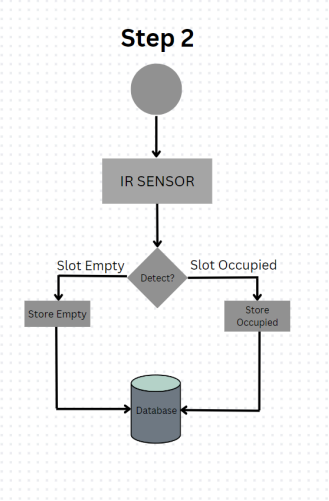
SMART LIGHTING SYSTEM

Step 1: Purpose & Requirements Specification

- Design a system to prevent wastage of electrical power by automating lights.
- Develop a smart lighting system that adjusts based on ambient light using an ESP32, LDR sensor, and LED.
- Monitor light intensity, trigger the LED when light levels drop, and update status to the Blynk cloud.
- Ensure real-time data collection, cloud integration, user-friendly interface, and secure data management.

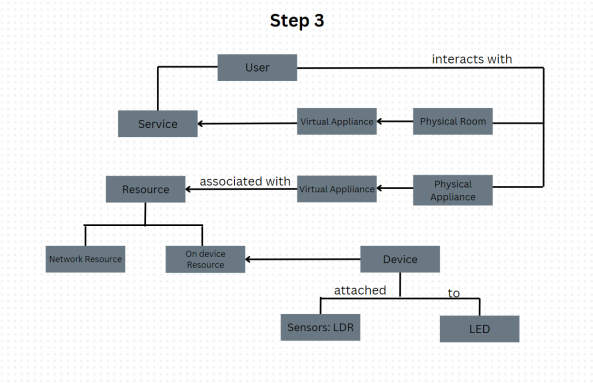
Step 2: Process Specification

- LDR sensor detects low light levels, triggering the ESP32 to turn on the LED.
- System continuously monitors light intensity and updates the LED state accordingly.
- Real-time data is sent to the Blynk cloud for user monitoring and control via the dashboard.



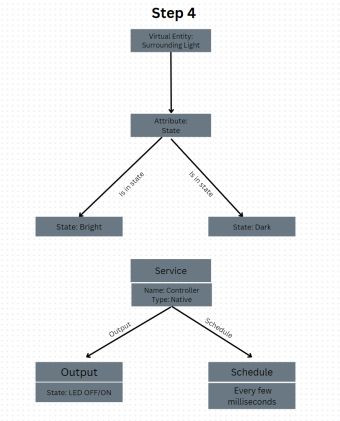
Step 3: Domain Model Specification

- Define key entities: ESP32, LDR sensor, LED, and Blynk cloud platform.
- Describe relationships between entities, such as how LDR readings affect the LED.
- Provide an abstract representation of these entities, independent of specific technology.



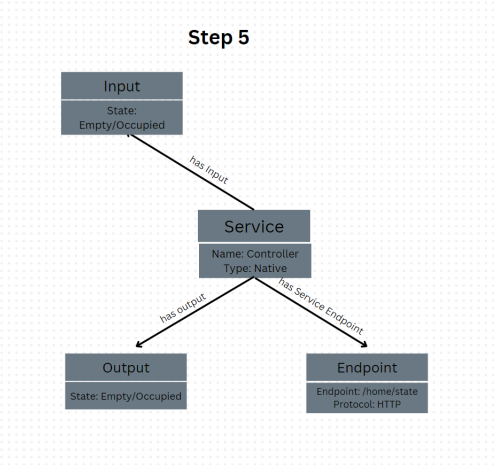
Step 4: Information Model Specification

- Structure the information within the system, detailing attributes of virtual entities like the LDR, LED, and cloud data points.
- Define how information flows and is managed within the system.
- Ensure the model adds detail to the domain model by specifying entity attributes and relationships.



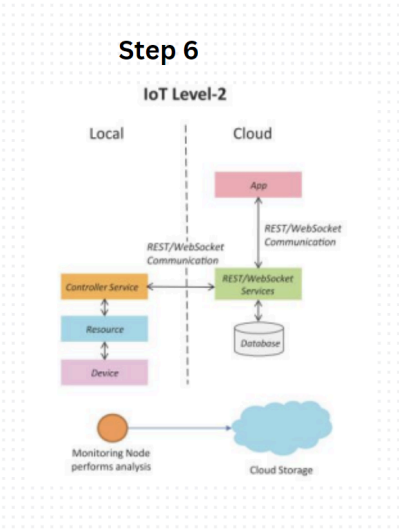
Step 5: Service Specifications

- Define key services: light detection, LED control, data logging, and cloud synchronization.
- Detail service inputs (LDR readings), outputs (LED state), and endpoints (Blynk API).



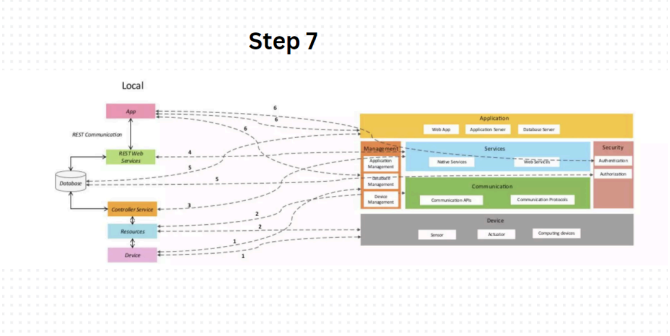
Step 6: IoT Level Specification

- Define the IoT deployment level, combining local processing with cloud-based management.
- Architecture using ESP32 for local processing and Blynk cloud for remote monitoring.
- Ensure the system operates efficiently at an intermediate IoT level.



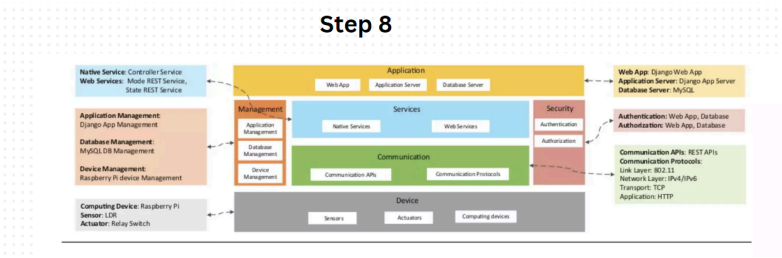
Step 7: Functional View Specification

- Group system functionalities into functional groups: data acquisition, processing, action, and communication.
- Ensure interaction between functional groups and domain model entities, like LDR readings influencing LED control.
- Provide a clear outline of how each functional group supports the system's overall operation.



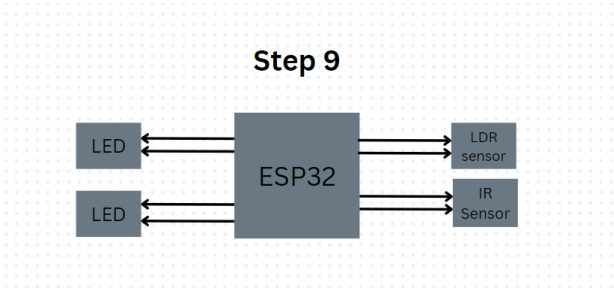
Step 8: Operational View Specification

- Define deployment options: hosting Blynk on a cloud server and securing data storage.
- Manage device configurations, including ESP32 setup and LDR/LED integration.
- Optimize system operation to ensure efficient and reliable performance.



