A Industry Internship Report
on
Development of the TradeX Enterprise Trade Finance Platform
& Technical Bootcamp Modules

Submitted for the Fulfillment of the Requirements for the degree of Bachelor of Technology

in

Computer Science and Engineering

by

Ojaswa Varshney

Roll No.: UI22CS54

Under the guidance of

Dr. Kaustubh Dhondge

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

July, 2026

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY SURAT-394190

Indian Institute of Information Technology Surat
Computer Science and Engineering Department

CERTIFICATE

This is to certify that candidate **Ojaswa Varshney** bearing Roll No: **UI22CS54** of B.TECH. IV, 8th Semester has successfully carried out the work on "**Development of the TradeX Enterprise Trade Finance Platform & Technical Bootcamp Modules**" for the partial fulfillment of the degree of Bachelor of Technology (B.Tech.) in **April, 2026**.

Faculty Supervisor: Name: *Dr. Kaustubh Dhondge* Sign:.....

1. Examiner 1: Name:*Dr. Reema Patel* Sign:.....

(Seal of the Institute)

June 30, 2026

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr Ojaswa Varshney** has successfully completed the internship in **IDFC FIRST Bank Limited** in Information Technology function from **January 27, 2026 to July 10, 2026** under the guidance of a dedicated mentor.

During the internship, **Mr Ojaswa Varshney** worked on live projects and demonstrated commitment, professionalism and strong willingness to learn. The internship program is designed to provide hands-on experience through impactful projects and real-world learning opportunities.

We appreciate **Mr Ojaswa Varshney** for the contributions made and wish continued success in their future endeavors.

For **IDFC FIRST Bank Limited**

Jennifer Lobo
Head - HR Operations

DECLARATION

This is to certify that
(i) This report comprises my original work towards the degree of Bachelor of Technology in Computer Science and Engineering at Indian Institute of Information Technology (IIIT) Surat and has not been submitted elsewhere for a degree,
(ii) Due acknowledgement has been made in the text to all other material used.

Signature of Student
Ojaswa Varshney

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my Faculty Supervisor, Dr. Kaustubh Dhondge, Department of Computer Science and Engineering, Indian Institute of Information Technology Surat, for their continuous guidance, valuable suggestions, and constant encouragement throughout this internship. Their mentorship and academic support have been instrumental in shaping my understanding of enterprise software engineering practices.

I would also like to express my heartfelt gratitude to the Director of IIIT Surat, Dr. Rajeev Shorey, for his visionary leadership in fostering an environment that promotes academic excellence, innovation, and industry collaboration. I sincerely thank the Head of the Department, Dr. Ritesh Kumar, Department of Computer Science and Engineering, IIIT Surat, for providing an excellent academic environment. I am equally thankful to the Training and Placement Cell and the Training and Placement Officer, Ms. Jyoti Ranjan, for facilitating this internship opportunity and extending continuous support.

I am deeply grateful to my industry mentors and the entire engineering team at IDFC FIRST Bank for providing me with the opportunity to work on the TradeX platform. Their guidance, technical discussions, and collaborative environment greatly enhanced my technical skills and provided valuable exposure to large-scale, secure banking architectures.

I am also immensely grateful to the bootcamp instructors and the talent acquisition team at IDFC FIRST Bank for structuring such a comprehensive learning path. The intensive training during the initial phase provided an invaluable foundation in modern enterprise technologies, bridging the gap between academic learning and industry requirements.

Finally, I express my heartfelt gratitude to my parents, family, and friends for their unwavering love, encouragement, patience, and belief in my abilities. Their constant support and motivation have been my greatest source of strength throughout this six-month internship journey and have played an invaluable role in the successful completion of this work.

ABSTRACT

The modernization of financial technology requires banking systems to be highly secure, scalable, and resilient. This report presents the work carried out during a six-month industry internship at IDFC FIRST Bank, where I was assigned to the engineering team responsible for developing TradeX, an enterprise-grade trade finance platform. During the internship, I contributed to the development, security enhancement, and architectural scaling of the platform's backend infrastructure.

Unlike conventional banking applications, TradeX operates on a highly concurrent, distributed microservices architecture to manage complex financial instruments such as Letters of Credit (LCs) and Bank Guarantees (BGs). Understanding this architecture and its stringent security requirements formed a major part of the internship.

The internship combined structured technical upskilling with hands-on engineering contributions. The initial phase involved an intensive bootcamp on modern enterprise technologies including Golang, React, Docker, and Apache Kafka, culminating in the Agile development of a highly concurrent Movie Booking Application to simulate lifestyle banking integrations.

Further contributions focused on the TradeX platform, where work included implementing Zero-Trust authentication using ORY Hydra, designing fine-grained Relationship-Based Access Control (ReBAC) with Permify, and orchestrating strict Maker-Checker approval workflows. The internship also involved integrating global and domestic financial messaging protocols like SWIFT and SFMS, and developing an Event-Driven Architecture utilizing Apache Kafka for asynchronous processing and audit logging.

