CS6.401 Software Engineering Project 3

Smart City System: Software Architecture Design

Team 16:

Team Members	Roll Number	Email
Chandu Chegu	2022201062	chegu.sai@students.iiit.ac.in
Nemalikanti V M Dheeraj	2022201022	dheeraj.nemalikanti@students.iiit.ac.in
Krishna Chaitanya Dama	2021202005	krishna.dama@students.iiit.ac.in
Sai Sumith	2022202004	patha.sumith@students.iiit.ac.in
Prem	2022201036	Prem.kanamarlapudi@students.iiit.ac.in

Task 1: Requirements and Subsystems

Functional Requirements:

1. **Data Collection and Management**: The system should be able to collect, manage, and store data from different types of IoT nodes, including air quality sensors, water quality sensors, water quantity sensors, solar power sensors, AC energy consumption sensors, and crowd monitoring sensors.
2. **Real-time Visualization**: The system should provide near real-time visualizations of different parameters, such as air quality, water quality, energy consumption, and crowd information, through interactive dashboards that are accessible through web and mobile apps.
3. **Alerts and Notifications**: The system should send SMS/email alerts to users in the event of higher water quality/air quality/energy consumption, or any emergency situation.
4. **Crowd Information**: The system should continuously monitor crowd information in real-time, including crowd forecasting and provide real-time crowd information for specific areas within the smart city.
5. **Energy Savings Recommendations**: The system should provide energy savings recommendations to stakeholders based on data collected from AC energy consumption sensors.
6. **Automated Control**: The system should allow for automated control of lamp posts, including configurable timings for on/off, based on data collected from sensors.
7. **Smart Classrooms**: The system should support automated on/off control of fans, lights, and ACs in smart classrooms, based on occupancy and usage patterns.
8. **Authentication**: The system should use sensors to authenticate users at the entrance of the campus, with a response time of no more than 1 second.
9. **External Stakeholder Access:** The system should provide easy means for external stakeholders to access and query up to 30 days of data generated by the Smart City IoT system.
10. **Third-party APIs**: The system should provide third-party APIs to allow different stakeholders to access datasets for performing experiments and research.

Non-functional Requirements:

1. **Interoperability**: The system should be able to communicate with different IoT nodes that may use different communication protocols, by utilizing standard interoperability middleware, such as OneM2M.
2. **Scalability**: The system should be scalable to handle the continuous inflow of data from a large number of IoT nodes, as well as accommodate future expansion and growth of the smart city system.
3. **Performance**: The system should provide real-time visualizations and alerts with low latency, ensuring high performance and responsiveness.

4. **Security**: The system should ensure the privacy and security of user data, by not storing any sensitive data related to the users, and implementing appropriate authentication and authorization mechanisms.
5. **Reliability**: The system should be reliable, with high availability and fault tolerance, to ensure uninterrupted operation of the smart city system.
6. **Usability**: The system should be easy to use and navigate, with user-friendly interfaces for web and mobile apps, catering to users with or without smartphones.
7. **Access Control**: The system should provide role-based access control mechanisms to restrict access to functionalities and data based on user roles and permissions.
8. **Data Retention**: The system should define data retention policies and ensure compliance with data retention regulations.
9. **Machine Learning Integration**: The system should be able to integrate ready-made machine learning models and algorithms for various functionalities, such as crowd forecasting and water quality forecasts.
10. **Power and Resource Constraints**: The system should be designed to handle IoT nodes with limited power and processing resources, and accommodate the constraints of outdoor deployments where external power sources may be limited.

Architecturally significant requirements:

1. **Interoperability**: The system should be able to communicate with and gather data from IoT nodes that may be using different IoT communication protocols. This is a key requirement because it ensures that the system can seamlessly integrate with different types of IoT devices and capture data from them, regardless of the communication protocol they use. This requires the use of standard interoperability middleware such as OneM2M to enable communication and data exchange between heterogeneous IoT devices.
2. **Scalability**: The system should be able to handle a large number of IoT nodes (300 in this case) and the continuous inflow of data from them. This is a key requirement as the system needs to process and analyze a significant amount of data in near real-time to provide timely and accurate services. Scalability ensures that the system can handle the increasing data volume and user demands without compromising performance and responsiveness.
3. **Real-time data processing**: The system needs to process data from IoT nodes, such as air quality, water quality, energy consumption, and crowd information, in near real-time to provide real-time visualizations, alerts, and forecasts. This is a key requirement as it enables the system to provide timely information and alerts to users, facilitating quick decision-making and effective management of resources.
4. **Ease of external stakeholder integration:** The system should provide easy means for external stakeholders to leverage the data generated by the Smart City IoT system, including the ability to query historical data up to 30 days. This is a key requirement as it promotes data sharing and collaboration among different stakeholders, such as researchers, policymakers, and businesses, for further analysis, experimentation, and innovation.
5. **User authentication and performance:** The system should be able to authenticate users at the entrance of the campus within 1 second using sensors. This is a key requirement as it ensures a quick and efficient user authentication process, enabling smooth access to the campus. Fast authentication also reduces waiting time and enhances the overall performance and user experience of the system.
6. **Flexibility in access control:** The system should provide flexibility in access control, allowing different stakeholders to have different levels of access to functionalities and data based on their roles and permissions. This is a key requirement as it ensures that stakeholders only have access to the relevant functionalities and data, preventing unauthorized access and ensuring data confidentiality and integrity.