Step 9: Device & Component Integration

- Integrate physical components: ESP32, LDR sensor, and LED, with proper configuration.
- Program the ESP32 to process LDR data and control the LED based on light thresholds.
- Connect the system to the Blynk cloud for remote access and control.



Step 10: Application Development

- Develop a Blynk app for real-time monitoring and manual control of the LED.
- Display LDR sensor data and provide user alerts based on specific conditions.
- Ensure the app offers a user-friendly interface, enabling easy interaction with the smart light system.

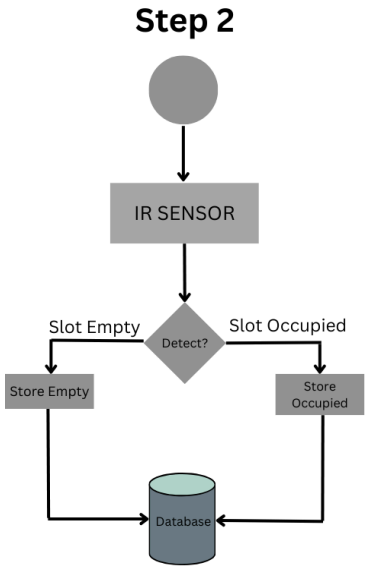
SMART PARKING SYSTEM

Step 1: Purpose & Requirements Specification

- Design a system to optimize parking space utilization by automating the detection of available parking spots.
- Develop a smart parking system using two IR sensors and an ESP32 that detects the occupancy of parking spots and updates the status to Firebase.
- Monitor parking spot occupancy in real-time, update Firebase accordingly, and provide a user-friendly interface for monitoring.
- Ensure real-time data collection, cloud integration with Firebase, and secure data management.

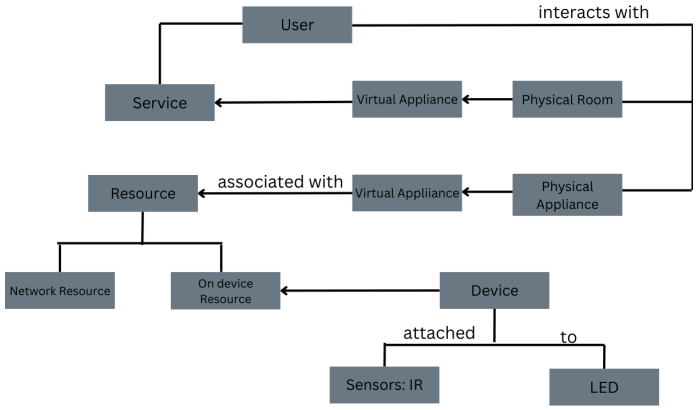
Step 2: Process Specification

- IR sensors detect the presence or absence of vehicles in designated parking spots.
- When a vehicle is detected, the ESP32 updates the occupancy status to Firebase.
- The system continuously monitors the occupancy status and updates Firebase in real-time, allowing users to view available parking spots through a web interface or app.



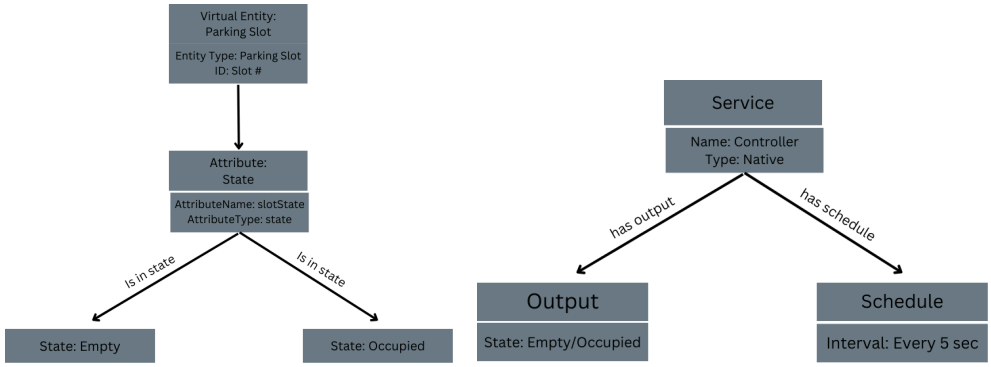
Step 3: Domain Model Specification

- Define key entities: ESP32, IR sensors, parking spots, and Firebase cloud platform.
- Describe relationships between entities, such as how IR sensor readings update the parking spot status in Firebase.
- Provide an abstract representation of these entities, independent of specific technology.



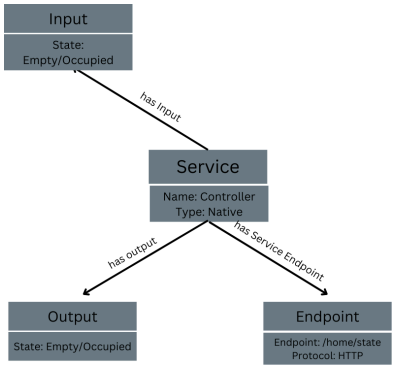
Step 4: Information Model Specification

- Structure the information within the system, detailing attributes of virtual entities like parking spots, IR sensors, and cloud data points.
- Define how information flows and is managed within the system, ensuring each parking spot's status is accurately reflected in Firebase.
- Ensure the model adds detail to the domain model by specifying entity attributes and relationships.



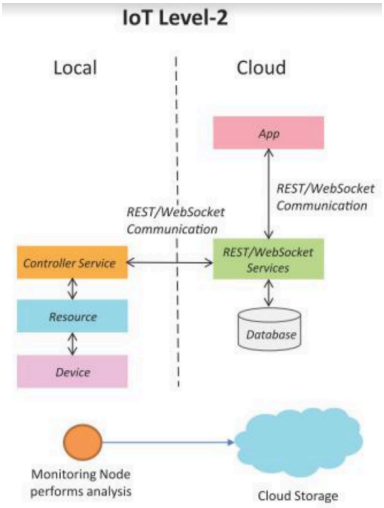
Step 5: Service Specifications

- Define key services: vehicle detection, parking spot status update, data logging, and Firebase synchronization.
- Detail service inputs (IR sensor readings), outputs (parking spot occupancy status), and endpoints (Firebase database).



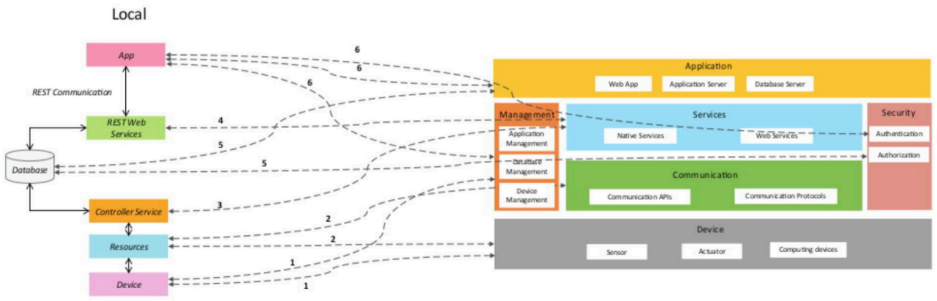
Step 6: IoT Level Specification

- Define the IoT deployment level, combining local processing with cloud-based management via Firebase.
- Architecture using ESP32 for local processing and Firebase cloud for remote monitoring and data storage.
- Ensure the system operates efficiently at an intermediate IoT level.



