Development Team Project: Design Document

# Problem Statement

The International space station represents a multination cooperation and endeavours to advance science and perform scientific experiments in an environment completely different from that of the earth (Hitt, 2015). The stations’ laboratories and living quarters require state of the art constant monitoring for life support, fire detection and equipment malfunction.

The underlying systems mentioned above require ongoing development to ensure proper functioning and reliability. The proposed system design addresses various elements of these critical systems to upgrade the existing infrastructure and provide a safe and secure living environment for the crew.

# Assumptions

* The deployment of the new updated system will be done concurrently with the present system remaining online. As the life support systems must remain online for crew safety.
* NASA personnel will perform all upgrades on the International Space Station. As contractor personnel are not trained for space travel.
* The upgrade will be performed within a specific time frame determined by the U.S. national information assurance policy.
* The underlying contractor will perform all training on the new system as the contractor is the main developer of the new system.
* All work will be performed in the Johnson Space Center in Huston, Texas. NASA’s primary location for mission control.
* All upgrades to the system will be tested in a simulated environment at Goddard Space Flight Center as Goddard Space Flight Center has working replicas of ISS onboard systems.
* All sensors manufacturing data will be leveraged from the existing infrastructure that exists on the International space station. The contractor is not responsible for the upgrading of sensors.

# System Design Proposals

We are proposing two types of system designs monolithic (see Appendix A, Fig 1) and microservice (see Appendix A, Fig 2) architectures. Both architectures contain an inherently similar application logic; however, the microservice architecture distributes the workload over two servers.

The following subsystem describes the application logic for both the monolith and microservice architectures.

### Engineering Data

The engineering data container is responsible for data duplication at ground control. The container periodically sends API requests to the sensor containers and persists the returned data to a database.

### Application Logic

Each component within the microservice architecture is made up of a topic, container and database container whereas the monolith architecture is made up of a topic, application logic and database. Both systems have a shared application logic regarding the functionalities within each component (See Appendix C, Fig 1).

#### Fire Monitoring system

The fire monitoring container reads data received at the smoke topic (see Appendix B, Fig 2) and persists the data to the database. If an abnormal reading is received, the smoke alarm is activated.

#### Electrical Monitoring

The electrical container reads data received at the electrical topic (see Appendix B, Fig 2) and persists the data to the database. If an abnormal reading is received, the electrical alarm is activated.

#### Temperature

The temperature container reads data received at the temperature topic (see Appendix B, Fig 1) and persists the data to the database. If an abnormal reading is received, the temperature alarm is activated.

#### Oxygen

The oxygen container reads data received at the oxygen topic (see Appendix B, Fig 1) and persists the data to the database. If an abnormal reading is received, the oxygen alarm is activated.

#### Identity Provider

Before accessing the dashboard, administrators log in via the identity provider microservice. Once the login is successful, the administrator is taken to the enhanced dashboard view, displaying the various sensors and alarms statuses. Administrators will have additional privileges which provide a variety of in-depth controls of the containers and subsystems.

#### Dashboard

The dashboard container receives sensor data in real-time and displays this data as a series of live graphs and alerts. Additionally, the dashboard provides a login button that allows administrators to log in and view additional features and controls within the dashboard and create user accounts within the system (see Appendix E, Fig 1).

# Technical Challenges

### Real-Time Sensor Data Storage

Receiving and storing data in real-time while persisting in a database is a complex undertaking. Within the monolithic and microservices architectures, we use Apache Kafka, an “open-source distributed event streaming platform”, to create data pipelines ensuring high-quality data transmission between sensors in the system (Apache, 2021). Furthermore, we ensure that the data stored within Kafka topics are persisted in a database to provide further redundancies to the storage provided by Kafka.

### Limited Storage Space

The size of the deployed software is limited due to computer storage space.

# System Security

### Secure Software Development Practices

#### Enforce Access Controls

* Role-based authentication will be enforced on all API endpoints as well as the dashboard and administrator pages.
* Referring to OWASP C7 subsection six, the system will generate/update a log on all API endpoint calls when authentication is required.
* Access control is managed by the identity provider container/application logic.

