**CrediChain – Blockchain-Based Academic Record Management**

**PHASE 2 – Implementation & Execution**

**College Name:** VIT Bhopal University

**Name:** Aakansh Yadav

# 1. Project Overview

**CrediChain** is a decentralized platform for academic record management developed with a custom blockchain architecture in JavaScript. It allows educational institutions to issue, verify, and manage student credentials securely, maintaining integrity, transparency, and privacy.  
  
Through the incorporation of a bespoke blockchain engine, web-based frontend, Node.js backend services, and minimalistic SQLite database, **CrediChain** provides a tamper-evident and verifiable solution for academic transcripts. The platform is entirely containerized with Docker, and deployment is supported through platforms such as Heroku and AWS, to provide cross-environment consistency and ease of testing.

This development phase covers the actual implementation and execution of the system, including:

* Blockchain creation and academic data integration
* Backend logic and database schema configuration
* Frontend development for students, faculty, and administrators
* Dockerized deployment for scalability and portability
* System-level testing and blockchain-based verification of records

# 2. System Architecture

## 2.1 Core Components

**1. Frontend (Client Side)**

* **Technologies**: HTML, CSS, JavaScript
* **Features**:
  + Forms for academic record entry, grade submission, and verification
  + Interactive UI for students, faculty, and institutions
  + Responsive layout with modern styling using CSS (including gradients, glass morphism, and animations)
  + Data visualization tools such as **Chart.js** for grade distribution and CGPA analytics

**2. Backend (Server Side)**

* **Technologies**: Node.js, SQLite
* **Features**:
  + RESTful APIs for student, faculty, subject, and grade management
  + Blockchain integration for hashing and verifying academic records
  + Secure SQLite database for storing structured academic data (with migration planned to MongoDB)
  + Role-based access control to separate permissions for students and faculty

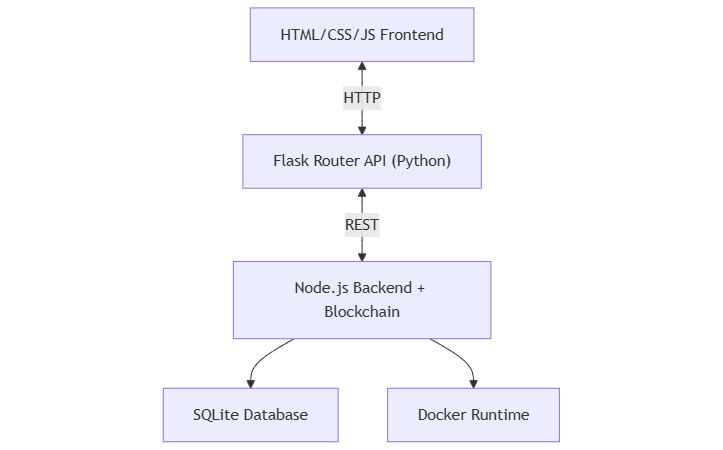
**3. Blockchain Engine**

* **Technologies**: JavaScript
* **Features**:
  + Custom-built Block and Blockchain classes for academic transaction logging
  + SHA256 hashing to ensure record integrity
  + Timestamping and immutable ledger for verification of grades and credentials
  + Verification mechanisms and badges indicating tamper-free

**4. Deployment Layer**

* **Technologies**: Docker, Python (Flask)
* **Features**:
  + Fully Dockerized services for frontend, backend, and blockchain components
  + Container orchestration for easy deployment and testing across environments
  + Volume mapping for persistent academic data storage
  + Support for environment-based configuration for flexible deployment (e.g., Heroku, AWS, Google Cloud)

## 2.2 Architecture Diagram



# 3. Implementation Steps

## Step 1: Blockchain Development (Node.js)

* **Created a Block class** with:
* **Attributes:**
  + **timestamp:** When the grade was issued
  + **data:** Includes student ID, subject, grade, and faculty ID
  + **previousHash:** Link to the previous block in the chain
  + **hash:** Unique SHA-256 hash for current block
  + **nonce:** Used to satisfy Proof-of-Work (PoW)
* **Methods:**
  + **calculateHash():** Computes the SHA-256 hash using the block's contents
  + **mineBlock(difficulty):** Performs PoW to secure the block before adding it
  + **isValid():** Checks if the block has been altered or is tampered
* **Created a Blockchain class** with:
  + **createGenesisBlock():** Initializes the blockchain with the first default block
  + **addBlock(data):** Adds a new academic record block to the chain after mining
  + **isChainValid():** Verifies the integrity of all blocks in the chain
  + **getBlock(hash):** Retrieves a specific academic record using its hash
  + **getBlocksByStudent(studentId):** Filters blocks related to a particular

In CrediChain, a custom blockchain was developed using Node.js and JavaScript to securely store and verify academic records such as grades. This blockchain ensures data immutability, transparency, and tamper detection—critical for maintaining the integrity of academic credentials.

This blockchain module is the core engine of CrediChain, providing a transparent and tamper-proof academic record system without relying on any third-party blockchain network.

## Step 2: Backend Development (Node.js + SQLite)

The backend of **CrediChain** was developed using **Node.js** and **Express**, with **SQLite** serving as the lightweight database for academic data storage. The system maintains structured and secure records of students, faculty, subjects, and grades.

Database Implementation

* Utilized SQLite for quick development and easy deployment.
* Defined key tables:
* **users:** Stores student and faculty credentials (with hashed passwords).
* **academic\_records:** Stores subject-wise grade records with references to student and faculty IDs.
* Used the better-sqlite3 library for efficient and synchronous DB operations.

**API Highlights:**

* **POST /api/login:** Authenticate users (students/faculty).
* **GET /api/users/:id:** Retrieve user profile by ID.
* **POST /api/records:** Add a new academic record.
* **GET /api/records/student/:id:** Fetch all academic records for a specific student.
* **GET /api/records/faculty/:id:** Fetch all records submitted by a specific faculty member.
* Each new record added to the database is also **hashed and added as a new block** to the blockchain, maintaining full traceability and tamper-resistance.

This ensures every grade entry is **tamper-proof** and **verifiable**.

