CENG 485

Introduction to Blockchain Technology

2024-2025 Fall

Final Project Report

VehiCrypto – Vehicle Maintenance Application

Names of members

Çağdaş GÜLEÇ 202111004

Doğukan POYRAZ 202111061

Arda YILDIZ 202111071

Ahmet Kerem ÖZTÜRK 202211405

Salih Barkın AKKAYA 202211004

Date

16.11.2024

Table of Contents

1. Introduction

In today's digital era, the automotive industry continues to embrace technological innovations to bolster efficiency, security, and user satisfaction. Vehicle maintenance record management stood to benefit greatly from modernization. Traditional paper-based documentation or centralized databases are susceptible to data loss, unauthorized access, and manipulation, lacking the transparency needed to foster trust among vehicle owners and service providers.

To address these issues, we developed VehiCrypto, a blockchain-based system that securely records and stores all maintenance operations performed on vehicles. By harnessing the immutable and decentralized characteristics of blockchain technology, VehiCrypto ensures that maintenance records remain tamper-proof, transparent, and readily accessible to authorized stakeholders. Vehicle owners gain full visibility into their vehicle's maintenance history, which increases trust and confidence in service providers. Meanwhile, service providers log their maintenance activities in a verifiable and secure manner.

2. General Overview

VehiCrypto provides a comprehensive, blockchain-based platform to record and track vehicle maintenance operations. With this system, vehicle owners can instantly verify their vehicle's maintenance history, and service providers can rest assured that their records are stored in a manner that cannot be altered. By leveraging blockchain's immutability, transparency, and security, VehiCrypto guarantees that vehicle maintenance histories remain secure and trustworthy. We have prioritized data security within a decentralized structure while offering a user-friendly interface.

Technologies Used:

- Frontend (Flutter): We used Flutter to create a modern, high-performance mobile application that runs smoothly on both Android and iOS devices. Users can access their vehicle maintenance history and add new maintenance information through this app.
- Backend (Solidity & Node.js/Python): Our backend includes smart contracts written in Solidity, which operate on a blockchain platform (e.g., Ethereum). We also employed Node.js (or Python) for RESTful API development, enabling secure and efficient data exchange between the frontend and the blockchain.

- Blockchain (Ethereum): All maintenance-related information is stored on the Ethereum network, benefiting from immutability and transparency. Both vehicle owners and service providers can verify transactions and data integrity.
- IPFS: We integrated the InterPlanetary File System for storing large files (e.g., service invoices, reports, or images). Rather than placing large files directly on the blockchain, we store only their unique content identifiers (CIDs) on-chain, optimizing for both cost and performance.

Project Significance: By adopting a decentralized approach, VehiCrypto has improved trust between vehicle owners and service providers, streamlined record-keeping, and digitized the entire vehicle maintenance process. Blockchain's security guarantees that stored data remains immutable and transparent—offering a modern, secure, and efficient alternative to traditional paper-based or centralized systems.

3. Design of Components

2.1. Frontend

The front end constitutes the end-user touchpoint of VehiCrypto. Developed using the Flutter framework, our application functions seamlessly on both Android and iOS devices.

Frontend Accomplishments:

1. **User Interface Design (UI Design):**
   - We created a user-friendly interface where vehicle owners input their 11-digit reference code to view detailed maintenance records.
   - An admin panel allows authorized personnel to enter vehicle maintenance data into the system.
   - The final design is minimalist, intuitive, and functional. We used Flutter's Material Design and Cupertino Widgets to ensure a consistent user experience.
2. **Organization and User Flow:**
   - We differentiated the user flow for owners and admins:
   - User Flow: After authentication, owners enter their 11-digit reference code to retrieve maintenance details.
   - Admin Flow: Authorized service providers or admins input maintenance data, which is then recorded on the blockchain.

- We used Flutter's Navigator and Route features to facilitate smooth navigation between screens.

