Software Quality

CMPUT 402 – University of Alberta

Project for Group Assignments: Tartan Smart Home Platform

Background

The purpose of this project is to provide a model problem for you to practice quality assurance and control techniques and methods presented in the course. You will be extending and evaluating an existing software project. The goals of this course are software quality *analysis*, not software development. The coding tasks are relatively straightforward and meant to provide a basis for evaluating quality in a software project. The software is in a rough shape and under-documented, not unusual for industrial software projects and early prototypes.

The system in this document as well as this document have been developed at Carnegie Mellon University by J. Gennari, C. Kästner, S. Echeverria, M. Velez. The group projects in this course are adapted from the original assignments on this system.

Project Overview

Internet of Things

Since the Internet's humble beginnings (with the emergence/adoption of TCP/IP on January 1, 1983, aka internet "flag day"), it has grown into a ubiquitous presence, touching nearly every person on planet Earth. What's next? Many experts indicate that the Internet of Things (IoT) will transform the Earth as much, or more than, the Internet itself. The IoT is about connecting sensors and actuators embedded in everyday things to the Internet. Simple temperature control, lighting, and security are common modern applications. These are the first baby-steps of IoT (IoT1). In the near future, vast systems will collect data and the IoT will use this information to provide predictive services (IoT2 and Iot3).

Internet and web applications have been following a person-to-machine (P2M) paradigm that originated with the earliest days of distributed computing. A user (person) submits requests to a server (machine) for services and data (like a person who orders a book on Amazon, performs a Google search, etc). The clients (persons) can be fixed or mobile, but the data sources are generally fixed. IoT1 applications follow this model as well. For example in 1983, members of CMU's School of Computer Science modified a Coke vending machine with micro-switches and a small computer and connected it to a PDP-10 computer, allowing faculty and students to query the machine to see what kinds of pop were available.

However, the IoT is evolving beyond simple P2M to include machine-to-person (M2P), and machine-to-machine (M2M) models. In M2P, machines will contact humans to assist in daily activities. In M2M, machines will communicate with one another to provide services to humans without ever involving or informing them. This will enable complex services far beyond P2M: machines will access vast amounts of data to anticipate needs, predict trouble, and identify opportunities to help humans.

IoT has revolutionary potential: smart homes, smart grid, energy management, remote medicine, smart logistics systems and packages, driverless cars and smart highways, driverless trains, and so on. However, there are many challenges to realizing this vision. The current Internet infrastructure and middleware is inadequate for M2P and M2M applications. In the IoT world, data will not be fixed and isolated in a “backend” server as it often is today. Instead, data sources will be more distributed and will be mobile (humans walking, ships sailing, etc.). New privacy and security issues will emerge. IoT devices must be recognized (discovery) and associated with an owner (registration) securely and privately. New physical security standards will have to be developed beyond simple data security rules (consider Asimov’s 3 laws of robotics!). There will be unprecedented scaling demands on the legacy Internet infrastructure, making IPV6 mandatory. There is and will continue to be a lot of fragmentation in the industry: different hardware, operating systems, protocols, middleware, infrastructure, etc. Legacy devices will have to be supported for tens of years rather than the two to four years of many modern devices.

We will use today’s IoT as a context for our project. Please note that purpose of this project is *not* to make you an IoT or smart home expert. Instead, the IoT context is being used as a model problem for you to practice the testing and quality assurance techniques and methods presented in the course. Because the goal of this course is *software* quality, we will not focus on hardware evaluation.

Tartan Inc.’s Smart Home Platform

Assume that your team is working for an organization, Tartan Inc. that is poised to enter the IoT market. As part of their business strategy Tartan Inc. has acquired a smart home controller made by a small company that has created smart thermostat devices in the pre-internet area and begun the process of converting it to a modern, service-based approach. The existing software originally controlled a HVAC unit (with sophisticated hard-coded logic and many user-programmable controls) but has recently been changed to also consolidate IoT elements installed in a home and/or business; it is called the *IoT Controller*. The company strategy is to compete with Nest and similar platforms and to eventually provide intelligent and convenient services millions of homes. To that end, the IoT controller will enable end users to communicate with sensors and control actuators installed in a home or business via an internet-connected system.

The course instructors represent both business development and the product manager.

Strategy and Current Status

As is typical in acquisitions there are challenges that must be overcome before the software is ready for market. The business strategy for Tartan Inc. is to create an extensible service-based platform where various data sources can be combined to enables smart controls. One can envision this approach enabling the platform to adapt as new smart home technologies become available. To put another way, Tartan Inc. wants to become a sole provider for all manner of smart home devices.

The Tartan Platform development team is preparing to release the first version of their software. However, before the platform is made available, it must be thoroughly evaluated to ensure that it meets its requirements and necessary quality thresholds and that the intended extensions are feasible at the desired quality. Your goal, which you will pursue over the course of the semester, is to conduct this evaluation.

System Requirements (excerpt)

Hardware

The hardware for the IoT Controller is still in development, but there is a simulator available that implements the proprietary hardware protocol and rudimentary behaviors of the IoT Controller hardware. Assume that the following hardware elements are (or will soon be) available:

- Home lighting: various lights and switches
- Temperature and humidity sensors.
- Actuators to automatically lock and unlock doors.
- A proximity sensor that can detect when a house is occupied.
- An alarm system that can be enabled/disabled with a passcode.
- An Heating Ventilation and Air Conditioning (HVAC) system that includes a heater, a chiller, and a dehumidifier.

Acquired IoT Controller

The IoT Controller has a number of functional requirements associated with it. Firstly, the controller shall require the user to login to the house control panel using a strong username and password. The software should detect attempts to bypass unauthorized access and take appropriate countermeasures. The software should also maintain a log of activity for future analysis. The software should provide accurate readings for all sensors and actuators in the house (lights, alarms, door, etc.). Of course, the house should use the sensors and actuators properly. This means the following:

Requirements

Table 1 lists the detailed original requirements for the IoT Controller at the time of acquisition by Tartan Inc.

