Complete Guide: Building Blockchain for TRACIENT

Blockchain & AI Enabled Income Traceability System

Document Version: 1.0
Date: December 5, 2025
Project: TRACIENT (Group 6 - Major Project)

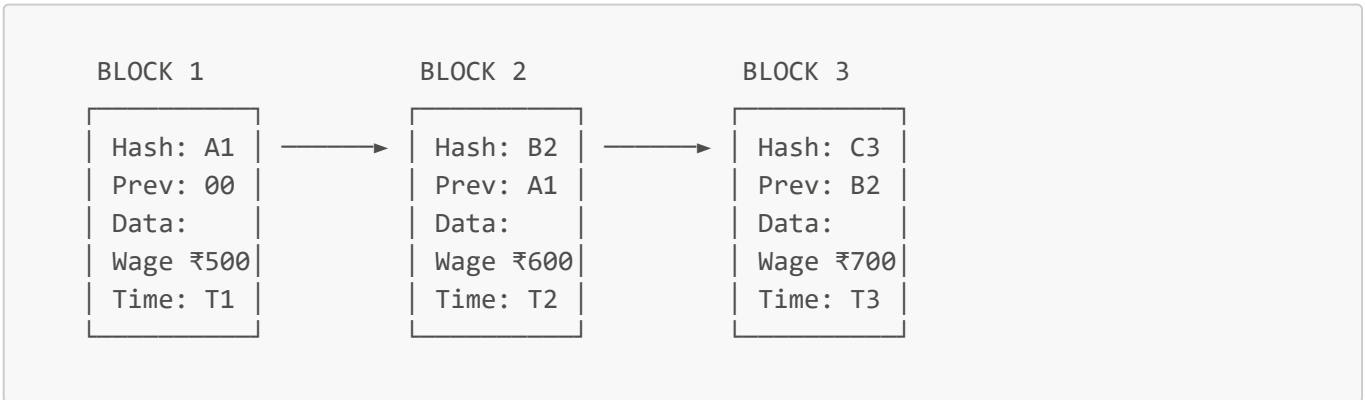
Table of Contents

PART 1: Blockchain Fundamentals

1.1 What is a Blockchain?

A blockchain is a **distributed, immutable digital ledger** that records data in linked blocks.

How Blockchain Works



Each block contains:

- **Hash** - Unique fingerprint of the block
- **Previous block's hash** - Creates the chain linkage
- **Data** - Wage records in TRACIENT's case

- **Timestamp** - When the block was created

1.2 Key Characteristics

Property	Description	Relevance to TRACIENT
Immutability	Data cannot be changed once recorded	Wage records cannot be falsified
Decentralization	No single point of control	Government, employers, banks all have copies
Transparency	All authorized parties can verify	Auditable welfare eligibility
Consensus	Network agrees on valid data	Prevents fraudulent wage entries
Security	Cryptographic protection	Worker identity remains private

1.3 Types of Blockchain

Type	Access	Use Case	Example
PUBLIC	Anyone	Cryptocurrency	Bitcoin, Ethereum
PRIVATE	Single org	Enterprise	Internal systems
CONSORTIUM	Multiple orgs	B2B	Supply chain
PERMISSIONED	Invited only	Government	Hyperledger Fabric

TRACIENT uses: PERMISSIONED (Hyperledger Fabric)

- Only verified parties (govt, employers) can participate
- No cryptocurrency/mining needed
- Privacy-preserving
- Fast transaction finality

PART 2: Why Hyperledger Fabric for TRACIENT?

