.2: System Design Components

Component	**Description**
Digital Banking Layer	Customizable platform for delivering digital banking services.
Core Banking System	Handles banking transactions, accounts, and customer data.
Microservices Backend	Developed using modern frameworks for flexibility and scalability.
Database Integration	Utilizes an enterprise-grade relational database for data storage and retrieval.
Middleware Platform	Ensures smooth communication between the system components.

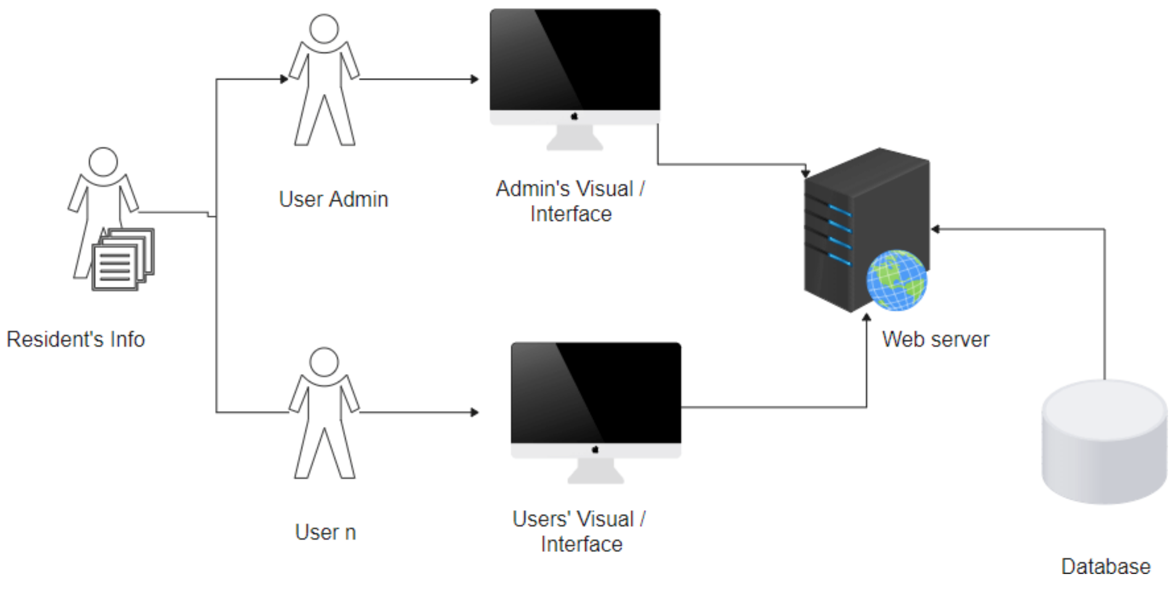


Figure 4.2: System Architecture Design

4.3 API Development and Testing Using Postman

APIs are crucial for ensuring that different modules and services within the system communicate effectively. Postman will be used for testing and validating these APIs, ensuring they meet functional, performance, and security requirements. The API testing includes verifying correct responses, validating error handling, ensuring that APIs perform efficiently under load, and confirming that they adhere to security protocols.

Table 4.3: API Testing Criteria Using Postman

API Testing Type	Description
Functional Testing	Verifying that the API meets the required functional behavior.
Performance Testing	Ensuring that the API performs well under load (speed, concurrency).
Security Testing	Checking for vulnerabilities such as SQL injection or cross-site scripting (XSS).
Response Validation	Ensuring that the API returns the expected responses based on input requests.

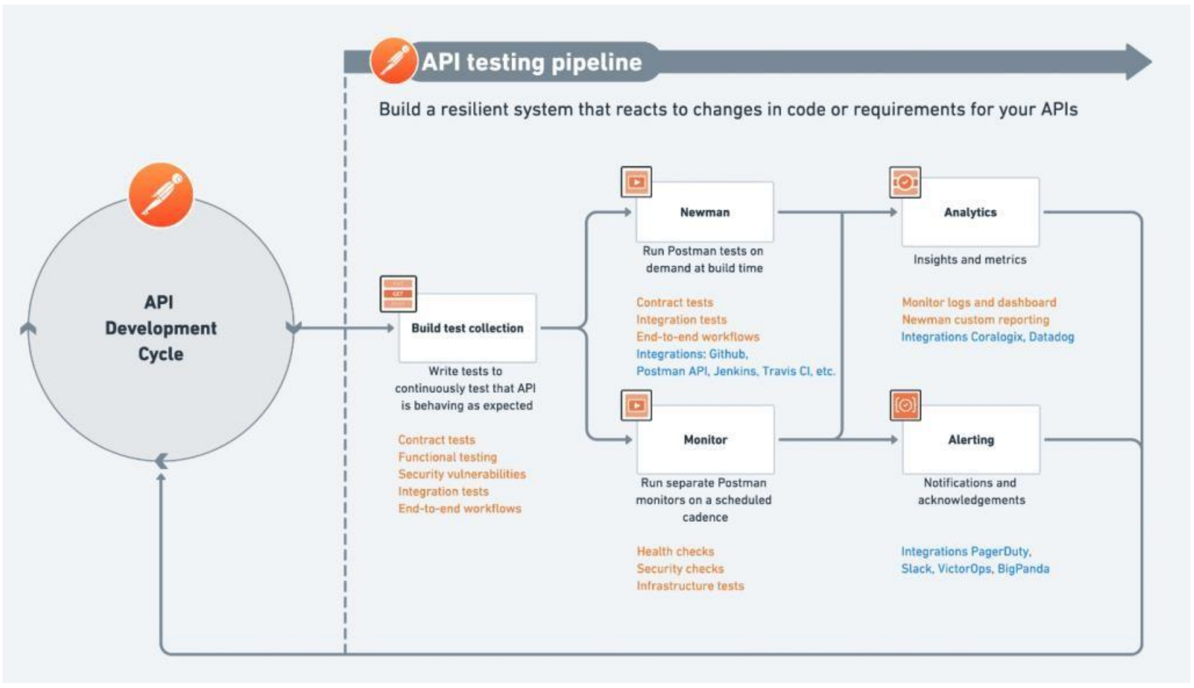

API testing pipeline
Build a resilient system that reacts to changes in code or requirements for your APIs
API Development Cycle
Build test collection
Write tests to continuously test that API is behaving as expected
Contract tests
Functional testing
Security vulnerabilities
Integration tests
End-to-end workflows
Newman
Run Postman tests on demand at build time
Contract tests
Integration tests
End-to-end workflows
Integrations: Github, Postman API, Jenkins, Travis CI, etc.
Monitor
Run separate Postman monitors on a scheduled cadence
Health checks
Security checks
Infrastructure tests
Analytics
Insights and metrics
Monitor logs and dashboard
Newman custom reporting
Integrations Coralogix, Datadog
Alerting
Notifications and acknowledgements
Integrations PagerDuty, Slack, VictorOps, BigPanda

Figure 4.3: API Testing with Postman

4.4 Development and Customization

4.4.1 Overview of the Development Stage

- The development and customization stage begins after the finalization of the system design blueprint.
- This phase involves translating planned architecture into fully functional digital banking applications.
- It defines how microservices and platform components will interact within a secure and scalable environment.

4.4.2 Microservices-Based Development

- Microservices are developed using Spring Boot for functionalities such as transaction handling, KYC, and reporting.
- Each service operates independently, enabling easier scalability, testing, and maintenance.
- Individual services can be updated without affecting the entire system.

4.4.3 Digital Banking Platform Customization

- The digital banking interface is tailored to meet client-specific needs across web and mobile platforms.
- Customization includes UI theming, dashboard personalization, localization, and branding adjustments.
- Components such as widgets and APIs are extended or reconfigured to enhance the overall user experience.

4.4.4 Application Development Using Extendibility Toolkit

- Extendibility toolkits are used to accelerate application development through low-code and configurable components.
- These toolkits support rapid prototyping and easy refinement based on evolving client needs.
- They are especially effective for building dashboards, reports, and administrative interfaces tailored to business processes.

4.4.5 Secure Configuration and File Management

- Secure file transfer tools are utilized to manage files between development and production environments.
- Configuration files—such as scripts, credentials, and certificates—are encrypted and tailored for each environment.
- Proper handling of these files ensures data security and adherence to regulatory compliance standards.

4.4.6 Middleware and Integration Management

- An enterprise-grade middleware platform is used to manage interactions between services and components.
- It facilitates service orchestration, load balancing, and centralized configuration management.
- The platform also supports secure communication and efficient session handling across the system.

4.4.7 Access Control and Security Policies

- Role-Based Access Control (RBAC) is implemented to define what each user can access or modify.
- Critical functions (e.g., transaction approval) are restricted to authorized personnel.
- Security features like audit logs and session policies are enforced.

4.4.8 Stakeholder Review and Feedback Loops

- Developed features are reviewed by stakeholders for validation.
- Feedback ensures the solution aligns with client expectations and regulatory standards.
- This process promotes continuous improvement and iterative refinement.

4.4.9 API Integration and Postman Usage

- Postman is used extensively for developing, testing, and validating RESTful APIs used across microservices.
- It supports sending requests like GET, POST, PUT, and DELETE to simulate frontend-backend interactions.

- Postman collections help maintain a version-controlled, shareable set of tests, enabling collaboration and debugging across teams.

4.4.10 Database Management

- An enterprise-grade relational database is used to securely store, manage, and query financial and customer data.
- Database procedures are implemented to automate complex business operations such as monthly reports, compliance checks, and account audits.
- Structured schemas are designed for modules like user management, transaction history, and KYC data to ensure data integrity and optimal performance.

