Tab 1

AGRICHAIN — Redefined System Architecture

You currently have:

Farmer → Distributor → Retailer → Consumer
Basic verification + QR tracking

That's a *linear tracking model.*

We're going to upgrade it into:

> **A decentralized, AI-powered agricultural trust & intelligence network**

1️⃣ First Major Decision: Public or Private Ethereum?

Let's be practical.

❌ Public Ethereum (Mainnet)

- High gas fees
- Slow confirmation
- Not scalable for daily agri transactions
- Not suitable for Aadhaar-linked identity

⚠️ Public Testnets

Good for demo. Not production.

✅ Best Choice: Permissioned Blockchain

Use:

- **Hyperledger Besu (private Ethereum network)**
- Or **Polygon Supernets**
- Or **Quorum**

My recommendation for your case:

🔒 **Private Ethereum (Besu) + Anchor Hash to Public Ethereum**

Why?

- Real supply chains need privacy
- You store land docs, Aadhaar verification hashes
- Government partnerships require controlled nodes
- You control validators (Farmer Board / FPO / Govt / Audit Authority)

🧠 Redefined AGRICHAIN Architecture

We divide system into 5 layers:

1. Identity & Trust Layer
2. Product Lifecycle Layer
3. AI Intelligence Layer
4. Compliance & Audit Layer
5. Consumer Transparency Layer

2️⃣ Identity & Trust Layer (Very Important)

Current model: user registers → gets private key.

We improve this.

🔐 Use DID (Decentralized Identity)

Each user gets:

- Wallet address
- DID ID
- Role-based smart contract access
- Verified credentials stored as hashes

Instead of storing documents on chain:

- Store in IPFS
- Store only hash in blockchain

For Aadhaar

Never store Aadhaar.
Store:

SHA256(aadhaar + salt)

3️⃣ Product Lifecycle Redefined

Instead of "verify same product"

We introduce:

🌾 Digital Product Passport (DPP)

Each product batch has:

- Batch ID
- Farm geo-location
- Soil quality data
- Fertilizer usage
- Harvest date
- Storage condition logs
- Transport temperature logs
- Ownership transfer history

Flow Becomes:

Farmer:

- Creates Batch NFT (Non-tradable)
- Uploads crop metadata
- AI validates anomaly
- Signs transaction

Distributor:

- Accepts custody
- Adds transport logs
- IoT sensor integration (future)
- Timestamp + geotag

Retailer:

- Confirms receipt
- Adds storage metadata
- Sets expiry

Consumer:

- Scans QR
- Views full traceability
- AI quality score
- Trust index

4️⃣ Smart Contract Structure

We don’t use one big contract.

We modularize:

Contracts:

1. IdentityRegistry.sol
2. RoleManager.sol
3. ProductBatch.sol
4. OwnershipTransfer.sol
5. AuditTrail.sol
6. AIOracle.sol

5️⃣ AI Integration (Your Competitive Advantage)

Now we make this intelligent.

👨‍🌾 Farmer AI Agent

- Price prediction (already you planned)
- Crop disease detection (image model)
- Yield estimation
- Fraud land detection

🚚 Distributor AI Agent

- Route optimization
- Spoilage risk detection
- Cold-chain anomaly alert

🏪 Retailer AI Agent

- Demand forecasting
- Waste prediction
- Smart reorder recommendation

🛒 Consumer AI Agent

- Allergy safety check
- Nutritional recommendation
- Authenticity trust score

---

6 AI + Blockchain Interaction

AI does NOT write directly to chain.

Flow:

AI Model → Backend validation → Oracle contract → Store decision hash on-chain

Never store raw ML output directly.

