Applying Blockchain to Energy Delivery Systems

DESIGN DOCUMENT

Team: sdmay20-12
Client: Grant Johnson
Adviser: Manimaran Govindarasu

Team Members:
Anthony Cosimo - *Test Engineer*
Jacob Dawson - *Project Manager*
Keegan Bloedel - *API Architect*
Katherine Ringgenberg - *UI Architect*
Steven Rein - *Blockchain Architect*
Dakota Moore - *Cybersecurity Manager*

sdmay20-12@iastate.edu
https://sdmay20-12.sd.ece.iastate.edu

Revised: 5 December 2019

Executive Summary

Development Standards & Practices Used

- Agile Software Development - two week sprints
- Test-driven Development
- Continuous Integration and Development
- Peer Reviews

Summary of Requirements

- API
    - API shall expose Smart Contract functions to Web Requests from the User Interface
    - API requests require authentication
    - API shall still be functional when the Blockchain network is down
- Blockchain Network
    - Blockchain Network shall have a minimum of five nodes
    - Blockchain Network shall have multiple orderer nodes
- Smart Contract Layer
    - Functions within the Smart Contract Layer shall be able to read, update, delete, and query data stored on the ledger
- User Interface
    - Allow user to display and query performance data stored in Blockchain network
    - Allow user to issue operator commands
- System-level
    - Each system shall be fault tolerant to other systems.
        - For example, if the Blockchain Network is down, the API shall return an appropriate error code to the UI, indicating that the Blockchain Network is down. The UI will then display a message notifying the user of the Blockchain Network being down.
    - Each component shall be deployed on Linux virtual machines made available through PowerCyber resources.

Applicable Courses from Iowa State University Curriculum

- COM S 309: Software Development Practices
- COM S 319: Construction of User Interfaces
- S E 329: Software Project Management
- S E 339: Software Architecture and Design

New Skills/Knowledge acquired that was not taught in courses

- Knowledge of HyperLedger Fabric and other HyperLedger technologies
- Knowledge of Blockchains concepts applicable to the problem domain
    - Orderers
    - Self-Signing Certificates
    - Node interactions
    - Raft Consensus
- Understanding and implementation of Docker tooling for containerized applications

Table of Contents

1	Introduction	6	
1.1	Acknowledgement	6	
1.2	Problem and Project Statement	6	
1.3	Operational Environment	7	
1.4	Functional Requirements	7	
1.5	Non-functional Requirements	8	
1.6	Intended Users and Uses	9	
1.7	Assumptions and Limitations	10	
1.8	Expected End Product and Deliverables	10	
2	Specifications and Analysis	11	
2.1	Proposed Design	11	
2.2	Design Analysis	14	
2.3	Development Process	15	
2.4	Design Plan	15	
3	Statement of Work	16	
3.1	Previous Work And Literature	16	
3.2	Technology Considerations	17	
3.3	Task Decomposition	18	
3.4	Possible Risks And Risk Management	19	
3.5	Project Proposed Milestones and Evaluation Criteria	21	
3.6	Project Tracking Procedures	22	
3.7	Expected Results and Validation	23	
4	Project Timeline, Estimated Resources, and Challenges	24	

4.1	Project Timeline	24	
4.2	Feasibility Assessment	25	
4.3	Personnel Effort Requirements	25	
4.4	Other Resource Requirements	27	
5	Testing and Implementation	27	
5.1	Process of Testing	27	
5.2	Interface Specifications	28	
5.3	Hardware and Software	28	
5.4	Functional Testing	28	
5.5	Non-Functional Testing	30	
5.6	Results	31	
6	Closing Material	31	
6.1	Conclusion	31	
6.2	References	31	
6.3	Appendices	31	

List of figures/tables/symbols/definitions

Figure 1: Use Case Diagram
Figure 2: Architecture Diagram
Figure 3: Level 0 Diagram
Figure 4: API Level 1 Diagram
Figure 5: Blockchain Level 1: Diagram
Figure 6: UI Level 1: Diagram
Figure 7: Gitlab Issues example
Table 1: Timeline of Proposed Work Schedules
Table 2: Personnel Efforts Requirements per Task

1. Introduction

1.1. Acknowledgement

Our team would like to thank and acknowledge PowerCyber Labs for allowing us to use their compute resources, Manimaran Govindarasu for advising our project, and Grant Johnson for being the client to our project.

1.2. Problem and Project Statement

1.2.1. Problem Statement

Energy Delivery Systems are deployed in an environment that is geographically distributed utilizing public internet infrastructure for communications. The integrity of measurements, commands, and authenticity of control devices performing communication are critical for trusted operations.

1.2.2. Proposed Solution

The purpose of this project is to develop a blockchain Energy Delivery System solution to solve the problem statement. The project is driven by the need described in the problem statement above, to create a secure network for a geographically diverse Energy Delivery System. A report on the security of Energy Delivery Systems from the National Research Council in 2012 stated that a powerful terrorist organization could cause a blackout for millions of people for many weeks (Terrorism and the Electric Power Delivery System, 2012). To help mitigate this risk, the use of blockchain technology would provide a secure network for managing Energy Delivery Systems by removing the dependency on a central service. Additionally, a permissioned blockchain system ensures a higher level of security when concerned with cyber attacks similar to a fifty-one percent attack. With this project we hope to deploy a permissioned blockchain system for interaction between devices on an Energy Delivery System and users managing those devices.

1.3. Operational Environment

The operating environment for this project is servers that will store the project and other data we may use, such as simulation data. Therefore, it should not be subject to any harsh weather conditions, and the servers should be properly maintained. These servers will be linux-based and deployed on a network that will allow for secure permissioned communication between all nodes.

1.4. Functional Requirements

1.4.1. API

1.4.1.1. The API shall expose the Smart Contract functions to Web Requests from the User Interface and/or Devices.

1.4.1.2. The API calls shall be authenticated to the Blockchain Node System.

1.4.1.3. Requests to the API shall require authentication.

1.4.1.4. When the Blockchain network is down the API shall return an appropriate error code indicating the loss of the Blockchain Network.

1.4.2. Blockchain Network

1.4.2.1. The Blockchain Network shall consist of at least five nodes for transaction consensus.

1.4.2.2. The Blockchain Network shall consist of multiple nodes to act as orderers.

1.4.2.3. The Blockchain Network shall use Raft consensus for the ordering service and the deterministic consensus algorithm.

1.4.3. Smart Contract Layer

1.4.3.1. The Smart Contracts shall implement the blockchain functions to read, update, delete, and query data stored on the ledger.

1.4.3.2. Users that are assigned to a channel are the only users allowed to utilize the Smart Contracts functions made available to that channel.

1.4.3.3. Each Smart Contract shall define more than one endorsing node for consensus.

1.4.4. User Interface

1.4.4.1. The User Interface shall be rendered in a modern web browser.

1.4.4.2. If a user has the authority to request specific measurements and/or metrics and does so via the User Interface, then the User Interface shall display those requested measurements and/or metrics.

1.4.4.3. If a user has the authority to issue specific operator commands, then User Interface shall provide the ability to issue those specific operator commands to the ledger.

