ChatGPT

Immutable Academic Credentials Verification System Project Plan

Introduction and Objectives

Academic credential fraud is a significant problem, with forged diplomas and transcripts undermining trust in education [1]. The goal of this project is to build an **Immutable Academic Credentials Verification System** using blockchain technology to issue and verify diplomas, certificates, and transcripts. By leveraging a blockchain's **tamper-proof ledger**, we ensure credentials cannot be faked or altered, and their authenticity can be easily verified by anyone with access to the ledger [2]. Traditional verification relies on centralized databases prone to single points of failure and lack transparency; in contrast, a distributed ledger provides an **immutable, tamper-proof** record of academic credentials [2]. For example, the University of Nicosia has issued all diplomas via blockchain since 2017 to eliminate credential fraud [3], demonstrating the practicality of this approach.

Project Scope: We will implement a private, permissioned blockchain network in Java (for both backend logic and a Swing GUI frontend). The system will use a *Proof of Authority (PoA)* consensus mechanism, meaning only pre-approved validator nodes (e.g. authorized institutions) can add new blocks to the chain. This design offers fast, low-cost transactions by relying on known, trusted validators instead of energy-intensive mining [4]. Validators will be **hardcoded** (a fixed list of known nodes configured in the network settings) to simplify consensus management. The blockchain will operate in a peer-to-peer (P2P) network of nodes so that every node maintains a copy of the ledger and communicates directly with others with no central server [5]. Each credential issued (containing fields such as student name, degree, graduation date, institute name, and student ID) will be recorded as a transaction in a new block. The system will provide two main functions via the GUI: - **Issue Credential:** An authorized node (validator) can input a student's details and issue a certified credential onto the blockchain. - **Verify Credential:** Any user can query the blockchain (via any node's GUI) to confirm the existence and authenticity of a credential (by searching for a student ID or certificate hash, etc.), and view the relevant record on the ledger.

Security is ensured through cryptography: each block will include a cryptographic hash of its content and the previous block's hash, forming an immutable chain (any tampering breaks the hash link). Additionally, credentials issued by a node will be digitally signed using the node's private key to prove they came from a valid authority. Only nodes holding one of the **hardcoded validator keys** can create valid blocks, and all other nodes will reject blocks not signed by a known validator. This PoA approach based on verified identities maintains integrity without expensive mining [4] [6].

Below is a high-level architecture diagram of the system, showing multiple university nodes in a P2P network issuing and verifying academic credentials on a shared ledger, with a Swing GUI for user interaction at each node.

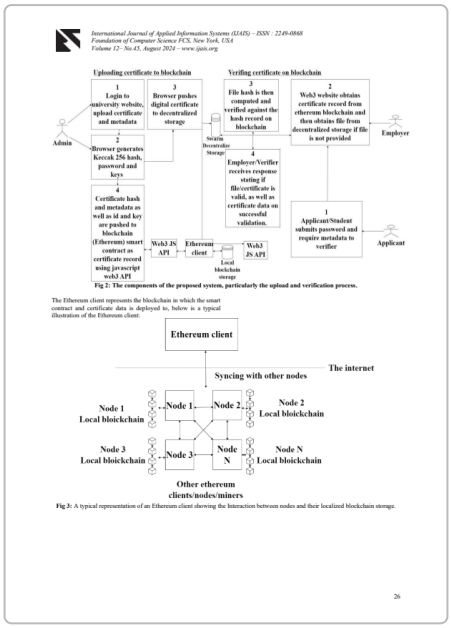

Fig. 1: Conceptual architecture of the academic credential blockchain system (multiple Java nodes in a P2P network with hardcoded validator nodes issuing blocks, and a GUI for issuing and verifying certificates).

System Design Overview

Blockchain Structure: The blockchain will consist of a series of blocks linked cryptographically. Each block contains a **block header** (including attributes like index/height, timestamp, the previous block's hash, and the current block's hash) and a **payload** with the credential data. Storing the previous block's hash in each new block ensures immutability – any change in an earlier block would invalidate all subsequent block hashes [7] [8]. We will use SHA-256 hashing for creating block hashes, a standard choice for strong cryptographic security in blockchains [9]. The first block is the **genesis block**, which will be hardcoded and contains no credentials (just a placeholder).