---

7 System Architecture Diagram (Conceptual)

Android App
↓
API Gateway (FastAPI / Node)
↓
AI Engine (PyTorch / Transformers)
↓
Private Ethereum (Besu)
↓
IPFS (Documents + Metadata)
↓
Public Ethereum Anchor (Daily Hash)

---

8 Database Design

Use hybrid model:

Component	Use
PostgreSQL	User metadata

Redis	Session cache
IPFS	Documents
Ethereum	Immutable audit

9️⃣ Language Compatibility Upgrade

You asked about better language automation.

Use:

- Google ML Kit Translation API
- DeepL API (better accuracy)
- Meta NLLB models (open-source multilingual)

But best approach:

Store content keys like:

dashboard.price_prediction.title

Then dynamically fetch translation.

Do NOT hardcode translations.

🔟 Security Model

You must implement:

- Multi-signature for batch disputes
- Slashing validator nodes
- Role-based transaction validation

- KYC status flag in smart contract
- Fraud detection AI flagging

1️⃣1️⃣ Add Advanced Features

Now we elevate it beyond tracking.

🌍 Government Integration

- Land registry API
- Soil health card integration

🏦 DeFi Extension

- Farmer collateral loans using batch NFT
- Crop insurance smart contracts

📊 ESG Reporting

- Carbon footprint tracking
- Water usage score

1️⃣2️⃣ Data You Should NOT Put on Blockchain

- Aadhaar number

- Personal phone numbers
- Raw images
- GPS raw logs

Only store:

- Hashes
- References
- Proofs

---

13 QR Code Structure

QR should contain:

agrichain://batchID=0x34f...

App fetches metadata from backend + blockchain.

Never expose raw node RPC to public app.

---

14 Governance Model

Who runs validators?

Options:

- Government agriculture dept
- Farmer Producer Organizations
- Certified distributors

- Independent audit body

Minimum 5 validator nodes.

1️⃣5️⃣ Final System Model (Mature Version)

This is no longer:

“Track product from farm to table.”

It becomes:

> A decentralized agricultural trust infrastructure with AI-based validation, fraud detection, and supply chain intelligence.

🚀 What You Built vs What You Should Build

Current:
Simple verification + timestamp

Upgraded:

- Identity layer
- Digital product passport
- AI-driven anomaly detection
- Oracle integration
- Modular smart contracts
- Hybrid architecture
- Governance model

Tracking becomes powerful only if it provides:

1️⃣ Anti-Counterfeit Protection

Consumer verifies:

- This tomato actually came from X farm
- It was not replaced in between

This matters in:

- Organic products
- Export goods
- High-value crops (saffron, spices, premium rice)

2️⃣ Food Safety & Recall System

If contamination happens:

Instead of recalling ALL products,
You recall only:

Batch ID 0x7829 from Farm A on 12 Jan.

That saves:

- Money
- Reputation
- Lives

3️⃣ Trust Score for Farmers

If Farmer A consistently:

- Delivers on time
- Has low spoilage
- Follows organic certification

His trust score increases.

Now:
Retailers prefer him.
Banks trust him for loans.

Tracking → Reputation economy.

4️⃣ Dispute Resolution

Example:

Distributor claims:
"Product was spoiled when I received it."

Farmer claims:
"It left my farm fresh."

Blockchain timestamp + condition logs settle dispute.

Without tracking?
It becomes blame game.

5️⃣ Export & Certification Support

If exporting to EU or US:
They require:

- Traceability
- Proof of pesticide usage
- Cold chain logs

Tracking enables compliance.

💡 How To Make Tracking Actually Valuable

Instead of just showing:

Timeline view

Show:

🔥 Smart Insights Layer

When consumer scans:

Show:

- Farm story
- Soil type
- Organic verification status
- Transport temperature safety status
- AI freshness score
- Estimated days remaining before spoilage
- Carbon footprint

Now scanning becomes meaningful.

