Cloud Based Home Automation

Rani Rajurkar
rrajurkar@scu.edu

Ningchen Liang
nliang@scu.edu

Inchara Raveendra
iraveendra@scu.edu

Kiran Kodali
kkodali@scu.edu

Abstract—**The increasing adoption of smart devices and the Internet of Things (IoT) has revolutionized modern living. Nevertheless, the existence of diverse control utilities for these gadgets is a cause for concern since it has led to fragmentation and disorder that make the dream of a perfect smart home impossible. This project aims at solving this problem by creating a Cloud Based Home Automation Application that can provide a single interface for controlling devices from different vendors regardless of their vendor restrictions. Our solution will rely on cloud technology as well as solid tech stacks in order to achieve customizable, secure , and coherent smart homes. The application contains a web-platform which integrates with devices so end-users have the option to register or manage their smart appliances via one central place. Thus, the end result is a scalable and user-friendly solution ensuring efficient control over residential automation networks.This report highlights our system's architecture, approach, outcomes, and potential future directions.**

INTRODUCTION

Use of smart devices in our everyday life has increasingly become a common sight in the era of Internet of Things (IoT). The present state of smart home control on one hand comes with convenience and automation, and on the other hand is often disorganised as users have to navigate through several applications to handle different gadgets from multiple vendors. Thus, this disintegration acts like a major obstacle towards a seamless and integrated smart home experience

A Cloud Based Home Automation Application proposed in our project unifies device control while also providing customization and interoperability for a more user-friendly and secure smart home environment. By employing cloud technologies as well as an elastic technology stack which incorporates React and Nginx for front-end; Express.js for device integration and back-end; Docker for containerization; and MongoDB for back-end data storage, we plan to revolutionize how people interact with their smart home ecosystems.

The key idea behind our approach include development of a web application with a unified interface, which would allow users to manage their smart home devices centrally. Moreover, our solution provides robust capability for integrating various devices made by different manufacturers to onboard and manage them. Additionally, through applying cloud computing's scalability and security features, our application delivers secure personalization of smart homes that are not restricted by vendor specific solutions.

BACKGROUND AND RELATED WORK

Home automation is a notion that has been around for some decades, in the initial years based on wired connections and proprietary protocols. Nevertheless, with the advent of Internet of Things (IoT) and proliferation of smart devices, home automation has drastically changed. The existing systems like Google Home and Apple Home have pioneered integration of smart devices into their ecosystems. These platforms are convenient, however they also come with vendor lock-in as well as compatibility problems for individuals who own several devices from different vendors.

A number of open-source and community-based initiatives have emerged to overcome these limitations by creating an integrated control platform for smart home devices. A key case here is the OpenHAB project, (openhab.org) that strives to integrate different home automation systems and technologies into a single solution. This means that with the OpenHAB platform, users can control their home environments remotely from any location since it supports various protocols for smart devices. In addition, there is also the Home Assistant project (home-assistant.io), which has come out as another major player in this field with its open-source home automation and all encompassing interface for controlling various types of smart devices.

However, due to the advancement of these existing solutions that have unified smart home control substantially, they require complex setups as well as configurations which might be hard for non-technical users. Moreover, most of them come in self-hosted forms which has limited their scalability and brought in security issues plus maintenance concerns. Lastly, as long as there is a fast changing landscape of smart devices, compatibility maintenance and guaranteeing smooth interoperability across multiple vendors will remain a daunting task so far.

With our Cloud-Based Home Automation Application, we hope to extend what these works started, taking advantage of the capabilities and versatility of cloud computing and distributed systems. Hosting on a cloud platform means that we can do away with the complications of local installments or maintenance, as well as enjoying scalability, security, and reliability characteristics offered by providers of cloud services. Besides that, it is important to note that this product also comes with an efficient device integration feature which integrates various devices in one place. This way the client who has a smart home developed by different producers will be able to fully control them in one app without changing it for each producer separately.

APPROACH

The primary objective of this project is to develop a unified application capable of controlling smart devices from various vendors, providing users with a centralized and seamless experience for managing their smart home ecosystem.