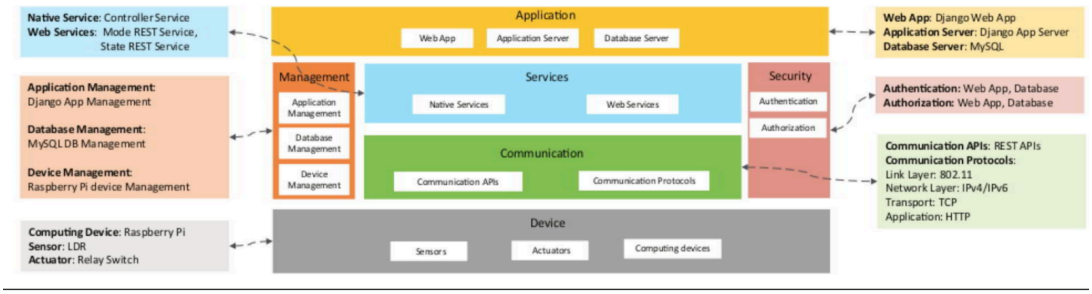
Step 7: Functional View Specification

- Group system functionalities into functional groups: data acquisition, processing, action, and communication.
- Ensure interaction between functional groups and domain model entities, like IR sensor readings influencing parking spot status updates.
- Provide a clear outline of how each functional group supports the system's overall operation.



Step 8: Operational View Specification

- Define deployment options: hosting the Firebase backend and securing data storage.
- Manage device configurations, including ESP32 setup and IR sensor integration.
- Optimize system operation to ensure efficient and reliable performance.



Step 9: Device & Component Integration

- Integrate physical components: ESP32, IR sensors, with proper configuration.
- Program the ESP32 to process IR sensor data and update parking spot status in Firebase based on vehicle detection.
- Connect the system to Firebase for remote access and control of parking spot availability.

Step 10: Application Development

- Develop a web or mobile application for real-time monitoring and manual control of parking spots.
- Display IR sensor data and provide user alerts when parking spots become available or occupied.
- Ensure the app offers a user-friendly interface, enabling easy interaction with the smart parking system.

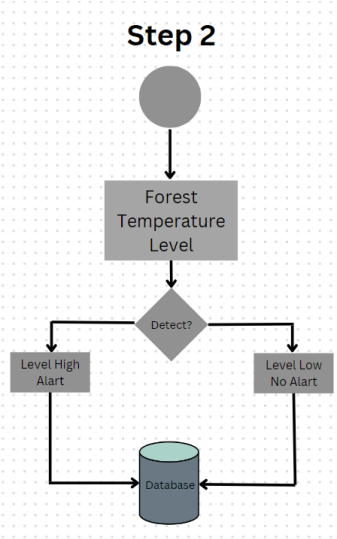
Forest Fire Detection

Step 1: Purpose & Requirement Specification

- The IoT-based system, built using ESP32, continuously monitors environmental parameters such as temperature, humidity, and smoke levels in forest areas.
- The ESP32 facilitates real-time data transmission to a central server or cloud platform via Wi-Fi or other IoT communication protocols. When sensor readings cross predefined thresholds, the system automatically sends alerts to authorities, ensuring quick response actions.
- Designed for remote forest areas, the system utilizes ESP32's low power consumption and supports solar power or battery backup. This ensures long-term, low-maintenance operation, making it a practical and scalable solution for large-scale forest fire detection.

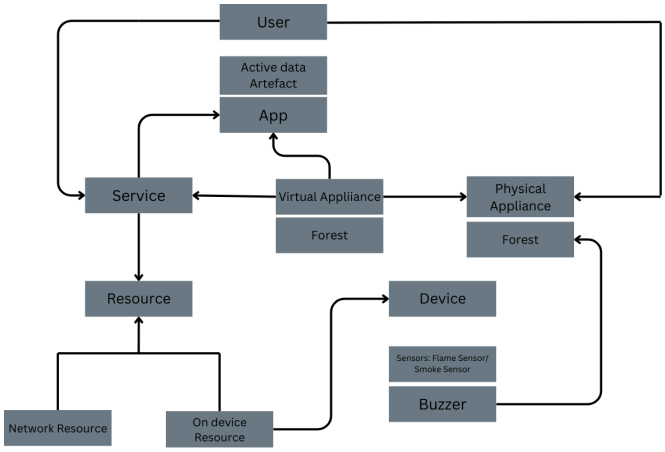
Step 2: Process Specification

- Fire detection sensors monitor temperature, humidity, and smoke levels, sending data to the ESP32 for processing.
- The ESP32 compares sensor data against thresholds to identify potential fires.
- Upon detecting a fire, the ESP32 triggers local alerts and sends notifications to authorities via Wi-Fi or IoT methods.



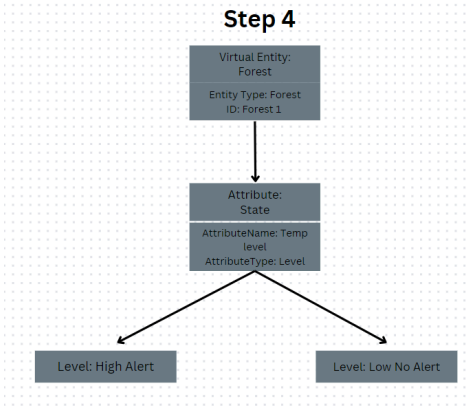
Step 3: Domain Model Specification

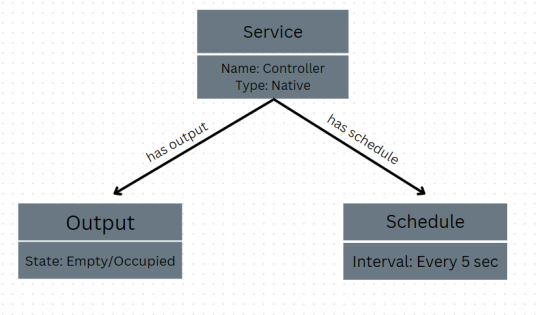
- Collect data on environmental conditions and transmit this data to the ESP32 for processing.
- The ESP32 processes the sensor data and uses the Blynk platform to communicate real-time data and alerts to the cloud or user's mobile device via the Blynk app.
- The Blynk app provides a user-friendly interface for monitoring the system, displaying sensor readings, and receiving fire alerts, enabling users to take immediate action.



Step 4: Information Model Specification

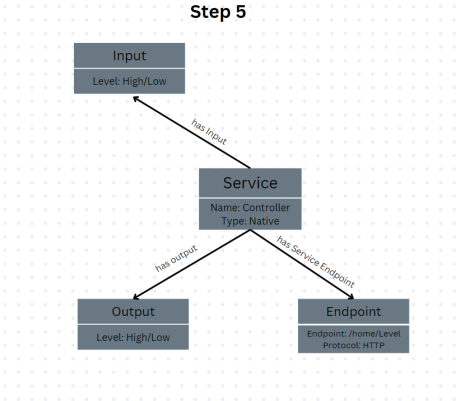
- Define the format for environmental data (temperature, humidity, smoke levels) collected by sensors, including units of measurement and data ranges.
- Specify the threshold values for each sensor parameter that trigger alerts, including the criteria for distinguishing between normal and fire-related conditions.





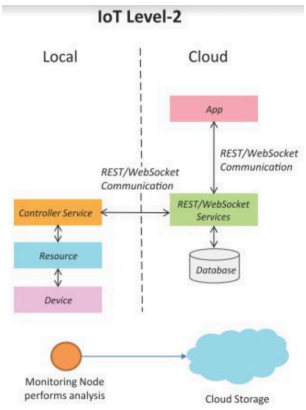
Step 5: Service Specifications

- The ESP32 processes sensor data, applies thresholds, and transmits results to the Blynk platform.
- Blynk sends real-time alerts and notifications to users and servers, including visual and audio cues.