1.4.4.4. Given the API is down the User Interface shall remain functional.

1.4.4.5. A user shall be able to view the results of metrics and measurements they had previously requested.

1.4.5. Operational Environment

1.4.5.1. The blockchain network shall run on a linux-based virtual machine provided by PowerCyber resources.

1.4.5.2. The API shall run on a linux-based virtual machine provided by PowerCyber resources.

1.4.5.3. The User Interface shall run on a linux-based virtual machine provided by PowerCyber resources.

1.4.5.3.1. The User Interface should be interacted with using a modern web browser, Chrome will be the supported browser.

1.5. Non-functional Requirements

1.5.1. API

1.5.1.1. The API shall have swagger documentation for each endpoint.

1.5.1.2. The API shall produce query responses in under 10 seconds.

1.5.1.3. The API shall have continuous integration and automated deployment.

1.5.2. Blockchain Network

1.5.2.1. Each node on the network shall be ran using Docker containers to allow for reliability when ran on different types of machines.

1.5.2.2. The Blockchain Network shall be deployed to PowerCyber resources by implementing a CI/CD pipeline.

1.5.2.3. The Blockchain Network should be able to run in a Linux-based system via the services provided by Amazon Web Services, however this is not a direct stipulation by the client, just a potential for additional complexity if needed.

1.5.3. Smart Contract Layer

1.5.3.1. The Smart Contract Layer shall have automated deployment and automated testing.

1.5.3.2. The Smart Contract Layer shall make updates to the Blockchain Network.

1.5.3.3. The Smart Contract Layer controllers and models shall contain documentation that describe their purpose and intended use.

1.5.3.4. The Smart Contract Layer shall have unit tests for all controllers.

1.5.4. User Interface

1.5.4.1. If the blockchain network is not available, the User Interface shall display loss of communication to the user.

1.5.5. Maintainability

1.5.5.1. The documentation made available through the project's wiki is descriptive enough for the client to understand how to modify the project properly when needed.

1.5.5.2. When the client wishes to modify the project, the project shall not be difficult to modify.

1.5.5.3. Continuous Integration and Continuous Deployment can run all tests and deploy to the necessary environments as needed by the client.

1.6. Intended Users and Uses

Our Energy Delivery System will have two main types of users: human users and devices. The use cases of these users will consist of the following:

1.6.1. Human users should be required to login into the system through a web user interface before interacting with the system.
1.6.2. Querying metrics and measurements from the blockchain system. This will be done by the human users.
1.6.3. Posting updated commands for devices to use. This will also be done by the human users, although there are potential use cases within the problem domain where the devices will give or share commands with each other.
1.6.4. Receive and execute commands provided by the user. This will be done by the devices.
1.6.5. Periodically post updated measurements. This will be done by the devices, although there is potential for the human users to make changes to the domain specific data, in cases of errors with metrics or measurements.

These use cases and users may have overlapping interactions with the system as mentioned, but generally the interactions will consist of the primary user in each use case, as mentioned above.

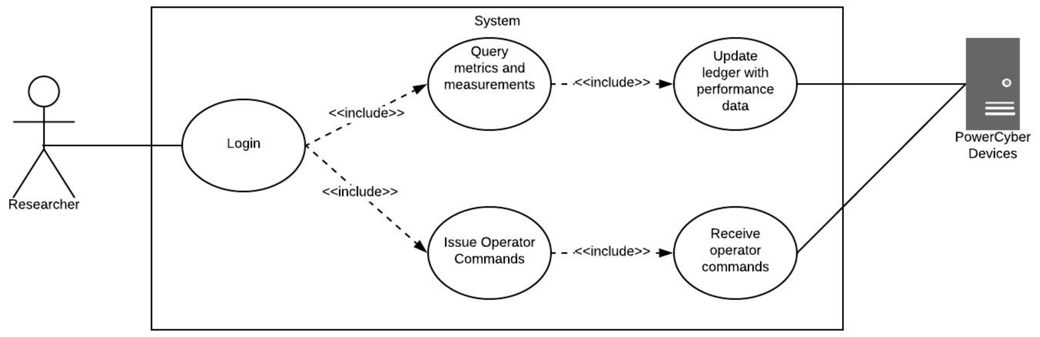


Figure 1: Use Case Diagram

1.7. Assumptions and Limitations

1.7.1. Assumptions

1.7.1.1. Server hardware and operating system environment will be made available through PowerCyber and these resources will be sufficient for the development.

1.7.1.2. We will be provided access to devices that use PowerCyber.

1.7.2. Limitations

1.7.2.1. There is no budget for the project, thus, we are constrained to using PowerCyber resources.

1.7.2.2. If the PowerCyber resources are found to be insufficient, the sponsor will either arrange for different resources or modify the scope to work with the existing resources.

1.7.2.3. Hyperledger Fabric must be used as the permission-based distributed ledger framework.

1.7.2.4. All Hyperledger Fabric related code must be written in JavaScript.

1.8. Expected End Product and Deliverables

1.8.1. A Fully Functional Blockchain Node System

There will be a Node System that is running within Virtual Machines within the PowerCyber network. The nodes in the Node System will help to endorse information updates to the ledger, provide immutability of the data within the ledger, and be fault tolerant to losses of nodes within the network. Communication with these nodes will be done by a multi-node ordering service. Unlike other blockchain systems, the orderer will allow our data to be deterministic instead of probabilistic. Additionally, the blockchain Node System will have smart contracts installed and instantiated, this will be for interaction between the nodes and the API system which will be utilized by the users for the provided use cases.

1.8.2. Authenticated Call from API to the Blockchain Node System

There will be an API that can utilize the Smart Contracts in order to create, read, update, delete, and query data available on the Node System. All calls to the Smart Contracts from the API shall be authenticated and verified by the allowed participants in each channel. The Smart contracts will be able to receive data from the nodes along with updating the data on the nodes.

1.8.3. Web-based User Interface

The web-based user interface will contain web pages that will allow authenticated users to query and view measurements, and post commands to devices.

1.8.4. Project Documentation

There will be a wiki that lives within the project repository that describes how to deploy updates to the Smart Contracts, Node System, API, and Web-based User Interface. This wiki will also have documentation regarding the development environment setup and required dependencies.

1.8.5. Continuous Integration and Deployment Infrastructure

There will be a CI/CD solution in place for the project. Continuous Integration will run on all merge requests and will run on the master branch weekly to ensure that the master branch remains clean. Continuous deployment will be available for the blockchain network, API, and web-based user interface.

2. Specifications and Analysis

2.1. Proposed Design

The software solution consists of the following components: a User Interface, an API, and a Blockchain Network. The API receives requests from the User Interface to view performance data and issue operator commands to PowerCyber devices. The API will authenticate those requests with the Membership Service Provider. After a request has been authenticated, the API will make a request to run a smart contract on the Blockchain Network that either queries performance measurements or updates operator commands for the PowerCyber devices.