I. Application Architecture

The Cloud Based Home Automation Application follows a microservices architecture, comprising four main components:

1) Dashboard Backend Server Container
2) Frontend Server Container
3) MongoDB Database Container
4) Device Integration Server Container

A. Dashboard Backend Server

The dashboard backend server is built using Express.js, a popular Node.js web application framework. It serves as the backbone of the application, facilitating Create, Read, Update, and Delete (CRUD) operations through a robust REST API.

1. REST API: The Express.js-based REST API enables the frontend web application to perform various actions, such as controlling devices, retrieving device states, and managing user preferences.

2. MongoDB Integration: The backend server interacts with the MongoDB database to store and retrieve essential application data, including enrolled widgets (devices), published applications, and the associated widget details for each application.

B. Frontend Server

The frontend server hosts the user interface of the application, built using React, a popular JavaScript library for building responsive and dynamic web applications.

1. User Interface: The React-based UI provides a seamless and intuitive experience for users to monitor and control their smart devices.

2. Backend Communication: The frontend communicates with the backend server to retrieve the latest

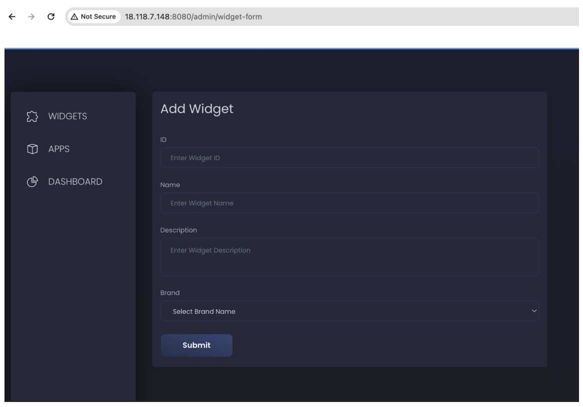


Figure 1. Add widget window

published application and its associated widgets using the GET /apps/latest API endpoint.

3. Device Integration Communication: To interact with the connected devices, the frontend communicates with the device integration server using specific endpoints for each supported vendor.

C. MongoDB Database

The application leverages MongoDB, a NoSQL document-oriented database, to store and manage application data.

1. Widgets (Devices): Information about enrolled widgets, including device ID, name, description, and associated brand, is stored in the database.

2. Published Applications: Details about published applications, such as the application name, description, and the list of associated widgets, are stored and retrieved from the database.

3. Data Persistence: The database architecture follows the Model-View-Controller (MVC) methodology to ensure data persistence and facilitate two-way binding between the application and the connected devices.

D. Device Integration Server

The Device Integration backend, serving as the linchpin of our smart home dashboard, ingeniously encapsulates the native APIs provided by various device manufacturers such as Philips Hue and LIFX. By implementing these wrapper APIs, our backend does more than just relay commands; it introduces a layer of enhanced security, ensuring that all interactions with our users' home devices are conducted in a secure and protected manner. Furthermore, this approach standardizes the communication protocol across devices from different brands, offering uniformity in handling disparate data formats and command structures. This uniformity is crucial for maintaining a seamless and intuitive user experience, regardless of the underlying device's brand or model.

Moreover, the integration facilitates unparalleled interoperability among a diverse array of smart home devices. By abstracting the complexities of each brand's unique API, our backend serves as a universal translator, allowing for the effortless orchestration of various smart devices. This not only simplifies the development and maintenance of the dashboard but also empowers users to customize and control their smart home ecosystem from a single, unified interface. The outcome is a beautifully synchronized symphony of smart devices, all working in concert to create a more intelligent, responsive, and harmonious living environment.

1. Vendor Integration: For demonstration purposes, the application integrates with two major vendors: Philips and LIFX.

2. Device Communication: LIFX smart devices connect directly to the home router, while Philips devices communicate through a dedicated Hue bridge, which connects to the Philips devices using the Zigbee protocol. The device integration server communicates with these devices and bridges using the HTTP protocol.

3. REST APIs: The device integration server exposes REST APIs for interacting with the connected devices, allowing the frontend to retrieve and update device states, such as on/off status, brightness, saturation, hue, and color.