3. **Data Input and Display Forms:**
   - **Admin Panel:**
     - A form collects vehicle maintenance details such as plate number, maintenance type, date, and reference code.
     - We employed Flutter's TextFormField and custom validators for robust input handling and error checking.
   - **User Panel:**
     - Owners can view their maintenance records via a ListView or Card-based layout, providing a clear overview of each service.
4. **API Integration:**
   - All data exchange with the backend occurs through RESTful APIs, using packages like http or dio.
   - The backend returns JSON, which the app parses and displays to the user.

Technical Challenges and Solutions:

- **Dynamic Data Display:**

We leveraged asynchronous data fetching with FutureBuilder to manage real-time blockchain queries.

- **Error Handling:**

We display clear, user-friendly messages for network errors or invalid input, improving the overall reliability of the application.

The finished frontend now delivers a streamlined experience for both users and admins, ensuring all maintenance data is easily accessible.

2.2. Backend

The backend architecture oversees data management, business logic, and security for VehiCrypto. Designed with a focus on blockchain integration, our backend combines RESTful APIs, Node.js/Python, and smart contracts to ensure smooth data flow and secure record-keeping.

Backend Accomplishments:

1. **API Design and Data Exchange:**
   - Admin-submitted vehicle maintenance data is validated, then forwarded to the blockchain for permanent storage. Users can retrieve their records via an 11-digit reference code.
   - We implemented a RESTful API that connects the Flutter frontend to the blockchain. Key endpoints include:
     - POST /addMaintenance – Accepts maintenance details from admins, stores them on the blockchain.
     - GET /getMaintenance – Retrieves maintenance records based on a user's reference code and returns them in JSON format.
2. **Smart Contracts and Blockchain Integration:**
   - We deployed Solidity smart contracts via Remix IDE. These contracts:
     - Securely store maintenance details on-chain.
     - Generate and handle the reference code for each record.
     - Allow authorized users to query vehicle data using the reference code.
3. **Data Validation and Security:**
   - We implemented strict checks to ensure that all input data is complete and valid (e.g., date, plate number, reference code).
   - All front-to-backend communication is conducted over HTTPS, ensuring encryption in transit.
4. **Blockchain Operations:**
   - Our contracts run on the Ethereum blockchain, handling both write and read operations.
   - When a user enters their reference code, the backend queries the relevant smart contract to retrieve records and delivers them to the frontend.

Technical Challenges and Solutions:

- Blockchain Performance:

To mitigate slow blockchain response times, we introduced a caching layer for frequently accessed records, reducing load on the network.

- API Security:

We structured different access levels for admin and user endpoints. Admin routes require an API key, while users rely on their reference code for secure retrieval.

By integrating blockchain technology in the backend, VehiCrypto ensures robust, transparent, and immutable data storage while maintaining a clean, organized infrastructure for managing requests.

2.3. Smart Contract

Our smart contract, written in Solidity, serves as the decentralized backbone of VehiCrypto. It governs how vehicle registration and maintenance records are stored, secured, and accessed.

Key Features We Have Implemented:

- **Vehicle Registration and Ownership Management:**

We designed the smart contract so that each vehicle can be registered on the blockchain, linking its unique VIN to the owner's address. This prevents duplicate registrations and ensures a single source of truth for vehicle ownership.

- **Authorized Service Provider Management:**

The contract maintains a list of authorized service providers, ensuring that only verified entities can write new maintenance records to the blockchain. This protects the system from unauthorized or fraudulent updates.

- **Maintenance Record Logging:**

Once authorized, service providers can post detailed maintenance records (work done, date, relevant documentation) directly to the blockchain. Each record is time-stamped and linked to both the vehicle and the provider, forming a transparent and traceable maintenance history.

- **Immutable and Transparent Record-Keeping:**

Records cannot be altered or removed after they are written to the blockchain, guaranteeing an unchangeable history. Vehicle owners and service providers can audit these records at any time.