## Step 3: Frontend Development (HTML/CSS/JavaScript)

* CrediChain features a clean, responsive, and user-friendly web interface, built using:
  + HTML/CSS for structure and layout
  + Bootstrap for responsive design
  + Vanilla JavaScript for dynamic functionality and API integration

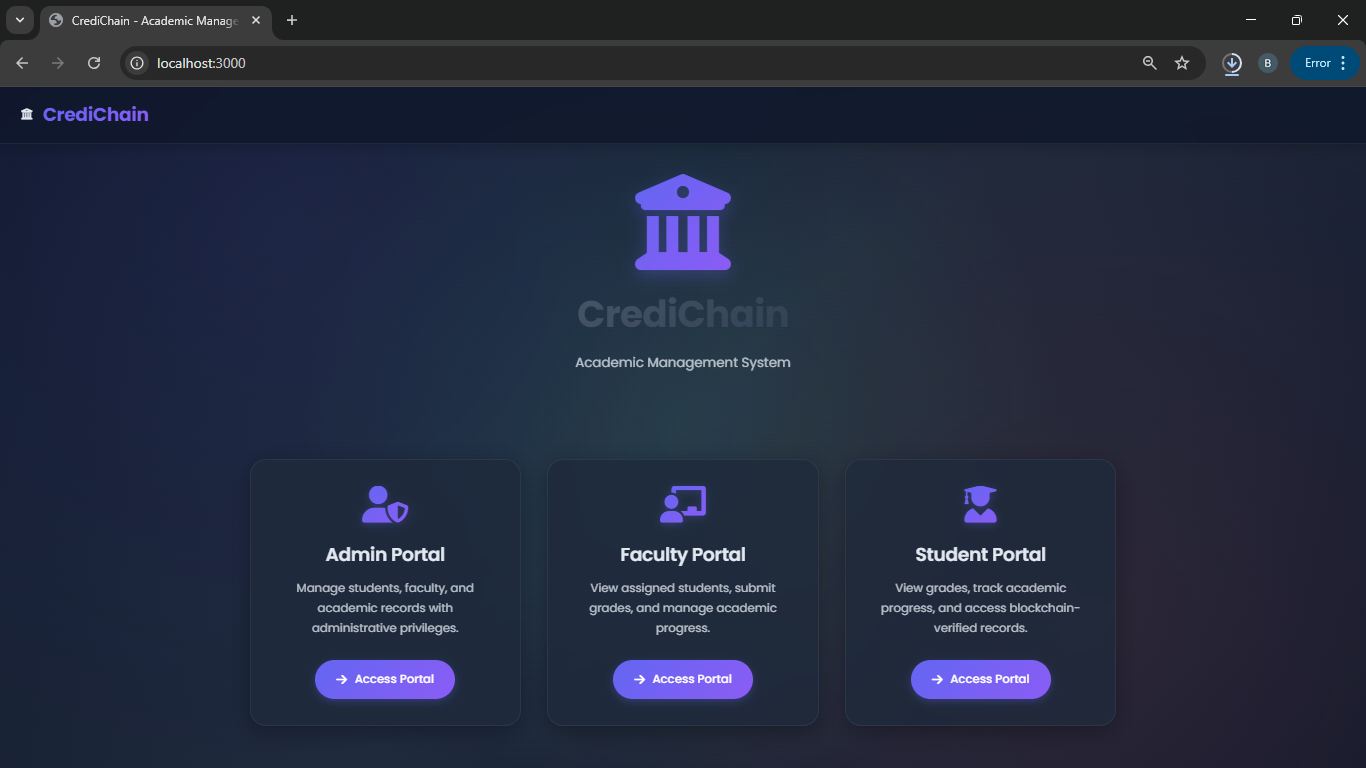
**Functional Sections:**

* **Home Page**: Highlights the advantages of blockchain in academic record management.
* **Register/Login:** Handles user authentication and access control for students and faculty.
* Dashboard (Role-Based):
* Faculty can:
* View assigned students
* Add and update grades
* Students can:
* View their subject-wise grades
* See grade distribution and calculate CGPA
* Verify grades on the blockchain
* **Real-Time Interaction**
* All operations (record submission, fetching, login) are handled via fetch() calls to the backend APIs.
* Frontend shows **user-friendly feedback alerts** for success, errors, or verification status.