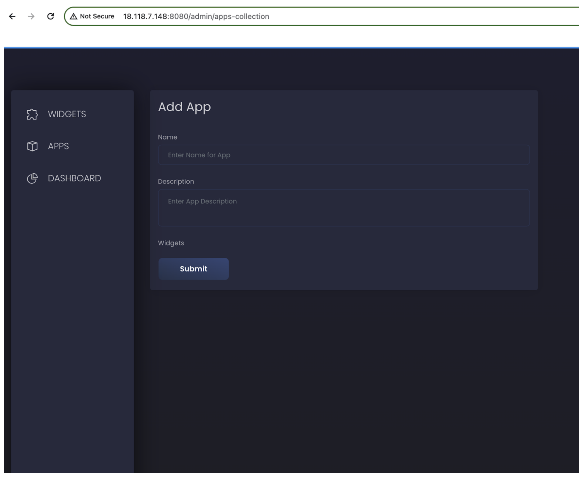


Figure 2. Add app window

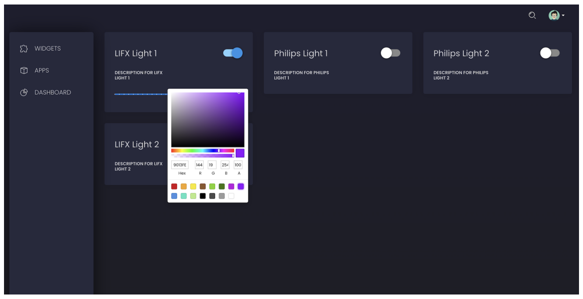

WIDGETS
APPS
DASHBOARD
LIFX Light 1
DESCRIPTION FOR LIFX LIGHT 1
Philips Light 1
DESCRIPTION FOR PHILIPS LIGHT 1
Philips Light 2
DESCRIPTION FOR PHILIPS LIGHT 2
LIFX Light 2
DESCRIPTION FOR LIFX LIGHT 2

Figure 3. Dashboard with 4 widgets subscribed and slider to change brightness and color picker

II. Application Workflow

A. Device Registration

1. Adding a New Device (Widget): When a new device is acquired, the user registers it in the application using the ”Add Widget” functionality. The user provides the device ID, name, description, and selects the associated brand (e.g., Philips or LIFX) from a dropdown list.

2. Widget API Endpoints: The widget registration information is sent to the backend server using the POST /widgets API endpoint. The application also implements duplicate name checking for widgets using the POST /widgets/duplicate endpoint to ensure consistent data management.

B. Application Publication

1. Creating a New Application: To create and publish a new application, the user navigates to the ”Add App” section. They provide a name and description for the application and select the desired widgets (devices) from a multi-select modal displaying all the registered widgets.

2. Application API Endpoints: The list of registered widgets is retrieved from the backend server using the GET /widgets endpoint. The new application is created by sending a request to the POST /apps endpoint, which also includes duplicate name checking using the POST /apps/duplicate endpoint.

C. Dashboard and Device Control

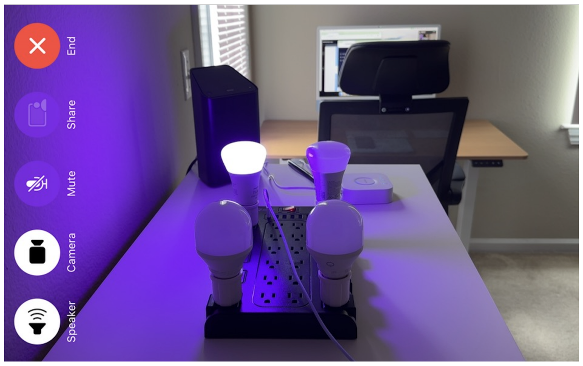
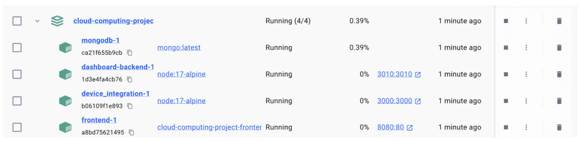


Figure 4. Example of lights changed to purple as chosen in the above color picker



cloud-computing-projec		Running (4/4)	0.39%		1 minute ago	
mongodb-1 ca21f655b9cb	mongo:latest	Running	0.39%		1 minute ago	
dashboard-backend-1 1d3e4fa4cb76	node:17-alpine	Running	0%	3010:3010	1 minute ago	
device_integration-1 b06109f1e893	node:17-alpine	Running	0%	3000:3000	1 minute ago	
frontend-1 a8bd75621495	cloud-computing-project-fronter	Running	0%	8080:80	1 minute ago	

Figure 5. Snapshot of docker containers including 4 containers used for the project

1. Viewing the Published Application: The user can open the dashboard to view the latest published application, retrieved using the GET /apps/latest endpoint. The dashboard dynamically renders cards for each included widget, displaying their details, brand, description, and name.