Table 1: Original IoT Controller Requirements

Requirement Description

The IoT Controller shall require the user to login to the house control panel using a username and password. The password has the following requirements: · Minimum length: 8 characters · At least one upper case character · At least one number · At least one symbol
The IoT Controller shall not allow the user to attempt to log in after three failed attempts.
The IoT Controller shall provide a service that allows users to determine the temperature/humidity, turn on and off lights, open and close the door, enable the alarm, and determine if anyone is home.
The IoT Controller should store sensor values, log all actions in the house, and allow the user to review the log at any time.
The IoT Controller shall allow the user to clear the log at any time.
The IoT Controller shall activate the alarm; the door is manually opened while the alarm is enabled or the house is suddenly occupied while alarm is enabled.
The IoT Controller shall detect when the house has been vacant for more than a user-specified amount of time. Once this time period has passed the system shall close the door, turn off the light, and enable the alarm.
The alarm must be disabled by the user in person by entering a user-defined passcode.
The IoT Controller shall automatically turn on the light when the house becomes occupied.

The IoT Controller shall allow the user to set the house temperature. The minimum temperature allowed is 50 degrees Fahrenheit (10 degrees Celsius). The maximum temperature allowed is 80 degrees Fahrenheit (27 degrees Celsius).
The IoT Controller shall allow new users to be added to the system. New users must provide passwords that adhere to the requirements in UC01.
The IoT Controller shall decide whether the house should heat or cool the house. The system shall prevent the heater and air conditioner from running at the same time.
The IoT Controller shall allow the user to turn on and turn off the dehumidifier. The dehumidifier can only be activated with the air conditioner.

Reliability
In the event of losing the connection to the internet or sensors or actuators during normal operation, the Tartan Platform detects the lost connection, logs the event, and reverts to the last consistent state. The house will always revert to the last consistent state if the connection is out for too long.

Consistency
When a user attempts to change the house state during normal operation, the Tartan Platform evaluates the request to ensure the new state is allowed and makes necessary modifications. The Tartan Platform always prevents the house from entering an inconsistent state (e.g., heating and cooling at the same time).

Performance
Keeping the system synchronized is very important. This means that the user interface accurately reflects the state of the house hardware at any given moment. The house must never out of sync more than a few seconds.

Security
The system should not allow unauthorized access of house resources data during normal operation. The system will prevent unauthorized access 100% of the time.
Privacy
Data collected from, and shared between smart homes controlled by the Tartan Smart Home Platform must be anonymized to protect homeowner privacy.

Extensibility

The smart home and home automation industry is quickly evolving. To keep up with innovation Tartan Inc. wants to be able to easily integrate new technologies. Specifically, new sensors, appliances, and smart home elements should be easily integrated with the Tartan Smart home. Integration of new elements should not take more than a day of effort. Another dimension of extensibility is the ability to support multiple houses so that information from multiple houses should be available to to make decisions.

Automation

New data sources and technologies to automate common home tasks should be supported. The platform should enable and manage decision making in the IoT Controller. Part of this requirement is the ability to support and share (anonymized) data across multiple home instances. New automation schemes should be deployable in not more than five minutes.

Tartan Smart Home Service Platform

The primary drivers for the services layer are extendibility, automation, and privacy. These requirements are meant to support business strategies above and will allow later transitions and innovations to build in intelligence through machine learning algorithms. Tartan Inc. has selected a RESTful web service architecture as the primary design strategy for the smart home platform. The have also selected Java as the primary technology due to its maturity and widespread support for web services. In particular, the Smart Home Platform will be based on the Dropwizard[1] web service framework. Table 2 describes the requirements for the Tartan Smart Home Platform.

Table 2: Tartan Smart Home Service Platform Requirements

Requirement Description
The Tartan Smart Home Platform shall require the user to login to the house control panel using a username and password. The password has the following requirements are still under consideration.
The Tartan Smart Home Platform shall allow a user to fetch the state and update the state remotely.

1 See http://www.dropwizard.io

The Tartan Smart Home Platform shall support multiple houses.
The Tartan Smart Home Platform shall support data logging at configurable intervals. The data logged shall reflect the state of the house.
The Tartan Smart Home Platform shall fully integrate with the IoT Controller. That is, the platform will fully support the functionality of the IoT Controller.

System Design

The Tartan Smart Home system is a distributed system composed of multiple elements. Figure 1 shows the end-to-end runtime view of the system. The **Tartan Smart Home Platform** is the heart of the system. This element connects the other elements using various protocols. Specifically, the platform enables RESTful communication with **external clients** and supports the custom **house communication protocol** with the **House Hubs** installed in the individual houses (see below for details). The platform is also responsible for logging house history and basic configuration. The Tartan Smart Home Platform has two major components: The **IoT Controller** and the **Tartan Smart Home Service**. These elements are described below.

Note that the IoT Controller is acquired software and many of the design choices reflect constraints and choices that were made when it was part of a different system. Unfortunately, this situation was unavoidable.

