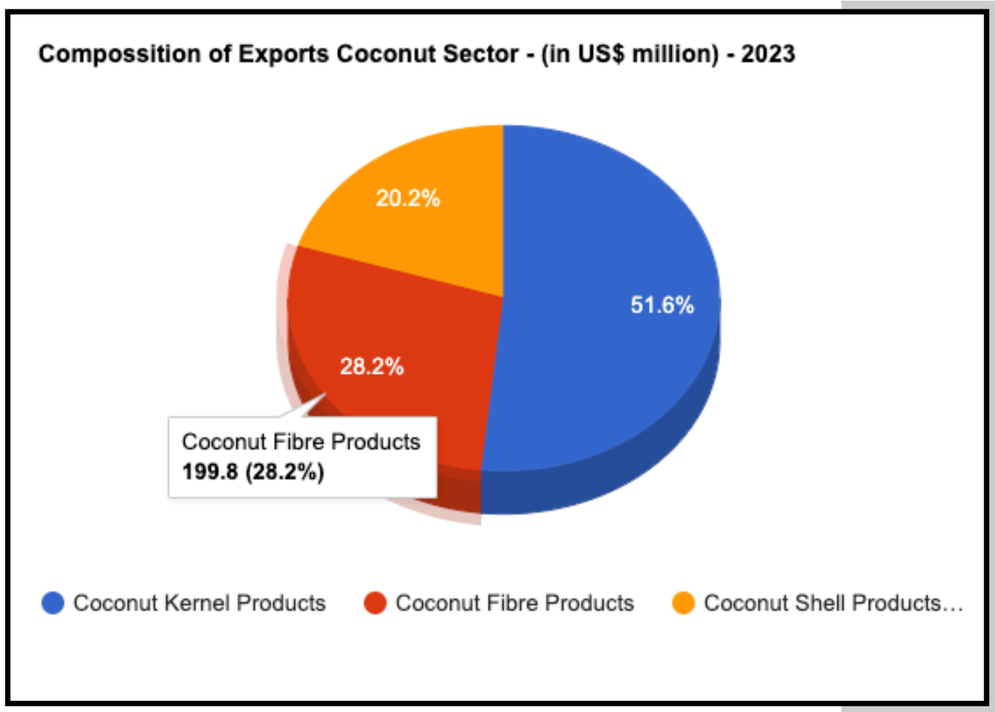
ENHANCING TRANSPARENCY IN THE COCONUT PEAT SUPPLY CHAIN THROUGH A SOFTWARE ENGINEERING APPROACH

Group - 24-25J-313

OUR PROJECT

- 4th largest coconut producer in the world
- The global coco-peat market is valued at USD 2.27 billion (2022) and is projected to reach USD 3.8 billion by 2031 (CAGR of 4.4%).



- Coco-peat, also known as coir pith, is a byproduct of the coconut industry.
- Used extensively in horticulture as a soil substitute and soil conditioner.

Field Visit

CURRENT CHALLENGES

1. Quality Control Issues
2. Inefficiencies in the supply chain
3. Lack of Transparency and Traceability
4. Reluctancy to adopt new technologies

All these information was gathered during the field visit

Research Question

How can transparency be enhanced in the coco-peat supply chain through a software engineering approach integrating blockchain, IoT, and customizable workflow management systems?

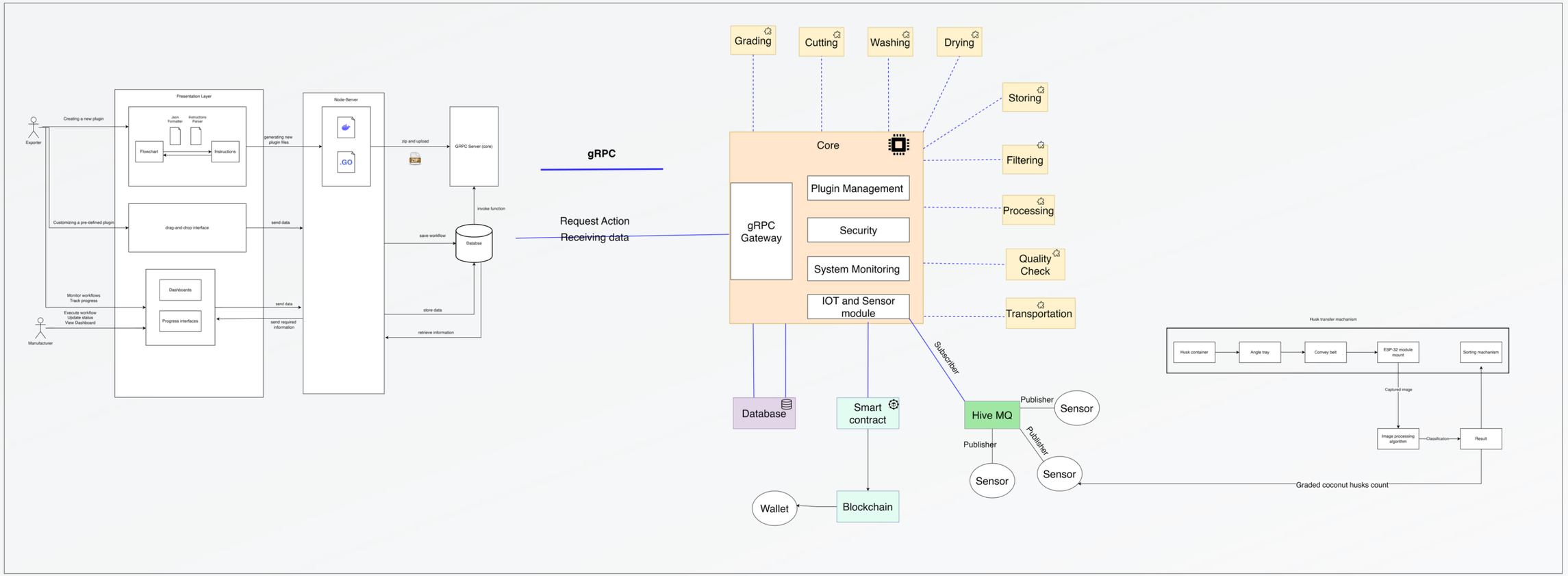
Main Objective

- To develop and implement a comprehensive software engineering solution that enhances transparency in the coco-peat supply chain.

Specific Objectives:

1. Analyze Current Supply Chain
2. Select Appropriate Technologies
3. Design System Architecture
4. Develop Smart Contracts
5. Integrate IoT Devices
6. Customize Workflow Management
7. Test and Validate the System

SYSTEM DIAGRAM



COMMERCIALIZATION

- Sri Lanka Export Development Board (SLEDB)

- Medium to small scale exporters

REFERENCES

1. Sri Lanka Export Development Board, "Sri Lanka Business Portal," Accessed: Aug. 6, 2024. [Online]. Available: https://www.srilankabusiness.com/
2. Alon Green Coir Products. (n.d.). [Online]. Available: https://alongreencoir.com/home. [Accessed: Aug. 06, 2024].
3. YouTube. (n.d.). [Online]. Available: https://www.youtube.com/results?search_query=alongreencoir. [Accessed: Aug. 06, 2024].
4. Biocell Pvt Ltd. (n.d.). [Online]. Available: https://www.srilankabusiness.com/exporters-directory/company-profiles/biocell-pvt-ltd/. [Accessed: Aug. 06, 2024].
5. Biogrow. (n.d.). [Online]. Available: https://fb.watch/sQKds9qyC4/. [Accessed: Aug. 06, 2024].

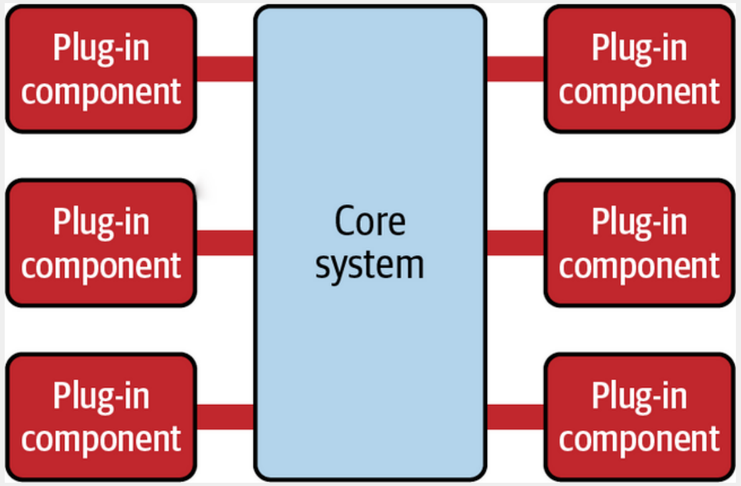
A NOVEL ARCHITECTURAL APPROACH FOR ENHANCING SYSTEM AVAILABILITY AND PLUGIN MANAGEMENT IN A MICROKERNEL ARCHITECTURE

IT21308284 - Vithanage H.D

Specialization - Software Engineering

INTRODUCTION

- Importance of an Architecture?
  - Ensures Scalability and Flexibility
  - Improves Maintainability and Extensibility
  - Optimizes Performance
  - Enhances Security
  - Reduces Risk and Technical Debt



- Available architecture
  a. monolithic architecture
  b. microservice architecture
  c. microkernel architecture
  d. service-oriented architecture

TRADE OFF ANALYSIS

Criteria	Monolithic Architecture	Microservices Architecture	Microkernel Architecture	Service-Oriented Architecture
Modularity	Low	High	High	Moderate
Scalability	Low	High	High	Moderate
Customizability	Limited	High	High	Moderate
Performance	High	Moderate	High	Moderate
Fault Tolerance	Low	High	Moderate	Moderate
Suitability	Small, simple applications	Distributed, large-scale applications	Systems requiring modularity, customizability	Enterprise systems with reusable and loosely coupled services.

[1,8]

RESEARCH GAP

- Integration of Advanced Technologies
  - Most Research focuses on the technologies in isolation rather than as a cohesive system. [2][5][6]
- Case Studies and Practical Implementations
  - Most existing literature focuses on theoretical models or implementation in different contexts. [4][5][6]
- System Stability and Plugin Management
  - lacks of failure recovery methods. [7]

RESEARCH QUESTION

How can a microkernel architecture be designed and implemented to enhance system availability and manage dynamic plugins effectively, while ensuring minimal performance overhead ?

OBJECTIVES

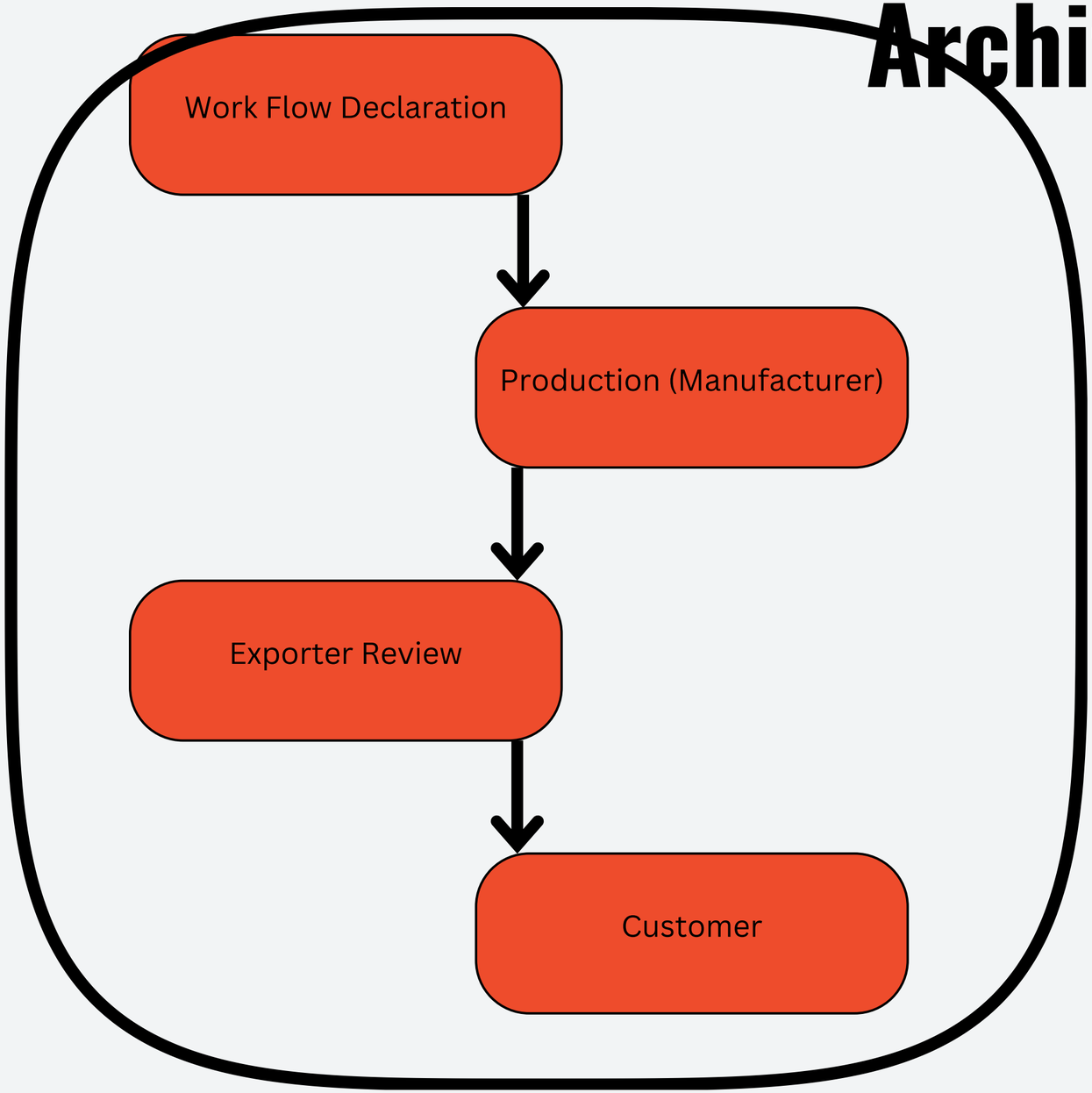
Main Objectives:

Develop a robust core system that integrates Blockchain and IoT technologies

Sub Objectives:

- Design and Development of the core system
- Implementation of Dynamic plugin management
- Integration of Modern technologies
- Evaluation of System performance

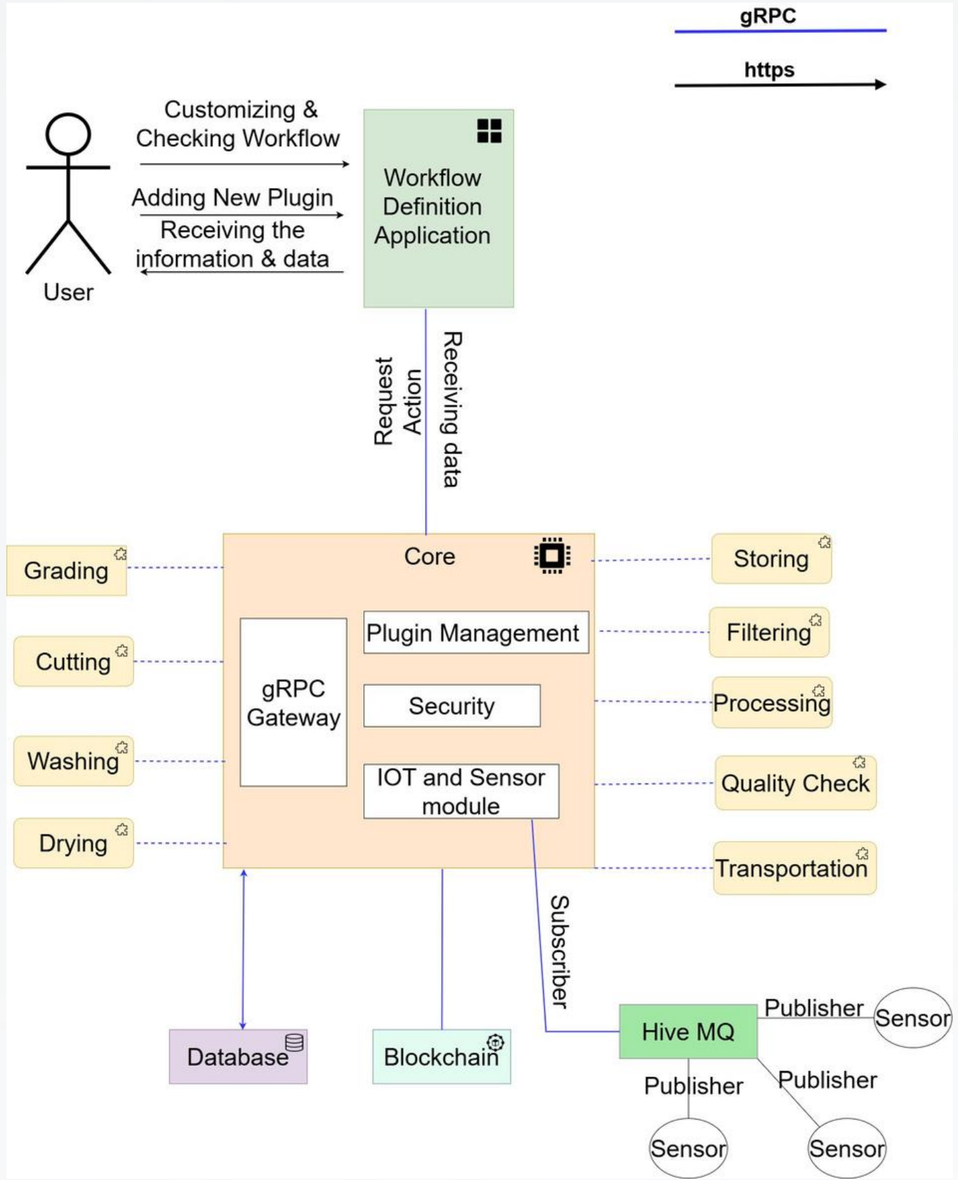
Key Areas Enhanced by The Architetcure



- Integrate all with all other components and act as core of the system for seamless communication.
- Exporter able to create new step of the workflow (plugin) in the runtime of the system and deploy it.
- Users can plug or unplug any step of the workflow.

METHODOLOGY

Component Diagram



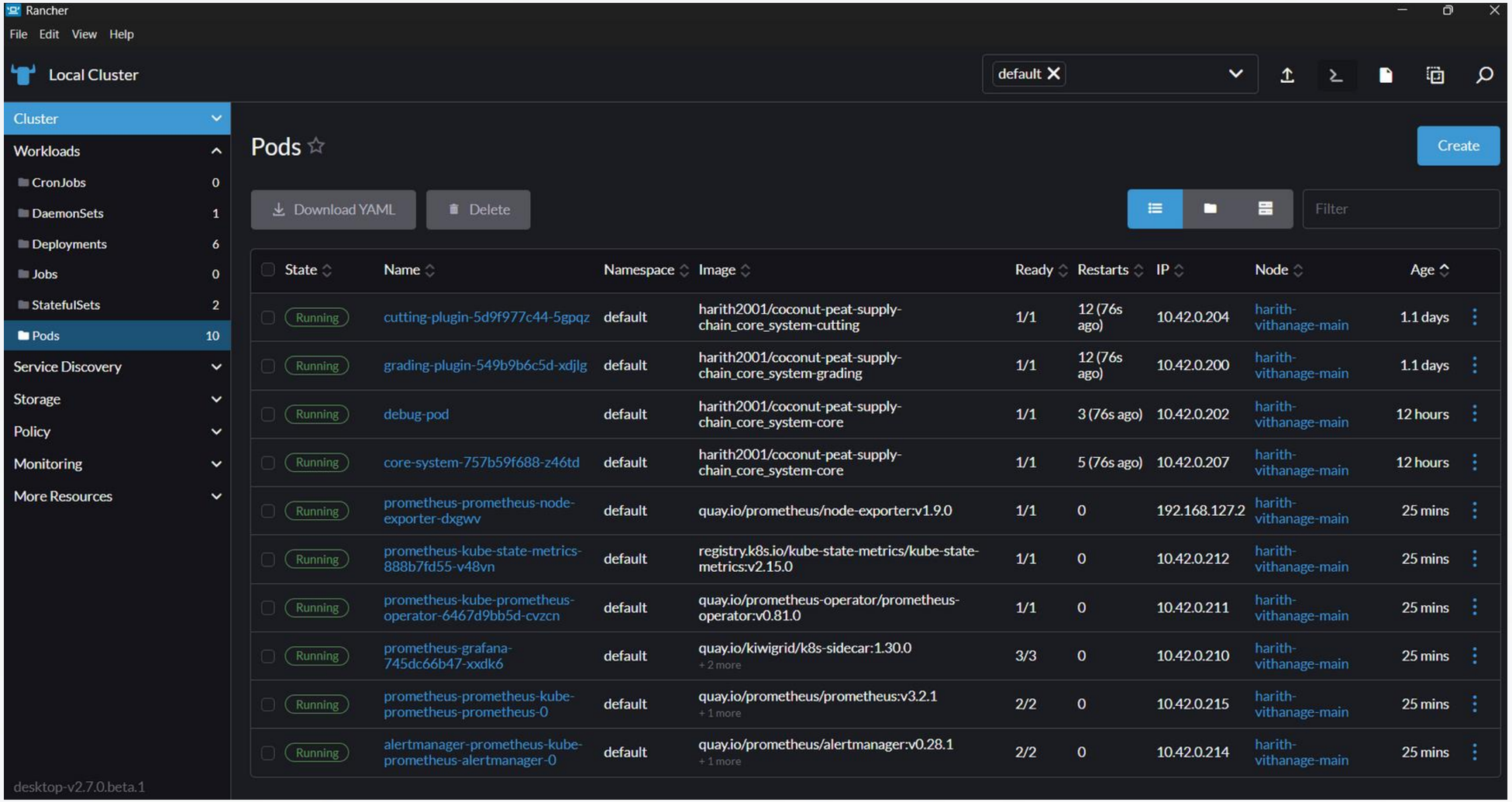
METHODOLOGY

Used Technologies

- Go
- GRPC
- MongoDB
- git
- Hive MQ
- GitHub
- BloomRPC
- Postman
- docker
- K3S
- Github Actions
- prometheus
- rancher dekstop

METHODOLOGY

Evidence for Completion



All the Pods are deployed in the K3s cluster

ISSUES & SOULTIONS

- **Dynamic gRPC Gateway Communication Issue**

Difficulty in orchestrating communication with the correct services dynamically when establishing the gRPC gateway.

- **Detecting and Integrating New IoT Sensors**

The application needed to track all connected sensors and detect when new ones were added dynamically.

- **Kubernetes & Docker Daemon Issues in Runtime Service Deployment**

Faced issues when deploying new services at runtime due to Docker daemon not running and Kubernetes failures inside the core pod.

SE BEST PRACTICES

- Version Control & Code Management
- Modular & Scalable Architecture
- Secure & Efficient Deployment Practices
- Observability & Monitoring
- Database & Performance Optimization

COMPLETION AND FUTURE WORKS

1.Implementation of Plugin Architecture

2.Implementation of Docker for containerization.

3.Implementation of Dynamic plugin management.

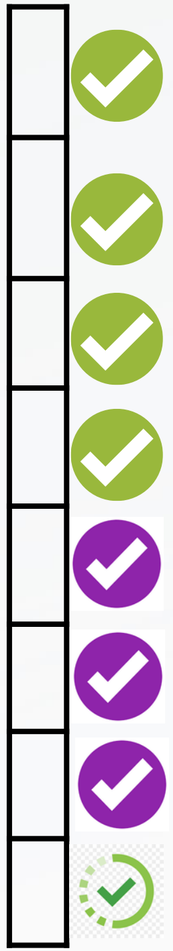
4.Implementation of New Plugin management.

5.Implementation and integration of the sensor module.

6.Implementation and integration of the blockchain module.

7.Implementation of K3S Orchestration.

8.Evaluating the system.



PROJECT REQUIRMENTS

Functional Requirements:

- The Core System should be resilient to errors.
- Plugin loading, execution, and unloading should be dynamic.
- A new plugin should be created in the runtime

User Requirements:

- Usability
- Accessibility
- Customization
- Real-time Data
- Security

Non-Functional Requirements:

- The core system shall respond to user requests within [specify time limit, e.g., 2 seconds]
- The plugin shall be able to handle increasing workloads without significant performance degradation.
- The core system shall employ industry-standard encryption techniques to protect data in transit and at rest.

REFERENCES

1. M. RICHARDS, *SOFTWARE ARCHITECTURE PATTERNS*, 2ND ED. [E-BOOK]. AVAILABLE: HTTPS://LEARNING.OREILLY.COM/LIBRARY/VIEW/SOFTWARE-ARCHITECTURE-PATTERNS/9781098134280/
2. X. YANG, H. SHEN, Z. WANG, AND X. DU, “MICRO-KERNEL OS ARCHITECTURE AND ITS ECOSYSTEM CONSTRUCTION FOR UBIQUITOUS ELECTRIC POWER IOT,” *IEEE ACCESS*, VOL. 8, PP. 200867-200878, 2020.
3. Y. ELSHATER, “MONOLITHIC KERNEL VS. MICROKERNEL,” *IEEE SOFTWARE*, VOL. 37, NO. 2, PP. 123-127, MAR.–APR. 2020.
4. J. LIEDTKE, “ON MICRO-KERNEL CONSTRUCTION,” IN *PROCEEDINGS OF THE FIFTEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES*, 1995, PP. 237-250.
5. R. NEUGEBAUER, “ADAPTIVE OBJECT MANAGEMENT FOR A RECONFIGURABLE MICROKERNEL,” *IEEE TRANSACTIONS ON COMPUTERS*, VOL. 45, NO. 4, PP. 543-550, APR. 1996.
6. S. HOU AND Z. YU, “A MICROKERNEL-BASED WORKFLOW MANAGEMENT SYSTEM,” *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS*, VOL. 43, NO. 6, PP. 1273-1285, NOV. 2013.
7. R. H. DENG, L. JIANG, B. QIN, AND F. BAO, “SCALABLE AND EFFICIENT MIDDLEWARE FOR REAL-TIME EMBEDDED SYSTEMS: A UNIFORM OPEN SERVICE-ORIENTED, MICROKERNEL-BASED ARCHITECTURE,” *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, VOL. 10, NO. 3, PP. 1470-1480, AUG. 2014.
8. V. BENAVENTE, C. RODRIGUEZ, L. YANTAS, R. INQUILLA, I. MOSCOL, AND Y. POMACHAGUA, "COMPARATIVE ANALYSIS OF MICROSERVICES AND MONOLITHIC ARCHITECTURE," IN PROC. 14TH IEEE INT. CONF. ON COMPUTATIONAL INTELLIGENCE AND COMMUNICATION NETWORKS (CICN), LIMA, PERU, 2022, PP. 177-182. DOI: 10.1109/CICN.2022.30.