List of all subsystems :

1.Sensor Manager

2.Deployer

3.Platform Manager

4.Application/Node Manager

5.Scheduler

6.Monitoring and Fault tolerance

Sensor Manager:

The four functions of sensor manager are:

1. **View/modify sensor metadata** - The metadata of the sensor such as type, model number and other technical specifications are displayed on the dashboard . It can modify the frequency with which sensor sends the data.
2. **VM Query Authentication** - It verifies whether the application has the necessary authorization to access the data it has requested .
3. **Data Preprocessing -** The data preprocessing module is responsible for preparing raw data generated by iot devices for further processing . It includes various data cleaning and normalization techniques.
4. **Data Visualization and analytics** - This module allows user to visualize data using graphs, charts, tables . It helps users understand graphs , plots and anomalies in the data. This module provides real time monitoring of data streams, allowing users to track changes as they happen.

Deployer:

This module is responsible for deploying applications to the appropriate nodes and ensuring that the associated sensors are correctly identified and managed . When it receives the application id and location of the configuration files from the scheduler , the deployment manager starts the deployment process. The deployment manager communicates with the load balancer which is responsible to find the appropriate nodes on which to deploy the application . The load balancer receives information from the deployer about the number of nodes required for the application and find the nodes with the least load . After that the deploys the application to the specific nodes and ensures that the application is running correctly.

Platform Manager:

The Platform Manager is a module that connects with a database containing applications and configuration settings. This module is responsible for handling requests for data related to the applications from other modules. Communication between the Platform Manager and other modules occurs through Kafka messaging.

When an end-user schedules an application, the scheduled information is sent to the Platform Manager. The Platform Manager then sends the scheduled information to the Scheduler module through Kafka messaging. This allows the Scheduler module to access the information needed to execute the scheduled application.

Application/Node Manager:

This functional purpose of this module is to manage the virtual machines . This is done by making decisions such as which virtual machines are used , when they are to be used and whether more virtual machines are needed . The applications provided by the application developer are run on virtual machines after proper allotment of nodes .

Scheduler:

The scheduler is responsible for scheduling the application to run at predetermined time intervals . When it receives application id and the location of the configuration file from the platform manager it fetches the configuration file and checks whether it satisfies the constraints of the platform.

Once the configuration file is verified the scheduler stores the scheduling details in a list , and sets up a cron job to trigger appropriate action at the scheduled time interval. When the end time is reached the scheduler sends a request to the deployer to stop the application .

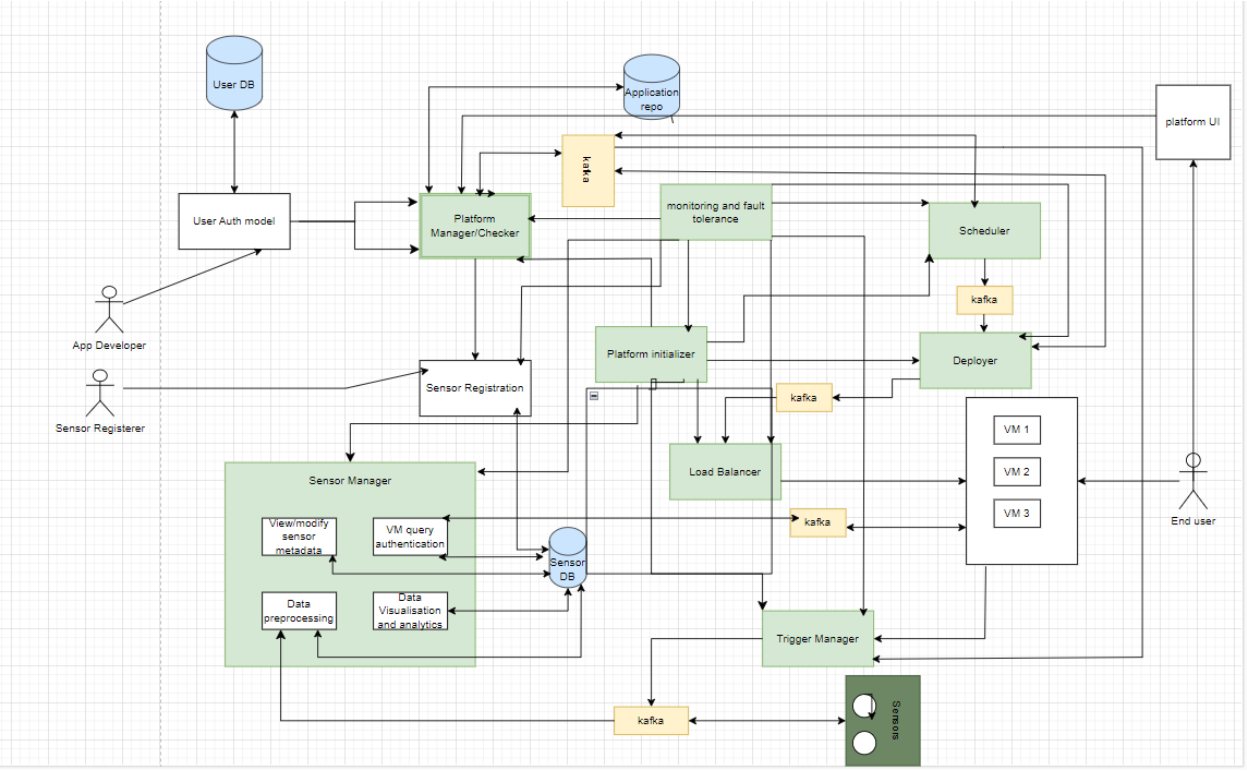
Monitoring and Fault tolerance:

The Monitoring Module is connected with every other module in the platform. To ensure that all modules are functioning correctly, each module sends a heartbeat message to the Monitoring Module every 20 seconds.