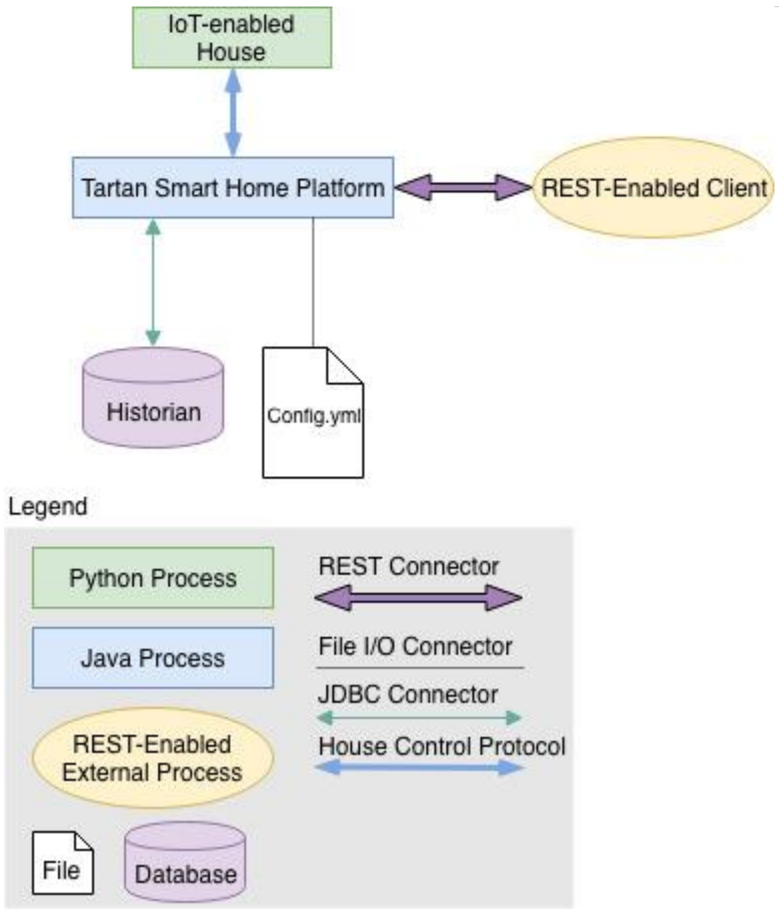


Figure 1: Tartan Smart Home System design

Managing House State

The state of the Tartan Platform is controlled through a collection of state variables. Combinations of these variables represent the overall state of the house. In general, state variables are boolean values that indicate whether a house control is enabled/on or disabled/off; however, some state variables are numeric and others are strings. Table 3 shows specification for the current set of state variables.

Table 3: State variables for the system

State Variable	Type	Description
tempReading	Integer	The value of the temperature sensor reading.
humidifierState	Boolean	True if the humidifier is on; false otherwise.

humidityReading	Integer	The value of the humidity sensor reading.
doorState	Boolean	True if the door is open; false otherwise.
lightState	Boolean	True if the light is on; false otherwise.
proximityState	Boolean	True if the house is occupied; false otherwise.
alarmState	Boolean	True if the alarm is enabled; false otherwise.
alarmActiveState	Boolean	True if the alarm is activated (i.e. alarm is sounding); false otherwise.
hvacSetting	String	If the HVAC is set to heat the house, this state is “Heater”. If the HVAC is set to cool the house, the state is “Chiller”.
heaterOnState	Boolean	True if the heater is on; false otherwise.
chillerOnState	Boolean	True if the air conditioner is on; false otherwise.

IoT Controller Software Design

The IoT Controller is a smart-home control system capable of controlling the following home systems:

- Alarm system that includes sensors to determine if a door is opened and the home is occupied.
- Smart lighting system capable of being remotely turned on or off.
- HVAC system that includes a dehumidifier.
- Actuators and sensors to close or open a door.

The Tartan Platform software uses a client/server design to communicate with the house. In this arrangement, the software running on the house is the server and the IoT Controller is the client. A direct client/server connection ensures that data is processed efficiently and reliably. The IoT Controller software manages user authentication for the system as a way to centralize security decisions. The UML class diagram design of the IoT Controller is shown in Figure 2. Table 4 describes the purpose of each class.

Table 4: Class descriptions for IoT Controller

Class	**Description**
TControlManager	This class is the focal point of state decisions. When the user requests a new state for the house, this class pre-processes the request to determine if it is allowed. If the new state is not allowed, then this class will put the house

	into a valid (previously accepted) state. If the new state is valid, then this class issues a state update.
IoTConnectManager	This class manages connections to the IoT House. That is, this class retrieves the active connection when needed. The IoTConnectManager ensures that only one connection to a house is made at a time.
IoTConnection	This class handles the low-level networking for the Tartan Platform.
IoTValues	Constant values used by the system. This class is abstract and cannot be instantiated directly.
LoginAttemptsExceededException	This is an exception used to indicate that the maximum number of login attempts has been exceeded.

The IoTControlManager class contains all the logic to evaluate and maintain state consistency. This class also maintains the last accepted state enabling the system to revert to a previous state as needed to promote reliability. Connections to the IoT house are controlled via the IoTConnectManager, which promotes performance and security.