(OWASP, 2021d)

#### Password Hashing Algorithms

* Within the identity provider container/Application Logic we will make use of the Argon2 library to securely hash password whilst applying salting to each hash.

(OWASP, 2021b)

# Patterns and Approaches

## Design Patterns

### Flask Blueprints

We will be making use of Flask Blueprints as a Flask design pattern to better structure the individual components within the system.

## Architectural Patterns

### Model View Controller

The dashboard within the proposed system will follow the MVC pattern to allow for greater modularity and customizability in the codebase (see Appendix D, Fig 1). The model represents the tables in the database, while the view represents the application front end, and the controller component contains all of the application logic (codeacademy, 2021).

# System Requirements

## Functional Requirements

* Real-time sensor data persistence to a MySQL database.
* Administrators can manipulate the system in various ways, such as silencing alarms, resetting containers and sensors.
* Engineering data will persist historical data for further offline analysis.
* The software will be compatible with existing ISS computing systems.

## Non-Functional Requirements

* The system will run on a proprietary Linux variant on the space station which ensures greater reliability and faster computation.
* The system should run uninterrupted 100% of the time, in the event of system failure backup system deployed.

## Hardware

### Ground Station

#### Server

HPE ProLiant DL380 Gen10 Server

2 x Intel Xeon Scalable 5218

128GB (4 x 32GB) Mem

4 x 2.4TB HDD + 2 x 480GB SSD + 2 x 240GB M.2

#### Storage

##### ASA AFF A800

* 6609.6 TB
* iSCSI

### International Space Station

* The pre-existing systems will provide hardware for the proposed system.
* Apache Kafka will require a minimum of 300GB solid-state storage for short term sensor data.
* A minimum of 800GB of storage will be required to persist historical sensor data.

# Tools and Libraries

## Tools

* Kafka
* MySQL

## Libraries

* Flask
* Flask JWT
* kafka-python
* Bleach
* SQLAlchemy
* Chart.JS
* Argon2-cffi
* Flask Argon2

## Code Linting

* Pylint
* Bandit

# References

codeacademy (2021) *MVC: Model, View, Controller | Codecademy*. Available at:<https://www.codecademy.com/articles/mvc> [Accessed: 13 June 2021].

Apache (2021) *Apache Kafka*. Available at: <https://kafka.apache.org/>.

Hitt, D. (2015) *'What Is the International Space Station?'*. Available at: <https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-the-iss-58.html>

OWASP (2021a) *C3: Secure Database Access*. Available at: <https://owasp.org/www-project-proactive-controls/v3/en/c3-secure-database>.

OWASP (2021b) *C5: Validate All Inputs*. Available at: <https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs>.

OWASP (2021c) *C7: Enforce Access Controls*. Available at: <https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls>.

OWASP (2021d) *Password Storage - OWASP Cheat Sheet Series*. Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html>