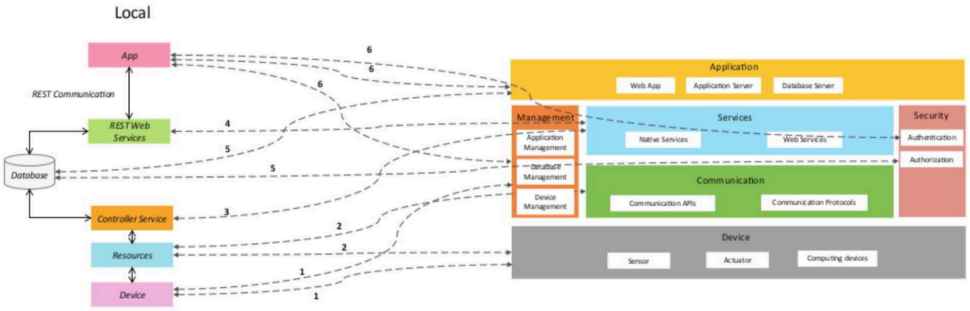
Step 6: IoT Level Specification

- The ESP32 connects with fire detection sensors to collect and process environmental data.
- Data is transmitted from the ESP32 to the Blynk platform via Wi-Fi or other IoT communication protocols.
- The Blynk app provides a user interface for monitoring sensor data and receiving alerts in real-time.·



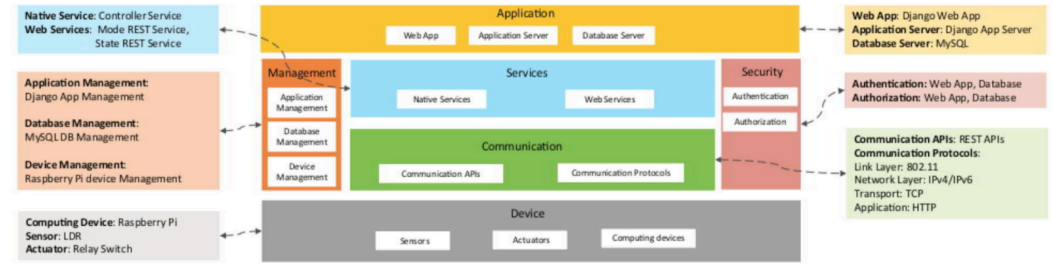
Step 7: Functional View Specification

- Sensors collect environmental data and send it to the ESP32 for analysis.
- The ESP32 processes the sensor data, checks for threshold breaches, and determines if a fire is present.
- The ESP32 triggers local alerts and communicates notifications to the Blynk app for user awareness and response.



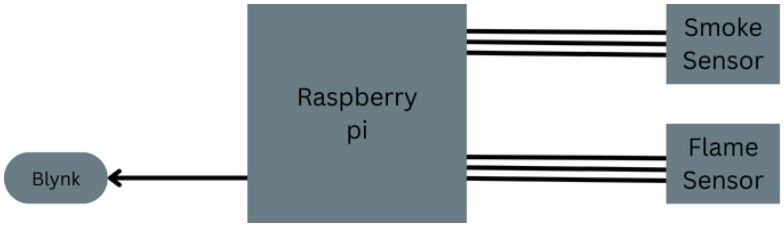
Step 8: Operational View Specification

- Sensors continuously monitor environmental conditions and transmit data to the ESP32.
- The ESP32 processes incoming data, performs threshold checks, and manages alert generation.
- The ESP32 sends processed data and alerts to the Blynk app, ensuring real-time notifications to users and authorities.



Step 9: Device & Component Integration

- Connect temperature, humidity, and smoke sensors to the ESP32 for data collection.
- Configure the ESP32 to communicate with the Blynk platform for data transmission and alert management.
- Connect alert sensors or notification modules to the ESP32 to trigger local alarms and send alerts to the Blynk app.



Step 10: Application Development

- Write and deploy firmware on the ESP32 to handle sensor data processing and communication.
- Develop and configure the Blynk app interface to display sensor data and receive alerts.
- Implement logic for triggering and managing alerts based on sensor data within the ESP32 and Blynk app.

Intrusion Detection System

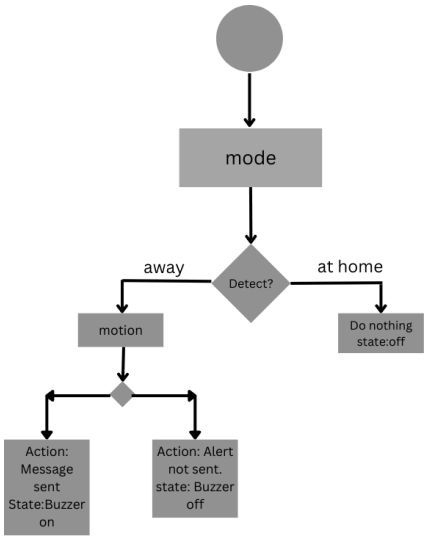
Step 1: Purpose & Requirements Specification

- Design an intrusion detection system that allows you to monitor the surroundings in a home to detect intruders.
- The system should have away and at home modes. In away mode, the system is fully functional and monitors the room for presence of intruders. The at home mode system is idle.
- Remote should provide remote monitoring and control functions.
- Apps should be available remotely and the system should perform local analysis of detection level.

Step 2: Process Specification

Process Flow:

- The PIR sensor continuously monitors for motion within its detection range.
- When motion is detected, the ESP32 triggers the buzzer to alert the presence of an intruder.
- The system sends real-time data and alerts to the Blynk cloud for user monitoring.
- Users receive immediate notifications on their mobile devices through the Blynk app.

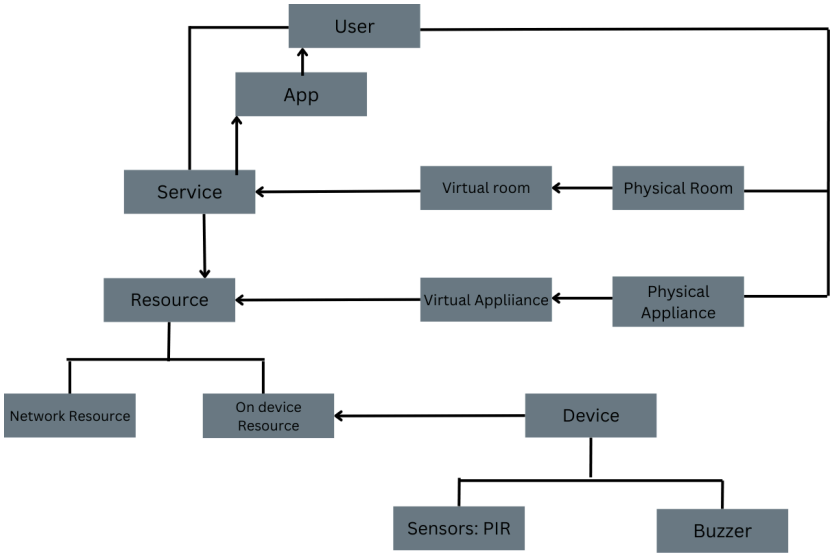


Step 3: Domain Model Specification

Entities:

- **ESP32:** Central controller that processes sensor data and controls the buzzer.
- **PIR Sensor:** Detects motion, signaling potential intrusions.

- **Buzzer:** Alerts users by sounding an alarm when motion is detected.
- **Blynk Cloud Platform:** Provides real-time monitoring and control, sending notifications to users.