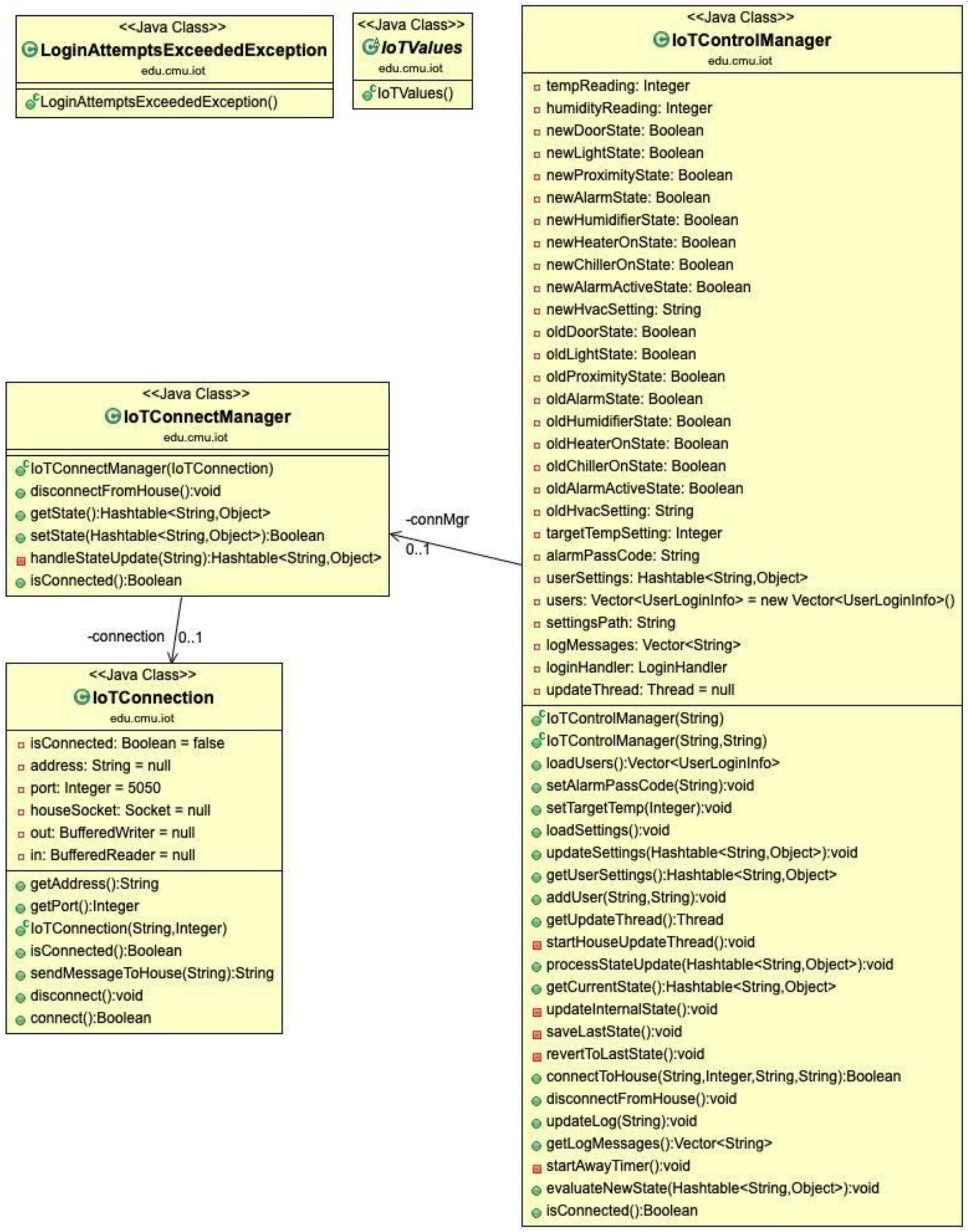

<<Java Class>>
LoginAttemptsExceededException
edu.cmu.iot
LoginAttemptsExceededException()
<<Java Class>>
IoTValues
edu.cmu.iot
IoTValues()
<<Java Class>>
IoTControlManager
edu.cmu.iot
tempReading: Integer
humidityReading: Integer
newDoorState: Boolean
newLightState: Boolean
newProximityState: Boolean
newAlarmState: Boolean
newHumidifierState: Boolean
newHeaterOnState: Boolean
newChillerOnState: Boolean
newAlarmActiveState: Boolean
newHvacSetting: String
oldDoorState: Boolean
oldLightState: Boolean
oldProximityState: Boolean
oldAlarmState: Boolean
oldHumidifierState: Boolean
oldHeaterOnState: Boolean
oldChillerOnState: Boolean
oldAlarmActiveState: Boolean
oldHvacSetting: String
targetTempSetting: Integer
alarmPassCode: String
userSettings: Hashtable<String,Object>
users: Vector<UserLoginInfo> = new Vector<UserLoginInfo>()
settingsPath: String
logMessages: Vector<String>
loginHandler: LoginHandler
updateThread: Thread = null
IoTControlManager(String)
IoTControlManager(String,String)
loadUsers():Vector<UserLoginInfo>
setAlarmPassCode(String):void
setTargetTemp(Integer):void
loadSettings():void
updateSettings(Hashtable<String,Object>):void
getUserSettings():Hashtable<String,Object>
addUser(String,String):void
getUpdateThread():Thread
startHouseUpdateThread():void
processStateUpdate(Hashtable<String,Object>):void
getCurrentState():Hashtable<String,Object>
updateInternalState():void
saveLastState():void
revertToLastState():void
connectToHouse(String,Integer,String,String):Boolean
disconnectFromHouse():void
updateLog(String):void
getLogMessages():Vector<String>
startAwayTimer():void
evaluateNewState(Hashtable<String,Object>):void
isConnected():Boolean
-connMgr
0..1
<<Java Class>>
IoTConnectManager
edu.cmu.iot
IoTConnectManager(IoTConnection)
disconnectFromHouse():void
getState():Hashtable<String,Object>
setState(Hashtable<String,Object>):Boolean
handleStateUpdate(String):Hashtable<String,Object>
isConnected():Boolean
-connection
0..1
<<Java Class>>
IoTConnection
edu.cmu.iot
isConnected: Boolean = false
address: String = null
port: Integer = 5050
houseSocket: Socket = null
out: BufferedWriter = null
in: BufferedReader = null
getAddress():String
getPort():Integer
IoTConnection(String,Integer)
isConnected():Boolean
sendMessageToHouse(String):String
disconnect():void
connect():Boolean

Figure 2: IoT Controller class diagram (Legend: UML Class Diagram)

The IoTControlManager class contains all the logic to evaluate and maintain state consistency. This class also maintains the last accepted state enabling the system to revert to a previous state as needed to promote reliability. Connections to the IoT house are controlled via the IoTConnectManager, which promotes performance and security.

The IoTControlWindow maintains a log of state decisions to allow the user to understand which states are accepted and/or rejected.

House Communication Protocol Format

The Tartan Platform talks to Home Hub boxes installed in the individual houses that then communicate with the various local sensors and actuators. The planform and home hubs interact through a proprietary communication protocol that operates over TCP, which ensures that connections persist throughout operation. The protocol is text-based with the following format:

Message ::= [Header]:[ParameterList].
Header :: = SU | GS | OK | SS
ParameterList ::= Parameter=Value; | ParameterList | None None ::= No arguments

A message has a header to indicate the type of message and optional parameter list (depending on the message). All messages terminate with a period (‘.’). The message header is separated from the message parameter list (if required) by a colon (‘:’). The header field represents the message type. Table 5 shows the supported header fields.

Table 5: Tartan protocol message headers

Parameter	**Message Body**	**Description**
GS	None	Request state status update from house.
SU	ParameterList	State status update from the house. The parameter list contains the returned state.
SS	ParameterList	Set the state in the house. The parameter list contains the new state.
OK	None	Response to a successful set state message.

The *ParameterList* is a series of key value pairs separated by a semicolon (‘;’). Table 6 describes the current supported parameters. Note that these parameters are not all required.

Table 6: Tartan protocol parameters

Parameter	**Value**	**Description**
TR	INTEGER	Temperature reading in Fahrenheit. This parameter is read-only.
HR	INTEGER	Humidity reading. This parameter is read-only.
HUS	1 or 0	Humidifier state. 1=ON,0=OFF

DS	1 or 0	Door state. 1=OPEN,0=OFF
LS	1 or 0	Light state. 1=ON,0=OFF
PS	1 or 0	Proximity state 1=HOME,0=AWAY
AS	1 or 0	Alarm state. 1=ENABLED,0=DISABLED
HES	1 or 0	Heater state. 1=ON,0=OFF
CHS	1 or 0	Chiller state. 1=ON,0=OFF

The protocol has the following semantics:

- A Get State (GS) request is results in a state update (SU) response. For example, the following get state message is valid:

  Request: "*GS*."
  Response: "*SU:TR=71;HR=51;HUS=0;DS=0;LS=0;PS=1;AS=1;CHS=1;HES=0.*"

  The request is to fetch the current state. The response indicates that the temperature is 71F, the humidity is 51%, the humidifier is off, the door is closed, the light is off, the air conditioner is on, the heater is off, the house is occupied, and the alarm is enabled.