Layer	Tech
Mobile	Flutter
Backend	FastAPI
AI	PyTorch + Transformers
Blockchain	Hyperledger Besu
Contracts	Solidity + Hardhat
DB	PostgreSQL
Cache	Redis
Storage	IPFS
Queue	Kafka
Container	Docker
Cloud	AWS

AGRICHain =
AI-powered, permissioned supply chain blockchain system with mobile app interface

So your stack must support:

- Identity + KYC
- Private blockchain
- IPFS document storage
- AI services
- Real-time APIs
- Role-based dashboards
- QR-based traceability

- Future scalability

🏗️ COMPLETE SYSTEM ARCHITECTURE

We divide into 7 layers:

1. Mobile App Layer (Flutter)
2. API Gateway Layer
3. Core Backend Layer
4. AI Engine Layer
5. Blockchain Layer
6. Storage Layer
7. DevOps & Infrastructure Layer

Let’s define each properly.

1️⃣ Mobile Layer (You Already Use Flutter)

✅ Flutter

Keep it.

Why?

- Cross platform
- Good for rural Android devices
- Fast iteration

Add:

- Riverpod / Bloc (state management)
- Dio (networking)
- Web3dart (blockchain interaction)
- QR code scanner package

But very important:

⚠️ Flutter should NEVER directly talk to blockchain node.
Always go through backend.

2️⃣ API Gateway Layer

This is your central control.

Recommended:

👉 **FastAPI (Python)**

Why:

- Fast
- Async
- AI integration friendly
- Clean documentation (Swagger auto-generated)

Alternative:

- Node.js (if your team prefers JS)

Since you're already working with AI and PyTorch,
FastAPI is better.

3️⃣ Core Backend Services

Use microservice-like modular structure.

Core Services:

Service	Purpose
Auth Service	OTP + JWT + role management
Identity Service	Aadhaar hash, KYC validation
Product Service	Batch creation & lifecycle
Tracking Service	Custody updates
AI Service	Prediction + anomaly detection
QR Service	Generate & validate QR
Notification Service	SMS / Push

4️⃣ Database Stack

Use Hybrid.

PostgreSQL

For:

- User profiles
- Role data
- Metadata
- Business info

Redis

For:

- Session cache
- OTP storage
- Rate limiting

5️⃣ Blockchain Layer

Now the important decision.

🔒 Use Hyperledger Besu (Private Ethereum)

Why:

- Ethereum compatible
- Supports permissioned network
- Enterprise grade
- You already understand Ethereum

Setup:

- 5 Validator Nodes
- IBFT 2.0 consensus
- Role-based access control

Smart Contract Development

Language:
👉 Solidity

Framework:
👉 Hardhat

Testing:
👉 Mocha + Chai

Security:
👉 Slither + Mythril static analysis

6️⃣ Document Storage

Never store documents on chain.

Use:

IPFS

Store:

- Land certificate
- Business license
- Batch metadata JSON

Blockchain stores:

- IPFS CID
- Hash

7️⃣ AI Layer

This is where AGRICHAIN becomes powerful.

Use:

PyTorch

For:

- Price prediction
- Spoilage detection
- Crop disease detection (image model)

HuggingFace Transformers

For:

- Document fraud detection
- OCR validation
- Language translation

AI Deployment

Use:

- FastAPI endpoints
- Run models via TorchServe
- Or Docker containerized AI microservice

---

8 Aadhaar Verification

Important:

Never store Aadhaar number.

Instead:

- Use UIDAI verification API (if allowed)
- Store only hash + verification flag

For MVP:

- Use OCR + checksum validation
- Hash value
- Manual review flag

9️⃣ QR System

Use:

QR contains:

agrichain://batch/{batchId}

Backend:

- Fetch blockchain logs
- Fetch metadata from IPFS
- Return structured product history

Never expose RPC endpoint.

🔟 Translation / Language Automation

Instead of static translations:

Use:

Option 1 (Best):

Meta NLLB model (offline multilingual AI)

Option 2:

Google Cloud Translation API

Option 3:

Open-source Argos Translate (offline lightweight)

Best scalable option:
Store content keys in DB + use translation API fallback.