If a module fails to send a heartbeat message to the Monitoring Module, a message will be stored in the log file indicating that the module has failed. This allows administrators to identify and address any issues that may arise within the platform.

Communication between all modules and the Monitoring Module occurs solely through Kafka messaging. This enables efficient and reliable communication between the modules and the Monitoring Module.

High level overview of the system



Task 2: Architecture Framework

Design Decisions

Title: **Microservices Architecture for Data Processing**
Status: Active

Context: Designing an architecture for processing and analyzing the vast amount of data generated by the IoT nodes, and providing various services such as real-time visualizations, alerts, and control features to the users.

Decision:

- Adoption of microservices architecture to improve scalability, availability, and maintainability of the system.

- Decomposition of the monolithic application into smaller, independent services that can be developed, deployed, and maintained separately.
- Use of containerization technologies such as Docker and Kubernetes to manage and orchestrate the microservices.
- Use of message-based communication such as RabbitMQ for inter-service communication.

Consequences:

- Improved scalability and availability due to the use of microservices architecture.
- Enhanced agility and maintainability due to the independent deployment and maintenance of individual services.
- Increased development and deployment complexity due to the distributed nature of the system.
- Increased operational overhead due to the need for managing the containerized environment.

Title: **Kafka-based Message Passing between Microservices**
Status: Active

Context: Designing an architecture for the Smart City System that involves a large number of microservices that need to communicate with each other in real-time. The microservices are responsible for processing the data generated by IoT nodes and providing various services such as real-time visualizations, alerts, and control features to the users.

Decision:

- Adopting Kafka as the messaging system to facilitate real-time communication between microservices.
- Setting up Kafka brokers and topics to enable the publishing and subscribing of messages by microservices.
- Using the Kafka Connect framework to ingest data from various sources such as databases and IoT devices into Kafka topics for processing by microservices.
- Implementing a schema registry to ensure compatibility between the message formats used by different microservices.

Consequences:

- Improved real-time communication between microservices due to the low-latency message passing provided by Kafka.
- Increased scalability and fault-tolerance due to the distributed and decoupled nature of Kafka-based message passing.
- Increased complexity due to the need for setting up and managing the Kafka brokers, topics, and schema registry.
- Increased development time and effort due to the need for implementing Kafka-based messaging in each microservice.

Title: Usage of oneM2M for Sensor Data Acquisition
Status: Active

Context: Designing an architecture for the Smart City System that involves a large number of IoT nodes deployed across the city for measuring air quality, water quality, water quantity, solar power, AC energy consumption, and crowd monitoring. The data provided by these IoT nodes needs to be collected and processed by the microservices for providing various services to the users.

Decision:

- Adopting oneM2M as the communication protocol between the IoT nodes and the microservices for data acquisition.
- Setting up a oneM2M-based communication infrastructure to enable data exchange between the IoT nodes and the microservices.
- Implementing a oneM2M-based data model to ensure compatibility between the sensor data produced by the IoT nodes and the data consumed by the microservices.
- Using oneM2M's resource-oriented communication model to enable discovery, control, and management of the IoT nodes by the microservices.

Consequences:

- Improved compatibility and interoperability between the IoT nodes and the microservices due to the use of oneM2M.
- Enhanced scalability and flexibility due to the distributed and resource-oriented nature of oneM2M-based communication.
- Increased complexity due to the need for setting up and managing the oneM2M infrastructure and data model.
- Increased development time and effort due to the need for implementing oneM2M-based communication in each microservice.

Stakeholders:

According to ISO 42010 standard when identifying the stakeholders following are to be considered and included when applicable

- users of the system
- operators of the system
- acquirers of the system
- owners of the system
- suppliers of the system
- developers of the system
- builders of the system
- maintainers of the system.

Now let us consider whatever that are applicable to our smart city system.

The possible stakeholders can be

1. **End Users :** End Users are certainly the citizens of the smart city who can access the information by using app or message/other services based on their connectivity(smartphone or not)
2. **IOT Engineers:** IOT Engineers are responsible for developing IOT devices.
3. **Software Architect:** He is responsible for designing the entire architecture.
4. **Software Developer:** The developer is responsible for developing the software.
5. **Government Organisation officials**
    - **Traffic Management:** This stakeholder is responsible.
    - **Urban Planning:** This stake holder is responsible for entire city plan required for smart city system.
6. **App Developer :** responsible for developing app .
7. **ML Engineer :** ML engineer is responsible for data preprocessing, data insights and various other ML tasks like prediction, recommendation system etc..
8. **Sensor Deployers:** This stakeholder deals with deploying the sensors
9. S**ecurity Engineers:** This security engineers are for handling the security aspect for our entire smart city system.