2. Device Control: Each widget card includes a slider for controlling the brightness of the connected device and a color picker for changing its color. These controls communicate with the device integration server to get and update the respective device states using the appropriate vendor-specific endpoints (e.g., localhost:3000/philips/lights/{philips-id}/state for Philips or localhost:3000/lifx/lights/lifx-id/state for LIFX).

3. Two-way Binding: The application follows the MVC pattern to ensure two-way binding between the application and the connected devices. If a device’s state is changed externally, the application reflects the updated state, and vice versa.

III. Authentication: LIFX and Philips Hue

Authentication for Smart Light Control with LIFX and Philips Hue involves distinct methods to ensure secure API access, reflecting their unique security approaches:

LIFX Authentication: Utilizes OAuth 2.0, requiring an access token for API requests. Tokens are generated in LIFX account settings and included in requests as bearer tokens in the HTTP Authorization header or as the username in HTTP Basic Authentication (password field left empty). Essential to use HTTPS for all requests to encrypt token transmission, with HTTP requests being rejected.

Philips Hue Authentication: Requires connecting to the Philips Hue bridge and generating an authorized user or ’username’ (hue-application-key) after discovering the bridge’s IP address. IP discovery can be done via the Philips Hue app, mDNS discovery

	Philips	LIFX
Switch	“on” : true/false	“power”: “on/off”
Slider	“bri”: [0-255]	“brightness”: [0-1] in steps of 0.1
Color Picker	“hue”: [0-255] “sat”: [0-65535]	“color”: “#ff0000”

Figure 6. Difference in attributes and data range for Philips and Lifx to set light state

apps, the Philips Hue broker server, or your router's DHCP table. The initial setup includes sending a POST request to the bridge with your application's name to generate the application key, necessitating a physical button press on the bridge for security.

Both systems emphasize secure API interactions to prevent unauthorized access, with LIFX focusing on token-based access and Philips Hue requiring a direct connection to the bridge and an application key generation process.

IV. Mathematical Computations and Data Manipulations for Dashboard

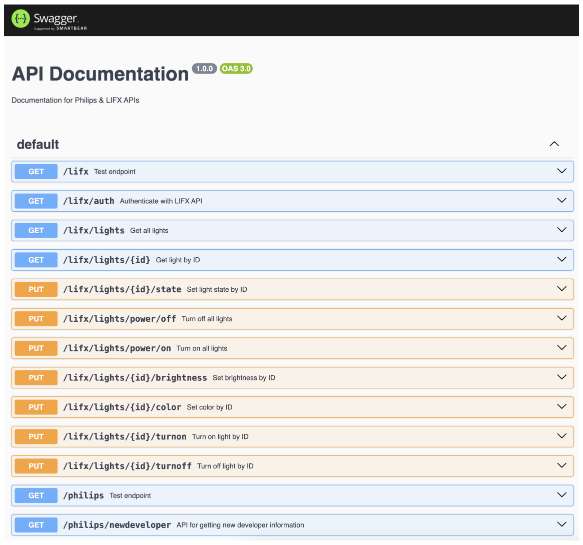
A. Setting the States

1. Power State:

- Philips Hue: Uses a Boolean value for the "on" key to represent the power state (true for on, false for off).
- LIFX: Expects a string value ("on" or "off") for the "power" key.

2. Brightness:

- User Interface: Presents a slider with a range of 0-100 in steps of 10 to enhance user experience and minimize excessive API calls.
- Philips Hue: Expects a brightness value in the range of 0-255. The UI value is multiplied by 2.55 to convert to this range.
- LIFX: Expects a brightness value in a decimal range of 0-1. The UI value is divided by 100 for this conversion.



