Smart Home Management System

Alexandru Georgescu
23124091

Domain Description:

The Smart Home Management System is designed to provide users with convenient control and monitoring capabilities for various smart home devices. The application consists of three main services: **Thermostat Service**, **Lighting Service**, and **Doorbell Service**.

- **Thermostat Service**:
  - This service allows users to monitor and control the temperature of their home environment. Users can query the current temperature, set desired temperature levels, manage boost settings for heating, and check the status of the hot water system.
- **Lighting Service**:
  - The Lighting Service enables users to control the lighting system in their home. Users can query the status of the lights, turn them on or off, adjust brightness levels, and change the color of the lights.
- **Doorbell Service:**
  - The Doorbell Service provides functionality related to a smart doorbell system. Users can check the live feed of the doorbell, turn it on / silence it, and receive notifications about doorbell events.

Service Definition and RPC:

Thermostat Service:

- Temperature Status:
  - Request: Empty message.
  - Response: Message containing the current temperature.

- Set Temperature:
  - Request: Message containing the desired temperature.
  - Response: Message confirming the temperature has been set.

- Boost Status:
  - Request: Empty message.
  - Response: Message containing boost status, including temperature and time remaining.

- Manage Boost:
  - Request: Message containing new boost temperature and time.
  - Response: Message confirming boost adjustment.

- Set Boost:
  - Request: Message indicating whether to enable or disable boost.
  - Response: Message confirming boost status change.

- Hot Water Status:

- Request: Empty message.
- Response: Message indicating hot water status (on or off).

- Set Hot Water:
  - Request: Message indicating whether to turn hot water on or off.
  - Response: Message confirming hot water status change.

- **Lighting Service**:
  - Light Status:
    - Request: Empty message.
    - Response: Message indicating whether the light is on or off.

  - Change Light On/Off:
    - Request: Message indicating wheter to enable or disable light.
    - Response: Message confirming light status changed.

  - Brightness Status:
    - Request: Empty message.
    - Response: Message containing the current brightness level.

  - Change Brightness:
    - Request: Message containing the new brightness level.
    - Response: Message confirming brightness adjustment.

  - Color Status:
    - Request: Empty message.
    - Response: Message containing the current color of the light.

  - Change Color:
    - Request: Message containing the new color.
    - Response: Message confirming color change.

- **Doorbell Service:**
  - LiveVideoFeed:
    - Request: Empty message.
    - Response: Message indicating the URL of the stream

  - TodaysEvents:
    - Request: Empty message.
    - Response: Message indicating events that happened in the current day

  - SilentModeStatus
    - Request: Empty message.
    - Response: Message indicating if silent mode is on/off

  - ToggleSilentMode
    - Request: Message indicating wheter to enable or disable silent mode(on/off)
    - Response: Message confirming change of silent mode

Overall RPC Types:

- The application utilizes unary RPC for most interactions between clients and

services, where a single request triggers a single response.
- Certain functionalities, such as managing boost settings and adjusting brightness levels, involve bidirectional streaming RPC, allowing continuous communication between the client and server.

Service Definitions:

- **Thermostat Service:**
  - RPCs: TemperatureStatus, SetTemperature, BoostStatus, ManageBoost, SetBoost, HotWaterStatus, SetHotWater
  - Message Definitions: TemperatureStatusRequest, TemperatureStatusResponse, SetTemperatureRequest, SetTemperatureResponse, BoostStatusRequest, BoostStatusResponse, ManageBoostRequest, ManageBoostResponse, SetBoostRequest, SetBoostResponse, HotWaterStatusRequest, HotWaterStatusResponse, SetHotWaterRequest, SetHotWaterResponse
- **Lighting Service:**
  - RPCs: LightStatus, BrightnessStatus, ChangeBrightness, ColourStatus, ChangeColour
  - Message Definitions: LightStatusRequest, LightStatusResponse, BrightnessStatusRequest, BrightnessStatusResponse, ChangeBrightnessRequest, ChangeBrightnessResponse, ColourStatusRequest, ColourStatusResponse, ChangeColourRequest, ChangeColourResponse
- **Doorbell Service:**
  - RPCs: LiveVideoFeed, TodaysEvents, SilentModeStatus, ToggleSilentMode
  - Message Definitions: LiveVideoFeedRequest, LiveVideoFeedResponse, TodaysEventsRequest, TodaysEventsResponse, SilentModeStatusRequest, SilentModeStatusResponse, ToggleSilentModeRequest, ToggleSilentModeResponse

Service Implementations

- The services are implemented using gRPC, a high-performance, open-source RPC framework.
- Each service is implemented as a separate module, with well-defined functionalities and message formats.
- Error handling and advanced features, such as server-streaming RPC, are incorporated into the implementation to ensure robustness and efficiency.

Naming Services:

- Services are named according to their respective functionalities and domains, following consistent naming conventions.
- Each service is identified by a unique name, making it easy for clients to interact with the desired service.

Remote Error Handling & Advanced Features:

- Error handling mechanisms are implemented to handle various types of errors, including network errors, service unavailable errors, and invalid request errors.
- Advanced features such as server-streaming RPC are utilized for functionalities that require continuous communication from the server to the client.