Each credential record (transaction) will include the student's Name, Degree, Graduation Date, Institution, and Student ID. Optionally, we can include a unique Certificate ID or a hash of a full certificate file if needed for verification purposes (similar to MIT's approach of hashing a digital certificate and storing that hash on-chain [10]). In our simplified design, we will record the fields directly on-chain for transparency, but sensitive data can be hashed or encrypted in future iterations for privacy.

Consensus – Proof of Authority: In this private network, we choose PoA consensus to allow only designated authorities to add blocks. In PoA networks, a fixed set of validators is approved in advance to secure the blockchain [11]. Our implementation will include a list of validator nodes (identified by a public key or node ID) hardcoded into the system configuration. Only these nodes can initiate the addition of new blocks (i.e., issue credentials). When a validator node creates a block, it will cryptographically sign the block. Other nodes will verify the signature against the known validator list and accept the block if valid. Blocks from non-authorized sources are ignored. This mechanism is efficient and fast since there's no mining competition – validators may follow a round-robin schedule to produce blocks in turn [12], or simply add blocks whenever they have a new credential to record (assuming low likelihood of collisions). Because validators are known entities staking their reputation, PoA provides trust and accountability in lieu of decentralization [4].

Peer-to-Peer Network: The blockchain nodes will form a P2P network over which they broadcast new blocks and synchronize the ledger. Each node will know the network addresses of the others (we can

hardcode a list of peer addresses for simplicity). When a node issues a new credential (creates a block), it will serialize that block and send it to all peers. Upon receiving a block, a node validates it (checks the previous hash matches its latest block, verifies the signature and data integrity) and if valid, appends it to its local chain. P2P communication ensures every node eventually has a consistent copy of the ledger [5]. We will implement simple messaging (using Java sockets or similar) for broadcasting blocks and perhaps requesting missing blocks when a node comes online (basic synchronization on startup).

User Interface (Swing GUI): Each node runs a desktop GUI application (Java Swing) that allows the user to interact with the blockchain functions: - The **Issuance interface** will have a form to input credential details (name, degree, etc.) and an "Issue Credential" button. Only users running a validator node should use this function (we may implement a simple role check or just assume only authorized personnel run validator nodes). When submitted, the node creates a new block with the entered data, links it to the chain, signs it, and broadcasts it to peers. - The **Verification interface** allows anyone (even on a non-validator node) to query the blockchain. This could be a search by student ID or scanning a QR code (if we generate one containing a credential ID). The GUI will then display whether a matching credential is found on-chain and its details. Users can also browse the entire blockchain via the GUI (e.g., view a list of all blocks/credentials) to audit the records. The GUI will parse the blockchain data into human-readable form for convenience.

Hardcoded Validators: The validator nodes (e.g., specific universities or the project administrators) will be identified by their public keys. We will generate a few key pairs for the validators during development and include their public keys in all nodes' code or configuration. This way, every node knows which signatures to trust for block creation. Initially, adding or removing validators will require code changes (since they're hardcoded), which is acceptable for a prototype. This simplifies development at the cost of flexibility.

Throughout development, we will adhere to best practices of blockchain systems such as input validation (to prevent malformed data), basic security (ensuring private keys stay secure, and network channels are not easily exploited), and data persistence (each node likely will save the blockchain to disk so that it can restart without losing data).

Below is a breakdown of the project into 12 weekly sprints, each with specific deliverables and milestones:

Sprint 1: Requirements Gathering & Research (Week 1)

