



**REVISION HISTORY**

| **Rev** | **Date of Revision** | **Description of Change** | **Prepared By** | **Reviewed By** | **Approved By** |
| --- | --- | --- | --- | --- | --- |
| 0.0 | 01/24/2022 | Initial creation of requirement solution document | Roshin Thomas |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Abbreviations**

| **Abbreviation** | **Description** |
| --- | --- |
| MKC | Markaz Knowledge City |
| WLS | Water Level Sensor |
| FM | Flow Meter |
| STP | Sewage Treatment Plan |

**List of Reference Documents**

| **Document Name / Description** | **Revision** | **Dated** |
| --- | --- | --- |
| Naico ITS\_Proposal\_\_Solution | 1.0 |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1.](#_heading=h.gjdgxs) Overview 5

[1.1.](#_heading=h.30j0zll) Introduction 5

[1.2.](#_heading=h.3znysh7) Document Structure 6

[1.3.](#_heading=h.2et92p0) Overview of Business Requirements 6

[2.](#_heading=h.tyjcwt) Architecture Inputs 7

[2.1.](#_heading=h.3dy6vkm) Functional Requirements 7

[2.2.](#_heading=h.1t3h5sf) Non-Functional Requirements 7

[2.2.1.](#_heading=h.4d34og8) Performance 7

[2.2.2.](#_heading=h.2s8eyo1) Maintainability and Supportability 7

[2.2.3.](#_heading=h.17dp8vu) Scalability 8

[2.2.4.](#_heading=h.3rdcrjn) Security 8

[2.3.](#_heading=h.26in1rg) Architecturally Significant Functional Requirements 8

[2.4.](#_heading=h.lnxbz9) Client’s Enterprise Architecture 8

[2.5.](#_heading=h.35nkun2) Derived Technical Requirements 8

[3.](#_heading=h.1ksv4uv) Conceptual Architecture - Business Processes & Use Cases 8

[3.1.](#_heading=h.44sinio)  Business Objective 9

[3.2.](#_heading=h.2jxsxqh)  Key roles and their influence on architecture 9

[4.](#_heading=h.z337ya) Logical Architecture - Solution Layers 10

[4.1.](#_heading=h.3j2qqm3) Core system components 11

[4.2.](#_heading=h.1y810tw) Logical expansion of Architecture blocks within Layers 12

[4.2.1.](#_heading=h.4i7ojhp) Web Application 12

[4.2.2.](#_heading=h.2xcytpi) API Gateway 13

[4.2.3.](#_heading=h.1ci93xb) API Layer 13

[4.2.4.](#_heading=h.3whwml4) Service Layer 13

[4.2.5.](#_heading=h.2bn6wsx) Database 13

[4.2.6.](#_heading=h.qsh70q) FAAS 13

[4.2.7.](#_heading=h.3as4poj) Cloud Gateway 14

[5.](#_heading=h.1pxezwc) Logical Architecture - User Interaction / Experience 14

[5.1.](#_heading=h.49x2ik5) Admin Dashboard 14

[5.2.](#_heading=h.2p2csry) Creating Tenants 14

[5.3.](#_heading=h.147n2zr) Tenant Dashboard 15

[5.4.](#_heading=h.3o7alnk) Device Administration 16

[5.5.](#_heading=h.23ckvvd) User Administration 17

[6.](#_heading=h.ihv636) Logical Architecture - Component Interaction Model 18

[6.1.](#_heading=h.32hioqz) Logical expansion of Architecture blocks 18

[7.](#_heading=h.1hmsyys) Data Model & Flow 18

[8.](#_heading=h.41mghml) Conceptual Object Design / Class Diagram 19

[9.](#_heading=h.2grqrue) API Specification 19

[10.](#_heading=h.vx1227) Deployment Infrastructure 19

[11.](#_heading=h.3fwokq0) Network Diagram 19

[12.](#_heading=h.4f1mdlm) Conclusion 19

# Overview

## Introduction

This document describes and rationalizes the solution architecture of MKC WIRAS Monitoring Solution. Primary audience of this document is the software engineering team.

MKC WIRAS Monitoring Solution aims to develop an integrated monitoring system across the MKC campus enabling timely and informed decisions. The system will enable real time monitoring of various resources within MKC campus and provide actionable insights on the demand, consumption and forecast of resources for optimal utilization, resulting in increased productivity with enhanced customer experience.

## Document Structure

This document has mainly four sections, each one is a succession of the previous.