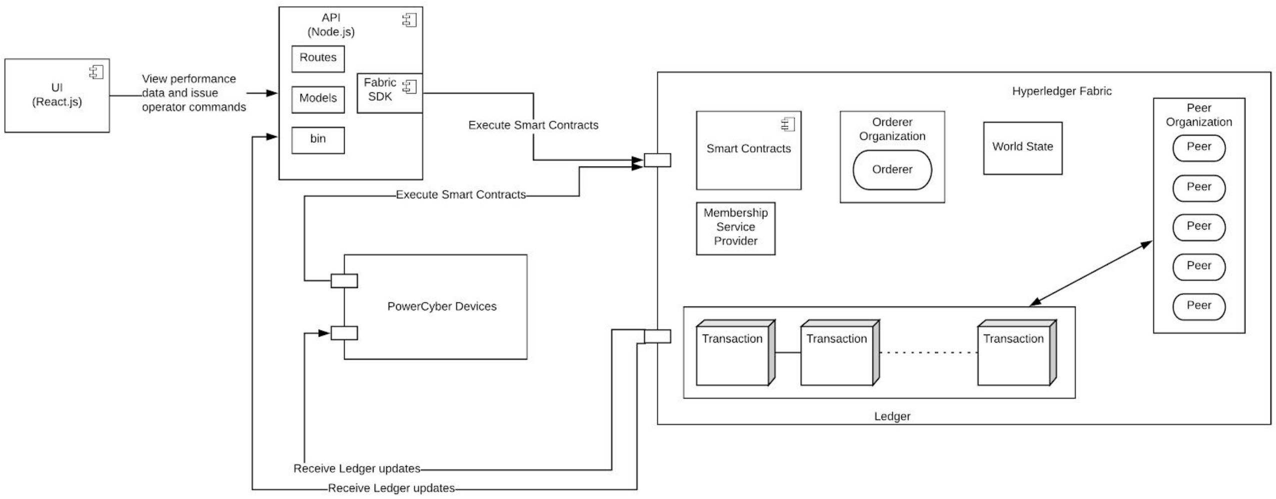


Figure 2: Architecture Diagram (above)

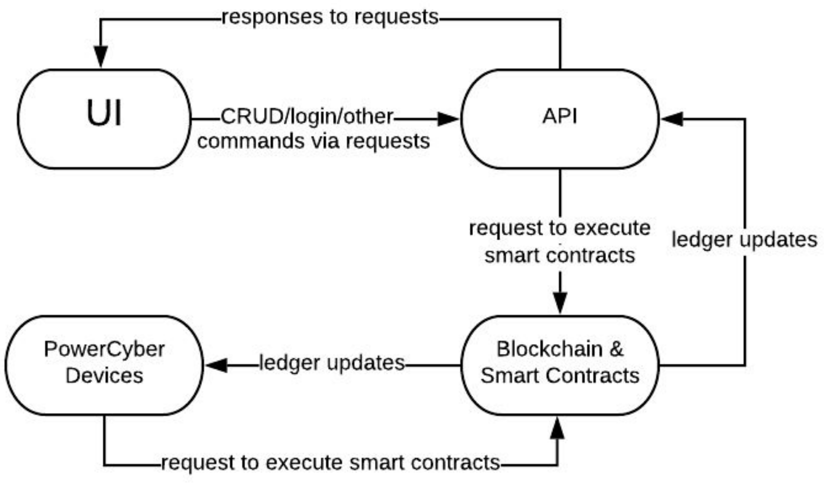

responses to requests
UI
CRUD/login/other commands via requests
API
request to execute smart contracts
ledger updates
PowerCyber Devices
ledger updates
Blockchain & Smart Contracts
request to execute smart contracts

Figure 3: Level 0 Diagram (above)

2.1.1. API

2.1.1.1. The API will run smart contracts when new performance data has been received by a PowerCyber Device.

2.1.1.2. The API will run smart contract when an authenticated user requests to view performance data.

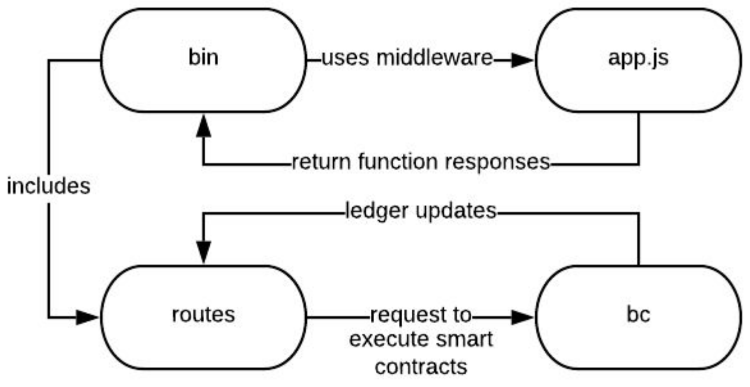


Figure 4: API Level 1 Diagram

2.1.2. Blockchain Network

2.1.2.1. The Blockchain Node System will report ledger updates to the API.

2.1.2.2. The Blockchain Node System will receive requests to run smart contracts from the API.

2.1.2.3. Given that the entity who has requested to run a Smart Contract is authorized for that requested data, the Blockchain Node System will run that Smart Contract in order to update the ledger.

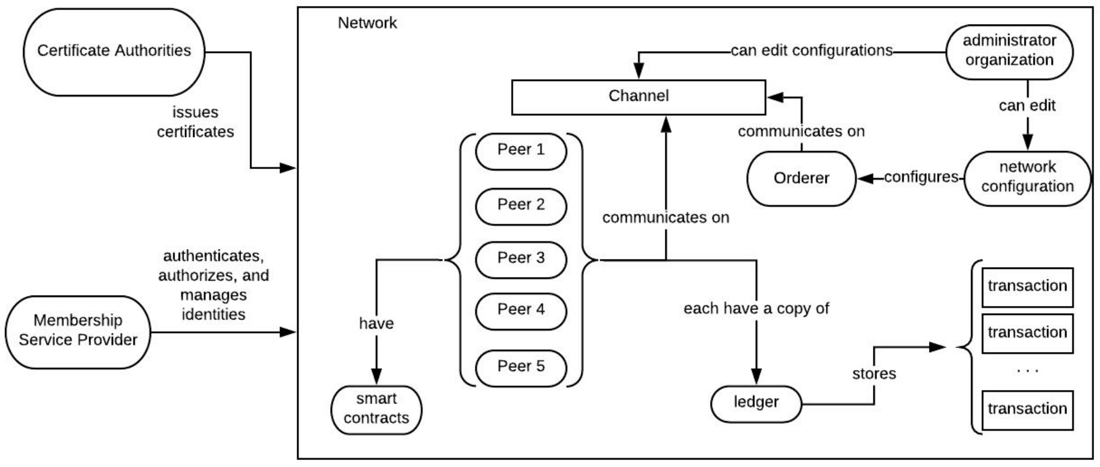


Figure 5: Blockchain Network Level 1 Diagram

2.1.3. Smart Contract Layer

2.1.3.1. Given that the entity who is requesting to query entries via a Smart Contract is authorized to do so, the Smart Contract Layer will be able to query entries in the Blockchain Network.

2.1.3.2. Given that the entity who is requesting to create an entry via a Smart Contract is authorized to do so, the Smart Contract Layer will be able to create an entry in the Blockchain Network.

2.1.3.3. Given that the entity who is requesting to update to an entry via a Smart Contract is authorized to do so, the Smart Contract Layer will be able to create an update to an entry in the Blockchain Network.

2.1.4. User Interface

2.1.4.1. The User Interface will allow a user, based on their authority, to request PowerCyber Device metrics.

2.1.4.2. The User Interface will allow a user, based on their authority, to issue operator commands on PowerCyber device(s).

2.1.4.3. The User Interface will allow a user to login.

2.1.4.4. Given that a user with required permissions to create a new account, the User Interface shall allow the user to create a new user.