4.4.11 Performance Optimization and Scalability

- Microservices are designed to scale horizontally by adding more service instances when load increases.
- Caching mechanisms and asynchronous calls improve response time and reduce database load.
- Performance metrics are collected to identify bottlenecks and optimize system throughput based on traffic and usage patterns.

Table 4.4: Breakdown of Development and Customization Tasks

Task Area	**Description**
Microservices Development	Using Spring Boot to build modular, independent services.
Digital Banking Customization	Adapting front-end and workflows to client-specific branding and requirements.
Application Development	Rapid prototyping and development with low-code extendibility toolkits.
Middleware Integration	Managing communication via enterprise-grade middleware platforms.
Secure File Transfer	Using secure file transfer tools to safely manage deployment files.
Configuration Management	Handling scripts and environment-specific settings.
RBAC Implementation	Ensuring secure user role access and audit capabilities.

4.5 Deployment

In the deployment stage, the system is transitioned to the production environment. Core services, such as those developed with microservices and enterprise-grade middleware platforms, will be deployed to ensure seamless access and availability. A comprehensive approach will be taken to ensure database integration, security protocols, and system configuration to support real-time transactions. Regular backups, system monitoring, and disaster recovery plans will be implemented to ensure high availability.

Table 4.5: Deployment Plan

Deployment Step	**Description**
Service Deployment	Deploying microservices and core banking services.
Database Integration	Integrating Database for real-time data management.
Security Setup	Configuring security measures such as data encryption and firewalls.
Monitoring Setup	Setting up performance monitoring tools for real-time tracking.

4.6 Post-Deployment Support and Maintenance

Once the system is deployed, it is critical to provide post-deployment support and maintenance to ensure optimal system performance. This includes regular updates, bug fixes, and enhancements. The system will be monitored to detect and resolve issues proactively. Additionally, security audits will be performed regularly to ensure that the system is secure against emerging threats.

Table 4.6: Post-Deployment Activities

Activity	**Description**
System Monitoring	Ensuring uptime and performance through continuous monitoring.
Regular Updates	Implementing necessary patches and updates based on user feedback.
Bug Fixes and Improvements	Addressing any system bugs and improving performance.
Security Audits	Conducting security audits to ensure the system remains secure.
Activity	Description

4.7 Benefits of the Proposed Methodology

The proposed methodology offers several advantages, such as scalability, flexibility, rapid development cycles, and enhanced customer experience. By leveraging modern tools like Spring Boot for microservices development and utilizing robust core banking and financial solutions, the system is designed to scale with the evolving business needs of financial institutions. Additionally, the use of modular design ensures that individual components can be upgraded without disrupting the entire system.

Table 4.7: Benefits of Proposed Methodology

Benefit	**Description**
Scalability	The system can grow in terms of user base and transaction volume.
Modularity	Components can be updated independently without affecting the entire system.
Fast Development	Use of low-code tools and modern frameworks results in quicker development cycles.
Regulatory Compliance	The system ensures adherence to financial and data security regulations.
Improved Customer Experience	The modular design provides a flexible and user-friendly experience.
Scalability	The system can grow in terms of user base and transaction volume.

4.8 Future Enhancements

The system is designed with future scalability in mind. Possible future enhancements include the incorporation of artificial intelligence (AI) for predictive analytics, the integration of chatbots for customer service, and the inclusion of blockchain technology to improve transaction transparency and security. The methodology is flexible, ensuring that emerging technologies can be integrated seamlessly.

Table 4.8: Future Enhancements

Enhancement	**Description**
AI Integration	Using AI for predictive analytics and improving decision-making.
Chatbot Integration	Enhancing the customer support experience with AI-powered chatbots.
Blockchain Integration	Incorporating blockchain for secure and transparent transactions.

CHAPTER 5

OBJECTIVES

5.1 To Understand Core Banking Solution

- Gain a thorough understanding of the functionalities and modules of Core Banking Solutions (CBS).
- Learn how CBS helps banks to manage their core operations, including account management, transactions, and reporting.
- Understand the integration of real-time services and customer management with CBS.

5.2 To Explore Database in Depth

- Understand the role of Database as the foundation for core banking systems, ensuring secure and scalable data management.
- Study database features, such as indexing, data recovery, and transaction management that enhance the performance of financial systems.
- Learn how Database handles large-scale, high-performance transaction systems within the banking sector.

5.3 To Learn Core Banking System Architecture for Financial Systems

- Investigate the multi-layered architecture of a core banking system, including components like memory structures, databases, and communication listeners.
- Understand how the architecture ensures high availability, performance, and disaster recovery in a banking environment.
- Gain insight into how the system architecture supports complex, transaction-heavy operations while maintaining scalability.

5.4 To Explore Core Banking Solutions and Financial Products

- Learn the functionalities of a digital banking interface platform for building seamless customer-facing solutions.
- Understand the role of a core banking system in backend banking operations, such as loan origination, deposits, and risk management.
- Explore how these systems interact with other tools, such as extendibility toolkits, and secure file transfer solutions, to streamline banking processes.

5.5 To Gain Expertise in Microservices Development for Financial Systems

- Learn to develop scalable microservices using Spring Boot, tailored for banking functions like transaction processing and account management.
- Understand how microservices in a core banking system can improve fault tolerance and agility.
- Explore how microservices integrate with existing systems for efficient data sharing and service orchestration.

5.6 To Understand the Role of APIs in Modern Banking

- Get familiar with how APIs enable interaction between different banking services, including payments, account management, and reporting.
- Learn how to design, develop, and test APIs that serve critical functions within a core banking system.
- Explore API security best practices to ensure data integrity and prevent unauthorized access.

5.7 To Master API Call Handling Using Postman

- Gain hands-on experience in using Postman for sending requests and validating responses in banking applications.
- Understand how to test RESTful APIs, handle headers, tokens, and response data for secure communication.
- Learn to automate API testing, including verifying the functionality and performance of critical services in core banking systems.

5.8 To Develop and Customize Digital Banking Platforms for Customer Channels

- Learn how to customize digital banking platforms for both web and mobile banking interfaces.
- Understand how to integrate digital banking platforms with other banking systems for seamless user experiences.
- Get familiar with how to use extensibility features of digital platforms to tailor the interface to different customer needs.

5.9 To Explore Core Banking Systems for Backend Banking Operations

- Gain expertise in using a core banking system for backend operations, such as loan management, account processing, and risk analysis.
- Understand how core banking systems integrate with other platforms to offer a complete banking solution.
- Learn about the flexibility of core banking systems for managing banking operations efficiently in a global context.

5.10 To Learn the Role of Extendibility Toolkits in Financial Systems

- Understand how rapid development tools accelerate the application development process.
- Explore how these tools enable quick prototyping, client feedback, and iterative development.
- Learn how to develop custom dashboards, reports, and interfaces using low-code platforms.

5.11 To Explore Middleware Solutions for Service Orchestration

- Learn how middleware solutions function for service orchestration and load balancing in financial systems.
- Understand the role of middleware in managing application sessions, security, and seamless service interactions.
- Gain experience in configuring and maintaining middleware solutions for enterprise-level fintech applications.

5.12 To Understand Secure File Management

- Learn how to use secure file transfer tools for file transfer between development and production environments.
- Understand how to encrypt configuration files, such as scripts, credentials, and certificates.

- Gain insights into how secure file management ensures compliance with regulatory standards and data protection laws.

5.13 To Implement Role-Based Access Control (RBAC) for Security

- Understand how RBAC works to limit access to sensitive functions based on user roles and responsibilities.
- Learn to implement RBAC in banking applications to ensure that only authorized personnel can perform critical tasks.
- Gain insights into how RBAC enhances security and compliance within a financial environment.

5.14 To Collaborate Effectively in a Fintech Team Environment

- Develop communication skills to interact with project stakeholders, team members, and clients.
- Learn how to present technical solutions to non-technical stakeholders and gather feedback for iterative development.
- Understand the importance of collaboration in ensuring successful project delivery and meeting business goals.

5.15 To Contribute to Business-Oriented Development in Fintech

- Understand how technical features align with business objectives, such as customer satisfaction, financial inclusivity, and compliance.
- Learn how to develop features that contribute to the overall goals of the organization, including increasing market reach and customer loyalty.
- Gain insights into how the development process supports business outcomes and regulatory compliance in the financial sector.

CHAPTER 6
SYSTEM DESIGN & IMPLEMENTATION

6.1 Overview of System Design

System design is the foundational phase in building any core banking solution, as it ensures all components work cohesively to meet both functional and non-functional requirements. In this stage, the system architecture, hardware, software, and network configurations are defined. For fintech solutions, the design primarily focuses on modularization, scalability, security, and seamless integration between microservices and other financial services.

Key elements of system design include:

- **Core banking functionalities:** Transactions, account management, loan processing, and reporting.
- **Scalability:** Ensuring the system can scale as the customer base grows.
- **Compliance and security:** Adhering to regulatory standards and ensuring data protection.
- **Integration with core banking systems:** Seamless integration of various banking modules and services.
- **API design:** Developing secure and efficient APIs for inter-service communication.

The goal is to create a flexible, secure, and scalable architecture that can handle the dynamic needs of digital banking while providing a seamless experience for users and administrators.