This report details the system architectures, implementation methodologies, engineering contributions, and key learnings gained during the internship, providing a comprehensive overview of modern, secure, and scalable banking infrastructure.

Contents

List of Principal Symbols and Acronyms

API	**A**pplication **P**rogramming **I**nterface
BG	**B**ank **G**uarantee
FSM	**F**inite **S**tate **M**achine
IdP	**Id**entity **P**rovider
JWT	**J**SON **W**eb **T**oken
LC	**L**etter of **C**redit
MT	**M**essage **T**ype (Legacy SWIFT Standard)
MX	**M**essage type **X**ML (ISO 20022 SWIFT Standard)
OIDC	**O**pen**ID** **C**onnect
OTP	**O**ne-**T**ime **P**assword
PKCE	**P**roof **K**ey for **C**ode **E**xchange
RBAC	**R**ole-**B**ased **A**ccess **C**ontrol
ReBAC	**R**elationship-**B**ased **A**ccess **C**ontrol
SBLC	**S**tand**b**y **L**etter of **C**redit
SFMS	**S**tructured **F**inancial **M**essaging **S**ystem
SWIFT	**S**ociety for **W**orldwide **I**nterbank **F**inancial **T**elecommunication
TDD	**T**est-**D**riven **D**evelopment

List of Figures

Chapter 1

Introduction to the TradeX System and Trade Finance

1.1 Background And Motivation

The modernization of financial technology requires banking systems to be highly secure, scalable, and resilient. During the internship at IDFC FIRST Bank, a major portion of the work was dedicated to the development and enhancement of TradeX, a comprehensive, enterprise-grade trade finance platform. Trade finance acts as the catalyst for international trade, mitigating the risks associated with cross-border transactions and providing necessary working capital to corporate clients.

Within this domain, traditional processes for handling Letters of Credit (LCs) and Bank Guarantees (BGs) are often paper-heavy, siloed, and prone to delays. The TradeX platform aims to digitalize this entire lifecycle.

1.2 Problem Statement

The global trade finance ecosystem suffers from significant operational friction due to decentralized, legacy infrastructure. Processing a single Letter of Credit involves multiple intermediaries, physical document verification, and fragmented communication, resulting in high latency, vulnerability to fraud, and poor client experience. The central engineering problem addressed during this internship is the design and implementation of a centralized, highly concurrent, and secure backend infrastructure capable of automating these workflows. Specifically, the challenge lies in digitizing the Maker-Checker authorization protocols, ensuring zero-trust identity verification, and scaling the ingestion and processing of thousands of real-time financial events without data loss or race conditions.

1.3 Scope and Contributions

This internship encompassed both foundational upskilling in enterprise architectures and direct contributions to production-scale applications. Key contributions include:

- Implementing robust concurrent booking engines in Golang to handle high-throughput transactions.
- Engineering secure, stateless authentication and authorization flows utilizing ORY Hydra and Permify.
- Developing asynchronous event-driven pipelines with Apache Kafka for audit logging and SWIFT message formatting.
- Utilizing Test-Driven Development (TDD) methodologies to achieve high test coverage across microservices.

1.4 Report Organisation

The remainder of this report is organized as follows:

- **Chapter 2:** Details the core tools and technologies used throughout the internship.
- **Chapter 3:** Presents the authentication security architecture using ORY Hydra.
- **Chapter 4:** Discusses fine-grained authorization implemented with Permify.
- **Chapter 5:** Describes the Maker-Checker approval workflows.
- **Chapter 6:** Details the integration with SWIFT and SFMS financial messaging.
- **Chapter 7:** Explores the event-driven architecture using Apache Kafka.
- **Chapter 8:** Outlines the technical bootcamp upskilling and the SkyFox Cinemas agile project.
- **Chapter 9:** Presents the conclusion and future scope.

1.5 Core Products Of TradeX

TradeX digitizes and automates the lifecycle of complex financial instruments. The platform is designed to handle multiple trade finance products, ensuring strict adherence to global regulatory and compliance standards.

1.5.1 Letters of Credit (LC)

A Letter of Credit is a foundational trade finance instrument where the bank guarantees that a buyer's payment to a seller will be received on time and for the correct amount. TradeX automates the issuance, amendment, and advising of Import and Export LCs. The system digitizes the presentation of shipping documents, allowing for rapid discrepancy checks and faster settlement times.

1.5.2 Bank Guarantees (BG)

A Bank Guarantee is a promise from the lending institution that if a particular borrower defaults on a loan or contract, the bank will cover the loss. Within TradeX, modules were developed to handle various forms of guarantees, including Performance Guarantees and Financial Guarantees. The system tracks the lifecycle of these instruments, from issuance and partial invocations to final expiry and cancellation.