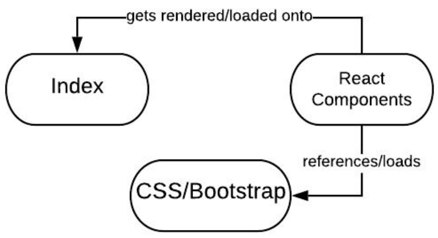


Figure 6: UI Level 1 Diagram

2.1.5. PowerCyber Devices

2.1.5.1. The PowerCyber Devices will receive and perform operator commands from the Blockchain Network.

2.1.5.2. The PowerCyber Devices will report performance data to the Blockchain Network.

2.2. Design Analysis

2.2.1. Infrastructure Analysis

2.2.1.1. Blockchain Network

We plan on using our Blockchain Network to handle the messaging between peers. Due to the importance of security, and the ability to remotely access data we believe that deploying Blockchain Network will be the best solution.

2.2.1.2. Smart Contracts
We plan to use HyperLedger Smart Contracts on our Blockchain nodes to build a permissioned blockchain. Smart contracts allow you to set Client, orderer, peer, and endorser roles to individual nodes. This increases the security of the blockchain. Due to the added features, and security of smart contracts, we believe it will be the best solution for the Blockchain Network.

2.2.2. Front-end Analysis

2.2.2.1. We decided to go with React.js for our front-end User Interface. Since React.js works well with our API, and well known, and powerful front-end development tool. Therefore, we believe React.js is the best way to go for designing the how the user will interact with the Blockchain.

2.3. Development Process

Our team will be taking an agile approach to the project. We will be using two-week sprints with a retro and demo at the end of each sprint as necessary. For all merge requests, there shall be tests to accompany all new work introduced into the master branch, therefore, we will be following a Test-Driven Development process for the project. All merge requests must pass all component and system-level tests, and be reviewed and approved by at least one other member on the project.

2.4. Design Plan

After gathering requirements and planning the development of solutions to the gathered requirements we will develop designs for each solution by utilizing data flow diagrams (such as Figures 3 - 6) and architecture diagrams (such as Figure 2) where appropriate.

The API will process requests from the UI and make requests to Smart Contracts using the Hyperledger Fabric Node SDK when necessary. All requests made to the API, except for requests for signing in, will be authenticated requests. If the Smart Contract Layer were to become unresponsive, the API will remain functional outside of the interactions with the Smart Contract Layer.

The Blockchain Network will consist of at least five nodes and multiple orderers to share messages between peers. The Blockchain Network will have a Raft Ordering Service for transaction ordering. The world state ledger will utilize CouchDB. Hyperledger Fabric cryptographic generation tools will be used to utilize self-signed certificates instead of an external certificate authority to authenticate interactions to and within the Blockchain Network. Measurement data will be stored on the Blockchain Network along with issued operator commands. The purpose of doing so is to provide and immutable transaction history of operator commands and reports of measurement data.

The Smart Contract Layer will send ledger updates to the API and to PowerCyber devices respectively. The updates will include operator commands and measurement updates reported by PowerCyber Devices. All updates will require endorsements from at least three nodes, given that there are five total nodes. The Smart Contract Layer will allow users and PowerCyber devices to invoke, through the API or directly, Smart Contract functions. These functions will be written in JavaScript and will provide the ability to read, write, query, and delete information stored in the Blockchain Network. The Smart Contract functions must installed on at least one Peer Node and instantiated on at least three other nodes for endorsement.

The UI will allow users to login, request metrics they are authorized to query, and issue operator commands that they are authorized to issue. If the API were to shutdown for any reason the UI will remain responsive outside of the interactions with the API.

3. Statement of Work

3.1. Previous Work and Literature

An article titled “Blockchain technology in the energy sector: A systematic review of challenges and opportunities” gives a lot of information on blockchains and how they can be used in energy solutions. One example this article gives is Brooklyn MicroGrid, which is a blockchain-based person-to-person energy trading platform.

The Brooklyn MicroGrid system completed a three month trial with the community. The article was written in February 2019. Upon further research into Brooklyn MicroGrid, it appears they are still up and running and have future goals of expanding and having fully automated transactions. This energy trading platform allows consumers to “sell their energy surplus directly to the neighbors by use of Ethereum-based smart contracts” (Andoni et al). This system’s ledger records “contact terms, transacting parties, volumes of energy injected and consumed by metering devices and crucially the chronological order of transactions” (Andoni et al). The article goes more in depth about blockchain potential and specifics involving the energy sector.

Our project will be looking at securing the communication network in an Energy Delivery System through the usage of blockchain. This differs from the Brooklyn

MicroGrid system because our system will be constrained by a necessity for high quality integrity within the data, whereas the Brooklyn MicroGrid system is concerned with the sale of an energy surplus.

3.2. Technology Considerations

HyperLedger Composer, a tool used for building a blockchain network and implementing smart contracts using HyperLedger Fabric, has recently been deprecated. To work around this, we looked into an alternative called Convector, supported by Hyperledger Labs. Convector provides similar functionality that Composer provides with some additional features such as an API Server Generator, configuration file generation, Smart Contract boilerplate code, etc. Also, convector allows the user to make configurations to a definitions JSON file for exposing Smart Contract functionality to an API. After further exploration of HyperLedger Convector, we determined that our use case should be fully implemented without this technology. By not using Convector, we will have more autonomy in our design and will be enabled to define the network, smart contracts, and API on our terms.

HyperLedger Fabric will be used for configuring and deploying our blockchain network. HyperLedger Fabric utilizes YAML configuration files for structuring and setting up the network. HyperLedger Fabric also uses Docker Composer for defining, creating, and deploying the containerized nodes to ensure all environment requirements are met. HyperLedger Fabric works using nodes that exist as the central communication for the network. These nodes ensure consistency is maintained with the state of the Ledger. HyperLedger Fabric has a key value database called CouchDB to deal with transaction logs and ledgers (NoSQL document store). CouchDB is the default database for HyperLedger Fabric, therefore the best choice when working with Fabric.

For development of our request API there are many technologies on the market to consider. With that in mind, there are three web frameworks the team has looked into. Django, written in Python, provides rapid development and a plethora of tools for development for teams. Flask, also written in Python, provides a simple and flexible developer experience. Comparing the two, Flask is most likely preferred if the focus is gaining experience and learning. Django would be preferred if the focus is on the final product and maintainability, being the older of the two. The final web framework we have considered is Node.js. The benefit of Node.js for our team is that we are already doing some programming in JavaScript through other components of the project. Therefore, the team has decided to move forward using Node.js for the API.

For the web user interface ReactJS will be used. The team decided upon this to give ourselves the opportunity to work with a framework that is growing within the developer community. Additionally, ReactJS is primarily used for building single page applications, which fits our expected solution for the web user interface.

3.3. Task Decomposition

When stating that an implementation of a feature will be happening, this suggests that the implementation will include the necessary source code, along with the implementation of any appropriate tests e.g. unit, integration, smoke.