- **Define Requirements:** Gather all functional requirements – the system must allow authorized issuance of credentials and public verification. Specify data fields for credentials (Name, Degree, Date, Institute, Student ID, etc.) and any formatting rules. Define user roles (issuer vs verifier) and their permissions.
- **Research Existing Solutions:** Conduct research on academic credentials on blockchain to draw insights. For example, review frameworks like Blockcerts (MIT's digital certificates) and other academic blockchain projects [13] [3]. Identify how those systems issue and verify certificates, and note features we might incorporate (e.g., using hashes, QR codes, etc.).
- **Technology Stack Decisions:** Confirm that Java will be used for both backend and frontend. Decide on any libraries or frameworks: for instance, use Java's built-in networking (sockets) for P2P, Swing for GUI, and standard crypto libraries for hashing/signing. Ensure all team members have the development environment set up (JDK, IDE).
- **Proof of Authority Plan:** Plan how to implement PoA in our context. Decide how many validator nodes we will simulate (e.g., 3 hardcoded validator nodes for the prototype) and how to identify

them (probably via public keys or simple IDs). Document the consensus approach (e.g., "any validator can add a block at will; others accept if valid" vs. a more coordinated round system). Simplicity will be prioritized due to time constraints.
- **Deliverables:** A project requirements document and a high-level design outline. This should include use-case descriptions (issuing and verifying process) and a draft of the system architecture diagram (to be refined in Sprint 2). We will also list the validator identities we plan to use. Team should have a clear understanding of what will be built.

Sprint 2: System Design & Architecture (Week 2)

- **Data Structure Design:** Design the core data structures: define a `Block` class (with fields: index, timestamp, previousHash, hash, issuerSignature, credentialData, etc.) and a `Blockchain` class (likely managing a list of Block objects and providing methods to add and validate blocks). Define a `Credential` or `Transaction` structure to hold the certificate fields within the block. Decide how the block hash is computed (concatenate relevant fields and hash using SHA-256).
- **Digital Signature Scheme:** Choose a cryptographic algorithm for signatures (e.g., ECDSA with a known curve, or RSA). Plan how each validator will sign a block – e.g., include a `validatorId` and `signature` in the block. Design the format of the signature (likely sign the block's content hash). Determine how public keys will be distributed (since they are hardcoded, perhaps store them in a config file or within the code of each node).
- **Networking Protocol:** Design the message protocol for P2P communication. Decide on message types, e.g., `NEW_BLOCK` broadcasts, `CHAIN_REQUEST` (to sync missing data), etc. Outline how peers will connect (maybe each node has a list of peer addresses). Because it's a small network, a simple approach like each node directly connecting to all others (full mesh) on startup and listening on a socket for incoming blocks is sufficient.
- **GUI Design:** Sketch the GUI layout and components in Swing. Plan the screens/panels needed: e.g., one panel for issuing credentials (form inputs and submit button), another for viewing the blockchain (could be a table listing blocks), and a search/verification panel (input a student ID or certificate hash and show results). Also include displays for status (e.g., connection status, number of blocks).
- **Draw.io Architecture Diagram:** Create a **system architecture diagram** illustrating the components and their interactions. This diagram should show the network of nodes (validators and possibly non-validator peers), the data flow when a credential is issued (node signing and broadcasting a new block to others), and the process of verification (a user querying a node for a record). Indicate the Swing GUI at each node and how it interfaces with the node's blockchain module. This diagram will be refined and finalized in this sprint [14].
- **Deliverables:** A detailed design document including class diagrams or descriptions for `Block` and other classes, the network protocol specification, and wireframes or sketches of the GUI. The **draw.io architecture diagram** mapping out the overall system is a key deliverable this week. Team should review and agree on this design before implementation starts.

Sprint 3: Blockchain Core Implementation (Week 3)

- **Block & Chain Classes Coding:** Implement the `Block` class in Java. Include methods to calculate its hash (e.g., a method `computeHash()` that generates the SHA-256 hash of the block's contents). Implement the `Blockchain` class with an in-memory list of blocks. Add a method `addBlock(Block newBlock)` that appends a block after validating it (checking the previous hash matches the last block's hash, etc.).