Swagger
Supported by SMARTBEAR

API Documentation 1.0.0 OAS 3.0

Documentation for Philips & LIFX APIs

default

GET /lifx Test endpoint
GET /lifx/auth Authenticate with LIFX API
GET /lifx/lights Get all lights
GET /lifx/lights/{id} Get light by ID
PUT /lifx/lights/{id}/state Set light state by ID
PUT /lifx/lights/power/off Turn off all lights
PUT /lifx/lights/power/on Turn on all lights
PUT /lifx/lights/{id}/brightness Set brightness by ID
PUT /lifx/lights/{id}/color Set color by ID
PUT /lifx/lights/{id}/turnon Turn on light by ID
PUT /lifx/lights/{id}/turnoff Turn off light by ID
GET /philips Test endpoint
GET /philips/newdeveloper API for getting new developer information

Figure 7. Swagger documentation of device integration APIs

3. Color:

- User Interface: Provides RGBA values through a color picker.
- LIFX: The RGBA values are converted into a hexadecimal code suitable for the API.
- Philips Hue: Requires "hue" (0-65535) and "saturation" (0-255). The RGBA values are converted into these hue and saturation values through an intermediate HSL conversion.

B. Fetching and Persisting the States

1. Power State:

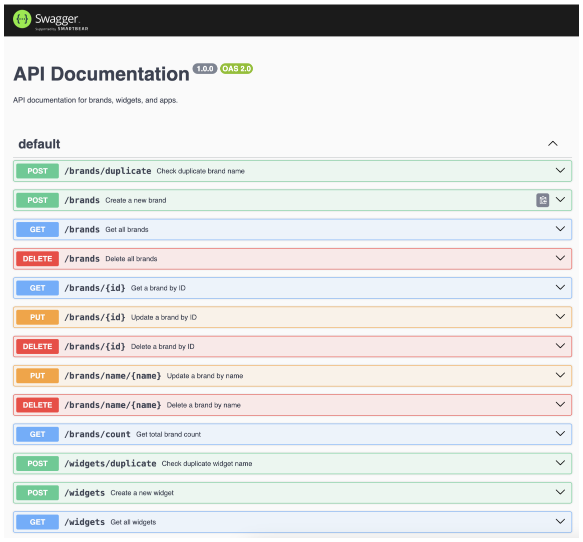
Philips Hue and LIFX: The power state ("on"/true for Philips, "on"/"off" for LIFX) fetched from the device's API is used to update the switch element in the UI.

2. Brightness:

- Philips Hue: Brightness values fetched in the range of 0-255 are converted back to the 0-100 range for the UI slider by dividing by 2.55.
- LIFX: Brightness values fetched in the decimal range of 0-1 are converted to the 0-100 range by multiplying by 100.

3. Color:

- Philips Hue: Fetches "hue" and "saturation" values, which are converted back to RGBA format for the color picker. This conversion involves using the hue (0-65535) and saturation (0-255) to calculate the



Swagger
Supported by SMARTBEAR

API Documentation 1.0.0 OAS 2.0

API documentation for brands, widgets, and apps.

default

POST /brands/duplicate Check duplicate brand name
POST /brands Create a new brand
GET /brands Get all brands
DELETE /brands Delete all brands
GET /brands/{id} Get a brand by ID
PUT /brands/{id} Update a brand by ID
DELETE /brands/{id} Delete a brand by ID
PUT /brands/name/{name} Update a brand by name
DELETE /brands/name/{name} Delete a brand by name
GET /brands/count Get total brand count
POST /widgets/duplicate Check duplicate widget name
POST /widgets Create a new widget
GET /widgets Get all widgets

Figure 8. Swagger documentation of backend APIs

equivalent RGB values and then converting these to RGBA.

- LIFX: Fetches color properties as ”hue,” ”sat,” and ”kelvin” values, which are then converted to RGBA format for updating the UI’s color picker. This involves a reverse conversion process from the device-specific color representation to a standard RGBA format that the UI can display.

The intricate color computations within the dashboard are facilitated through a sophisticated suite of data manipulation functions. These functions are essential in translating color values between different formats, enabling seamless interaction between the user interface and the smart lighting devices. The process involves:

- rgbToHsl
- rgbToHueSat
- hexToRgb
- rgbToHex
- hueSatToHex
- hslToRgb

These functions collectively enable the dashboard to accurately represent and manipulate color settings across different smart lighting systems, ensuring that the user’s color preferences are precisely applied and displayed, regardless of the underlying hardware’s color specification requirements.

