PROJECT DESIGN DOCUMENT

SUBMITTED BY:

GROUP: 3

G. UMA MAHESWARA RAO (201250819)

HANUMANTHA REDDY S (201250822)

CHERUKUPALLI PRABHAKAR (201250810)

S.D. NAGESWARA PAVAN KUMAR (201150836)

THAKKAR HEMAL NARENDRA (201350890)

AVINASH CHANDWANI (201350882)

SHELKE NAMITA RAVINDRA (201405633)

SINGARAJU SUGUNA MALLIKA 201450866)

J VAMSHI VIJAY KRISHNA(201450865)

IOT Platform

1. **Introduction to the Project**

   The objective of the project is to build platform that manages peripheral devices or sensors (internet of things) and android application, communicate asynchronously with each other. Platform covers wide range of issues to deal with like registration of devices, data management, device management including rule management and trigger management, application management.

Functional overview of Team's module (1-2 parah. & Block diagram)

Ø Developing an application for the Internet of Things is a lot EASIER and QUICKER using our platform. We want you to save your time and money by using this. You can focus on building amazing new product and not worrying about communications protocols, infrastructure, databases, etc.

Ø We have added security for each and every user and his App. This platform is easily manageable, easy to develop and emulate different types of app.

Ø Our platform sends ping request to each and every device find its health and maintains a list of healthy devices.

Ø Whenever a request of data from any app, than it will search for healthy/free devices, and then it sends information directly to App.

Ø It also maintains rule engine, whose work to store common rules related to sensory device.

Ø It allows each device to update rule, when some condition is fulfilled. It triggers appropriate action corresponding to that.

2. Test cases

2.1Test cases- used to test the team's module

Test Case 1 : User registers for app and specifies rules for the app such as for Traffic Monitoring app rules will be like (if traffic density is greater than 70% => Suggest an alternative route). The rules are sent to device manager.

Test Case 2: When User initially registers an application . All available devices are shown to the User. This is done by getting the devices from the device manager.

Test Case 3: User can update the rules at any time i.e. he can change traffic density of above rule to 60 %. Then the updated rule is sent to the device manager.

Test Case 4 : When a User wants to register a new device (new sensor etc.) . The app manager contacts the device manager to register the device.

Test case 5: Device manager should update the rules when aPass make such a request using API call.

Test case 6: Trigger should work when rule specified by application is satisfied.

Test case 7: Device manager queries for all the devices presently active.

Test case 8: Device manager queries to registry that whether a particular device is active or not by specifying its id.

Test case 9: The list of currently active devices should be returned correctly to the proxy server.

Test case 10: Registry should provide correct information of the devices.

Test case 11: Should tell whether a device with provided id is active.

2.2 Overall project test cases (relevant to the module)

- Integration testing:
- How & what to test, after integrating with other modules
- This shd be based on overall use-cases listed in Group reqts doc

3. Solution design considerations

(Group level- to be discussed by all three teams, and captured in all three team-design documents).

3.1 Design big picture

- one diagram listing all components in the solutions and interactions therein

3.2 Environment to be used

1. Android, Linux, Apache Tomcat/Jboss

3.3 Technologies to be used

- Java RESTFul WebServices
- Node.js
- MongoDb
- Android App
- Bluetooth/WIFI
- Java Programming Language

3.4 Approach for Device gateways

Rasbpery pie will act as device gateway hardware component

3.5 Devices type/interfaces and information structure

1. Sensors (Mobile in this case)
2. Rasbpery pi as gateway
3Intermediate server hosting RESTFul services
4. Filter server. This embeds rules engine, defined by the business, for further forwarding of this information
5. Receiving device: This will be mobile apps in this case.

3.6 Devices registry & repository

A database will be maintained which holds repository information about all the sensors. Any sensor information coming to the RESTFUl server, will be validated whether this is a new sensor or an existing sensor. In the latter case, the registry will be added for the sensor device.