- **Access Control and Security:**

We implemented strict role-based logic in the contract. Owners and authorized providers can only perform actions pertinent to their roles. The smart contract inherently uses cryptographic methods to authenticate addresses, making the system secure against tampering.

By actualizing these features, VehiCrypto capitalizes on blockchain's decentralized strengths, ensuring that all vehicle maintenance data remains trustworthy, transparent, and easily verifiable.

2.4. IPFS Integration

To support large file storage (service reports, images, invoices), we integrated the InterPlanetary File System (IPFS) into VehiCrypto. This choice helps keep the on-chain data footprint small while enabling secure, distributed file storage.

Key Features We Have Implemented:

- **Decentralized File Storage:**

We store files across a distributed network of IPFS nodes, removing dependence on any single server. This reduces the risk of downtime, data loss, or targeted attacks.

- **Content-Addressable Hashing:**

Each file is identified by a cryptographic hash (its CID), ensuring authenticity and integrity. Any change to the file results in a different hash, making tampering immediately evident.

- **Efficient Data Retrieval:**

IPFS locates files using a distributed hash table (DHT) and retrieves content from the nearest nodes, boosting performance and reducing load times, especially for frequently accessed files.

- **Integration with Smart Contracts:**

Rather than putting full files on-chain, we store only the IPFS hashes within our Solidity smart contracts. This strikes a balance between transparency (blockchain immutability) and efficient file handling (IPFS decentralization).

- **Enhanced Data Security and Privacy:**

For sensitive documents, we apply encryption before uploading to IPFS. Access control is managed at the application layer, ensuring that only authorized parties with the appropriate decryption keys can view sensitive data.

- **Scalability and Cost Efficiency:**

Offloading large files to IPFS reduces transaction fees and computational overhead on the Ethereum network, ensuring the system remains both cost-effective and scalable.

- **File Persistence and Pinning Services:**

To guarantee long-term file availability, we leverage IPFS pinning services that keep critical documents hosted, even if the original uploader goes offline.

- **Reliability and Redundancy:**

Multiple copies of each file are distributed across the IPFS network, guaranteeing redundancy. Users can be confident that their records remain safe and accessible regardless of individual node status.

Through our successful IPFS integration, VehiCrypto provides a robust, decentralized storage solution that complements blockchain's immutability. This allows users to manage large maintenance files effortlessly while benefiting from enhanced privacy and security.