Concerns of each stakeholder:

1. **End Users:**
    - Concerns regarding data privacy and security
    - Ease of use and accessibility of the mobile app
    - Accuracy and reliability of the information provided by the app
2. **IOT Engineers:**
    - Compatibility of IoT devices with different communication protocols
    - Power and processing requirements for the IoT devices
    - Security and privacy of the data transmitted by the IoT devices
3. **Software Architect:**

- Scalability of the architecture to handle large amounts of data
- Integration of different IoT devices and communication protocols
- Security and privacy of the data collected and transmitted

4. **Software Developer:**
    - Ability to implement the architecture design efficiently
    - Meeting the requirements of different stakeholders
    - Ensuring the software is reliable and secure
5. **Government Organisation officials:**
    - **Traffic Management:**
        - Real-time data accuracy for effective traffic management
        - Integration with existing traffic management systems
    - **Urban Planning:**
        - Integration with existing urban planning systems
        - Scalability to handle large amounts of data for city planning purposes
6. **App Developer:**
    - Ensuring the app is user-friendly and accessible
    - Integration with the architecture and the IoT devices
    - Security and privacy of user data
7. **ML Engineer:**
    - Ensuring the data preprocessing and insights are accurate and reliable
    - Developing effective prediction and recommendation systems
    - Ensuring the security and privacy of the data used in the ML models
8. **Sensor Deployers:**
    - Proper deployment and installation of the IoT devices
    - Ensuring the reliability and accuracy of the data collected
    - Security and privacy of the data collected and transmitted
9. **Security Engineers:**
    - Ensuring the security and privacy of the data collected and transmitted
    - Implementing security measures to prevent data breaches and cyber-attacks
    - Monitoring the system for potential security threats and taking appropriate actions.

Views and Viewpoints

1. **End Users:**
    - Data Privacy and Security Viewpoint
    - Usability and Accessibility Viewpoint
    - Information Accuracy and Reliability Viewpoint
2. **IoT Engineers:**
    - Compatibility Viewpoint
    - Power and Processing Requirements Viewpoint

- Data Security and Privacy Viewpoint

3. **Software Architect:**
   - Scalability Viewpoint
   - Integration Viewpoint
   - Data Security and Privacy Viewpoint
4. **Software Developer:**
   - Efficiency Viewpoint
   - Requirements Compliance Viewpoint
   - Software Reliability and Security Viewpoint
5. **Government Organization Officials:**
   - Traffic Management Real-time Data Accuracy Viewpoint
   - Traffic Management Integration Viewpoint
   - Urban Planning Integration and Scalability Viewpoint
6. **App Developer:**
   - User Interface and Experience Viewpoint
   - Integration with Architecture and IoT Devices Viewpoint
   - Data Security and Privacy Viewpoint
7. **ML Engineer:**
   - Data Preprocessing and Insights Accuracy Viewpoint
   - Prediction and Recommendation Systems Viewpoint
   - Data Security and Privacy Viewpoint
8. **Sensor Deployers:**
   - Deployment and Installation Viewpoint
   - Data Accuracy and Reliability Viewpoint
   - Data Security and Privacy Viewpoint
9. **Security Engineers:**
   - Data Security and Privacy Viewpoint
   - Cybersecurity Viewpoint
   - Security Monitoring Viewpoint

The possible views for above viewpoints are

1. **Data Privacy and Security Viewpoint:**
   - Encryption of sensitive data
   - Secure data storage
   - Access control measures
   - Compliance with relevant regulations and standards
2. **Usability and Accessibility Viewpoint:**
   - User-friendly interface design
   - Intuitive navigation and functionality

- Accessibility features for users with disabilities
- Multilingual support

3. **Information Accuracy and Reliability Viewpoint:**
    - Data validation and verification procedures
    - Real-time data updates
    - Data quality control measures
    - Reliable data sources
4. **Compatibility Viewpoint:**
    - Compatibility testing with different IoT devices and protocols
    - Compatibility with existing systems and infrastructure
    - Interoperability with other devices and services
5. **Power and Processing Requirements Viewpoint:**
    - Power consumption optimisation techniques
    - Efficient data processing algorithms
    - Effective resource management strategies
    - Low latency communication protocols
6. **Scalability Viewpoint:**
    - Horizontal and vertical scaling options
    - Load balancing mechanisms
    - Partitioning and sharding techniques
    - Distributed computing architectures
7. **Integration Viewpoint**:
    - API and middleware design
    - Protocol and data format standardisation
    - Interoperability with other software and systems
    - Integration testing and validation procedures
8. **Efficiency Viewpoint:**
    - Code optimisation and performance tuning
    - Effective memory management strategies
    - Use of efficient algorithms and data structures
    - Minimisation of resource usage and overhead
9. **Requirements Compliance Viewpoint:**
    - Adherence to functional and non-functional requirements
    - Compliance with industry standards and best practices
    - Traceability of requirements throughout the development process
    - Requirements validation and verification procedures
10. **Software Reliability and Security Viewpoint:**
    - Thorough testing and validation procedures

- Effective error handling and recovery mechanisms
- Security measures such as authentication, authorisation, and auditing
- Use of secure coding practices and vulnerability scanning tools