3.3.1. Requirements Gathering

3.3.1.1. Gather Functional Requirements

3.3.1.2. Gather Non-functional Requirements

3.3.2. Gain Domain Knowledge

3.3.2.1. Meet with PowerCyber research team

3.3.2.2. Research Energy Delivery Systems

3.3.3. Implement a functional Blockchain Network

3.3.3.1. Configure Organizations

3.3.3.2. Configure Channels

3.3.3.3. Configure Orderers

3.3.3.4. Configure MSP

3.3.3.5. First Deployment

3.3.3.6. Deployment of Multiple Nodes

3.3.3.7. Deployment of Five Nodes Concurrently

3.3.4. Implement the Smart Contract Layer

3.3.4.1. CRUD Metrics Data

3.3.4.2. CRUD Operator Commands

3.3.4.3. First Deployment

3.3.5. Implement the API

3.3.5.1. Implement endpoints for logging into software suite

3.3.5.2. Implement endpoints for querying measurement and metrics data

3.3.5.3. Implement endpoints for querying operator commands history

3.3.5.4. Implement endpoints for creating operator commands

3.3.5.5. Integrate with Smart Contract Layer

3.3.6. Implement the UI

3.3.6.1. Implement routes for querying measurement and metrics

3.3.6.2. Implement routes for issuing operator commands to specific devices

3.3.6.3. Implement routes for querying operator commands on specific devices

3.3.6.4. Implement routes for account management

3.3.6.5. Secure UI of security vulnerabilities

3.3.7. Integrate with PowerCyber Devices

3.3.7.1. Implement PowerCyber devices reporting performance data to Blockchain Network

3.3.7.2. Implement PowerCyber devices receiving operator commands from Blockchain Network

3.3.7.3. Implement database and login criteria for previous and new users

3.3.8. System-level Testing

3.3.8.1. Implement end-to-end tests for measurements and metrics

3.3.8.2. Implement end-to-end tests for Operator Commands

3.3.8.3. Implement end-to-end tests for Authentications

3.4. Possible Risks and Risk Management

Title: Blockchain could possibly be the wrong solution for the problem we intend to solve.

Risk: Avoid

Information: The time to complete blockchain consensus may take too long for the desired use cases. Also, the resources required to deploy and maintain a blockchain may not be valuable enough for the desired use cases.

Mitigation Action: We will work with the PowerCyber team to determine appropriate use cases (measurements and commands) that would benefit from using a Blockchain Network.

Title: Team's lack of knowledge on the domain could lead to easy to detect flaws being introduced into the project.

Risk: Mitigate

Information: The architecture of the project could be at risk due to the lack of knowledge on the domain. Classes, data-flow, and test assertions could be invalid resulting in a time consuming refactor.

Mitigation Action: Gain domain knowledge by visiting PowerCyber, interviewing domain experts, and asking questions to the client and advisor when necessary. Due to the team's frequent two week sprints and and high client interaction, increased communication and quick feedback can be easily achieved.

Title: UI could possibly present security flaws that could allow malicious activity against our project.

Risk: Mitigate

Information: The security of the project is at risk when any part of it is at risk. If a malicious user could access the UI and then use a SQL injection or XSS attack, they could gain access to protected data or inject malicious code into the project.

Mitigation Action: Secure the UI by implementing checks for text input and implementing other security features to prevent any and all kind of possibly cyber attacks on the UI.

Title: A User's login could be compromised

Risk: Mitigate

Information: There are several ways that a malicious attacker could get ahold of another user's login information and use it to gain access to the system. This could give the attacker access to private information and possibly controls to the blockchain.

Mitigation Action: A log will be made and updated with every action from a user. This log will then be analyzed to make sure that there are no abnormalities that could show unauthorized access. Two-factor authentication and HTTPS will also be implemented to help secure login information.

Title: Integration of system components could fail

Risk: Mitigate

Information: The integration between the Smart Contract Layer and the API, the PowerCyber Devices and the Blockchain Network, etc. is at risk due to the team's inexperience with Smart Contracts and PowerCyber Devices.

Mitigation Action: Utilize and maintain Swagger documentation and automate integration and acceptance tests.

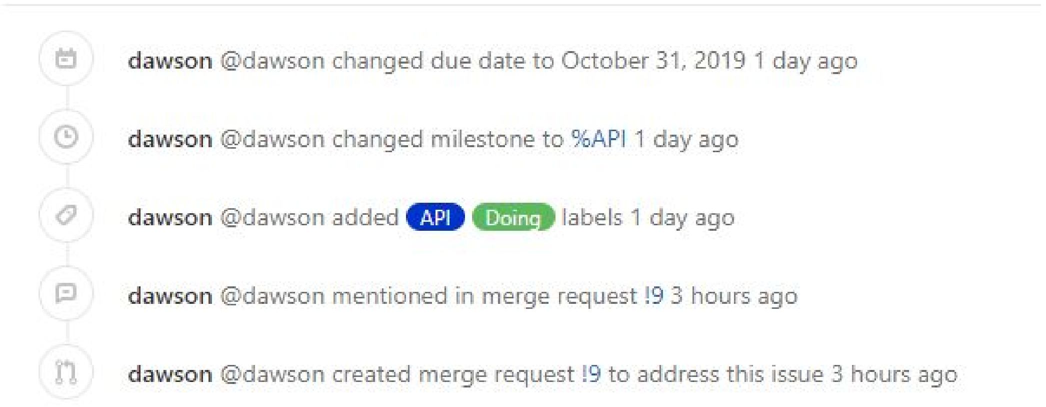
3.5. Project Proposed Milestones and Evaluation Criteria

- Successful communication between five nodes in the blockchain network occurs.
    - Occurs when five nodes can successfully communicate.
    - Occurs when communication and consensus is done using Raft.
- Smart contracts within nodes can carry out a task.
    - Occurs when smart contracts are successfully deployed to blockchain nodes.
    - Occurs when the task that the smart contract intended to carry out happens successfully.
- Successful communication between blockchain nodes in regards to problem domain.
    - Occurs when blockchain nodes are successfully communicating using definitions of the data within the problem domain.
- API provides reasonable responses queried by the user.
    - Occurs when the API functionality exists for all defined smart contracts and responses are given successfully in a reasonable time interval.
- User interface makes a successful API call.
    - Occurs when the API call is triggered by the user interface successfully carries out its function.
- User interface allows and verifies user login data

    - Occurs when a user can securely log in using their personal credentials.
    - Occurs when the UI will verify that a command input is from a user with the correct permissions.
- User interface is secure.
    - Occurs when the UI uses HTTPS, is void of known vulnerabilities for XSS or SQL injection attacks, and allows users to login securely.
- User interface provides accurate and visually pleasing data.
    - Occurs when the client is satisfied with the readability of displayed metrics and the user interface, overall.

3.6. Project Tracking Procedure

The team will use a Gantt Chart to track progress that can then be shared with stakeholders. Tracking through this Gantt chart will be updated at the end of every sprint, and will primarily be for the visibility of those outside of the team. We plan to use the Gantt chart for a high-level view on project tasks, relationships between tasks, and milestones. Then we will be using Gitlab Issues to break up tasks on the Gantt chart into smaller tasks. We have chosen to use Gitlab Issues due to its ability to produce burndown charts and allows us to use custom formats for issue tracking. We will use these issues to track individual and team progress throughout the project. These issues include a title, description on what will be worked on, due dates, labels, and assigned members. When an issue is completed, it is marked as so and archived, leaving a detailed list of what was worked on for the project. Looking at the example below, we can see that Gitlab provides the ability to not only explain what the specific task will implement, but also displays a record of changes to the issue.



dawson @dawson changed due date to October 31, 2019 1 day ago

dawson @dawson changed milestone to %API 1 day ago

dawson @dawson added API Doing labels 1 day ago

dawson @dawson mentioned in merge request !9 3 hours ago

dawson @dawson created merge request !9 to address this issue 3 hours ago

Figure 7: Gitlab Issues example

3.7. Expected Results and Validation

Our desired outcome is to produce and implement a suite of software that utilizes a secure Blockchain Network and can integrate with devices used at PowerCyber. This software suite shall add integrity to the existing system at PowerCyber.