The repository will consist of two types of information : 1. Registry information 2. Live Data information

3.7 Approach to get device information (event streams)

Two categories of events, will trigger message pushing:

1. Live sensor data
2. Alive Acknowledgement status

3.9 Communication overview

Protocols:

1. BLE/WIFI
2. RESTFUL Communication
3. Push/Pull Mechanism between filter server and receiving device

3.10 Filter Server

Filter server embeds rules engine, which becomes the core of planning in turn bringing PUSH/PULL automation.

3.11 Logic server

Logic server technically means rules engine as described above.

3.12 Interactions between modules

_Interaction Between Sensors and Gateway

- Sensors push the data to Gateways via bluetooth.

Interaction Between Gateway and Filter Server

- Gateways push the data to Filter Server calling **PushDeviceData**.

Interaction Between Registry and FilterServer

- Filter Server will use **IsDeviceActive** api of Registery to find if a device is active
- Filter Server will use **GetActiveList** api of Registery to get a list of active devices

Interaction Between AppManager and Filter Server

- AppManager will use **RegisterApp** api of Filter Server to register an app
- AppManager will use **UpdateRule** api of Filter Server to update an existing rule

Interaction Between Filter Server and AppManager

- Filter Server will use **GetAuthorisation** api of AppManager to authorize actions for particular app.
- Filter Server will use **AppendDevicedata** api of AppManager to append new data to cloud storage.

- File/Wire format definitions

All module to module interactions are JSON-RPC based.

File Format: JSON

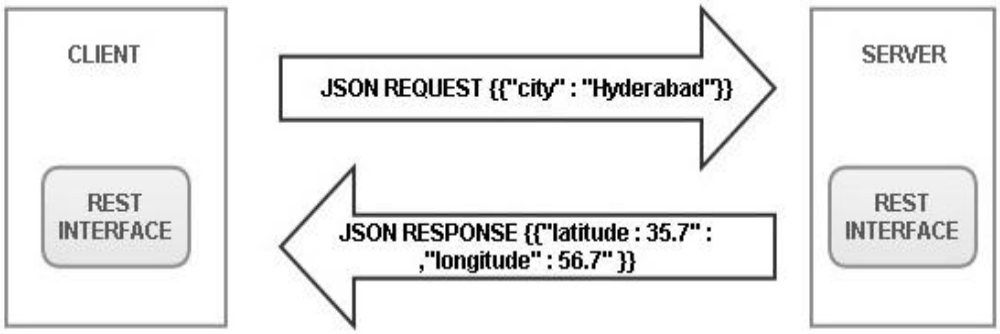
- Devices will be communicating to the platform via HTTP Rest call using JSON as data exchanging format.
- Devices will do RPC calls to Gateway servers and Gateway server will do RPC calls to the Filter Server
- Information Structure :
  - Register: On the start of a device, it will register itself to the platform and gets assigned with Device ID.

- ✓ Request
  {
    Device-Type: “Traffic-Sensor”
    IP-address: 10.1.1.3
    Version: 1.0
  }
- ✓ Response
  {
    Device-ID: 1236789
    Status: “success”
  }

- Health Ping: The devices will do health ping to registry or to Gateway periodically.
  - ✓ Request
    {
      Device-Type: “Traffic-Sensor”
      IP-address: 10.1.1.3
      Version: 1.0
      Status: “active”
    }
  - ✓ Response
    {
      Status: “Success or Fail”
    }

- The device will need to send data to the platform.
  - ✓ Request
    {
      Command-ID: “123489”
      Status: “Success or Fail”
      Command Parameters :{}
    }
  - ✓ Response
    {
      Status: “Success or Fail”
    }
- The devices may get the command from the android app.

  - Filter Server will use **IsDeviceActive** api of Registry to find if a device is active
  - Filter Server will use **GetActiveList** api of Registry to get a list of active devices
  - AppManager will use **RegisterApp** api of Filter Server to register an app
  - AppManager will use **UpdateRule** api of Filter Server to update an existing rule
  - Filter Server will use **GetAuthorisation** api of AppManager to authorize actions for particular app.
  - Filter Server will use **IsDeviceActive** api of Registery to find if a device is active
  - Filter Server will use **GetActiveList** api of Registery to get a list of active devices
  - AppManager will use **GetDeviceType** api of Registery to get a list of type of devices



3.13 Wire and file formats

All module to module interactions are JSON-RPC based.