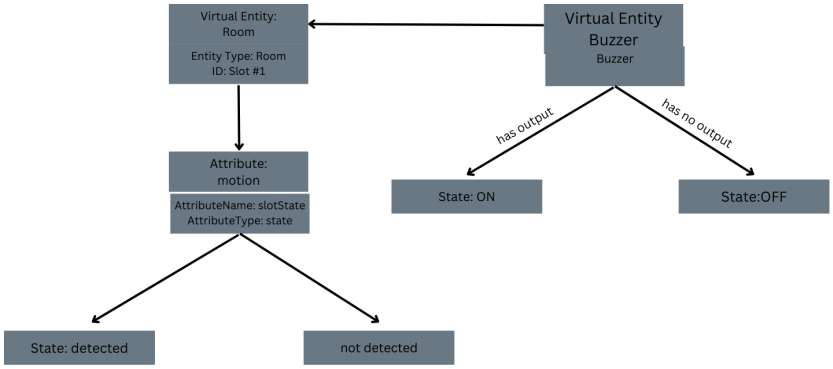


Step 4: Information Model Specification

- PIR Sensor Data: Captures and processes motion detection data.
- Buzzer State: On or off, based on the sensor's input.
- Cloud Data Points: Intrusion alerts, sensor readings, and system status are updated to the Blynk cloud.

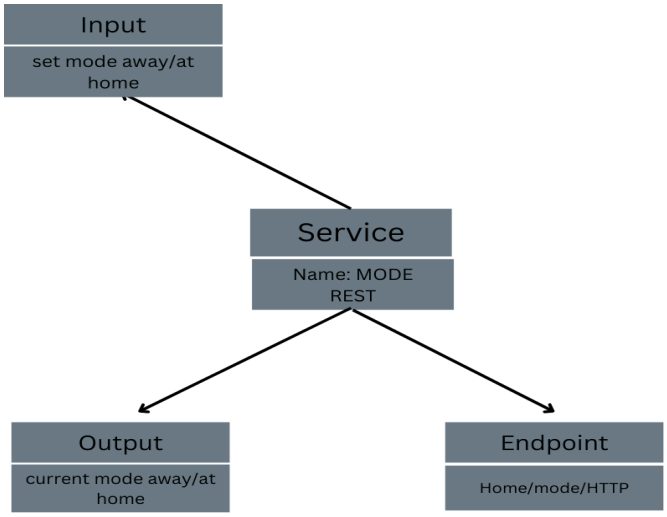
Information Flow:

- Sensor data is processed locally by the ESP32, triggering the buzzer and updating the cloud.
- The system ensures consistent and accurate data flow between the hardware components and the Blynk cloud.



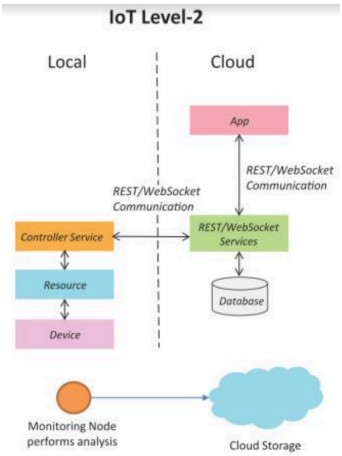
Step 5: Service Specifications
Key Services:

- Motion Detection: Monitors PIR sensor readings and triggers alerts.
- Alert Notification: Controls the buzzer and sends intrusion alerts to the Blynk cloud.
- Data Logging: Logs intrusion events and sensor data in the cloud for user review.
- Cloud Synchronization: Updates the Blynk platform with real-time data for monitoring and alerting.



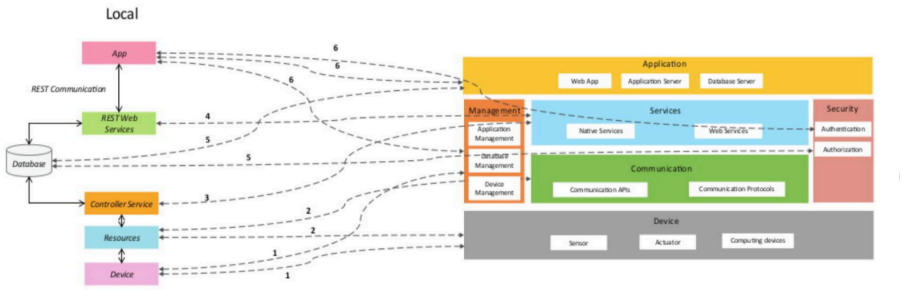
Step 6: IoT Level Specification
IoT Level:

- Combine local processing on the ESP32 with cloud-based monitoring via Blynk.
- Operate at an intermediate IoT level, ensuring efficient and responsive intrusion detection with remote alert capabilities.



Step 7: Functional View Specification
Functional Groups:

- Data Acquisition: Captures motion data from the PIR sensor.
- Processing: Analyzes sensor data to detect intrusions.
- Action: Triggers the buzzer and sends alerts when an intrusion is detected.
- Communication: Syncs data with the Blynk cloud and sends notifications to users.



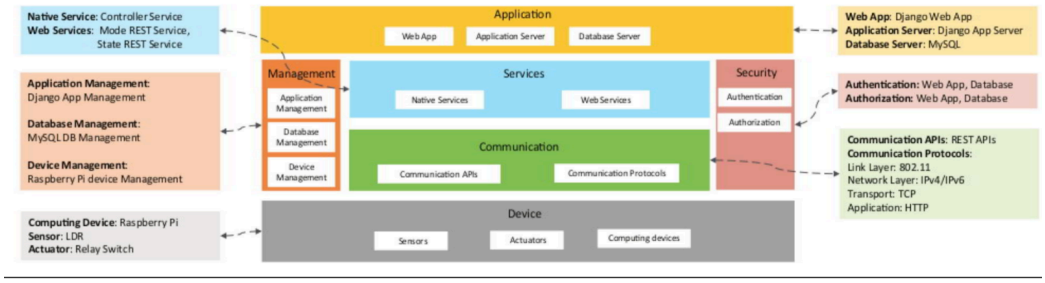
Step 8: Operational View Specification

Deployment Options:

- Host the Blynk app on a cloud server for easy access and monitoring.
- Securely store data, ensuring user privacy and data integrity.

Device Management:

- Configure the ESP32, PIR sensor, and buzzer.
- Ensure reliable communication between the ESP32 and the Blynk cloud for real-time data synchronization.



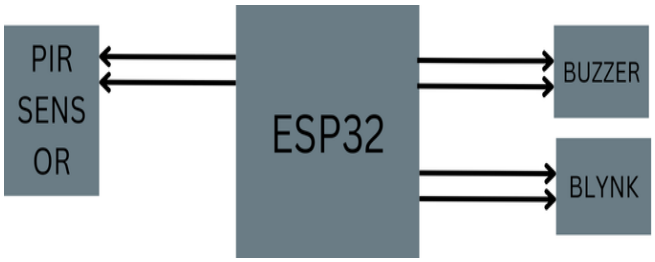
Step 9: Device & Component Integration

Physical Integration:

- Integrate the ESP32, PIR sensor, and buzzer with appropriate wiring and configuration.
- Program the ESP32 to process motion detection data and control the buzzer.

Cloud Integration:

- Connect the system to the Blynk cloud, allowing users to monitor and control the system remotely.



Step 10: Application Development

- Blynk App Development:
    - Create a Blynk app interface for real-time monitoring of intrusion detection.
    - Display motion detection data and system status, providing user alerts and controls.
    - Ensure the app is user-friendly, enabling easy interaction with the intrusion detection system and immediate responses to alerts.