- A Set State (SS) request includes the new state as a list of parameters and results in an OK response if successful. For example, the following set state message is valid:

  Request: "*SS:HUS=0;DS=0;LS=0;PS=1;AS=0;CHS=1;HES=0.*"
  Response: "*OK.*"

  The request is to turn off the humidifier, close the door, turn off the light, indicate that the house is occupied, disable the alarm, turn on the air conditioner, and turn off the heater. The OK response means the new state was applied successfully.

Tartan House Hub and Simulator

To use the Smart Home system, customers would buy a Tartan **House Hub**, plug it into their wifi and connect it online with the Smart Home Platform. The House Hub is a small and cheap box that speaks the House Communication Protocol with the platform and is connected to the various sensors and actuators in the house. Once released, you expect that the platform will communicate with thousands or millions of these hubs in different houses.

The House Hub is developed by a separate team and they are not ready yet. However, they have provided a simple simulator to mimic components of the house as well as their effects and the effects of users of the system. The simulator is written in Python and provides a way to simulate users entering and leaving the house, opening and closing the door, and turning on and turning off the light. Having a simulator might also be useful for testing, without requiring actual people having to interact with doors and heating systems.

The simulator included with the Tartan System fully implements the protocol described above, but it is rudimentary in terms of functionality; there are many opportunities for improvement.

Tartan Service Layer

To support the stated goals of extensibility and automation, the Tartan Service is implemented as a RESTful web service. Specifically, the Service Layer is implemented completely in Java via Dropwizard. The Dropwizard framework pulls together established Java libraries to quickly create services. JSON serialization/deserialization is handled via Jackson, Jetty is used for HTTP, Jersey is used to handle REST requests, and Hibernate is used to manage persistency. See the Dropwizard documentation for more information. Dropwizard requires certain packages and classes be defined to build a service. Each project must define a main application class, a configuration class, a resource management class, and others. The package layout for the Tartan Service Layer is shown in Figure 3.

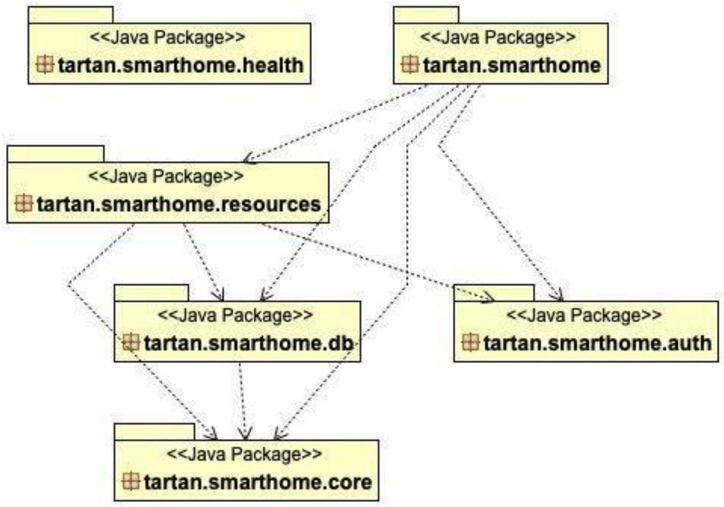


Figure 3: Tartan Service Layer Packages (Legend: UML Class Diagram)

Figure 4 shows the UML class diagram for the Tartan Service Layer. More significantly, Table 7 explains the purpose of each class.