2.1 Hyperledger Fabric Overview

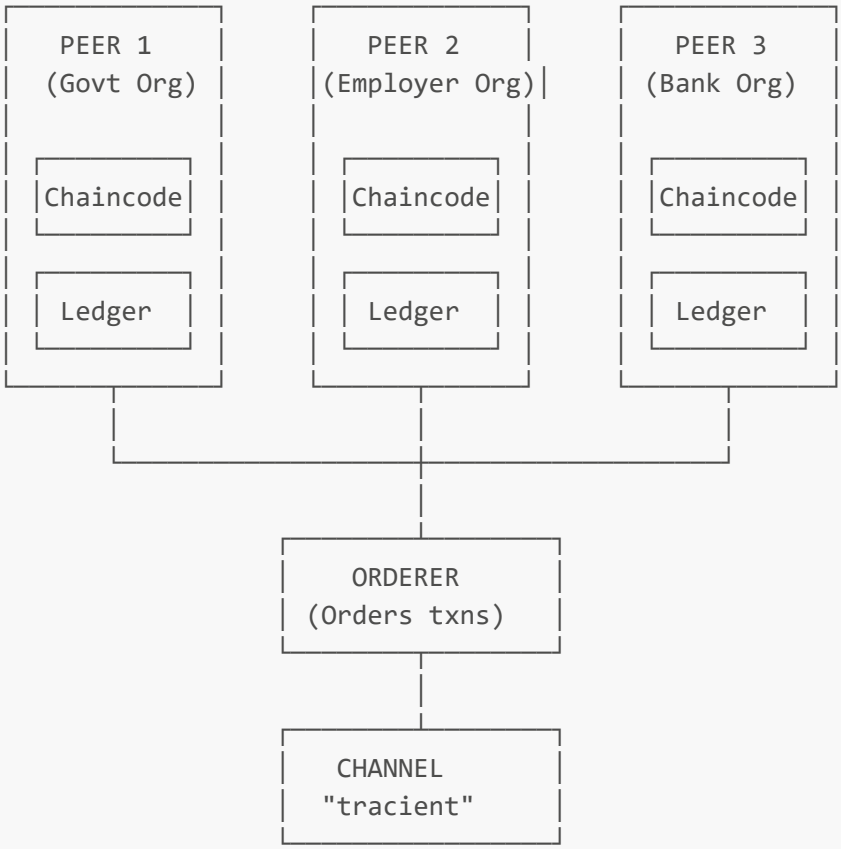
Hyperledger Fabric is an **enterprise-grade, permissioned blockchain** platform.

Why It's Perfect for TRACIENT

Feature	Benefit
Permissioned Network	Only govt, verified employers, banks can join
No Cryptocurrency	No mining, no tokens, just data storage
Private Data Collections	Worker identity stays private from other employers

Feature	Benefit
Smart Contracts (Chaincode) in Go	Business logic: RecordWage, QueryHistory, ClassifyWorker
Membership Service Provider (MSP)	Certificate-based identity management
Channel Architecture	Separate channels for different states/departments