6.2 System Architecture

The architecture of the core banking solution is built using a microservices framework and is centered around a core banking system. The key architectural features are:

- **Microservices-based design:** Ensures modularity and independence of individual services, such as customer management, transactions, and loan services.
- **Service-Oriented Architecture (SOA):** Supports integration with external services, APIs, and third-party applications like payment gateways, mobile apps, and reporting tools.
- **Scalability and high availability:** Utilizing tools for container orchestration, ensuring the system can scale horizontally based on demand.

- **Data security and compliance:** The database system ensures secure data storage with built-in encryption and backup features. Role-based access control (RBAC) is applied to restrict sensitive data access.
- **API-driven communication:** RESTful APIs for seamless communication between microservices, ensuring real-time transactions and updates.

This architecture allows the solution to be adaptable to changing business needs while maintaining a high level of security and performance.

6.3 Functional Modules

In the system design, several key functional modules are identified to address core banking operations. These include:

- **Account Management:** Enabling customers to open, manage, and view account details through both web and mobile interfaces.
- **Transaction Processing:** Handling deposits, withdrawals, transfers, and other financial transactions. Each transaction is processed in real-time to ensure consistency and reliability.
- **Loan Management:** Provides functionalities like loan origination, processing, and reporting. The loan module is highly customizable for various loan types and terms.
- **Customer Authentication & KYC:** Facilitates secure login and onboarding of new users, implementing Know Your Customer (KYC) procedures to ensure compliance with regulatory requirements.
- **Reporting and Analytics:** Generation of real-time financial reports and insights for customers and administrators.

Each module is developed as an independent microservice, allowing for faster deployments and easier maintenance.

6.4 Implementation of Core Banking and Digital Banking Solutions

The core banking system and digital banking platform are integral parts of the system, ensuring the solution meets both customer-facing and backend operational requirements.

- Digital Banking Platform Implementation:
    - **Frontend Customization:** Tailoring the digital banking interface to provide an intuitive, responsive, and personalized user experience.
    - **Channel Integration:** The platform supports integration with multiple

    - channels, including mobile, web, and ATM systems, providing customers with a seamless omnichannel experience.
    - **Extensibility:** Utilizing an extensibility toolkit to customize platform features without modifying the core system.
- Core Banking System Implementation:
    - **Backend Integration:** The core banking system is integrated to handle all critical backend banking functions, including loans, deposits, and account management.
    - **Customization:** The core functions of the system are customized to meet the specific needs of the financial institution, such as loan approval workflows and reporting.

Both the digital banking platform and the core banking system contribute to creating a unified solution, offering flexible, scalable, and user-friendly banking services.

6.5 Database Design and Integration

The database design is critical for maintaining data integrity, security, and performance within a financial system. A robust relational database management system (RDBMS) is used as the primary database due to its scalability and reliability.

Key considerations in the database design include:

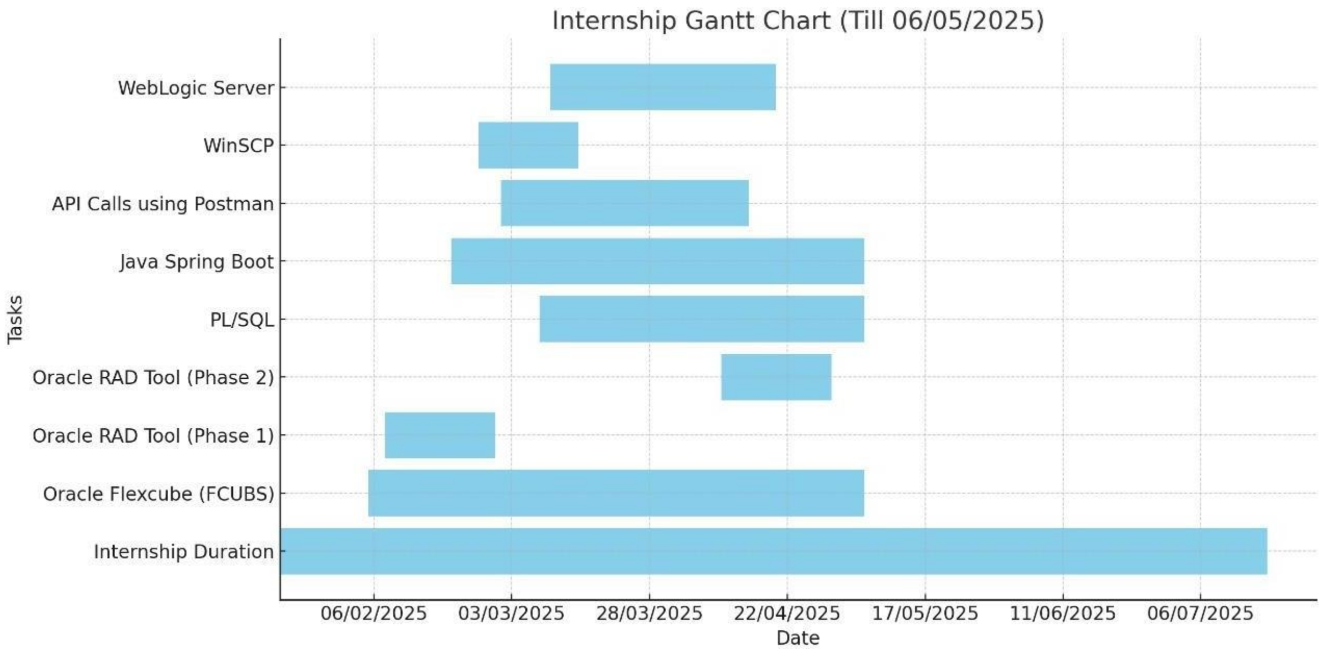
- **Data Modeling:** Designing tables, schemas, and relationships to represent customers, transactions, accounts, loans, and more.
- **Normalization:** Ensuring that the database is normalized to minimize data redundancy while ensuring efficient access and storage.
- **Data Security:** Implementing encryption for sensitive customer data and utilizing built-in security features like Transparent Data Encryption (TDE).
- **Backup and Recovery:** Setting up automated backup mechanisms to ensure data integrity in case of system failures.
- **Integration with Core Banking System (CBS):** The database integrates seamlessly with the core banking system, supporting real-time transaction processing.

6.6 API Design and Integration

APIs play a critical role in connecting different modules within the banking system, allowing communication between microservices and external systems.

- **API Development:** The APIs are built using REST architecture, ensuring that services can be accessed efficiently across different platforms.
- **API Security:** OAuth 2.0 authentication and encryption are implemented to secure data exchange between services and external clients.
- **Integration with Core Banking and External Systems:** APIs are developed to allow seamless communication between the core banking system and external services such as payment gateways and financial networks.
- **API Testing:** The APIs are tested using automated tools to ensure that they function as expected. Tests are conducted for critical operations, such as transaction processing and customer authentication.

CHAPTER-7
TIMELINE FOR EXECUTION OF INTERNSHIP
(GANTT CHART)



CHAPTER 8
OUTCOMES

- **Proficiency with Core Banking Solutions (CBS)**
  The internship provided hands-on experience with Core Banking Solutions (CBS), including modules for account management, transactions, and compliance. Interns gained practical knowledge in managing and implementing core banking systems, crucial for modern banking operations.
- **Deep Understanding of Database Management**
  Interns gained valuable experience working with relational databases, learning about data storage, retrieval, and management. The program provided exposure to SQL and PL/SQL, allowing participants to perform complex queries and database operations essential for banking applications.
- **Understanding of System Architecture and Integration**
  The internship offered an understanding of system architecture, focusing on how different components of a core banking solution interact to deliver robust and scalable services for financial institutions. This insight into system design and integration was crucial for developing effective fintech solutions.
- **Hands-On Experience with Financial Products**
  Interns gained expertise in using financial products that are critical in the fintech industry. They learned how to configure, implement, and support these systems for banking operations, ensuring a better understanding of financial product integration.
- **Application of Microservices in Financial Systems**
  Exposure to microservices and modern development practices helped interns understand scalable, modular, and maintainable software design in the fintech sector. This experience allowed them to contribute to the development of efficient financial applications using cutting-edge technologies.
- **Utilizing Extensibility Toolkits for Application Development**
  Interns learned how to work with extensibility toolkits, which enabled quicker customization and development of core banking solutions. By using these tools, interns

were able to streamline the development process, improving efficiency and reducing the time needed for application deployment.

- **Experience with Application Server Deployment**
  The internship provided exposure to using an enterprise application server for deploying and managing core banking applications in a production environment. This experience helped interns understand the complexities of enterprise-level deployment and management of financial applications.
- **Working with Secure File Transfers**
  Interns used secure file transfer tools for moving files between local systems and remote servers. This experience provided valuable insights into how secure file handling is integral to banking and fintech operations, ensuring data integrity and confidentiality.

CHAPTER 9
RESULTS AND DISCUSSIONS

- **Hands-On Experience with Core Banking Solutions**
  Interns gained practical exposure to Core Banking Solutions (CBS), learning the intricacies of core banking operations such as account management, transactions, and compliance. This experience emphasized the complexity and importance of robust banking software for large-scale financial institutions.
- **Enhanced Skills in Database Management and PL/SQL**
  Through interaction with relational databases, interns strengthened their proficiency in SQL and PL/SQL, managing critical banking data and writing complex queries. This hands-on experience highlighted the significance of data integrity and performance in banking systems.
- **In-Depth Understanding of System Architecture**
  Interns explored system architecture, learning how different components work together to provide scalable and secure banking solutions. This understanding of the infrastructure proved essential for developing and optimizing financial applications.
- **Experience with Modern Development Practices (Microservices & Spring Boot)**
  Exposure to a microservices-based architecture integrated with Spring Boot enabled interns to grasp modern development techniques. The focus on modular, microservices-based architectures underscored the need for scalability, flexibility, and ease of maintenance in fintech applications.
- **API Integration and Management**
  Interns gained hands-on experience with API integration, learning how to ensure seamless communication between core banking systems and third-party services. This experience highlighted the importance of effective API management in creating interconnected, real-time banking solutions.