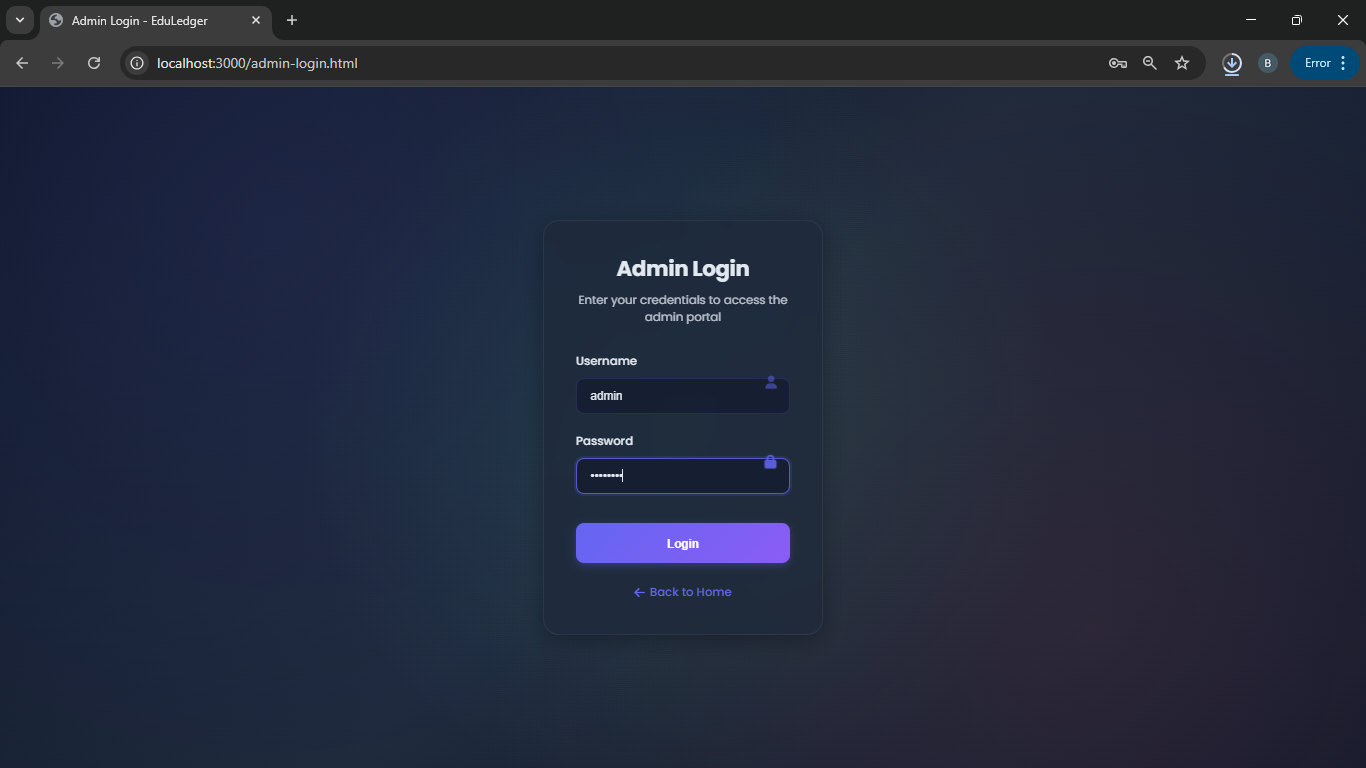
## Step 5: Testing and Validation

| **Test Case** | **Input** | **Expected Result** | **Status** |
| --- | --- | --- | --- |
| Add Academic Record | Valid student/faculty details | Record saved to DB and new block mined on the blockchain | ✅ |
| View Records by Student | Student ID | List of academic records with subject, grade, timestamp | ✅ |
| Faculty Login with Valid Credentials | Faculty username and password | Login successful, dashboard access granted | ✅ |
| Add Incomplete Record | Missing subject/grade fields | Error message displayed, record not saved | ✅ |
| Blockchain Integrity Check | Tampered data in a block | Chain validation fails, flagged as invalid | ✅ |

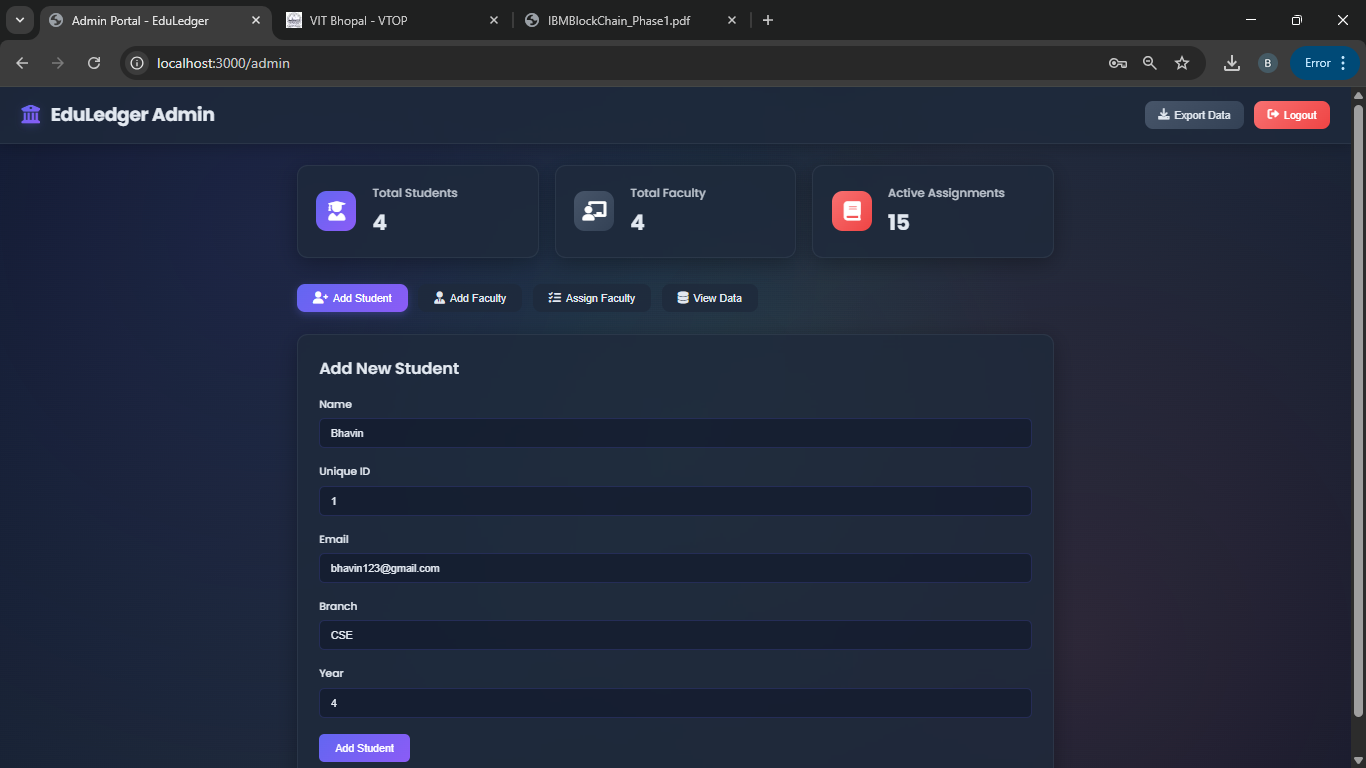
# 4. Screenshots:



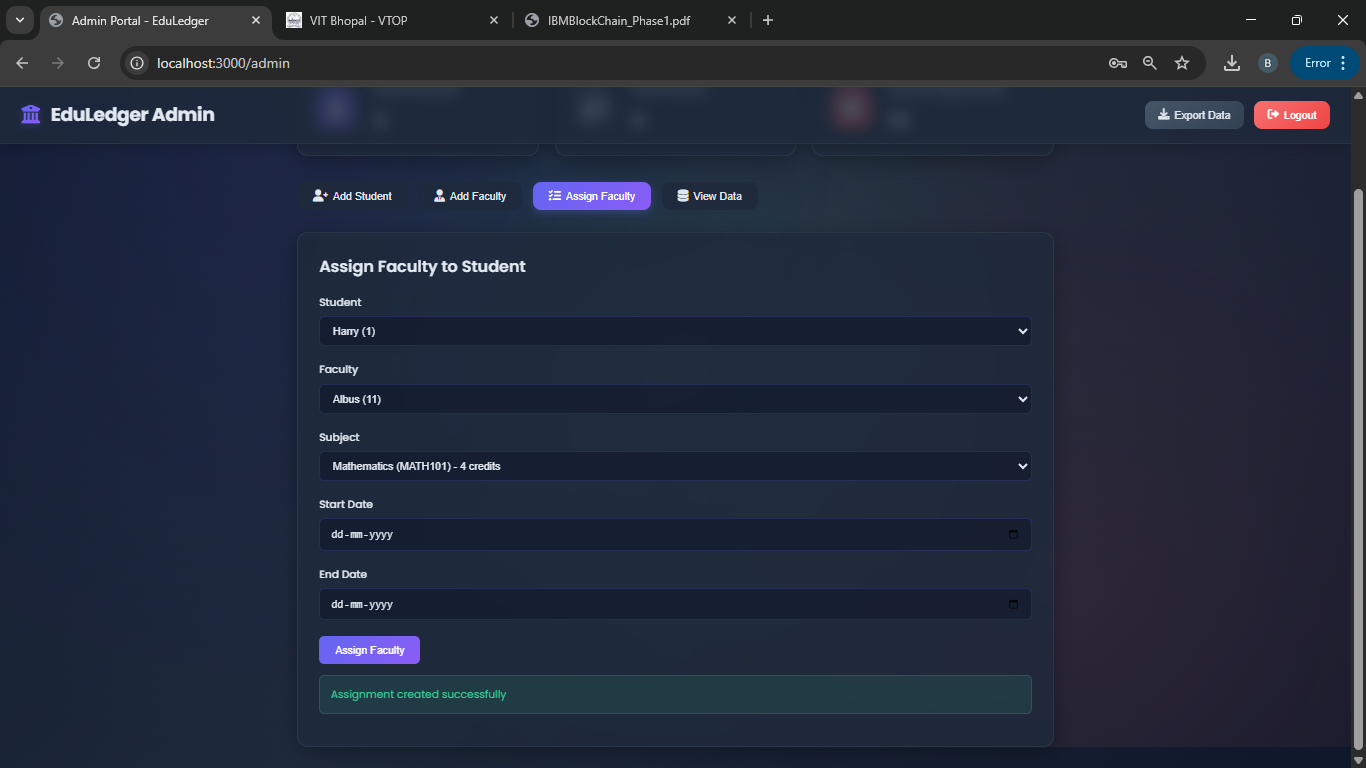
**Figure 1:** Home page of the website



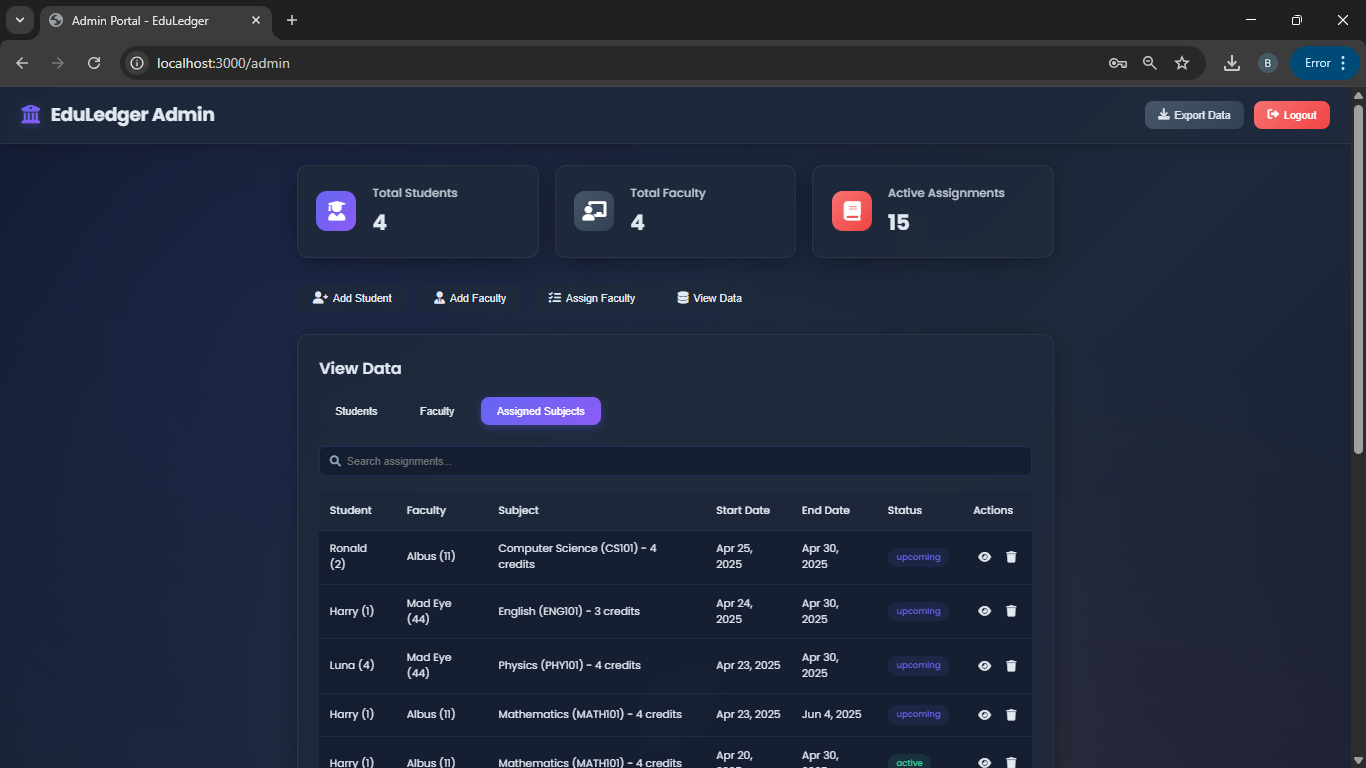
**Figure 2:** Logging in to the website