****

## Overview of Business Requirements

MKC Smart Monitoring Solution aims to make MKC a Smart City providing the ability to remotely monitor, manage and control devices, and to create new insights and actionable information from streams of real-time or near real-time data. Automation of the following sectors would be done in a phased manner to develop MKC to be truly Smart City.

• Water Level Monitoring

• Sewage Treat Plant – Monitoring the quality parameters

• Power Consumption Monitoring

• Automated Vehicle Parking System

• Automated environment monitoring system

The MKC Smart Monitoring Solution will be an IoT based application with a responsive web interface that will run on both web and mobile platforms. There will be two types of users- Admin & Tenant. Admin & Tenant can login to the dashboard using registered username and password. Admin will have privilege to view the dashboard of all facilities inside MKC campus while Tenant will have privilege to view the dashboard of assigned facilities. Admin will also have the privilege to add new users and assign the privilege to view data of their respective facilities. Admin and Tenant dashboard will consist of data related to water resources, power resources and Sewage Treatment Plant.

As a pilot the Water Level Monitoring system will be developed to have an application dashboard showing MKC Admin and MKC tenants the water consumption on a day to day basis. Admins can also analyze, control and optimize the water consumption by various tenants in MKC campus. In the MKC facility there are three main sources of water. Mini Dam, Well and Open Pond. All the facilities within MKC campus are using water from above provided water resources. So, in order to know the water level of above water resources and control it in MKC facilities, MKC requires a dashboard where it can monitor and take adequate actions in a timely manner.

# Architecture Inputs

The major architectural inputs for MKC Smart Monitoring Solution was derived from

* MKC WIRAS Monitoring Solution rev 1.0
* Naico ITS\_Proposal\_\_Solution

## Functional Requirements

MKC WIRAS Monitoring Solution rev 1.0 has documented functional requirements in a detailed manner. Level of coverage and depth is sufficient enough to perform architecture and HLD for the project.

## Non-Functional Requirements

NFR as documented in MKC WIRAS Monitoring Solution rev 1.0. are

### Performance

The system should be able to perform without efficiency degradation as more sensors and tenants are added into the application. Near real-time operational data should be available to controls in the subsequent phases to take corrective actions.

### Maintainability and Supportability

Application logging and monitoring along with necessary documentation should be available to ensure maintainability and supportability.

### Scalability

Ensure the scalability of the application when adding more sensors, more tenants and more tenant facilities. Scale up at the network layer and the real time data storage should be considered as more sensors and tenants are added with no efficiency degradation.

### Security

The solution application should make sure to have all proper validations in place and also make sure the data is being received from correct sensor and edge device by validating their device IDs.

## Architecturally Significant Functional Requirements

Architecturally significant functional requirements are those that are subtle but has a significant impact on the architecture (say, money, timeline, complexity, compliance, etc.) and may get easily overlooked unless explicitly called out and tracked.

This will be a living list.

1. Ability for an admin to add new devices real time which would require device template to be updated in the device registry to accommodate for any new type of IOT device.
2. Ability to see real time to near real time operation data in subsequent phases of the project
3. Ability to allow duplex mode of communication
4. Ability for storage and time series analysis of device big data
5. Ability to access the application modules from Mobile devices

## Client’s Enterprise Architecture

MKC has no established EA to be referenced.

## Derived Technical Requirements

Since NFR are derived from FR for this project, many of the technical requirements are also documented in that process, hence it is not required to have separate technical requirements captured.

# Conceptual Architecture - Business Processes & Use Cases

MKC WIRAS Monitoring solution will be an IoT application that is cloud native, following microservice pattern and serverless-based with a responsive web interface that will run on both web and mobile platforms.

The application will provide tenants the ability to monitor and control their resources. A super user will be able to add tenants with necessary roles and permissions. Tenants in turn can add other sub-tenants or users into the application. A tenant can view his/her, sub-tenants and user level data. The privileges and access will narrow as we go down the hierarchy.

Tenants can add devices (bulk provision or add individual devices) and manage their Lifecyle. They can also view their current state and any error codes transmitted to detect issues.

The data from the devices would be available from the dashboard and would indicate levels of usage, quality parameters, consumption etc. There would also be the ability to set alerts or notifications to indicate a high or low threshold of the resources.

It should be possible to control any actuators or devices from the application when required. Hence a duplex mode of communication should be supported.