- **Genesis Block:** Create a genesis block at network initialization. Hardcode it with a predefined previousHash (e.g., "0" or some constant) and no credential data. Ensure all nodes start with the identical genesis block in their chain.
- **Hash Linking and Verification:** Ensure that when adding a new block, the code recalculates the block's hash and compares the stored `previousHash` value with the latest block's hash on the chain. This guarantees the chain's integrity. Provide a method to **validate the entire blockchain** (iterate through blocks, checking hash links and signatures) for debugging and future use.
- **Unit Tests:** Write unit tests or simple main-method tests for the blockchain structure. For example, create a small chain of 2-3 blocks locally and verify that tampering with a block's data causes a hash mismatch to be detected. This helps validate our immutability logic early.
- **Credential Data Handling:** Implement a simple container for credential info. For now, this can just be fields in the Block or a nested `Credential` object. Ensure that when we output or display a block, these fields are readable (e.g., override `toString()` for Block to show credential details neatly).
- **Deliverables:** Java classes for Block and Blockchain with hashing and linking logic completed. A demo (could be a simple console output) showing a few blocks chained together and the integrity check passing. This forms the foundation of the ledger.

Sprint 4: Implementing Proof of Authority & Security (Week 4)

- **Validator Identity Setup:** Create a class or configuration for **Validator** identities. For the prototype, generate a few key pairs (using Java's KeyPair generator). Assign each a unique ID or name (e.g., "Validator1 – University A"). Hardcode their public keys in an array or config file that every node can access. Distribute the corresponding private keys to the simulated nodes that will act as those validators (in practice, each validator node would securely store its own private key).
- **Block Signing:** Update the Block structure to include a `validatorId` (or public key) and a `signature` field. Implement signing: when a node (if it is a validator) creates a new block, it uses its private key to sign the block's hash or contents. Use a secure signature algorithm (e.g., ECDSA with SHA-256).
- **Signature Verification:** Implement verification logic that runs in `addBlock()`: when a block is received, check that the `validatorId` is in the known validator list, then verify the block's signature using the corresponding public key. Only accept the block if the signature is valid and the data/hashes are correct. If a block fails this check, reject it and possibly log a warning.
- **Consensus Rules:** Because we trust validators, we won't implement a complex consensus algorithm like PBFT in this timeframe. Instead, define simple rules: e.g., only one block can occupy a given index in the chain, and we always trust the first valid block for each index we receive (assuming validators won't maliciously fork the chain). If two blocks with the same index are ever seen (which is unlikely if validators coordinate or if only one issues at a time), the node could either reject the later one or choose by some rule (like lowest hash or first received). Document this approach. Our assumption is that validators are honest (a reasonable assumption in a permissioned consortium of universities).
- **Testing PoA Mechanism:** Write tests or simulate scenarios: Try to add a block with a fake validator signature and ensure it's rejected. Test that a valid validator's block is accepted. If possible, simulate two validator nodes adding blocks in sequence to ensure the chain remains consistent.
- **Deliverables:** Enhanced blockchain code with digital signature-based authority control. A short report on how the PoA consensus is implemented in code (for project documentation). At this point, the blockchain should only accept blocks from the hardcoded validators, fulfilling the PoA requirement [4] .

Sprint 5: Peer-to-Peer Network Development (Week 5)

- **Networking Setup:** Implement the P2P networking layer. This can be as simple as a TCP socket server thread on each node and client connections to peers. Decide on a port number for communication (or one per node). For each node instance, start a server socket to listen for incoming connections/messages.
- **Peer Discovery/Connection:** Hardcode a list of peer addresses (IP and port for each node) in a config file or within the code. When a node starts, it will attempt to connect to each peer in the list (except itself). Establish a simple handshake to register the connection. Because validators are few, a static list is manageable.
- **Message Protocol:** Define a message format, e.g., simple text or JSON messages. The key message types to support now:
- **New Block Broadcast:** When a validator node creates a block, it broadcasts a message containing the block data to all peers.
- **Chain Sync Request:** When a node comes online or if it detects it's missing some blocks, it can request the latest chain or a specific block from peers.
- Optionally, **ACKs** or receipts for reliability (or we keep it simple and assume UDP-like fire-and-forget for new blocks, relying on periodic sync).
- **Implement Broadcast:** Code the function that serializes a Block (to JSON or another format) and sends it to all connected peers. On the receiving side, parse the message, reconstruct the Block object, and invoke the blockchain's `addBlock()` to validate and append it.
- **Threading and Concurrency:** Ensure the network handling is on separate threads so that the GUI (to be built) remains responsive. Use synchronization where needed when accessing the blockchain from multiple threads (e.g., when a network thread receives a block at the same time a user might be adding a new block locally).
- **Basic Network Test:** Run multiple instances of the application (or create a simple test harness) on localhost to simulate nodes. For example, run Node A and Node B. Have Node A create a block; verify that Node B receives it and adds to its chain. Also test that Node B's chain persists or can be requested by Node A if A starts late. This will likely involve printing logs for blocks sent/received.
- **Deliverables:** A working P2P communication module integrated with the blockchain. A demonstration of two or more Java processes maintaining a shared blockchain state via the network. At this stage, we effectively have a rudimentary private blockchain network running.