11. **Traffic Management Real-time Data Accuracy Viewpoint:**
    - Real-time data collection and analysis
    - Use of sensors and other IoT devices for data gathering
    - Integration with traffic modelling and simulation tools
    - Real-time traffic visualisation and monitoring capabilities
12. **Traffic Management Integration Viewpoint:**
    - Integration with traffic signal control systems
    - Integration with emergency response systems
    - Coordination with public transportation systems
    - Integration with other smart city initiatives
13. **Urban Planning Integration and Scalability Viewpoint:**
    - Integration with geographic information systems (GIS)
    - Scalable data storage and retrieval mechanisms
    - Integration with other urban planning tools and systems
    - Real-time urban planning visualisation and analysis capabilities
14. **User Interface and Experience Viewpoint:**
    - Use of responsive and aesthetically pleasing design
    - Consistent user interface and navigation across different platforms
    - Effective use of colour, typography, and other design elements
    - User testing and feedback collection mechanisms
15. **Data Preprocessing and Insights Accuracy Viewpoint:**
    - Data cleaning and transformation techniques
    - Use of appropriate data models and algorithms
    - Data visualisation and analysis tools
    - Accuracy validation and verification procedures
16. **Prediction and Recommendation Systems Viewpoint:**
    - Use of machine learning and artificial intelligence algorithms
    - Effective data preprocessing and feature selection techniques
    - Model training and validation procedures
    - Integration with other software systems and services
17. **Deployment and Installation Viewpoint:**
    - Effective deployment and installation procedures
    - Configuration management tools and techniques
    - Use of containerisation and virtualisation technologies
    - Deployment validation and verification procedures

18. **Data Accuracy and Reliability Viewpoint:**
    - Sensor Calibration and Maintenance View
    - Data Quality Assurance and Control View
    - Data Verification and Validation View
19. **Cybersecurity Viewpoint:**
    - Threat Assessment and Risk Analysis View
    - Incident Response and Recovery View
    - Access Control and Authorisation View
20. **Security Monitoring Viewpoint:**
    - System and Network Monitoring View
    - Vulnerability and Threat Detection View
    - Security Metrics and Reporting View

	End Users	**IoT Engineer**	**Software Architect**	**Government Organisation officials**	**App Developer**	**ML Engineer**	**Sensor Deployers**
Data privacy of users	x		x	x	x	x	
Ease of Use of app	x		x		x		
Compatibility of IOT Devices		x	x			x	
Security and privacy of data transmitted		x	x			x	x
Scalability of Architecture		x	x			x	
Reliability Of Software			x		x	x	
Realtime data accuracy		x				x	x
Preprocessing is accurate						x	
Accuracy of prediction and recommendation						x	
Reliability and accuracy of data collected		x		x			x

Task 3: Architecture Tactics and Patterns

The architectural tactics that we plan to use in implementation are:

- **Microservices Architecture**: A microservices architecture can help with the scalability and maintainability of the system. By breaking down the system into small, loosely coupled services, it becomes easier to update and scale individual components without affecting the entire system.

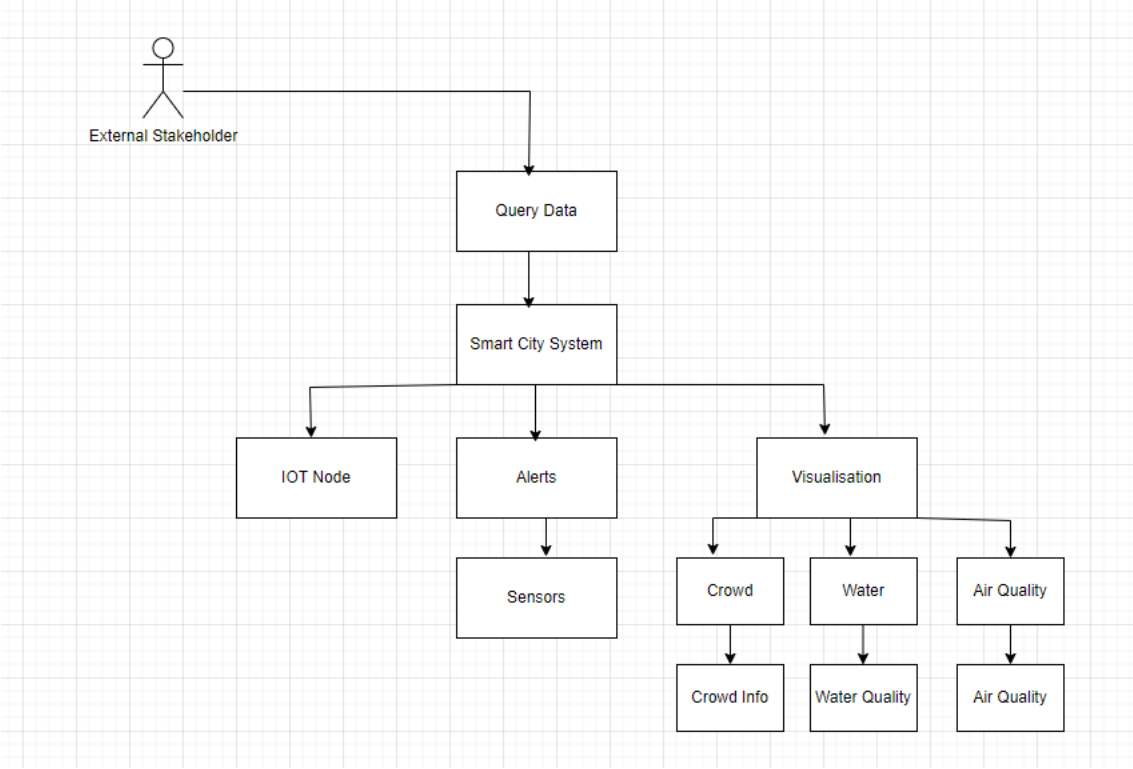
- **Edge Computing**: Edge computing involves processing data closer to the source of the data, instead of sending it to a centralised cloud. By processing data at the edge, it can reduce latency, improve data security and privacy, and reduce network bandwidth requirements. This approach can also help with the continuous monitoring of crowd information in real-time by enabling data processing closer to the sensors
- **Caching**: Caching can help with the performance of the system by reducing the number of requests to the database or other data sources. By storing frequently accessed data in a cache, it becomes quicker to access the data and reduces the load on the database. This approach can be useful for handling the continuous inflow of data from different IoT nodes and supporting near real-time visualisations.
- **Data Encryption and Authentication**: To ensure data privacy and security, data encryption and authentication can be used. Data encryption can be used to protect sensitive data, while authentication can ensure that only authorised users have access to the system. These measures can help with ensuring that the system does not store any sensitive data related to users and that not every stakeholder has access to all functionalities/data