Client - Graphical User Interface (GUI)/Command Prompts:

- Clients interact with the services through a user-friendly graphical interface or command-line prompts, providing intuitive access to smart home functionalities.
- Examples of client interactions include querying light status, adjusting brightness levels, turning the heat on/off and activating the doorbell.

GitHub:

- **https://github.com/alxgrgsc/smart_home_systems**

SOURCE CODE

PROTO FILES

smart_lighting.proto

```
syntax = "proto3";
package smart_home;

// Lighting service
service Lighting {
    rpc LightStatus(LightStatusRequest) returns (LightStatusResponse) {}
    rpc BrightnessStatus(BrightnessStatusRequest) returns (BrightnessStatusResponse) {}
    rpc ColourStatus(ColourStatusRequest) returns (ColourStatusResponse) {}
    rpc ChangeLightStatus(ChangeLightStatusRequest) returns (ChangeLightStatusResponse) {}
    rpc AdjustBrightness(AdjustBrightnessRequest) returns (AdjustBrightnessResponse) {}
    rpc ChangeColour(ChangeColourRequest) returns (ChangeColourResponse) {}
}

// Lightstatus request
message LightStatusRequest {
}

// Lightstatus response
message LightStatusResponse {
    string message = 1;
}

// BrightnessStatus request
message BrightnessStatusRequest {
```

```
}

// BrightnessStatus response
message BrightnessStatusResponse {
    int32 brightnessLevel = 1;
}

// ColourStatus request
message ColourStatusRequest {
}

// ColourStatus response
message ColourStatusResponse {
    string colour = 1;
}

// ChangeLightStatus request
message ChangeLightStatusRequest {
    bool isOn = 1;
}

// ChangeLightStatus response
message ChangeLightStatusResponse {
    string message = 1;
}

// AdjustBrightness request
message AdjustBrightnessRequest {
    int32 newBrightnessLevel = 1;
}

// AdjustBrightness response
```

```
message AdjustBrightnessResponse {
  string message = 1;
}


// ChangeColour request
message ChangeColourRequest {
  string colour = 1;
}

// ChangeColour response
message ChangeColourResponse {
  string message = 1;
}
```

smart_doorbell.proto

```
syntax = "proto3";
package smart_home;

// Doorbell service
service Doorbell {
  rpc LiveVideoFeed(LiveVideoFeedRequest) returns (stream LiveVideoFeedResponse) {}
  rpc TodaysEvents(TodaysEventsRequest) returns (stream TodaysEventsResponse) {} // Change return type to stream
  rpc SilentModeStatus(SilentModeStatusRequest) returns (SilentModeStatusResponse) {}
  rpc ToggleSilentMode(ToggleSilentModeRequest) returns (ToggleSilentModeResponse) {}
}

// Request to get live video feed
message LiveVideoFeedRequest {}
```

```
// Response containing the live video feed URL
message LiveVideoFeedResponse {
  string liveVideoUrl = 1;
  string description = 2;
}

// Request to get today's events
message TodaysEventsRequest {}

// Response containing an event statement
message TodaysEventsResponse {
  string eventStatement = 1;
}

// Request to get the silent mode status
message SilentModeStatusRequest {}

// Response containing the silent mode status
message SilentModeStatusResponse {
  bool isSilentModeOn = 1;
}

// Request to toggle the silent mode
message ToggleSilentModeRequest {
  bool toggle = 1;
}

// Response indicating the result of toggling the silent mode
message ToggleSilentModeResponse {
  string message = 1;
}
```

smart_thermostat.proto

```
syntax = "proto3";

package smart_home;


// Thermostat service

service Thermostat {

  rpc TemperatureStatus(TemperatureStatusRequest) returns
(TemperatureStatusResponse) {}

  rpc SetTemperature(SetTemperatureRequest) returns (SetTemperatureResponse)
{}

  rpc BoostStatus(BoostStatusRequest) returns (BoostStatusResponse) {}

  rpc ManageBoost(ManageBoostRequest) returns (ManageBoostResponse) {} //
Renamed from BoostTemperature

  rpc SetBoost(SetBoostRequest) returns (SetBoostResponse) {}

  rpc HotWaterStatus(HotWaterStatusRequest) returns (HotWaterStatusResponse)
{}

  rpc SetHotWater(SetHotWaterRequest) returns (SetHotWaterResponse) {}

}


// TemperatureStatus request

message TemperatureStatusRequest {

}


// TemperatureStatus response

message TemperatureStatusResponse {

  double currentTemperature = 1;

}


// SetTemperature request

message SetTemperatureRequest {

  double setTemperature = 1;

}
```

```
// SetTemperature response
message SetTemperatureResponse {
    string message = 1;
}

// BoostStatus request
message BoostStatusRequest {
}

// BoostStatus response
message BoostStatusResponse {
  bool isBoostActive = 1;
  int32 boostTimeRemaining = 2;
  double boostTemperature = 3;//
}

// ManageBoost request
message ManageBoostRequest {
    double boostTemperature = 1;
    int32 boostTime = 2; //
}

// ManageBoost response
message ManageBoostResponse {
    string message = 1;
}

// SetBoost request
message SetBoostRequest {
  bool setBoost = 1;
}
```

```
// SetBoost response
message SetBoostResponse {
  string message = 1;
}

// HotWaterStatus request
message HotWaterStatusRequest {
}

// HotWaterStatus response
message HotWaterStatusResponse {
    string message = 1;
}

// SetHotWater request
message SetHotWaterRequest {
    bool setHotWater = 1;
}

// SetHotWater response
message SetHotWaterResponse {
    string message = 1;
}
```