A VISUAL PROGRAMMING TOOL FOR WORKFLOW MANAGEMENT

IT21291678 - Dehipola H. M. S. N

Specialization - Software Engineering

INTRODUCTION

- Coco-peat manufacturing has **multiple steps** like grading, cutting, washing, and drying.
- Existing workflow tools are **too complex and need more technical expertise**.
- Manufacturers and exporters **struggle to change workflows easily** when producing coco-peat.
- A **simple, user-friendly visual tool** is needed for them to manage workflows —**no programming, or technical knowledge** needed.
- This tool introduces **a visual, domain-specific approach that eliminates the middleman**, enabling seamless workflow customization

RESEARCH GAP

- Existing workflow management tools are not tailored for the unique needs of coco-peat manufacturing.
- Most solutions require software developers or IT experts for modifications.
- Manufacturers and exporters need to adjust workflows frequently, but current tools lack adaptability.
- Non-technical users struggle with programming-based workflow management.

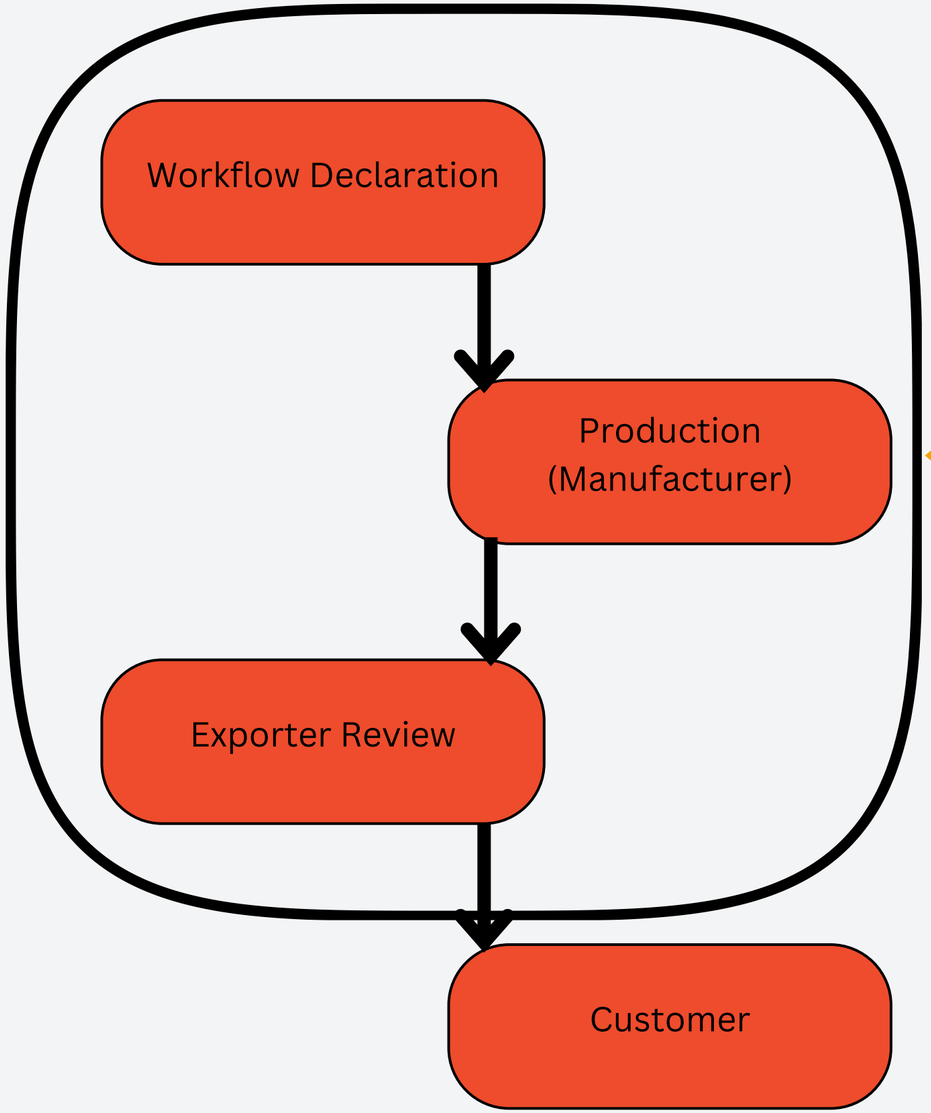
RESEARCH QUESTION

How can a **domain-specific, visual workflow customization tool** be developed to allow **non-technical users** (exporters & manufacturers) to **modify and manage** coco-peat manufacturing workflows without technical expertise?

SOLUTION

- **Allows manufacturers and exporters to easily manage workflows** without IT knowledge.
- **Built with a Domain-Specific Language (DSL)** to create new steps(plugins) in coco-peat manufacturing.
- **Drag-and-Drop Interface** – No coding required, users can visually create and modify workflows.
- **Eliminates the Middleman** – Users can customize workflows without technical experts.
- **Flexible & Adaptable** – Works for different manufacturers, enabling quick workflow changes.

Key Areas Enhanced by This Tool



- Exporters can define and customize workflows visually, eliminating the need for technical expertise.
- Manufacturers follow and execute the customized workflow, ensuring a seamless process without requiring technical expertise.
- Exporters can review workflows seamlessly, ensuring flexibility in manufacturing.

OBJECTIVES

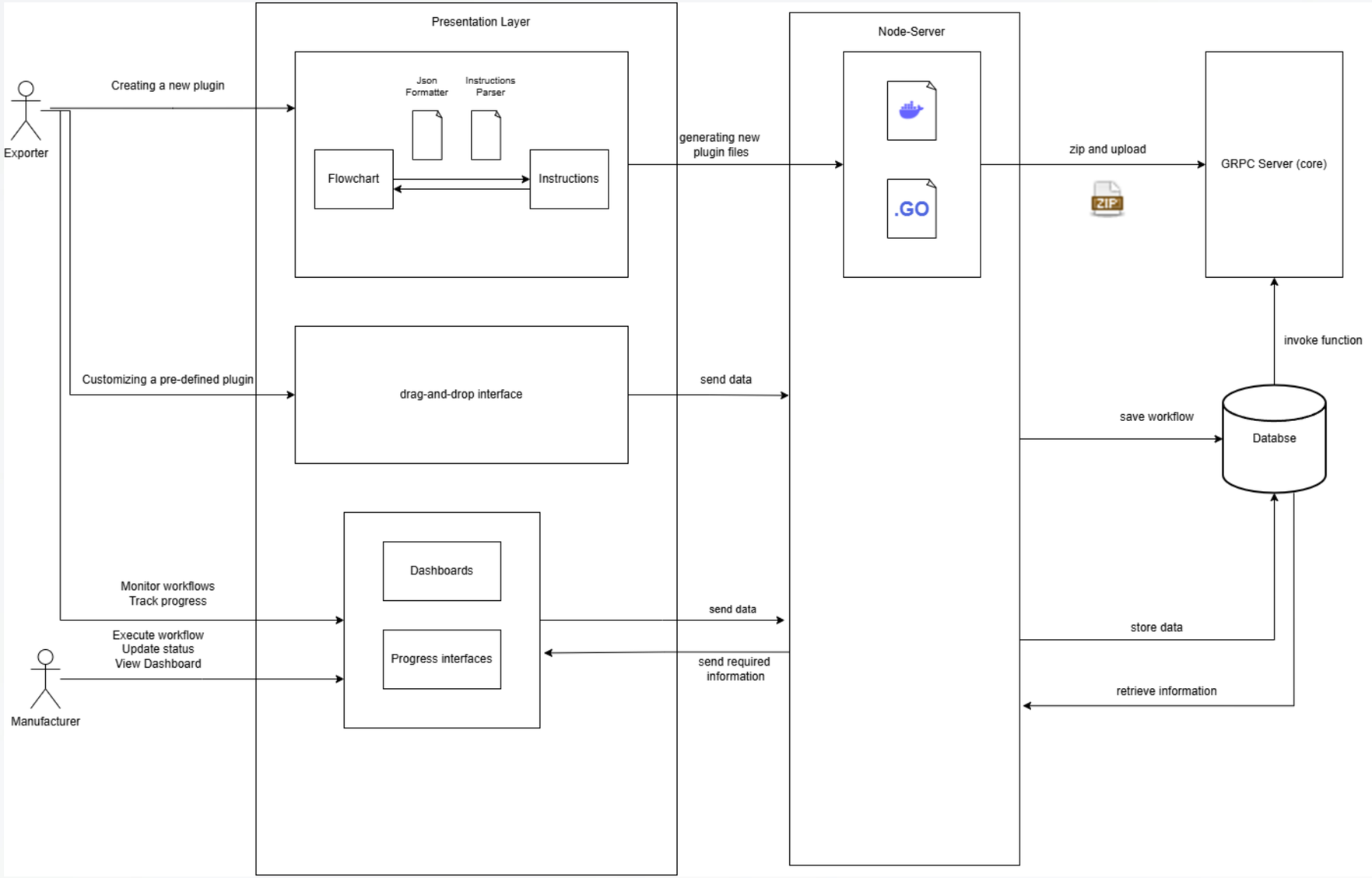
Main Objective: To develop a user-friendly, visual workflow customization tool that enables exporters to define workflows and manufacturers to execute them seamlessly without requiring technical expertise.

Specific Objectives:

- Enable non-technical users to customize workflows.
- Develop a drag-and-drop interface for workflow customization.
- Ensure seamless workflow execution.
- Reduce dependency on technical experts.
- Enhance flexibility in manufacturing processes.

METHODOLOGY

System Diagram



TECHNICAL DECISIONS

REST API for internal communication

- Easy to implement, stateless communication

JWT (JSON Web Tokens):

- No need for server-side session storage.
- Signed and encrypted tokens ensure integrity.

Role-Based Access Control (RBAC):

- Roles (Exporter, Manufacturer) control access to resources.
- Ensures appropriate user access, preventing unauthorized actions.

TECHNICAL DECISIONS

Flowchart for Plugin Creation

- Simplifies complex workflows for non-technical users.
- Ensures consistency in the plugin creation process.

Converting Flowchart to JSON format

- Allows programmatic parsing and logic execution.
- Easily integrates with different systems and platforms.
- Easy to modify and extend the flowchart logic.

Role-Based Access Control (RBAC):

- Roles (Exporter, Manufacturer) control access to resources.
- Ensures appropriate user access, preventing unauthorized actions.

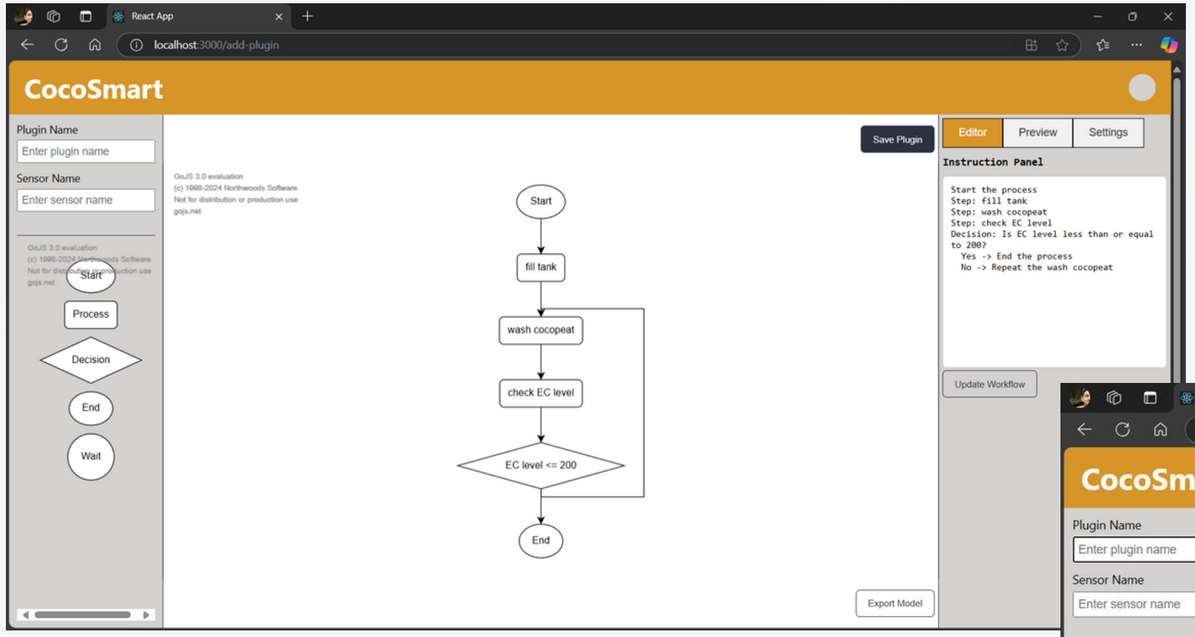
TECHNICAL DECISIONS

Zipped Plugin Files

- Reduces bandwidth and storage requirements.
- Consolidates multiple files into a single, manageable package.
- Ensures integrity and prevents incomplete uploads.