CHAPTER 10
CONCLUSION

The internship provided an invaluable opportunity to gain practical, hands-on experience with Core Banking Solutions (CBS), which are widely used in the fintech sector. By working with core banking systems, interns developed a solid understanding of the fundamental operations that power these systems. The exposure to these solutions allowed for a deeper insight into the complexities of account management, transaction processing, and compliance handling. Additionally, the experience with relational databases and PL/SQL gave interns the tools to manage and manipulate financial data effectively, reinforcing the importance of robust data handling in banking applications.

Moreover, the internship offered exposure to enterprise-level architecture, providing a clear understanding of how different components work together to deliver scalable and secure banking solutions. Interns were also introduced to modern software development practices by working with microservices, Spring Boot, and low-code tools. The combination of microservices-based development with core banking solutions and the integration of technologies such as middleware solutions and secure file transfer tools further demonstrated the value of modularity, security, and efficiency in building fintech applications. These experiences have prepared interns for the complexities of real-world application development and deployment in the financial services industry.

Finally, the hands-on experience with API management and testing using Postman has enhanced interns' ability to ensure smooth communication between systems, a critical component of modern banking platforms. Understanding how to integrate APIs and manage them effectively has become increasingly essential in fintech, where third-party services and external integrations play a significant role. The overall exposure to Core Banking Solutions and their integration with other tools, combined with the opportunity to work within a licensed fintech environment, has provided a comprehensive foundation for future careers in banking technology and fintech development.

REFERENCES

[1] Core Banking Systems: Foundations of Modern Financial Services. (2025). Retrieved from https://example.com/core-banking-systems

[2] Comprehensive Overview of Core Banking Product Suites. (2025). Retrieved from https://example.com/core-banking-products

[3] Building Microservices with a Modern Java Framework. (2025). Retrieved from https://example.com/microservices-java-framework

[4] Understanding Toolkits for Accelerated Software Development. (2025). Retrieved from https://example.com/toolkits-software-development

[5] Enterprise Application Servers: Architecture and Deployment. (2025). Retrieved from https://example.com/enterprise-app-servers

[6] Secure File Transfer Clients for Development Environments. (2025). Retrieved from https://example.com/sftp-ftp-client

[7] Procedural SQL Techniques for Financial Applications. (2025). Retrieved from https://example.com/procedural-sql

[8] API Platforms for Modern Software Development. (2025). Retrieved from https://example.com/api-platforms

[9] High-Performance Databases for Financial Workloads. (2025). Retrieved from https://example.com/finance-databases

[10] Enterprise System Architecture for Financial Institutions. (2025). Retrieved from https://example.com/enterprise-finance-architecture

[11] Anjali R. Nair. (2024). *Integration of core banking systems with modern digital banking layers for omnichannel services*. The study highlights REST API and middleware communication, transactional improvements, and security frameworks for scalable digital transformation. Retrieved from https://example.com/digital-banking-architecture

[12] Rajeev Kumar and Meenal Deshpande. (2023). *Middleware platforms for enhancing performance and fault tolerance in enterprise financial applications*. The paper analyzes load balancing, session persistence, and enterprise component usage in clustered core banking environments. Retrieved from https://example.com/middleware-performance-fintech

[13] Sneha Murthy. (2025). *Extensibility toolkits and procedural programming for customized fintech workflows*. Case studies illustrate rapid development of banking processes using stored procedures and toolkits, emphasizing secure integration and compliance. Retrieved from https://example.com/fintech-toolkit-development

APPENDIX-A

PSUEDOCODE

Core Banking System (CBS) – Organization Maintenance Screen Implementation

The requirement was to develop a maintenance screen within the Core Banking System (CBS) using an extendibility toolkit, which includes both detailed and summary views. The purpose of this screen was to capture organization details from the user, validate the input, process the information, and store it in the database if the data was correctly formatted. The screen consisted of fields such as organization ID (auto-generated), organization name, type, establishment date, address, mobile number, email, primary account, and country. Additionally, a multi-record tab was included to capture department details within the organization.

Validation rules implemented:

1. Organization ID must be auto-generated upon clicking the “Populate” button.
2. Email address must follow a valid format.
3. Country code must validate phone numbers specific to India.
4. If the country is India, the mobile number must start with ‘+91’ and contain exactly 10 digits (excluding the region code).
5. For other countries, the phone number can be of any length.
6. The establishment date of the organization must be at least one year prior to the current date.
7. Address lines must not contain special characters.
8. Address lines should auto-capitalize when the user exits the input field.
9. Each department HOD must be unique in the multi-record tab.

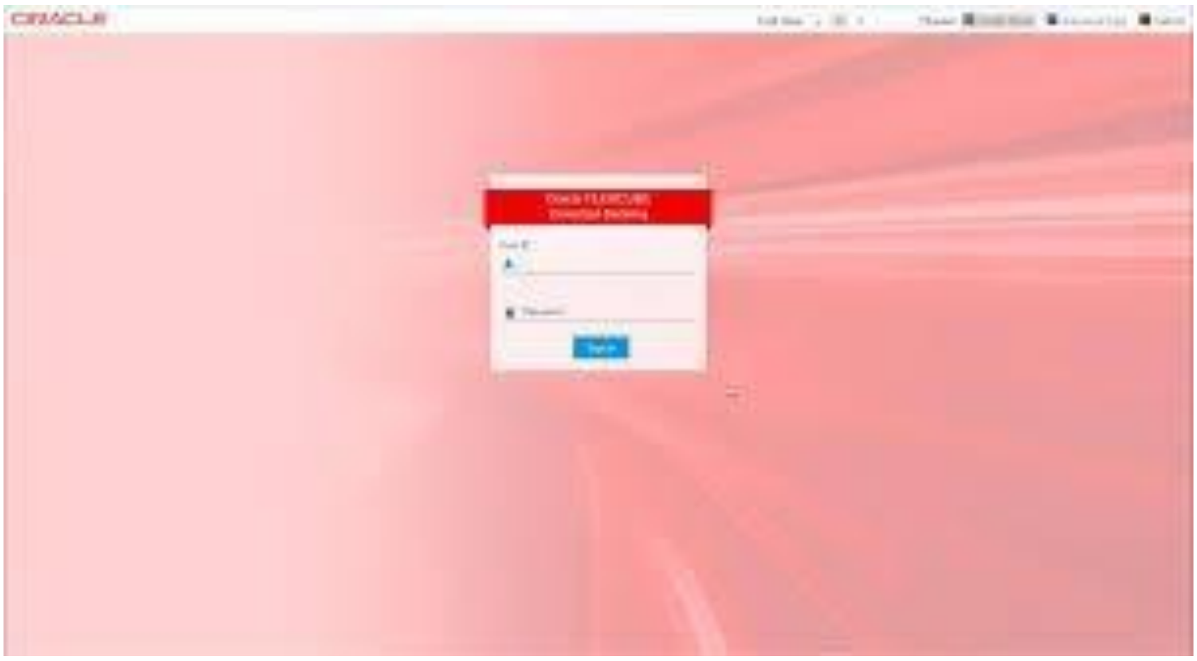
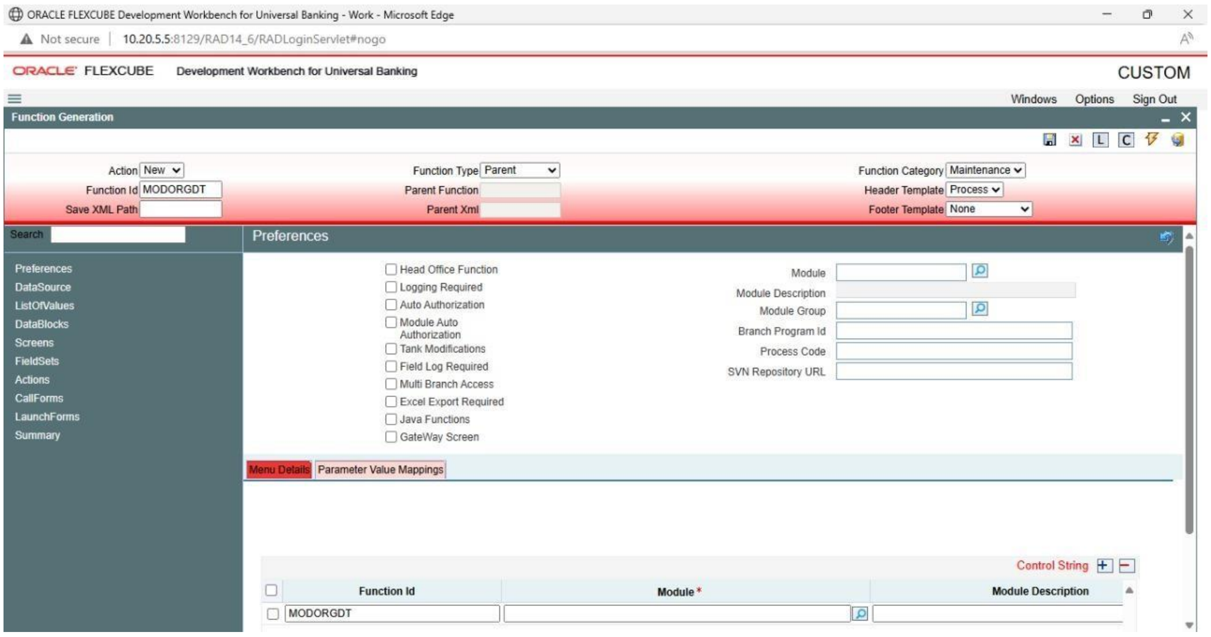


