

**Deakin University**

Project Title: Smart Home Lighting System

Project Proposal

Student Name: Hayes Duong

Student ID: 222610226

Date of Submission: July 19, 2025

Document Version 1.0

# High-Level Problem / Problem Description

In today's rapidly evolving technological landscape, smart home solutions are becoming increasingly prevalent, offering convenience, energy efficiency, and enhanced security.

* However, many existing smart lighting systems suffer from limitations such as complex setup processes, interoperability issues between different manufacturers' devices, and a lack of truly adaptive and scalable control.

I am undertaking this project to address these challenges by developing a robust, user-friendly, and scalable smart home lighting system that prioritizes seamless integration and intelligent automation.

* The ultimate vision is to create an intelligent and adaptive system that integrates with a user's lifestyle, optimizing energy consumption while enhancing comfort and ambiance.
* System that not only allows for remote control but also intelligently adjusts lighting based on environmental factors, occupancy, and user preferences, providing a truly personalized and efficient lighting experience.

The main outcomes of this project will include:

* A functional IoT-enabled smart lighting system prototype capable of controlling multiple lights.
* A scalable architecture demonstrated through AWS deployment.
* A Node-RED flow-based processing system for data aggregation and control.
* An event-based microservice architecture for enhanced flexibility and maintainability.
* A secure deployment of the solution.
* Comprehensive documentation, including a GitHub repository of the code and evidence of scalability experiments.

This project will differentiate itself by focusing heavily on scalability and interoperability from the ground up, utilizing a microservices architecture and cloud-native solutions like AWS.

* While many existing smart lighting systems offer basic control, our solution will emphasize advanced automation based on real-time data, and a flexible architecture that can easily integrate new devices and features without major overhauls.
* The use of Node.js, Node-RED, and AWS as core technologies will provide a robust and extensible foundation for a truly smart and future-proof lighting system.

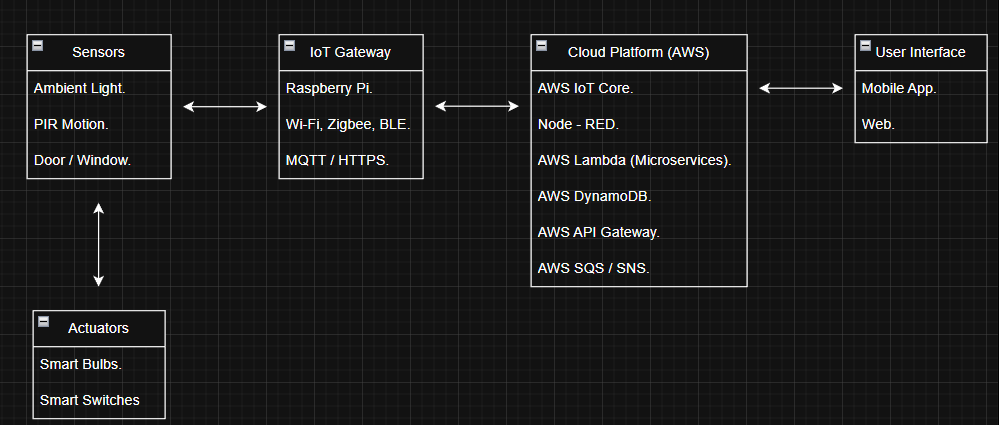
# Solution overview

**A / Proposed Solution**

I am proposing an IoT-enabled smart home lighting system that leverages sensors to gather environmental data (e.g., light intensity, occupancy) and user input to intelligently control lights.

* The system will consist of IoT devices (smart lights, sensors), a gateway for data collection, a cloud-based backend for processing and storage, and a user interface for control and monitoring.
* The core processing logic will be built using Node-RED flows, and the overall architecture will be based on event-driven microservices deployed on Amazon Web Services (AWS).