2.2 Fabric Network Components

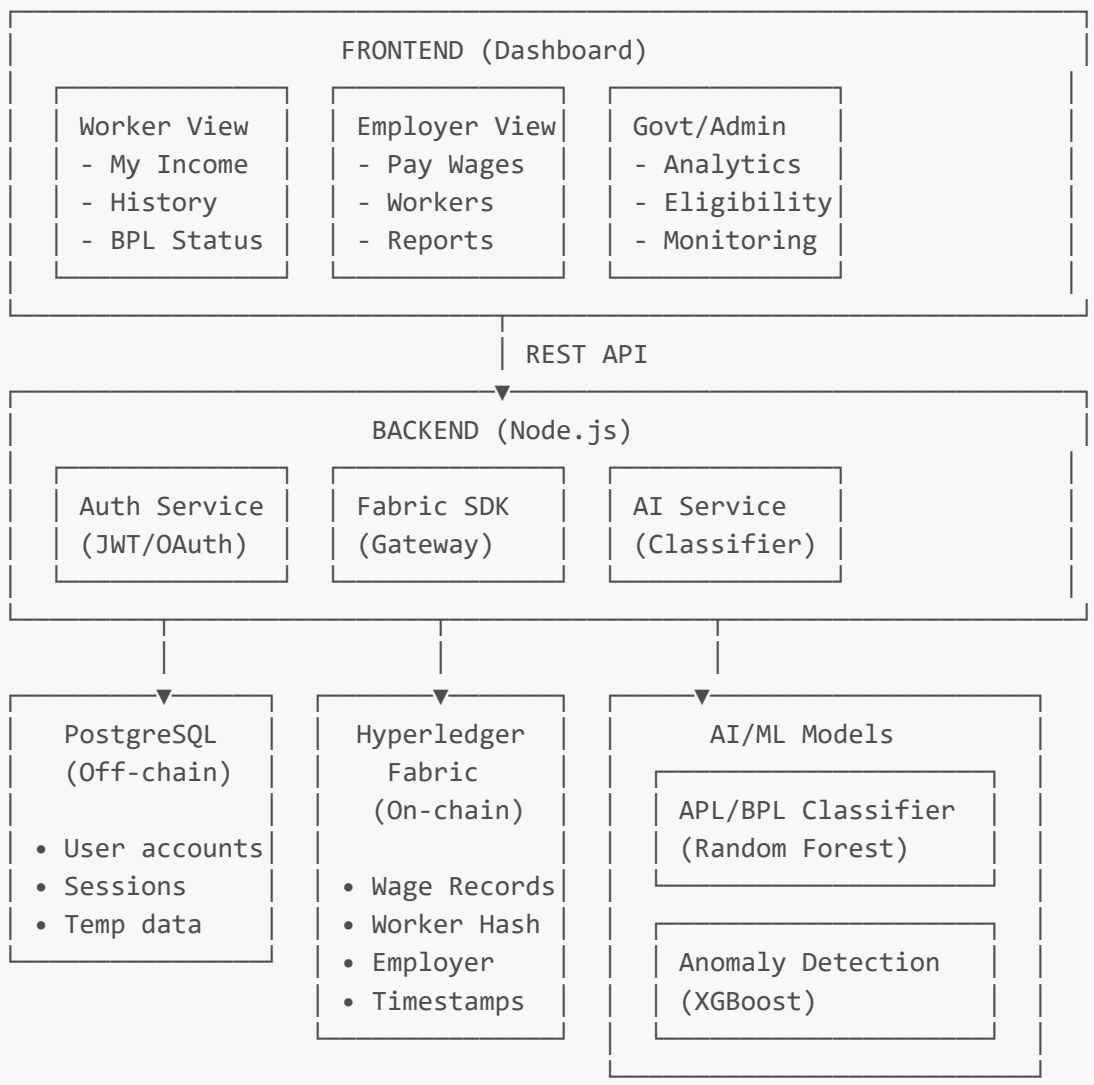


Component Descriptions

Component	Description
PEER	Node that holds ledger copy + executes chaincode
ORDERER	Orders transactions into blocks
CHANNEL	Private subnet for specific participants
CHAINCODE	Smart contract (your business logic)
MSP	Manages identities and certificates
CA	Certificate Authority (issues identities)

PART 3: TRACIENT System Architecture

3.1 Complete System Overview



3.2 Data Flow

Step 1: Employer Pays Worker

```
Employer App ──► Backend API ──► Hash(Aadhaar) ──► Blockchain
```

Data Stored on Blockchain:

```
{
  "wageID": "WAGE001",
  "workerIdHash": "sha256(aadhaar)",
  "employerIdHash": "sha256(pan)",
  "amount": 1500.00,
  "jobType": "construction",
```

```
  "timestamp": "2025-12-05T10:30:00Z"
}
```

Step 2: AI Classification Triggered

```
Backend ──► Query Worker History ──► Calculate Features ──► AI Model
```

Features Calculated:

- Total income (24 months)
- Income variability
- Number of employers
- Payment patterns

AI Outputs:

- BPL/APL Classification (APL/BPL Model)
- Anomaly Flag (Anomaly Detection Model)

Step 3: Results Stored & Dashboard Updated

```
AI Result ──► Store Classification on Blockchain ──► Update Dashboard
```

Classification Record:

```
{
  "workerIdHash": "sha256(aadhaar)",
  "classification": "BPL",
  "confidence": 94.5,
  "anomalyFlag": false,
  "policyVersion": "2025-Q4",
  "timestamp": "2025-12-05T10:30:05Z"
}
```

PART 4: Smart Contract (Chaincode) Explained

4.1 What is Chaincode?

Chaincode is **Hyperledger Fabric's smart contract** - the business logic that runs on the blockchain.