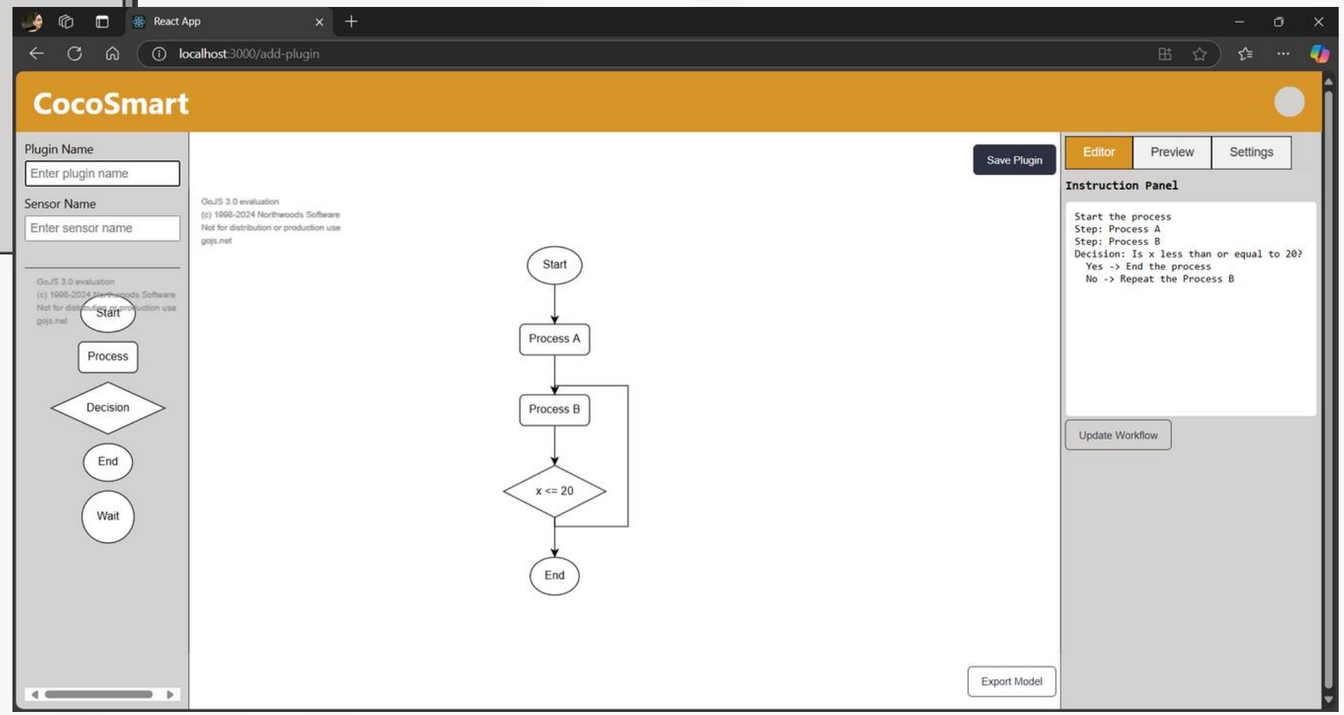
METHODOLOGY

Evidence for Completion



Converting flowchart to instructions

Converting instructions to flowchart



METHODOLOGY

Evidence for Completion

dslGenerator.js

frontend > src > utils > dslGenerator.js > generateDSL

```
export function generateDSL(json) {
  const { nodes, links } = json;
  let instructions = "";

  // Validation Step 1: Check that 'nodes' and 'links' are valid arrays
  if (!Array.isArray(nodes)) {
    throw new Error("Invalid input: 'nodes' must be an array.");
  }
  if (!Array.isArray(links)) {
    throw new Error("Invalid input: 'links' must be an array.");
  }

  // Validation Step 2: Validate nodes
  const nodeKeys = new Set();
  nodes.forEach((node) => {
    if (!node.key) throw new Error(`Node with missing 'key' found.`);
    if (nodeKeys.has(node.key))
      throw new Error(`Duplicate node key found: ${node.key}`);
    nodeKeys.add(node.key);

    // Check for valid category and text
    if (
      !node.category ||
      !["Start", "Process", "Decision", "End"].includes(node.category)
    ) {
      throw new Error(`Invalid node category found: ${node.category}`);
    }
    if (!node.text)
      throw new Error(
        `Node with key ${node.key} is missing 'text' description.`
      );
  });

  // Validation Step 3: Validate links
  links.forEach((link) => {
    if (!link.from || !link.to) {
      throw new Error(
```

Code snippet of instructions generating function

Code snippet of json format generating function

jsonGenerator.js

frontend > src > utils > jsonGenerator.js > generateJSON > lines.forEach() callback

```
export function generateJSON(instructions) {

  const lines = instructions.split("\n");
  const json = { nodes: [], links: [] };

  let lastNode = null;
  let decisionNode = null;
  let yesNode = null;
  let noNode = null;
  let actionNodes = {}; // To track action nodes by label

  lines.forEach((line, index) => {
    if (line.startsWith("Start the process")) {
      json.nodes.push({
        id: 1,
        key: 1,
        type: "Start",
        category: "Start",
        label: "Start",
        text: "Start",
      });
      lastNode = 1;
    } else if (line.startsWith("Step")) {
      const label = line.match(/Step: (.+)/)[1];
      const id = json.nodes.length + 1;
      json.nodes.push({
        id,
        key: id,
        type: "Action",
        category: "Process",
        label,
        text: label,
      });
      if (lastNode) json.links.push({ from: lastNode, to: id });
      lastNode = id;
      actionNodes[label] = id; // Store the action node with its label
    } else if (line.startsWith("Decision")) {
```

METHODOLOGY

Evidence for Completion

Code snippets of new plugin creation function

pluginController.js

backend > controllers > pluginController.js > processAll > processAll

```
const fs = require("fs");
const path = require("path");
const archiver = require("archiver");
const grpc = require("@grpc/grpc-js");
const protoLoader = require("@grpc/proto-loader");
const axios = require("axios");
const { updateFile } = require("../controllers/fileWriter");
const { generateFile } = require("../controllers/fileWriter");

// Load the protobuf
const PROTO_PATH = path.join(
  __dirname,
  "../grpc-node-server/file_service.proto"
);
const packageDefinition = protoLoader.loadSync(PROTO_PATH, {
  keepCase: true,
  longs: String,
  enums: String,
  defaults: true,
  oneofs: true,
});

const fileServiceProto =
  grpc.loadPackageDefinition(packageDefinition).fileservice;

// Create a client
const client = new fileServiceProto.FileService(
  "localhost:50051",
  grpc.credentials.createInsecure()
);

exports.processAll = async (req, res) => {
  try {
    const {
      updateContent,
      goFileContent,
      plugin_name,
```

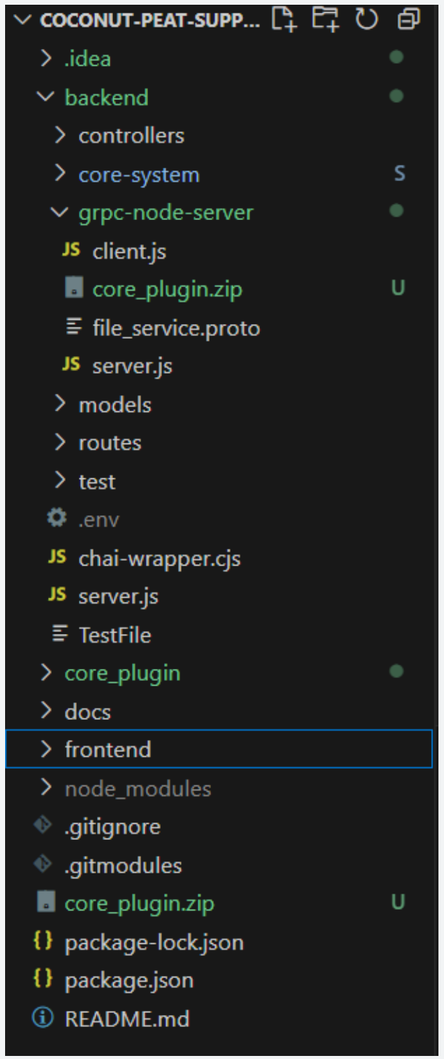
pluginController.js

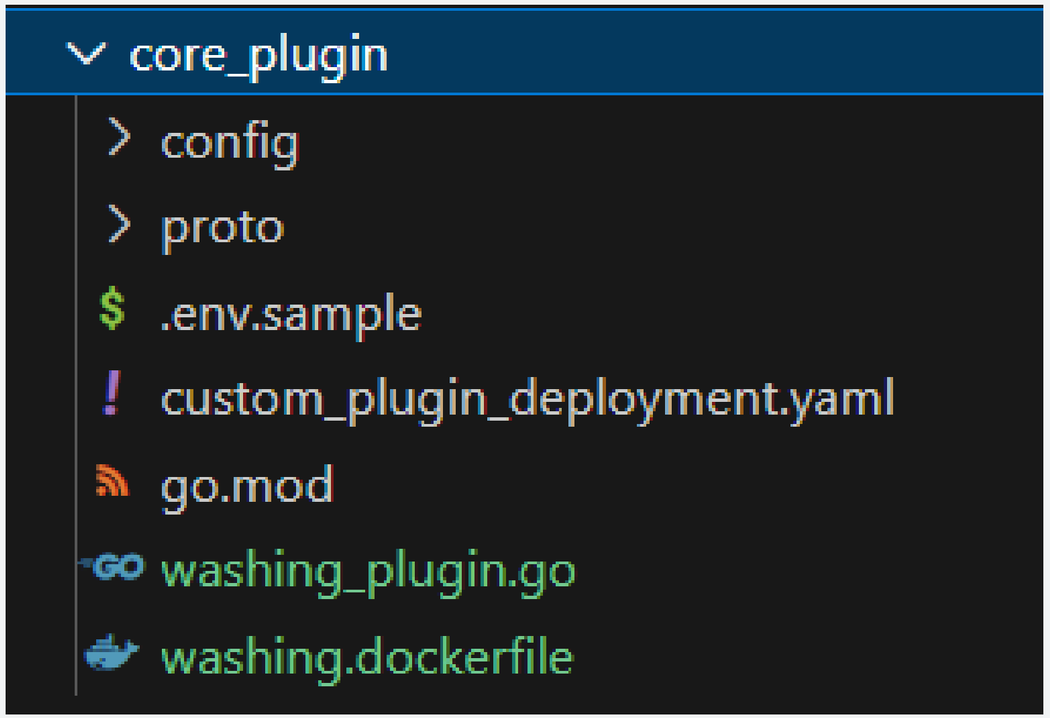
backend > controllers > pluginController.js > processAll > processAll

```
exports.processAll = async (req, res) => {
      .then(() => {
      });
    })
    .catch((err) => console.error("Error zipping folder:", err));
} catch (err) {
  console.error("Error processing all steps:", err);
  res
    .status(500)
    .json({ message: "Error processing all steps", error: err.message });
}
};

// Folder Zip method
function zipFolder(folderPath, outputZipPath) {
  return new Promise((resolve, reject) => {
    // Create a file to stream the archive data to.
    const output = fs.createWriteStream(outputZipPath);
    const archive = archiver("zip", { zlib: { level: 9 } }); // Best compression level

    // Listen for events
    output.on("close", () => {
      console.log(`Zipped ${archive.pointer()} total bytes`);
      resolve();
    });

    archive.on("error", (err) => reject(err));

    // Pipe archive data to the output file
    archive.pipe(output);

    // Append the folder to the archive
    archive.directory(folderPath, false); // `false` prevents nesting in a subfolder

    // Finalize the archive
    archive.finalize();
  });
```

METHODOLOGY

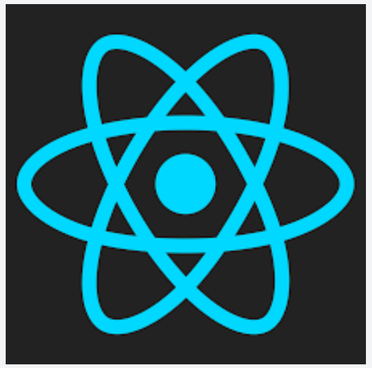
Evidence for Completion

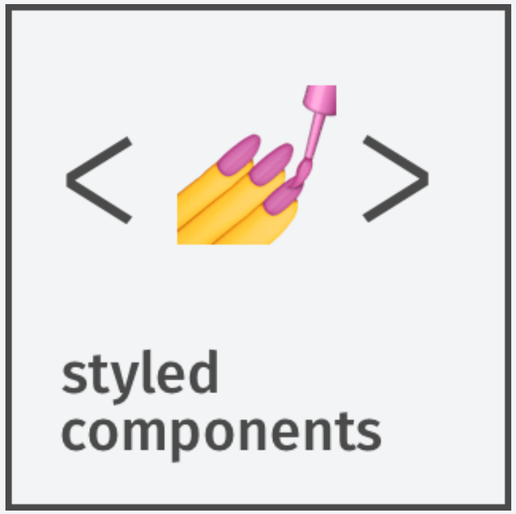




- Folder directory after uploading the zipped file to the grpc server(core)
- Includes new docker file and go file

Technologies





gRPC

PROJECT REQUIREMENTS

Functional Requirements:

- Customizable workflows for different manufacturing processes.
- Drag-and-drop interface for easy workflow setup.
- Real-time updates and monitoring of workflows.
- Role-based access control.