## Business Objective

1. The application will allow registration of tenants and sub-tenants into the system with necessary authorizations and authentications to view only data relevant to them
2. The application will allow devices (/gateways/aggregators) to be provisioned in bulk and from the application
3. The application will allow near real time data from the sensors a to be displayed on the dashboard (depending on the type and capability of sensor the data might be streaming or batch)
4. Users will be able to view the data and insights from the dashboard
5. Admin users would be able to add or remove tenants, view the data for tenants and manage the lifecycle of devices
6. In subsequent phases of development, the application would also allow control of actuators from the application.
7. The application should offer guaranteed reliable service available to the subscribers with defined performance, security, and availability measures

## Key roles and their influence on architecture

1. **Super User**

Super user is the initial user created who can bulk provision devices and add tenants.

Administer authorizations and access levels.

Administer lifecycle of the devices.

1. **Tenants**

Tenants are a separate business entity or organization who purchase or use tenant devices and physical entities. Tenants may have multiple sub-tenants and/or hundreds of users.

A tenant can administer their owned or leased devices and add sub-tenants and users

1. **Sub-tenants**

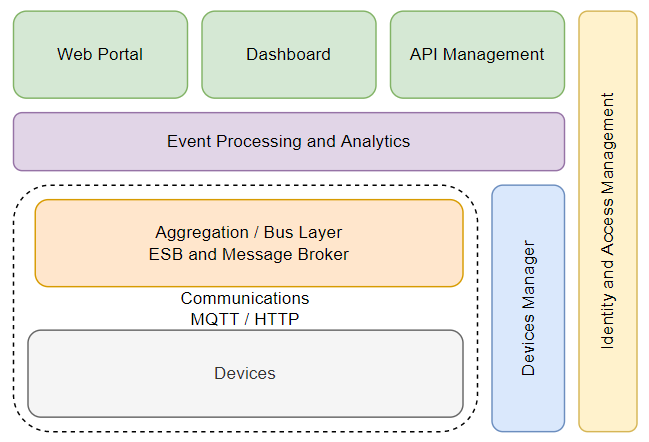
Sub-tenants are separate business entities who may be customers of the tenants.

They can purchase or use tenant devices and/or assets. Sub-tenants may have multiple users and millions of devices and/or assets.

1. **Users** - users are able to browse dashboards and manage devices to which access has been provisioned.
2. [**Devices**](https://thingsboard.io/docs/user-guide/ui/devices/) - basic IoT entities that may produce telemetry data. E.g. sensors, actuators, switches

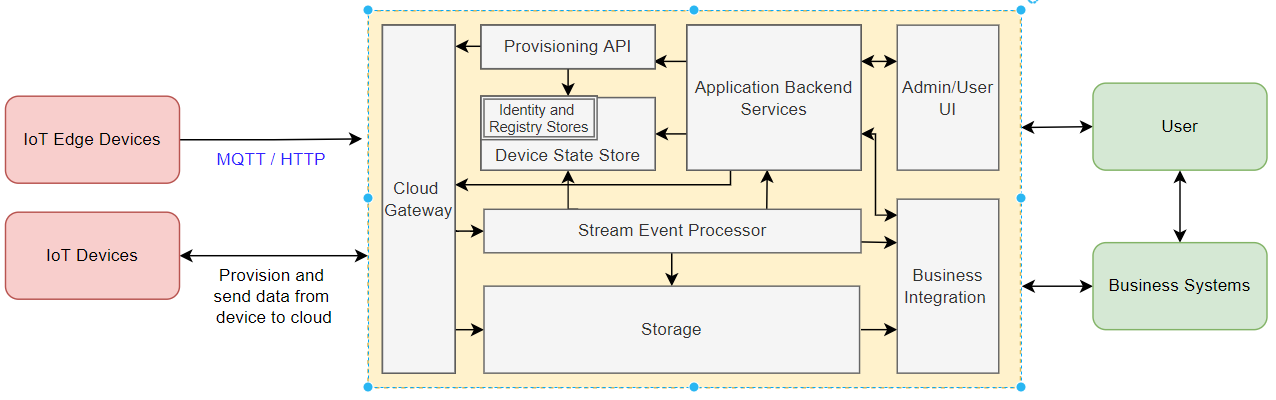
# Logical Architecture - Solution Layers

The high-level layers in the architecture can be divided as client/external communication, event processing and analytics, aggregation layer, transports, and devices along with two cross-cutting layers of device management and identity and access management.