Sprint 6: Node Interaction & Sync (Week 6)

- **Chain Synchronization:** Enhance the network protocol to handle node restarts or late joins. For instance, when a node starts, it should ask a peer for the latest blockchain if its own chain is empty or outdated. Implement a `REQUEST_CHAIN` message and a handler that responds with either the full chain or recent blocks. This ensures new or recovering nodes catch up to the current ledger.
- **Conflict Resolution:** Although our PoA approach minimizes conflicts, implement a simple strategy in case two valid chains are encountered (e.g., if two validator nodes issued blocks simultaneously and a network partition occurred). A common approach is **longest chain wins** [7]. We can use chain length (or total number of blocks) as a tiebreaker – the node will accept the chain which has more blocks. Since validators are cooperating, forks should be rare, but this provides resilience.
- **Error Handling:** Add error detection for network messages (e.g., bad format) and robust logging. If an invalid block is received, log an event and ignore it (ensuring a malformed or malicious message doesn't crash the node).

- **Performance Check:** Sending the entire chain for sync might be inefficient if the chain grows. Implement a basic optimization: maybe just send missing blocks (if the new node sends its last known index and the peer responds with blocks after that). Given 12 weeks, the chain won't be huge, but good to consider scalable practices.
- **Security Consideration:** Since this is a private network, we may not need encryption for P2P messages initially (especially all nodes are presumably within a controlled environment). But note that in a real deployment, TLS or message signing would be used to prevent MITM or rogue injections. Document this as a future enhancement.
- **Deliverables:** Improved networking that can gracefully handle nodes coming online/offline. Demonstration: if a node is launched after some blocks were created, it successfully syncs the missing blocks from a peer. The network should be robust by the end of this sprint, ensuring all nodes remain consistent.

Sprint 7: Credential Issuance Feature (Week 7)

- **GUI Implementation – Issuing Credentials:** Begin building the Swing GUI for issuing credentials. Create a form interface (JFrame or JPanel) with input fields for all credential data (Name, Degree, Date, Institute, Student ID). Provide validation for inputs (e.g., date format, required fields).
- **Issue Flow Integration:** When the user clicks "Issue" on the GUI, trigger the process in the backend: construct a new Block with the entered data, fill in block header fields (index = lastIndex+1, timestamp = now, previousHash = last block's hash). Then sign the block with the node's validator private key and add it to the local chain (which will run validation checks as well). After adding locally, broadcast the new block to peers.
- **Feedback to User:** The GUI should give feedback such as "Credential issued successfully! Block #X added to blockchain." or display any errors (e.g., "You are not authorized to issue" if by chance a non-validator tries, or validation errors if fields are wrong).
- **Prevent Duplicate Entries:** Decide how to handle if the same student ID is issued twice. Possibly allow multiple credentials (like multiple degrees) but ensure each is separate. However, if trying to issue the exact same degree twice, we might warn the user. This can be a simple check in the UI layer.
- **Store Credentials Locally:** Ensure that once a credential is issued, it's saved as part of the blockchain data structure and ideally persisted to disk. We might implement a simple file save (serialize the blockchain or new blocks to a file) after each addition so that restarting the node keeps past data. This persistence can be in JSON, binary, or a simple text log of blocks.
- **Testing Issuance:** Simulate an issuing scenario: Use the GUI on a validator node to issue a couple of sample credentials. Verify the blocks appear on that node's chain and also propagate to other nodes' chains. Check that fields are correctly recorded and displayed.
- **Deliverables:** A functional "Issue Credential" GUI component integrated with the backend. By now, an authorized user (e.g., an admin at a validator node) can add new credential records to the blockchain through a user-friendly interface. This fulfills the primary data entry part of the project.