**B / High-Level Block Diagram**

****

This high-level block diagram illustrates the interconnected components of the smart home lighting system:

* Sensors (Input): Ambient light sensors, PIR motion sensors, and potentially door/window sensors.
* Actuators (Output): Smart light bulbs or smart switches.
* IoT Gateway: A central hub (e.g., Raspberry Pi) that collects data from sensors, translates protocols, and sends commands to actuators.
* Cloud Platform (AWS): The backbone of the system, including AWS IoT Core, Lambda Functions (Microservices), DynamoDB (NoSQL Database), API Gateway, and SQS/SNS (Messaging Services).
* Node-RED: For flow-based programming, automation rules, data aggregation, and integration.
* User Interface: Mobile App/Web Dashboard for monitoring and control.

**System Interconnection:**

* Sensors like ambient light and PIR gather real-time data and communicate wirelessly (Wi-Fi, Zigbee, BLE) with the IoT Gateway.
* Actuators, primarily smart lights, receive commands from the gateway.

**IoT Gateway**:

* Acts as a bridge, collecting raw sensor data and forwarding it to the AWS cloud via secure MQTT or HTTPS.
* It also receives commands from the cloud and transmits them to actuators.

**Cloud Processing (AWS Lambda, Node-RED):**

* Data Ingestion: Sensor data is ingested by AWS IoT Core, which can trigger Lambda functions.
* Node-RED Flows: Used for data aggregation, filtering, simple processing, and defining complex automation rules (e.g., "if motion detected and ambient light is low, turn on lights").
* Microservices (AWS Lambda):
  + Complex business logic, such as predictive lighting, energy optimization, or user management, will be implemented as independent microservices using Node.js and deployed as AWS Lambda functions.
  + These services communicate via AWS messaging services (SQS/SNS).
* Storage (AWS DynamoDB):
  + All collected sensor data, device states, user configurations, and automation rules will be stored in AWS DynamoDB due to its scalability and ability to handle high-velocity data from IoT devices.

**C / Aggregation, Filtering, and Processing**

* ***Aggregation***: Sensor data (e.g., light intensity, occupancy events) will be collected at regular intervals. Aggregation might involve calculating average light levels or counting motion events.
* **Filtering**: Techniques like moving averages or thresholding will be applied to raw sensor data to ensure quality and relevance (e.g., filtering out spurious motion detections).
* ***Processing***:
  + **Rule-based**: Implementing "if-then" rules using Node-RED flows.
  + **Event-driven**: Responding to events like light state changes.
  + **Predictive Analytics (Future)**: Using historical data for proactive lighting adjustments.
  + **User Preference Management**: Processing user input to update lighting scenes and rules.

**D / Scalability**

Scalability is a core requirement, achieved through:

* Microservices Architecture: Independent, loosely coupled services scaled based on demand.
* Serverless Computing (AWS Lambda): Automatic scaling of code execution.
* Managed Databases (AWS DynamoDB): Seamless scalability for storage and throughput.
* Message Queues (AWS SQS/SNS): Decoupling microservices for asynchronous communication.
* AWS IoT Core: Designed to handle billions of devices and trillions of messages.
* Containerization (Potentially AWS ECS/EKS for Node-RED): Horizontal scaling for Node-RED instances if needed.

**E / Testing Plan**

The testing plan will cover several phases:

* Unit Testing: Individual components and functions.
* Integration Testing: Communication and interaction between system parts.
* System Testing: End-to-end testing with real-world scenarios.
* Scalability Testing: Assessing performance under load by simulating numerous devices and data streams.
* Security Testing: Verifying secure deployment aspects.
* User Acceptance Testing (UAT): Engaging users to ensure usability and requirements are met (if time permits).

# Implementation Plan

**A / Hardware/Simulation and Communication**

For the initial prototype, I will utilize a combination of simulated components and readily available hardware:

* Simulated Hardware (Initial Phase):
* Simulated Sensors: Python scripts or Node-RED injection nodes will generate and publish simulated sensor data to AWS IoT Core.
* Simulated Actuators: Dummy APIs or messages will simulate light bulb state changes.
* Physical Hardware (Later Phase/Ideal Scenario):
* IoT Gateway: Raspberry Pi 4 Model B running Node-RED.
* Sensors: BH1750FVI Digital Light Sensor Module and HC-SR501 PIR Motion Sensor Module.
* Actuators: Smart Light Bulbs (e.g., Philips Hue, Tuya) or ESP32/ESP8266 microcontrollers with relays.
* Communication Technologies:
* Wi-Fi: For Raspberry Pi to internet, and Wi-Fi enabled smart bulbs.
* MQTT: Primary protocol for IoT Gateway and AWS IoT Core communication.
* Zigbee/Bluetooth Low Energy (BLE): For local communication with specific smart home devices if needed.