V. Development and Deployment

A. Local and Development Environments

1. Docker Compose: Docker Compose is utilized for local and development environments, providing a seamless mechanism for bringing up and managing the application containers using YAML scripts.

2. Environment Setup and Teardown: Scripts are provided for setting up and cleaning up the development environment, including starting and stopping the containers.

3. Networking Considerations: Docker Networks: Use Docker networks to facilitate communication between containers. Exposing Ports: Expose only the necessary ports to the outside world for security. The frontend might be on ports 80/443, while backend APIs could be on other ports not exposed publicly. Load Balancing: Docker Swarm

B. Production Deployment

1. Docker Swarm: For large-scale production-grade deployments, Docker Swarm is employed for better utilization of the orchestration layer, enabling easy scaling up and down of containers based on load and traffic demands.

2. High Availability and Scalability: Docker Swarm ensures high availability and efficient resource utilization across multiple Docker hosts, allowing the application to scale horizontally to meet increasing demands.

VI. API Documentation and Maintenance

Swagger is used for API documentation and maintenance, providing a comprehensive and user-friendly interface for exploring and testing the available APIs.

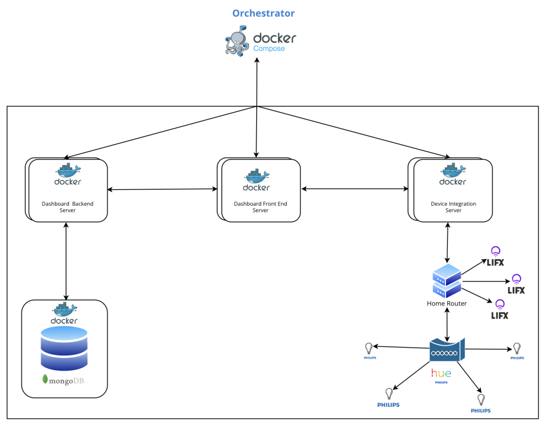


Figure 9. System architecture of local development using docker compose

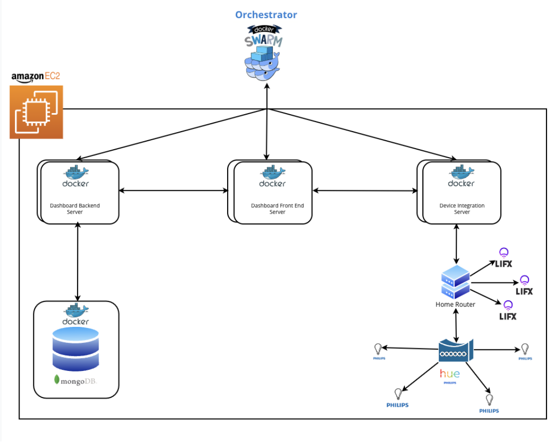


Figure 10. System architecture of production environment using docker swarm on EC2 instance for better scaling and control over orchestration layer

This tool allows developers to catch and fix issues, as well as test the APIs in local or development environments, ensuring smooth integration and functionality.

By leveraging a microservices architecture, integrating with various smart device vendors, and employing containerization and orchestration technologies, the Cloud Based Home Automation Application aims to deliver a scalable, reliable, and user-friendly solution for managing smart home ecosystems from a centralized platform.

OUTCOME

The Cloud Based Home Automation Application is built on the principles of scalability and flexibility using the capabilities of cloud computing and container orchestration. By means of AWS cloud platform coupled with Docker and Docker Swarm, all its components: front-end, back-end, as well as device integration modules are containerised for uniformity in deploying them to different environments.

A React powered application which is front-end, hosted on an Nginx web server, can scale up or down effectively with the number of users and their various demands. In this case scaling is done by dynamically resizing the docker swarm cluster's frontend containers. Also, there are separate containerized services for backend APIs and device integration components allowing for independent scaling and resource allocation based on their individual computational needs.

This comprises Docker Swarm, which is a robust tool for orchestration of containers that play an important role in the management and coordination of containerized application components. It can enable automatic scaling and load balancing thus making sure that the application is able to handle different levels of user traffic as well as device integration workloads without any compromise on its performance or availability. Docker Swarm built-in load balancing mechanisms distribute incoming requests across multiple containers which provides efficient utilization of available resources and minimizes overloading risk on any single component. Moreover, Docker Swarm auto scaling allows the application to scale up or down the number of containers dynamically based on predefined scaling policies hence ensuring optimal resource usage and meeting shifting requirements within the smart home ecosystem.