1. **Communication Layer** - Enables the devices to communicate with other systems
2. **Event processing and analytics** - Takes events from the bus and enables to act on it
3. **Aggregation** - Aggregates and brokers communications between devices. Aggregates and combines communications from different devices and routes communications to a specific device, and bridges and transforms between different protocols.
4. **Devices** - IoT devices that either indirectly or directly attach to the Internet.
5. **Device Management** - Communicates with devices and provides both individual and bulk control of devices. Maintains the list of device identities and maps these into owners. It must also work with the identity and access management layer to manage access controls over devices.
6. **Access Management** - Provides identify and access management services.

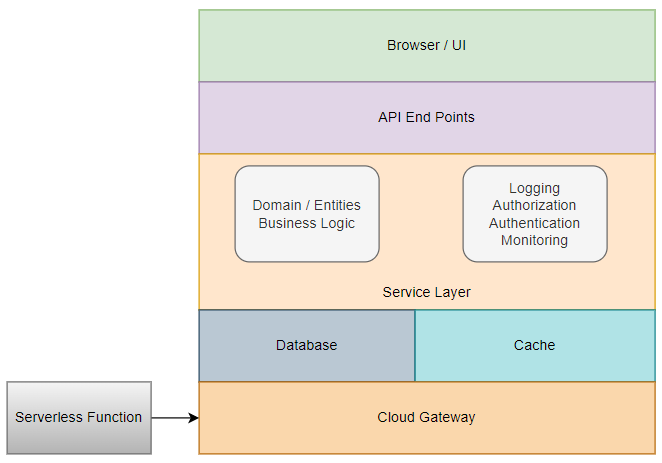
These components in the layers are detailed below and give a high-level overview of the solution.



## Core system components

* + 1. **Devices** (and/or on-premise edge gateways) that have the ability to securely register with the cloud, and connectivity options for sending and receiving data with the cloud
    2. The **cloud gateway** acts as a hub providing secure connectivity, protocol translations, data and event ingestion and bi-directional communication with devices including device management with command and control capabilities.
    3. **Stream processors** process large streams of data records and evaluate rules for those streams.
    4. **Storage** can be divided into warm path for data that is required to be available near real time and cold path for long term storage.
    5. The **user interface** includes admin and user screens that helps visualize data and facilitate device management.
    6. **Business integration** facilitates executing actions based on insights from device telemetry data. Integration could include storage of informational messages, alarms, sending alert notification and more.
    7. **Provisioning API** provides an external interface for modifying the device identity store and the topology and entity store.
    8. **Application Backend** services support application services for the web application. This could be device lifecycle management, data transformation services or user management services.

## Logical expansion of Architecture blocks within Layers



### Web Application

Web application will be the one where the users can interact with the application. This will comprise front-end technology which will be hosted on a cloud platform. API calls will be made to the server side to post and retrieve data.

### API Gateway

Since individual backend modules are designed as microservices, access to a particular service will be through an API Gateway. The API Gateway pattern would enable decoupling of the web application from the microservices and also allow aggregations when necessary to reduce roundtrips. This will be in a backend for frontends pattern.

### API Layer

NodeJS would be used to create backend services. Considering the community support, maturity, easiness of development and more importantly security aspects of the application, NodeJS will be the technology used for creating web api layers. Web API will be having endpoints secured with a token based authentication system which will authorize the user roles. The user roles and corresponding permissions are mapped in the database. So, the api endpoint will be available for only the users with corresponding roles and permissions.

### Service Layer

The service layer contains all the business logic. This acts as a middle layer between the API endpoints and the database. The business logic can communicate to external applications (application front end, background services etc) only through the API endpoints. The service layer would be developed as microservices to ensure greater scalability and flexibility to choose appropriate technologies. The subsystems would be built as discrete services that are independently deployable, and able to scale independently and also communicate over REST/HTTPS using JSON.

### Database

All the application data will be residing in the database. For databases, we have options like relational database like Microsoft SQL and NoSQL database like MongoDB. The application which stores user data, device metadata etc could utilize open source Postgres. The device data being timeseries data it is advisable to use MongoDB Timeseries, Postgres Timescale or InfluxDB. Since the microservice pattern is followed ideally individual subsystems/services would have their own database.