The architectural patterns that we plan to use are:

- **Microservices Architecture**: A microservices architecture can help with the scalability and maintainability of the system. By breaking down the system into small, loosely coupled services, it becomes easier to update and scale individual components without affecting the entire system. This approach can also help with fault tolerance, as the failure of one service does not bring down the entire system. It also helps teams to work independently.
- **Event-driven Architecture:** This pattern would help in handling the continuous inflow of data from different IoT nodes, and enable near real-time processing and visualization of data. The event-driven architecture would allow different modules of the system to communicate with each other asynchronously using events, enabling faster and more efficient processing of data. It also allows for loose coupling between components.
- **API Gateway Pattern:** This pattern would help in providing easy means for external stakeholders to leverage the data generated by the Smart City IoT system. An API Gateway can be implemented to provide a unified interface for external stakeholders to access the system's functionalities and data. The API Gateway can also enforce access control and provide rate limiting to ensure that not every stakeholder has access to all types of functionalities/data.
- **Circuit Breaker Pattern**: This pattern would help in handling situations where the system experiences high traffic or overload, and prevent the system from crashing or becoming unresponsive. Circuit Breakers can be implemented to monitor the health of different components of the system and temporarily halt requests to a failing component until it becomes healthy again. This would ensure the system's availability and prevent cascading failures.
- **Gateway Pattern:** This pattern would enable the system to handle different IoT communication protocols used by the different IoT nodes. A gateway component can be implemented for each communication protocol to enable translation of the messages to a common format, which can then be consumed by other modules of the system.
- **Publisher Subscriber Pattern:** The publisher-subscriber pattern is particularly useful in IoT systems because it allows for scalable and efficient communication between devices and services, as well as ensuring data consistency and reliability. Additionally, this pattern enables the devices to operate autonomously and with a high degree of flexibility, as they can publish data at any time, and subscribers can receive that data in near real-time.

Architecture Diagrams

Use Case Diagram



Note: **C4 model diagrams are in the end of this document**

Task 4 : Architecture Analysis

Analysed **Scheduler** and D**eployer** implementations with **Publish and Subscribe** Pattern with focus on **Latency** and **scalability. (Please see attached code files).**

If we don't use publish and subscribe pattern in messaging between microservices then we need to use normal request and reply pattern which has a lot of disadvantages.

Disadvantages of Request and Reply pattern:

1. **Tight Coupling:** Request-reply communication requires services to be tightly coupled. If a service needs to communicate with another service, it must know the specific endpoint or location of that service. This creates a dependency on the location of the service and makes it difficult to scale the system.
2. **Limited Scalability**: The request-reply pattern is not very scalable. As the number of services and requests increases, the system can become overwhelmed and slow down or even crash.

Scheduler.py :

- Acts as a Kafka producer(Publisher) for topic "scheduler_to_deployer" and Kafka consumer(Subscriber) for "scheduling_apps".
- It Subscribes to "scheduling_apps" and gets the schedule info.
- After getting the scheduling info we used python library called "Schedule" which will perform scheduling day-wise.
- when an app got scheduled a "start" message is sent to deployer using kafka topic "scheduler_to_deployer".
- When the scheduled time is over then a "stop" message is sent to deployer using the same kafka topic

Producer.py :

- Acts as a Kafka producer
- Has info. about scheduling hard coded which on executing is sent to “scheduling_apps” topic

Deployer.py :

- When it receives data from Scheduler.py, it deploys the application.

Analysis:

1. **Throughput:** The maximum throughput of Kafka depends on various factors such as the number of brokers, network bandwidth, message size, etc. Assuming a typical message size of 1KB and a network bandwidth of 1 Gbps, Kafka can achieve a maximum throughput of around 80,000 messages per second with 3 brokers. However, it's important to note that the actual throughput may vary based on the workload and other factors.
2. **Latency:** The end-to-end latency of Kafka depends on various factors such as the network latency, message size, serialization/deserialization time, processing time, etc. Assuming a typical message size of 1KB and a network latency of 10 ms, the end-to-end latency of Kafka can be around 20-30 ms for a simple workload with a single consumer group. However, the latency can increase significantly for complex workloads with multiple consumer groups and processing stages.

Trade-offs: There are some trade-offs associated with using Kafka as the messaging protocol in an IoT system.

1. **Scalability vs. Complexity:** Kafka provides high scalability and fault-tolerance, but it also introduces additional complexity in terms of managing topics, partitions, consumer groups, etc. This can increase the development and operational costs of the system.
2. **Throughput vs. Latency:** Kafka provides high throughput, but it also introduces some additional latency due to serialization/deserialization, network latency, etc. This can affect the end-to-end latency of the system and may not be suitable for some real-time applications where low latency is critical.