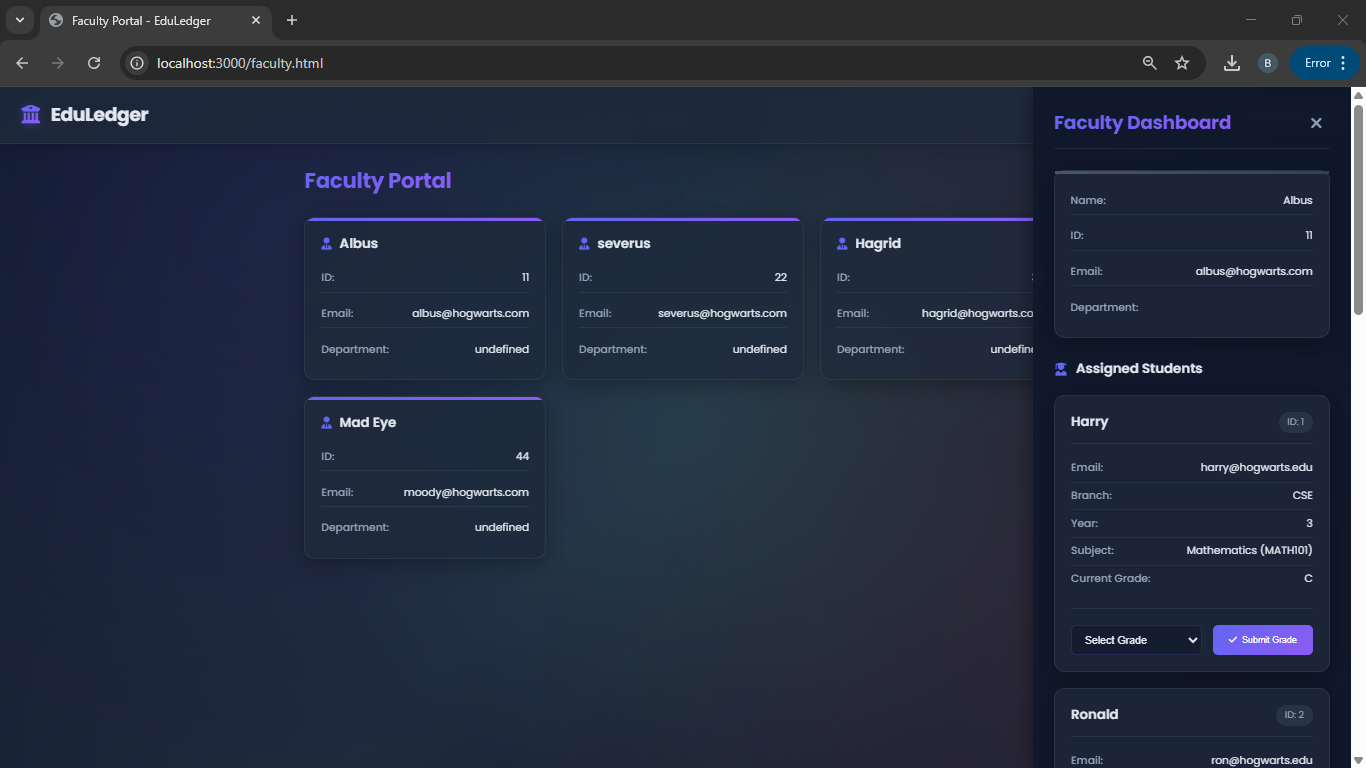
**Figure 3:** Data entry for student



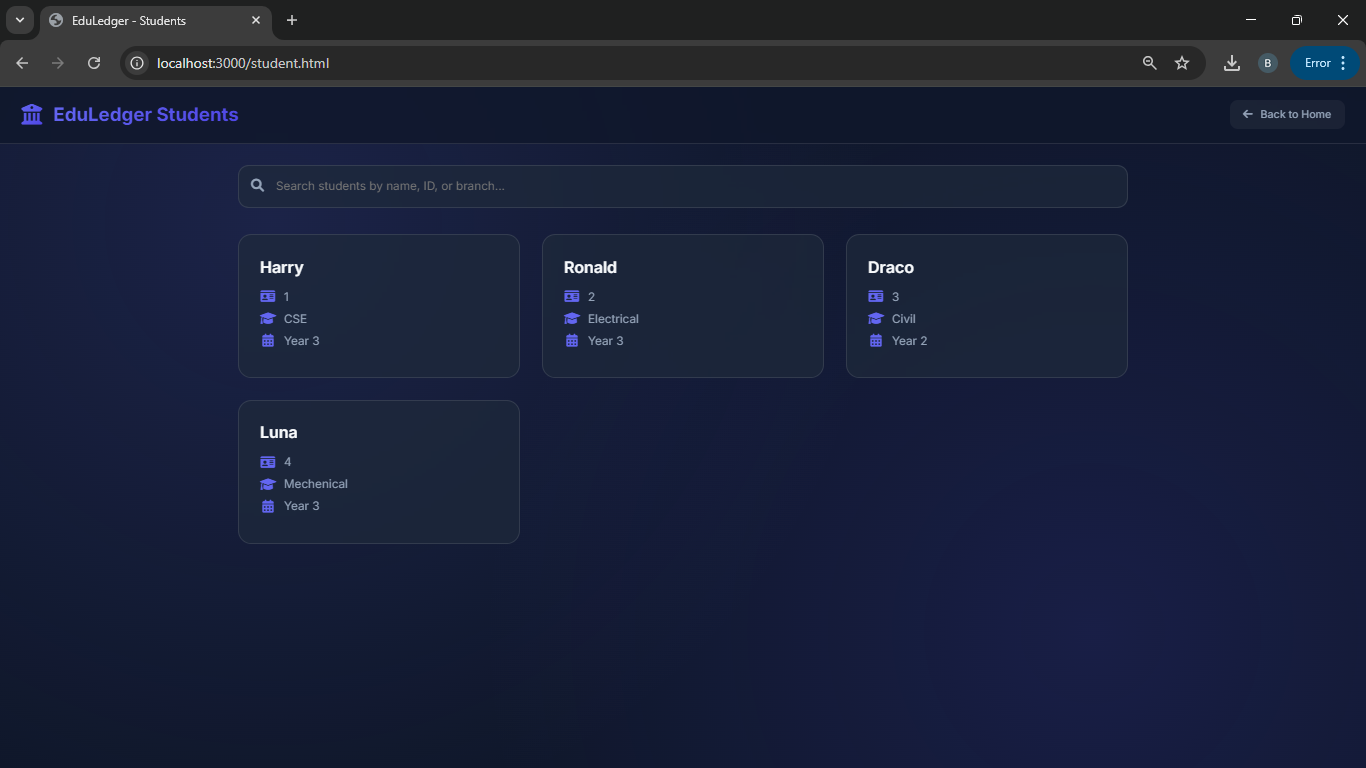
**Figure 4:** Data saving in the database