### FAAS

Serverless functions can be utilized to pull data from message queue, apply routing rules to ingested device data, sent alert notifications etc. FASS also enables scalability of the application easily without needing additional resources

### Cloud Gateway

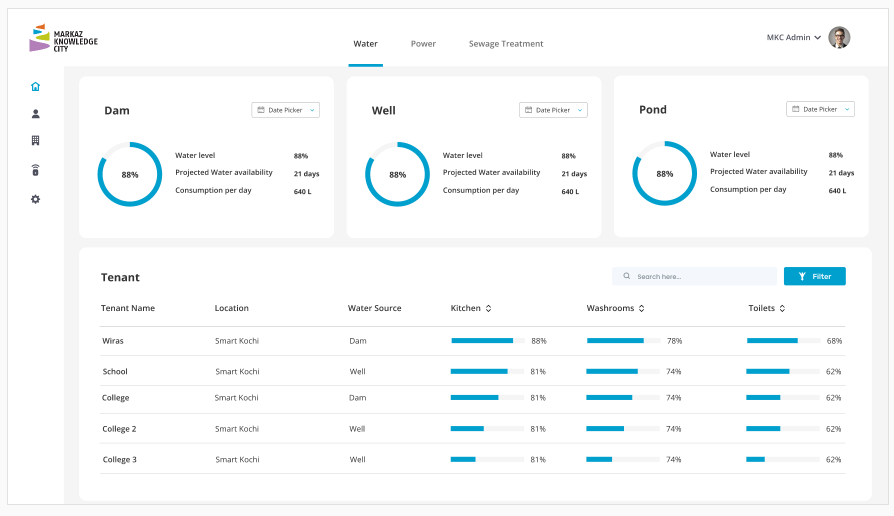
A cloud gateway or IoT gateway provides a secure device-cloud-device (D2C and C2D) communication for IoT devices. It also provides device registration and authentication, device management tools and message delivery/acknowledgment features.

# Logical Architecture - User Interaction / Experience

The below diagram represents a UI and role wise access.

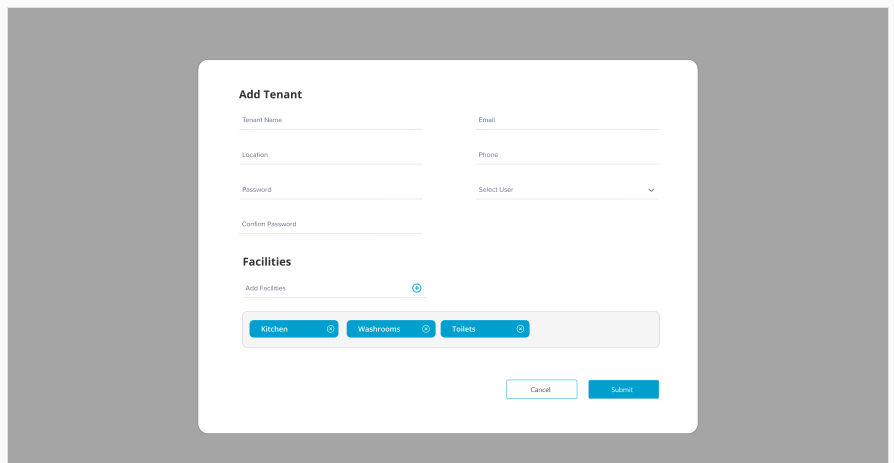
## Admin Dashboard

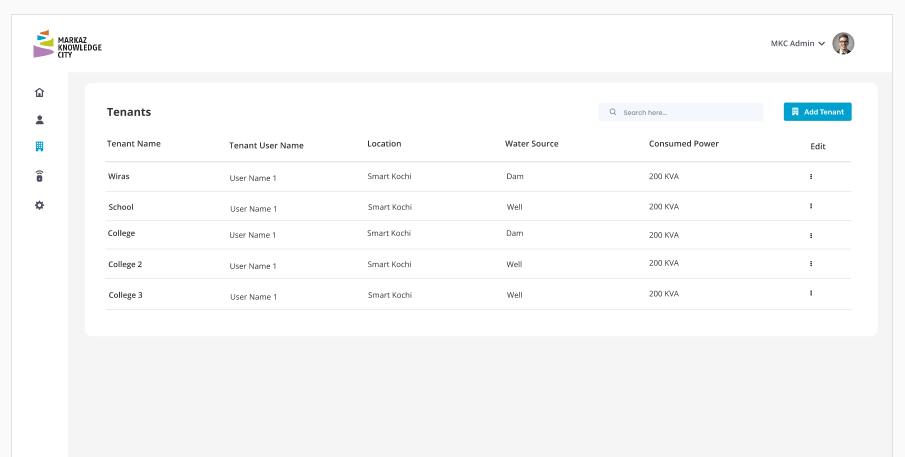
This is the landing page where the admin lands when logged into the application. Here admin will be able to view the dashboards