# 

# Appendices

## Appendix A

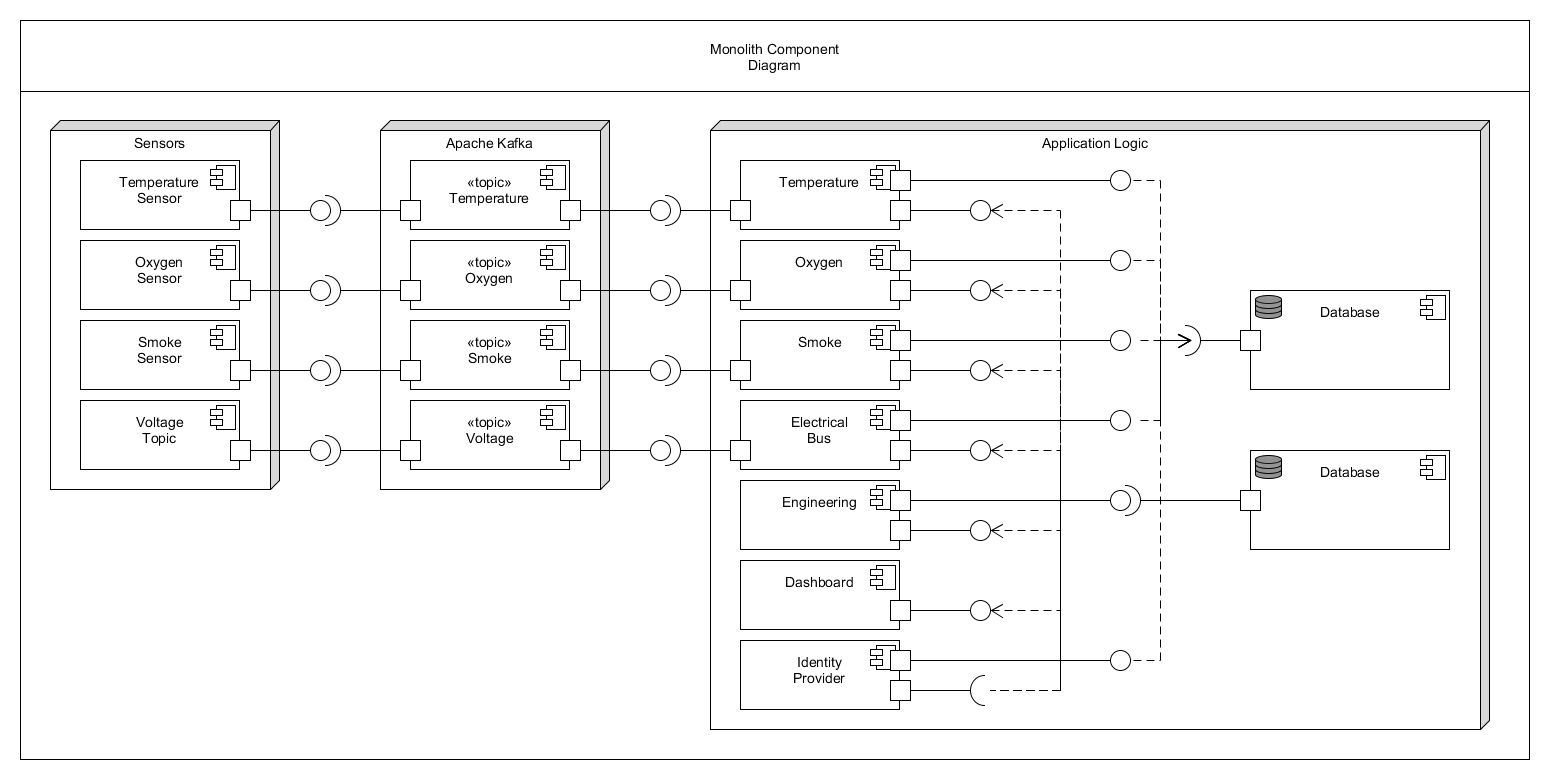


Figure 1