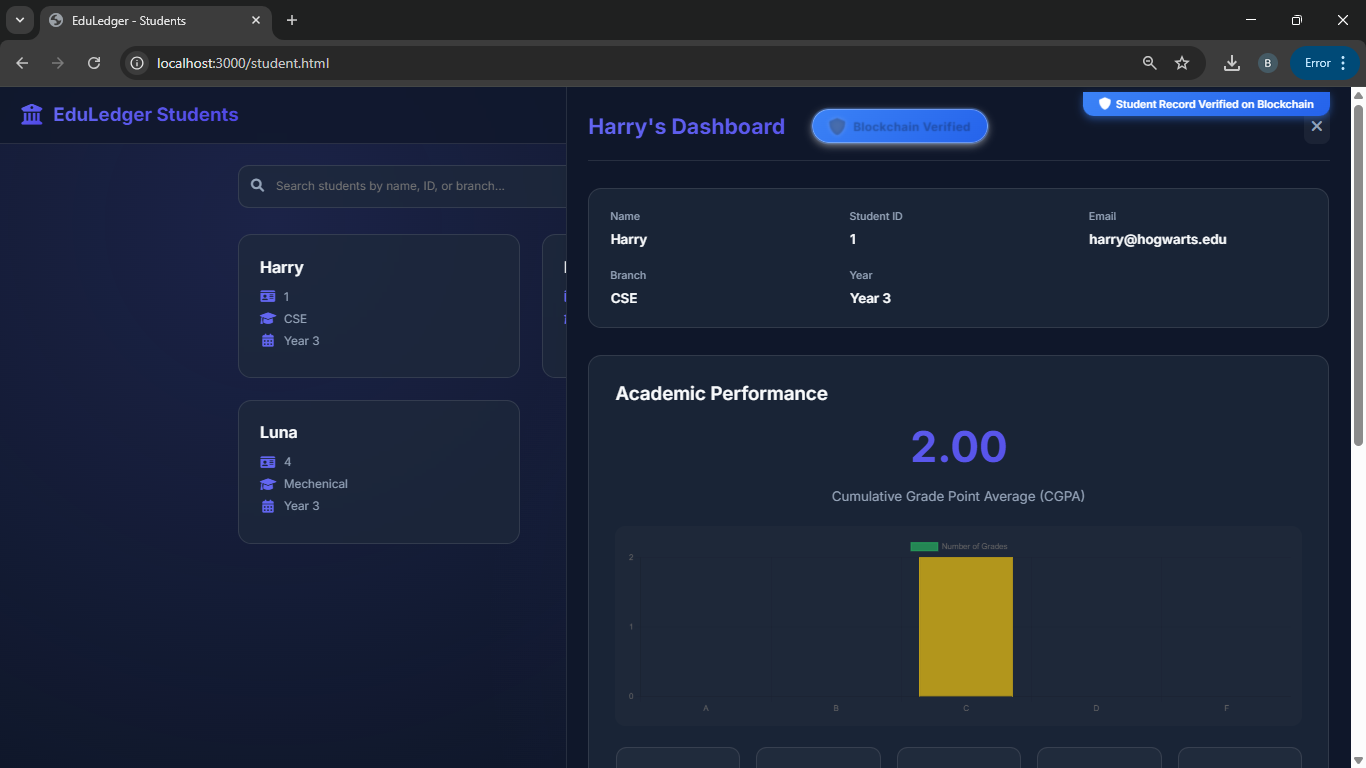
**Figure 5:** Assigned Subjects



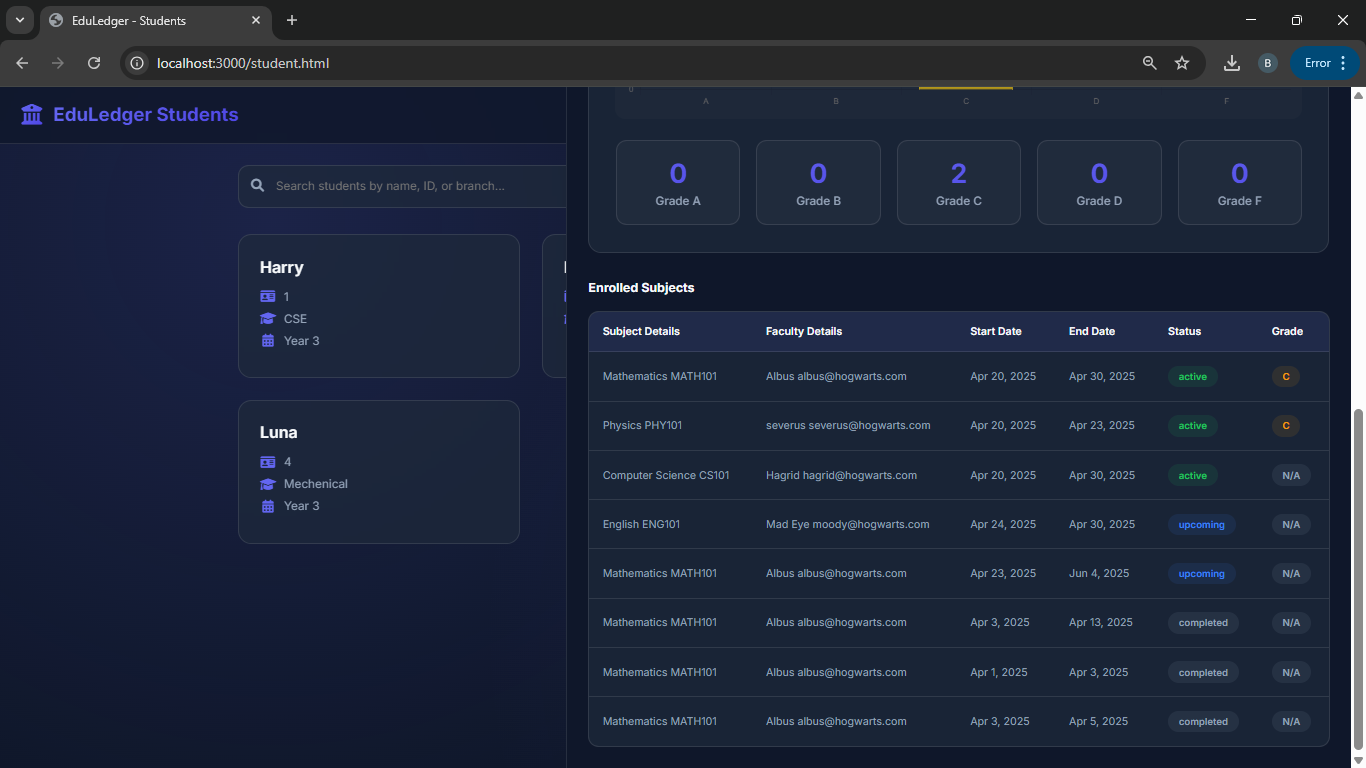
**Figure 6:** Faculty Portal



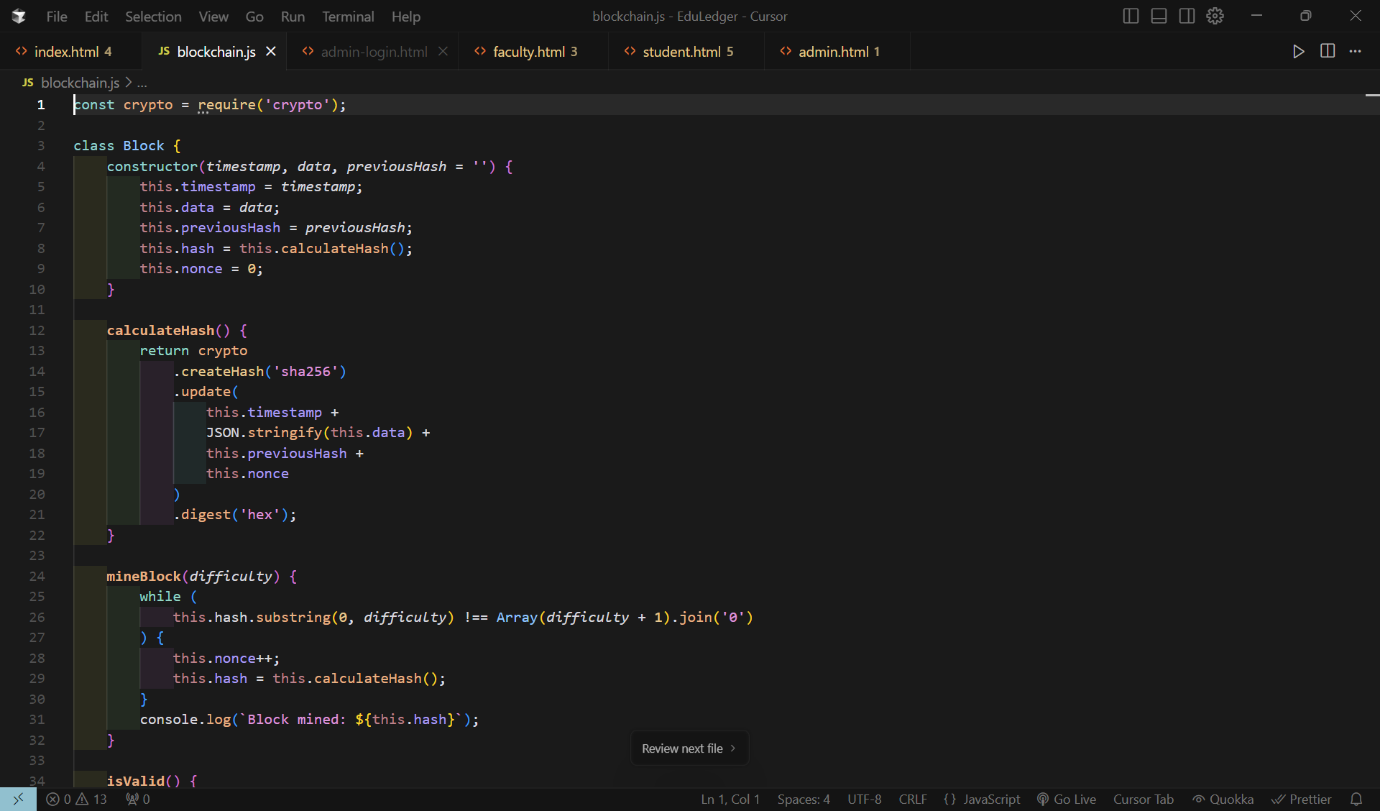
**Figure 7:** Student Dashboard



**Figure 8:** Verification



**Figure 9:** Enrolled Subjects

**Figure 10:** Blockchain.js

# 5. Code Snippets

## 5.1 Block Structure (CrediChain/blockchain.js)

const crypto = require('crypto');

class Block {

    constructor(timestamp, data, previousHash = '') {

        this.timestamp = timestamp;

        this.data = data;

        this.previousHash = previousHash;

        this.hash = this.calculateHash();

        this.nonce = 0;

    }

    calculateHash() {

        return crypto

            .createHash('sha256')

            .update(

                this.timestamp +

                JSON.stringify(this.data) +

                this.previousHash +

                this.nonce

            )

            .digest('hex');

    }

    mineBlock(difficulty) {

        while (

            this.hash.substring(0, difficulty) !== Array(difficulty + 1).join('0')

        ) {

            this.nonce++;

            this.hash = this.calculateHash();

        }

        console.log(`Block mined: ${this.hash}`);

    }

    isValid() {

        return this.hash === this.calculateHash();

    }

}

## 5.2 Blockchain Structure (CrediChain/blockchain.js)

class Blockchain {

    constructor() {

        this.chain = [this.createGenesisBlock()];

        this.difficulty = 4; // Adjust difficulty as needed

    }

    createGenesisBlock() {

        return new Block(

            Date.now(),

            {

                studentId: '0',

                subject: 'Genesis',