File Format: JSON

- Devices will be communicating to the platform via HTTP Rest call using JSON as data exchanging format.
- Devices will do RPC calls to Gateway servers and Gateway server will do RPC calls to the Filter Server
- Information Structure :
  - Register: On the start of a device, it will register itself to the platform and gets assigned with Device ID.
    - ✓ Request

      {

      Device-Type: “Traffic-Sensor”
      IP-address: 10.1.1.3
      Version: 1.0

      }
    - ✓ Response

      {

      Device-ID: 1236789
      Status: “success”

      }
  - Health Ping: The devices will do health ping to registry or to Gateway periodically.
    - ✓ Request

      {

      Device-Type: “Traffic-Sensor”
      IP-address: 10.1.1.3
      Version: 1.0
      Status: “active”

      }
    - ✓ Response

      {

      Status: “Success or Fail”

      }

  - The device will need to send data to the platform.
    - ✓ Request

      {

Command-ID: “123489”
Status: “Success or Fail”
Command Parameters :{}
}

- ✓ Response
{
Status: “Success or Fail”
}

The devices may get the command from the android app.

3.14 User’s view (end UI on mobile)

1. View for updating rules
2. Notification screen.

5. User’s view of system

- What is an application- for each type.
  Camera Sensor:
  Motion sensor : To control optimum usage of resources like Power.
  Temperature Sensor : To control AC
  Location Sensor: Useful for all tracking purpose.
  Biometrics Sensor: User can give attendance and view the history.

- How are analytics rules defined
  User defines their rules from the provided UI screen on the mobile app
- How is it deployed/setup. And where (on which module/process/location et al)
  Deployed on Android Mobiles
- How to start & stop
  A scope to shutdown the application.

User interactions

- How will the user use this system
  Through Android Application
- what & how to configure
  UI Screen to modify their preferences of registration.
- What are How to deploy
  Install the app on android mobile

6. Key Data structures

Definition & other details of deice types
SensorTYpe, SensorId, SensorData, checkSum
Wire formats
Data format mentioned above
Persisted data (registry, rules, et al)
MongoDB will be used to store persistent data
APIs- interaction data objects
XML in webservice communication
App Logic definitions
Database schema, In memory and cacheing data structures.

7. Interactions & Interfaces

APIs
RESTFUL APIs
Android APIs

Rules Engine Java API

- User interactions

Through Android App.

- Module to Module interactions

Through communication gateway and RESTFul

- File/Wire format definitions

SensorTYpe, SensorId, SensorData, checkSum

Types of communication:

RESTFul, Bluetooth, WiFi,

8. Persistence

Mongo DB Data structures

9. The three modules; that the three team will work on

Internal design overview (additional details like above, for each model)

Low Level Design (for each of the 3 modules)

1. Lifecycle of the module

- How will the module/its components be started, monitored and stopped (if applicable).
  1. To check Sensors are up
     - By alive signals
  2. To check RESTFUL and filter server are up
     - by testing web services methods

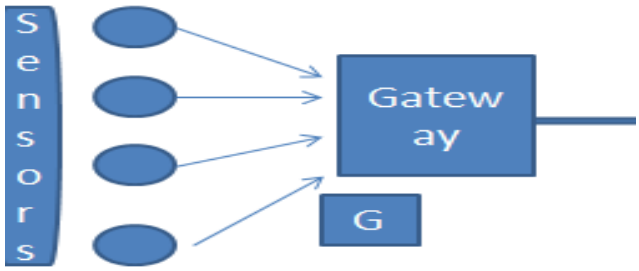
2. List the sub modules

1. Sensor/Gateway
2. RESTFulServer
3. Filter Server and receiving mobile applications.

Brief overview of each sub module

6.1.1 Gateway

- Hardware & Technology stack
  RasberryPi
  Node.js
- Functionality
  - Receives the data from the devices
  - Prepares the data as per the wire protocol and pushes the data to filter server
- Block Diagram