1️⃣1️⃣ DevOps & Deployment

Now we build serious infra.

Containerization:

Docker for:

- Backend
- AI
- Blockchain nodes
- IPFS

Orchestration:

Kubernetes (if scaling)
Otherwise:
Docker Compose for MVP

Cloud:

For MVP:

- AWS EC2 or DigitalOcean

For scaling:

- AWS EKS
- Managed PostgreSQL
- Managed Redis

---

12 Security Stack

Must include:

- JWT Authentication
- Role-based access control
- Smart contract access modifier
- Rate limiting
- IP whitelisting for validator nodes
- HTTPS only

---

13 Event Architecture

Use:

Kafka or RabbitMQ

For:

- Blockchain event listeners
- AI triggers
- Notification triggers

⚠️ What You Should NOT Use

- Firebase as main backend
- Public Ethereum mainnet
- Direct blockchain calls from Flutter
- Store documents on-chain

📈 Development Phases

Phase 1 – Core MVP

- Private chain
- Basic batch tracking
- QR view
- KYC minimal

Phase 2 – AI Integration

- Price prediction
- Fraud detection

Phase 3 – Intelligence Layer

- Trust score
- Spoilage alerts
- Demand forecasting

Phase 4 – Governance & Scaling

- Multi-validator
- Anchor hash to public Ethereum

Roles for Each

Define Roles Clearly (No Overlap Chaos)

With 5 members, here's the optimal split:

1️⃣ **Blockchain Engineer**

- Smart contracts
- Besu setup
- Node configuration
- Event listeners

2️⃣ **Backend Engineer**

- FastAPI
- PostgreSQL
- Redis
- Auth & Role management

3️⃣ **AI Engineer**

- Price prediction
- OCR verification
- Fraud detection
- Model APIs

4️⃣ **Mobile Engineer (Flutter)**

- UI/UX
- State management

- API integration
- QR scanning

5️⃣ **DevOps / Infra Engineer**

- Docker
- Deployment
- CI/CD
- Monitoring
- Security hardening

Tech Stack

agrichain/

├── mobile/ (Flutter)

├── backend/ (FastAPI)

├── blockchain/ (Hardhat + Solidity)

├── ai-service/ (future)

├── infra/ (Docker configs)

STACK SUMMARY

Layer	Tech
Mobile	Flutter
Backend	FastAPI
AI	PyTorch + Transformers
Blockchain	Hyperledger Besu
Contracts	Solidity + Hardhat
DB	PostgreSQL
Cache	Redis
Storage	IPFS
Queue	Kafka
Container	Docker
Cloud	AWS

🏗️ PHASE 1 — CORE INFRASTRUCTURE (Foundation)

This phase builds your backbone.

1️⃣ Private Blockchain Setup

- Hyperledger Besu
- 4–5 validator nodes
- IBFT 2.0 consensus
- RPC restricted to backend only

Deliverable:
✅ Stable running private Ethereum network

2️⃣ Identity Smart Contracts

Contracts to build:

- IdentityRegistry.sol
- RoleManager.sol

Features:

- Register user
- Assign role
- Verify KYC flag
- Role-based access modifier

Deliverable:
✅ Users can be registered and assigned roles securely

3️⃣ Backend Auth System

Implement:

- Phone OTP login
- JWT access tokens
- Role-based middleware
- Aadhaar hash storage (NOT raw number)

Deliverable:
✅ Secure login + role-based API access

🧱 PHASE 2 — PRODUCT LIFECYCLE (Core Value)

Now we introduce actual supply chain.

4️⃣ ProductBatch Smart Contract

Features:

- Create batch
- Transfer custody
- Store IPFS metadata CID
- Emit events

No AI yet.
No fancy dashboards.
Just correct blockchain lifecycle.