Non-Functional Requirements:

- Fast response time (under 1 second for most actions).
- Strong encryption for sensitive data (in transit and at rest).
- 99.9% system uptime and high availability.

PROJECT REQUIREMENTS

User Requirements:

- Non-technical users must be able to customize workflows easily.
- Users should be able to generate reports and analytics.
- Intuitive interface with minimal learning curve.

System Requirements:

- Compatibility with existing manufacturing hardware and software.
- Scalability to handle multiple concurrent users and large datasets.
- Secure storage of workflow configurations.

SOFTWARE ENGINEERING BEST PRACTICES

- **Modular Code Structure** - Organized components for maintainability and scalability.
- **Efficient State Management** - Ensured only necessary updates trigger UI changes.
- **Secure & Reliable Data Handling** - Implemented data validation before processing workflows.
- **Optimized Performance** - Minimized unnecessary API calls for better efficiency.
- **Robust Error Handling & Logging** - Implemented try-catch blocks & structured logging for debugging.

IMPLEMENTATION CHALLENGES AND SOLUTIONS

Implementation Issue	Solution Implemented
Dragging and dropping workflow steps was not working properly.	Optimized the drag-and-drop logic using React Beautiful DnD for smoother interactions.
Exporters needed to customize workflows without writing code.	Developed a Domain-Specific Language (DSL) and linked it to a visual editor for easy modifications.
Generating new files when creating a new plugin.	Used structured file naming to keep workflow data organized.
Zipping the newly created plugin.	Used Node.js archiver library to handle compression.

Completion of Tasks

- **Developed the visual workflow customization tool.**
- **Integrated Domain-Specific Language (DSL) for new plugin creation functionality.**
- **Implemented file generation & ZIP upload feature.**
- **Established secure communication with core system.**
- **Workflow serialization and core system communication via gRPC implemented.**
- **User authentication API (JWT) and role-based access implemented.**

Future Works

- Enhancing UI/UX for better user experience.
- Perform unit, integration, security and performance testing.
- Complete user documentations.
- Complete instant data updates.

REFERENCES

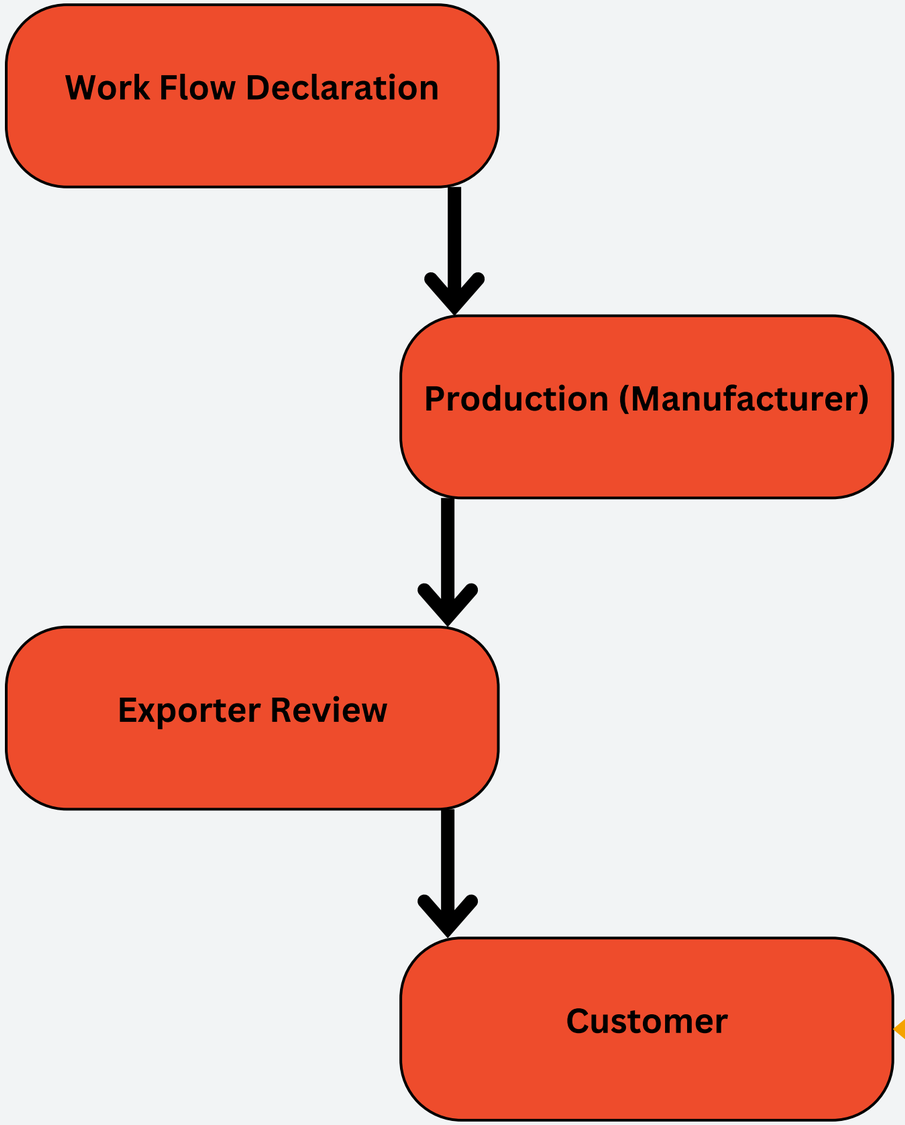
- I. Paoletti and R. S. Naboni, "Robotics in the Construction Industry: Mass Customization or Digital Crafting?" in Advances in Production Management Systems (APMS 2012), Part I, IFIP AICT 397, pp. 294-300, 2013.
- W.-H. Chen and M. J. Lercher, "ColorTree: A batch customization tool for phylogenic trees," BMC Research Notes, vol. 2, no. 155, pp. 1-4, Jul. 2009.
- C. Datta, C. Jayawardena, I. H. Kuo, and B. MacDonald, "RoboStudio: A Visual Programming Environment for Rapid Authoring and Customization of Complex Services on a Personal Service Robot," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 2012, pp. 7-12.
- R. C. Murphy et al., "Design, implementation and field tests of a socially assistive robot for the elderly: HealthBot version 2," in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Jun. 2012.
- A. Kaspar, L. Makatura, and W. Matusik, "Knitting Skeletons: A Computer-Aided Design Tool for Shaping and Patterning of Knitted Garments," Aug. 2019.
- A. Nourani, H. Ayatollahi, and M. S. Dodaran, "Mobile task management tool that improves workflow of an acute general surgical service," ANZ Journal of Surgery, vol. 85, no. 9, pp. 760-765, Sep. 2015.
- Z. Hou and Z. Yu, "Research of the Workflow Management System Based on Microkernel," Journal of Theoretical and Applied Information Technology, vol. 47, no. 1, pp. 266-271, Jan. 2013.
- A. Fillmore et al., "Socially Assistive Robot HealthBot: Design, Implementation, and Field Trials," IEEE Systems Journal, vol. 10, no. 3, pp. 1-8, Aug. 2016.
- R. Dhond et al., "ProjectFlow: a configurable workflow management application for point of care research," JAMIA Open, vol. 4, no. 3, pp. 1-8, 2021.

INTEGRATION OF BLOCKCHAIN

IT21576966 - Weedagamaarachchi K.S
Specialization - Information Technology

CURRENT SUPPLY CHAIN CHALLENGES

- Lack of transparency.
- Issues with traceability.
- Quality control problems.
- Inefficiencies and fraud risks.

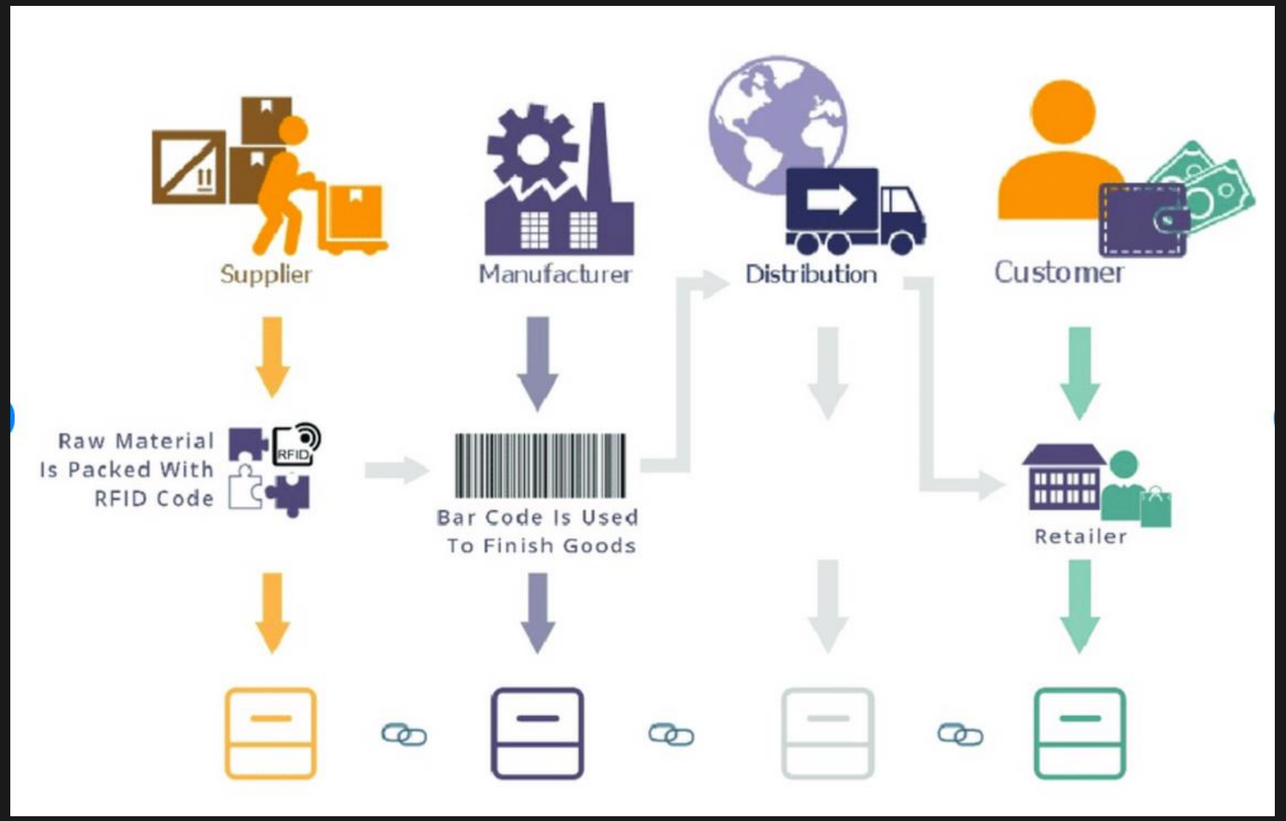


- A separate platform with blockchain for the coco peat supply chain is crucial to ensure transparency, traceability, and trust among stakeholders.
- Blockchain provides an immutable ledger recorded in a tamper-proof manner.
- Eiminates data manipulation, ensuring that manufacturers, exporters, and customers can verify the authenticity of coco peat quality, sustainability claims, and compliance with regulations.

What is Blockchain ?