1.5.3 Standby Letters of Credit (SBLC)

Often used in international trade, an SBLC functions similarly to a Bank Guarantee but is governed by different international customs (such as UCP 600 or ISP98). TradeX treats SBLCs as a specialized asset class, applying distinct workflow validations and Swift messaging formats to ensure legal compliance across jurisdictions.

1.5.4 Documentary Collections

TradeX also facilitates Documentary Collections, a process where the exporter entrusts the handling of commercial and financial documents to their bank. The system automates both Documents Against Payment (D/P) and Documents Against Acceptance (D/A), streamlining the tracking of bills of exchange and ensuring secure communication between the remitting and collecting banks.

1.6 System Architecture Of TradeX

To support these complex financial products, TradeX is built upon a distributed microservices architecture. This paradigm ensures that individual components of the trade lifecycle—such as user authentication, transaction processing, and messaging—operate independently. By decoupling these services, the platform achieves high availability, fault tolerance, and the ability to scale horizontally during peak banking hours.

Chapter 2

Tools and Technologies

This chapter provides an overview of the core technologies, frameworks, and tools utilized during the internship at IDFC FIRST Bank. The technology stack was chosen to ensure high performance, security, and scalability for enterprise-grade financial applications.

2.1 Golang

Go (or Golang) is a statically typed, compiled programming language designed at Google. It was the primary language used for backend development due to its exceptional concurrency model (Goroutines) and performance [1]. It powered the core microservices and API gateways of the TradeX platform.

2.2 React

React is a popular JavaScript library for building user interfaces. It was used to develop the frontend dashboards for both the TradeX platform and the SkyFox Cinemas module, providing a highly responsive and dynamic Single Page Application (SPA) experience [4].

2.3 Docker

Docker is a platform designed to help developers build, share, and run modern applications using containers. It was heavily utilized to containerize the microservices, ensuring consistent environments from local development through to staging and production [9].

2.4 Apache Kafka

Apache Kafka is a distributed event streaming platform used for high-performance data pipelines, streaming analytics, and data integration. In the TradeX architecture, Kafka facilitated the event-driven communication between decoupled microservices [10].

2.5 ORY Hydra

ORY Hydra is a hardened, OAuth 2.0 and OpenID Connect (OIDC) provider. It is highly scalable and was integrated into the system to manage API security, issue JWTs, and handle stateless authentication across the banking modules [13].

2.6 Permify

Permify is an open-source authorization service based on Google's Zanzibar paper. It was implemented to handle complex, fine-grained access control policies and Role-Based Access Control (RBAC) required for the Maker-Checker workflows in Trade Finance [14].

2.7 PostgreSQL

PostgreSQL is a powerful, open-source object-relational database system. It served as the primary persistent data store for transaction records, user profiles, and audit logs, chosen for its strong ACID compliance and reliability [15].

2.8 Git

Git is a distributed version control system used for source code management. It was essential for collaborative development, tracking changes, and managing feature branches and pull requests across the engineering team [16].

Chapter 3

Authentication Security with ORY Hydra

3.1 Overview Of API Security In Banking

In enterprise financial applications handling high-value Trade Finance products like LCs and BGs, securing Application Programming Interfaces (APIs) is paramount. Traditional session-based authentication is insufficient for a distributed microservices architecture like TradeX. Sessions require sticky routing and central databases, which create bottlenecks. To address this, the system leverages ORY Hydra, an industry-standard OAuth 2.0 and OpenID Connect (OIDC) provider, to enforce zero-trust network principles [13].

3.2 Understanding ORY Hydra

ORY Hydra is not a traditional identity provider (it does not manage user passwords, biometrics, or user databases); rather, it acts as a highly scalable authorization server. It is responsible for issuing, validating, and revoking access tokens. When a corporate client or bank employee attempts to log into TradeX, the authentication request is delegated to a secure internal Identity Provider (IdP), while Hydra manages the OAuth 2.0 flow to issue a JSON Web Token (JWT), as illustrated in Figure 3.1.