4.2 TRACIENT Chaincode Functions

Write Functions

Function	Description
`RecordWage(wageID, workerHash, employerHash, amount, jobType)`	Stores a new wage payment on the ledger
`UpdateClassification(workerHash, classification, confidence)`	Stores AI classification result
`RegisterWorker(workerHash, encryptedDetails)`	Registers a new worker in the system
`RegisterEmployer(employerHash, orgName, gstin)`	Registers a new employer

Read Functions

Function	Description
`ReadWage(wageID)`	Retrieves a single wage record
`QueryWageHistory(workerHash)`	Gets all wages for a worker (for AI input)
`GetWorkerClassification(workerHash)`	Returns current BPL/APL status
`GetWagesByDateRange(workerHash, startDate, endDate)`	Retrieves wages in a time period
`GetAggregateStats(workerHash)`	Returns total income, avg monthly, etc.

4.3 Data Structure (Go)

```
// WageRecord - Data Structure on Blockchain
type WageRecord struct {
    WorkerIDHash   string  `json:"workerIdHash"`    // SHA256(Aadhaar) - Privacy!
    EmployerIDHash string  `json:"employerIdHash"`  // SHA256(PAN)
    Amount         float64 `json:"amount"`          // Wage amount in INR
    Currency       string  `json:"currency"`        // "INR"
    JobType        string  `json:"jobType"`         // "construction", "domestic",
etc.
    Timestamp      string  `json:"timestamp"`       // ISO 8601 format
    PolicyVersion  string  `json:"policyVersion"`   // "2025-Q4"
}
```

PART 5: Privacy & Security

5.1 The Problem

Cannot store Aadhaar/PAN directly on blockchain (privacy law compliance)

5.2 Solution: Hash-based Anonymization

```
Aadhaar: 1234-5678-9012
         │
         ▼
SHA-256("1234567890120" + salt)
         │
         ▼
Hash: "a1b2c3d4e5f6..."  ← This is stored on blockchain
```

Benefits:

- ☑ Cannot reverse hash to get Aadhaar
- ☑ Same Aadhaar always produces same hash (linkable)
- ☑ Different Aadhaar produces different hash

5.3 Additional Privacy Layers

Technique	Description
Private Data Collections	Sensitive data only shared with authorized organizations
Zero-Knowledge Proofs (ZKP)	Prove "income < threshold" without revealing actual income
Attribute-Based Access Control (ABAC)	Employers see only their workers' data

PART 6: Step-by-Step Implementation Process

Phase 1: Local Proof of Concept (Weeks 1-4)

Week 1-2: Environment Setup

- Install Docker, WSL2, Go, Node.js, Python
- Download Hyperledger Fabric binaries
- Start test-network
- Deploy sample chaincode

Week 3: Custom Chaincode

- Extend your tracient chaincode
- Add RecordWage, QueryHistory functions
- Test via CLI

Week 4: Basic Integration

- Create Node.js backend with Fabric SDK
- Connect AI models via REST API
- Simple React frontend

Deliverable: Working local prototype

Phase 2: MVP Development (Weeks 5-10)

Week 5-6: Containerization

- Dockerfile for each service
- docker-compose.yml orchestration
- Environment configuration

Week 7-8: Enhanced Features

- User authentication (JWT)
- Role-based access (Worker, Employer, Admin)
- AI model integration pipeline

Week 9-10: Dashboard Development

- Worker dashboard (income history, status)
- Employer dashboard (wage submission)
- Admin dashboard (analytics, monitoring)