lighting_server.js

```
//Purpose: Server for the smart lighting service. It provides the implementation for the lighting service methods, such as querying the current status of the light, turning the light on or off, adjusting the brightness level, and changing the colour of the light.

//import modules
const grpc = require('@grpc/grpc-js');
```

```
const protoLoader = require('@grpc/proto-loader');
const PROTO_PATH = "./lighting_service/smart_lighting.proto";
const packageDefinition = protoLoader.loadSync(PROTO_PATH);
const smartHomeProto =
grpc.loadPackageDefinition(packageDefinition).smart_home;

//initialize default states
let lightStatus = 0; // 0 off / 1 on
let brightnessLevel = 100; // brightness
let currentColour = 'white'; // colour
//allowed colours
const ALLOWED_COLOURS = ['white', 'red', 'green', 'blue', 'yellow', 'orange',
'purple', 'pink'];

//create gRPC server
const server = new grpc.Server();


// gRPC service method for LightStatus
function getLightStatus(call, callback) {
  console.log('Request for Light Status received');
  let statusMessage = lightStatus ? 'Light is on' : 'Light is off';
  console.log('Sending Light Status:', statusMessage);
  callback(null, { message: statusMessage });
}

// gRPC service method for BrightnessStatus
function getBrightnessStatus(call, callback) {
    console.log('Request for Brightness Status received');
    brightnessLevel = Math.max(0, Math.min(100, brightnessLevel));
    console.log('Sending Brightness Status:', brightnessLevel, '%');
    callback(null, { brightnessLevel: brightnessLevel });
```

```
}

// gRPC service method for AdjustBrightness
function adjustBrightness(call, callback) {
  console.log('Request to adjust brightness received');
  const newBrightnessLevel = call.request.newBrightnessLevel;
  brightnessLevel = Math.max(0, Math.min(100, newBrightnessLevel));
  console.log('Brightness level:', brightnessLevel,'%');
  callback(null, { message: `Brightness level: changed to ${brightnessLevel}%` });
}

// gRPC service method for ChangeLightStatus
function changeLightStatus(call, callback) {
  console.log('Request to change light status received');
  let newLightStatus = call.request.isOn;
  if (typeof newLightStatus !== 'boolean') {
    const errorMessage = 'Invalid value for new light status. Must be a boolean (true or false).';
    console.error(errorMessage);
    callback({ code: grpc.status.INVALID_ARGUMENT, message: errorMessage });
    return;
  }
  lightStatus = newLightStatus;
  console.log('New light status:', lightStatus ? 'on' : 'off');
  callback(null, { message: `Light status: changed to ${lightStatus ? 'on' : 'off'}` });
}

// gRPC service method for ColourStatus
function getColourStatus(call, callback) {
    console.log('Request for Colour Status received');
    console.log('Sending Colour Status:', currentColour);
    callback(null, { colour: currentColour });
```

```
}

// gRPC service method for ChangeColour
function changeColour(call, callback) {
    console.log('Request to change colour received');
    const newColour = call.request.colour.toLowerCase();
    if (ALLOWED_COLOURS.includes(newColour)) {
        currentColour = newColour;
        console.log('Colour:', currentColour);
        callback(null, { message: `Colour changed to ${newColour}` });
    } else {
        callback({ code: grpc.status.INVALID_ARGUMENT, message: 'Invalid colour. Allowed colours: ' + ALLOWED_COLOURS.join(', ') });
    }
}


//add gRPC service methods to the server
server.addService(smartHomeProto.Lighting.service, {
    LightStatus: getLightStatus,
    BrightnessStatus: getBrightnessStatus,
    ColourStatus: getColourStatus,
    ChangeColour: changeColour,
    AdjustBrightness: adjustBrightness,
    ChangeLightStatus: changeLightStatus
});

//bind and start the server
server.bindAsync('127.0.0.1:50051', grpc.ServerCredentials.createInsecure(), (err, port) => {
    if (err) {
        console.error('Error starting server:', err);
```

```
    } else {
        console.log('Server started successfully, listening on port', port);
    }
});
```