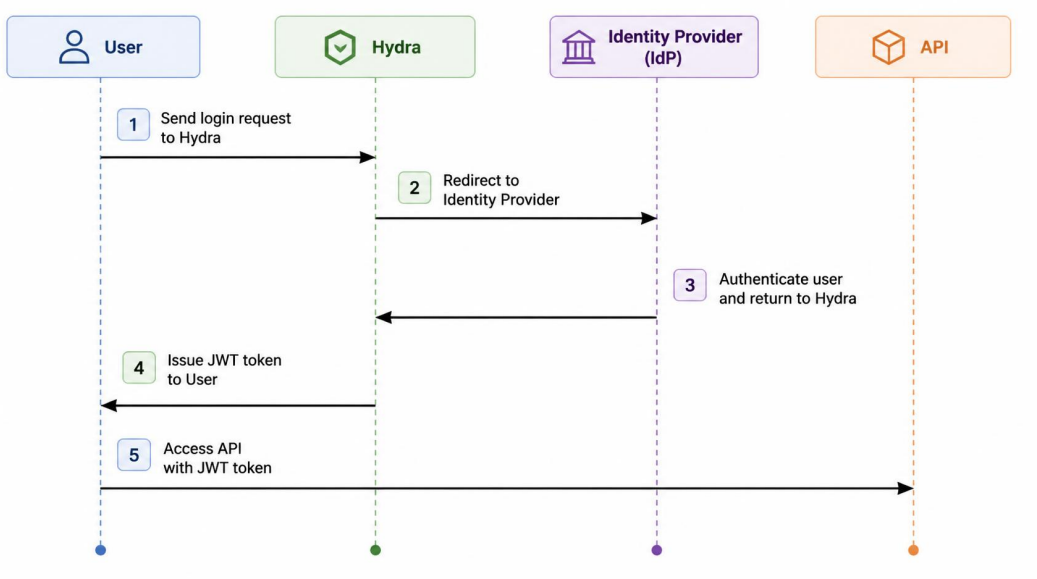

Figure 3.1: ORY Hydra Authentication Workflow

3.2.1 The Authorization Code Flow With PKCE

To secure Single Page Applications (SPAs) and mobile clients, TradeX utilizes the OAuth 2.0 Authorization Code Flow with Proof Key for Code Exchange (PKCE). This prevents authorization code interception attacks. When a user logs in, the client application generates a cryptographically random ‘code_verifier‘ and sends a hashed ‘code_challenge‘ to Hydra. Once the internal IdP authenticates the user, Hydra issues an authorization code. The client must then present the original ‘code_verifier‘ to exchange this code for an access token, ensuring the entity requesting the token is the exact same entity that initiated the login.

3.2.2 Stateless Authentication Workflow

The implementation of Hydra allows TradeX to maintain stateless authentication. Once Hydra issues a JWT, downstream microservices—such as the Letter of Credit module or the Reporting module—can independently verify the token’s cryptographic signature using a public key endpoint (JWKS). They do not need to query a central database for every request, which drastically reduces network latency and database overhead.

3.2.3 Token Lifecycle And Introspection

In a banking environment, a compromised token can lead to severe financial loss. Therefore, Hydra is configured with strict token lifecycles. Short-lived access tokens

(e.g., expiring in 15 minutes) are paired with secure HTTP-only refresh tokens. For highly sensitive operations, microservices utilize Hydra's Token Introspection endpoint. This allows the backend to perform real-time checks to ensure a token has not been prematurely revoked due to suspicious activity, providing an active layer of defense alongside the passive JWT signature validation.

Chapter 4

Fine-Grained Authorization using Permify

4.1 The Need For Complex Access Control

While ORY Hydra handles authentication (verifying the identity of the user), a separate, highly granular mechanism is required for authorization (determining exactly what that user is allowed to do). In a banking environment, standard Role-Based Access Control (RBAC) is often too rigid. For instance, an employee might have the role of a "Trade Finance Manager," but they should only be authorized to view and approve Bank Guarantees originating from their specific regional branch, up to a specific monetary limit. To implement this, TradeX uses Permify [14].

4.2 Introduction To Permify

To solve this complex authorization matrix, the architecture incorporates Permify, an open-source authorization service inspired by Google's Zanzibar paper. Permify utilizes Relationship-Based Access Control (ReBAC), allowing TradeX to define permissions based on the dynamic relationships between users, organizational hierarchies, and specific financial resources.

4.2.1 Schema Definition In ReBAC

Unlike traditional RBAC tables, Permify operates on a defined schema that maps out entities, relations, and actions. For example, an entity could be a 'branch', a relation could be 'manager', and an action could be 'approve_trade'. The schema allows for hierarchical inheritance, meaning if a user is granted access to a parent entity (e.g., a regional hub), Permify automatically calculates their inherited permissions for all child entities (e.g., local branches under that hub) without duplicating database records.

4.2.2 Implementation Of ReBAC In Trade Finance

In the TradeX platform, Permify is used to construct a distributed authorization graph. When a user attempts to approve a multimillion-dollar SBLC, the microservice queries the Permify engine with a specific tuple: *"Does User X have the specific relationship to approve SBLC Document Y?"* Permify evaluates the stored relational data—verifying branch allocation, departmental limits, and delegation rules—and returns a fast, deterministic allow or deny response.