Deliverable: Functional MVP with dashboards

Phase 3: Production Prep (Weeks 11-16)

Week 11-12: Security Hardening

- Implement hashing for Aadhaar/PAN
- Access control policies
- Security testing

Week 13-14: Testing

- Unit tests (chaincode, API, AI)
- Integration tests
- Performance testing

Week 15-16: Documentation & Deployment

- API documentation
- System architecture docs
- Cloud deployment (optional)

Deliverable: Production-ready system

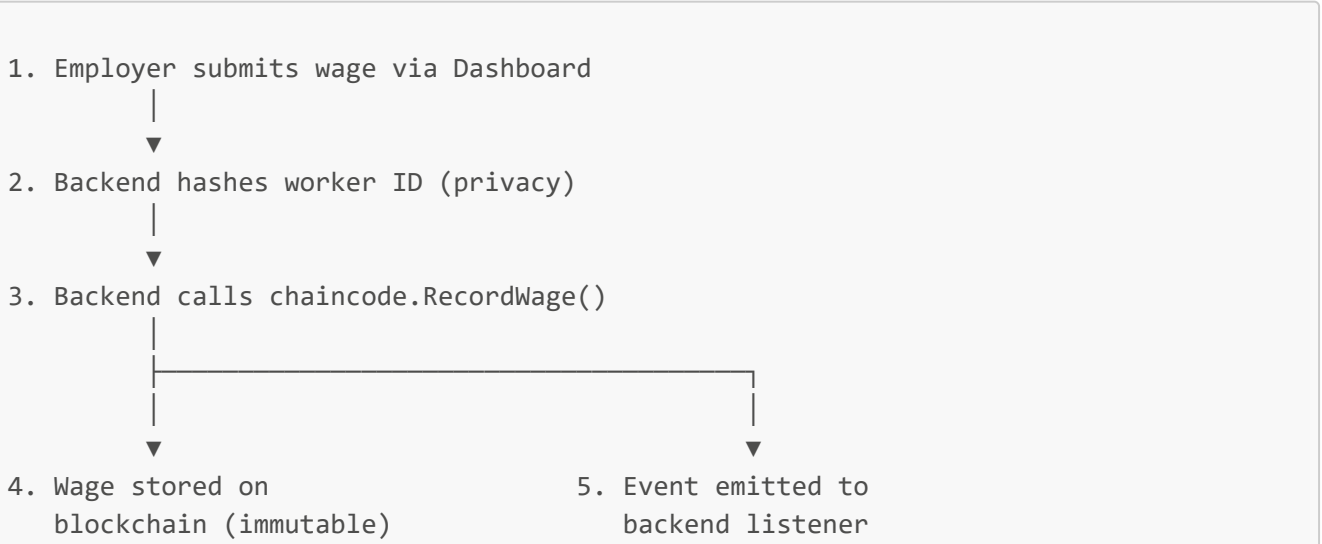
Technology Stack Summary

Layer	Technology	Purpose

Layer	Technology	Purpose
Frontend	React.js	User interfaces
	Tailwind CSS	Styling
	Chart.js	Data visualization
Backend	Node.js	API server
	Express.js	REST framework
	Fabric SDK	Blockchain connection
	JWT	Authentication
Blockchain	Hyperledger Fabric	Distributed ledger
	Go (chaincode)	Smart contracts
	Docker	Container runtime
AI/ML	Python	Model serving
	scikit-learn	APL/BPL model
	XGBoost	Anomaly detection
	Flask/FastAPI	Model API
Database	PostgreSQL	Off-chain data
	CouchDB	Fabric state database
DevOps	Docker Compose	Local orchestration
	WSL2	Windows-Linux bridge

PART 7: How AI Integrates with Blockchain

7.1 Integration Flow



```
                    │
                    ▼
6. Backend queries
   worker's full history
                    │
                    ▼
7. Features calculated
   (total income, patterns)
                    │
                    ▼
8. AI Models invoked:
   ├─ APL/BPL Classifier
   └─ Anomaly Detector
                    │
                    ▼
9. Results stored on
   blockchain + dashboard
```