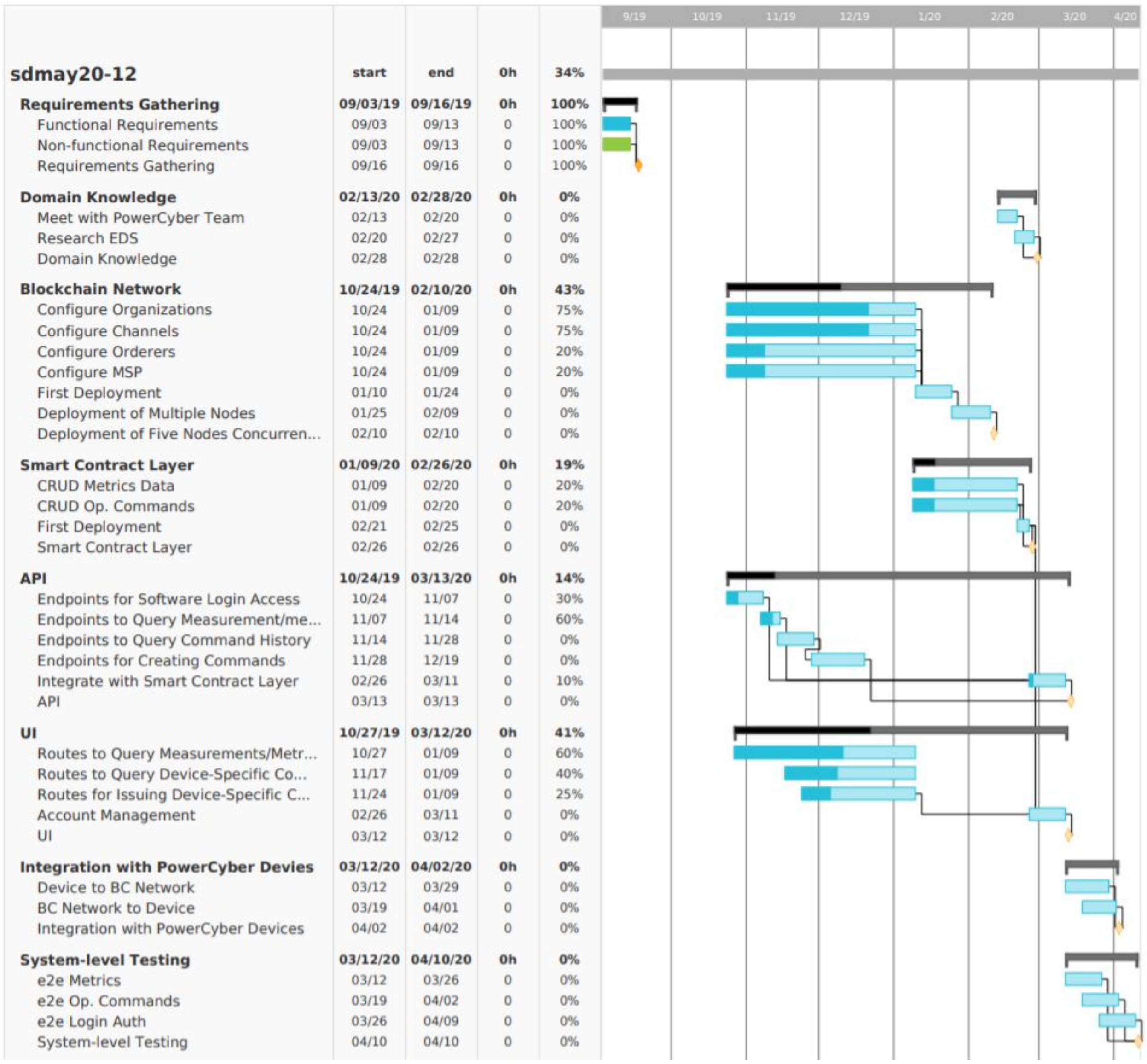
To confirm that our solution works at a higher level the following will need to be found true by the end of the project.

- The software suite can receive performance data from PowerCyber devices
- The software suite can store performance data in it's Blockchain Network
- Users of the software suite can query performance data from the software suite's UI
- Users of the software suite can issue operator commands to PowerCyber devices from the software suite's UI
- Operator commands issued through the software suite's UI will be entered into the Blockchain Network and distributed to the PowerCyber devices
- PowerCyber devices can receive and execute commands issued to them via the Blockchain Network
- Removing Blockchain peer nodes does not interrupt validation and committing to the ledger
- Removing Blockchain orderer nodes does not interrupt validation and committing to the ledger
- Removing communications to the Blockchain Network or the API results in a displayed error and continued responsiveness of the UI

4. Project Timeline, Estimated Resources, and Challenges

4.1. Project Timeline

For the majority of section 4, when stating that an implementation of a feature will be happening, this suggests that the implementation will include the necessary source code, along with the implementation of any appropriate tests e.g. unit, integration, smoke.



sdmay20-12	start	end	0h	34%
Requirements Gathering	**09/03/19**	**09/16/19**	**0h**	**100%**
Functional Requirements	09/03	09/13	0	100%
Non-functional Requirements	09/03	09/13	0	100%
Requirements Gathering	09/16	09/16	0	100%
Domain Knowledge	**02/13/20**	**02/28/20**	**0h**	**0%**
Meet with PowerCyber Team	02/13	02/20	0	0%
Research EDS	02/20	02/27	0	0%
Domain Knowledge	02/28	02/28	0	0%
Blockchain Network	**10/24/19**	**02/10/20**	**0h**	**43%**
Configure Organizations	10/24	01/09	0	75%
Configure Channels	10/24	01/09	0	75%
Configure Orderers	10/24	01/09	0	20%
Configure MSP	10/24	01/09	0	20%
First Deployment	01/10	01/24	0	0%
Deployment of Multiple Nodes	01/25	02/09	0	0%
Deployment of Five Nodes Concurren...	02/10	02/10	0	0%
Smart Contract Layer	**01/09/20**	**02/26/20**	**0h**	**19%**
CRUD Metrics Data	01/09	02/20	0	20%
CRUD Op. Commands	01/09	02/20	0	20%
First Deployment	02/21	02/25	0	0%
Smart Contract Layer	02/26	02/26	0	0%
API	**10/24/19**	**03/13/20**	**0h**	**14%**
Endpoints for Software Login Access	10/24	11/07	0	30%
Endpoints to Query Measurement/me...	11/07	11/14	0	60%
Endpoints to Query Command History	11/14	11/28	0	0%
Endpoints for Creating Commands	11/28	12/19	0	0%
Integrate with Smart Contract Layer	02/26	03/11	0	10%
API	03/13	03/13	0	0%
UI	**10/27/19**	**03/12/20**	**0h**	**41%**
Routes to Query Measurements/Metr...	10/27	01/09	0	60%
Routes to Query Device-Specific Co...	11/17	01/09	0	40%
Routes for Issuing Device-Specific C...	11/24	01/09	0	25%
Account Management	02/26	03/11	0	0%
UI	03/12	03/12	0	0%
Integration with PowerCyber Devies	**03/12/20**	**04/02/20**	**0h**	**0%**
Device to BC Network	03/12	03/29	0	0%
BC Network to Device	03/19	04/01	0	0%
Integration with PowerCyber Devices	04/02	04/02	0	0%
System-level Testing	**03/12/20**	**04/10/20**	**0h**	**0%**
e2e Metrics	03/12	03/26	0	0%
e2e Op. Commands	03/19	04/02	0	0%
e2e Login Auth	03/26	04/09	0	0%
System-level Testing	04/10	04/10	0	0%

Implementing a Blockchain Network will take the longest compared to other groups of tasks due to the team's inexperience with the technology and gaining access to the resources for hosting the Blockchain Network. So far, from what we have noticed during our experiments with Hyperledger Fabric, implementing the Smart Contract Layer should not be too difficult.