## Creating Tenants

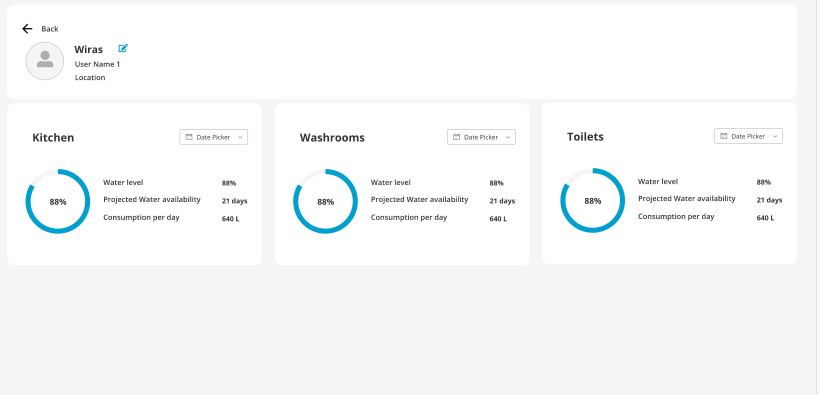
These is the screen for the Admin to Create or Administer Tenants





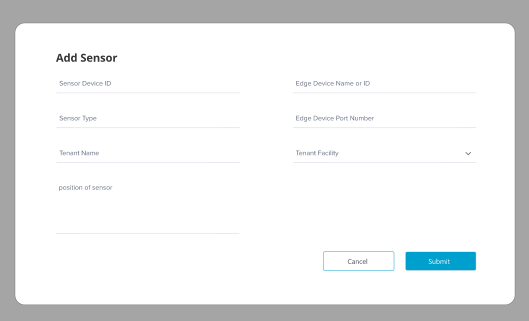
## Tenant Dashboard

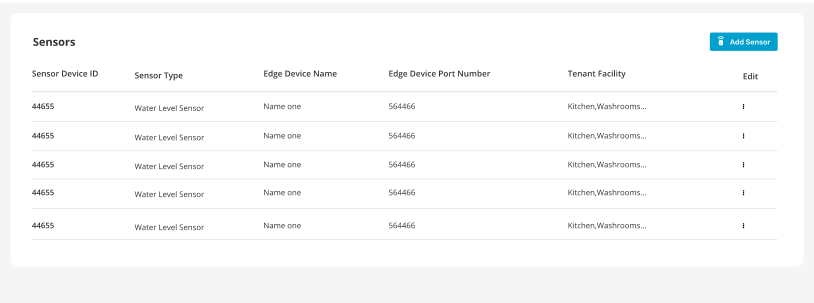
This is the page where a Tenant lands when logging in



## Device Administration

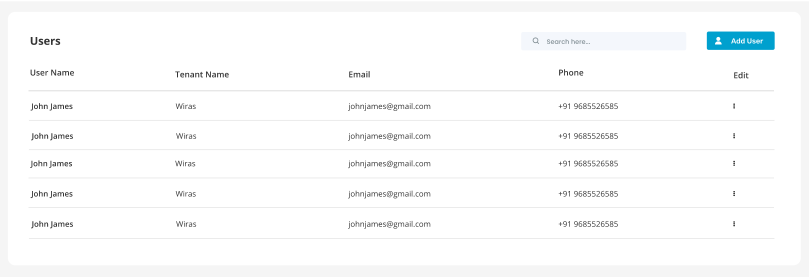
These are the screens for the tenants to create / administer various devices