4.2.3 Decoupling Authorization Logic At The API Gateway

A major architectural learning from integrating Permify was the decoupling of authorization logic from the core application code. Instead of writing complex, hard-coded `if-else` permission checks across various microservices, the security rules are centralized within the Permify schema and enforced at the API Gateway layer. This ensures that unauthorized requests are dropped before they ever reach the internal microservices, resulting in consistent, auditable security policy enforcement across the entire TradeX ecosystem.

Chapter 5

Approval Workflows: 2-Eye, 4-Eye, and 6-Eye Mechanisms

5.1 The Maker-Checker Principle

In financial software, the "Maker-Checker" principle is a fundamental security and compliance requirement. It mandates that the individual who initiates a transaction (the Maker) cannot be the same individual who authorizes it (the Checker). This segregation of duties is critical for preventing internal fraud, mitigating unauthorized data modifications, and catching human errors before financial instruments are issued.

5.2 State Machine Implementation

To handle complex approval matrices, the backend architecture relies on a strict Finite State Machine (FSM). A trade does not simply exist as "approved" or "unapproved." Instead, it traverses specific states: `DRAFT` $\rightarrow$ `PENDING_CHECKER_1` $\rightarrow$ `PENDING_CHECKER_2` $\rightarrow$ `AUTHORIZED` or `REJECTED`. The FSM ensures that a transaction cannot skip a required step, effectively hardcoding regulatory compliance into the application layer.

5.3 Approval Hierarchies In TradeX

Depending on the financial value, instrument type, and risk profile associated with a trade, the TradeX system dynamically routes the transaction through different approval tiers.

5.3.1 2-Eye Workflow

The 2-eye workflow involves a single individual. In this scenario, the user creates and submits a request that requires no further authorization. To minimize risk, this workflow is strictly limited to non-financial operations, such as updating user UI preferences, downloading read-only reports, or saving a draft of a transaction that has not yet been submitted to the bank network.

5.3.2 4-Eye Workflow

The 4-eye workflow is the standard Maker-Checker process utilized for the majority of daily trade operations, such as standard Documentary Collections or low-value Import LCs. It requires two distinct individuals. Once a Maker submits a trade, TradeX locks the record, ensuring data immutability. A designated Checker then reviews the transaction details against physical documents. If discrepancies are found, the Checker rejects the transaction, which reverts the state machine and returns the record to the Maker with mandatory rejection comments.

5.3.3 6-Eye Workflow

For high-value transactions, complex syndicated guarantees, or trades involving high-risk jurisdictions, the 6-eye workflow is triggered. This involves one Maker and two independent Checkers. The transaction must pass through a sequential multi-tiered approval matrix. For example, the issuance of a high-value Bank Guarantee may require primary approval from a Branch Supervisor (Checker 1) and a final authorization from a Central Compliance Officer (Checker 2) before the official SWIFT message is generated.

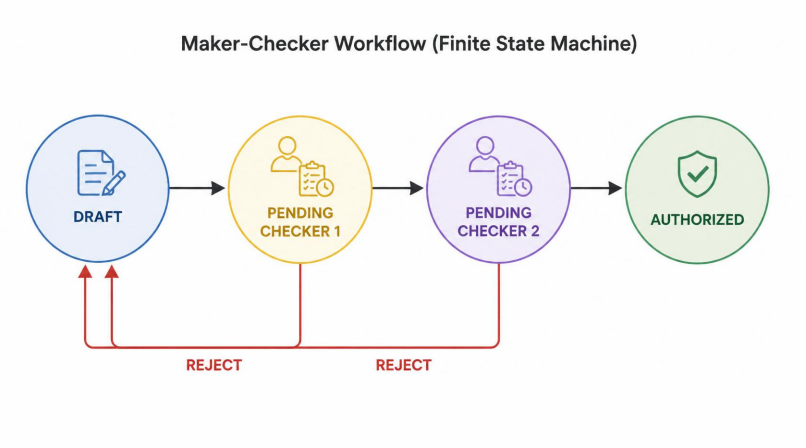

Figure 5.1: Maker-Checker Workflow: A Finite State Machine (FSM) flowchart.

5.3.4 Audit Trails And Immutability

Every transition within the Maker-Checker workflow is cryptographically logged. The system records the exact timestamp, the user ID, the IP address, and the specific state change. During the Checker phase, the payload is strictly immutable; the Checker cannot modify the trade amount or details. They can only approve or reject. This guarantees that the final approved transaction is mathematically identical to what the Maker originally submitted.

Chapter 6

Financial Messaging Integration: SFMS and SWIFT

6.1 Global And Domestic Financial Communication

A trade finance platform is only as effective as its ability to communicate securely with the broader financial ecosystem. Once a trade—such as an LC or BG—clears all internal Maker-Checker authorizations, TradeX must transmit the financial instructions to external banking networks. This is achieved through the seamless integration of SWIFT and SFMS protocols [17].