lighting_client.js

```
// Purpose: Client for the smart lighting service. It allows the user to interact with the lighting service by querying the current status of the light, turning the light on or off, adjusting the brightness level, and changing the colour of the light.



//import modules
const grpc = require('@grpc/grpc-js');
const chalk = require('chalk');
const protoLoader = require('@grpc/proto-loader');
const inquirer = require('inquirer');

//load the protocol buffer definition
const packageDefinition = protoLoader.loadSync('./lighting_service/smart_lighting.proto', {});
const smartLighting = grpc.loadPackageDefinition(packageDefinition).smart_home;

function main() {
  return async function () {
    try {
      //create a new gRPC client instance
      const client = new smartLighting.Lighting('localhost:50051', grpc.credentials.createInsecure());
      let continueQuery = true;

      do {
        //blank line for spacing
        console.log();
```

```
    //prompt user to select an option
    const { choice } = await inquirer.prompt({
      type: 'list',
      name: 'choice',
      message: chalk.yellow('Hue Lighting:'),
      choices: [
        'Light Status',
        'Turn Light On/Off',
        'Brightness Status',
        'Adjust Brightness',
        'Colour Status',
        'Change Colour',
        new inquirer.Separator(),
      ],
    });

    //handle user's choice
    switch (choice) {
      case 'Light Status':
        //get light status
        await new Promise((resolve, reject) => {
          client.LightStatus({}, (error, response) => {
            if (error) {
              console.error(error);
              reject(error);
            } else {
              console.log('Light status:', chalk.yellow(response.message));
              console.log();
              resolve();
            }
          });
        });
```

```
            break;
          case 'Brightness Status':
            //get brightness status
            await new Promise((resolve, reject) => {
              client.BrightnessStatus({}, (error, response) => {
                if (error) {
                  console.error(error);
                  reject(error);
                } else {
                  console.log('Brightness level:', response.brightnessLevel, '%');
                  console.log();
                  resolve();
                }
              });
            });
            break;
          case 'Colour Status':
            //get colour status
            await new Promise((resolve, reject) => {
              client.ColourStatus({}, (error, response) => {
                if (error) {
                  console.error(error);
                  reject(error);
                } else {
                  console.log('Current colour:', chalk.yellow(response.colour));
                  console.log();
                  resolve();
                }
              });
            });
            break;
          case 'Turn Light On/Off':
```

```
        //turn light on or off
        const { isOnInput } = await inquirer.prompt({
          type: 'list',
          name: 'isOnInput',
          message: 'Turn light on or off:',
          choices: ['On', 'Off'],
        });
        const isOn = isOnInput === 'On';

        // check current light status before changing
        let currentStatus;
        await new Promise((resolve, reject) => {
          client.LightStatus({}, (error, response) => {
            if (error) {
              console.error(error);
              reject(error);
            } else {
              currentStatus = response.isOn;
              console.log();
              resolve();
            }
          });
        });

        if ((isOn && currentStatus === 'On') || (!isOn && currentStatus === 'Off')) {
          console.log(chalk.yellow(`The light is already ${currentStatus}.`));
        } else {
          await new Promise((resolve, reject) => {
            client.ChangeLightStatus({ isOn }, (error, response) => {
              if (error) {
                console.error(error);
                reject(error);
```

```
            } else {
              console.log(chalk.yellow(response.message));
              resolve();
            }
          });
        });
      }
      break;
    case 'Adjust Brightness':
      //adjust brightness
      const { newBrightnessLevel } = await inquirer.prompt({
        type: 'input',
        name: 'newBrightnessLevel',
        message: 'Enter new brightness level(0-100):',
        validate: function (value) {
          var valid = !isNaN(parseFloat(value)) && isFinite(value) && value >= 0 &&
value <= 100;
          return valid || 'Please enter a number between 0 and 100';
        },
      });

      await new Promise((resolve, reject) => {
        client.AdjustBrightness({ newBrightnessLevel:
parseInt(newBrightnessLevel, 10) }, (error, response) => {
          if (error) {
            console.error(error);
            reject(error);
          } else {
            console.log(chalk.yellow(response.message));
            resolve();
          }
        });
```

```
        });
        break;
      case 'Change Colour':
        //change colour
        const { newColour } = await inquirer.prompt({
          type: 'list',
          name: 'newColour',
          message: 'Enter new colour:',
          choices: ['white', 'red', 'green', 'blue', 'yellow', 'orange', 'purple', 'pink'],
        });

        await new Promise((resolve, reject) => {
          client.ChangeColour({ colour: newColour }, (error, response) => {
            if (error) {
              console.error(error);
              reject(error);
            } else {
              console.log(chalk.yellow(response.message));
              console.log();
              resolve();
            }
          });
        });
        break;
      default:
        console.log(chalk.red('Invalid choice. Please select an option from the list.'));
        break;
    }
    //prompt user if they want to select another query
    const { anotherQuery } = await inquirer.prompt({
      type: 'list',
      name: 'anotherQuery',
```

```
            message: 'Do you want to select another query?',
            choices: ['Yes', 'No']
        });

        //continue or exit based on user's choice
        continueQuery = anotherQuery === 'Yes';
      } while (continueQuery);
    } catch (error) {
      //handle error
      if (error.code === grpc.status.UNAVAILABLE) {
        console.error(chalk.red('Error: Server is unavailable. Please try again later.'));
      } else {
        console.error(chalk.red('Error:', error.message));
      }
    }
  }

//export the main function for main_client.js to call
module.exports.main = main;
```