- Interactions
  - Gateways push the data to Filter Server calling **PushDeviceData**

- API
  - PushDeviceData

Filter Server

- Technology Stack:-
  Node.js
  MongoDB

- Parts
  Rules Engine
  - Rules Table

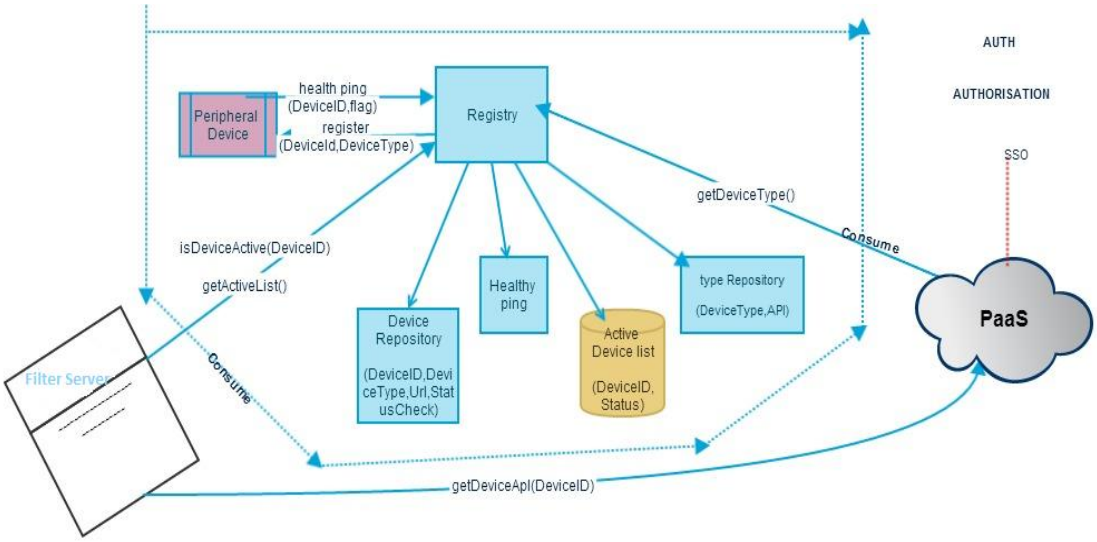
  Registry
  - Device Repository
  - Type Repository
  - Active Device List