6.2 SWIFT Architecture

The Society for Worldwide Interbank Financial Telecommunication (SWIFT) is the global standard for international wire transfers and trade finance messaging. SWIFT acts as a secure messaging network that banks use to send standardized financial instructions to one another.

6.2.1 ISO 20022 vs. Legacy MT Formats

During the internship, I navigated the complex transition from legacy Message Type (MT) formats to the modern ISO 20022 XML standards (MX).

- **Legacy MT:** Formats like MT700 (Issue of a Documentary Credit) or MT760 (Issue of a Demand Guarantee) rely on rigid block structures (Blocks 1 through 5) with strict alphanumeric character limits.
- **ISO 20022 (MX):** Modern formats use highly structured XML, allowing for richer data payloads, better compliance screening, and easier parsing.

The backend system was tasked with supporting both structures, requiring the development of dynamic serializers that could convert internal database representations into either MT or MX payloads depending on the receiving bank's capabilities.

6.2.2 Network Acknowledgments (ACK/NACK)

Transmitting a message is only half the process. The system must also process asynchronous responses from the SWIFT network. When TradeX dispatches a message, it waits for an ACK (Acknowledgment) confirming the message was successfully routed, or a NACK (Negative Acknowledgment) indicating a failure due to formatting errors or network issues. The backend processes these callbacks to automatically update the transaction status on the user dashboard.

6.3 SFMS Architecture

For domestic transactions and intra-country bank guarantees within India, the system integrates with the Structured Financial Messaging System (SFMS). SFMS serves as the secure backbone for national electronic data interchange, facilitating networks like NEFT and RTGS.

6.3.1 Message Parsing And Cryptographic Security

Integrating SFMS required the development of highly reliable domestic parsing engines. Security is maintained by signing these messages cryptographically. Before an SFMS message is placed on the outbound queue, the payload is signed using the bank's digital certificates, ensuring absolute data integrity, non-repudiation, and confidentiality during network transit.

Chapter 7

Event-Driven Architecture with Apache Kafka

7.1 The Need For Asynchronous Processing

In a monolithic application, processes generally happen synchronously. For a Trade Finance operation, a single API call might need to update a database, send an email, generate a PDF report, and format a SWIFT message. If these happen sequentially, it creates massive system bottlenecks and latency. To achieve true decoupling, fault tolerance, and high throughput, the TradeX system relies on an Event-Driven Architecture powered by Apache Kafka.

7.2 Kafka Brokers And Partitions

Apache Kafka is a distributed event streaming platform. Unlike traditional message queues that delete messages once read, Kafka acts as a distributed commit log. Within TradeX, the architecture utilizes several core components:

- **Brokers:** The servers that store the data. TradeX utilizes a cluster of brokers to ensure high availability.
- **Topics and Partitions:** Messages are categorized into Topics (e.g., `tradex-swift-outbound`). To scale horizontally, these topics are split into Partitions, allowing multiple consumers to read from the same topic simultaneously without blocking each other.

Figure 7.1 illustrates the interaction between producers, brokers, and consumers within the TradeX ecosystem.

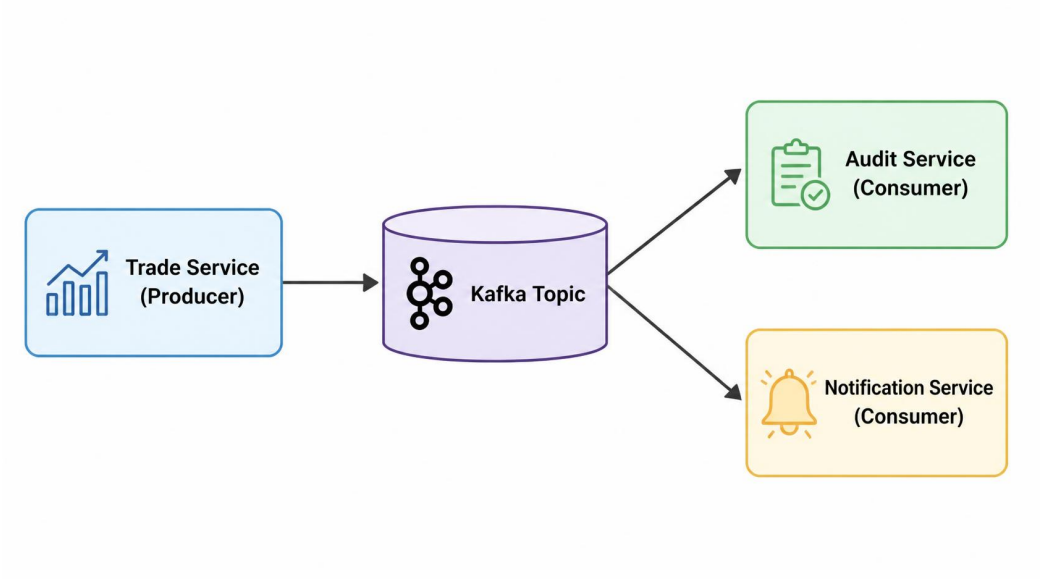

Figure 7.1: Event-Driven Architecture using Apache Kafka

7.3 Consumer Groups For Scalability

As trade volume increases during peak hours, a single microservice might struggle to process all incoming events. Kafka solves this using Consumer Groups. If the `Notification Service` is falling behind on sending approval emails, new instances of the service can be spun up. Kafka automatically rebalances the partitions, assigning different partitions to the new instances, thereby multiplying the processing speed without altering the codebase.