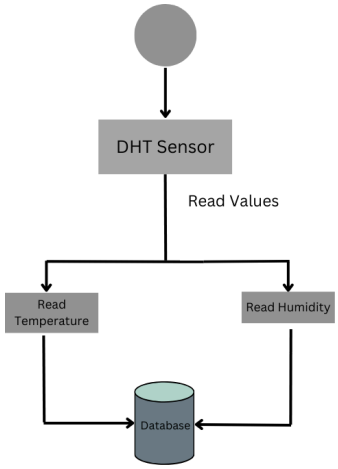
Weather Reporting

Step 1: Purpose & Requirements Specification:

Design a weather reporting app that allows users to monitor real-time temperature and humidity data from a remote location. The system should capture data using a DHT sensor connected to an ESP32 and store it in Firebase. The mobile app built using MIT App Inventor should display the current temperature and humidity and send notifications if the values exceed a specific threshold.

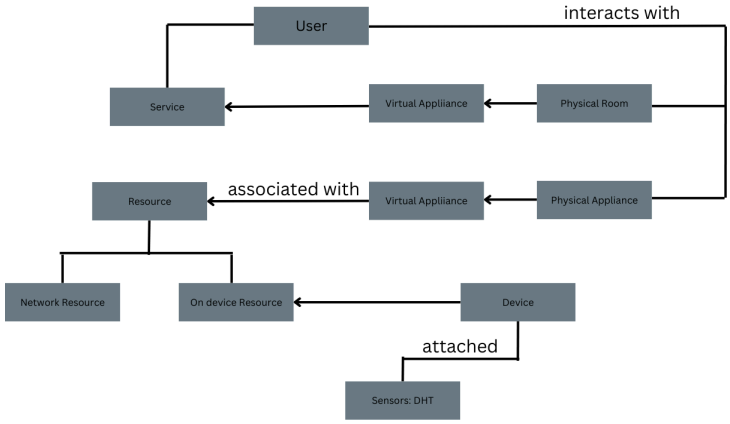
Step 2: Process Specification
Process Flow:

- The DHT sensor continuously measures temperature and humidity.
- The ESP32 reads the sensor data and uploads it to Firebase in real-time.
- The MIT App Inventor app retrieves the data from Firebase and displays it on the user's device.
- Notifications are sent via the app if temperature or humidity goes beyond predefined thresholds.



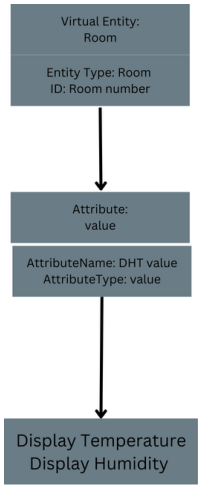
Step 3: Domain Model Specification
Entities:

- ESP32: Microcontroller that interfaces with the DHT sensor and manages data communication with Firebase.
- DHT Sensor: Measures temperature and humidity levels.
- Firebase Realtime Database: Stores temperature and humidity data and facilitates real-time data syncing.
- MIT App Inventor App: Provides a user interface for monitoring weather data and receiving alerts.



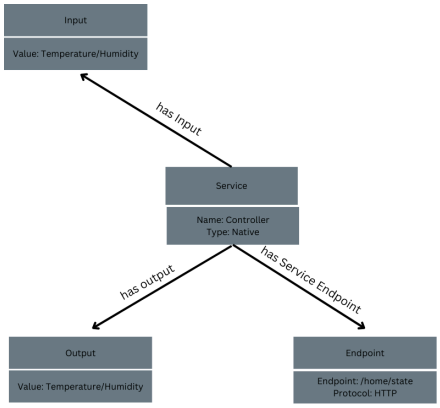
Step 4: Information Model Specification
Information Elements:

- Temperature Data: Captured and processed by the DHT sensor.
- Humidity Data: Captured and processed by the DHT sensor.
- Threshold Alerts: Notifications triggered when sensor readings exceed specific values.
- Data Flow: Sensor data is collected by the ESP32, sent to Firebase, and displayed in the mobile app.



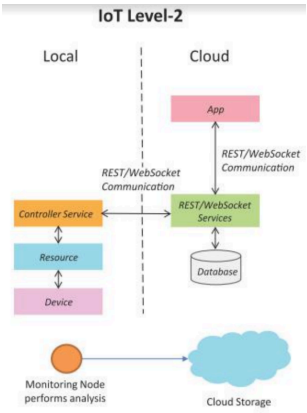
Step 5: Service Specifications
Key Services:

- Data Collection: Continuously captures temperature and humidity data from the DHT sensor.
- Alert Notification: Sends alerts to users when temperature or humidity thresholds are exceeded.
- Real-Time Monitoring: Displays current weather data in the app using real-time updates from Firebase.
- Historical Data Logging: Logs past sensor data for trend analysis and review.



Step 6: IoT Level Specification

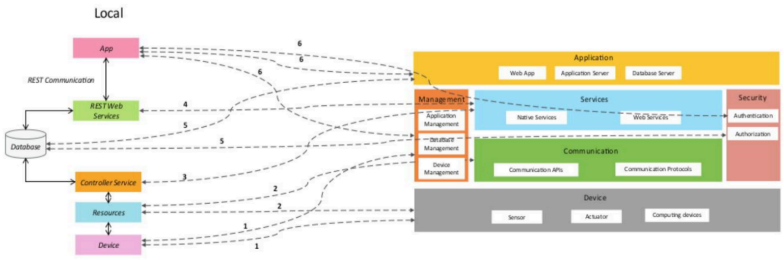
IoT Level: Operate at an intermediate IoT level, combining local data collection on the ESP32 with cloud-based data storage and real-time app monitoring via Firebase.



Step 7: Functional View Specification

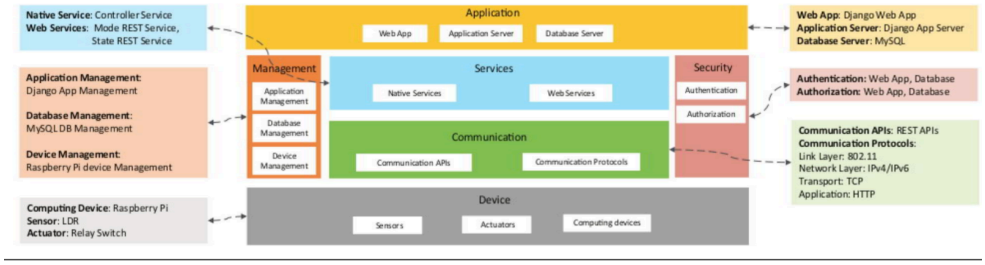
Functional Groups:

- Data Acquisition: Collects temperature and humidity data from the DHT sensor.
- Processing: ESP32 processes the data and determines if thresholds are breached.
- Communication: Data is uploaded to Firebase and synchronized with the mobile app.
- User Interaction: The app interface allows users to monitor real-time data and receive notifications.