doorbell_server.js

```
//Purpose: Server for the smart doorbell service. It provides the implementation for the doorbell service methods, such as live video feed, today's events, silent mode status, and toggling silent mode.

//import modules
const grpc = require('@grpc/grpc-js');
const protoLoader = require('@grpc/proto-loader');
const PROTO_PATH = "./doorbell_service/smart_doorbell.proto";
const packageDefinition = protoLoader.loadSync(PROTO_PATH);
const smartHomeProto = grpc.loadPackageDefinition(packageDefinition).smart_home;
```

```
//initialize default state
let silentMode = false;

//create a new gRPC server
const server = new grpc.Server();

//gRPC service method for LiveVideoFeed
function liveVideoFeed(call) {
  console.log('Request for Live Video Feed received');
  const videoUrl = "https://rb.gy/tqh0js";
  const description = "Dublin Pub Live Stream";
  call.write({ liveVideoUrl: videoUrl });
  call.write({ description: description });
  call.end();
}

//gRPC service method for TodaysEvents
function todaysEvents(call) {
  console.log('Request for Today\'s Events received');
  const events = ["#1 event", "#2 event", "#3 event"];
  events.forEach(event => {
    const detectionType = ["human", "animal"];
    const randomType = detectionType[Math.floor(Math.random() *
detectionType.length)];
    const eventStatement = `${event} with type: ${randomType} on ${new
Date().toISOString().slice(0, 10)}`;
    call.write({ eventStatement });
  });
  call.end();
}
```

```
//gRPC service method for SilentModeStatus
function silentModeStatus(call, callback) {
  console.log('Request for Silent Mode Status received');
  callback(null, { isSilentModeOn: silentMode });
}

//gRPC service method for ToogleSilentMode
function toggleSilentMode(call, callback) {
  console.log('Request for Toggling Silent Mode received');
  silentMode = call.request.toggle;
  const message = silentMode ? "Silent mode turned on" : "Silent mode turned off";
  console.log('Silent Mode:', message);
  callback(null, { message });
}



//add gRPC service methods to the server
server.addService(smartHomeProto.Doorbell.service, {
    LiveVideoFeed: liveVideoFeed,
    TodaysEvents: todaysEvents,
    SilentModeStatus: silentModeStatus,
    ToggleSilentMode: toggleSilentMode
});

//bind and start server
server.bindAsync('127.0.0.1:50052', grpc.ServerCredentials.createInsecure(), (err,
port) => {
    if (err) {
        console.error('Error starting server:', err);
    } else {
        console.log('Server started successfully, listening on port', port);
```

```
    }
});
```

doorbell_client.js

```
// Purpose: Client for the smart lighting service. It allows the user to interact with the lighting service by querying the current status of the light, turning the light on or off, adjusting the brightness level, and changing the colour of the light.

//import modules
const grpc = require('@grpc/grpc-js');
const chalk = require('chalk');
const protoLoader = require('@grpc/proto-loader');
const inquirer = require('inquirer');

//load the protocol buffer definition
const packageDefinition = protoLoader.loadSync('./doorbell_service/smart_doorbell.proto', {});
const smartDoorbell = grpc.loadPackageDefinition(packageDefinition).smart_home;


function main() {
  return async function () {
    try {
      //create a new gRPC client instance
      const client = new smartDoorbell.Doorbell('localhost:50052', grpc.credentials.createInsecure());
      let continueQuery = true;

      do {
        //blank line for spacing
        console.log();

        //prompt user to select an option
```

```
    const { choice } = await inquirer.prompt({
      type: 'list',
      name: 'choice',
      message: chalk.yellow('Eufy Doorbell:'),
      choices: [
        'Live Video Feed',
        'Today\'s Events',
        'Silent Mode Status',
        'Toggle Silent Mode',
        new inquirer.Separator(),
      ],
    });

    //handle user's choice
    switch (choice) {
      case 'Live Video Feed':
        //live video feed
        await new Promise((resolve, reject) => {
          const liveVideoFeedStream = client.LiveVideoFeed({});
          liveVideoFeedStream.on('data', response => {
            if (response.liveVideoUrl) {
              console.log('Live Video Feed URL:', chalk.yellow(response.liveVideoUrl));
            }
            if (response.description) {
              console.log('Description:', chalk.yellow(response.description));
              console.log();
            }
          });
          liveVideoFeedStream.on('error', error => {
            reject(error);
          });
          liveVideoFeedStream.on('end', () => {
```

```
            console.log();
            resolve();
          });
        });
        break;
      case 'Today\'s Events':
        //today's events
        await new Promise((resolve, reject) => {
          const todaysEventsStream = client.TodaysEvents({});
          todaysEventsStream.on('data', event => {
            console.log();
            console.log('Event:', chalk.yellow(event.eventStatement));
          });
          todaysEventsStream.on('error', error => {
            reject(error);
          });
          todaysEventsStream.on('end', () => {
            console.log();
            console.log('End of events');
            resolve();
          });
        });
        break;
      case 'Silent Mode Status':
        //silent mode status
        await new Promise((resolve, reject) => {
          client.SilentModeStatus({}, (error, response) => {
            if (error) {
              console.error(error);
              reject(error);
            } else {
              const status = response.isSilentModeOn ? 'on' : 'off';
```

```
                console.log('Silent Mode Status:', chalk.yellow(status));
                console.log();
                resolve();
              }
            });
          });
          break;
        case 'Toggle Silent Mode':
          const { toggleInput } = await inquirer.prompt({
            type: 'list',
            name: 'toggleInput',
            message: 'Toggle silent mode:',
            choices: ['On', 'Off'],
          });
          const toggle = toggleInput === 'On';

          //toggle silent mode
          await new Promise((resolve, reject) => {
            client.ToggleSilentMode({ toggle }, (error, response) => {
              if (error) {
                console.error(error);
                reject(error);
              } else {
                console.log(chalk.yellow(response.message));
                console.log();
                resolve();
              }
            });
          });
          break;
        default:
          console.log(chalk.red('Invalid choice. Please select an option from the list.'));
```

```
            break;
        }

        //prompt user if they want to select another query
        const { anotherQuery } = await inquirer.prompt({
          type: 'list',
          name: 'anotherQuery',
          message: 'Do you want to select another query?',
          choices: ['Yes', 'No']
        });

        //continue or exit based on user's choice
        continueQuery = anotherQuery === 'Yes';

      } while (continueQuery);
    } catch (error) {
      //handle errors
      if (error.code === grpc.status.UNAVAILABLE) {
        console.error(chalk.red('Error: Server is unavailable. Please try again later.'));
      } else {
        console.error(chalk.red('Error:', error.message));
      }
    }
  }
}

//export main function for main_client.js to call
module.exports.main = main;
```