4. Use Case Diagram – Context Diagram

4.1. Use Case Diagram

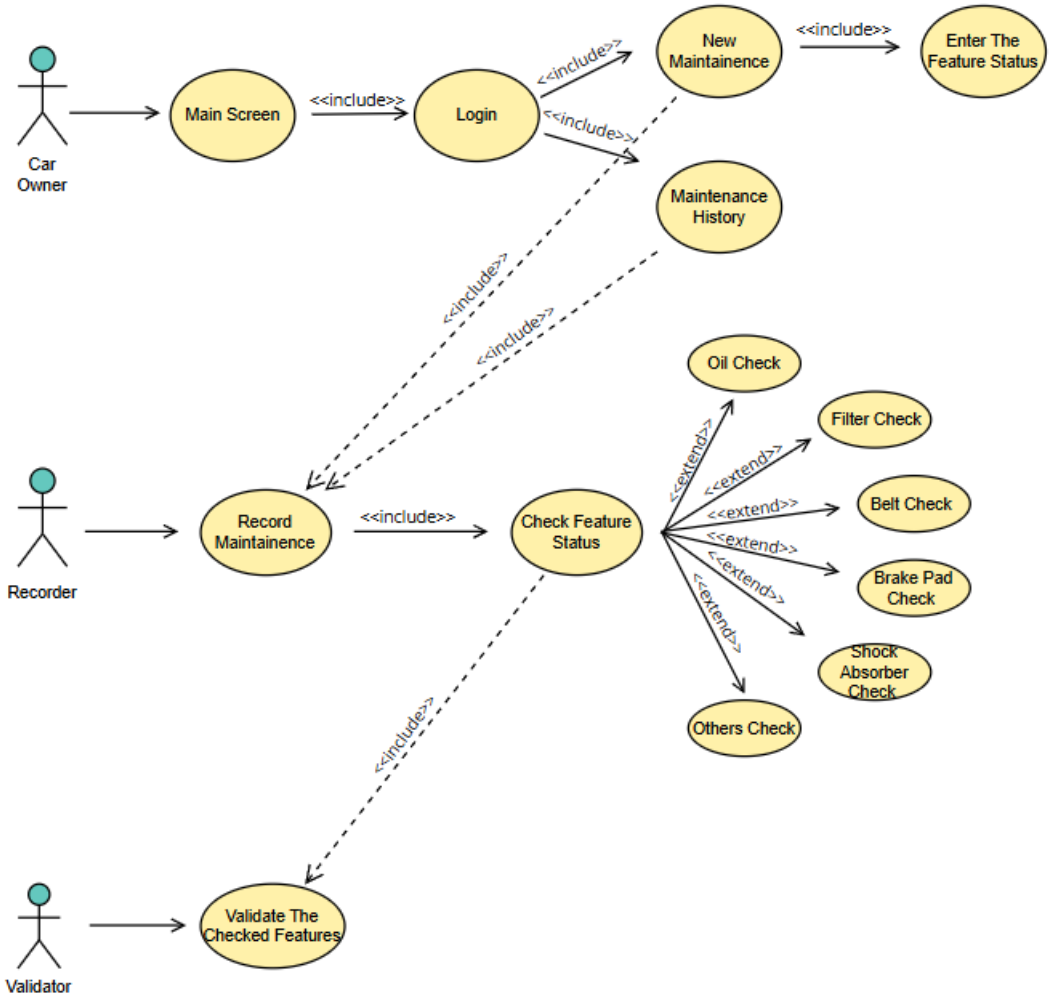

Figure 1. Use Case Diagram of Maintenance System

4.2. Context Diagram

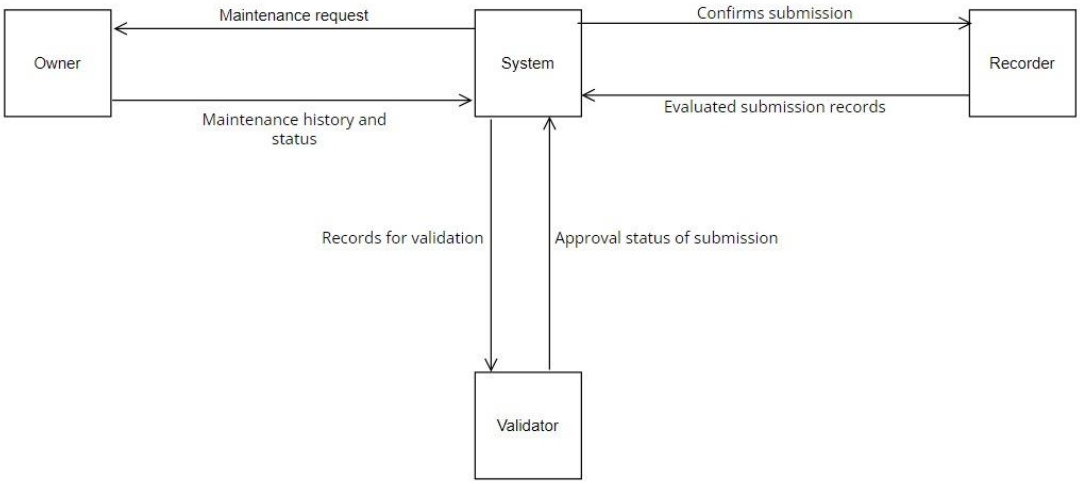

Figure 2. Level 0 DFD of Maintenance System