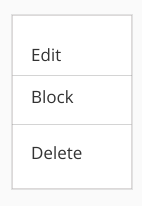




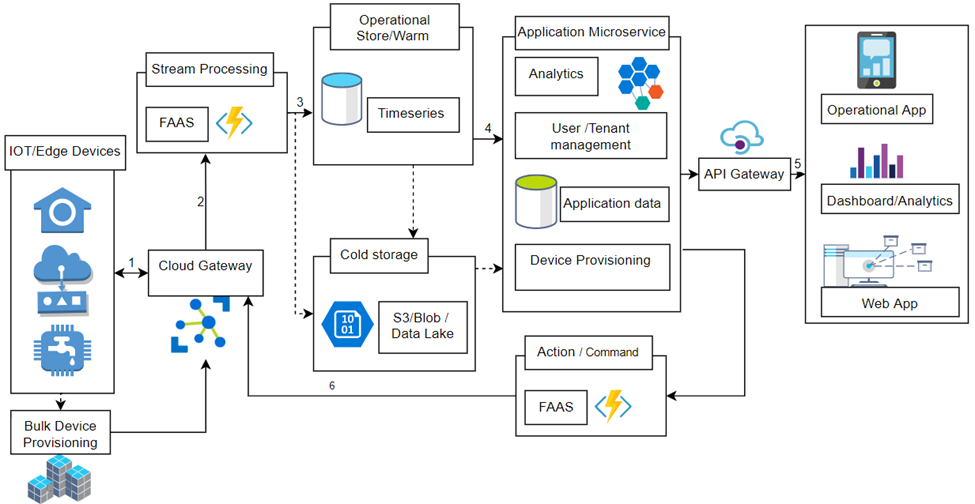
## User Administration

These are the screens for the tenants to create / administer their users





# Logical Architecture - Component Interaction Model



## **1.1.** **Data Flow**

1. Events generated from IoT devices are sent to IoT Gateway as a stream of messages (the frequency and pattern would vary based on the actual device but a generic flow is considered). IoT Gateway stores streams of data in partitions for a pre-configured amount of time.

2. A stream processing (serverless function) picks up the messages in real time from Cloud Gateway, processes the data based on the business logic and sends the data to the serving layer for storage.

3. The stream processor stores the data directly or after transformation to a time series database (warm storage). As a timeseries database would be costly to store all historical data, the Cloud Gateway can route the raw data or from stream processing into a datalake or blob storage for long term (cold storage)

4. The backend services are built on top of the storage layer (exposed APIs) and communicate to the Web, mobile and other applications can be built on the storage layer. For example, you can expose APIs based on the storage layer data for third-party uses.

5. User Interfaces communicate via the API gateway to the underlying services for dashboards, analytics, device lifecycle management etc.

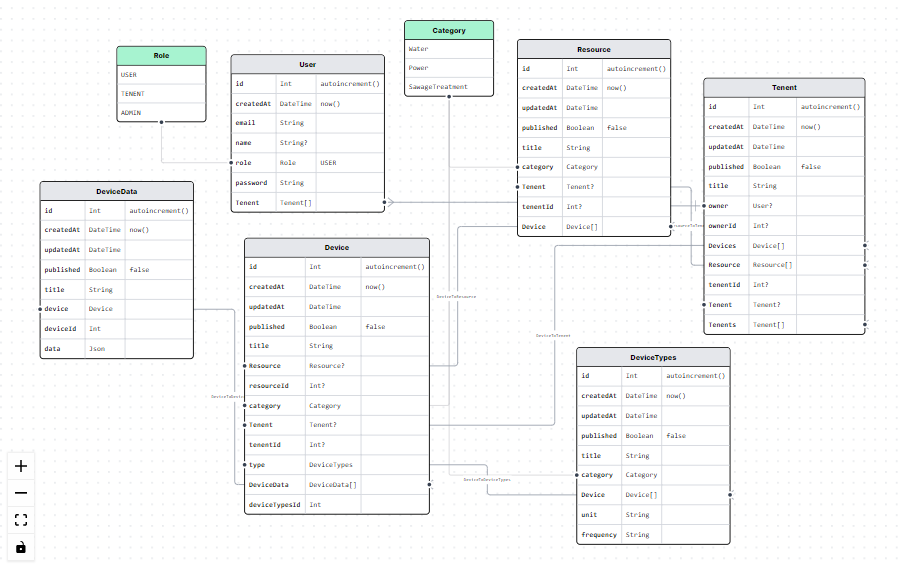
6. Any device error code, pre-defined thresholds etc trigger actions such as notifications (functions), commands to device (Cloud Gateway) etc.

## Logical expansion of Architecture blocks

Below is the expansion of the blocks, their purpose and options are also listed.