Deliverable:
✅ Batch moves from Farmer → Distributor → Retailer

5️⃣ QR Tracking System

QR contains:

agrichain://batch/{batchId}

Backend:

- Reads blockchain logs
- Fetches IPFS metadata
- Returns structured tracking timeline

Deliverable:
✅ Consumer can scan and view history

Now you have a working blockchain supply chain.

🧠 PHASE 3 — AI ENHANCEMENT

Now we add intelligence.

6️⃣ AI Service Layer

Separate microservice.

Start small:

- Price prediction model
- Basic anomaly detection

Later:

- Crop disease detection (image)
- Spoilage risk prediction

AI writes:

- Result hash to chain
- Detailed result in DB

Deliverable:
✅ AI-enhanced dashboard for farmer

🌟 PHASE 4 — TRUST & REPUTATION

This is where AGRICHAIN becomes powerful.

7️⃣ Trust Score System

Score based on:

- On-time delivery
- Spoilage rate
- Verification status
- Dispute history

Store:

- Score in backend
- Hash on chain

Deliverable:
✅ Reputation visible in dashboards

🔐 PHASE 5 — GOVERNANCE & SCALING

8️⃣ Add Multi-Signature Approvals

For:

- Dispute resolution
- Batch rejection

9️⃣ Public Chain Anchor

Once per day:

- Take hash of all transactions
- Anchor to Ethereum public network

This provides:

- Tamper proof assurance

Adhaar

Purpose Of Aadhaar Verification?

Be honest:

Aadhaar verification is NOT for blockchain.
It is NOT for tracking.
It is NOT for consumers.

It is for:

✅ Identity assurance

✅ Fraud prevention

✅ Legal accountability

✅ Government integration readiness

If a farmer uploads fake land,
or a distributor disappears after fraud,
you must know who they are.

🚨 But Important:

You should NOT store Aadhaar number.
You should NOT store raw PDF.
You should NOT put it on blockchain.

You store:

- Verification status (true/false)
- Hash of ID + salt
- Verification timestamp

That's it.

👨‍🌾 Farmer — Yes

Because:

- Land ownership
- Subsidy eligibility
- Certification

🚚 Distributor — Yes

Because:

- Logistics responsibility
- Liability

🏪 Retailer — Yes

Because:

- Business license validation

🛒 Consumer — Optional

Unless:

- You're issuing digital wallet
- Or government-linked benefits

Don't overcomplicate consumer onboarding.

Clean Strategy For MVP

Here's what I suggest:

Farmer

- Aadhaar format validation
- Pre-approved land certificate binding

Distributor

- GST format validation
- Pre-approved license binding

Retailer

- GST / FSSAI format validation
- Pre-approved license binding

All approvals simulated via GovernmentAuthority smart contract.

Tab 5

🔵 PART A — FULL BESU NETWORK SETUP (STEP-BY-STEP)

We are building a **Permissioned IBFT 2.0 network with 4 validators**.

For MVP you can run all locally.
For production you distribute across servers.

🧱 STEP 1 — Install Besu

On Linux / WSL / Ubuntu server:

sudo apt update
sudo apt install openjdk-17-jdk -y

Download Besu:

wget https://hyperledger.jfrog.io/artifactory/besu-binaries/besu/24.1.1/besu-24.1.1.tar.gz
tar -xvf besu-24.1.1.tar.gz
cd besu-24.1.1

Verify:

./bin/besu --version

🧱 STEP 2 — Create Validator Nodes

Create 4 folders:

mkdir validator1 validator2 validator3 validator4

Generate key pairs:

./bin/besu public-key export-address --node-private-key-file=validator1/key

Repeat for all validators.