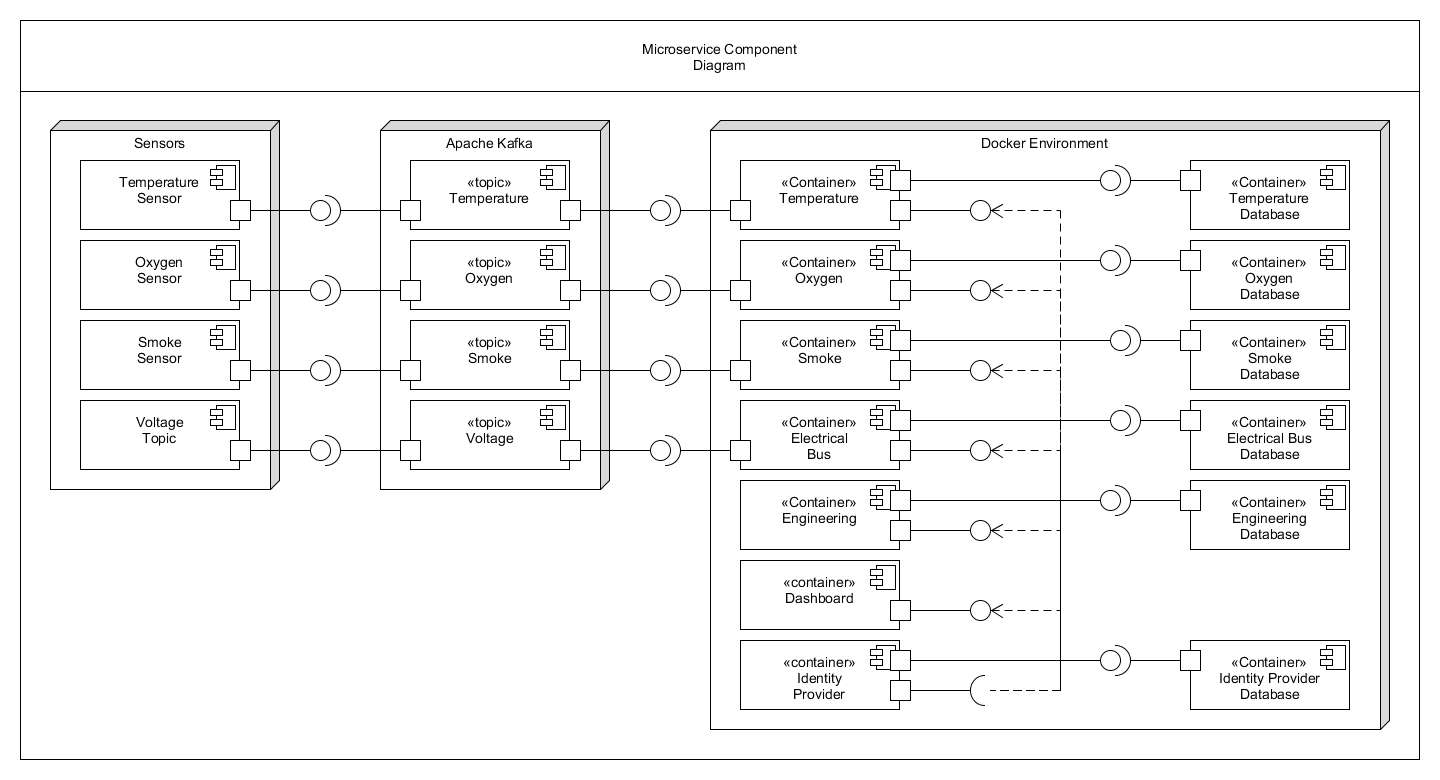


Figure 2

## Appendix B

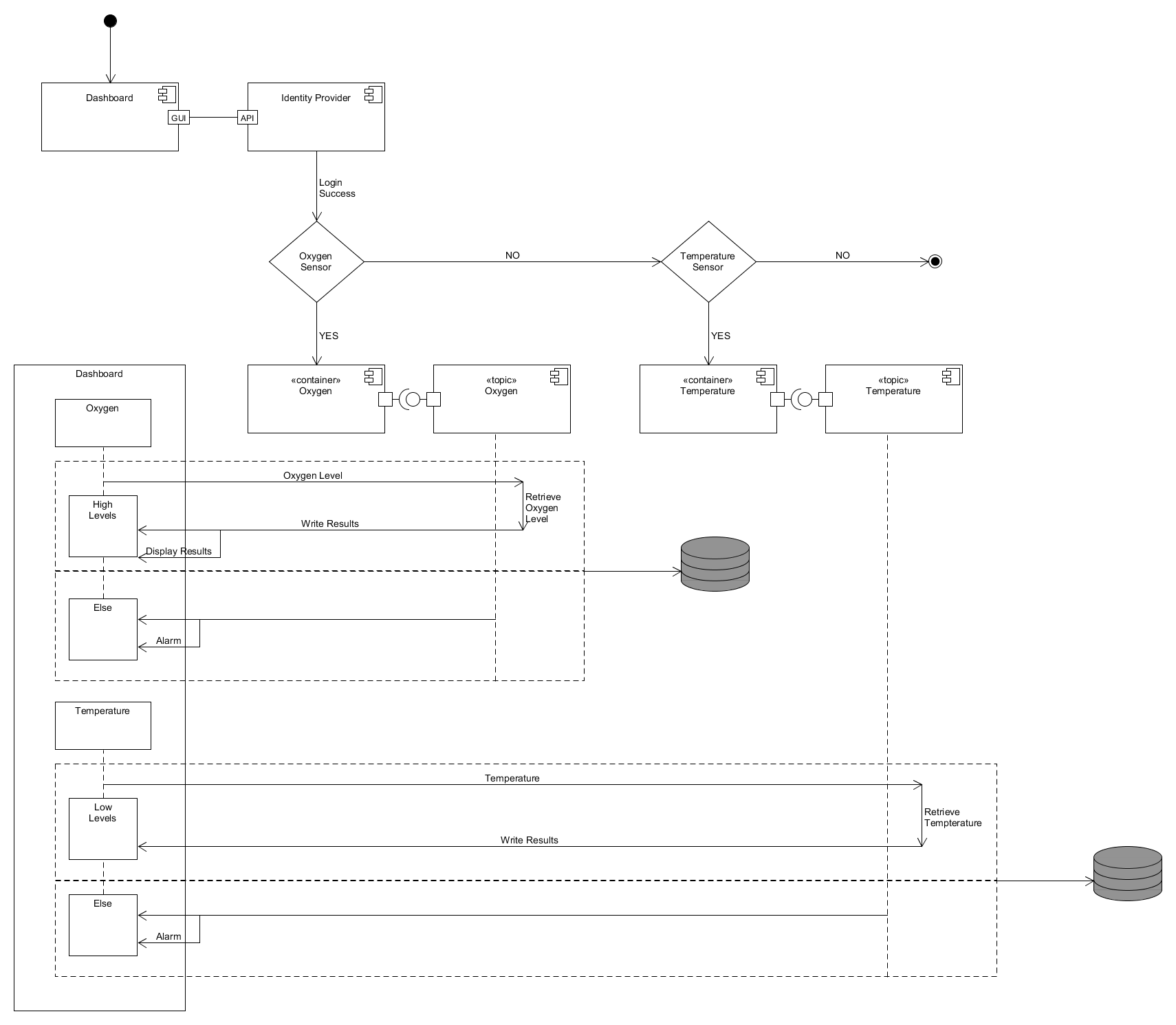