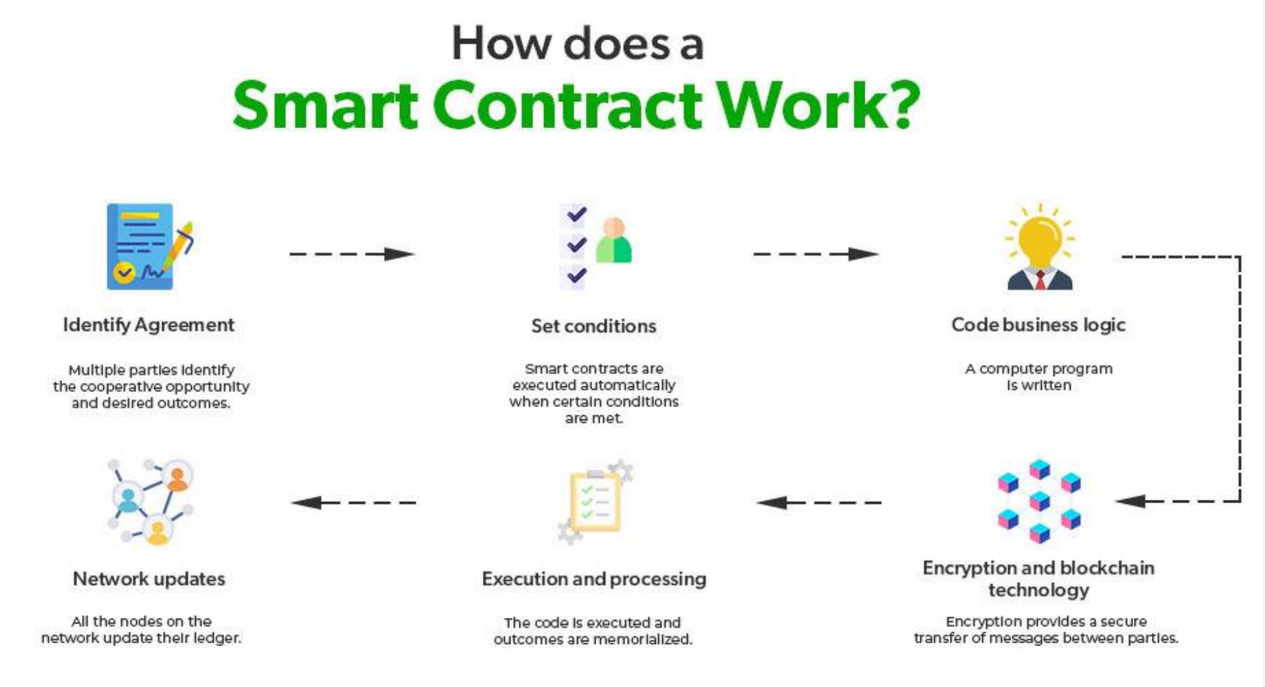
Blockchain is a type of digital ledger that records transactions across multiple computers in a way that ensures the data cannot be altered

1. Decentralized
2. Immutable
3. Transparency



What is a Smart Contract ?

A smart contract is a program that exists on the blockchain that run when some predetermined conditions are met.



What is Gas Fee ?

- transaction cost required to perform operations on a blockchain network, particularly on Ethereum and other smart contract platforms.
- It compensates network validators (miners or stakers) for using computational resources to process transactions and execute smart contracts.

RESEARCH GAP

While existing studies and best practices address gas efficiency in Solidity smart contracts, there is limited focus on systematically analyzing how code complexity in **common design patterns** directly affects gas usage.

RESEARCH PROBLEM

- Lack of transparency and traceability in the coco-peat supply chain.
- Traditional approaches fail to ensure data integrity and real-time visibility.
- How to effectively integrate blockchain to enhance transparency and traceability in the coco-peat supply chain.

How can the application of techniques to reduce code complexity in existing smart contract design patterns enhance gas efficiency without compromising functionality and security in Solidity-based blockchain applications?

OBJECTIVES

Main Objective: To develop and evaluate techniques for reducing code complexity in Solidity smart contract design patterns, thereby enhancing gas efficiency while maintaining functionality and security.

Specific Objectives:

1. Identify key pain points in the current supply chain.
2. Select an appropriate blockchain platform.
3. Design smart contracts.
4. Implementing blockchain.
5. Test and validate the system.

Methodology

Overview of Blockchain Technologies

Platform	Features	Pros	Cons
Ethereum	Decentralized, smart contracts, public public blockchain	Wide adoption, robust smart contract functionality	Scalability issues, **high transaction fees**
Hyperledger Fabric	Permissioned blockchain, modular architecture	High scalability, privacy features, features, enterprise-focused	More complex setup, less decentralized
Corda	Permissioned blockchain, designed for designed for financial institutions	High privacy, efficient for bilateral transactions	Limited to financial use cases, less flexibility

Smart Contract Testing

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract ProductFactoryComplex {
    address public owner;
    mapping(address => bool) public admins;
    mapping(address => bool) public productsRegistry;

    struct Product {
        string name;
        uint256 state; // 0: Created, 1: Shipped, 2: Delivered
        address owner;
    }

    mapping(address => Product) public products;

    event ProductCreated(address indexed productAddress, string name);
    event StateUpdated(address indexed productAddress, uint256 newState);

    modifier onlyAdmin() {
        require(admins[msg.sender], "Not an admin");
        _;
    }

    modifier onlyOwnerOrAdmin() {
        require(msg.sender == owner || admins[msg.sender], "Not authorized");
        _;
    }

    constructor() {    947081 gas 895400 gas
        owner = msg.sender;
        admins[owner] = true;
    }

    function addAdmin(address _admin) external onlyOwnerOrAdmin {    29258 gas
        admins[_admin] = true;
    }

    function createProduct(string memory _name, address _productAddress) external onlyAdmin {    infinite gas
        require(!productsRegistry[_productAddress], "Product already exists");
        products[_productAddress] = Product(_name, 0, msg.sender);
        productsRegistry[_productAddress] = true;
        emit ProductCreated(_productAddress, _name);
    }