	Publish-Subscribe Pattern	Request-Reply Pattern
Response Time	In this pattern, the publisher sends messages to the broker, which then distributes the messages to all subscribers. This adds an additional hop in the communication process, which can increase the response time. However, the asynchronous nature of the pattern allows subscribers to consume messages at their own pace, which can improve the overall system performance.	In this pattern, the client sends a request to the server, which then sends a response back to the client. This results in a direct and synchronous communication between the two parties, which can reduce the response time. However, the client needs to wait for the response before proceeding, which can result in slower overall system performance if the server takes a long time to respond.
Throughput	In this pattern, the broker can handle multiple publishers and subscribers, allowing for high throughput. Additionally, since subscribers can consume messages at their own pace, the overall system can handle large volumes of messages without becoming overwhelmed.	In this pattern, the server can handle multiple requests and responses, allowing for high throughput. However, since the communication is synchronous, the server needs to process each request before sending a response, which can limit the overall throughput.

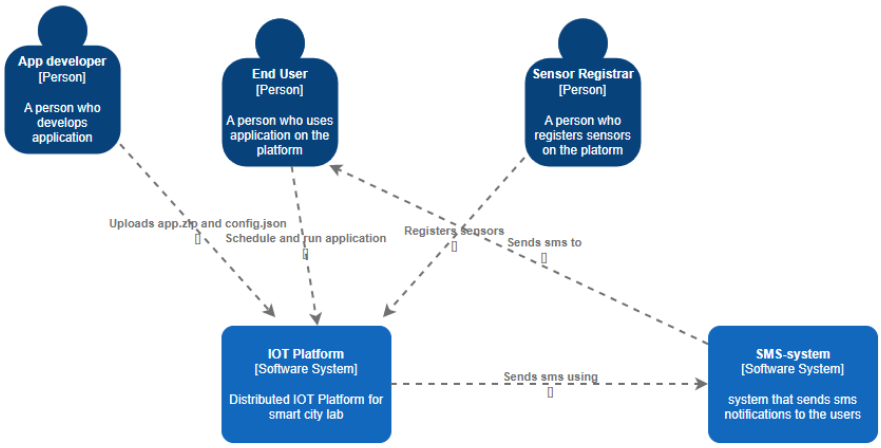
Video links for task 4: https://iiitaphyd-my.sharepoint.com/:v:/g/personal/dheeraj_nemalikanti_students_iiit_ac_in/ESrP24Vcs2dIrvPO_mcUq3wBqOHU7Uemflb274jyame=npBBQs

Summary

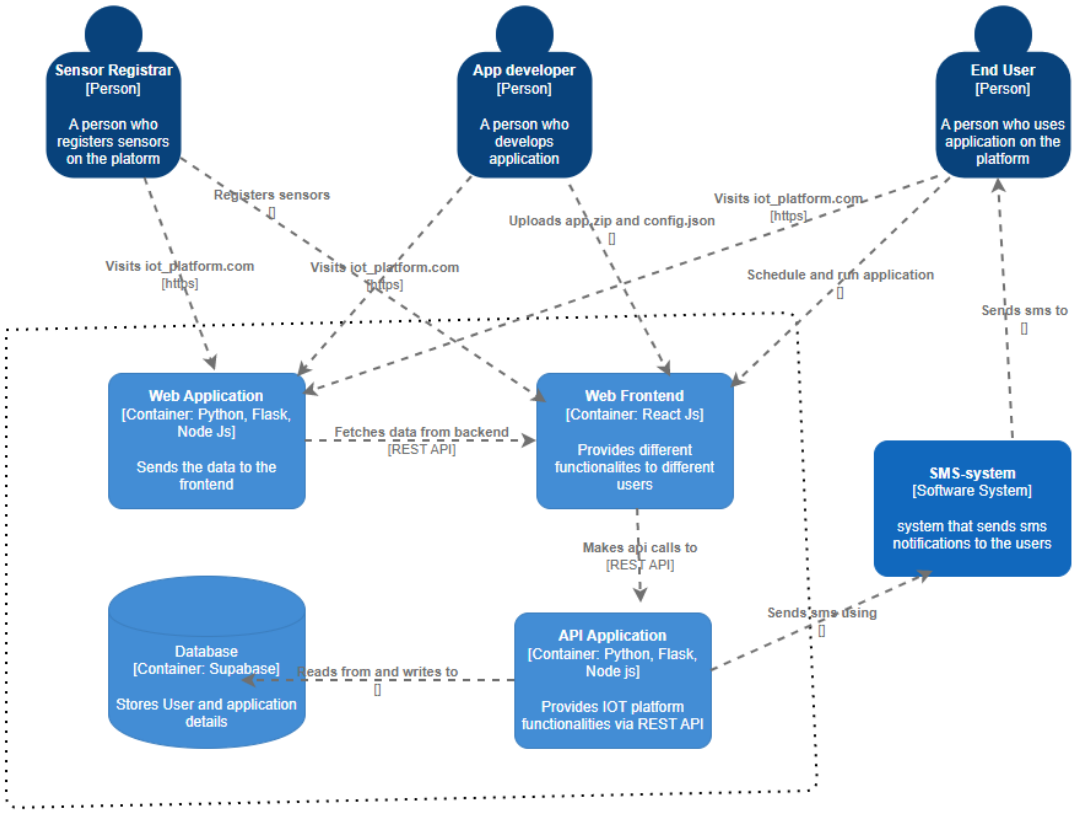
We have designed the architecture for Smart City system. We gathered the functional, non functional and architecturally significant requirements. We listed all the subsystems and drew the architecture diagram of the whole system. Then we identified the stakeholders and also identified their views and viewpoints.