Figure 1

## 

Figure 2

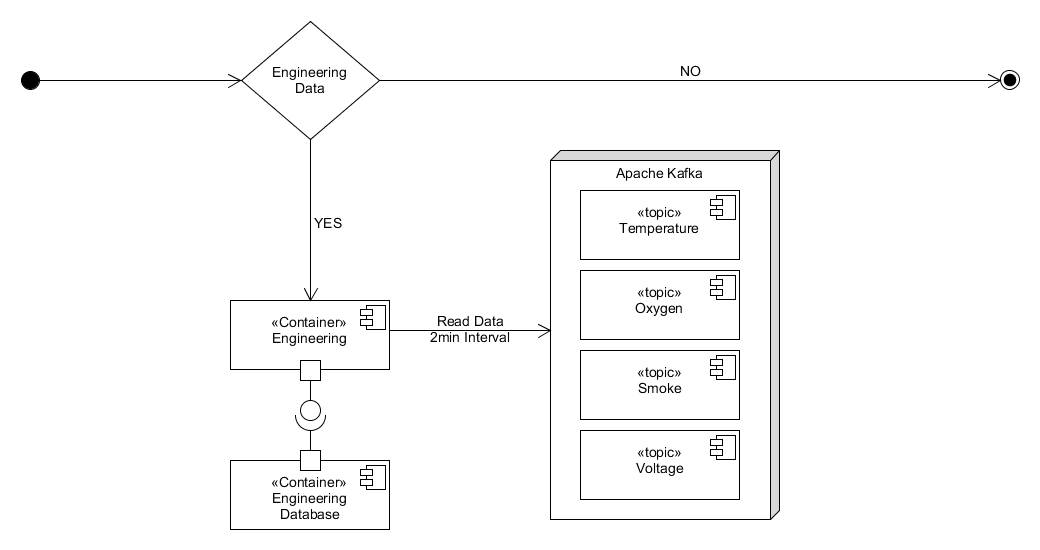


Figure 3