    function updateState(address _productAddress, uint256 _newState) external onlyAdmin {    infinite gas
        require(productsRegistry[_productAddress], "Product not registered");
        require(_newState > products[_productAddress].state, "Invalid state transition");
```

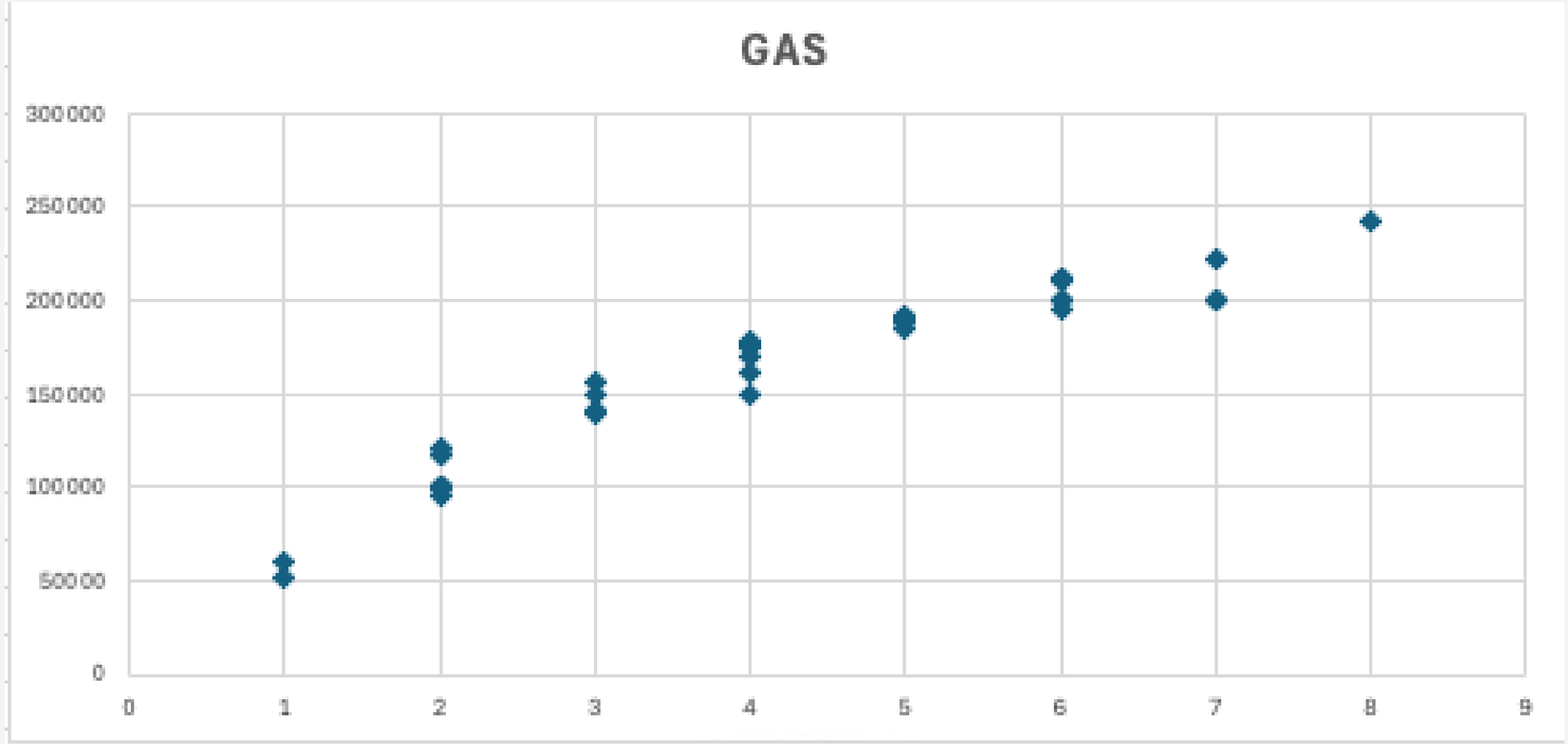
A product creation contract

Calculating the Code Complexity

$$\text{Cyclomatic Complexity} = E - N + 2P$$

Where:

- E = Number of edges in the control flow graph.
- N = Number of nodes in the control flow graph.
- P = Number of connected components (usually 1 for a single function).



	A	B
1	Complexity	Gas
2	3	156032
3	2	120444
4	4	178478
5	5	190901
6	6	212456
7	2	99567
8	4	160834
9	2	117879
10	5	188976
11	2	95787
12	4	175432
13	6	200874
14	5	191754
15	2	100437
16	7	222054
17	1	50921
18	4	169239
19	8	241753
20	5	189634
21	3	149765
22	4	176909
23	2	118076
24	6	210897
25	7	200342
26	3	139654
27	7	200898
28	4	149004
29	5	185798
30	6	199987
31	2	100898
32	5	188432
33	6	195043
34	3	141233
35	1	60921

With this,

Code Complexity $\propto$ Gas amount

Techniques to reduce the cyclomatic complexity

1. Calling outer contract functions
2. Avoiding deep nesting
3. Reducing data structure
   complexity
4. Minimizing data types
5. Eleminating validation logics
6. Bulk operations
7. Refactoring repeated logic
8. Leveraging libraries (Solidity)

Efficiency

With the implementation of multiple techniques
The gas amount went from
1,224,933 to 880,161

28% decrease the gas amount spent for the deployment of this contract

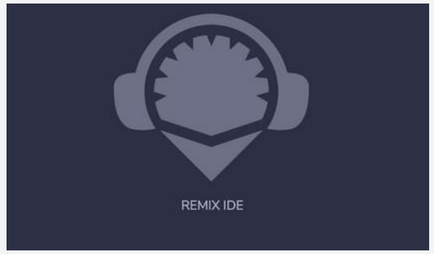
Methodology

Technology used

The selected Blockchain technology

Etherium development environment

Smart contract writing language



Solidity deployment Environment

Digital wallet

Version Controlling

COMPLETION AND FUTURE WORKS

1. **Deploying the test blockchain network**
2. **Deployment and testing of techniques to reduce code complexity**
3. **Deployment of smart contracts**
4. **Implementing the gas efficient smart contracts**
5. **Implementation Blockchain network to the coco peat supplychain.**
6. Implementation of the test network
7. Evaluating the system.

Functional Requirents

- Real-time data capture.
- Immutable ledger.
- Smart contract automation.
- User access control.
- Traceability interface.

Non-Functional Requirements

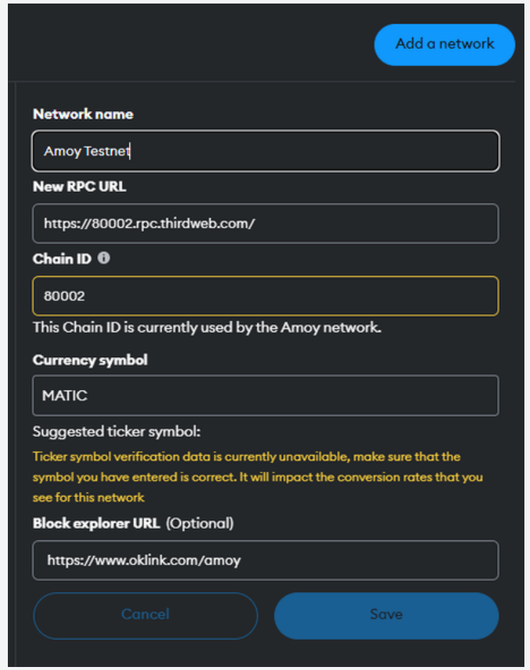
- The application should process at least 100 transactions per second during peak usage
- The system should have a recovery time objective (RTO) of 15 minutes.

Set backs and Decisions

- Polygon amoy connection errors - Used Etherium test network (holesky) instead.

- Error while conneting to the wallet - Meta mask, have to refresh the screen everytime

METAMASK



SE BEST PRACTICES

- Version Control & Code Management
- Follow Secure Smart Contract Development
- Secure & Efficient Deployment Practices
- Observability & Monitoring
- Optimization

References

1. Sawant, S. R., Shah, S., & Poladia, J. A. "Chaining Success: How Blockchain Reshapes the Landscape of Supply Chain," 2023 6th International Conference on Advances in Science and Technology (ICAST), 2023, pp. 94-99. doi: 10.1109/ICAST59062.2023.10455024.
2. Sangeetha, A. S., Shunmugan, S., & Murugan, G. "Blockchain for IoT Enabled Supply Chain Management - A Systematic Review," Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2020, pp. 48-52. doi: 10.1109/I-SMAC49044.2020.9243481.
3. Mohan, M., Rajakumar, R., & Arumugam, K. "Blockchain Enabled Secure Agri-Goods Traceability using RFID in Supply Chain Management," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 8, no. 8, pp. 456-462, June 2019.
4. Poladia, J. A., Shah, S., & Sawant, S. R. "Blockchain for IoT Enabled Supply Chain Management: A Systematic Review," International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 3, pp. 230-235, Sept. 2019.
5. Sawant, S. R., Shah, S., & Poladia, J. A. "Chaining Success: How Blockchain Reshapes the Landscape of Supply Chain," 2023 6th International Conference on Advances in Science and Technology (ICAST), 2023, pp. 94-99. doi: 10.1109/ICAST59062.2023.10455024.
6. Thangaraj, A., & Krishnan, S. "Data Protection and Export for Transaction Ledgers in Permissioned Blockchain Platforms," Journal of Advanced Research in Dynamical and Control Systems, vol. 12, no. 4, pp. 123-130, April 2020.