Figure A1: Custom-developed Core Banking System (CBS) UI screen for core banking operations



ORACLE FLEXCUBE Development Workbench for Universal Banking - Work - Microsoft Edge

Not secure | 10.20.5.5:8129/RAD14_6/RADLoginServlet#nogo

ORACLE FLEXCUBE Development Workbench for Universal Banking CUSTOM

Windows Options Sign Out

Function Generation

Action	New	Function Type	Parent	Function Category	Maintenance
Function Id	MODORGDT	Parent Function		Header Template	Process
Save XML Path		Parent Xml		Footer Template	None

Search

Preferences
DataSource
ListOfValues
DataBlocks
Screens
FieldSets
Actions
CallForms
LaunchForms
Summary

Preferences

- ☐ Head Office Function
- ☐ Logging Required
- ☐ Auto Authorization
- ☐ Module Auto Authorization
- ☐ Tank Modifications
- ☐ Field Log Required
- ☐ Multi Branch Access
- ☐ Excel Export Required
- ☐ Java Functions
- ☐ GateWay Screen

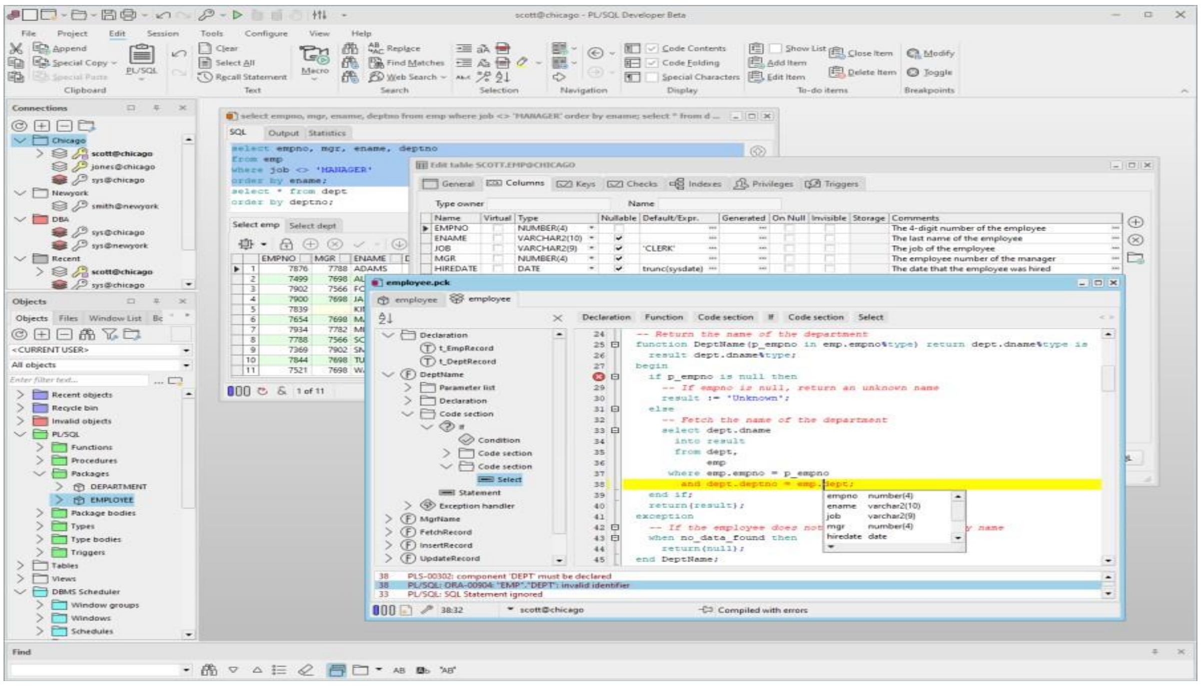
Module	
Module Description	
Module Group	
Branch Program Id	
Process Code	
SVN Repository URL	

Menu Details | Parameter Value Mappings

Control String

	Function Id	Module *	Module Description
☐	MODORGDT		

Figure A2: Custom core banking system screen developed using extendibility toolkit or enhanced UI functionality


scott@chicago - PL/SQL Developer Beta
File Project Edit Session Tools Configure View Help
Clipboard Text Search Selection Navigation Display To-do items Breakpoints
Connections
Objects
employee.pck
Declaration Function Code section If Code section Select
Find

```
select empno, mgr, ename, deptno
from emp
where job <> 'MANAGER'
order by ename;
select * from dept
order by deptno;
```

Select emp

	EMPNO	MGR	ENAME
1	7876	7788	ADAMS
2	7499	7698	AL
3	7902	7566	FC
4	7900	7698	JA
5	7839		KI
6	7654	7698	M
7	7934	7782	MI
8	7788	7566	SC
9	7369	7902	SM
10	7844	7698	TU
11	7521	7698	WA

1 of 11

Edit table SCOTT.EMP@CHICAGO — Columns

Name	Virtual	Type	Nullable	Default/Expr.	Generated	On Null	Invisible	Storage	Comments
EMPNO		NUMBER(4)							The 4-digit number of the employee
ENAME		VARCHAR2(10)	✓						The last name of the employee
JOB		VARCHAR2(9)	✓	'CLERK'					The job of the employee
MGR		NUMBER(4)	✓						The employee number of the manager
HIREDATE		DATE	✓	trunc(sysdate)					The date that the employee was hired

```
-- Return the name of the department
function DeptName(p_empno in emp.empno%type) return dept.dname%type is
  result dept.dname%type;
begin
  if p_empno is null then
    -- If empno is null, return an unknown name
    result := 'Unknown';
  else
    -- Fetch the name of the department
    select dept.dname
      into result
      from dept,
           emp
     where emp.empno = p_empno
       and dept.deptno = emp.dept;
  end if;
  return(result);
exception
  -- If the employee does not                          y name
  when no_data_found then
    return(null);
end DeptName;
```

```
empno     number(4)
ename     varchar2(10)
job       varchar2(9)
mgr       number(4)
hiredate  date
```

```
38   PLS-00302: component 'DEPT' must be declared
38   PL/SQL: ORA-00904: "EMP"."DEPT": invalid identifier
33   PL/SQL: SQL Statement ignored
```

38:32   scott@chicago   Compiled with errors

Figure A3: PL/SQL code implementation used for adding validations and functional enhancements in the core banking system (CBS)

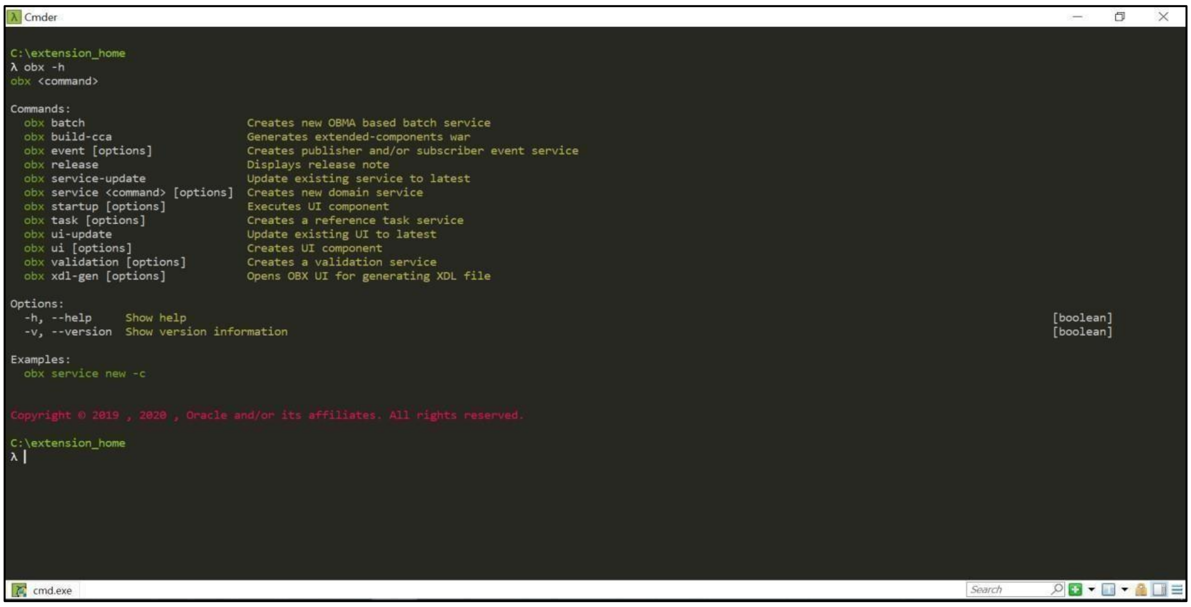
APPENDIX-B

SCREENSHOTS

Extendibility Toolkit Integration

The requirement was to develop a new student service using a payment product processor’s extendibility toolkit, which combines both graphical and command-line interfaces.