Save:

- node key
- public address

🧱 STEP 3 — Create Genesis File (IBFT 2.0)

Create `genesis.json`:

Important fields:

- consensus: ibft2
- blockperiodseconds: 2
- epochlength: 30000
- gasLimit
- initial validator list

Example structure (conceptual):

```
{
 "config": {
   "chainId": 1337,
   "ibft2": {
     "blockperiodseconds": 2,
     "epochlength": 30000
   }
 },
 "gasLimit": "0x1fffffffffffff",
 "alloc": {}
}
```

Use Besu tool to generate extraData for validators.

🧱 STEP 4 — Start Validator Nodes

Each node:

```
./bin/besu \
 --data-path=validator1 \
 --genesis-file=genesis.json \
 --rpc-http-enabled \
 --rpc-http-api=ETH,NET,WEB3 \
 --host-allowlist="*" \
 --rpc-http-port=8545 \
 --p2p-port=30303
```

Each validator uses different ports.

Once started:

✔ Blocks should begin producing
✔ `eth.blockNumber` increases

If blocks increase every 2 seconds → network healthy.

🧱 STEP 5 — Lock Down RPC

For production:

- Disable public RPC
- Restrict by IP
- Use reverse proxy (NGINX)
- Enable TLS

🏛️ WHY THIS NETWORK IS STRONG

Because:

- Multi-validator
- Byzantine Fault Tolerant
- Permissioned
- Deterministic finality
- No mining

This is enterprise-grade.

🔵 PART B — SMART CONTRACT ARCHITECTURE (PRODUCTION READY)

Now we design contracts cleanly.

We implement modular, upgrade-friendly structure.

📁 Hardhat Setup

Inside `blockchain/`:

npm init -y
npm install --save-dev hardhat
npx hardhat

Install:

npm install @openzeppelin/contracts dotenv

Configure `hardhat.config.js` to connect to Besu RPC.

🧱 CONTRACT 1 — IdentityRegistry.sol

Purpose:
Manage user roles & verification flags.

Core features:

- registerUser(address, role)
- setVerified(address, bool)
- getRole(address)
- onlyGovernment modifier

Roles enum:

- FARMER
- DISTRIBUTOR
- RETAILER
- GOVERNMENT
- AUDITOR

🧱 CONTRACT 2 — CertificateRegistry.sol

Purpose:
Approve certificates before binding.

Stores:

- aadhaarHash → docHash
- wallet → docHash binding

Functions:

- approveCertificate(bytes32 aadhaarHash, bytes32 docHash)
- bindCertificate(address wallet, bytes32 aadhaarHash)

Enforces:

- Only government can approve
- Aadhaar can bind once
- Wallet can bind once

🧱 CONTRACT 3 — ProductBatch.sol (CORE)

Stores:

```
struct Batch {
  uint256 batchId;
  address currentOwner;
  uint256 quantity;
  bytes32 metadataHash;
  uint256 parentBatchId;
  bool active;
  uint256 createdAt;
}
```

Functions:

- createBatch(uint quantity, bytes32 metadataHash)

- transferBatch(uint batchId, address newOwner)
- splitBatch(uint batchId, uint splitQuantity)

Rules:

✔ Only owner can split
✔ Quantity reduces
✔ Cannot oversplit
✔ Auto close when quantity = 0

Emit events for backend indexing.

🧱 CONTRACT 4 — AuditTrail.sol (Phase 2)

Stores:

- AI risk flags
- Recall status
- Compliance markers

Does not store heavy data.
Only hashes.

🧱 CONTRACT 5 — TrustScore.sol (Phase 3)

Stores:

- trustScore snapshot

- lastUpdated

Score calculation happens off-chain.

🧱 CONTRACT 6 — PublicAnchor.sol (Future)

Daily:
Backend calculates Merkle root
Stores hash on chain

Later:
Anchors that hash to Ethereum mainnet.

🔁 CONTRACT DEPLOYMENT ORDER

1. IdentityRegistry
2. CertificateRegistry (needs Identity address)
3. ProductBatch (needs Identity)
4. AuditTrail
5. TrustScore

Save addresses in config file.