Implementing the UI shall be close to straight-forward. Some members of the team have previous experience using React.js and feel comfortable implementing necessary features.

The time to complete integration with PowerCyber devices is currently unknown. The estimation in the Gantt chart is our best guess as of right now. We will have a more accurate estimation for this task once we have visited with the PowerCyber team.

4.2. Feasibility Assessment

Depending on the use cases we learn about when we visit with the PowerCyber team, the correct use case will be chosen that is best suited for a Blockchain implementation given the latency and data criticality constraints. Some foreseen challenges that we see as of now is the time for the nodes to come to a consensus when a Smart Contract is executed. Depending on the requirements of the devices at PowerCyber, the time to come to a consensus may be too slow. For example, a device may need to update the Blockchain Network every two seconds, we can foresee our nodes taking longer than two seconds to come to a consensus. Thus, rendering our software suite incorrect for that use case.

If there are some use cases that we can handle due to the bottleneck of coming to a consensus across all nodes, then our project will obsolete provide a simple and easy way for researchers to view measurements and metrics and to issue operator commands to PowerCyber devices. While doing so, our software suite will be providing immutable performance data and operator commands transaction history.

4.3. Personnel Effort Requirements

Tasks and Milestones	Estimated Time to Complete
3.3.1. Requirements Gathering	**10 Days**
3.3.1.1. Gather Functional Requirements	5 Days
3.3.1.2. Gather Non-functional Requirements	5 Days
3.3.2. Domain Knowledge	**16 Days**
3.3.2.1. Meet with PowerCyber research team	8 Days

3.3.2.2. Research Energy Delivery Systems	8 Days
3.3.3. Creation/Deployment of the Blockchain Network	**60 Days**
3.3.3.1. Configure Organizations	7 Days
3.3.3.2. Configure Channels	7 Days
3.3.3.3. Configure Orderers	7 Days
3.3.3.4. Configure MSP	7 Days
3.3.3.5. First Deployment	15 Days
3.3.3.6. Deployment of Multiple Nodes	22 Days
3.3.3.7. Deployment of Five Nodes Concurrently	60 Days
3.3.4. Creation/Implementation of the Smart Contract Layer	**42 Days**
3.3.4.1. CRUD Metrics Data	14 Days
3.3.4.2. CRUD Operator Commands	14 Days
3.3.4.3. First Deployment	14 Days
3.3.5. Creation/Deployment/Implementation of the API	**80 Days**
3.3.5.1. Implement endpoints for logging into software suite	14 Days
3.3.5.3. Implement endpoints for querying measurement and metrics data	7 Days
3.3.5.3. Implement endpoints for querying operator commands history	14 Days
3.3.5.4. Implement endpoints for creating operator commands	21 Days
3.3.5.5. Integrate with Smart Contract Layer	14 Days
3.3.6. Creation/Deployment/Implementation of the UI	**61 Days**
3.3.6.1. Implement routes for querying measurement and metrics	21 Days
3.3.6.2. Implement routes for querying operator commands on specific devices	8 Days
3.3.6.3. Implement routes for issuing operator commands to specific devices	18 Days
3.3.6.4. Implement account management	14 Days
3.3.7. Integration with PowerCyber Devices	**30 Days**
3.3.7.1. Device to Blockchain Network	15 Days
3.3.7.2. Blockchain Network to Device	15 Days
3.3.8. System-level Tests	**42 Days**

Implement end-to-end tests for measurements and metrics	14 Days
Implement end-to-end tests for Operator Commands	14 Days
Implement end-to-end tests for Authentications	14 Days
Total Estimated Time to Complete the Project:	**341 Days**

4.4. Other Resource Requirements

For our project, a resource we will need access to is the ISU ECpE PowerCyber facilities. We will additionally be using Virtual Machines (VMs) as an additional resource that is required to be able to test our blockchain network before we implement it with PowerCyber facilities.

5. Testing and Implementation

5.1. Process of testing

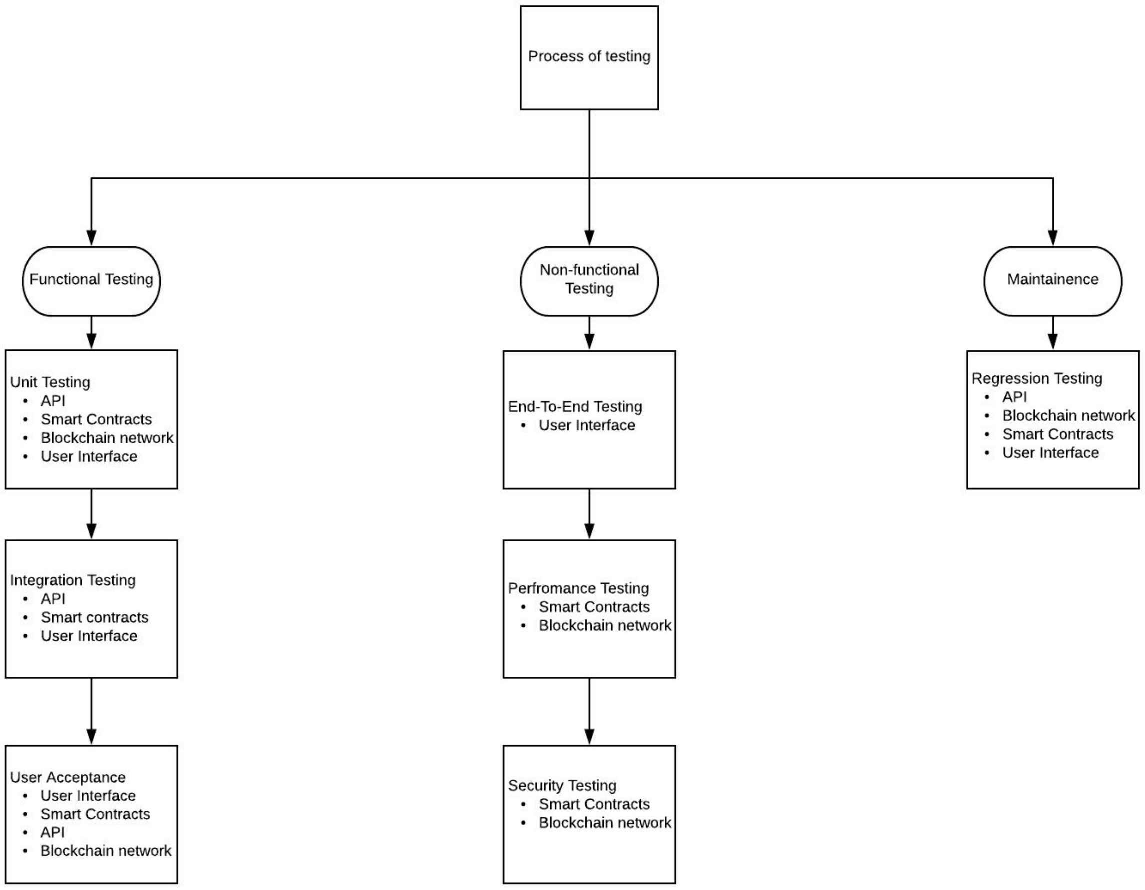


Figure 8: Process of testing

Figure 8 displays an overview of the process of testing. We will be following each flow as we progress forward in the development of our project. For each flow, we will implement the testing in the first block before moving on to the next and so on. This way we are able to break down our testing into concise parts to know what needs to be accomplished for our testing.