Based on the requirements gathered we also mentioned some of the architectural tactics and patterns that we would like to incorporate in our Smart City System. After this we performed architectural analysis. We did analysis for reliability and scalability .We also drew the use case diagram for the system.

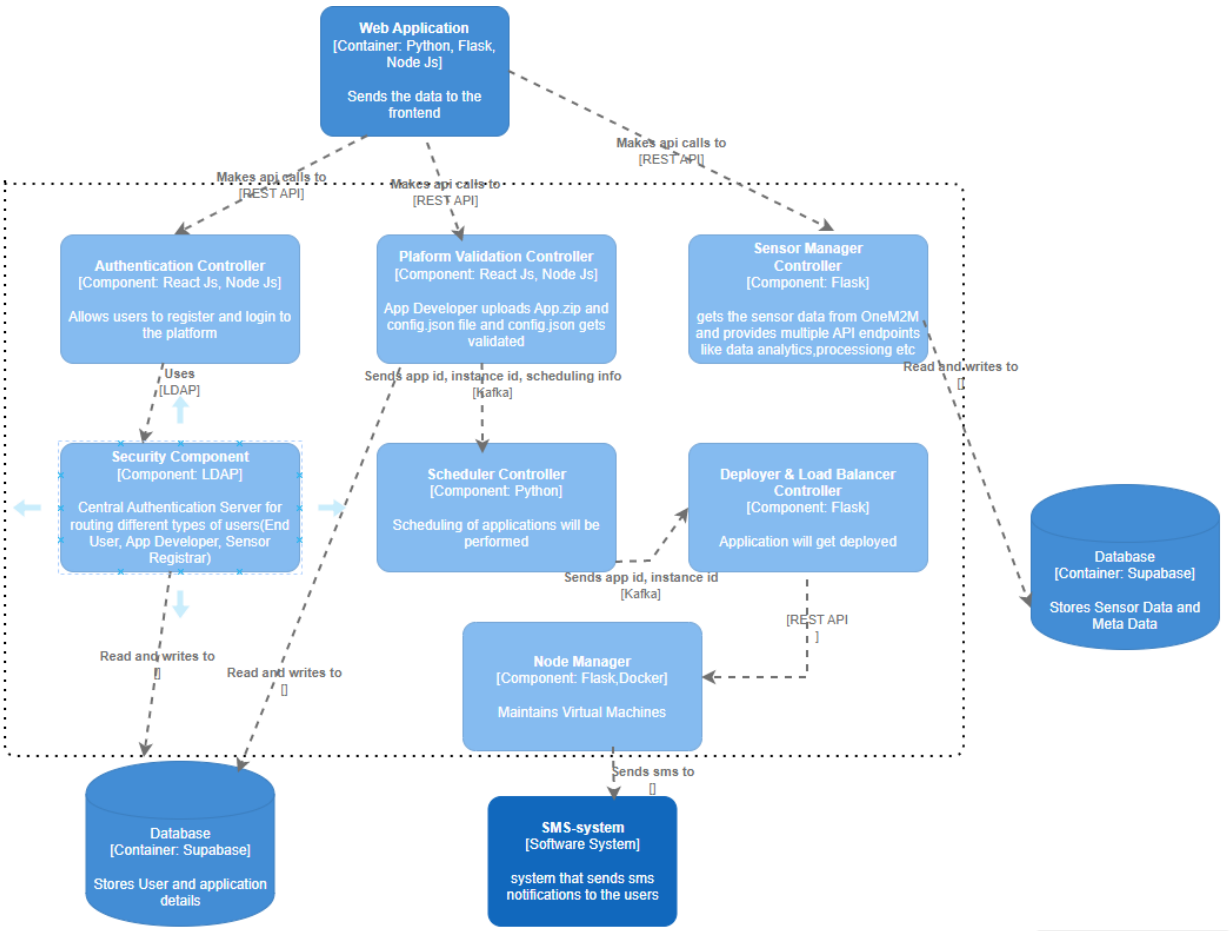
Context Diagram



Container Diagram



Component Diagram


Web Application
[Container: Python, Flask, Node Js]
Sends the data to the frontend
Makes api calls to
[REST API]
Makes api calls to
[REST API]
Makes api calls to
[REST API]
Authentication Controller
[Component: React Js, Node Js]
Allows users to register and login to the platform
Plaform Validation Controller
[Component: React Js, Node Js]
App Developer uploads App.zip and config.json file and config.json gets validated
Sensor Manager Controller
[Component: Flask]
gets the sensor data from OneM2M and provides multiple API endpoints like data analytics,processiong etc
Uses
[LDAP]
Sends app id, instance id, scheduling info
[kafka]
Read and writes to
[]
Security Component
[Component: LDAP]
Central Authentication Server for routing different types of users(End User, App Developer, Sensor Registrar)
Scheduler Controller
[Component: Python]
Scheduling of applications will be performed
Deployer & Load Balancer Controller
[Component: Flask]
Application will get deployed
Database
[Container: Supabase]
Stores Sensor Data and Meta Data
Sends app id, instance id
[Kafka]
[REST API
]
Read and writes to
[]
Read and writes to
[]
Node Manager
[Component: Flask,Docker]
Maintains Virtual Machines
Sends sms to
[]
Database
[Container: Supabase]
Stores User and application details
SMS-system
[Software System]
system that sends sms notifications to the users