🔗 BACKEND BLOCKCHAIN GATEWAY

Use Python `web3.py`.

Responsibilities:

- Sign transactions
- Call contract functions
- Listen to events
- Store event data in PostgreSQL

Never let frontend interact directly.

📡 EVENT LISTENER SERVICE

Backend worker subscribes to:

- BatchCreated
- BatchSplit
- BatchTransferred

Builds relational lineage table.

📱 FRONTEND FLOW

User action →
Flutter →
Backend →
Blockchain →
Event →
Backend DB →
Response →
Flutter UI

Clean separation.

🔐 FUTURE GOVERNANCE UPGRADE

Later:

- MultiSigDispute contract
- Freeze batch function
- Slashing logic
- Public anchor integration

Your architecture supports this.

Blockchain Configuration

ibftConfig.json :

```
{
  "genesis": {
    "config": {
      "chainId": 1337,
      "ibft2": {
        "blockperiodseconds": 2,
        "epochlength": 30000
      }
    },
    "gasLimit": "0x1fffffffffffff"
  },
  "blockchain": {
    "nodes": {
      "generate": true,
      "count": 4
    }
  }
}
```

To Create the Network with N nodes :

bin\besu.bat operator generate-blockchain-config
--config-file=E:\besu\besu-network\ibftConfig.json --to=E:\besu\besu-network\generated
--private-key-file-name=key

To start each validator :
Validator 1 :
bin\besu.bat --data-path=E:\besu\besu-network\validator1\data
--genesis-file=E:\besu\besu-network\generated\genesis.json
--node-private-key-file=E:\besu\besu-network\validator1\key --rpc-http-enabled
--rpc-http-api=ETH,NET,WEB3,IBFT --rpc-http-port=8545 --p2p-port=30303 --host-allowlist="*"

Validator 2 :
bin\besu.bat --data-path=E:\besu\besu-network\validator2\data
--genesis-file=E:\besu\besu-network\generated\genesis.json
--node-private-key-file=E:\besu\besu-network\validator2\key --rpc-http-enabled
--rpc-http-api=ETH,NET,WEB3,IBFT --rpc-http-port=8546 --p2p-port=30304 --host-allowlist="*"

Validator 3 :
bin\besu.bat --data-path=E:\besu\besu-network\validator3\data
--genesis-file=E:\besu\besu-network\generated\genesis.json
--node-private-key-file=E:\besu\besu-network\validator3\key --rpc-http-enabled
--rpc-http-api=ETH,NET,WEB3,IBFT --rpc-http-port=8547 --p2p-port=30305 --host-allowlist="*"

Validator 4 :
bin\besu.bat --data-path=E:\besu\besu-network\validator4\data
--genesis-file=E:\besu\besu-network\generated\genesis.json
--node-private-key-file=E:\besu\besu-network\validator4\key --rpc-http-enabled
--rpc-http-api=ETH,NET,WEB3,IBFT --rpc-http-port=8548 --p2p-port=30306 --host-allowlist="*"

To verify the count of the Peers :
curl -X POST http://127.0.0.1:8545 -H "Content-Type: application/json" -d
"{\"jsonrpc\":\"2.0\",\"method\":\"net_peerCount\",\"params\":[],\"id\":1}"

Invoke-RestMethod -Method Post -Uri http://127.0.0.1:8545 -Body
'{"jsonrpc":"2.0","method":"net_peerCount","params":[],"id":1}' -ContentType
"application/json"

Tab 7

🧱 PHASE 1 — CLEAN IBFT MULTI-VALIDATOR NETWORK (LOCAL)

We will:

1. Remove old Clique setup
2. Create 4 validator nodes
3. Generate proper IBFT genesis
4. Start all nodes
5. Verify block production

Follow carefully.

🔥 STEP 0 — Clean Previous Network

Delete your old `data` folder completely.

For example:

E:\besu\besu-network\data

Remove everything inside.