thermostat_server.js

```
//import modules
const grpc = require('@grpc/grpc-js');
const protoLoader = require('@grpc/proto-loader');

//load the protocol buffer definition
const packageDefinition = protoLoader.loadSync('./thermostat_service/smart_thermostat.proto', {});
const smartThermostat = grpc.loadPackageDefinition(packageDefinition).smart_home;

//initialize default states
let currentTemperature = 20.5;
let boostTemperature = 20;
let isBoostActive = false;
let boostTimeRemaining = 15;
let isHotWaterOn = false;

//create a new gRPC server
const server = new grpc.Server();

//method to get the current temperature
function temperatureStatus(call, callback) {
  console.log('Request for Temperature Status received');
  console.log('Sending Temperature Status:', currentTemperature);
  callback(null, { currentTemperature: currentTemperature });
}

//method to set the temperature
function setTemperature(call, callback) {
  console.log('Request to set temperature received');
  currentTemperature = call.request.setTemperature;
```

```
  console.log('Temperature set to:', currentTemperature + '\u00B0C');
  callback(null, { message: `Temperature set to ${currentTemperature}\u00B0C` });
}

//method to get the boost status
function boostStatus(call, callback) {
  console.log('Request for Boost Status received');
  let boostStatusMessage = 'Boost Status: ';
  if (isBoostActive) {
    boostStatusMessage += ` ${boostTemperature}\u00B0C / ${boostTimeRemaining} minutes`;
  } else {
    boostStatusMessage += 'Off';
  }
  console.log(boostStatusMessage);
  callback(null, { isBoostActive: isBoostActive, boostTemperature: boostTemperature, boostTimeRemaining: boostTimeRemaining });
}

//method to manage the boost
function manageBoost(call, callback) {
  console.log('Request to manage boost received');
  if (call.request.boostTemperature && call.request.boostTime) {
    isBoostActive = true;
    boostTemperature = call.request.boostTemperature; // update boostTemperature
    boostTimeRemaining = call.request.boostTime;
    console.log(`Boost adjusted. Temperature: ${boostTemperature}\u00B0C, Time remaining: ${boostTimeRemaining} minutes`);
    callback(null, { message: `Boost adjusted. Temperature: ${boostTemperature}\u00B0C, Time remaining: ${boostTimeRemaining} minutes` });
  } else {
    callback({
      code: grpc.status.INVALID_ARGUMENT,
```

```
    message: 'Invalid request. boostTemperature and boostTime are required to manage boost.',
  });
 }
}

//method to set the boost status
function setBoost(call, callback) {
 console.log('Request to set boost received');
 isBoostActive = call.request.setBoost;
 if (!isBoostActive) {
  currentTemperature = 20.5;
  boostTimeRemaining = 30;
 }
 console.log('Boost set to:', isBoostActive ? 'On' : 'Off');
 callback(null, { message: `Boost set to: ${isBoostActive ? 'On' : 'Off'}` });
}

//method to adjust the boost
function adjustBoost(call, callback) {
 console.log('Request to adjust boost received');
 isBoostActive = true;
 boostTemperature = call.request.boostTemperature; // update boostTemperature
 boostTimeRemaining = call.request.boostTime;
 console.log(`Boost adjusted. Temperature: ${boostTemperature}\u00B0C / Time remaining: ${boostTimeRemaining}`);
 callback(null, { message: `Boost adjusted. Temperature: ${boostTemperature}\u00B0C, Time remaining: ${boostTimeRemaining} minutes` });
}

//method to get the hot water status
function hotWaterStatus(call, callback) {
 console.log('Request for Hot Water Status received');
```

```
  console.log('Sending Hot Water Status:', isHotWaterOn ? 'On' : 'Off');
  callback(null, { message: `${isHotWaterOn ? 'On' : 'Off'}` });
}

//method to set the hot water status
function setHotWater(call, callback) {
  console.log('Request to set hot water status received');
  isHotWaterOn = call.request.setHotWater;
  console.log('Hot water status set to:', isHotWaterOn ? 'On' : 'Off');
  callback(null, { message: `Hot water status set to ${isHotWaterOn ? 'On' : 'Off'}` });
}

// add the service methods to the server
server.addService(smartThermostat.Thermostat.service, {
  TemperatureStatus: temperatureStatus,
  SetTemperature: setTemperature,
  BoostStatus: boostStatus,
  ManageBoost: manageBoost,
  SetBoost: setBoost,
  AdjustBoost: adjustBoost,
  HotWaterStatus: hotWaterStatus,
  SetHotWater: setHotWater
});

// bind and start the server
server.bindAsync('127.0.0.1:50053', grpc.ServerCredentials.createInsecure(), (err,
port) => {
  if (err) {
    console.error('Error starting server:', err);
  } else {
    console.log('Server started successfully, listening on port', port);
  }
```

```
});
```