1. To begin, the command-line interface under the toolkit's root directory is accessed, and the command xdl-gen is executed. This starts the toolkit on port 8080, enabling the generation of the Extensibility Definition Language (XDL) file for the service.
2. Once the XDL file is generated, the command service mn -c is executed.
3. This triggers the creation of the required maintenance service. Appropriate configuration options are selected to meet service requirements.
4. Upon successful generation, a new folder is created in the toolkit’s root directory, named after the service.
5. Navigate into the service folder, open the command prompt at that location, and run gradle clean build. This generates a WAR (Web Application Archive) file, packaging the service for deployment.



```
C:\extension_home
λ obx -h
obx <command>

Commands:
  obx batch                      Creates new OBMA based batch service
  obx build-cca                  Generates extended-components war
  obx event [options]            Creates publisher and/or subscriber event service
  obx release                    Displays release note
  obx service-update             Update existing service to latest
  obx service <command> [options]  Creates new domain service
  obx startup [options]          Executes UI component
  obx task [options]             Creates a reference task service
  obx ui-update                  Update existing UI to latest
  obx ui [options]               Creates UI component
  obx validation [options]       Creates a validation service
  obx xdl-gen [options]          Opens OBX UI for generating XDL file

Options:
  -h, --help     Show help                                                   [boolean]
  -v, --version  Show version information                                    [boolean]

Examples:
  obx service new -c


Copyright © 2019 , 2020 , Oracle and/or its affiliates. All rights reserved.

C:\extension_home
λ |
```

Figure B1: Initiating the Extendibility Toolkit via Command Line Interface

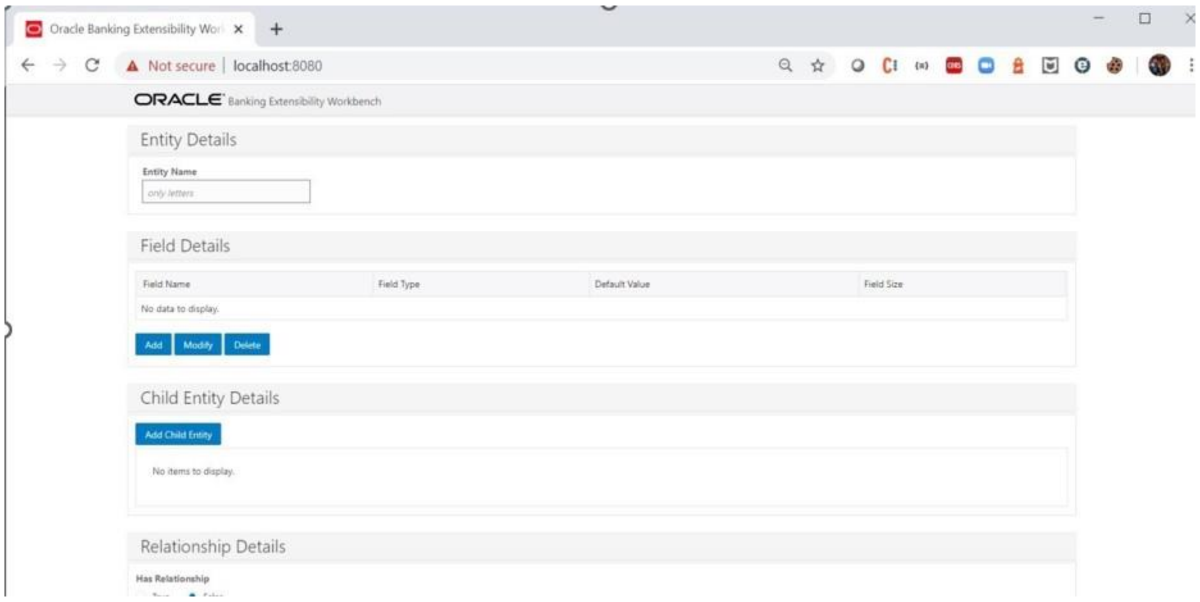
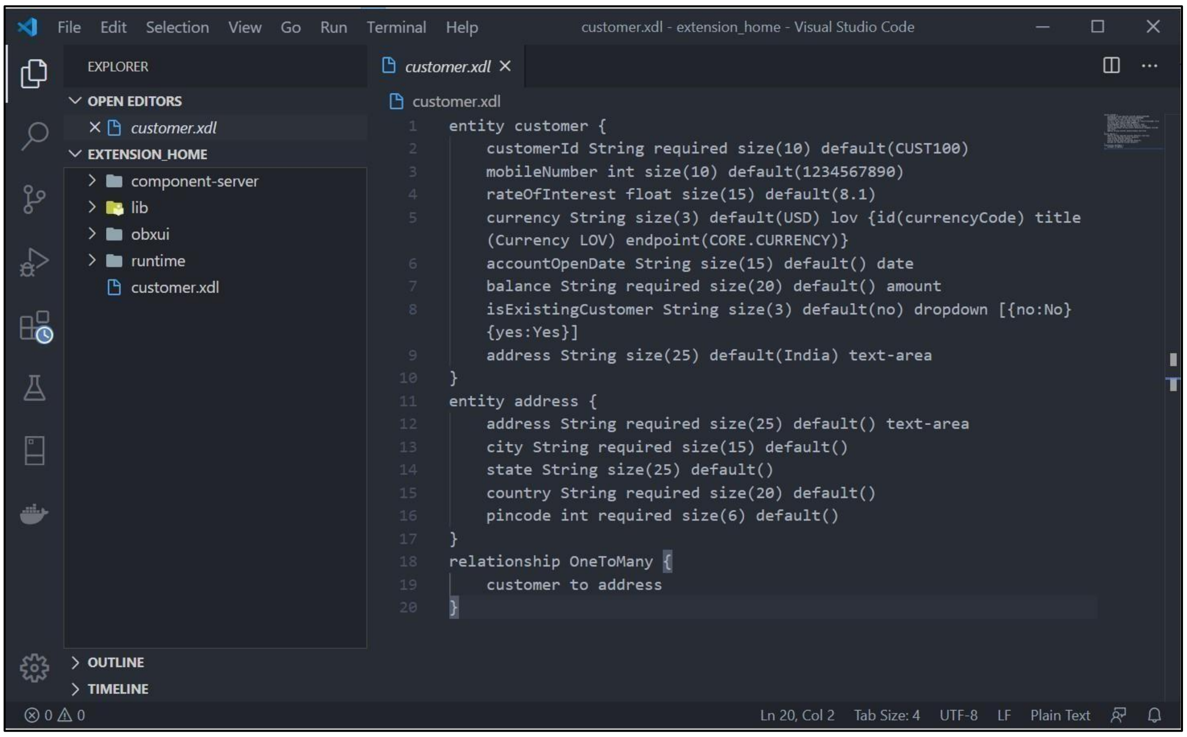
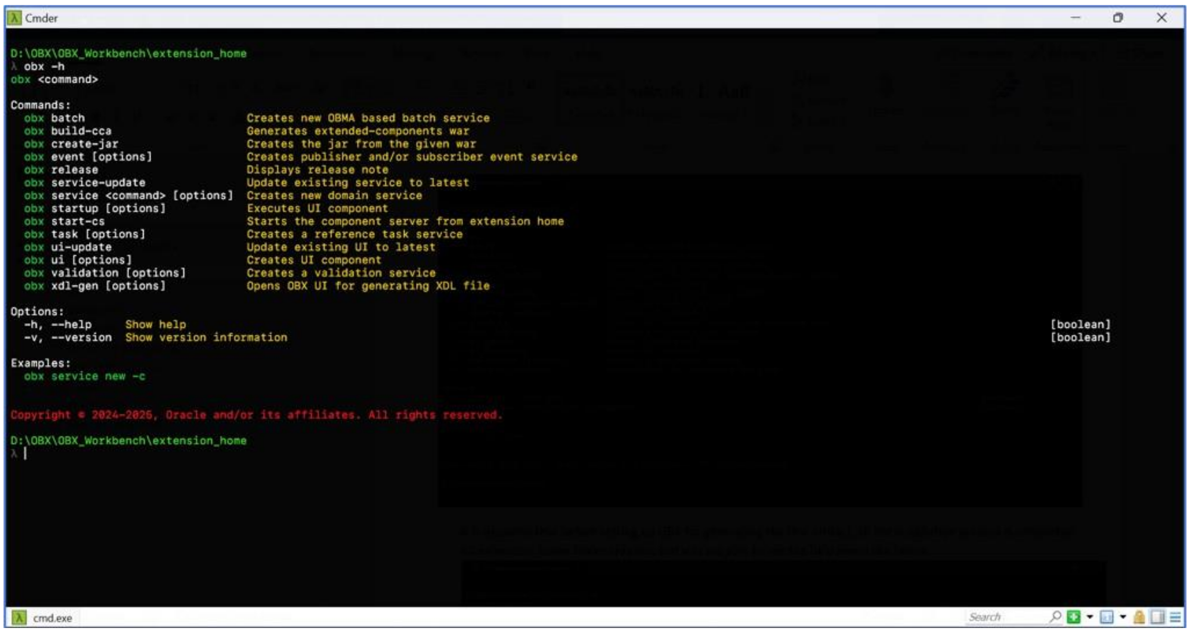


Figure B2: Extendibility Toolkit running locally on port 8080 for service generation