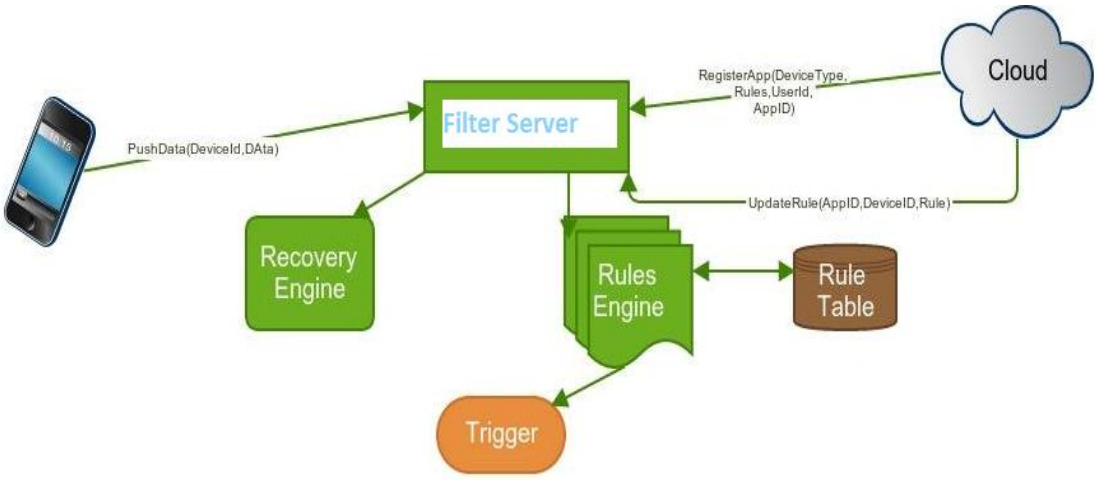
  Recovery Engine
  Data Manager

- Functionality
  Filter Server will take data from sensory devices and perform actions according the rules given by the user.
  - Register the application and pair user with corresponding devices.
  - Take rules from user and convert rules into the action using rules engine.
  - Take data from the devices and if any rule applies then take appropriate action.
  - Before taking action it will check for authorization.

  Different devices will register on the registry and it will check if the devices are active or not.
  - Peripheral devices will register on registry.
  - It will check if the devices are in active state or not.
  - It will provide different API for filter server for giving list of active devices.

- Block Diagram





- Interactions
  - Filter Server will use **IsDeviceActive** api of Registry to find if a device is active
  - Filter Server will use **GetActiveList** api of Registry to get a list of active devices
  - AppManager will use **RegisterApp** api of Filter Server to register an app
  - AppManager will use **UpdateRule** api of Filter Server to update an existing rule
  - Filter Server will use **GetAuthorisation** api of AppManager to authorize actions for particular app.
  - Filter Server will use **IsDeviceActive** api of Registery to find if a device is active

- Filter Server will use **GetActiveList** api of Registery to get a list of active devices
- AppManager will use **GetDeviceType** api of Registery to get a list of type of devices

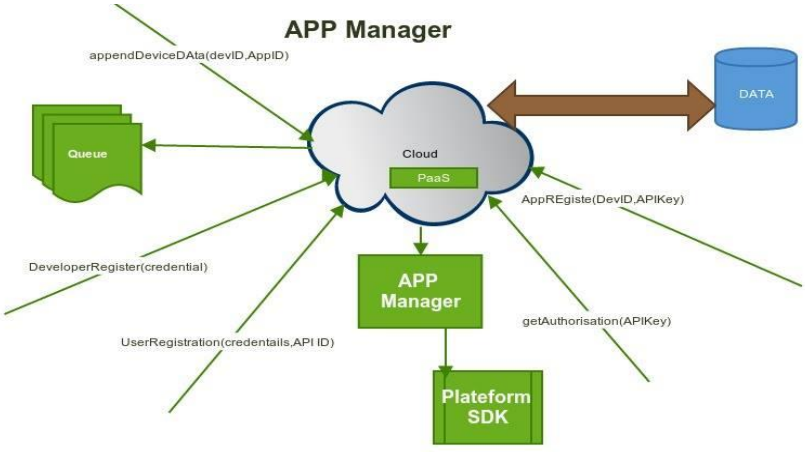
- API
  - Register Application
  - UpdateRules
  - PushData
  - Register
  - GetActivelist
  - GetDeviceType
  - IsdeviceActive

AppManager

- Technology Stack:-
  Android

- Parts
  AppManager
  Platform SDK

- Functionality
  It will register developer App and user and give authorization token and communicate with Filter Server ,and work as bridge between user and devices.
  - It will register developer and generate authentication token.
  - Developer can choose the type of devices they want to use.
  - User registration and pairing with devices.
  - Take device data from app manager and store at the cloud storage.
- Block Diagram



- Interactions
  - App Manager will use **GetDeviceType** api of Registery to get a list of device types.
  - AppManager will use **RegisterApp** api of DeviceManager to register an app

- AppManager will use **UpdateRule** api of DeviceManager to update an existing rule
- Filter Server will use **GetAuthorisation** api of AppManager to authorize actions for particular app.

- API
  - DeveloperRegister
  - AppRegister
  - GetAuthorization
  - UserRegistration
  - AppendDeviceData