Smart city management

TEAM - 37

# SMART CITY MANAGEMENT

ABSTRACT :

Smart City Management is like the brain behind making cities work better. It's about using technology, data, and input from citizens to improve how cities run, making them more efficient and nicer to live in.

In this system, we have different parts:

- Citizens: These are the people who live in the city and use its services.

- Services: These are the things the city provides to its people, like water, transportation, and garbage collection.

- Infrastructure: This is all the physical stuff in the city, like roads, buildings, and utilities.

- IoT Devices: These are special gadgets with sensors and internet connection that help gather data about the city.

- Incidents: These are unexpected things that happen, like accidents or power outages.

- Maintenance: This is all the work done to keep the city's stuff in good shape, like fixing broken roads or maintaining buildings.

- Permits: These are official permissions given by the city for certain activities or projects.

- Complaints: These are the problems citizens report, like potholes or noisy neighbors.

- Analytics: This is the process of looking at all the data collected to find useful information and make decisions.

- Dashboard: This is like a control panel that shows important information about the city's operations in an easy-to-understand way.

Together, these parts help make cities smarter and better places to live by using technology and citizen input to solve problems and make things run more smoothly.

INTRODUCTION :

Smart city management refers to the strategic and integrated approach taken by city authorities to leverage technology, data, and citizen engagement to improve the efficiency, sustainability, and livability of urban areas. It involves the use of various digital solutions and innovative technologies to address the complex challenges faced by modern cities and enhance the quality of life for residents.

At the core of smart city management is the integration of information and communication technologies (ICT) with physical infrastructure and public services. This integration allows cities to collect, analyse, and utilise vast amounts of data from sensors, devices, and other sources to make informed decisions and optimise resource allocation

**Summary of entities and its attributes:**

**Citizen**: Citizens are essential stakeholders in the smart city ecosystem, as they interact with various city services, infrastructure, and systems on a daily basis.

**Attributes:** id (primary key), name , address , contact name ,Email

**Service:** The Service entity represents the various public services offered by the city to its residents and businesses

**Attributes:** id (primary key), name , description

**Infrastructure:** The Infrastructure entity represents the physical assets, facilities, and systems that form the backbone of the city's built environment and support the delivery of essential services and functions to its residents and businesses.

**Attributes :** id (primary key), name , description

**IoT Device** : The IoT Device entity represents the physical devices equipped with sensors, actuators, and connectivity capabilities that enable them to collect, transmit, and receive data from the surrounding environment in real-time

**Attributes :** Id (primary key), name , description

**Incident** : The Incident entity represents any unexpected event, issue, or occurrence that requires attention, intervention, or resolution from city authorities or service providers.

**Attributes :** id (primary key), date , description , location , citizen id(foreign key)

**Maintenance** : The Maintenance entity represents planned or reactive activities undertaken to ensure the proper functioning, safety, and reliability of city infrastructure and assets.

**Attributes** : id (primary key), date , description , infrastructure id(fr)

**Permit** **:**The Permit entity represents authorization granted by city authorities for specific activities, projects, or events within the city limits.

**Attributes**:id (primary key), date\_issues , date\_expiry , description , citizen\_id(fr)

**Complaint** **:**The Complaint entity represents concerns, issues, or grievances reported by citizens regarding various aspects of city services, infrastructure, or quality of life.

**Attributes :** id (primary key), date , location , description , citizen\_id(fr**)**

**Analytics** **:**The Analytics entity represents the process of analyzing and deriving insights from data collected within the smart city ecosystem to inform decision-making, optimize resource allocation, and improve urban management strategies.

**Attributes :** id (primary key), description

**Dashboard** **:**The Dashboard entity represents a visual interface or platform that provides stakeholders, including city officials, policymakers, and citizens, with real-time or near-real-time insights into various aspects of city operations, services, and performance metrics

**Attributes :** id (primary key), description

**Summary of entity relationship:**

• Citizen to Incident: (One-to-one)

While it's possible for a Citizen to report multiple incidents, it's also conceivable that a specific incident might be directly tied to one Citizen. For instance, in the case of a personal injury or property damage incident, there might be a direct one-to-one relationship between a Citizen and that particular Incident.

• Citizen to Permit:( One-to-Many )A citizen can have multiple permits

• Citizen to Complaint: (One-to-Many )A citizen can make multiple complaints)

• Incident to Maintenance: (One-to-Many )An incident may require multiple maintenance actions

• Service to Infrastructure: (Many-to-Many )A service may involve multiple infrastructures, and an infrastructure may be associated with multiple services

• Infrastructure to IoT\_Device: (Many-to-Many )An infrastructure may have multiple IoT devices, and an IoT device may be associated with multiple infrastructures.

ER DIAGRAM :

A diagram of a network

Description automatically generated

FUNCTIONAL REQUIREMENT :

Functional requirements in Unified Modeling Language (UML) refer to the specifications or descriptions of the system's functions or capabilities. UML provides various diagrams to represent these requirements visually. Here are some common ways functional requirements are depicted in UML:

• **Use Case Diagrams**: Use case diagrams in UML represent the functional requirements of a system from the user's perspective. They show the interactions between actors (users or external systems) and the system itself. Each use case represents a specific functionality or action that the system performs in response to an actor's request.

• **Activity Diagrams**: Activity diagrams in UML depict the flow of activities or actions within a system. They can be used to model the workflow of a particular use case or business process, showing the sequence of actions, decision points, and branching logic.