7.4 Practical Implementations In TradeX

Kafka is utilized to solve several complex engineering challenges within the bank.

7.4.1 Auditing And Compliance

Every action taken by a user—from logging in to approving a Bank Guarantee—must be strictly audited. Instead of slowing down the main transaction thread to write audit logs to a database, the core service acts as a Kafka Producer, immediately firing an event to an `audit` topic. A separate, dedicated Audit Consumer service reads this topic and handles the heavy database writes asynchronously.

7.4.2 Idempotency In Financial Transactions

In distributed systems, network timeouts can cause a producer to retry sending a message, potentially leading to duplicate events. In banking, processing a "Transfer Funds" event twice is catastrophic. To prevent this, the backend Kafka producers are configured for idempotency. Furthermore, consumer microservices implement idempotent processing logic, checking unique transaction IDs against the database before executing an action, guaranteeing exactly-once semantics even if Kafka delivers a message more than once.

Chapter 8

Technical Bootcamp and SkyFox Cinemas Project

8.1 Phase 1: Technical Upskilling And Dojo Exercises

Prior to transitioning into the core TradeX platform, the internship commenced with an intensive, month-long technical bootcamp designed to elevate programming capabilities to enterprise standards. The curriculum focused heavily on IDFC FIRST Bank's modern technology stack, specifically Golang for high-performance backends, React for dynamic frontends, Docker for containerization, and Apache Kafka for event-driven workflows [1, 4, 9, 10].

To solidify theoretical knowledge, the bootcamp utilized a "Dojo" system—timed, hands-on coding challenges. Notable exercises included building a concurrent web scraper and a rate-limiter in Go, forcing interns to handle race conditions and mutex locks in isolated environments. Additionally, local Kafka instances were set up to write Go producers and consumers simulating transactional events, ensuring a deep understanding of message persistence.

8.2 Phase 2: Agile Project Simulation (SkyFox Cinemas)

Following the upskilling phase, the cohort transitioned into a simulated Agile engineering team to build a comprehensive Movie Booking Module named SkyFox Cinemas. This project addressed the growing industry trend of integrating lifestyle services directly into banking super-apps [8].

8.2.1 Navigating Aggressive Sprints

The simulation enforced real-world pressures, dividing the project into two-week sprints managed via Jira [12]. A significant challenge was the uneven timeline; Sprint 2 was compressed to merely 2.5 days of actual development work, requiring strict prioritization of backlog items and scale-downs of non-critical features to meet delivery deadlines.

8.2.2 Security And Architectural Workarounds

Operating within a simulated banking environment required strict adherence to internal policies, leading to several innovative engineering solutions:

- **Mock OTP Verification:** Initially, a live Gmail API was utilized for user verification. However, bank firewalls blocked these external calls. A rapid pivot was required to engineer a secure, internal Mock Verification system utilizing Go's `crypto/rand` package to generate unpredictable 6-digit OTPs with strict 3-minute expiration timers.
- **Base64 Profile Personalization:** Due to restrictions preventing the use of external cloud storage (like AWS S3), user profile images could not be stored conventionally. To solve this, a React utility was written to compress images via HTML5 Canvas and convert them to Base64 strings [7, 6]. To prevent performance degradation, the large Base64 strings were isolated in the database schema and lazy-loaded only when the user navigated to their profile.

8.2.3 Concurrency Control For Seat Booking

The most technically rigorous feature was the core booking engine. During peak ticket sales, the system had to prevent race conditions and double-bookings. Utilizing Go's concurrency model, a backend locking mechanism was implemented using Go routines, database transactions, and Mutex locks [1, 3]. If two users clicked the exact same seat at the same millisecond, the requests were processed sequentially; the first received a temporary lock, while the second received a graceful error, entirely preventing data corruption. Additionally, the end-to-end flow was polished with dynamic QR code generation and PDF ticket downloads.

8.3 Test-Driven Development (TDD) And Code Coverage

To mitigate the risk of system outages, the Agile team strictly adhered to Test-Driven Development (TDD) [11]. Utilizing the "Red-Green-Refactor" cycle, tests were written before feature implementation.