Sprint 8: Credential Verification Feature (Week 8)

- **GUI Implementation – Verification:** Create an interface in the Swing GUI for verifying credentials. This could be a search bar or form where a verifier (e.g., employer or student) enters a query, such as a Student ID or a Certificate ID. For simplicity, we might use Student ID as the key lookup (assuming each student ID is unique per credential, though a student might have multiple credentials – in that case, the system can list all matches).

- **Search Logic:** Implement a function in the backend to search the blockchain for a record. This could iterate over all blocks (or maintain an index mapping IDs to blocks for efficiency). If a matching credential is found, prepare the data for display. If not found, return a “not found” result.
- **Verification Criteria:** When displaying the result, the node should also **verify the blockchain integrity and signature** of that credential to reassure the user. Essentially, since our nodes maintain only valid chains, any found record is already verified. But we can explicitly re-verify the block’s hash link and signature at lookup time to double-check authenticity (and show a green check mark or similar indicating “Verified Authentic”).
- **Display Results:** Design the output panel to show credential details clearly (e.g., "Credential Found: John Doe, B.Sc. in Computer Science, 2025, University X, ID:12345 – *Verified on Blockchain*"). If multiple records match (like multiple degrees for the same ID), list them in a list or table. If none, show "No record found. The credential is not verified on the blockchain."
- **Chain View:** Additionally, implement a **Blockchain Viewer** in the GUI that allows any user to browse all blocks. This could simply display a scrollable list of blocks (with their index, student name, degree, etc.). This feature enhances transparency and is useful for debugging/teaching. Ensure this list updates when new blocks come in via network.
- **Testing Verification:** Use the GUI to lookup credentials that were issued in Sprint 7. Test both existing and non-existing IDs. Also test on a non-validator node’s GUI to ensure even a regular node can perform verification (since the ledger copy is there). All nodes in the network should give the same verification results.
- **Deliverables:** Completed verification functionality in the GUI. At this point, we have a user-friendly way for anyone to confirm credential authenticity by querying the blockchain, fulfilling the primary verification requirement of the project.

Sprint 9: Refinement and UX Improvements (Week 9)

- **Improve GUI Usability:** Refine the layout and aesthetics of the Swing GUI. Ensure the input forms are clear and labels are properly aligned. Perhaps add menus or tabs to switch between “Issue” and “Verify” sections. Make sure long lists (like the blockchain view) are scrollable and the window is resizable for convenience.
- **Input Validation & Error Messages:** Add more robust validation for data fields (e.g., check that date is a valid date, student ID is numeric or fits expected format, etc.). Provide user-friendly error messages or highlight fields if input is invalid. Also, handle corner cases like missing fields.
- **Confirmation and Logging:** After issuing a credential, maybe prompt for confirmation "Are you sure you want to issue this credential to blockchain?" to avoid mistakes, as blockchain entries are permanent. Maintain a log area in the UI to show recent actions or network events (e.g., "Received new block #5 from Validator2").
- **Concurrency and Performance:** Check that the GUI remains responsive during network operations. If needed, use SwingWorker or background threads for any tasks that might pause the UI (like reading a large chain file, etc.). The dataset is small now, but good to ensure responsiveness.
- **Persistency Improvements:** If not done, implement saving the blockchain state to disk at graceful shutdown or periodically. Conversely, load the existing blockchain from disk on startup before connecting to peers (then sync any new blocks). This ensures even if all nodes go offline, the data isn’t lost.
- **Security Considerations:** Add a simple authentication for the issuance function on the GUI. Since in a real scenario only authorized staff should issue credentials, we might add a login prompt or at least a confirmation password for issuing. This could be as simple as a hardcoded password or a role flag when launching the app (e.g., launch with a “--validator” flag that enables issuing features).

- **Deliverables:** A polished version of the application with improved UI/UX and reliability. Screenshots of the GUI and a brief user guide (how to issue, how to verify) can be prepared for documentation. The system should feel more user-friendly and robust after this sprint.