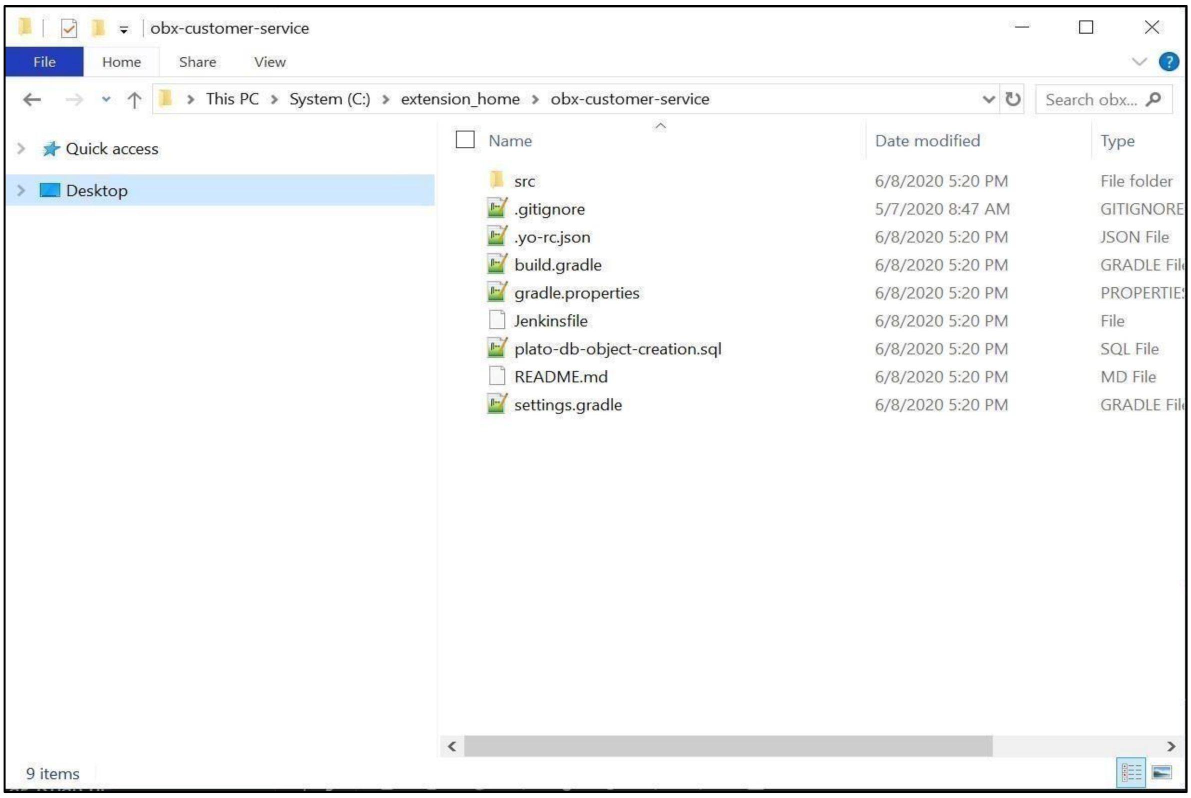
```
entity customer {
    customerId String required size(10) default(CUST100)
    mobileNumber int size(10) default(1234567890)
    rateOfInterest float size(15) default(8.1)
    currency String size(3) default(USD) lov {id(currencyCode) title
    (Currency LOV) endpoint(CORE.CURRENCY)}
    accountOpenDate String size(15) default() date
    balance String required size(20) default() amount
    isExistingCustomer String size(3) default(no) dropdown [{no:No}
    {yes:Yes}]
    address String size(25) default(India) text-area
}
entity address {
    address String required size(25) default() text-area
    city String required size(15) default()
    state String size(25) default()
    country String required size(20) default()
    pincode int required size(6) default()
}
relationship OneToMany {
    customer to address
}
```

Figure B3: Content of the XDL file for the new service



```
D:\OBX\OBX_Workbench\extension_home
λ obx -h
obx <command>

Commands:
  obx batch                          Creates new OBMA based batch service
  obx build-cca                      Generates extended-components war
  obx create-jar                     Creates the jar from the given war
  obx event [options]                Creates publisher and/or subscriber event service
  obx release                        Displays release note
  obx service-update                 Update existing service to latest
  obx service <command> [options]    Creates new domain service
  obx startup [options]              Executes UI component
  obx start-cs                       Starts the component server from extension home
  obx task [options]                 Creates a reference task service
  obx ui-update                      Update existing UI to latest
  obx ui [options]                   Creates UI component
  obx validation [options]           Creates a validation service
  obx xdl-gen [options]              Opens OBX UI for generating XDL file

Options:
  -h, --help     Show help                                                            [boolean]
  -v, --version  Show version information                                             [boolean]

Examples:
  obx service new -c

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

D:\OBX\OBX_Workbench\extension_home
λ |
```

Figure B4: Generating a maintenance service

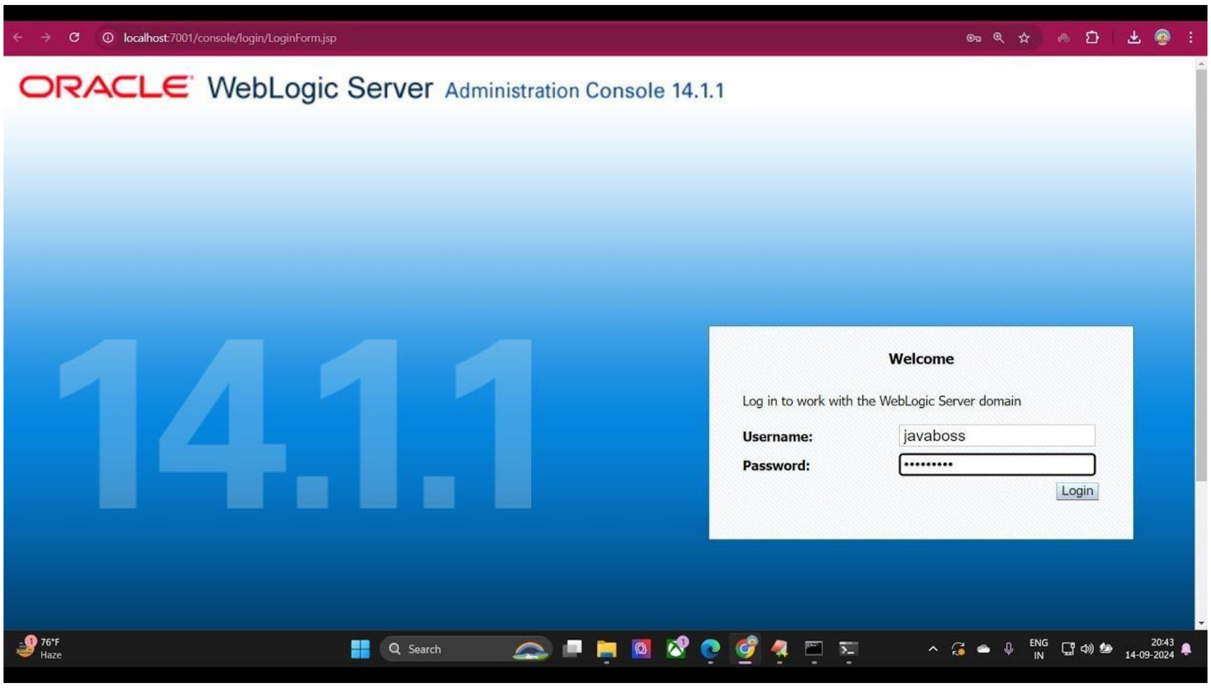


Name	Date modified	Type
src	6/8/2020 5:20 PM	File folder
.gitignore	5/7/2020 8:47 AM	GITIGNORE
.yo-rc.json	6/8/2020 5:20 PM	JSON File
build.gradle	6/8/2020 5:20 PM	GRADLE Fil
gradle.properties	6/8/2020 5:20 PM	PROPERTIE
Jenkinsfile	6/8/2020 5:20 PM	File
plato-db-object-creation.sql	6/8/2020 5:20 PM	SQL File
README.md	6/8/2020 5:20 PM	MD File
settings.gradle	6/8/2020 5:20 PM	GRADLE Fil

Figure B5: Service generated using the Extendibility Toolkit

The newly generated WAR (Web Application Archive) file contains all the necessary web components required to run and access the service. This includes entities, repositories, service layers, and controllers, all organized within the generated service folder by the extendibility toolkit. Additionally, an API specification document is included in YAML (Yet Another Markup Language) format. When the gradle clean build command is executed, it compiles all required components and packages them into the WAR file.