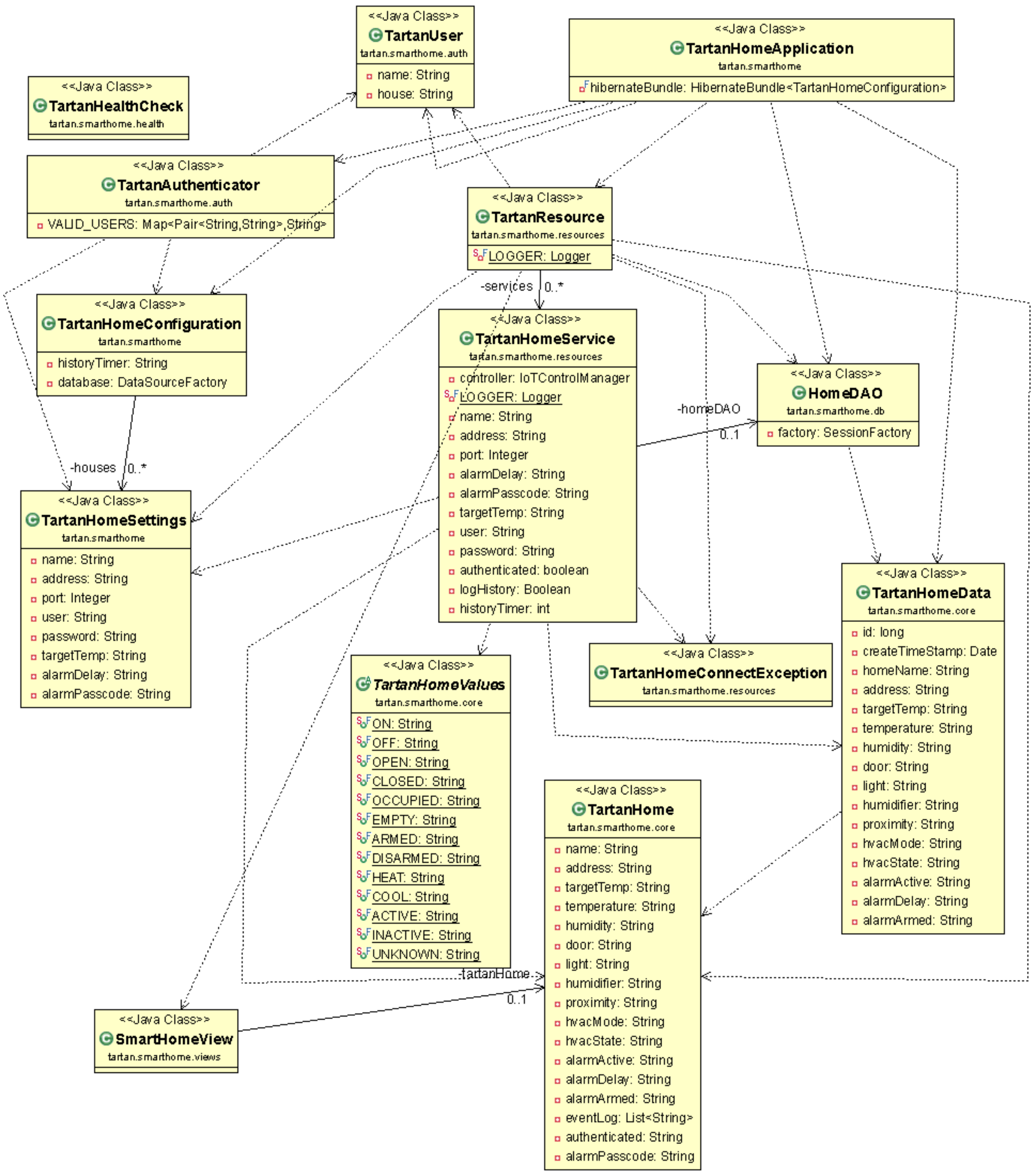


Figure 4: Tartan Service Layer class diagram (Legend: UML Class Diagram)

Table 7: Tartan Service Layer class descriptions

Class	**Description**
TartanHomeApplication	This class is the driver for the system. It loads configuration for the services layer and runs the RESTful services. See https://www.dropwizard.io/1.0.0/docs/manual/core.html#application for information on DropWizard application classes.
TartanHomeConfiguration	Configuration settings read from system configuration file. See https://www.dropwizard.io/1.0.0/docs/manual/core.html#configuration for information on Dropwizard configuration classes.
SmartHomeView	Dropwizard views provide easy ways to translate JSON models to HTML. See https://www.dropwizard.io/1.3.5/docs/manual/views.html for more information. This view renders a house state in HTML.
TartanResource	This is the Dropwizard resource class that implements the RESTful interface using Jersey[2]. The core URL for the Tartan Smart Home Platform is “smarthome” and there are two requests supported: ● *GET: [URL]/state/[HOUSE NAME]* The GET request is used to fetch the state of a specific house by name (which is configured in via initial settings). For example, the following URL could be used to fetch the house state for a house named “mse” in a server instance running on localhost port 8080: http://localhost:8080/smarthome/state/mse ● *POST: [URL]/update/[HOUSE NAME]* The POST request is used to update the state of a specific house by name (which is configured in via initial settings). For example, the following URL could be used to update the house state for a house named “mse” in a server instance running on localhost port 8080: http://localhost:8080/smarthome/update/mse Note that both requests require an authenticated user. Documentation on Dropwizard resource classes is available at

2 See https://jersey.github.io

	https://www.dropwizard.io/1.3.5/docs/getting-started.html#creating-a-resource-class
TartanHomeService	This class implements the core intelligence of the service platform. The service layer currently bridges the web interface and hardware layers of the system. This layer also serves as a natural point to integrate additional services.
TartanHomeConnectException	This class represents an exceptions when a home cannot not be reached.
TartanHealthCheck	Dropwizard supports a variety of checks to ensure that a service is running properly. See https://www.dropwizard.io/1.3.5/docs/getting-started.html#creating-a-health-check for more information. Currently, this is not implemented in the Tartan Home Platform.
HomeDAO	This class is the data access object for the system. The Tartan platform uses Hibernate[3] to manage Java data stores (see the Historian below).
TartanHomeValues	These are constants used by the system. they are organized into a class for convenience.
TartanHomeData	This is the entity class used by Hibernate to store data.
TartanHome	This is the primary model used to serialize/deserialize house state to JSON via Jackson[4]. This class is used to represent the current state of a house.
TartanUser	This class represents an authenticated user.
TartanAutenticator	This class manages authentication of users https://www.dropwizard.io/1.3.5/docs/manual/auth.html

Historian

To support data analytics and automated decision making information from the Tartan Service Platform a historian has been implemented to save house state at a set interval. In the current system data is logged for each connected house every five seconds. The following information is logged in the TartanHome database:

Table 8: Historian database parameters

Parameter	**Description**

3 http://hibernate.org

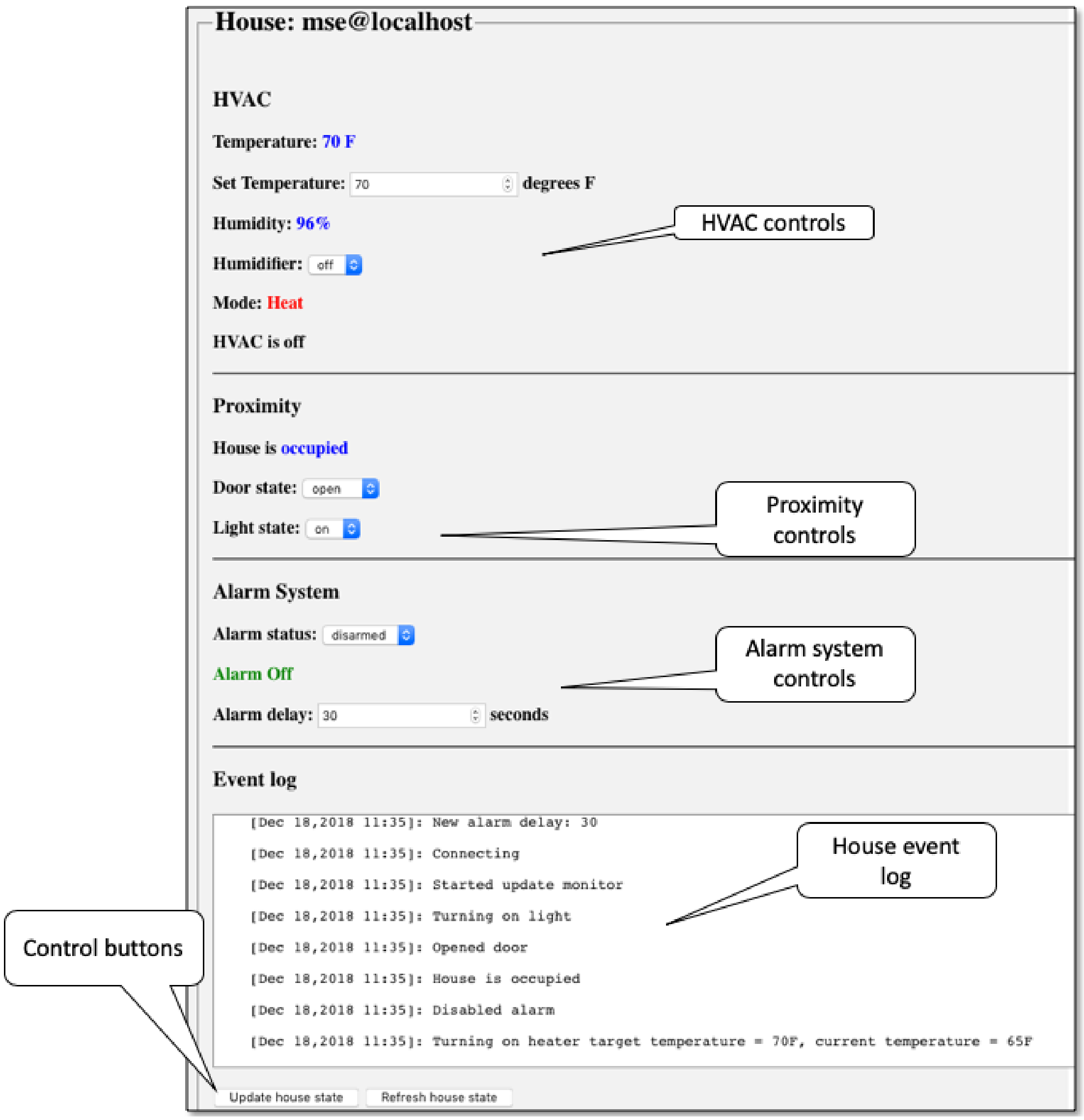
4 https://github.com/FasterXML/jackson

ID	The database generated ID for the entry
address	The house network address.
home_name	The house name.
alarm_active_state	Indicator of whether or not the house's alarm is active (i.e. going off).
alarm_enabled_state	Indicator of whether or not the alarm is enabled (i.e. armed).
alarm_delay	The amount of time in seconds that the alarm should wait before sounding after a breakin is detected.
door_state	Indicator of whether the door was open or closed.
humidifier_state	Indicator of whether the dehumidifier.
humidity	The humidity reading.
hvac_mode	An indicator for whether the HVAC system is set to the heater or the air conditioner.
hvac_state	An indicator of whether the HVAC system is on.
light_state	An indicator of whether the home lighting is on. Currently, it is assumed there is only a single light.
proximity_state	An indicator of whether this house is occupied.
target_temp	The user-set target temperature in the house.
temperature	The temperature in the house.

Web User Interface

The simple web interface used for the Tartan Home is shown in Figure 5. This interface is generated via Dropwizard views using Apache Freemarker[5]. The template file to render this named *smartHome.ftl.* Refer to that file for more information.

5 See https://freemarker.apache.org and https://www.dropwizard.io/1.3.5/docs/manual/views.html



House: mse@localhost

HVAC

Temperature: 70 F

Set Temperature: 70 degrees F

Humidity: 96%

Humidifier: off

Mode: Heat

HVAC is off

Proximity

House is occupied

Door state: open

Light state: on

Alarm System

Alarm status: disarmed

Alarm Off

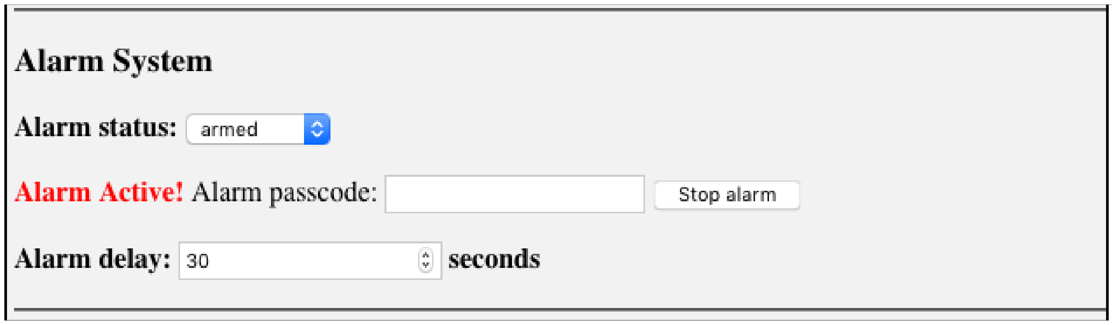
Alarm delay: 30 seconds

Event log

```
[Dec 18,2018 11:35]: New alarm delay: 30
[Dec 18,2018 11:35]: Connecting
[Dec 18,2018 11:35]: Started update monitor
[Dec 18,2018 11:35]: Turning on light
[Dec 18,2018 11:35]: Opened door
[Dec 18,2018 11:35]: House is occupied
[Dec 18,2018 11:35]: Disabled alarm
[Dec 18,2018 11:35]: Turning on heater target temperature = 70F, current temperature = 65F
```

Figure 5: Tartan System web user interface

The interface allows for basic control of a house and manages basic HTTP authentication. Notably, the interface shows a house event log so that homeowners can view how automation reasoning is working. The web interface is generally static and unchanging. One notable exception is that when an alarm sounds, the Alarm System subsection will display a place to enter the alarm passcode, as is shown in Figure 6.


Alarm System

Alarm status: armed

Alarm Active! Alarm passcode:

Alarm delay: 30 seconds

Figure 6: Tartan System web user interface to silence alarm

Platform Installation and Configuration

The initial configuration for the Tartan Smart Home System is done via a YAML file named config.yml. The configuration file contains the initial settings needed for the system including

- The configuration information for the Hubs connected to the platform.
- The timer for the data historian (see below).
- The configuration settings for database

The team has prepared docker containers to quickly get the system up and running, but for understanding how the system actually works, we describe the technical components and their configuration below.