Sprint 10: Testing and Quality Assurance (Week 10)

- **Integration Testing:** Now conduct thorough testing of the entire system in various scenarios:
- **Multiple Nodes Test:** Run at least three node instances (including at least two validators) simultaneously. Issue several credentials from different validator nodes and ensure all nodes receive and append those blocks correctly.
- **Verification Consistency:** Check that a credential issued on one node is verifiable on all other nodes' GUIs (consistency of data). Also test verification by someone who joins the network later (to ensure sync doesn't miss it).
- **Malicious Scenario:** (If possible) simulate or imagine a malicious node trying to inject a bad block. Ensure that our signature check and validator whitelist prevent any unauthorized block from entering the chain [4] [6]. This might be done via a custom test where we craft a fake block.
- **Tamper Test:** Manually alter the local chain data on one node (e.g., edit the saved file to change a credential) and run the app – it should detect an inconsistency (hash mismatch) and either refuse to start or self-heal by fetching correct data from peers. This tests immutability enforcement.
- **Performance Test:** Measure how long it takes to issue a credential and propagate to all nodes (should be quick, a matter of milliseconds on localhost). If possible, also test with, say, 100 blocks in chain to see that UI still works smoothly.
- **Bug Fixing:** Record any bugs or issues found (e.g., synchronization bugs, UI glitches, etc.) and fix them. Pay special attention to data consistency and error handling.
- **User Acceptance Test:** If there are stakeholders or instructors, prepare a demo where someone can use the system to issue and verify a credential. Incorporate their feedback on usability or any missing features.
- **Deliverables:** A test report summarizing the test cases and outcomes. By end of this sprint, any critical bugs should be resolved, and the system should be stable and secure in handling the main use cases.

Sprint 11: Deployment Preparation (Week 11)

- **Packaging:** Prepare the application for deployment. This could involve bundling the Java application into an executable JAR file or native package. Ensure all dependencies are included or specified. Also include any configuration files (like the list of validators or initial peers).
- **Documentation:** Write comprehensive documentation for the system:
- **User Manual:** How to install and run a node, how to use the GUI to issue and verify credentials.
- **Technical Documentation:** Explain the system architecture, the blockchain data structures, and consensus mechanism (PoA), so that future developers or reviewers understand the implementation [4]. Include the final version of the architecture diagram and possibly a flowchart of the issuance and verification processes.
- **Draw.io Diagram File:** Ensure the architecture diagram created is neatly formatted; if required, deliver the draw.io source file so it can be edited or viewed by others. This diagram should clearly map all components (nodes, validators, GUI, data flow) as per project requirements.
- **Simulated Deployment:** If possible, test the application in an environment similar to deployment. For instance, run nodes on different machines or different JVMs to mimic a real network (could be done on VMs or simply on one machine using different ports). This is to verify that there are no hardcoded localhost assumptions and the system works over a network.

- **Performance Tuning:** If any performance bottlenecks were noticed (not likely in a small project), do minor tuning. For example, if block broadcast was synchronous, consider making it asynchronous to speed up issuing; or if verification search is slow, consider indexing by student ID.
- **Deliverables:** Deployment artifacts (like the JAR file) and a complete documentation set. At this point, the project should be ready for handoff or presentation, with all features implemented and documented.

Sprint 12: Final Review and Presentation (Week 12)