## Appendix C

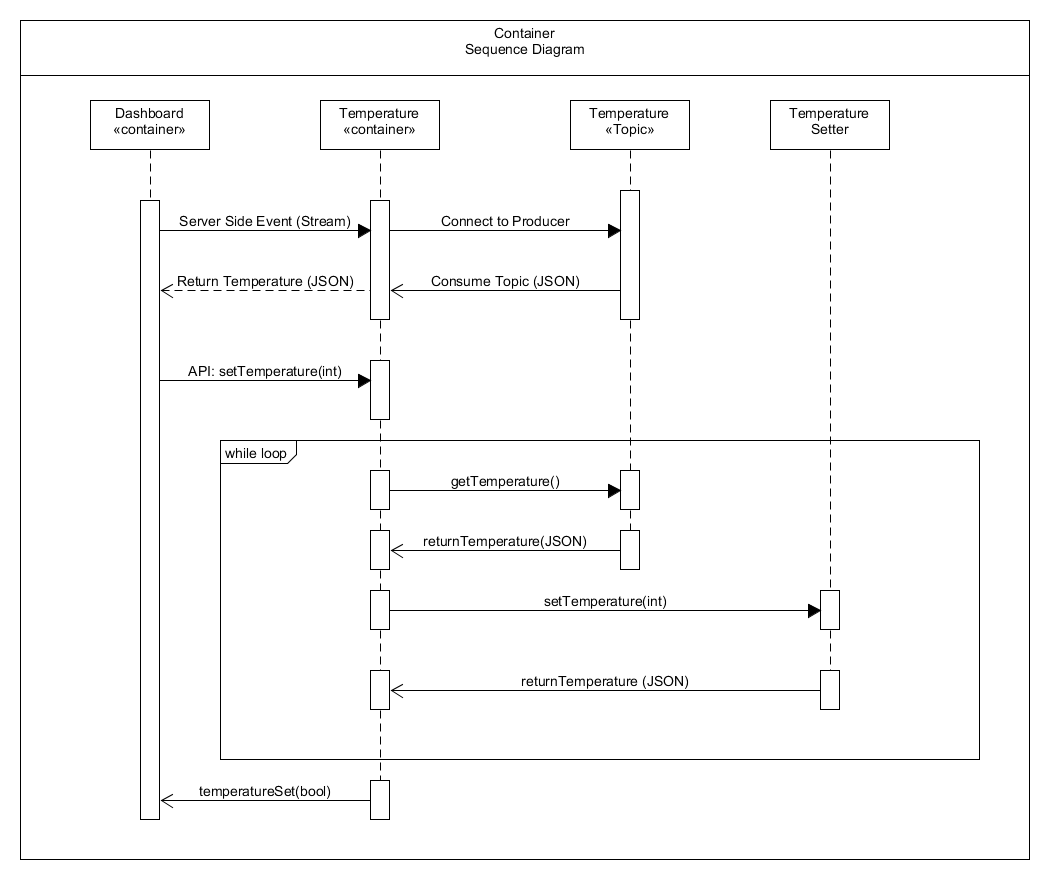


Figure 1

## Appendix D

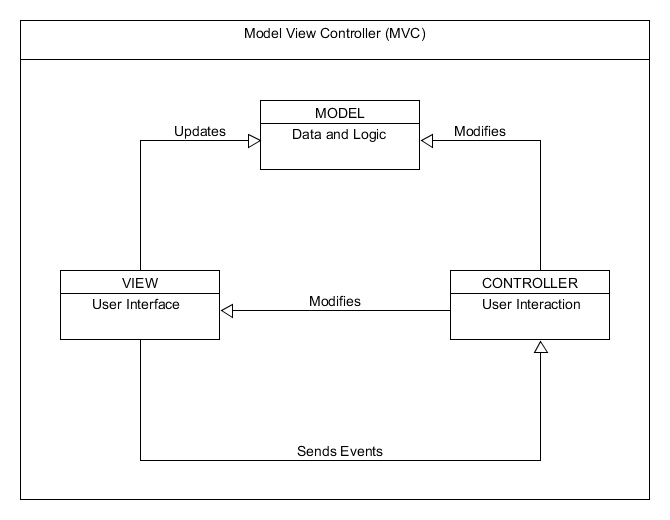