thermostat_client.js

```
// Purpose: Client for the smart thermostat service. It allows the user to interact with the thermostat service by querying the current temperature, setting the temperature, checking the boost status, managing the boost, checking the hot water status, and setting the hot water on or off.


//import modules
const grpc = require('@grpc/grpc-js');
const chalk = require('chalk');
const protoLoader = require('@grpc/proto-loader');
const inquirer = require('inquirer');

//load the protocol buffer definition
const packageDefinition = protoLoader.loadSync('./thermostat_service/smart_thermostat.proto', {});
const smartThermostat = grpc.loadPackageDefinition(packageDefinition).smart_home;



function main() {
  return async function () {
    try {
      //create a new gRPC client instance
      const client = new smartThermostat.Thermostat('localhost:50053', grpc.credentials.createInsecure());

      let continueQuery = true;

      do {
        //blank line for spacing
        console.log();
        //prompt user to select an option
```

```
    const { choice } = await inquirer.prompt({
      type: 'list',
      name: 'choice',
      message: chalk.yellow('Hive Thermostat (\u00B0C):'),
      choices: [
        'Temperature Status',
        'Set Temperature',
        'Boost Status',
        'Manage Boost',
        'Hot Water Status',
        'Set Hot Water',
        new inquirer.Separator(),
      ],
    });

    //handle user's choice
    switch (choice) {
      case 'Temperature Status':
        //get temperature status
        await new Promise((resolve, reject) => {
          client.TemperatureStatus({}, (error, response) => {
            if (error) {
              console.error(error);
              reject(error);
            } else {
              console.log('Temperature Status:', chalk.yellow(response.currentTemperature + '\u00B0C'));
              resolve();
            }
          });
        });
        break;
```

```
        case 'Set Temperature':
          //set temperature
          const { setTemperature } = await inquirer.prompt({
            type: 'input',
            name: 'setTemperature',
            message: 'Enter new temperature(\u00B0C):',
            validate: function (value) {
              var valid = !isNaN(parseFloat(value));
              if (!valid) {
                return 'Please enter a valid number';
              }
              var temp = parseFloat(value);
              if (temp < 5 || temp > 35) {
                return 'Please enter a temperature between 5\u00B0C and 35\u00B0C';
              }
              return true;
            },
          });

          await new Promise((resolve, reject) => {
            client.SetTemperature({ setTemperature: parseFloat(setTemperature) },
(error, response) => {
              if (error) {
                console.error(error);
                reject(error);
              } else {
                console.log(chalk.yellow(response.message));
                resolve();
              }
            });
          });
          break;
```

```
        case 'Boost Status':
          //get boost status
          await new Promise((resolve, reject) => {
            client.BoostStatus({}, (error, response) => {
              if (error) {
                console.error(error);
                reject(error);
              } else {
                let boostStatusMessage = 'Boost Status: ';
                if (response.isBoostActive) {
                  boostStatusMessage += ` ${response.boostTemperature}\u00B0C / ${response.boostTimeRemaining} minutes`;
                } else {
                  boostStatusMessage += 'Off';
                }
                console.log(chalk.yellow(boostStatusMessage));
                resolve();
              }
            });
          });
          break;
        case 'Manage Boost':
          //manage boost
          const { manageBoostChoice } = await inquirer.prompt({
            type: 'list',
            name: 'manageBoostChoice',
            message: 'Select an operation:',
            choices: ['Set Boost On/Off', 'Adjust Boost'],
          });

          switch (manageBoostChoice) {
            case 'Set Boost On/Off':
```

```
            //set boost
            const { setBoost } = await inquirer.prompt({
              type: 'list',
              name: 'setBoost',
              message: 'Turn boost on or off:',
              choices: ['On', 'Off'],
            });

            await new Promise((resolve, reject) => {
              client.SetBoost({ setBoost: setBoost === 'On' }, (error, response) => {
                if (error) {
                  console.error(error);
                  reject(error);
                } else {
                  console.log(chalk.yellow(response.message));
                  resolve();
                }
              });
            });
            break;
          case 'Adjust Boost':
            //adjust boost
            const { boostTemperature, boostTime } = await inquirer.prompt([{
              type: 'input',
              name: 'boostTemperature',
              message: 'Enter boost temperature(\u00B0C):',
              validate: function (value) {
                var valid = !isNaN(parseFloat(value));
                if (!valid) {
                  return 'Please enter a valid number';
                }
                var temp = parseFloat(value);
```

```
                if (temp < 5 || temp > 35) {
                  return 'Please enter a temperature between 5\u00B0C and 35\u00B0C';
                }
                return true;
              },
            }, {
              type: 'input',
              name: 'boostTime',
              message: 'Enter boost time(minutes):',
              validate: function (value) {
                var valid = !isNaN(parseFloat(value));
                if (!valid) {
                  return 'Please enter a valid time in minutes between 15 and 240';
                }
                var time = parseFloat(value);
                if (time < 15 || time > 240) {
                  return 'Please enter a time between 15 minutes and 240 minutes';
                }
                return true;
              },
            }]);

            await new Promise((resolve, reject) => {
              client.ManageBoost({ boostTemperature: parseFloat(boostTemperature), boostTime: parseFloat(boostTime) }, (error, response) => {
                if (error) {
                  console.error(error);
                  reject(error);
                } else {
                  console.log(chalk.yellow(response.message));
                  resolve();
                }
```

```
            });
          });
          break;
        }
        break;
      case 'Hot Water Status':
        //get hot water status
        await new Promise((resolve, reject) => {
          client.HotWaterStatus({}, (error, response) => {
            if (error) {
              console.error(error);
              reject(error);
            } else {
              console.log('Hot Water Status:', chalk.yellow(response.message));
              resolve();
            }
          });
        });
        break;
      case 'Set Hot Water':
        //set hot water
        const { setHotWater } = await inquirer.prompt({
          type: 'list',
          name: 'setHotWater',
          message: 'Turn hot water on or off:',
          choices: ['On', 'Off'],
        });

        await new Promise((resolve, reject) => {
          client.SetHotWater({ setHotWater: setHotWater === 'On' }, (error, response)
=> {
            if (error) {
```

```
                console.error(error);
                reject(error);
            } else {
                console.log(chalk.yellow(response.message));
                resolve();
            }
          });
        });
        break;
      default:
        console.log(chalk.red('Invalid choice. Please select an option from the list.'));
        break;
    }
    //prompt user if they want to select another query
    const { anotherQuery } = await inquirer.prompt({
      type: 'list',
      name: 'anotherQuery',
      message: 'Do you want to select another query?',
      choices: ['Yes', 'No']
    });
    //continue or exit based on user's choice
    continueQuery = anotherQuery === 'Yes';

  } while (continueQuery);
 } catch (error) {
  //handle errors
  if (error.code === grpc.status.UNAVAILABLE) {
    console.error(chalk.red('Error: Server is unavailable. Please try again later.'));
  } else {
    console.error(chalk.red('Error:', error.message));
  }
 }
```

```
  }
}

//export main function for main_client.js to call
module.exports.main = main;
```