7.2 AI Model Usage

Model	Trigger	Input	Output
APL/BPL Classifier	New wage recorded OR monthly batch	Worker's 24-month income data, household info	BPL/APL classification + confidence
Anomaly Detector	Every wage transaction	Transaction patterns, timing, sources	Normal/Anomaly flag + risk score

PART 8: Dashboard Architecture

8.1 Worker Dashboard

Features:

- My Income Summary (total, monthly avg)
- Transaction History (all wages received)
- Current BPL/APL Status
- Welfare Eligibility (schemes I qualify for)
- Income Trend Chart

8.2 Employer Dashboard

Features:

- Pay Wage (form to submit new payment)
- My Workers List
- Payment History
- Monthly Expense Reports
- Compliance Status

8.3 Government/Admin Dashboard

Features:

- Overview Statistics (total workers, wages, classifications)
- BPL/APL Distribution (pie chart, by region)
- Anomaly Alerts (flagged suspicious transactions)
- Income Distribution Analysis
- Welfare Scheme Eligibility Reports
- Real-time Monitoring
- Policy Impact Analysis

PART 9: What You Need to Learn/Do

9.1 Knowledge Prerequisites

Blockchain Concepts

Topic	Study Time
Hashing (SHA-256)	2 hours
Public/Private Key Cryptography	3 hours
Merkle Trees	1 hour
Consensus Mechanisms	2 hours
Smart Contracts Concept	2 hours

Hyperledger Fabric

Topic	Study Time
Fabric Architecture (Peers, Orderers)	3 hours
Channels and MSP	2 hours
Chaincode Development (Go)	5 hours
Fabric SDK (Node.js)	4 hours
Docker & Docker Compose	3 hours

Programming

Topic	Study Time
Go Language Basics	8 hours
Node.js + Express	Already known

Topic	Study Time
React.js	Already known
Python (for AI)	☑ Done

Total Estimated Study Time: ~35 hours

9.2 Recommended Learning Resources

Topic	Resource	Time
Blockchain Basics	YouTube: "Blockchain Explained" by 3Blue1Brown	30 min
Hyperledger Fabric	Official Docs: hyperledger-fabric.readthedocs.io	4 hours
Fabric Tutorial	Fabric Samples: test-network walkthrough	3 hours
Go Language	Tour of Go (tour.golang.org)	4 hours
Chaincode Dev	Fabric Smart Contract Tutorial	3 hours

PART 10: Summary - Your Next Steps

Already Completed ☑

- AI Models (APL/BPL + Anomaly Detection) - Working!
- Chaincode structure (tracient/chaincode.go) - Basic version
- Project documentation

Next Steps (In Order)

Step	Task	Description
1	Setup WSL2 + Docker on Windows	Required for running Hyperledger Fabric
2	Download Fabric binaries & run test-network	Follow blockchain/README.md instructions
3	Deploy your tracient chaincode to test-network	Test RecordWage, QueryHistory via CLI
4	Create Node.js backend with Fabric SDK	REST API that connects to blockchain
5	Integrate AI models with backend	Flask API for Python models
6	Build React dashboard	Worker, Employer, Admin views
7	Dockerize everything	docker-compose for one-command startup

Key Takeaways

1. **Hyperledger Fabric** is the right choice for TRACIENT - permissioned, private, no crypto mining

2. **Chaincode (Go)** is your smart contract - defines what data is stored and how
3. **Privacy is critical** - never store raw Aadhaar/PAN, always hash first
4. **AI runs off-chain** - queries blockchain data, processes it, stores results back
5. **Dashboard is the user interface** - connects to backend which talks to blockchain + AI
6. **Development is phased** - start local, then containerize, then harden for production

---

Document Prepared for: TRACIENT Project (Group 6)
Date: December 5, 2025