More so, deployment and management of an application's components become simplified by Docker Compose which is a utility for defining and running multi-container Docker applications. This means that services, networks, and volumes can be configured in a declarative way allowing easier updates, rollbacks and scaling adjustments. So in general AWS cloud infrastructure combined with Docker containerization and Docker Swarm orchestration enables creation of robust, scalable foundation for the Cloud Based Home Automation Application that assures reliability as well as the efficient performance to complement dynamically changing requirements of smart home environments.

ANALYSIS AND FUTURE WORK

Our Cloud Based Home Automation Application offers a complete solution for integrating smart home control, but there are some limitations and areas for further improvement:

1. **Device Compatibility and Integration**: To improve the usability and appeal of this application in the future, one possible direction might be to expand device compatibility by developing more integration modules with other vendors besides smart bulbs and prospering smart home technologies.

2. **User Experience and Customization**: The current web interface provides basic functionality for controlling devices and monitoring them. Further versions can consider advanced user experience features like customizable dashboards, automation rule creation, intelligent recommendations based on user preferences as well as device usage patterns.

3. **Security and Privacy Considerations**: While our application uses cloud-based infrastructure and secure communication protocols to it, there is room to explore more security measures like implementing strong authentication mechanisms and authorization mechanisms. It should be a priority moving forward to incorporate privacy-preserving techniques as well as complying with relevant data protection regulations.

4. **Scalability and Performance Optimization**: In such scenarios where we have increased the number of connected devices alongside users, the application's scalability and responsiveness could improve more through employing serverless architectures, caching mechanisms, and advanced strategies for load balancing.

5. **Integration with Smart City Initiatives**: Our

Cloud Based Home Automation Application could potentially be extended to cater to broader smart city use cases such as building automation, energy management and urban infrastructure monitoring. - Explorations of integrations with smart city platforms and leveraging IoT technologies for seamless data exchange may lead to new areas of innovation and collaboration.

CONCLUSION

The Cloud Based Home Automation Application is a significant step towards consolidating the control and management of smart home ecosystems. With the use of cloud technology and a strong technological foundation, our solution is aimed at solving the fragmentation and disorganization that characterize today's smart home landscape.

This project provides a customizable and secure platform to allow users control and monitor their smart devices from different vendors. This is achieved by developing a web application with an integrated interface, back-end architecture that can be scaled up and down, and robust device integration capabilities. However, the current system requires additional development efforts to easily accommodate a wider variety of smart devices and brands beyond Philips and LIFX without extensive individualized work. Enhancing the flexibility of the architecture to support a broader range of devices remains a challenge and an area for future improvement.

The incorporation of cloud computing, containerization and automation tools guarantees that the application can easily scale up or down, and will be resilient, resource-efficient to cope with the increased complexities of smart homes. Although we met the primary objectives in achieving a holistic, user friendly smart home experience there are still future improvements. Dealing with device compatibility enhancements, improving user customization and experience, enhancing privacy and security measures as well as exploring integration with smart city projects are some possible directions for this study.

The overall view of our Cloud Based Home Automation Application is that it signals a big step towards changing the way users communicate with their smart home ecosystems and enabling smooth, seamless, safe and efficient integration of IoT devices in our daily lives.

REFERENCES

1. OpenHAB Project. (n.d.). Retrieved January 30,2024 from https://www.openhab.org/
2. Home Assistant. (n.d.). Retrieved January 30, 2024 from https://www.home-assistant.io/
3. Docker Documentation. (n.d.). Retrieved from https://docs.docker.com/
4. AWS Documentation. (n.d.). Retrieved from https://aws.amazon.com/documentation/
5. React Documentation. (n.d.). Retrieved from https://reactjs.org/docs/getting-started.html
6. Express.js Documentation. (n.d.). Retrieved from https://expressjs.com/en/guide/routing.html
7. Philips Hue authentication documentation. (n.d.). Retrieved from https://developers.meethue.com/develop/hue-api-v2/getting-started/#follow-3-easy-steps
8. LIFX authentication documentation. (n.d.). Retrieved from https://api.developer.lifx.com/reference/authentication