- **Final Polishing:** Fix any last-minute issues or cosmetic problems. Double-check that all planned features are implemented. Small enhancements can be added if time permits (for example, improving the visual representation of the blockchain in the UI, or adding a feature to export the blockchain data).
- **Project Presentation:** Prepare a presentation or demo script to showcase the project. This might involve creating slides that explain the problem (credential fraud), our blockchain solution, and demonstrating how a credential is issued and later verified live on the system. Emphasize how the blockchain ensures authenticity (perhaps by showing what happens if tampering is attempted – the system will catch it).
- **Evaluation Criteria Check:** Make sure all project requirements are met: Java-based P2P blockchain, PoA consensus with hardcoded validators, Swing GUI for issue/verify, and the ledger storing the required fields (name, degree, date, institute, student ID). Verify that the system indeed provides an **immutable academic credential ledger** that is easily verifiable.
- **Future Directions:** In documentation or presentation, include a discussion of possible future improvements: e.g., adding more dynamic consensus (allow adding validators through a governance process), improving security (encrypting P2P traffic, better authentication for issuance), scaling the system, or integrating with existing university databases. Also mention potential to use QR codes on physical certificates that link to the blockchain record, etc., to show forward-thinking.
- **Team Retrospective:** Do a final team meeting to reflect on the 12-week development process. Note what went well and lessons learned, especially since this project involves complex concepts (blockchain, cryptography, distributed systems). This can be useful for the project report.
- **Deliverables:** The final deliverable is the fully functional system, presented and delivered to the stakeholders. Additionally, the final report (including all citations and references used for research) is completed. All source code, the draw.io diagram, and user manuals are packaged for submission. The project is now ready to be evaluated or deployed as needed.

By following these sprints, at the end of 12 weeks we will have a comprehensive system where academic credentials are securely issued onto a blockchain and can be verified by anyone. The use of a Java-based PoA blockchain ensures fast and efficient operation suited for an educational context, where trust is placed in known authorities (universities) rather than anonymous miners [4]. The immutable ledger of degrees and certificates provides a tamper-proof audit trail that **simplifies verification and builds trust** in the authenticity of academic qualifications [2]. This phased plan ensures steady progress from design to deployment, resulting in a successful implementation of an immutable academic credentials verification system.

Sources:

- Saleh et al., *"Blockchain-Enhanced System for Certificate Verification,"* IJCRT 2021 – Notes the use of blockchain to eliminate diploma fraud; University of Nicosia issues all diplomas via blockchain since 2017 [3].
- Poojary & Rajeshwari, *"Blockchain Based Academic Credentials Verification System,"* IJSREM 2025 – Emphasizes a tamper-proof ledger using SHA-256 hashing for immutable credential verification [2] [9].
- MIT Media Lab, *Digital Certificates Project*, 2016 – Describes signing a digital certificate and recording its hash on the blockchain for later verification [10].
- Changelly Academy, *"What is Proof-of-Authority (PoA)?,"* 2025 – Definition of PoA consensus (approved validators create blocks based on identity/reputation) and its high performance in private networks [4] [11] [12].
- Medium article by Alex (2023) – Explanation of blockchain fundamentals (blocks with hashes linking, P2P network for decentralization and consensus) [7] [5].
- Wikipedia, *"Proof of Authority,"* – Notes that in PoA, only approved validator nodes can validate and create new blocks [6], providing trust based on validator identity rather than computational work.

---

[1] Diploma certificate verification Blockchain architecture. | Download Scientific Diagram
https://www.researchgate.net/figure/Diploma-certificate-verification-Blockchain-architecture_fig6_361804558

[2] [9] Blockchain Based Academic Credentials Verification System – IJSREM
https://ijsrem.com/download/blockchain-based-academic-credentials-verification-system/

[3] IJRTI
https://ijcrt.org/papers/IJCRT2107283.pdf

[4] [11] [12] What is Proof-of-Authority (POA) Consensus in Blockchain?
https://changelly.com/blog/what-is-proof-of-authority-poa/

[5] [7] [8] I Built A Peer-2-Peer Blockchain From Scratch. 'Twas Fun. | by Alex | Medium
https://medium.com/@alexfoleydevops/i-built-a-peer-2-peer-blockchain-from-scratch-twas-fun-ecad704c0836

[6] Proof of authority - Wikipedia
https://en.wikipedia.org/wiki/Proof_of_authority

[10] [13] What we learned from designing an academic certificates system on the blockchain | by MIT Media Lab | MIT MEDIA LAB | Medium
https://medium.com/mit-media-lab/what-we-learned-from-designing-an-academic-certificates-system-on-the-blockchain-34ba5874f196

[14] Utilizing Blockchain Technology for University Certificate Verification System
https://www.ijais.org/archives/volume12/number45/oluwaseyi-2024-ijca-451977.pdf