                grade: 'A',

                facultyId: '0'

            },

            '0'

        );

    }

    addBlock(data) {

        const newBlock = new Block(

            Date.now(),

            data,

            this.getLatestBlock().hash

        );

        newBlock.mineBlock(this.difficulty);

        this.chain.push(newBlock);

        return newBlock;

    }

    getBlocksByStudent(studentId) {

        return this.chain.filter(block =>

            block.data.studentId === studentId

        );

    }

    getBlocksBySubject(subject) {

        return this.chain.filter(block =>

            block.data.subject === subject

        );

    }

}

module.exports = {

    Block,

    Blockchain

};

## 5.3 SQlite Model for Academic Records (CrediChain/server.js)

 db.serialize(() => {

        // Create Students table

        db.run(`

            CREATE TABLE IF NOT EXISTS students (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                name TEXT NOT NULL,

                unique\_id TEXT UNIQUE NOT NULL,

                email TEXT UNIQUE NOT NULL,

                branch TEXT NOT NULL,

                year INTEGER NOT NULL,

                created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

            )

        `);

## 5.4 User Registration & Password (CrediChain/server.js)

app.post('/api/admin\_login', (req, res) => {

    const { username, password } = req.body;

    if (username === ADMIN\_CREDENTIALS.username && password === ADMIN\_CREDENTIALS.password) {

        res.json({ success: true, message: 'Login successful' });

    } else {

        res.json({ success: false, message: 'Invalid username or password' });

    }

});