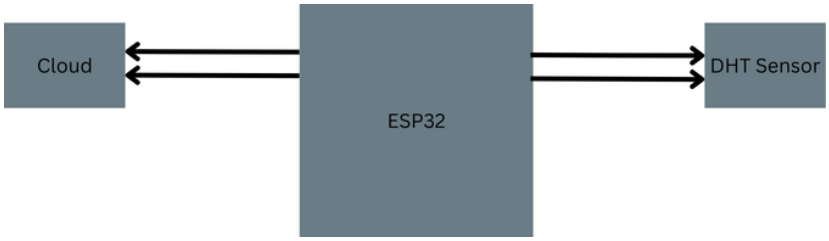
Step 8: Operational View Specification
Deployment Options:

- Deploy the app via MIT App Inventor, making it accessible on Android devices.
- Store sensor data securely in Firebase, ensuring real-time updates and reliable performance. Device Management:
- Configure the ESP32 to read and upload sensor data.
- Ensure robust communication between the ESP32, Firebase, and the mobile app.



Step 9: Device & Component Integration
Physical Integration:

- Connect the DHT sensor to the ESP32.
- Program the ESP32 to read data from the DHT sensor and send it to Firebase. Cloud Integration:
- Link the ESP32 with Firebase, ensuring seamless data flow.
- Configure Firebase to work with the MIT App Inventor app for real-time data display.



Step 10: Application Development
MIT App Inventor Development:

- Design an intuitive interface in MIT App Inventor for monitoring weather data.
- Display real-time temperature and humidity on the app's main screen.
- Implement notifications for threshold breaches and ensure the app is responsive and easy to use.

Smart Irrigation System

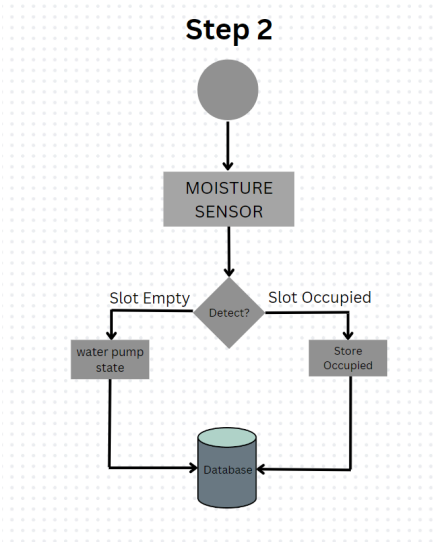
Step 1: Purpose & Requirements Specification:

The purpose of the Smart Irrigation System is to automate the watering of plants based on real-time soil moisture levels, thereby optimizing water usage and ensuring that plants receive adequate hydration. The system aims to reduce water wastage and the manual effort required to maintain a garden or agricultural field. The system will leverage IoT technology to monitor soil moisture levels and control the irrigation process.

The Smart Irrigation System aims to automate plant watering by monitoring real-time soil moisture levels, optimizing water use, and reducing manual effort. The system includes a soil moisture sensor, an ESP32 or Arduino microcontroller, a water pump, and a relay module, all powered by a reliable power source. The mobile app, built using MIT App Inventor, allows users to monitor moisture levels, control the pump, set thresholds, and receive notifications. Data is stored in Firebase for real-time access and historical analysis, with user authentication ensuring secure access. The system operates reliably in various environmental conditions, is scalable for larger areas, and can integrate with weather data to adjust irrigation schedules. Usability, security, and power efficiency are key non-functional requirements, ensuring the system is user-friendly and sustainable.

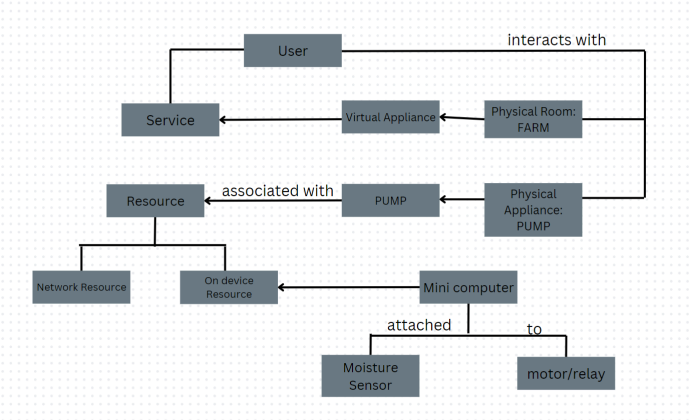
Step 2: Process Specification
Process Flow:

- The soil moisture sensor continuously monitors the moisture levels in the soil.
- The ESP32 reads the moisture data from the sensor.
- If the moisture level falls below a predefined threshold, the ESP32 triggers the water pump via a relay module.
- The ESP32 uploads the real-time soil moisture data and pump status to Firebase.
- The MIT App Inventor app retrieves the moisture data and pump status from Firebase.
- The app displays the current soil moisture level and irrigation status on the user's device.
- Users can manually control the water pump from the app if needed.
- The app sends notifications to the user if the moisture level drops too low or exceeds the threshold.
- Historical data on moisture levels and irrigation times is logged in Firebase for user access.
- The system operates automatically, but users can adjust moisture thresholds and settings through the app.



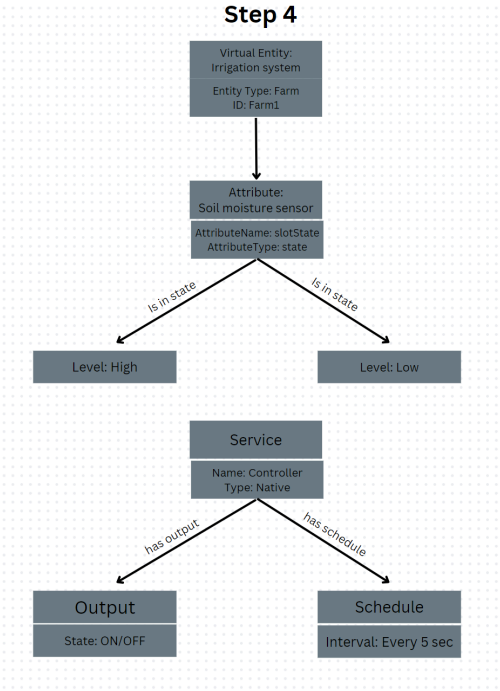
Step 3: Domain Model Specification
Entities:

- **ESP32/Arduino:** Microcontroller that interfaces with the soil moisture sensor, controls the water pump, and manages data communication with Firebase.
- **Soil Moisture Sensor:** Measures the moisture content in the soil, providing real-time data to the ESP32/Arduino.
- **Relay Module:** Acts as an intermediary between the ESP32/Arduino and the water pump, enabling the pump to be activated or deactivated based on soil moisture levels.
- **Water Pump:** Delivers water to the plants, controlled automatically by the relay module or manually through the app.
- **Firebase Realtime Database:** Stores soil moisture data, irrigation status, and user-defined thresholds, facilitating real-time data syncing between the ESP32/Arduino and the mobile app.
- **MIT App Inventor App:** Provides a user interface for monitoring soil moisture levels, controlling the water pump, setting thresholds, and receiving alerts and notifications.



Step 4: Information Model Specification
Information Elements:

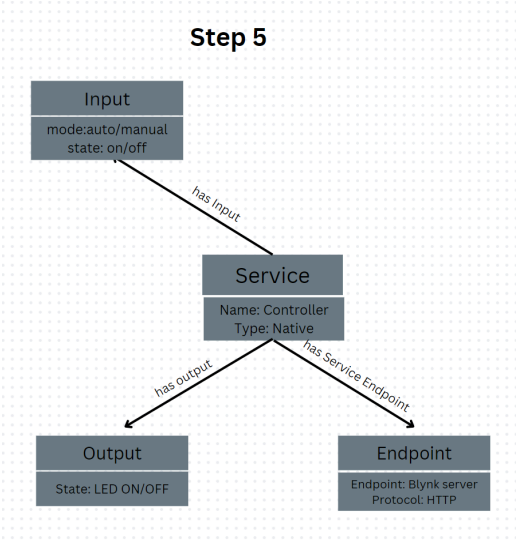
- **Soil Moisture Data:** Captured by the soil moisture sensor and processed by the ESP32/Arduino to determine the moisture level in the soil.
- **Irrigation Status:** Indicates whether the water pump is currently active or inactive, based on soil moisture readings.
- **Threshold Alerts:** Notifications triggered when soil moisture levels fall below or exceed predefined thresholds, prompting irrigation actions.
- **Manual Control Commands:** User-initiated commands sent from the mobile app to the ESP32/Arduino to manually start or stop the water pump.
- **Data Flow:** Soil moisture data is collected by the ESP32/Arduino, sent to Firebase for storage and real-time updates, and displayed in the mobile app for user monitoring and control.