• **Sequence Diagrams**: Sequence diagrams illustrate how objects interact in a particular scenario or use case over time. They show the sequence of messages exchanged between objects and the order of execution of these messages. Sequence diagrams can be used to model the behavior of a system in response to user interactions or external events.

• **State Machine Diagrams**: State machine diagrams represent the behavior of a system or object in response to internal or external events. They show the states that an object can be in and the transitions between these states based on events or conditions. State machine diagrams are particularly useful for modeling the lifecycle of objects or the behavior of complex systems with multiple states.

• **Class Diagrams**: While class diagrams primarily depict the static structure of a system, they can also capture some aspects of functional requirements. By showing the relationships between classes, attributes, and methods, class diagrams can help clarify how different parts of the system collaborate to fulfill specific functionalities.

These are some of the main ways UML can be used to represent functional requirements. Each type of diagram provides a different perspective on the system's behavior and functionality, allowing stakeholders to better understand and communicate the requirements.

|  |
| --- |
| Service |
| Attributes:  - id: int  - Name: string  - description: string |
| Operations:  + getId(): int  + setId(id: int): void  + getName(): string  + setName(Name: string): void  + getDescription(): string  + setDescription(description: string): void |

UML/CLASS DIAGRAM :

|  |
| --- |
| Name: Citizen |
| Attribute:  - Id : String  - Name : String  - Address : String  - Contact Number :String  - Email address :String |
| Operations :  + getId(): int  + setId(id: int): void  + getName(): string  + setName(name: string): void  + getAddress(): string  + setAddress(address: string): void  + getContactNumber(): string  + setContactNumber(contact Number: string): void |

|  |
| --- |
| Infrastructure |
| Attributes:  - id: int  - Name: string  - description: string |
| Operations:  + getId(): int  + setId(id: int): void  + getName(): string  + setName(Name: string): void  + getDescription(): string  + setDescription(description: string): void |

|  |
| --- |
| Iot Device |
| Attributes:  - id: int  - Name: string  - description: string |
| Operations:  + getId(): int  + setId(id: int): void  + getName(): string  + setName(Name: string): void  + getDescription(): string  + setDescription(description: string): void |

|  |
| --- |
| Incident |
| Attributes:  - id: int  - date: int  - description: string  - Location: String  - citizen id : int |
| Operations:  + getId(): int  + getDate(): int  + getDescription(): string  + getLocation(): string  + getCitizenId(): int  + setId(id: int): void  + setDate(date: int): void  + setDescription(description: string): void  + setLocation(location: string): void  + setCitizenId(citizenId: int): void |

|  |
| --- |
| Maintenance |
| Attributes:  - id: int  - date: int  - infrastructureId: int  - description: string |
| + getId(): int  + setId(id: int): void  + getInfrastructureId(): int  + setInfrastructureId(infrastructureId: int): void  + getDescription(): string  + setDescription(description: string): void  +getDate() : int  + setDate(date : int) : void |

|  |
| --- |
| Permit |
| Attributes:  - id: int  - date\_issues: int  - date\_expiry: int  - description: string  - citizen\_id: int |
| Operations:  + getDateIssues(): int  + getDateExpiry(): int  + getDescription(): string  + getCitizenId(): int  + setId(id: int): void  + setDateIssues(date\_issues: int): void  + setDateExpiry(date\_expiry: int): void  + setDescription(description: string): void  + setCitizenId(citizen\_id: int): void  + toString(): string |

|  |
| --- |
| Complaint |
| Attributes:  - id: int  - date: int  - description: string  - Location: String  - citizen id : int |
| Operations:  + getId(): int  + getDate(): int  + getDescription(): string  + getLocation(): string  + getCitizenId(): int  + setId(id: int): void  + setDate(date: int): void  + setDescription(description: string): void  + setLocation(location: string): void  + setCitizenId(citizenId: int): void |

|  |
| --- |
| Analytics |
| Attributes:  - id: int  - description: string |
| + getId(): int  + setId(id: int): void  + getDescription(): string  + setDescription(description: string): void |

|  |
| --- |
| Dashboard |
| Attributes:  - id: int  - description: string |
| + getId(): int  + setId(id: int): void  + getDescription(): string  + setDescription(description: string): void |

CHALLENGES LIST :

It can take time to grasp exactly what the project needs to do. We need to have a clear idea about the project.

Figuring out what parts (like objects, classes, or tables) should be in our diagrams and code can be tough. we want to make sure to get everything important.

It's important to ensure that our diagrams and code match each other. Changes in one should be reflected in the other.

Sometimes while inserting the record in the tables can get complicated, and it's hard to make sense of them. and we need to break them down into smaller, easier-to-understand parts so that

without having any idea about mysql workbench it is difficult to create and insert the record

While inserting /creating the record we were in a rush to enter the record due to this we spent a lot of time and had to learn some syntax also.

Figuring out how different parts of the system relate to each other can be tricky. Use diagrams to help make it clear. (mainly in entity relations)

At the end of the Er-diagram, we didn't cross-check the relations more than once due to this it took a little extra time.

While writing the java code we faced few difficulties like syntax errors..

And also while uploading the files into the git it was somewhat confused but although we learnt and managed to complete them.

At the end of the project, we realized that we needed to make a list of things and have an idea about the project before implementing it in.