The base URL for the API endpoints is typically defined within the controller class. To explore the API structure, the YAML file can be uploaded to tools like Swagger Editor for visualization and testing.


localhost:7001/console/login/LoginForm.jsp
ORACLE WebLogic Server Administration Console 14.1.1
14.1.1
Login
76°F Haze
Search
ENG IN
20:43 14-09-2024

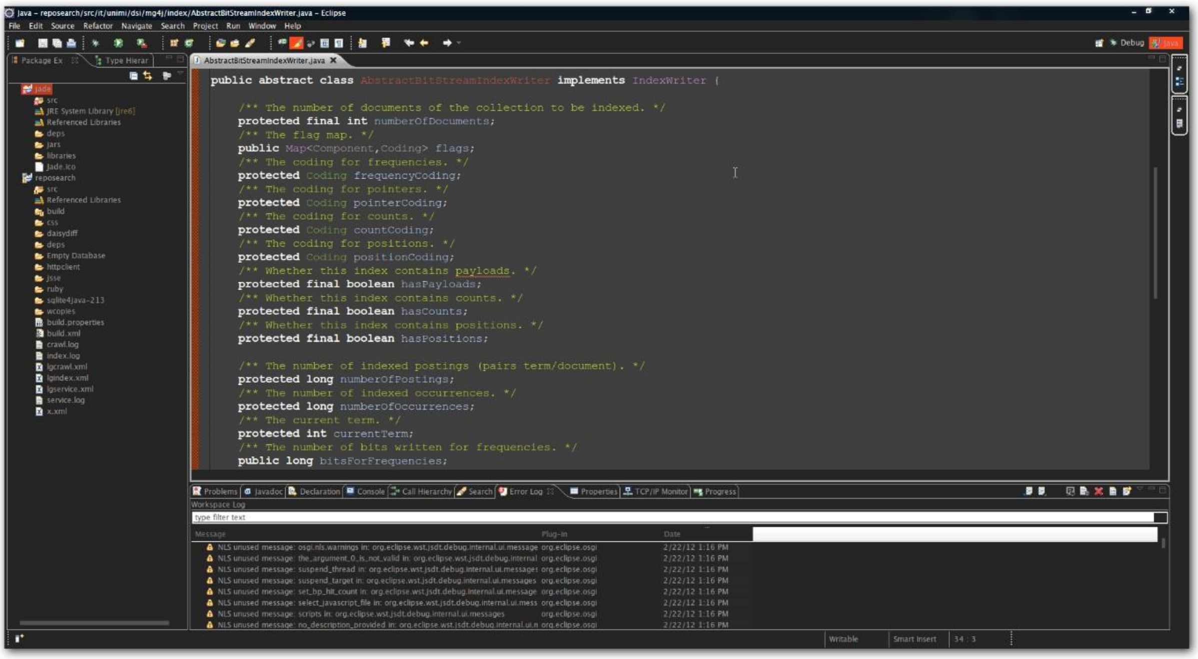
Welcome

Log in to work with the WebLogic Server domain

Username: javaboss

Password: •••••••••

Figure B6: Deployment file configured and deployed using an enterprise application server

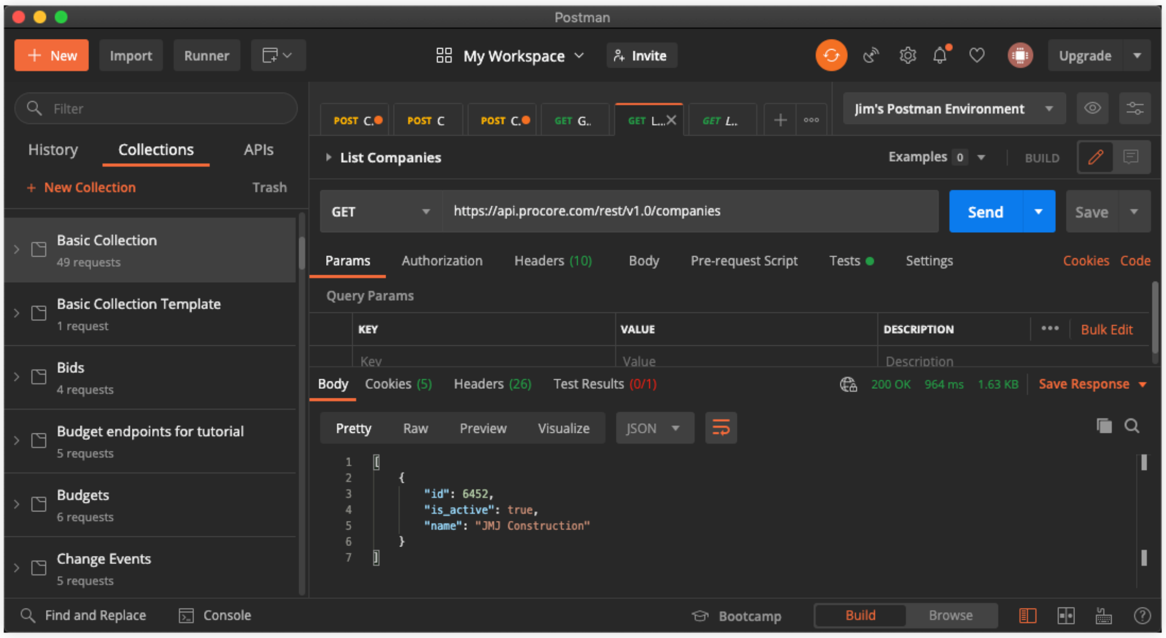


```
public abstract class AbstractBitStreamIndexWriter implements IndexWriter {

    /** The number of documents of the collection to be indexed. */
    protected final int numberOfDocuments;
    /** The flag map. */
    public Map<Component,Coding> flags;
    /** The coding for frequencies. */
    protected Coding frequencyCoding;
    /** The coding for pointers. */
    protected Coding pointerCoding;
    /** The coding for counts. */
    protected Coding countCoding;
    /** The coding for positions. */
    protected Coding positionCoding;
    /** Whether this index contains payloads. */
    protected final boolean hasPayloads;
    /** Whether this index contains counts. */
    protected final boolean hasCounts;
    /** Whether this index contains positions. */
    protected final boolean hasPositions;

    /** The number of indexed postings (pairs term/document). */
    protected long numberOfPostings;
    /** The number of indexed occurrences. */
    protected long numberOfOccurrences;
    /** The current term. */
    protected int currentTerm;
    /** The number of bits written for frequencies. */
    public long bitsForFrequencies;
```

```
Message                                                                                      Plug-in            Date
NLS unused message: osgi.nls.warnings in: org.eclipse.wst.jsdt.debug.internal.ui.message    org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: the_argument_0_is_not_valid in: org.eclipse.wst.jsdt.debug.internal     org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: suspend_thread in: org.eclipse.wst.jsdt.debug.internal.ui.messages      org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: suspend_target in: org.eclipse.wst.jsdt.debug.internal.ui.messages      org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: set_bp_hit_count in: org.eclipse.wst.jsdt.debug.internal.ui.message     org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: select_javascript_file in: org.eclipse.wst.jsdt.debug.internal.ui.mess  org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: scripts in: org.eclipse.wst.jsdt.debug.internal.ui.messages             org.eclipse.osgi   2/22/12 1:16 PM
NLS unused message: no_description_provided in: org.eclipse.wst.jsdt.debug.internal.ui.n    org.eclipse.osgi   2/22/12 1:16 PM
```

Java - reposearch/src/it/unimi/dsi/mg4j/index/AbstractBitStreamIndexWriter.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Ex
Type Hierar
AbstractBitStreamIndexWriter.java
Problems Javadoc Declaration Console Call Hierarchy Search Error Log Properties TCP/IP Monitor Progress
Workspace Log
type filter text
Writable
Smart Insert
34 : 3

Figure B7: Spring Boot application opened in Spring Tool Suite (STS) for code Modification



GET https://api.procore.com/rest/v1.0/companies

200 OK 964 ms 1.63 KB

```
[
    {
        "id": 6452,
        "is_active": true,
        "name": "JMJ Construction"
    }
]
```

Figure B8: API testing and validation performed using Postman

APPENDIX-C

ENCLOSURES

SUSTAINABLE DEVELOPMENT GOALS

SUSTAINABLE DEVELOPMENT GOALS
17 GOALS TO TRANSFORM OUR WORLD
1 NO POVERTY
2 ZERO HUNGER
3 GOOD HEALTH AND WELL-BEING
4 QUALITY EDUCATION
5 GENDER EQUALITY
6 CLEAN WATER AND SANITATION
7 AFFORDABLE AND CLEAN ENERGY
8 DECENT WORK AND ECONOMIC GROWTH
9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
10 REDUCED INEQUALITIES
11 SUSTAINABLE CITIES AND COMMUNITIES
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION
14 LIFE BELOW WATER
15 LIFE ON LAND
16 PEACE, JUSTICE AND STRONG INSTITUTIONS
17 PARTNERSHIPS FOR THE GOALS
SUSTAINABLE DEVELOPMENT GOALS

1. **Goal 4 – Quality Education**

Hands-on learning through real-world banking and fintech projects. Practical exposure to technologies like core banking systems, payment processors, and Spring Boot enriched your technical education.

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.

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 10 – Reduced Inequalities**

Supported backend systems that enable financial services to reach underserved communities through digital banking. Internships like this provide access to career opportunities for fresh graduates.

6. **Goal 11 – Sustainable Cities and Communities**

Improved access to financial services via digital platforms contributes to urban and rural development.

7. **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.

8. **Goal 17 – Partnerships for the Goals**

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

Dr.Joe Arun Raja -20211CEI0118 iNTERSHIP REPORT.pdf

ORIGINALITY REPORT

11%	7%	4%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Presidency University Student Paper	9%
2	Submitted to M S Ramaiah University of Applied Sciences Student Paper	1%
3	etd.uum.edu.my Internet Source	<1%
4	www.hipo.ro Internet Source	<1%
5	Submitted to University of Greenwich Student Paper	<1%
6	fastercapital.com Internet Source	<1%
7	Submitted to Da Vinci Institute Student Paper	<1%
8	Hamid, Brahim, and Jon Perez. "Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation", Journal of Systems and Software, 2016. Publication	<1%
9	Submitted to University of Keele Student Paper	<1%
10	ijsred.com Internet Source	<1%
11	www.docstoc.com	

Internet Source

<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		