Figure 1

Appendix E

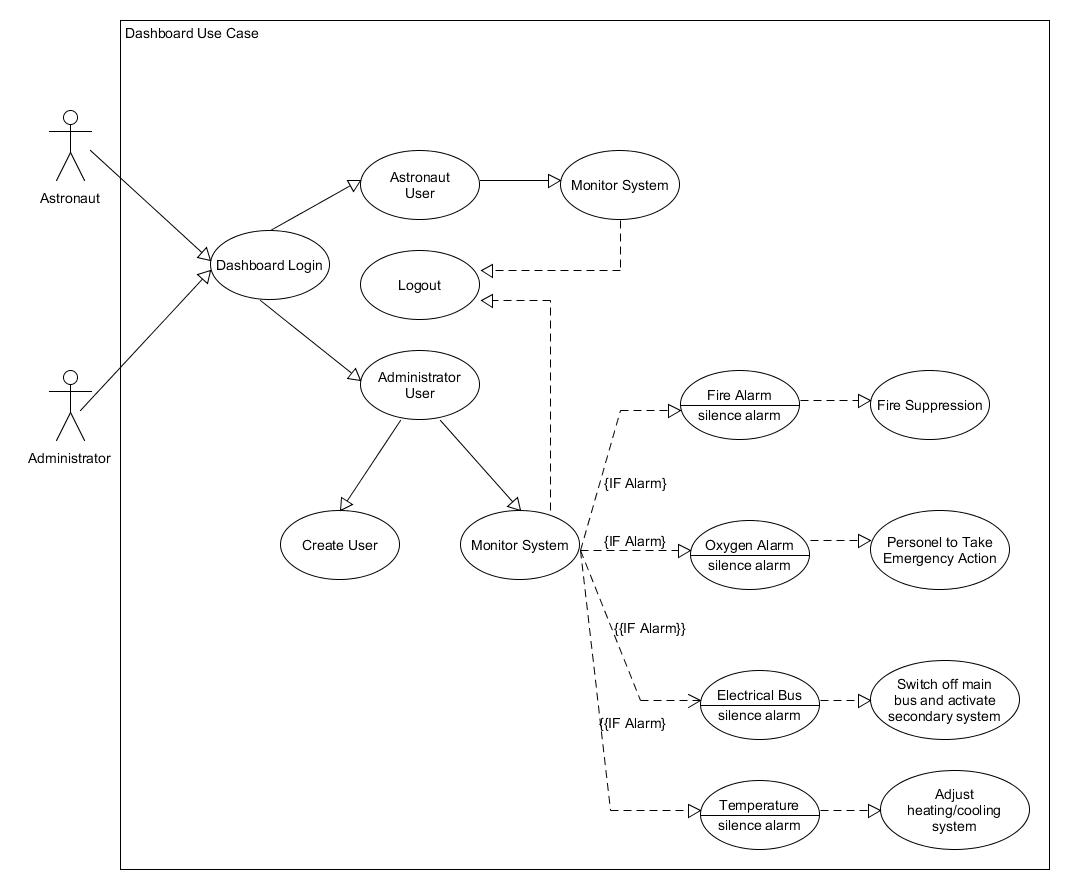


Figure 1