Step 5: Service Specifications
Key Services:

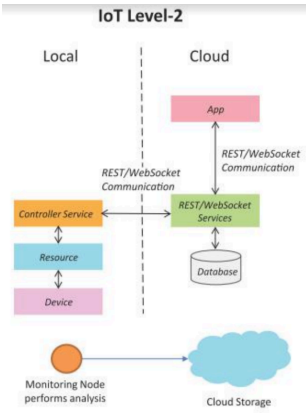
- **Data Collection:** Continuously captures soil moisture data from the soil moisture sensor, providing real-time input to the ESP32/Arduino.
- **Automated Irrigation Control:** Automatically triggers the water pump based on soil moisture levels, ensuring plants are watered when needed.
- **Alert Notification:** Sends alerts to users when soil moisture levels drop below or exceed predefined thresholds, allowing for timely intervention.

- **Real-Time Monitoring:** Displays current soil moisture levels and irrigation status in the mobile app using real-time updates from Firebase.
- **Manual Control:** Allows users to manually start or stop the water pump through the mobile app, overriding automatic controls if necessary.
- **Threshold Management:** Enables users to set and adjust soil moisture thresholds in the mobile app, customizing the irrigation system's behavior.
- **Historical Data Logging:** Logs past soil moisture data, irrigation events, and user interactions for trend analysis and review.
- **Data Syncing:** Ensures seamless synchronization of data between the ESP32/Arduino, Firebase, and the mobile app, maintaining consistency across the system.



Step 6: IoT Level Specification

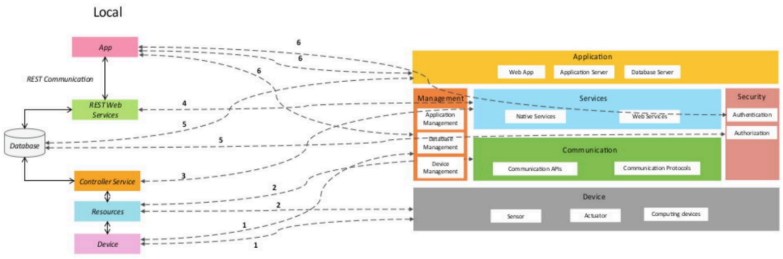
IoT Level: Operate at an intermediate IoT level, combining local data collection on the ESP32 with cloud-based data storage and real-time app monitoring via Firebase.



Step 7: Functional View Specification

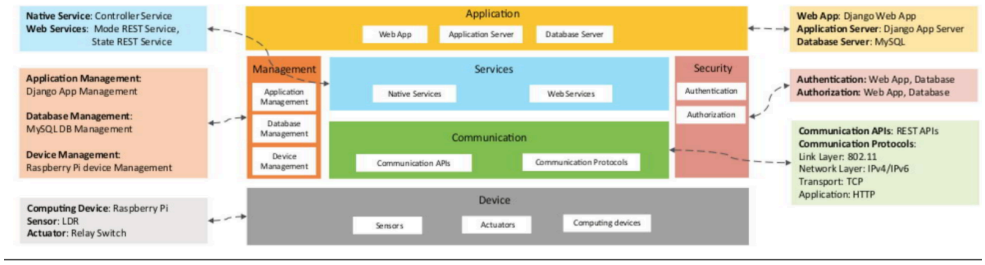
Functional Groups:

- **Data Acquisition:** Collects soil moisture data from the soil moisture sensor, providing essential input for irrigation decisions.
- **Processing:** The ESP32/Arduino processes the soil moisture data, determines if the moisture level is below or above the set thresholds, and decides whether to activate the water pump.
- **Control Mechanism:** Manages the activation and deactivation of the water pump based on soil moisture data and user-defined thresholds.
- **Communication:** Handles the upload of real-time soil moisture data and pump status to Firebase and ensures synchronization with the mobile app for consistent data access.
- **User Interaction:** The mobile app interface allows users to monitor real-time soil moisture levels, control the water pump, set moisture thresholds, and receive alerts and notifications.
- **Data Logging and Review:** Logs and stores historical soil moisture data, irrigation events, and user commands for future review and analysis.



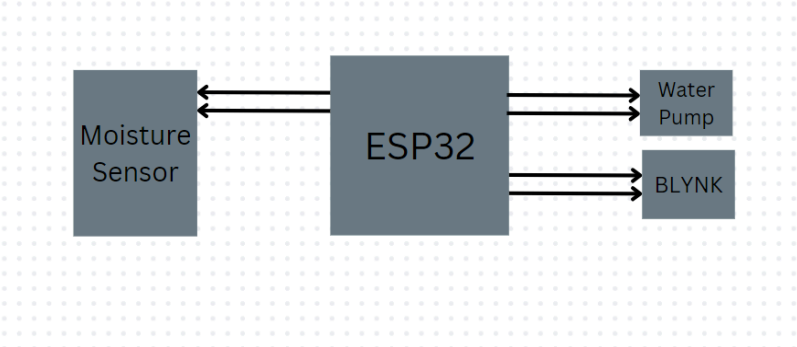
Step 8: Operational View Specification
Deployment Options:

- **Mobile App Deployment:** Deploy the app through MIT App Inventor, making it easily accessible on Android devices for real-time monitoring and control of the irrigation system.
- **Cloud Storage:** Store sensor data securely in Firebase, providing real-time updates, reliable performance, and easy scalability for managing multiple devices or large-scale irrigation systems.
- **Device Management:** Configure the ESP32 to read soil moisture data from the sensor, process it, and upload the data to Firebase efficiently.
- **Communication Robustness:** Ensure stable and secure communication between the ESP32, Firebase, and the mobile app, maintaining real-time data synchronization and system reliability.
- **Power Management:** Optimize the power usage of the ESP32 and connected components to ensure long-term operation, especially in remote or outdoor environments.



Step 9: Device & Component Integration
Physical Integration:

- **Sensor Connection:** Connect the soil moisture sensor to the ESP32/Arduino, ensuring proper wiring and stable signal transmission for accurate moisture level readings.
- **Pump Control Integration:** Connect the water pump to the ESP32/Arduino via a relay module, enabling the microcontroller to automate the pump's operation based on soil moisture data.
- **Microcontroller Programming:** Program the ESP32/Arduino to read data from the soil moisture sensor, process it, and control the water pump as per the set thresholds.
- **Power Supply Setup:** Ensure all components, including the ESP32/Arduino, sensor, relay module, and water pump, are powered correctly for consistent and reliable operation.



Step 10: Application Development
MIT App Inventor Development:

- Design an intuitive interface in MIT App Inventor for monitoring weather data.
- Display real-time temperature and humidity on the app's main screen.
- Implement notifications for threshold breaches and ensure the app is responsive and easy to use.