**B / Data Design and Storage**

The data design will primarily focus on using AWS DynamoDB, a NoSQL document database, for its scalability, low latency, and flexibility.

1. Data Collected:

* Sensor Data: deviceId, timestamp, dataType, value, location.
* Actuator/Light State Data: lightId, timestamp, state, commandSource, location.
* User Preferences/Automation Rules: userId, ruleId, ruleName, conditions, actions, schedule.

2. How it is Stored (DynamoDB Tables):

* SensorData Table: Primary Key: deviceId (Partition Key), timestamp (Sort Key).
* LightStates Table: Primary Key: lightId (Partition Key), timestamp (Sort Key).
* UserRules Table: Primary Key: userId (Partition Key), ruleId (Sort Key).

This schema allows for efficient querying of time-series data and flexible storage of user-defined rules.

**C / Cloud Computing Deployment (AWS)**

Amazon Web Services (AWS) will be central to deploying our scalable IoT solution:

* IoT Core: Secure and scalable messaging broker for all IoT devices.
* Lambda: Serverless computing for all business logic, data processing, and microservices (Node.js).
* DynamoDB: Persistent storage for all sensor readings, device states, and user configurations.
* Node-RED Deployment: Can be on an EC2 instance or containerized using AWS ECS/EKS.
* API Gateway: Exposing RESTful APIs for the user interface.
* CloudWatch: Monitoring performance, logging, and setting alarms.
* IAM: Managing permissions and securing access to AWS resources.
* Optional Services: SQS for message queuing, SNS for notifications, S3 for larger files/backups.

# Project Plan

|  |  |  |
| --- | --- | --- |
| Week | Planned | Outcomes |
| 1 | **Project Scoping and Initial Setup**:   * Define problem. * Gather requirements. * Confirm tech stack. * Set up AWS and Git. | * Clear problem statement. * Initial project structure. * Configured AWS account. |
| 2 | **Solution Design and High-Level Architecture:**   * Develop block diagram, data model, scalability design, security plan. | * Detailed high-level design, data schema, Node-RED plan. |
| 3 | **Data Collection (Simulation) and Node-RED Basic Flows:**   * Develop simulated sensors. * Configure AWS IoT Core. * Create basic Node-RED ingestion flows. * Set up DynamoDB tables. | * Simulated sensor data flowing to AWS IoT Core. * Initial Node-RED flows. * DynamoDB tables created. |
| 4 | **Microservice Architecture (Core Logic) & Data Storage:**   * Develop "Sensor Data Processor" and "Light Control" Lambda functions. * Integrate with DynamoDB, unit test. | * Core Lambda functions developed and integrated with DynamoDB. |
| 5 | **Node-RED Automation and Actuator Integration (Simulation):**   * Develop advanced Node-RED automation flows. * Integrate with simulated actuators. * Implement error handling. | * Functional Node-RED automation. * Simulated light control. |
| 6 | **Deployment & Scalability Experiments:**   * Deploy all components to AWS. * Conduct initial scalability tests. * Optimize services. | * Initial deployed solution. * Preliminary scalability test results. |
| 7 | **Secure Deployment & Monitoring:**   * Implement security best practices. * Set up CloudWatch alarms and dashboards, troubleshoot. | * Secure AWS deployment, comprehensive monitoring in CloudWatch. |
| 8 | **Final Review, Refinement, and Documentation:**   * Code review, refactor, final scalability experiments, report writing, GitHub README. | * Optimized code. * Comprehensive project report. * Updated GitHub repository. |
| 9 | **Final Submission Preparation:**   * Compile report. * Finalize GitHub. * Collect evidence. * Review all submission requirements. | * Complete submission package. * All evidence prepared. |