The Golang backend testing strategy was divided into three layers [2]:

- **Controller Tests:** Validating routing logic and HTTP status codes using the `httptest` package.
- **Service Tests:** Utilizing Table-Driven Tests and `gomock` to validate core business logic without database connections.
- **Repository Tests:** Running isolated Docker containers with test databases to verify complex SQL queries and Mutex locks.

The frontend UI was validated using Jest and the React Testing Library (RTL) [5]. This rigorous TDD approach ultimately yielded exceptional code reliability, achieving a 95.2% test coverage in the Go Services (Business Logic) layer and approximately 85.0% coverage across the React frontend.

Chapter 9

Conclusion and Future Scope

9.1 Conclusion

The internship at IDFC FIRST Bank provided an invaluable opportunity to bridge the gap between academic theory and enterprise-level software engineering. Starting with an intensive technical bootcamp, a strong foundation in modern backend and frontend technologies was developed, specifically utilizing Golang, React, Docker, and Apache Kafka. The Agile project simulation, SkyFox Cinemas, further honed the ability to deliver scalable and robust applications under strict deadlines, simulating real-world pressures and constraints.

Transitioning to the TradeX platform provided deep insights into the complexities of global trade finance. Working on critical modules involving Letters of Credit, Bank Guarantees, and Documentary Collections exposed the stringent security, compliance, and messaging standards required in the banking sector. Implementing advanced architectural patterns—such as Zero-Trust security with ORY Hydra, fine-grained authorization with Permify, strict Maker-Checker finite state machines, and event-driven processing with Kafka—significantly elevated technical proficiency. The hands-on experience with SWIFT and SFMS messaging protocols also underscored the importance of interoperability and cryptographic security in financial networks.

Overall, the internship not only enhanced programming capabilities but also instilled a profound understanding of software architecture, security, and the rigorous quality assurance required to build fault-tolerant financial systems.

9.2 Future Scope

While significant milestones were achieved during the internship, there remain several avenues for future enhancement within the TradeX platform and associated modules:

- **AI-Driven Discrepancy Checking:** Implementing Machine Learning models to automatically scan and identify discrepancies in uploaded trade documents

(e.g., Bills of Lading) before human Checker review, further reducing processing time.

- **Blockchain Integration for Trade Finance:** Exploring decentralized ledgers to provide real-time, immutable tracking of Letters of Credit across multiple international banks, thereby increasing transparency and reducing fraud.
- **Cloud-Native Scaling:** Further optimizing the microservices architecture for deployment on managed Kubernetes clusters, enabling autoscaling based on real-time SWIFT messaging volumes during peak global trading hours.
- **Enhanced Analytics Dashboard:** Developing real-time analytics for corporate clients to predict cash flow constraints based on their historical Trade Finance activities and pending Letters of Credit.

References

[1] Go Programming Language Documentation, "Concurrency: Goroutines and Channels," See `https://go.dev/doc/effective_go#concurrency`

[2] Go Standard Library, "Package testing: Automated testing for Go packages," See `https://pkg.go.dev/testing`

[3] PostgreSQL Documentation, "Explicit Locking and Concurrency Control," See `https://www.postgresql.org/docs/current/explicit-locking.html`

[4] ReactJS Official Documentation, "React: A JavaScript library for building user interfaces," See `https://react.dev/learn`

[5] React Testing Library Documentation, "Testing UI Components from a User Perspective," See `https://testing-library.com/docs/react-testing-library/intro/`

[6] Mozilla Developer Network (MDN) Web Docs, "Base64 encoding and decoding algorithms," See `https://developer.mozilla.org/en-US/docs/Glossary/Base64`

[7] Mozilla Developer Network (MDN) Web Docs, "Optimizing and converting images via HTML5 Canvas," See `https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toDataURL`

[8] Chris Richardson, *Microservices Patterns: With examples in Java,* Manning Publications, 2018.

[9] Docker Documentation, "Containerizing applications and microservices," See `https://docs.docker.com/get-started/`

[10] Apache Kafka Documentation, "Introduction to Event Streaming and Distributed Systems," See `https://kafka.apache.org/documentation/`

[11] Kent Beck, *Test-Driven Development: By Example,* Addison-Wesley Professional, 2002.

[12] Ken Schwaber and Jeff Sutherland, "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game," See https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[13] Ory Corp, "ORY Hydra Documentation: OAuth 2.0 and OpenID Connect," See https://www.ory.sh/docs/hydra/

[14] Permify, "Permify Documentation: Open Source Authorization Service," See https://docs.permify.co/

[15] PostgreSQL Global Development Group, "PostgreSQL Documentation," See https://www.postgresql.org/docs/

[16] Software Freedom Conservancy, "Git Documentation," See https://git-scm.com/doc

[17] SWIFT, "SWIFT Standards and Financial Messaging," See https://www.swift.com/standards