Dependencies

The platform is written in Java, using Java 1.8. If OpenJDK is used, dependencies for javafx may need to be installed additionally (e.g., "apt-get install openjfx" on Ubuntu). Gradle 4.8 is used as a build system and already included with the gradlew wrapper. Various Java dependencies are automatically downloaded during the build from the public Maven repository. MySQL 5.7 is used as database engine. Python 3.7 is used to execute the house simulator/hub.

Database Configuration

The underlying database used in the current system is MySQL[6]. the config.yml file contains the basic settings for this database:

6 https://www.mysql.com/

```
database:
   # the name of the JDBC driver to use
   driverClass: com.mysql.cj.jdbc.Driver

   # the DB username
   user: tartan

   # the DB password
   password: tartan1234

   # the JDBC URL; the database is called TartanHome
   url:
jdbc:mysql://localhost/TartanHome?useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&ser
verTimezone=UTC

   # Allow Hibernate to create tables
   properties:
       hibernate.dialect: org.hibernate.dialect.MySQLDialect

       # leave it to hibernate to update/create the database.
       # Warning, this is generally considered a bad setting for production
       hibernate.hbm2ddl.auto: update
```

That is, the system expects access to a MySQL database on localhost with a user named “`tartan`” with a password of “`tartan1234`” and a database named “`TartanHome`”. You can use the sample SQL script init.sql found inside the Database subfolder to create the database and the user. Tables within this database are automatically created when the system is run. Note that these credentials are specified in the system YAML configuration file (config.yml).

Building the Platform

The system is currently developed in two separate repositories. One repository contains only the IoT Controller (separated for legacy reasons), whereas the other repository contains the remaining Tartan Service Platform code as well as the house simulator (the Hub). The Tartan Service Platform repository depends on the IoT Controller implementation.

You can create new group repositories of the IoT Controller and the Hub with the following links (one for each project element):

- IoTController: https://classroom.github.com/g/T88_M5ak
- Hub: https://classroom.github.com/g/1RkymiN9

The current implementation uses Gradle as build system. You may want to use the gradle wrapper "gradlew" installed in the repositories which will automatically download and use the correct gradle version. To that end, replace "gradle" by "./gradlew" in Ubuntu and macOS, or "gradlew.bat" in Windows. You can supply the same parameters to **gradlew** as simple **gradle**, e.g., “./gradlew build”

To initialize gradlew if needed, you need to have gradle installed, then run “gradle wrapper” in the required directories (smart-home and smart-home/Platform).

In its current configuration, the Platform depends on a jar file from the IoT Controller project. The default practice to create this .jar locally is to execute "./**`gradlew publishToMavenLocal`**" inside the IoT Controller project (inside *iot-controller/* folder). This will place the IoT Controller .jar file in a location where it can be found by the Platform project automatically.

In the Platform project, executing **`"gradle shadowJar"`** inside the **`"/Platform"`** subfolder will create a single "**`tartan-1.0-SNAPSHOT.jar`**" file that contains the Platform and all dependencies. This file will be created inside the "/**`Platform/build/libs`**" subfolder. The Platform can then be launched with **`"java -jar tartan-1.0-SNAPSHOT.jar <parameters>`**" or simply via **`"./gradlew run"`** (see below for more details about the parameters).

Many of these decisions were made for legacy reasons. Your team may change all of these technical decisions and tooling if preferred.

Running the System

The system consists of the following components that must be launched:

1. One or more House Hubs (the house simulator, part of the TartanSmartHome repo)
2. The MySQL database
3. The Tartan Service Platform

A docker setup is provided with a finished configuration in the platform repository that starts two houses. You can launch it with "**`docker-compose up --build`**". If you want to run this version, note that there is a separate config file, config.docker.yml, that is used in this case.

Windows differences

- Install and open “Docker Desktop”, which is Docker for Windows.
- Under “Container / Apps” remember to “Start” the “smart-home” container
  - If you do not yet see the “smart-home” container, you may need to try the "**`docker-compose up --build`**" first so docker is aware of it.

- Remember to create the gradle wrappers with the “gradle wrapper” command

Below are the instructions to launch the components individually without docker. However, it is recommended to use Docker Compose instead of manually controlling the start and shutdown of the components.

The House Simulator

The house simulator is a simple Python program that takes two command line arguments: the host and port to listen for incoming connections. To run the house simulator simply execute the following command (port will need to be changed for each house if simulating multiple houses):

- **`python3 simple_server.py localhost 5050`**

The start a second house, consistent with the default config.yml configuration, execute the following command on another command window:

- **`python3 simple_server.py localhost 5051`**

The Historian Database

You must start the MySQL server to log house history. MySQL is typically started via the following command in Ubuntu:

- **`sudo systemctl start mysql`**

And in Windows:

- **Type “services” in the Windows search bar and open the “Services” application**
- **Scroll down to MySQL and start the service**

If running manually (not using `docker-compose`) then you need to create the database and user manually as well. The following example is for a Windows system.

```
cd C:\Users\UserName\Documents\CMPUT 402\Tartan\smart-home\Database
"C:\ProgramFiles\MySQL\MySQL Server 5.7\bin\mysql.exe" -u root -p
source init.sql
```

The Tartan Smart Home Service

This, again, requires MySQL server. The Platform can be run using the following command (from */Platform* subfolder):

- **`java -jar tartan-1.0-SNAPSHOT.jar server config.yml`**

It can also be executed with "`./gradlew run`", which will pass the same parameters automatically. Note that the platform must be launched after the database and the house hub/simulator scripts are running.

This is the standard way to run a Dropwizard application[7]. The *server* command runs your application as an HTTP server and the YAML file *config.yml* contains the initial configuration settings. If the system is running correctly, then you should be able to access it by navigating to the following URL in a web browser:

- http://localhost:8080/smarthome/state/<housename>

Note that "mse" and "cmu" are the names of the default houses configured in config.yml. The default username and password for access to the system is also configured in this file ("admin" and "1234" for "mse", "admin" and "5678" for "cmu").

Possible issues

While trying to run The Platform, you may encounter the following error:
javax.net.ssl.SSLHandshakeException: No appropriate protocol (protocol is disabled or cipher suites are inappropriate)

Appending **&enabledTLSProtocols=TLSv1.2** to the end of the *database.url* property in *config.yml* will fix the issue (see the StackOverflow solution).

7 See https://www.dropwizard.io/1.3.5/docs/getting-started.html