| **Infrastructure components** | **Purpose** | **Azure** | **AWS alternative** | **Open Source/**  **Alternatives** |
| --- | --- | --- | --- | --- |
| Cloud Gateway | Acts as a hub allowing IoT devices to connect to the cloud and securely send and receive data and commands.  Support for several messaging patterns (device-to-cloud telemetry, uploading files from devices, and request-reply ).  Allows devices to securely communicate (token-based authentication or X.509 certificate authentication)  Device communication is secured using the Transport Layer Security (TLS) standard.  Easy scale up as the number of devices increases. | IoT Hub | IoT Core | ThingsBoard |
| Application Database | Host the application DB | Postgres (DBaaS) | Postgres AWS RDS | Postgres |
| Time Series Database | Warm store for storing IoT telemetric data. | Cosmo DB | Timestream | Timescale (Postgres), MongoDB Time series,Influx |
| Load Balancer | Used for load balancing between the nodes | Azure Load balancer: Standard | AWS Classic load balancer | Nginx (Plus) |
| FAAS | Serverless functions will be used to evaluate streaming data and also to enable commands (C2D) or notification based on device data stream and any thresholds set. | Azure functions | AWS Lambda | OpenFaaS |
| Authentication | Authentication of the user to the web application. | Azure AD B2C | AWS Cognito | Keycloak |
| Networking | Provides enhanced security and network isolation, and integrates well with load balancing and firewalls. | Azure Virtual Network | AWS VPC | On-premise |
| Key Management | Used to store application secret keys, encryption keys, SSL certificate keys, etc. | Azure Key Vault | AWS KMS | Knox |

# Data Model & Flow



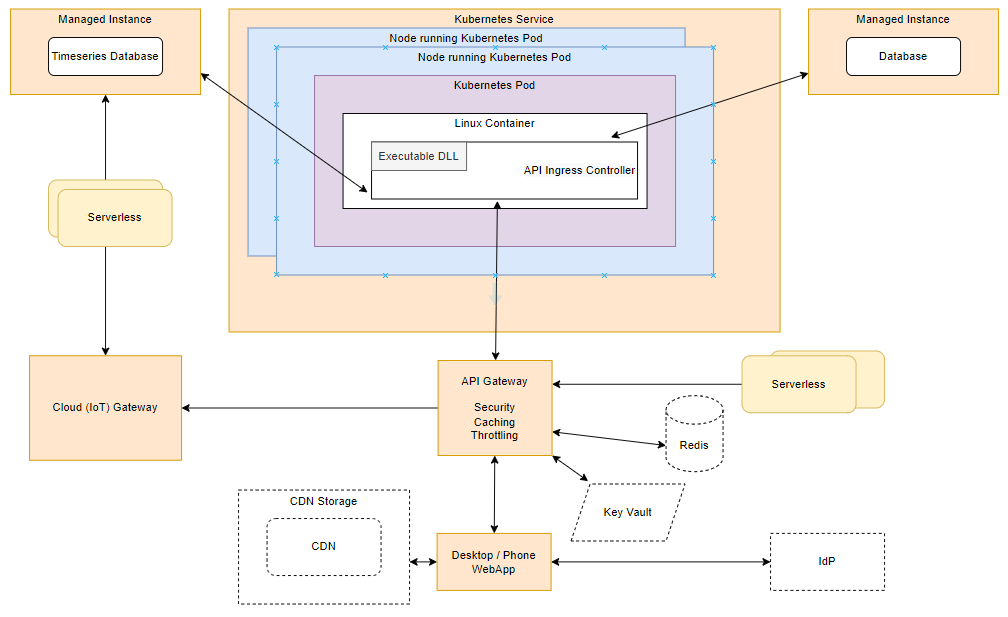
# Conceptual Object Design / Class Diagram

<<Would be updated once low level design is completed >>

# API Specification

<<Would be updated once low level design is completed >>

# Deployment Infrastructure



# 

# Architectural Decision Record

| **Issue** | **Decision** | **Assumptions** | **Constraints/Implications** | **Status** |
| --- | --- | --- | --- | --- |
| Microcontrollers with edge devices – intelligent edge. | Microcontrollers will not be used. |  | The application has to perform the conversion, device status check, added resilience etc. |  |
| Microservice architecture | Monolith | Number of users will be limited though the number of devices could scale. | Conversion in future to microservice would have significant development effort. To ease it we can effectively ensure each module is separately envisioned and communicates through APIs. |  |
| Response time | Caching, CDN | Since no SLA has been defined considering cost Caching or CDN for frontend app will not be utilized |  |  |
| Automatic Device synchronization (example Azure DPS) | Device provisioning will have to be done by a user with the necessary privilege. | The number or devices added in subsequent cycles would be less and not warrant an additional provisioning service. | Bulk provision scripts should be provided. |  |
| Device security and authentication | D2C and C2D communication would be secured with TLS standards. No key/token based device security support is required. | Data will be available at a public endpoint. |  |  |

# Conclusion

The architecture is designed after considering the best practices and making use of the latest technology trends.

# 