7. Rajakumar, R., & Mohan, M. "Role of Blockchain Technology in Supplychain Management," International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 3, pp. 98-102, Sept. 2019.
8. Arumugam, K., & Thangaraj, A. "Revolutionizing Secure Commercialization in Agriculture Using Blockchain Technology," International Journal of Advanced Science and Technology (IJAST), vol. 29, no. 9, pp. 456-462, June 2020.
9. Krishnan, S., & Arumugam, K. "Factors Influencing the Effective Information Sharing in Sri Lankan Export-Led Manufacturing Supply Chains," Journal of Industrial Engineering and Management, vol. 14, no. 3, pp. 450-465, July 2021.

COCONUT HUSK GRADING SYSTEM USING IMAGE PROCESSING AND COMPUTER VISION. IOT AUTOMATION FOR COCO PEAT QUALITY ASSURANCE

IT21576966 - Manditha K.D
Specialization - Information Technology

BACKGROUND

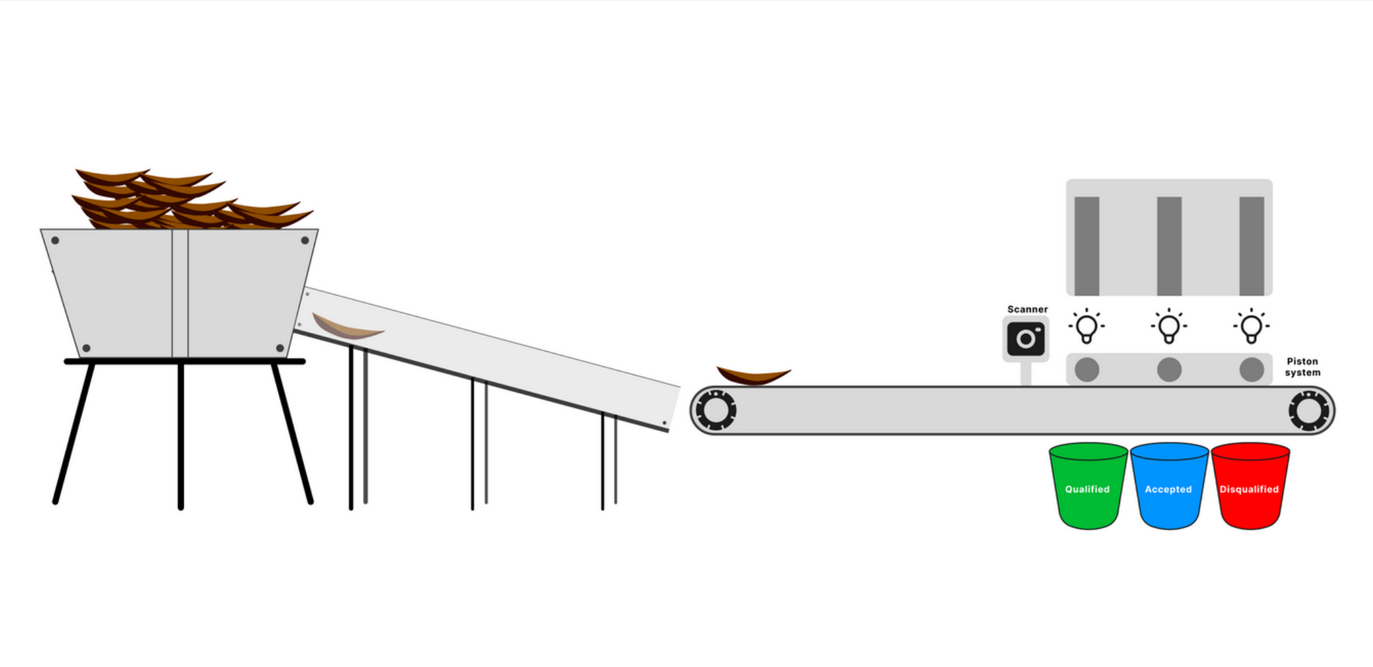
- Using qualified husks is necessary to produce high-quality cocopeat products.
- Finding quality coconut husks is a challenge

PROBLEM DEFINITION

- Lack of knowledgeable workers in the industry.
- Human errors

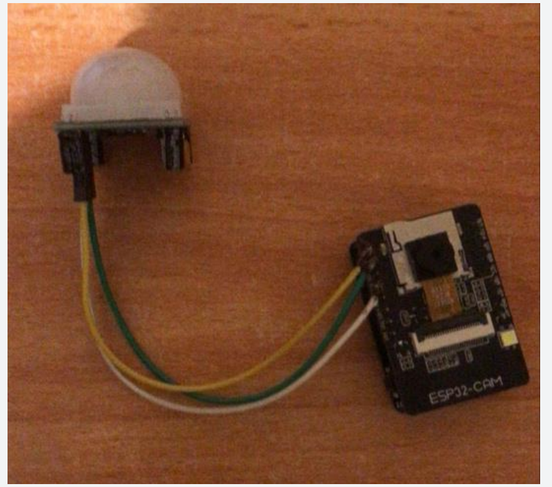
PROPOSED SOLUTION

Automating the coconut husk grading system cost-effectively



PP1 PROGRESS

- Identified the suitable image processing technique.
- Created the basic image processing algorithm for the husk grading
- Identified the suitable hardware solution cost-effectively.
- Built a basic simple model to showcase the automation of the husk grading system.

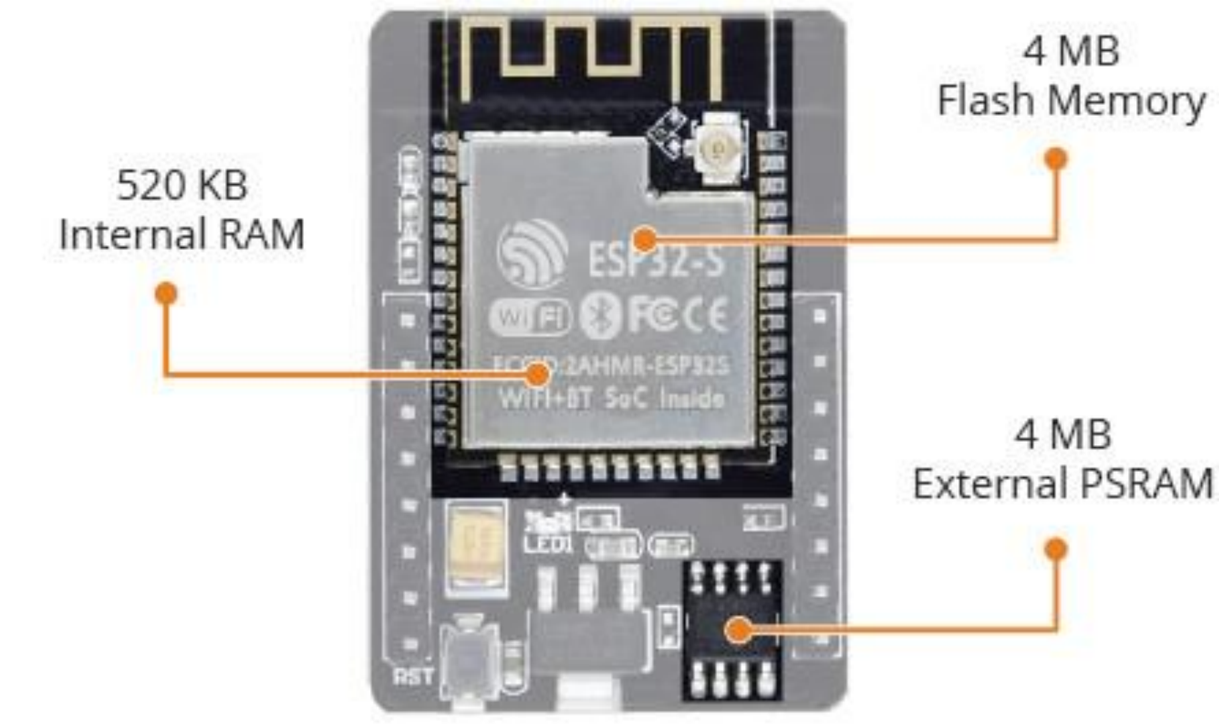


PP2 OBJECTIVES

- Optimizing the image processing algorithm in order to be compatible with the ESP-32 module.
- Building a system to detect a husk, capture an image, and run the image processing algorithm.
- Track a count and publish data to the Hive MQ

METHODOLOGY

INEFFICIENT MEMORY IN THE ESP 32 CAM MODULE



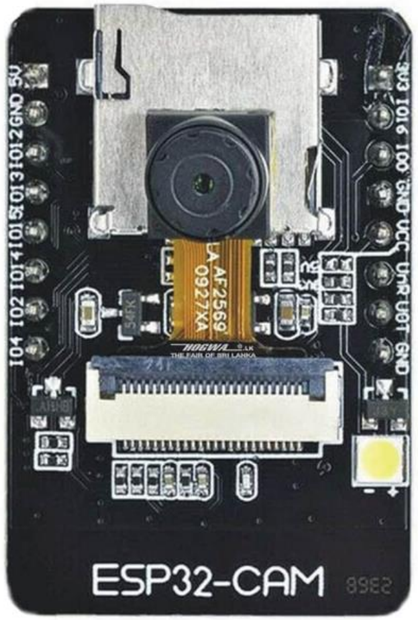
METHODOLOGY

USING NORMAL ESP 32 MODULE FOR OBJECT DETECTION AND OTHER TASKS

- object detection
- send capture command

METHODOLOGY

USING ESP 32 CAM MODULE FOR IMAGE PROCESSING



- Waiting for capture command from the normal esp 32 (memory issue occurred)
- capture image
- run the algorithm
- send the result to the normal esp 32 (memory issue occurred)

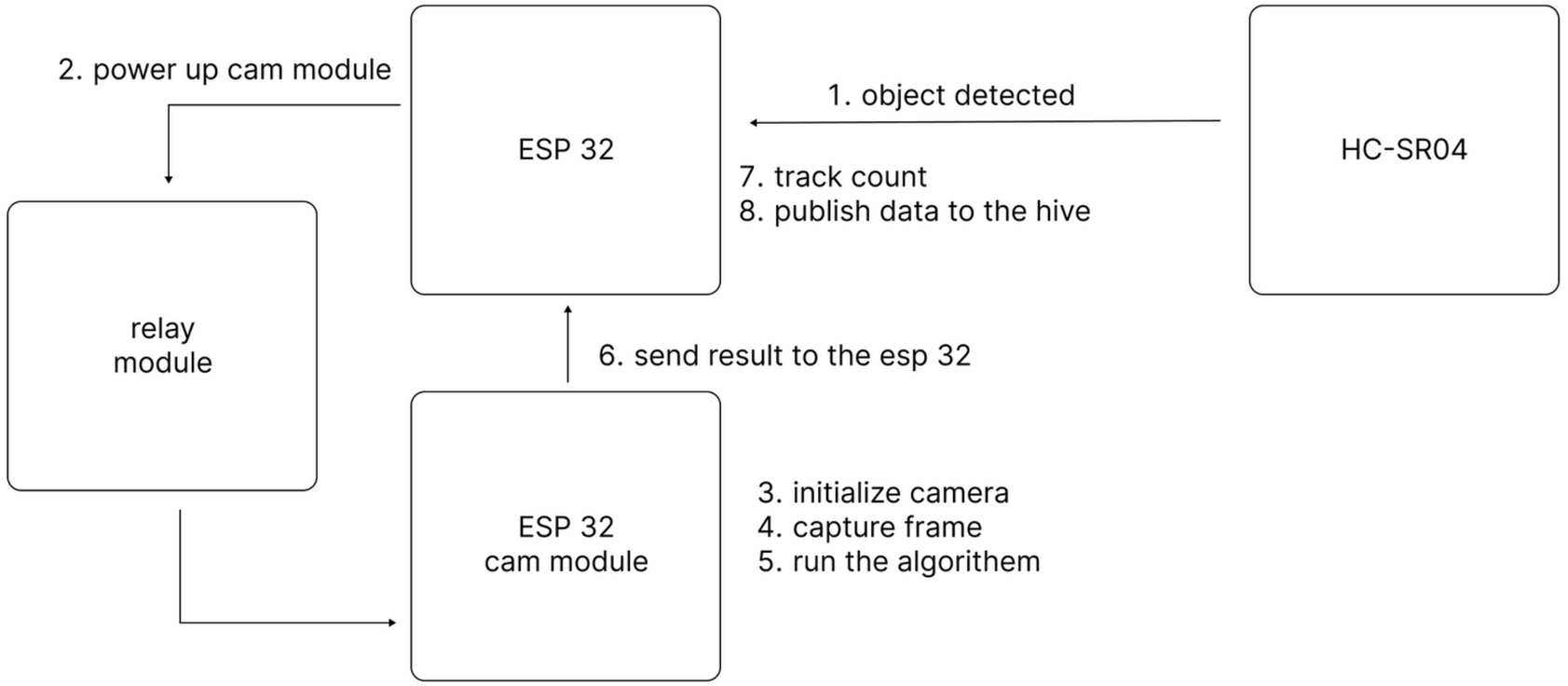
METHODOLOGY

APPLYING A POWER MANAGEMENT METHOD

- Default no power to the cam module
- once an object is detected,
- powering up the cam module
- cut power once done.

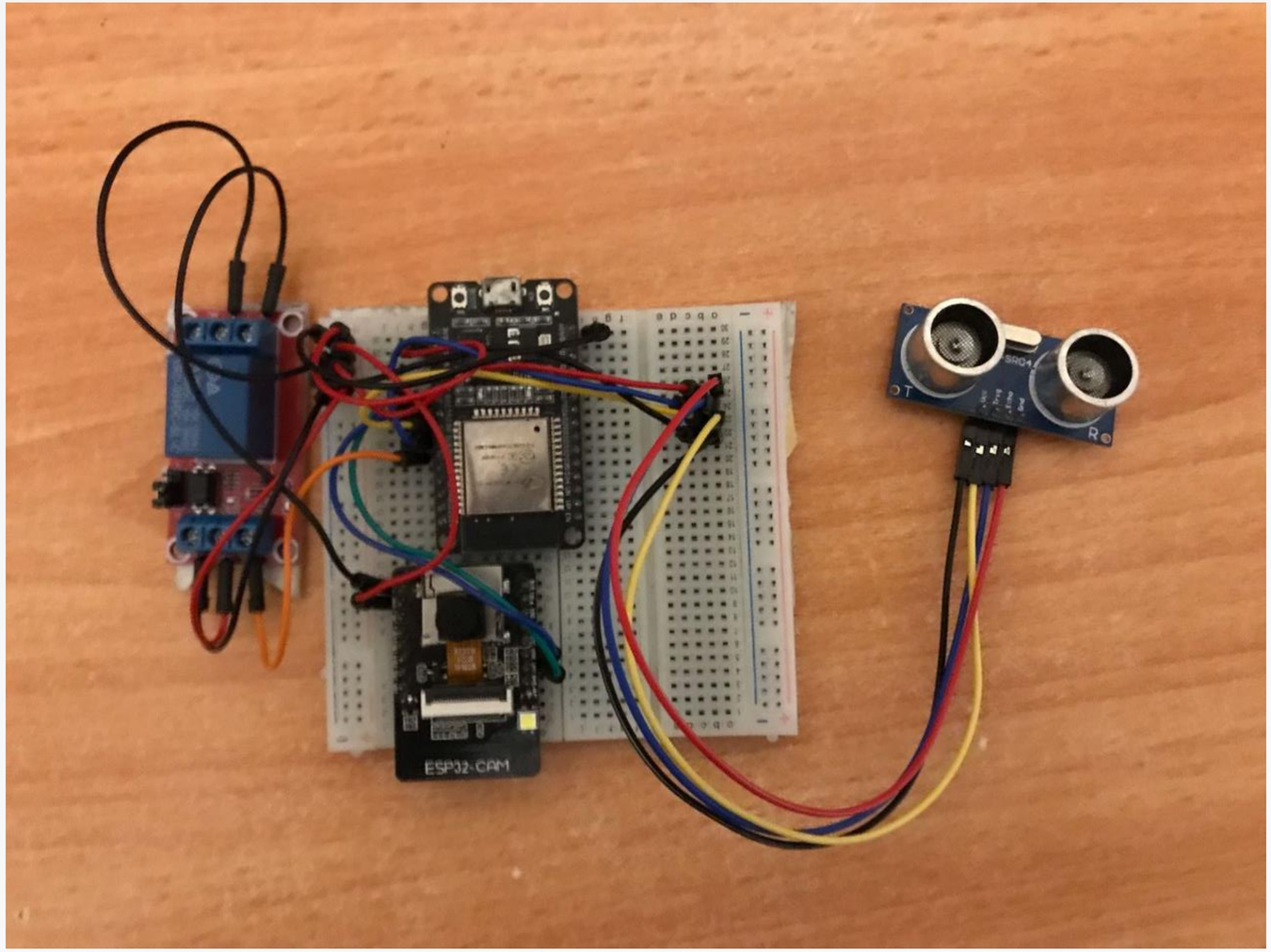
METHODOLOGY

FINAL MODEL



METHODOLOGY

WORKING MODULE



ISSUES & LIMITATIONS

- Low camera quality
- Low FPS
- Memory allocation issues
- Low process timing
- Accuracy

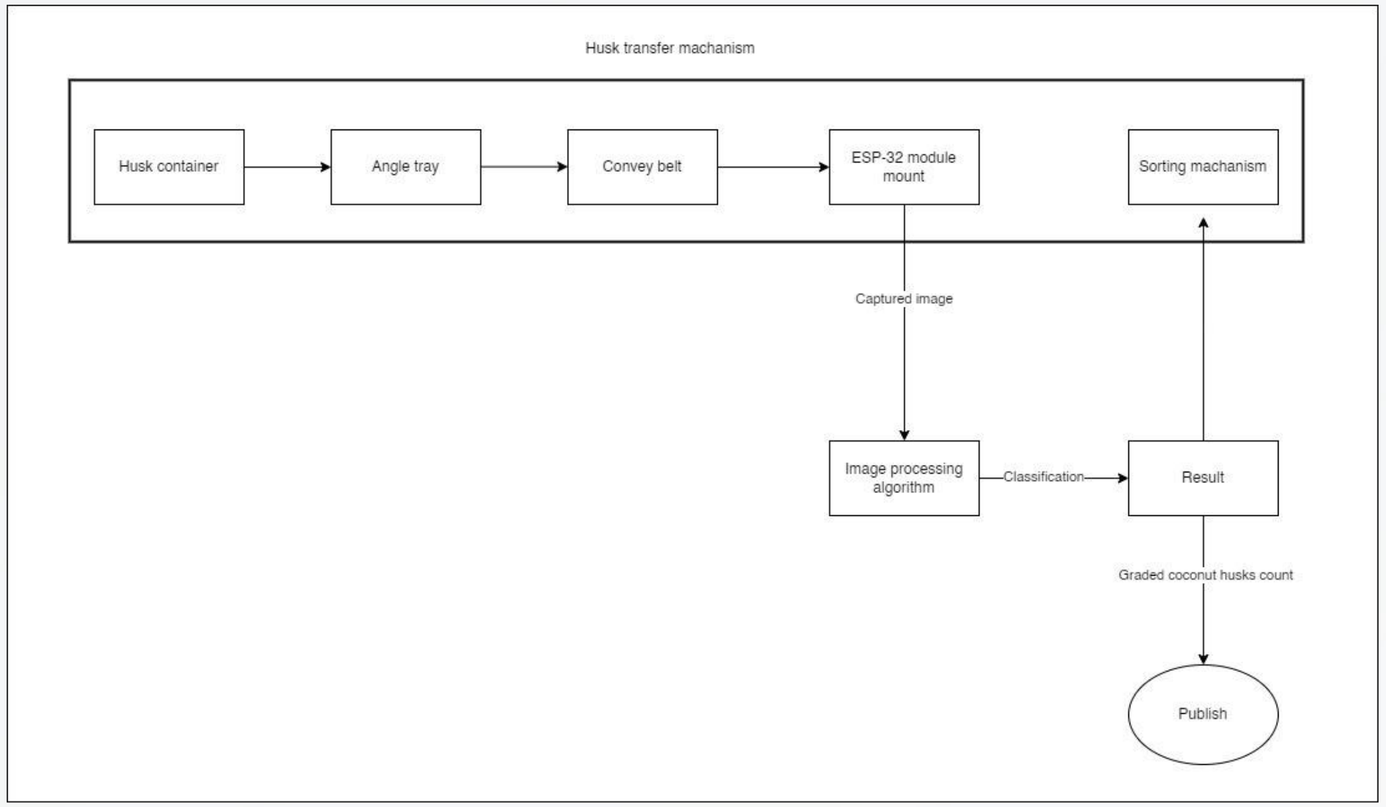
FUTURE WORKS

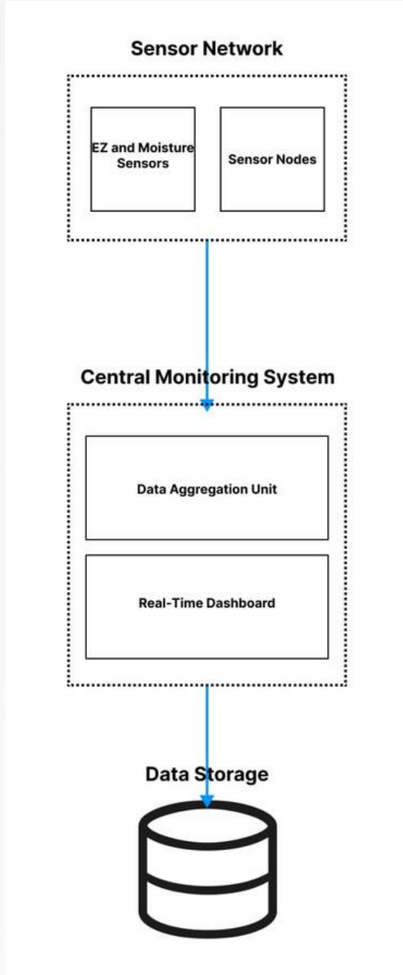
- Optimizing the husk grading system to be more efficient and accurate.
- Build up EZ level automation system.

SE BEST PRACTICES

- Version control
- Readable and clean code
- Modular Code Design
- Timing and State Management
- Resource Management
- Power Management Considerations

SYSTEM DIAGRAM





TECHNIQUES AND TECHNOLOGIES

TECHNIQUES

- Downscaling Techniques
- Color Space Conversion
- HSV-Based Classification

TECHNOLOGIES

- UART (Serial Communication)
- Arduino

FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

FUNCTIONAL

- Image capturing
- Image processing and classification
- Real time processing
- Object detection

NON - FUNCTIONAL

- Accuracy
- Reliability
- Scalability
- Compatibility

REFERENCES

1.G. S. CHIU AND T. L. FONG, “FRUIT CLASSIFICATION USING IMAGE PROCESSING,” PROCEDIA COMPUTER SCIENCE, VOL. 133, PP. 150-156, 2018.

2.R. K. GUPTA AND A. S. ANJUM, “DETECTION OF FIRE USING IMAGE PROCESSING TECHNIQUES WITH LUV COLOR SPACE,” MATERIALS TODAY: PROCEEDINGS, VOL. 5, NO. 1, PP. 12777-12783, 2018.

3.N. R. PAVITHRA, D. M. BHALERAO, AND D. K. VERMA, “BEEF QUALITY IDENTIFICATION USING COLOR ANALYSIS AND K-NEAREST NEIGHBOR CLASSIFICATION,” JOURNAL OF FOOD PROCESSING AND PRESERVATION, VOL. 44, NO. 11, E14800, 2020.

4.S. PANDEY, “APPLICATION OF IMAGE PROCESSING AND INDUSTRIAL ROBOT ARM FOR QUALITY ASSURANCE PROCESS OF PRODUCTION,” INTERNATIONAL JOURNAL OF EMERGING TRENDS IN ENGINEERING RESEARCH, VOL. 8, NO. 7, PP. 3605-3609, 2020.

5.MILICA BABICA , MOJTABA A. FARAHANIA , THORSTEN WUEST, IMAGE BASED QUALITY INSPECTION IN SMART MANUFACTURING SYSTEMS:

6.K. P. J. HEMACHANDRAN, “IMAGE PROCESSING IN AGRICULTURE,” INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES, VOL. 6, NO. 6, PP. 5234-5237, 2015.

THANK YOU !