5.2. Interface Specifications

All components within the software suite will have to agree on consistent models to represent performance data, types of devices to interact with, and operator commands. This is yet to be determined until we visit the PowerCyber team.

5.3. Hardware and Software

5.3.1. API

5.3.1.1. Unit tests and integration tests for the API will use the Jest Javascript testing framework. With this framework we will write automated tests to make HTTP requests to our endpoints and unit test classes and functions.

5.3.2. Blockchain Network

5.3.2.1. All new chaincode will be added to a chaincode subdirectory and will be tested when relaunching the Blockchain Network.

5.3.3. Smart Contract Layer

5.3.3.1. We will be using the Jest Javascript testing framework for testing smart contracts.

5.3.4. User Interface

5.3.4.1. For writing unit tests, acceptance tests, and end-to-end tests, the Jest Javascript test runner will be used along with the React Testing Library. The Jest test runner will allow us to access the DOM via jsdom for testing React components and to use mocks. The React testing library will allow us to test React components without relying on their implementation details. For end-to-end tests we will remove the use of mocks and have all requests made to their respective endpoints.

5.3.5. PowerCyber Devices

5.3.5.1. Assuming we will not have access to PowerCyber devices often, we will have to examine some example performance data from the devices we will be working with and create a simulation data collection to use for integration tests between the Blockchain Network and PowerCyber devices.

5.4. Functional Testing

5.4.1. Unit Testing

All individual testable tasks will have their testing consisting of one or more tests for each component of the task. Each task that involves configuring or creating a part of the blockchain, smart contracts, API and UI will go through individual unit tests once they are complete as to confirm that the task is fully accomplished and that component can start to be safely integrated. The creation and integration tasks for the blockchain are tested to ensure the organizations, channels, orderers and MSP are all configured correctly. This will be done by testing the contents of each one individually on the running blockchain network. Smart Contracts will be tested to ensure that the metrics data and the operator commands are tracked and updated properly throughout all contracts, as well as testing to make sure that the deployed smart contracts are interacting properly with the blockchain network. Each one of the API's endpoints will be tested to ensure that you can use it to query or create the specified data, and the integration with smart contracts will be tested by configuring a smart contract through the API and checking it's contents. The UI will have unit tests to ensure that you're able to navigate to and use each component of it.

5.4.2. Integration Testing

Integration testing of the blockchain network will consist of deploying the nodes for the network and testing that the blockchain network is running and it's able to accurately execute the smart contracts as well as update all nodes on the network with a new contract. The integration of the blockchain with the API will be tested by running commands through the API to query, log in, and create commands/data that will be tested against known data. The integration of the API with the smart contracts will be tested by trying to execute a contract through the API and checking that it was properly executed. The Integration between the API and UI will be tested by using the UI to execute commands in the API and comparing what returns with known data. The integration with PowerCyber devices will be tested to make sure that the device is able to connect to the blockchain to be queried or report data, as well as tests to ensure it's accurately and securely reporting the information by attempting to intercept and/or alter the data between the device and the network.

5.4.3. System (end-to-end) Testing

The system testing will consist of testing the entire software from end-to-end. A test will be done to ensure that a user can access the UI, query some data and/or execute some command by having the UI use the API to execute said query/command on the blockchain, and then comparing the result with what should've been accomplished. A test will also be done to ensure that a device on PowerCyber will be able to connect to the blockchain network and continuously update data on it by using the API to constantly update the data stored on the nodes. A last test will be done to ensure that an unauthorized user isn't able to access the UI and ensuring no one is able to intercept and understand data being transferred into and out of the network.

5.5. Non-functional Testing

5.5.1. API

There will be unit tests for query response times appropriate for each endpoint. For example, the query response time making a GET request to "/api/metrics/" will be much greater than the query response time for "/api/metrics/1". The expected query response time will be different for each endpoint.

5.5.2. Blockchain Network

Testing the blockchain network can be a difficult task since most of what makes up the network is just configuration files. To do some minor integration testing with the blockchain network, we can use a HyperLedger Fabric CLI (Command Line Interface) docker image. After the blockchain has been started and the chaincode has been installed and instantiated, we can use the CLI to make some basic queries to test data in the chaincode, to ensure that a connection can be established with the network.

5.5.3. Smart Contract Layer

There will be integration tests for the Smart Contract layer to assert that ledger updates do occur when smart contracts are executed.

There will be an eslint rule to ensure that there is JSDoc for all controllers, models, functions, classes, etc.

5.5.4. User Interface

There will be automated acceptance tests in place to simulate loss of communication between the User Interface and the API, and loss of communication between the API and Blockchain Network.

Unit tests for making unauthenticated requests via the API and to the Smart Contract layer will be in place to ensure that all requests must be authenticated. There will be performance tests for the Smart Contract Layer by using simulated data from the PowerCyber devices to profile the time it takes for all nodes to come to a consensus on Smart Contract execution.

5.6. Results

During our research phase, we learned that one of the software’s we wanted to use for our project became depreciated and this led to us having to research further into possible replacements. This led to us using Convector, which we learned is supported by Hyperledger Labs. Again, after further research into Convector, we learned that it did not provide the flexibility when defining our blockchain network that would be necessary for the project.

6. Closing Material

6.1. Conclusion

So far we have gathered the requirements from the client and have created a simple blockchain. We also have VM now provided by PowerCyber to setup a more advanced blockchain, as well as have a single environment for everyone to work on. Our plan is to use the PowerCyber VMs to create our network, and to use either proved dummy data or data provided by PowerCyber to test our blockchain.

6.2. References

M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. Mccallum, and A. Peacock, “Blockchain technology in the energy sector: A systematic review of challenges and opportunities,” Renewable and Sustainable Energy Reviews, vol. 100, pp. 143–174, Feb. 2019.

Terrorism and the Electric Power Delivery System. (2012). National Academies Press, p.1.

6.3. Appendices

CouchDB:
http://docs.couchdb.org/en/stable/

Express:

https://expressjs.com/en/4x/api.html

HyperLedger Fabric:

https://hyperledger-fabric.readthedocs.io/en/release-1.4/

HyperLedger Convector:

https://docs.covalentx.com/article/71-getting-started

Moesif Orign & CORS Changer:

https://chrome.google.com/webstore/detail/moesif-orign-cors-changer/digfbfaphojjndkpccljibejjbppifbc

PowerCyber Labs:

http://powercybersec.ece.iastate.edu/powercyber/welcome.php

Raft:

https://raft.github.io/

React.JS:

https://reactjs.org/docs/getting-started.html