## 5.5 Add Faculty routes (CrediChain/server.js)

app.post('/api/add\_faculty', (req, res) => {

    const { name, unique\_id, email, research\_area, post } = req.body;

    db.run(

        'INSERT INTO faculty (name, unique\_id, email, research\_area, post) VALUES (?, ?, ?, ?, ?)',

        [name, unique\_id, email, research\_area, post],

        function(err) {

            if (err) {

                res.json({ success: false, message: 'Unique ID or Email already exists' });

            } else {

                res.json({ success: true, message: 'Faculty added successfully' });

            }

        }

    );

});

app.get('/api/get\_faculty', (req, res) => {

    db.all('SELECT \* FROM faculty', [], (err, rows) => {

        if (err) {

            res.status(500).json({ error: err.message });

            return;

        }

        res.json(rows);

    });

});

## 5.6 Admin Login (EduLodger/public/admin-login.html)

<script>

        document.getElementById('loginForm').addEventListener('submit', async (e) => {

            e.preventDefault();

            const username = document.getElementById('username').value;

            const password = document.getElementById('password').value;

            try {

                const response = await fetch('/api/admin\_login', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json'

                    },

                    body: JSON.stringify({ username, password })

                });

                const result = await response.json();

                if (result.success) {

                    window.location.href = '/admin';

                } else {

                    showMessage('loginMessage', false, result.message);

                }

            } catch (error) {

                showMessage('loginMessage', false, 'Error connecting to server');

            }

        });

        function showMessage(elementId, success, message) {

            const messageElement = document.getElementById(elementId);

            messageElement.textContent = message;

            messageElement.className = `message ${success ? 'success' : 'error'}`;

            messageElement.style.display = 'block';

            if (!success) {

                setTimeout(() => {

                    messageElement.style.display = 'none';

                }, 3000);

            }

        }

    </script>

# 6. Challenges Faced

| **Challenge** | **Solution** |
| --- | --- |
| Blockchain validation for academic records | Implemented isChainValid() and isValid() methods to ensure data integrity |
| Handling asynchronous backend tasks | Used async/await with proper try-catch blocks in Node.js API routes |
| Securing user credentials | Integrated bcryptjs for hashing and verifying passwords securely |
| Connecting frontend with backend APIs | Configured CORS and tested fetch() calls for smooth JSON-based API communication |
| Docker container communication | Properly exposed ports (3000 for backend, 8000 for frontend) in docker-compose.yml |
| Preserving student data in SQLite | Used Docker volumes to retain database content across container restarts |

**7. Evaluation Criteria Mapping**

| **Criteria** | **Outcome** |
| --- | --- |
| Correctness | All core functionalities such as user registration/login, student/faculty portals, grade submission, retrieval, CGPA calculation, and blockchain-based verification work accurately and as intended. |
| Completeness | Full-stack implementation including custom blockchain, Node.js backend, SQLite database, HTML/CSS/JS frontend, and Docker-based deployment is completed and fully integrated. |
| Code Quality | Codebase is modular with proper class-based design for Blockchain and API routing. Follows best practices with clean, maintainable structure across frontend and backend. |
| Security Measures | Passwords are securely hashed using blockchain.js; CORS and input validation prevent unauthorized access. Blockchain ensures tamper-proof record storage. |
| UI/UX Design | Interface is responsive and user-friendly, with Bootstrap layout and JavaScript interactivity for faculty/student dashboards and record management. |
| Blockchain Accuracy | Custom blockchain performs SHA-256 hashing, nonce-based mining, and full chain integrity validation. All records are immutable and verifiable. |
| Testing and Debugging | Manual tests verify record consistency, blockchain integrity, user authentication, and API communication. Chain tampering simulations correctly trigger invalid status. |
| Deployment | Entire system containerized using Docker with services for backend, blockchain, database, and Flask-based frontend routing. Ports are properly mapped (3000 backend, 8000 frontend). |
| Innovation | CrediChain uses a fully custom-built blockchain for academic record verification, avoiding third-party chains. Its Dockerized setup promotes modularity and portability. |
| Documentation | Complete documentation provided for system architecture, implementation steps, APIs, testing outcomes, and future scope. |

# 8. Current Output

* **User Registration/Login:**
* Fully working with hashed password storage
* **Faculty Portal Output**
* Dashboard view with assigned students and subject lists
* Grade submission interface with success/failure alerts
* Ability to update existing grades
* Secure authentication with session tracking
* **Student Portal Output**
* Personal dashboard showing academic records
* Real-time CGPA calculation
* Grade distribution graphs using Chart.js
* Blockchain-based grade verification showing “Verified”/“Tampered” badges
* **Blockchain System:**
* Each academic record submitted (grades, subjects, student-faculty mapping) is hashed and added as a new block in the custom blockchain.
* **Blockchain Console Logs**
* Logs mined blocks with nonce and hash
* Chain integrity status ("Chain is valid" or "Chain is invalid")
* **Docker Output**
* Successful build and startup of containers via docker-compose
* Port mapping confirmed:
* Frontend on <http://localhost:8000>
* Backend API on <http://localhost:3000>

# 9. Future Scope

| **Feature** | **Description** |
| --- | --- |
| MongoDB integration | Planned migration from SQLite to MongoDB for better scalability and support for document-based academic records. |
| Peer-to-peer networking | Enable multiple institutions (schools/universities) to participate in a distributed blockchain network for decentralized academic data sharing and verification. |
| Student QR Code Generation | Allow students to generate QR codes for sharing academic records securely with recruiters or institutions. |
| Academic Analytics | Integrate dashboards for grade distribution, CGPA trends, and subject performance using historical academic data. |
| Mobile interface | Develop a PWA or native mobile application to give students and faculty mobile access to records and verification features. |
| Record encryption with access keys | Implement per-student encryption of records, viewable only by authorized users (students, faculty, institutions) with secure access keys. |

# 10. Conclusion

**CrediChain** is a blockchain-based academic record management system designed to enhance data integrity, transparency, and security. By replacing traditional systems with a decentralized solution, **CrediChain** ensures that academic records are tamper-proof, verifiable, and easily accessible to authorized users.

The platform combines a Node.js backend, a responsive frontend, and an SQLite database, all containerized using Docker for seamless deployment. Key features include user authentication, grade submission, CGPA calculation, and blockchain-based verification, making it a reliable and efficient system for both students and faculty.

**CrediChain** lays the groundwork for future enhancements such as mobile access, MongoDB integration, secure QR-based sharing, and cross-institutional blockchain networks—offering a scalable and innovative approach to modern education systems.