We start fresh.

🧱 STEP 1 — Create New Network Structure

Inside:

E:\besu\besu-network\

Create:

validator1
validator2
validator3
validator4
genesis

So structure becomes:

E:\besu\besu-network\
├── validator1\
├── validator2\
├── validator3\
├── validator4\
└── genesis\

🔑 STEP 2 — Generate Validator Node Keys

Inside each validator folder:

Run (from besu root):

bin\besu.bat public-key export-address
--node-private-key-file=E:\besu\besu-network\validator1\key

Repeat for:

- validator2
- validator3
- validator4

For each node you will get:

- A `key` file (private node key)

- A printed public address

Save all 4 public addresses.
We need them.

🧾 STEP 3 — Prepare IBFT Validators File

Inside:

E:\besu\besu-network\genesis\

Create file:

validators.json

Put all 4 validator addresses in array format:

```
[
 "0xValidatorAddress1",
 "0xValidatorAddress2",
 "0xValidatorAddress3",
 "0xValidatorAddress4"
]
```

(Replace with actual addresses.)

🏗️ STEP 4 — Generate IBFT extraData

Run:

bin\besu.bat rlp encode --from=E:\besu\besu-network\genesis\validators.json

This will output a long hex string.

Copy that.

That is your `extraData`.

📜 STEP 5 — Create IBFT Genesis File

Create:

E:\besu\besu-network\genesis\genesis.json

Use this template:

```
{
 "config": {
   "chainId": 1337,
   "ibft2": {
     "blockperiodseconds": 2,
     "epochlength": 30000
   }
 },
 "difficulty": "0x1",
 "gasLimit": "0x1fffffffffffff",
 "extraData": "PASTE_GENERATED_EXTRADATA_HERE",
 "alloc": {}
}
```

Paste the generated extraData exactly.

🌐 STEP 6 — Create Static Nodes File

We need nodes to discover each other.

Start validator1 temporarily:

```
bin\besu.bat ^
--data-path=E:\besu\besu-network\validator1 ^
--genesis-file=E:\besu\besu-network\genesis\genesis.json ^
--rpc-http-enabled ^
--rpc-http-api=ETH,NET,WEB3,IBFT ^
--p2p-port=30303 ^
--rpc-http-port=8545
```

When started, copy the enode URL from logs.

Stop it.

Repeat for each validator to get their enode URLs.

Now create:

static-nodes.json

Inside each validator folder.

Content:

```
[
 "enode://node1",
 "enode://node2",
 "enode://node3",
 "enode://node4"
]
```

Each validator must have same static-nodes.json.

🚀 STEP 7 — Start All Validators

Start them in 4 separate terminals.

Validator1:

```
bin\besu.bat ^
--data-path=E:\besu\besu-network\validator1 ^
--genesis-file=E:\besu\besu-network\genesis\genesis.json ^
--rpc-http-enabled ^
--rpc-http-api=ETH,NET,WEB3,IBFT ^
--rpc-http-port=8545 ^
--p2p-port=30303
```

Validator2:

Use different ports:

- rpc-http-port=8546

- p2p-port=30304

Validator3:

- rpc-http-port=8547
- p2p-port=30305

Validator4:

- rpc-http-port=8548
- p2p-port=30306

✅ STEP 8 — Verify Network Is Working

Open one RPC endpoint:

Check block number increasing:

Using Python or curl:

eth_blockNumber

Block number should increase every 2 seconds.

If yes:

🎉 Your IBFT multi-validator network is running.

⚠️ VERY IMPORTANT CHECK

Run:

ibft_getValidatorsByBlockNumber("latest")

It should return all 4 validator addresses.

If it does → network healthy.

🧠 Once This Works

Then we move to:

1. Hardhat configuration for Besu RPC
2. Deploy IdentityRegistry
3. Deploy CertificateRegistry
4. Deploy ProductBatch