main_client.js

```
const thermostatClient = require('./thermostat_service/thermostat_client');
const lightClient = require('./lighting_service/lighting_client');
const doorClient = require('./doorbell_service/doorbell_client');
console.log('Doorbell');
const inquirer = require('inquirer');

async function main() {
  let continueQuery = true;

  do {
    const { clientChoice } = await inquirer.prompt({
      type: 'list',
      name: 'clientChoice',
      message: 'Which client do you want to interact with?',
      choices: ['Hive Thermostat', 'Hue Lighting', 'Eufy Doorbell'],
    });

    switch (clientChoice) {
      case 'Hive Thermostat':
        console.log('Loading Thermostat Client...');
        await thermostatClient.main()();
        break;

      case 'Hue Lighting':
        console.log('Loading Light Client...');
```

```
        await lightClient.main()();
        break;
      case 'Eufy Doorbell':
        console.log('Loading Doorbell Client...');
        await doorClient.main()();
        break;
      default:
        console.log('Invalid choice. Please select an option from the list.');
        break;
    }

    const { anotherQuery } = await inquirer.prompt({
      type: 'list',
      name: 'anotherQuery',
      message: 'Do you want to select another service?',
      choices: ['Yes', 'No']
    });

    continueQuery = anotherQuery === 'Yes';
  } while (continueQuery);
}

main();
```